ANTÓNIO MIGUEL BARRADAS FIGUEIREDO

OTIMIZAÇÃO TECNOLÓGICA PARA A GESTÃO INTELIGENTE DOS RECURSOS ENERGÉTICOS NUMA UNIDADE HOTELEIRA NO ALGARVE

UNIVERSIDADE DO ALGARVE
Instituto Superior de Engenharia
2025

ANTÓNIO MIGUEL BARRADAS FIGUEIREDO

OTIMIZAÇÃO TECNOLÓGICA PARA A GESTÃO INTELIGENTE DOS RECURSOS ENERGÉTICOS NUMA UNIDADE HOTELEIRA NO ALGARVE

Mestrado em Engenharia Mecânica

(Especialidade em Energia, Climatização e Refrigeração)

Trabalho realizado sob a orientação de:

Professora Doutora Cláudia Dias Sequeira

Professora Doutora Manuela Moreira da Silva

UNIVERSIDADE DO ALGARVE
Instituto Superior de Engenharia
2025

OTIMIZAÇÃO TECNOLÓGICA PARA A GESTÃO INTELIGENTE DOS RECURSOS ENERGÉTICOS NUMA UNIDADE HOTELEIRA NO ALGARVE

Declaração de autoria da obra

Declaro ser o autor desta obra, que é original e inédita. Os autores e obras consultados são devidamente citados no texto e constam da listagem de referências incluídas.

AGRADECIMENTOS

A realização desta dissertação não teria sido possível sem o contributo e o apoio de diversas pessoas e instituições, às quais expresso o meu mais profundo reconhecimento.

Em primeiro lugar, expresso o meu sincero reconhecimento à Universidade do Algarve instituição que, ao disponibilizar os meios, os recursos e as ferramentas indispensáveis à educação e à investigação científica, tornou possível a concretização deste trabalho. O ambiente académico de excelência proporcionado pela Universidade constituiu um alicerce fundamental para o meu desenvolvimento científico, técnico e pessoal.

Aos meus colegas de mestrado, deixo uma palavra de apreço pela partilha de experiências, pelo espírito de entreajuda e pela motivação constante ao longo deste percurso. O convívio académico e humano foi determinante para ultrapassar os desafios encontrados e para enriquecer esta etapa da minha formação.

À minha família e a todos aqueles que fazem parte do meu núcleo mais próximo, deixo um agradecimento profundo e sincero. Este percurso não se constrói de forma isolada, pois ninguém se faz sozinho. O apoio, a compreensão e a presença constante de cada um foram verdadeiros pilares que sustentaram não apenas a realização desta dissertação, mas também o meu crescimento pessoal e académico.

Quero agradecer, às minhas orientadoras de dissertação, Professora Doutora Cláudia Dias Sequeira e Professora Doutora Manuela Moreira da Silva, manifesto a minha sincera gratidão pela orientação rigorosa, pela disponibilidade permanente e pelas valiosas recomendações metodológicas e científicas, que foram fundamentais para o desenvolvimento deste trabalho.

Dirijo um agradecimento especial à Professora Doutora Cláudia Dias Sequeira, pela determinação e motivação transmitidas que foram decisivas para o aprofundamento do estudo na área da manutenção. O seu exemplo de profissionalismo e paixão pela área constituiu um estímulo permanente, que marcou de forma significativa o desenvolvimento deste trabalho e o meu crescimento académico e pessoal.

RESUMO

A eficiência energética e a sustentabilidade são hoje fatores críticos para o setor do turismo, exigindo soluções inovadoras que conciliem conforto, competitividade e responsabilidade ambiental. Neste contexto, a presente dissertação centra-se na gestão de energia, na monitorização contínua e na otimização de recursos energéticos numa unidade hoteleira de grande dimensão no Algarve, tendo como tema central a aplicação dos conceitos da indústria 4.0 e da inteligência artificial para promover eficiências energética e hídrica e apoiar a manutenção preditiva. O estudo analisa três anos de dados de eletricidade, água e gás, correlacionados com variáveis operacionais como ocupação e sazonalidade, aplicando modelos inteligentes para prever consumos, identificar anomalias e quantificar desvios operacionais. A integração da indústria 4.0 com algoritmos de inteligência artificial revela-se decisiva para transformar grandes volumes de dados em informação estratégica, permitindo não apenas antecipar falhas e otimizar ciclos de manutenção, mas também reduzir de forma consistente a pegada carbónica associada às operações hoteleiras. A monitorização contínua dos consumos de eletricidade, água e gás, quando aliada a modelos preditivos, possibilita identificar padrões de desperdício e implementar medidas corretivas em tempo real, assegurando que os recursos são utilizados de forma mais eficiente. Neste contexto, os gráficos de controlo assumem um papel central como ferramentas de apoio à decisão, uma vez que permitem visualizar desvios face ao comportamento esperado, quantificar o erro relativo diário e estabelecer limites de alerta para consumos anómalos. A utilização destas ferramentas não só reforça a capacidade de diagnóstico operacional, como também contribui para a definição de estratégias de mitigação alinhadas com os objetivos de neutralidade carbónica. Assim, a aplicação integrada destas metodologias transforma o hotel numa infraestrutura inteligente e resiliente, capaz de responder dinamicamente às exigências de sustentabilidade, eficiência e rentabilidade, ao mesmo tempo que se posiciona como referência no setor turístico pela sua gestão responsável dos recursos energéticos e ambientais.

Palavras Chave: *Gestão de energia; Monitorização; Otimização de recursos energéticos; Indústria 4.0; Manutenção preditiva; Eficiências Energética e Hídrica.*

ABSTRACT

Energy efficiency and sustainability are today critical factors for the tourism sector, demanding innovative solutions that reconcile comfort, competitiveness, and environmental responsibility. In this context, this dissertation focuses on energy management, continuous monitoring, and the optimization of energy and water resources in a large hotel unit in the Algarve, with its central theme being the application of Industry 4.0 concepts and Artificial Intelligence (AI) to promote both energy and water efficiency while supporting predictive maintenance. The study analyzes three years of electricity, water, and gas consumption data, correlated with operational variables such as occupancy and seasonality, applying intelligent models to forecast consumption, detect anomalies, and quantify operational deviations.

The integration of Industry 4.0 with AI algorithms proves decisive in transforming large volumes of data into strategic information, enabling not only the anticipation of failures and the optimization of maintenance cycles but also the consistent reduction of the carbon footprint associated with hotel operations. Continuous monitoring of electricity, water, and gas consumption, when combined with predictive models, makes it possible to identify waste patterns and implement corrective measures in real time, ensuring more efficient use of resources. In this context, control charts emerge as central decision-support tools, as they allow the visualization of deviations from expected behavior, the quantification of daily relative error, and the establishment of alert thresholds for anomalous consumption.

The use of these tools not only strengthens operational diagnostic capacity but also contributes to the definition of mitigation strategies aligned with carbon neutrality goals. Ultimately, the integrated application of these methodologies transforms the hotel into an intelligent and resilient infrastructure, capable of dynamically responding to the demands of sustainability, efficiency, and profitability, while positioning itself as a reference in the tourism sector through the responsible management of energy and environmental resources.

Keywords: Energy management; Monitoring; Energy resources optimization; Industry 4.0; Predictive maintenance; Energy and water efficiency.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÓNIMOS

Acrónimo	Significado
AQS	Água Quente Sanitária
APA	Agência Portuguesa do Ambiente
AVAC	Aquecimento, Ventilação e Ar Condicionado
Big Data	Grande volume de dados
CNN	Convolutional Neural Networks (Redes Neurais Convolucionais)
CO_2e	Emissões equivalentes de dióxido de carbono
CPOR	Custo por Quarto Ocupado (Métrica de eficiência/impacto financeiro)
CPS	Sistemas Ciber-Físicos (Cyber-Physical Systems)
DL	Deep Learning (Aprendizagem Profunda)
DGEG	Direção-Geral de Energia e Geologia
DT	Decision Tree (Árvore de Decisão)
EMS	Sistemas de Gestão Energética (Energy Management Systems)
ERSAR	Entidade Reguladora dos Serviços de Águas e Resíduos
ET	Empreendimentos Turísticos
ET27	Estratégia Turismo 2027
GEE	Gases com Efeito de Estufa
GHG Protocol	Greenhouse Gas Protocol (Protocolo GEE)
GPL	Gás Liquefeito de Petróleo
GRU	Gated Recurrent Unit (Tipo de rede neural recorrente)
GTC	Gestão Técnica Centralizada
I4.0	Indústria 4.0 (Quarta Revolução Industrial)
IA	Inteligência Artificial
IoT	Internet das Coisas e Serviços (Internet of Things)
ISO 50001:2018	Norma internacional para sistemas de gestão de energia
ISO 14064-1:2018	Norma internacional para inventário de emissões
KPI	Key Performance Indicator (Indicador Chave de Desempenho)
LIME	Local Interpretable Model-agnostic Explanations
LSTM	Long Short-Term Memory (Tipo de rede neural recorrente)

MAE	Erro Absoluto Médio (Mean Absolute Error)
ML	Machine Learning (Aprendizagem de Máquina)
MSE	Erro Quadrático Médio (Mean Squared Error)
OMT	Organização Mundial do Turismo
PLN	Processamento de Linguagem Natural
PNEC 2030	Plano Nacional Energia e Clima 2030
Q1 / Q3	Quartis (25% / 75% da distribuição estatística)
R^2	Coeficiente de Determinação
RASARP 2024	Relatório Anual dos Serviços de Águas e Resíduos em Portugal
RF	Random Forest
RNA	Redes Neuronais Artificiais
RNC 2050	Roteiro para a Neutralidade Carbónica
RNN	Redes Neuronais Recorrentes
RMSE	Raiz do Erro Quadrático Médio (Root Mean Squared Error)
SCADA	Supervisory Control and Data Acquisition
SHAP	SHapley Additive exPlanations
SVR	Support Vector Regression
SVM	Support Vector Machines
TIC	Tecnologias da Informação e Comunicação
UTA	Unidades de tratamento de ar
XAI	Explainable Artificial Intelligence (Inteligência Artificial Explicável)
XGB	Xtreme Gradient Boost

1

INTRODUÇÃO

O conceito de sustentabilidade consolidou-se nas últimas décadas como um dos pilares centrais para o desenvolvimento das sociedades contemporâneas. De acordo com o Relatório Brundtland (1987)[1], o desenvolvimento sustentável é definido como o desenvolvimento que satisfaz as necessidades do presente sem comprometer a capacidade das gerações futuras satisfazerem as suas próprias necessidades. Sendo esta definição a base para a formulação de políticas globais e estratégias empresariais, integrando as três dimensões fundamentais: económica, social e ambiental. No setor do turismo, a sustentabilidade assume um papel bastante relevante, dado o impacto direto que esta atividade exerce sobre os ecossistemas, as comunidades locais e a economia global. No contexto empresarial, a sustentabilidade traduz-se na adoção de práticas que conciliem a eficiência energética, a gestão da água e a redução de resíduos, sem comprometer a qualidade do serviço prestado ao cliente.

Do ponto de vista da sustentabilidade, o mais importante é mitigar os efeitos nas alterações climáticas. É necessário reduzir as emissões dos gases com efeito de estufa e, para isso, diminuir significativamente o consumo das energias mais poluentes. Contudo, os avanços deverão ser cuidadosos para que não se crie um impacto negativo sobre a economia, sobre o ambiente e sobre setores socias. Só seremos sustentáveis quando encontrarmos o balanço ideal entre os referidos fatores: económico, ambiental e social. Por isso fomenta-se que as novas tecnologias da energia devem ser bem investigadas, projetadas e desenvolvidas, para que seja possível garantir uma transformação sustentável a longo prazo.

Devido à preocupação com o ambiente, tem sido objetivo da sociedade ao longo das últimas evoluções tecnológicas, investir em práticas sustentáveis e fazer uma mudança significativa no que toca à gestão dos recursos energéticos e hídricos. Essas práticas fomentam uma

estratégia de gestão inteligente através da monitorização continua, da eficiência energética e hídrica e da integração de novas tecnologias.

No contexto empresarial, setores como o da hotelaria que, historicamente, dependem significativamente de recursos elétricos para atender às necessidades operacionais e oferecer serviços de qualidade aos hóspedes, enfrentam um grande desafio em conseguir otimizar essa variável, bem como na gestão da água, um recurso indispensável ao bom funcionamento de um hotel. Conforme as práticas sustentáveis, não basta só reduzir o consumo de eletricidade pela substituição de lâmpadas, pela troca de equipamentos mais eficientes ou por aplicar melhorias no isolamento. Em relação à água também existe espaço para grandes avanços do ponto de vista da sua utilização de uma forma mais eficiente. As otimizações das ferramentas de gestão técnica centralizada também se evidenciam nos conceitos das práticas sustentáveis.

A crescente pressão para reduzir consumos energéticos e emissões de gases com efeito de estufa tem levado organizações de diferentes setores a repensar as suas estratégias de gestão de energia. No contexto atual, marcado pela transição energética e pela digitalização dos processos, torna-se cada vez mais relevante integrar práticas de monitorização contínua, manutenção eficiente e previsão de consumos. A importância de uma boa gestão energética ao longo da operação anual poderá trazer impactos consideráveis na despesa anual dos seus recursos energéticos, bem como na identificação de eventuais pontos críticos de consumo.

A Indústria 4.0, com as suas tecnologias emergentes, abre novas possibilidades para transformar dados em informação estratégica, permitindo decisões mais rápidas, precisas e sustentáveis. É neste enquadramento que se insere a presente dissertação, cujo foco é compreender como a gestão de energia, aliada à manutenção de ativos e ao uso de modelos preditivos baseados em dados, pode contribuir para a otimização de recursos e para a sustentabilidade organizacional.

1.1 O OBJETIVOS PRINCIPAIS

Prende-se a necessidade de estudar variáveis de consumo energético e hídrico e avaliar o respetivo comportamento durante o ciclo de vida de um sistema, assim como, fundamentar uma avaliação abrangente do consumo energético em instalações e equipamentos, identificação de ineficiências energéticas e recomendações para melhorias.

Admitindo a aquisição de dados, e o seu processamento e avaliação, é importante compreender como as informações baseadas em histórico de dados podem ser usadas para tomar decisões sobre a eficiência energética, identificar oportunidades de economia de energia e implementar medidas de redução de consumo. Como a avaliação contínua da eficiência energética em

hotelaria deve ser usada como um processo estratégico, que combina monitorização em tempo real, análise estatística, manutenção preditiva para garantir que as metas da sustentabilidade sejam alcançadas.

O objetivo deste estudo é compreender as estratégias de gestão de energia, no âmbito da manutenção de ativos, da monitorização e da previsão de consumos energéticos. Procura-se desenvolver técnicas de manutenção orientadas para a otimização dos recursos energéticos, bem como implementar os conceitos da Indústria 4.0. Pretende-se estudar tecnologias de monitorização auxiliadas complementarmente por modelos estatísticos e preditivos que contribuem para o desenvolvimento e interpretação da informação adquirida. Além disso, visa-se compreender como a manutenção e reparação dos ativos consumidores impactam a gestão energética e hídrica de uma unidade hoteleira.

Reconhecer de que forma a era digital tem contribuído para o desenvolvimento da aquisição e processamento de dados, como a inteligência artificial é uma ferramenta para tratar dados confiáveis e que por sua vez suporte as grandes tomadas de decisão.

No âmbito da manutenção de edifícios, importa perceber como é que a aplicação das tecnologias de inteligência artificial a um histórico de dados promove a otimização de recursos energéticos, a boa arte da manutenção preditiva e um desenvolvimento sustentável numa unidade hoteleira.

Assim, o principal objetivo desta dissertação é desenvolver uma ferramenta de apoio à gestão de energia, baseada em modelos de previsão de consumo elétrico. Essa ferramenta visa propor soluções que aumentem a eficiência energética do hotel, permitindo enfrentar os desafios associados à pegada de carbono, identificar possíveis problemas nos sistemas do hotel e delinear caminhos para sustentabilidade futura. No final deste trabalho será possível quantificar a redução de pegada carbónica associada à redução de consumos energéticos, às ações de melhoria para a sustentabilidade ambiental de um empreendimento hoteleiro.

2

ESTADO DA ARTE

2.1 TURISMO SUSTENTÁVEL

A Organização Mundial do Turismo (OMT) define o turismo sustentável como aquele que "tem plenamente em conta os impactos atuais e futuros, económicos, sociais e ambientais, para responder às necessidades dos visitantes, da indústria, do ambiente e das comunidades locais" (OMT, 2005)[2]. O turismo sustentável está intrinsecamente ligado ao conceito de desenvolvimento sustentável, consagrado pelo Relatório Brundtland (1987), que defende a satisfação das necessidades presentes sem comprometer as gerações futuras. No turismo, este princípio traduz-se na procura de um equilíbrio entre a viabilidade económica da atividade, a preservação ambiental e o respeito pelas comunidades locais.

O "Guia de Boas Práticas de Sustentabilidade para a Animação Turística" (Turismo de Portugal, 2023) também explica como o turismo sustentável deve integrar as três dimensões fundamentas: a ambiental, a social e a económica. Reforça que a Estratégia Turismo 2027 (ET27) para Portugal já se compromete com o papel do setor na concretização dos Objetivos de Desenvolvimento Sustentável (ODS) das Nações Unidas, posicionando o país como um dos destinos mais competitivos e sustentáveis do mundo através de objetivos estratégicos que abrangem estas três áreas [3]. Os objetivos estratégicos são explicados como:

A *Dimensão Ambiental,* assente na preservação dos recursos naturais e na minimização dos impactos negativos da atividade turística. No âmbito da ET27 são definidos objetivos claros, como o desenvolvimento da eficiência energética, a promoção de uma gestão racional da água e a implementação de práticas eficazes de gestão de resíduos. Relativamente à eficiência energética e neutralidade carbónica, as empresas do setor devem alinhar-se com o Roteiro para a Neutralidade Carbónica (RNC 2050) e com o Plano Nacional Energia e Clima 2030 (PNEC 2030), procurando reduzir as emissões de gases com efeito de estufa. Entre as boas práticas destacam-se a utilização de veículos ambientalmente mais eficientes, a monitorização do consumo de combustível e eletricidade, a autoprodução ou aquisição de energia proveniente

de fontes renováveis, bem como a medição e compensação das emissões de carbono. A meta estabelecida prevê que mais de 90% das empresas do turismo adotem medidas de utilização eficiente de energia. A eficiência hídrica é considerada essencial para assegurar a disponibilidade de água em qualidade e quantidade adequadas. As empresas são incentivadas a planear e gerir os seus consumos através da manutenção regular de equipamentos para prevenir fugas, do registo e análise sistemática do consumo, da realização de auditorias, da instalação de dispositivos de poupança de água e da sensibilização de colaboradores e clientes. Também neste domínio, a ET27 estabelece como objetivo que mais de 90% das empresas promovam uma utilização eficiente da água.

No que se refere à economia circular e gestão de resíduos, adota-se o princípio dos "4R" (Redução, Reutilização, Recuperação e Reciclagem) como forma de minimizar o desperdício. Entre as medidas recomendadas incluem-se a disponibilização de recipientes para deposição diferenciada de resíduos, a preferência por produtos de limpeza com rótulo ecológico, a compra a granel e de produtos locais, bem como a eliminação progressiva de artigos plásticos de utilização única. A meta definida aponta para que mais de 90% das empresas desenvolvam ações consistentes de gestão eficiente de resíduos. Por fim, a conservação da biodiversidade e da natureza é entendida como indissociável do desenvolvimento económico. As empresas turísticas são chamadas a participar em ações de voluntariado ambiental, a envolver-se em projetos de conservação e a transmitir informação sobre os recursos naturais, de modo a promover comportamentos responsáveis e adequados entre os clientes.

A *Dimensão Social* do turismo sustentável visa assegurar que a atividade turística gera impactos positivos nas populações residentes e promove a inclusão social. A acessibilidade é entendida de forma abrangente, não se restringindo apenas às pessoas com deficiência, mas englobando também famílias com crianças, idosos, grávidas e outros grupos com necessidades específicas. Trata-se, simultaneamente, de uma responsabilidade social e de uma oportunidade de negócio no crescente mercado do Turismo Acessível. Entre as boas práticas destacam-se a disponibilização de estacionamento reservado, a oferta de informação detalhada sobre acessibilidade, o acesso a cadeiras de rodas, a garantia de boas condições de circulação e a existência de instalações sanitárias adaptadas.

No que respeita à gestão de recursos humanos, reconhece-se que as pessoas são o verdadeiro motor da mudança. Assim, é fundamental promover o equilíbrio entre vida pessoal e profissional, a igualdade de género, a criação de um ambiente de trabalho seguro, a formação periódica, a realização de avaliações de desempenho e a garantia de condições laborais justas, incluindo remuneração adequada e horários flexíveis. A valorização de colaboradores locais,

através da criação de bolsas de emprego, bem como o incentivo ao voluntariado ambiental e cultural, reforçam a ligação entre a atividade turística e a comunidade. A relação com as comunidades locais constitui outro pilar essencial. O turismo sustentável deve contribuir para o bem-estar e a qualidade de vida das populações residentes, partilhando tradições, incentivando a aquisição de produtos e serviços locais, desenvolvendo atividades que envolvam a comunidade e apoiando iniciativas de preservação do património e das tradições.
A *Dimensão Económica* do turismo sustentável centra-se na viabilidade a longo prazo do negócio, na criação de riqueza e na distribuição equitativa dos benefícios. A sustentabilidade económica do negócio implica a capacidade das empresas desenvolverem as suas atividades de forma contínua, assegurando a perenidade do setor, a regeneração e valorização dos territórios e comunidades, bem como o bem-estar dos colaboradores.
A sustentabilidade é um pilar fundamental no turismo pois assegura a preservação dos recursos naturais, culturais e sociais dos destinos, garantindo a sua atratividade e longevidade. Promove um setor mais equilibrado e consciente, ao preparar profissionais para enfrentar desafios ambientais, sociais e económicos. No contexto da hotelaria, estratégias sustentáveis incluem o uso de recursos locais, a gestão ecológica dos transportes, a eficiência energética e hídrica, assim como a adoção dos princípios da economia circular. Embora algumas soluções exijam um investimento inicial mais elevado, revelam-se rentáveis a médio e longo prazo, ao recorrerem a fatores de produção renováveis. Além de reduzir o impacto ambiental e combater o desperdício, a sustentabilidade reforça a competitividade das empresas, responde às expectativas dos consumidores e contribui para a inclusão e o desenvolvimento das comunidades locais, beneficiando toda a cadeia de valor do turismo.

2.2 EFICIÊNCIA ENERGÉTICA E HÍDRICA

A gestão integrada de energia e água é hoje reconhecida como um pilar essencial da sustentabilidade empresarial, não apenas pela redução de custos operacionais, mas também pelo alinhamento com metas globais de descarbonização (ISO 50001:2018). Mais do que uma prática de controlo de custos, trata-se de uma estratégia de sustentabilidade que permite alinhar a operação com metas ambientais globais, como a descarbonização e a utilização racional dos recursos hídricos. A eficiência operacional traduz-se na capacidade de reduzir consumos sem comprometer o conforto dos utilizadores ou a qualidade dos serviços prestados. A aplicação de sistemas de monitorização contínua, suportados por indicadores de desempenho energético definidos pela ISO 50001:2018, possibilita a identificação de padrões de consumo, a deteção precoce de anomalias e a implementação de medidas corretivas ou preventivas. Estas medidas

incluem desde a otimização de sistemas de climatização e produção de água quente sanitária (AQS), até à integração de fontes renováveis e à automação inteligente de equipamentos. Estudos recentes demonstram que a hotelaria pode reduzir entre 15% e 25% do consumo energético através da aplicação de boas práticas de gestão e manutenção preventiva.

Conforme Fichera A, *(2020)* [4] explicou, um sistema de gestão de energia centra-se na medição, a monitorização e controlo do desempenho energético dos edifícios. Contudo, a aplicação dos princípios da norma ISO 50001:2018 pode ser desafiante. Nesse sentido, o autor propõe uma metodologia composta por sete etapas: definição dos limites físicos e funcionais para os indicadores de desempenho energético; identificação dos fluxos energéticos como eletricidade e gás; quantificação de variáveis influentes, como graus-dia e horas de escuridão, com recurso à análise estatística; recolha precisa de dados via sensores; definição dos indicadores de desempenho energético, como kWh/m²; estabelecer uma linha de base energética de 12 meses para comparação; e monitorização contínua com gráficos de controlo para detetar desvios e avaliar melhorias. A implementação eficaz destas etapas requer um sistema de gestão de energia ativo, formação estatística do gestor e disponibilidade de dados atualizados e coerentes com os objetivos da análise.

A manutenção dos ativos consumidores de energia, tem um grande impacto na gestão de energia de uma organização. O controlo de condição de equipamentos tem demonstrado que, no contexto empresarial, setores como o da hotelaria que historicamente dependem significativamente de recursos elétricos para atender às necessidades operacionais e oferecer serviços de qualidade aos hóspedes, enfrentam um grande desafio em conseguir otimizar essa variável. Estudos realizados em Portugal mostram que o consumo de eletricidade, no sector do turismo, representa cerca de 71% do valor global, que por sua vez representa cerca 390 000 emissões de GEE (t CO2e), em 2020 [5].

Conforme as boas praticas, não basta só reduzir eletricidade pela substituição de lâmpadas, pela troca de equipamentos mais eficientes ou por aplicar melhorias no isolamento. As otimizações das ferramentas de gestão técnica centralizada (GTC) também se evidenciam nos conceitos das boas praticas energéticas. A importância de uma boa gestão energética ao longo da operação anual poderá trazer impactos consideráveis na despesa anual dos seus recursos energéticos[6]. A previsão de consumos é uma boa prática de gestão da energia, usar esta ferramentas permite avaliar o desempenho do edifício, otimizar as operações, detetar e diagnosticar falhas de equipamentos[7].

No entanto, a eficiência hídrica deve ser considerada de forma complementar à energética, dado que ambos os recursos estão interligados. A água é utilizada intensivamente em hotéis, não apenas para consumo humano, mas também em piscinas, lavandarias, cozinhas e sistemas de rega. Além disso, a produção e o aquecimento de água representam uma parcela significativa do consumo energético, reforçando a necessidade de uma abordagem integrada (Becken, 2014)[8]. Estudos realizados em Portugal evidenciam que o setor do turismo apresenta consumos de água significativamente superiores à média urbana, com valores que podem variar entre 200 e 850 litros por hóspede/dia, dependendo da tipologia do hotel e da presença de infraestruturas como piscinas e spas (APA, 2020). Assim, a gestão eficiente da água não é apenas uma questão de responsabilidade ambiental, mas também de competitividade económica. As boas práticas de eficiência hídrica incluem a instalação de redutores de caudal, a reutilização de águas cinzentas para rega, a monitorização de perdas em piscinas e a implementação de sistemas inteligentes de irrigação baseados em sensores de humidade do solo.

A análise do consumo energético e hídrico, para além do potencial de gestão de despesas, ou custos, pode potenciar a manutenção de um equipamento ou de uma unidade técnica. É importante realçar que os métodos tradicionais de analise de condição de máquina são bastante interessantes, mas também devem ser complementados com outras variáveis e geridas via Gestão Técnica Centralizada (GTC). Nos softwares de GTC, mais do que controlo e instrução operativa também é importante gerir a manutenção através de soluções de monitorização de variáveis e gestão de alarmes. Com base nesta ideia, associam-se as tecnologias de IA por forma a que se possam prever ocorrências de falha e otimização energética.

Interessa perceber como é que a aplicação das tecnologias de IA a um conjunto de dados históricos promove a otimização de recursos energéticos e a manutenção preditiva, contribuindo assim para a adoção de práticas sustentáveis numa unidade hoteleira.

Pensa-se na simples observação dos consumos elétricos, ou dos consumos de energia, como eles se transformam em indicadores de condição. Condição, do próprio ativo pois este tende para um tipo de consumo, num determinado regime de operação. Já temos um sintoma de falha, aplicado por vários tempos na técnica de manutenção. Por forma a se conseguir interpretar os consumos de energia de um determinado ativo, é necessário identificar e compreender quais os fatores externos que podem ser influentes. É crucial que se identifique as variáveis independentes para prever o consumo energético, assim como para aumentar a eficácia da previsão e as medidas de eficiência energética[9].

Parte da energia utilizada pela sociedade é devida aos edifícios, estes consumem entre 20 a 40% do total, sendo que metade desse consumo está relacionado aos sistemas de aquecimento, ventilação e ar condicionado (AVAC). Dentro do AVAC, os chillers ou as bombas de calor, são os equipamentos principais e estes representam um grande consumo energético, mais de 50% desse consumo total. Portanto, otimizar o modo de funcionamento dos chillers é crucial para reduzir o consumo de energia para todo o sistema AVAC, e até mesmo para o consumo total do edifício [9].

A quantidade de energia consumida pelos chillers está diretamente ligada aos pontos de ajuste da temperatura de água gelada fornecida e ao estado do seu funcionamento. Além disso, fatores como capacidade frigorifica e clima também influenciam o seu consumo. Pesquisas indicam que a seleção e a regulação adequada das variáveis de operação dos chillers podem resultar numa redução de energia até 25%, mantendo o conforto interno. O funcionamento eficiente dos chillers depende do seu sistema de controle e das variáveis de otimização. Isso sugere que a eficiência energética dos chillers está intrinsecamente ligada à maneira como são controlados e às configurações usadas para operá-los de forma otimizada [11].

2.3 CONCEITOS DA INDÚSTRIA 4.0

A Indústria 4.0 (I4.0) é reconhecida como a Quarta Revolução Industrial, caracterizada pela digitalização e integração de tecnologias avançadas nos processos produtivos. O conceito foi inicialmente apresentado na Feira de Hannover (Alemanha, 2011) como parte de uma estratégia nacional para modernizar a indústria através da automação inteligente e da interconexão digital (Kagermann, Wahlster & Helbig, 2013)[12].

O alicerce da Indústria 4.0 encontra-se nos sistemas ciber-físicos (CPS, do inglês *Cyber-Physical Systems*.), que permitem a comunicação em tempo real entre máquinas, sensores, produtos e pessoas, criando cadeias de valor inteligentes e altamente flexíveis. Estas cadeias de valor inteligente, são implementadas com três características: 1. horizontal (entre empresas e processos), 2. ponta a ponta (ao longo de todo o ciclo de vida do produto, baseada em TIC.) e 3. vertical (entre os diferentes níveis da fábrica). As suas principais vantagens incluem otimização em tempo real, flexibilidade para personalização, resiliência a falhas, transparência nos processos e maior eficiência de recursos e energia.

As tecnologias que fomentam a indústria 4.0 são multifacetadas e podem ser categorizadas em fundamentos tecnológicos, infraestruturas de rede e sistemas de software de suporte:

O elemento tecnológico central da indústria 4.0, conforme já referido são os CPS, compostos por microcomputadores (os conhecidos *embedded systems*) preparados para serem conectados em redes sem fios e à Internet. Estes sistemas são capazes de trocar informações autonomamente, desencadear ações e controlar-se mutuamente de forma independente. A implementação da I4.0 requer a integração consistente de TIC nas estratégias tradicionais de alta tecnologia da indústria. A Internet das Coisas e Serviços (IoT) é o fator que desencadeia a quarta revolução industrial. Esta tecnologia permite a interligação de recursos, informação, objetos e pessoas.

A interconexão maciça necessária para a I4.0 exige o desenvolvimento e a expansão de infraestruturas de rede robustas e seguras, nomeadamente o lançamento do protocolo de internet IPv6 que aumentou drasticamente o número de endereços disponíveis e permitiu o *networking* direto e universal de objetos inteligentes através da internet. Plataformas de Baseadas em Nuvem, serve como o ambiente para a interconexão maciça de sistemas e o desenvolvimento de novos modelos de serviços e distribuição de dados. É essencial tanto para o armazenamento de informações recolhidas dos CPS quanto para a computação. Permite a partilha de recursos de processamento e dispositivos, bem como a execução de análises em qualquer lugar.

A inteligência da indústria 4.0 reside na capacidade de processar dados e gerir sistemas complexos através de software avançado. "Os *Algoritmos Inteligentes* podem ser aplicados a grandes quantidades de dados (*Big Data*) registados, por dispositivos inteligentes, para fornecer serviços inovadores"[12]. O conceito de algoritmos inteligentes é um pilar tecnológico da I4.0 e representa a aplicação prática de técnicas de inteligência artificial (IA) e ferramentas de aprendizagem de máquina (*machine learning*), permitindo assim processos adaptativos e preditivos aplicáveis na análise de tendências, monitorização de processos, diagnóstico de falhas e controlo de processos.

A modelagem e a simulação, portanto, a capacidade de simular/prever, é um componente vital para desenvolver sistemas de produção inteligentes. Pode ser usada para prever o comportamento de sistemas reais, validar escolhas de design, apoiar decisões de agendamento e intervenções de manutenção. A robotização, remete para o papel dos sistemas automatizados avançados e, em particular, dos robôs, dentro da nova infraestrutura de produção em rede, projetados para comunicar e aprender a interagir diretamente com o operador humano.

No âmbito da manutenção de ativos, a evolução das práticas tradicionais para modelos de manutenção preditiva e manutenção 4.0 tem sido impulsionada pela digitalização e pela integração de sensores inteligentes. Estes sistemas permitem recolher dados em tempo real

sobre o desempenho dos equipamentos, correlacionando falhas potenciais com consumos energéticos anómalos, Silvestri et al. 2020[13]. A possibilidade de gerar novo conhecimento a partir do processamento de dados é o aspeto chave para esta manutenção inteligente. Novas tecnologias como monitorização baseada em sensores e planeamento dinâmico são cruciais para a redução de custos de manutenção em 20% a 30%[14]

As técnicas de monitorização têm sido alvo de um crescente interesse por parte da comunidade científica e industrial, estas técnicas podem permitir um prognóstico fiável e ainda reduzir em grande escala os investimentos em manutenção e melhorar os resultados globais das organizações [15]. Técnicas que utilizam a inteligência artificial, poderão ter grande impacto e são por isso, uma das áreas que maior interesse tem despertado, assim como o crescente número de publicações.

A tomada de decisão é um elemento de grande importância no mundo das indústrias, o que nos leva a uma necessidade de procurar o máximo de informação confiável para ter eficiência ao tomá-la, pois esta é a fase que reflete o processo de estudo de todos os dados adquiridos.

Confiar em dados gerados sinteticamente e produzidos por plataformas de teste ou modelos matemáticos pode não generalizar bem para as condições reais de um bem. Quanto melhor forem as variáveis produzidas, no tratamento de dados durante o ciclo de vida do respetivo ativo, melhor será o planeamento das intervenções de manutenção, melhor se conserva sua condição, assim como melhor se gere os recursos inerentes.

Conforme F.A. Alenizi [16] com a evolução tecnológica e industrial, a tendência é reduzir os custos, gerir recursos e ativos, assim como procurar a melhor qualidade com eficiência. O conceito de indústria 4.0 surge devido a muitos estudos relacionados com o tema, também referente ao avanço na indústria de transformação ao longo da *Era Digital*. Sem dúvida que a ciência da computação contribui para o desenvolvimento desta revolução industrial onde soluções como a inteligência artificial (IA) causam efeitos profundos nos sistemas de produção e modelos de negócio. Ainda assim é necessário compreender que o conceito I4.0, não é simplesmente a digitalização dos processos mecânicos, pois isso já persiste por vários tempos. Deve-se manter o foco em assumir que recolha de dados em tempo real poderá tornar as operações muito mais dinâmicas, quando esses dados foram tratados e transformados em indicadores para a decisão [17].

Os conceitos da Quarta Revolução Industrial, dão enfase à introdução de tecnologias digitais no setor industrial ou hoteleiro. Destacam-se a conexão entre I4.0 e eficiência energética, a integração com a IoT e os desafios enfrentados pelas empresas. Além disso, são exploradas pesquisas sobre a interseção entre economia circular, a produção inteligente e gestão de

energia, evidenciando a importância de sistemas de monitorização de energia e a implementação de tecnologias I4.0.

Alarcón, et al., em *Renewable and Sustainable Energy Reviews* (2021) evidencia que diversos estudos destacam a relação entre a I4.0 e a eficiência energética, a integração de manutenção e gestão de energia, bem como a ligação entre economia circular e cadeia de suprimentos. Também, que as pesquisas revelam desafios na implementação de tecnologias I4.0, especialmente no que diz respeito à adoção de ferramentas baseadas em IoT e a analise da *Big Data*. Alem disso, destaca os benefícios na relação da monitorização de desempenho, da manutenção preditiva com a redução dos custos energéticos. [18]

As tendências sublinham mais do que nunca a necessidade de utilizar a aprendizagem de máquina, ou os conhecidos algoritmos de *machine learning* (ML), assim como outras tecnologias de IA para ajudar nesta transformação. A IA pode ajudar, com a sua capacidade de aprender e de se adaptar a ambientes em mudança, com a sua capacidade de obter informações breves e perspicazes a partir de dados complexos e de alta dimensão e com a sua aptidão para descobrir automaticamente padrões e relações nos dados. Os conceitos da indústria 4.0, como qualidade 4.0, dependem fortemente do uso de IA. [19]

A IA pode ajudar, com a sua capacidade de aprender e de se adaptar a ambientes em mudança, com a sua capacidade de obter informações breves e perspicazes a partir de dados complexos e de alta dimensão e com a sua aptidão para descobrir automaticamente padrões e correlações nos dados.[16]

As ferramentas destinadas ao controlo ou, mais corretamente, à monitorização do consumo energético devem assegurar a supervisão contínua e a aquisição sistemática de dados. Os softwares SCADA (*Supervisory Control and Data Acquisition*) desempenham um papel central neste processo, permitindo recolher informação ao longo do tempo e, assim, construir um histórico operacional do ativo. Este registo, que deve abranger desde a fase de aquisição até ao fim de vida útil do equipamento, constitui a base para decisões informadas de gestão e manutenção.

2.4 CONCEITOS SOBRE INTELIGÊNCIA ARTIFICIAL

A Inteligência Artificial (IA), no enquadramento de Mitchell (1997), deve ser entendida como o esforço científico e tecnológico de desenvolver sistemas computacionais capazes de aprender a partir da experiência e com isso melhorar o seu desempenho em tarefas específicas. Tom M. Mitchell é um cientista da computação norte-americano, pioneiro no campo da aprendizagem de máquina, cuja obra “Machine Learning” em 1997, ajudou a consolidar a IA

como disciplina científica e aplicada. O autor coloca a aprendizagem de máquina, ou o *machine learning* (ML), no centro da IA definindo-a de forma precisa: "Um programa de computador é dito aprender a partir da experiência E, relativamente a uma classe de tarefas T e a uma medida de desempenho P, se o seu desempenho em tarefas de T, medido por P, melhora com a experiência E." - (Mitchell T, 1997)[20].

A IA é um campo multidisciplinar da ciência da computação que procura desenvolver sistemas capazes de executar tarefas que normalmente requerem inteligência humana, como raciocínio, aprendizagem, perceção e tomada de decisão. O ML é um subcampo da IA que permite que sistemas aprendam padrões a partir de dados em vez de dependerem apenas de regras de programação explicitas explícitas. Os principais paradigmas incluem a aprendizagem supervisionada, a não supervisionada e a aprendizagem por reforço.
Segundo Russell & Norvig, 2021, a Inteligência Artificial é um campo multidisciplinar que procura desenvolver sistemas capazes de executar tarefas que normalmente requerem inteligência humana, como raciocínio, aprendizagem, perceção e tomada de decisão[21]. Dentro deste domínio, destacam-se várias áreas fundamentais:

- *Machine Learning* (Aprendizagem de Máquina): constitui um dos subcampos mais relevantes da IA, focando-se no desenvolvimento de algoritmos e modelos que permitem aos sistemas aprender a partir de dados e melhorar o desempenho em tarefas específicas sem necessidade de programação explícita. Mitchell (1997) definiu-o de forma clássica como a capacidade de um programa melhorar o seu desempenho numa tarefa (T), medido por uma métrica (P), com base na experiência (E). Os principais paradigmas incluem a aprendizagem supervisionada, a não supervisionada e a aprendizagem por reforço.
- *Deep Learning* (Aprendizagem Profunda): ramo do machine learning que utiliza redes neuronais artificiais (RNA) profundas, compostas por múltiplas camadas, capazes de modelar padrões complexos em grandes volumes de dados. Este modelo tem sido responsável por avanços significativos em visão computacional, reconhecimento de fala e processamento de linguagem natural (Schmidhuber, 2015)[22].
- *Knowledge Representation and Reasoning* (Representação e Raciocínio do Conhecimento): área que estuda como o conhecimento pode ser representado computacionalmente (através de lógicas formais, ontologias ou grafos de conhecimento), permitindo que os sistemas raciocinem, infiram novas relações e resolvam problemas complexos.

- *Robotics* (Robótica): integra hardware e software para criar máquinas inteligentes capazes de executar tarefas físicas ou interagir com o ambiente. A robótica moderna combina perceção, planeamento e ação, sendo aplicada em setores como a indústria, saúde e serviços.
- *Computer Vision* (Visão por Computador): campo que procura dotar os computadores da capacidade de interpretar e processar informação visual, como imagens ou vídeos, de forma semelhante à visão humana. Avanços em redes convolucionais (CNNs) revolucionaram este domínio [23], permitindo aplicações em veículos autónomos, diagnóstico médico e segurança.
- *Natural Language Processing* (Processamento de Linguagem Natural – PLN): área que desenvolve sistemas capazes de compreender, interpretar e gerar linguagem humana. O surgimento da arquitetura de rede neuronal *The Transformers* impulsionou aplicações como tradução automática, chatbots e análise de sentimentos, tornando o PLN uma das áreas mais dinâmicas da IA contemporânea (Vaswani et al., 2017) [24].

Um dos conceitos mais discutidos na literatura sobre Inteligência Artificial é o da "caixa negra". Este termo refere-se a modelos de elevada complexidade, como as redes neuronais profundas, nos quais não é possível compreender de forma clara como as decisões ou previsões são geradas. Nestes casos, o utilizador apenas observa os dados de entrada e os resultados de saída, sem acesso transparente ao processo interno de decisão. Conforme Schmidhuber, 2015, essa falta de transparência levanta desafios significativos em termos de confiança, auditoria e responsabilidade, sobretudo em áreas críticas como a saúde, a justiça ou a gestão de recursos. Por outro lado, existem os chamados modelos de "caixa branca", que são totalmente interpretáveis. Exemplos incluem regressões lineares, árvores de decisão ou sistemas baseados em regras, nos quais é possível identificar claramente a influência de cada variável no resultado. Estes modelos são preferidos em contextos onde a explicabilidade é mais importante do que a precisão preditiva. Entre os dois extremos, surgem ainda os modelos de "caixa cinzenta", que combinam componentes interpretáveis com partes mais "opacas", procurando equilibrar precisão e transparência.

Mais recentemente, tem-se falado no conceito de "caixa de vidro", associado ao movimento da IA explicável (XAI – *Explainable Artificial Intelligence*). O objetivo é desenvolver técnicas que tornem modelos complexos mais transparentes, sem comprometer o seu desempenho. Métodos como o LIME (*Local Interpretable Model-agnostic Explanations*), o SHAP (SHapley Additive exPlanations), os mapas de calor em redes convolucionais ou os

mecanismos de atenção em Transformers são exemplos de abordagens que procuram abrir a "caixa negra" e fornecer explicações compreensíveis para humanos (Ribeiro et al., 2016)[25]. Assim, o debate em torno da "caixa negra" não se limita a uma questão técnica, mas envolve também dimensões éticas e sociais. O grande desafio atual da inteligência artificial é encontrar um equilíbrio entre precisão preditiva e explicabilidade, garantindo que os sistemas não sejam apenas poderosos, mas também confiáveis, auditáveis e alinhados com princípios de transparência.

2.4.1 APLICAÇÃO NA HOTELARIA

A IA afirma-se progressivamente como uma tecnologia transformadora, capaz de redefinir paradigmas estabelecidos em diversos setores. Considerando a transformação digital da hotelaria, a IA constitui-se um acelerador para eficiência operacional, personalização da experiência do hóspede e outros avanços na sustentabilidade. A sua aplicação prática exige compreensão dos conceitos fundamentais e dos desafios éticos e técnicos associados.

A Inteligência Artificial tem vindo a transformar a hotelaria em múltiplas dimensões, desde a relação com o cliente até à gestão de recursos. No atendimento ao cliente, soluções como chatbots baseados em PLN e assistentes virtuais nos quartos permitem reservas, pedidos e apoio 24/7, reduzindo custos e aumentando a conveniência. Estudos recentes mostram que tecnologias de check-in remoto e gestão inteligente de acessos são particularmente valorizadas em contextos pós-pandemia, pela redução do contacto físico e pela agilidade no serviço [26].

Na gestão de receitas, a IA é aplicada em modelos preditivos de procura e otimização dinâmica de tarifas, integrando-se com sistemas de *business intelligence* para apoiar decisões estratégicas. Estes métodos permitem ajustar preços em tempo real, maximizando a rentabilidade e a taxa de ocupação.

Outro eixo central é a personalização da experiência do hóspede, onde algoritmos de recomendação analisam preferências e históricos de consumo para sugerir serviços, enquanto técnicas de segmentação comportamental suportam campanhas de marketing mais eficazes. Esta personalização é apontada como um fator-chave para a fidelização e diferenciação competitiva [27].

No plano da eficiência operacional, a IA contribui para a automação de tarefas administrativas e para a manutenção preditiva, através de sensores e algoritmos que antecipam falhas técnicas, reduzindo custos e aumentando a fiabilidade dos serviços. Por fim, no domínio da sustentabilidade, a hotelaria tem adotado sistemas de monitorização inteligente de consumos energéticos e hídricos, apoiados por modelos preditivos que identificam desperdícios e

sugerem intervenções, alinhando-se com metas de eficiência e responsabilidade ambiental [14][21].

A inteligência artificial representa uma oportunidade estratégica para o setor hoteleiro, promovendo inovação, eficiência e sustentabilidade. A sua implementação deve ser fundamentada, ética e alinhada com os valores da hospitalidade. A literatura recente aponta para um futuro promissor, onde a IA será parte integrante da experiência hoteleira, desde a operação até à personalização e gestão ambiental.

2.5 MONITORIZAÇÃO

A monitorização constitui um processo sistemático de recolha, análise e interpretação de dados com o objetivo de acompanhar o desempenho de sistemas, processos ou organizações ao longo do tempo. No contexto da gestão de energia e recursos em edifícios de serviços, como as unidades hoteleiras, a monitorização é um elemento central para a eficiência operacional, sustentabilidade e melhoria contínua.

Segundo Cabello Eras et al. (2016), a monitorização energética em hotéis permite identificar padrões de consumo, detetar desvios em tempo real e apoiar a implementação de medidas de eficiência, reduzindo custos e emissões de carbono. De forma semelhante, Santiago (2021)[28],destaca que a recolha sistemática de dados em unidades hoteleiras das Canárias possibilitou não apenas a quantificação detalhada dos consumos por setor (quartos, cozinha, spa, iluminação comum), mas também a identificação de oportunidades de integração de energias renováveis e de redução da pegada carbónica.

A literatura científica demonstra que a monitorização contínua de consumos energéticos e hídricos permite identificar padrões de utilização, detetar anomalias e implementar medidas de eficiência que, de outra forma, permaneceriam ocultas (Teixeira, 2021)[29].

A monitorização deve estar alinhada com normas internacionais como a ISO 50001:2018, que estabelece a necessidade de definir linhas de base energéticas e indicadores de desempenho energético para avaliar melhorias de forma contínua[30].

No setor hoteleiro, a monitorização assume várias dimensões:

- Operacional: acompanhamento de consumos energéticos e hídricos em tempo real, permitindo identificar desperdícios e otimizar processos.
- Manutenção preditiva: sensores e sistemas de monitorização contínua antecipam falhas técnicas em equipamentos críticos (AVAC, Caldeiras, Centrais de Bombagem).

- Sustentabilidade: monitorização de indicadores ambientais (emissões, consumos por hóspede/noite) para alinhamento com certificações ambientais e metas de descarbonização.
- Gestão estratégica: integração dos dados monitorizados em sistemas de *Business Intelligence*, apoiando decisões de investimento e benchmarking entre unidades.

Os gráficos de controlo são ferramentas estatísticas desenvolvidas para o controlo e monitorização em tempo real do desempenho de um processo, sendo largamente aplicados, inclusive na gestão de energia. No contexto da gestão de recursos energéticos e hídricos, adaptado a unidades hoteleiras, os gráficos de controlo são cruciais para a monitorização e para a deteção de desvios anómalos nos consumos, permitindo identificar rapidamente situações de ineficiência, desperdício ou falhas técnicas em equipamentos. Através da definição de linhas de base energéticas e de indicadores de desempenho, os gráficos de controlo possibilitam comparar o consumo real com o esperado, distinguindo variações normais de variações especiais que exigem intervenção[4].

Em unidades hoteleiras, este método é particularmente relevante para acompanhar consumos de energia elétrica, água e climatização, áreas que representam uma fatia significativa dos custos operacionais. A utilização de gráficos de controlo, aumenta a sensibilidade da monitorização, permitindo não só detetar anomalias em tempo real, mas também avaliar tendências de melhoria ou degradação do desempenho energético ao longo do tempo.

É importante considerar, os gráficos de controlo enquadram-se como uma ferramenta essencial de gestão proativa de recursos em hotelaria, alinhada com normas internacionais como a ISO 50001, promovendo eficiência, sustentabilidade e melhoria contínua.

Assim, a monitorização não deve ser entendida apenas como recolha de dados, mas como um processo integrado de gestão, que fornece informação fiável e em tempo útil para suportar decisões estratégicas e operacionais. A sua aplicação em hotelaria contribui para a redução de custos, aumento da competitividade e cumprimento de metas de sustentabilidade, em linha com as exigências europeias regulamentadas e as expectativas dos clientes cada vez mais atentos à responsabilidade ambiental.

3

METODOLOGIA

A análise crítica da literatura especializada evidencia uma lacuna substancial na integração de metodologias baseadas em Inteligência Artificial (IA) aplicadas à gestão da manutenção no setor hoteleiro. A incorporação destas tecnologias configura-se como um elemento estratégico para a otimização da eficiência operacional, a redução de custos, o aumento da disponibilidade e fiabilidade dos equipamentos, bem como para o fortalecimento de práticas de gestão energética e ambientalmente responsáveis.

A metodologia delineada inicia-se com uma revisão sistemática da literatura, abrangendo estudos científicos e técnicos de âmbito nacional e internacional, com o objetivo de caracterizar o estado da arte relativamente a temáticas como a eficiência energética de sistemas, as tecnologias de monitorização e aquisição de dados, e os paradigmas emergentes da indústria 4.0. Esta etapa contempla a identificação das variáveis com maior correlação com o consumo energético e hídrico, assim como a definição de variáveis derivadas que possam complementar e enriquecer um sistema de monitorização inteligente. No contexto específico da unidade hoteleira em estudo, a análise incidirá sobre os consumos de energia elétrica, água potável e gás propano, considerando tanto os valores globais como a sua desagregação por setores e pontos de utilização.

Por forma a compreender como fatores como ocupação e sazonalidade influenciam o comportamento padrão de consumos na unidade hoteleira de recursos ao longo do tempo analisaram-se séries temporais relacionados com o consumo diário de energia elétrica, água e gás numa instalação monitorizada.

Os principais métodos utilizados para caracterização dos consumos de recursos de energia e água potável são descritos seguidamente:

i. Estruturação dos Dados: Os dados utilizados foram adquiridos a partir de registos diários referentes aos valores acumulados dos contadores de energia elétrica, água potável e gás propano, devidamente organizados em tabelas por tipo de recurso. Esses

dados foram integrados em conjuntos de informações e submetidos a processos de validação, uniformização e organização temporal.

ii. Series Temporais: Criação de variáveis derivadas que representam diferentes dimensões temporais (como ano, mês, trimestre, dia da semana, semana do ano, entre outros), permitindo a análise dos consumos com base em padrões sazonais e comportamentais.

iii. Análise Estatística: aplicação de estatísticas descritivas, tais como o uso de médias, medianas, quartis evidenciados em diagramas de caixa ao longo das series temporais.

iv. Observações de tendências e flutuações nos consumos, tanto nos globais como nos parciais, ao longo do tempo. Foram desenvolvidas visualizações que possibilitam a identificação de comportamentos típicos, picos de uso e possíveis anomalias.

v. Análise por Setores: Para alem da análise de valores globais, os consumos foram segmentados por áreas/serviços e tipo de cliente, permitindo observar como diferentes ambientes contribuem para o perfil geral de consumo. A desagregação de consumo, através da identificação dos consumidores parciais viabiliza a identificação de setores com maior potencial de otimização.

vi. Investigação de correlações: Para compreender como a ocupação afeta o consumo de recursos, foi realizada uma avaliação das correlações entre variáveis que representam a ocupação e os valores de consumo registados. Posteriormente, modelos estatísticos foram aplicados para quantificar o impacto da ocupação sobre o consumo de energia elétrica.

vii. Avaliação de variáveis indicadores de eficiência operacional: A eficiência operacional foi analisada através da construção e interpretação de variáveis derivadas que relacionam o consumo de energia e água com métricas de ocupação e atividade. Esta abordagem permitiu identificar desvios relativos e padrões de desempenho ao longo do tempo. Os principais procedimentos incluem: *Cálculo de variações percentuais*: Foram estimadas as taxas de variação diária do número de clientes e do consumo energético, permitindo observar a sensibilidade do consumo face à flutuação da ocupação. *Erro relativo entre consumo e ocupação*: A métrica foi definida como o desvio percentual entre a variação do consumo e a variação da ocupação, normalizado pela magnitude da ocupação. Esta variável permite identificar dias em que o consumo não acompanha proporcionalmente a presença de clientes, sugerindo ineficiências ou comportamentos atípicos. *Eficiência por cliente*: A variável consumo por cliente foi monitorizada ao longo do tempo e segmentada por períodos sazonais. A respetiva

variação foi utilizada como indicador direto da eficiência energética por unidade de ocupação.

viii. Modelagem/simulação: Foi aplicada uma técnica de aprendizagem de máquina a este conjunto de dados, com o objetivo de otimizar os consumos globais do sistema. Para isso, procedeu-se ao tratamento e processamento de um bloco de dados que refletem um padrão representativo do comportamento do caso em estudo. Essa abordagem visa apoiar a análise da condição do ativo, proporcionando insights que auxiliem na sua gestão e operação.

3.1 DEFINIÇÃO CONTEXTUAL E TÉCNICA

O presente estudo tem como foco a análise integrada dos consumos de energia elétrica, água e gás numa unidade hoteleira, localizadas na região turística do Algarve. Localizado em Olhos de Água, Albufeira, este hotel de grande dimensão distingue-se pela sua estrutura imponente e pela diversidade de serviços oferecidos, em regime de tudo incluído. Com um total de 500 quartos, distribuídos por um edifício principal, que concentra 450 quartos, e um edifício adicional com 50 quartos. A oferta gastronómica é composta por 4 restaurantes, apoiados por duas cozinhas principais (cada uma servindo dois restaurantes) e complementada por uma cozinha de pastelaria. No que respeita ao lazer e bem-estar, os hóspedes podem usufruir de 5 bares, 5 piscinas exteriores, um spa, um ginásio e ainda uma discoteca. Para suporte operacional, o hotel dispõe ainda de uma lavandaria própria. O sistema de AVAC do hotel é composto por ventiloconvetores para climatização dos quartos e por unidades de tratamento de ar (UTA) que asseguram a ventilação e climatização das zonas comuns e de lazer. A produção de águas quentes sanitárias (AQS) é garantida por caldeiras a gás propano, que também fornecem aquecimento. O arrefecimento é realizado por um grupo de chillers instalados em paralelo. A base de dados utilizada foi composta por medições diárias registadas ao longo de um período de 3 anos, permitindo examinar tendências sazonais e padrões comportamentais de uso. A análise incide sobre dados diários entre abril a outubro de 2023, 2024 e desde abril até agosto de 2025, meses que concentram o maior fluxo turístico e, consequentemente, maior requisição dos recursos energéticos.

O contexto envolve a avaliação da eficiência do consumo em relação à ocupação dos espaços, sendo esta mensurada por variáveis como o número de ocupantes e sua distribuição ao longo do tempo. O objetivo é entender como fatores operacionais e sazonais influenciam o consumo de recursos, assim como quais os que devem ser utilizados para apoiar as decisões sobre otimização e sustentabilidade. Um dos objetivos principais deste estudo será a conhecer os

locais que representam um maior consumo de energia em circuitos desagregados, relativamente aos consumos de energia e água, bem como um conjunto de boas práticas ambientais que permitam a criação de uma cadeia de responsabilidades, a diversos níveis, para a gestão de energia. No final, será possível identificar os recursos necessários para apoiar técnica e financeiramente a implementação das medidas de melhoria da eficiência energética, seria de extrema importância a formação das equipas de forma a dissemina à adoção de comportamentos energeticamente eficientes e boas práticas ambientais na estrutura interna da empresa.

O presente estudo foi desenvolvido com o auxílio da plataforma Google Colab, utilizando a linguagem de programação Python. Foi elaborado um script capaz de identificar padrões de consumo diário do hotel com base nas variáveis energéticas coletadas: eletricidade, água e gás. O Google Colab atua como uma plataforma central para o desenvolvimento rápido e colaborativo, combinando a recolha de dados, a modelação preditiva e a entrega de painéis interativos. A integração com sistemas de gestão técnica, em unidades hoteleiras proporciona uma implementação mais rápida, visibilidade dos dados em tempo real e a capacidade de iterar protótipos antes da migração para sistemas à escala, garantindo que as ferramentas digitais estão alinhadas com as necessidades operacionais.

O tratamento de dados, desenvolvido em Python dentro da plataforma Google Colab, utilizando bibliotecas como *pandas, numpy, matplotlib, seaborn* e *plotly* para tratamento e visualização de dados. As etapas técnicas envolvem:

- Importação e consolidação de dados de fontes externas (planilhas Excel separadas para energia, água e gás);
- Limpeza e padronização do grupo de dados;
- Criação de variáveis temporais (dia da semana, mês, trimestre, etc.) com base no índice de datas;
- Análise exploratória dos consumos agregados por cliente.
- Correlação entre consumo e ocupação utilizando *heatmaps* e *coeficientes de Pearson*;
- Criação de variáveis indicadoras de eficiência / performance.
- Aplicação de modelos de regressão linear para estimar o impacto da ocupação sobre o consumo de energia elétrica;
- Modelagem preditiva, onde foram aplicados modelos de ML e DL para prever consumos com base em ocupação, sazonalidade e perfil de clientes.

4

INTERPRETAÇÃO E TRATAMENTO DE DADOS

A avaliação do desempenho energético de uma unidade hoteleira fica refletido nas faturas, tais como, energia elétrica, água potável e consumo de gás propano, importante será refletir como essas variáveis dependentes se devem apresentar para quantificar a sustentabilidade dos recursos energéticos numa unidade.

Conhecer, através de uma boa monitorização, qual o comportamento dos consumidores de energia e água, e como podemos manipulá-los de forma a garantir desenvolvimento no processo de otimização de recursos energéticos. Ter em consciência que o produto desse comportamento é o tipo de operação do edifico, permite definir quais os objetivos e medidas de intervenção para se delimitar onde queremos estar no futuro.

A monitorização e análise de consumos não se limitam a um exercício de contabilização, mas configuram-se como um instrumento estratégico de manutenção e gestão energética. Na combinação de tarefas a se realizar para manter ou restaurar o bom funcionamento de um determinado ativo, analisar o seu histórico de dados e transformar dados operacionais em decisões estratégicas, será o princípio para a melhoraria da performance energética.

Conhecer os valores globais do consumo de energia e água, durante o passado, presente e futuro, ajudam a despistar como um hotel se comporta de forma temporal, concretamente em relação aos meses do ano, dias da semana, semanas do ano, ao ano e ao dia. Considerar a variável independente, ocupação, como o maior indicador de tendência de comportamentos poderá ajudar a prever como iremos estar em um futuro próximo.

4.1 INTRODUÇÃO ÀS VARIÁVEIS EM ESTUDO

4.1.1 CONSUMO DIÁRIO (KWH/DIA, M³/DIA, KG/DIA)

O consumo diário, expresso em diferentes unidades (energia elétrica em kWh, água em m³ e gás em kg), constitui a base para a monitorização do desempenho energético e hídrico de uma unidade hoteleira. Esta variável permite, identificar padrões de consumo ao longo do tempo,

distinguindo dias de maior ou menor intensidade energética. Também, relacionar consumos com variáveis climáticas e operacionais, como temperatura, humidade e taxa de ocupação. Detetar anomalias que possam indicar desperdícios, fugas ou ineficiências nos sistemas de climatização, aquecimento de água ou processos de lavandaria e cozinha.

A literatura sublinha que a análise diária é essencial para implementar estratégias de energy management, uma vez que possibilita ajustes operacionais em tempo quase real (Becken & Simmons, 2002)[31]

4.1.2 CONSUMO DIÁRIO POR CLIENTE (KWH/DIA/CLT, M³/DIA/CLT, KG/DIA/CLT)

A normalização do consumo em função do número de hóspedes é uma métrica crítica para a avaliação da eficiência energética em hotéis. Esta variável permite comparar períodos com diferentes níveis de ocupação, eliminando o viés causado por variações sazonais. Avaliar a intensidade de uso dos recursos por hóspede, indicador frequentemente utilizado em benchmarks internacionais de sustentabilidade hoteleira (Bohdanowicz & Martinac, 2007)[32]. Estabelecer metas de eficiência, alinhadas com certificações ambientais como ISO 50001. Assim, o consumo por cliente é um indicador mais robusto do que o consumo absoluto, pois reflete a eficiência relativa da operação.

4.1.3 VARIÁVEIS DE PEGADA CARBÓNICA

No âmbito da avaliação da sustentabilidade e do desempenho energético, foi necessário quantificar a pegada carbónica associada ao consumo de recursos. Para tal, foram aplicados fatores de emissão específicos a cada vetor energético e ao consumo de água, permitindo converter os consumos em emissões equivalentes de dióxido de carbono (CO_2e).

As emissões de GEE associadas ao consumo de água potável foram estimadas com base em fatores de emissão específicos, conforme orientações técnicas do RASARP 2024[33], onde o fator de emissão aplicado de 0,29 kg CO_2e/m^3 resulta da soma ponderada das emissões geradas nas duas principais fases do ciclo urbano da água: 0,06 kg CO_2e/m^3 correspondentes à distribuição em baixa, que inclui o consumo energético e emissões diretas da rede até o ponto de consumo, e 0,10 kg CO_2e/m^3 referentes à produção e transporte em alta, abrangendo o tratamento, captação e transporte da água até a rede de distribuição. Também, foram consideradas as emissões associadas à gestão de resíduos sólidos urbanos, com fatores de emissão de 0,04 kg CO_2e/kg para a recolha e 0,09 kg CO_2e/kg para a triagem e tratamento, refletindo o impacto das etapas logísticas e processuais do ciclo de resíduos.

Para a energia elétrica, foi adotado o fator de emissão de 0,167 kg CO_2e/kWh, conforme publicado pela Agência Portuguesa do Ambiente (APA) no relatório de 2025[34]. Relativamente ao gás propano, foi considerado o fator de emissão de 3,26 kg CO_2e/kg, conforme indicado pela DGEG (2025), "Trajetórias de Fatores de Conversão de Energia e de Emissão de GEE para Portugal"[35].

A criação destas variáveis permite quantificar a contribuição de cada vetor energético e hídrico para a pegada carbónica global. Identificar setores prioritários para intervenções de eficiência energética. Normalizar as emissões por ocupação, facilitando o benchmarking anual e a comparação com referências nacionais e internacionais. Apoiar a definição de indicadores de desempenho ambiental (KPIs) alinhados com políticas de sustentabilidade e metas de descarbonização.

4.1.4 Estatísticas Descritivas do Grupo de Dados

A caracterização estatística dos dados constitui uma etapa essencial na análise exploratória, permitindo compreender a sua distribuição, identificar valores atípicos e estabelecer limites de referência para a interpretação dos resultados. Entre os principais indicadores utilizados destaca-se a mediana, por sua vez, corresponde ao valor central da distribuição e é menos influenciada por *outliers*, sendo frequentemente mais representativa do comportamento típico do sistema. Os quartis, nomeadamente o primeiro *(Q1, 25%)* e o terceiro *(Q3, 75%)*, permitem avaliar a dispersão e a assimetria da distribuição, enquanto a amplitude interquartil *(IQR = Q3 – Q1)* mede a variabilidade central dos dados e é utilizada para identificar valores atípicos. Os valores mínimo e máximo representam os extremos da distribuição e fornecem informação sobre a amplitude total, ainda que possam ser influenciados por anomalias. A média representa o valor médio dos dados e traduz a tendência central, embora seja sensível a valores extremos. O desvio-padrão quantifica a dispersão em torno da média, sendo que valores elevados indicam maior variabilidade e valores baixos sugerem maior homogeneidade.

A utilização destes indicadores estatísticos permite identificar padrões de consumo típicos e desvios ocasionais, estabelecer limiares estatísticos para definição de níveis de alerta, apoiar a interpretação de representações gráficas como diagramas de caixa e fornecer uma base sólida para a aplicação de técnicas de modelação preditiva e de deteção de anomalias.

4.1.5 PREVISÕES / MODELAÇÃO DE VARIÁVEIS

A previsão de consumos, baseada em modelos estatísticos ou de *machine learning*, é uma ferramenta estratégica para a gestão energética. A sua importância reside em antecipar necessidades de energia e água, permitindo otimizar contratos de fornecimento e reduzir custos. Apoiar a tomada de decisão operacional, ajustando a climatização, a produção de água quente ou a rega em função das condições esperadas. Contribuir para a sustentabilidade, ao alinhar a operação com cenários de eficiência e redução de desperdícios. Estudos recentes demonstram que a previsão de consumos em hotéis, quando integrada com variáveis climáticas e de ocupação, pode reduzir em até 15% os custos energéticos.[36]

4.1.6 ERRO RELATIVO

O erro relativo mede a diferença entre os valores previstos e os valores observados, em termos percentuais. Esta variável é fundamental porque avalia a fiabilidade dos modelos de previsão, garantindo que as estimativas são suficientemente precisas para suportar decisões de gestão. Permite calibrar e melhorar continuamente os modelos, ajustando-os às especificidades do hotel. Evita decisões incorretas, que poderiam resultar de previsões enviesadas ou pouco representativas. Na literatura de gestão energética, a validação de modelos através do erro relativo é considerada uma prática essencial para assegurar a robustez das ferramentas de previsão.

Para interpretar o erro relativo diário de forma mais consistente, foram criadas variáveis auxiliares que contextualizam a relação entre consumo energético e ocupação. Entre elas destaca-se a variação percentual diária de clientes, obtida a partir da diferença no número de hóspedes entre dois dias consecutivos. Esta métrica expressa a taxa de variação da ocupação e permite identificar momentos em que ocorreram alterações abruptas no perfil de clientes. A sua inclusão na análise do erro relativo é essencial, pois desvios significativos no consumo podem estar diretamente associados a estas flutuações. Assim, a utilização de variáveis derivadas como a taxa de variação da ocupação possibilita uma leitura mais robusta do erro relativo, distinguindo entre desvios explicados por fatores operacionais (como variações de ocupação) e desvios que indiciam potenciais ineficiências ou anomalias no sistema energético.

4.1.7 VARIÁVEIS METEOROLÓGICAS E ÍNDICE DE CALOR

No âmbito da análise energética, foi necessário integrar variáveis meteorológicas que influenciam diretamente os padrões de consumo. Os dados de temperatura e humidade relativa

foram recolhidos a partir da estação meteorológica da unidade hoteleira, garantindo a representatividade das condições ambientais do período em estudo.

As variáveis consideradas foram:

- Temperatura média diária (ºC) – valor médio da temperatura do ar ao longo do dia, indicador da carga térmica global.
- Temperatura máxima (ºC) – pico diário de temperatura, relevante para identificar situações de stress térmico.
- Temperatura mínima (ºC) – valor mais baixo registado, associado ao conforto noturno e à inércia térmica do edifício.
- Amplitude térmica – diferença entre máximo e mínimos, indicador da variabilidade térmica diária.
- Humidade relativa média (%) – valor médio diário, indicador do conforto hidrotérmico.
- Humidade relativa máxima (%) – pico de saturação, associado a potenciais desconfortos e impacto em sistemas de ventilação.
- Humidade relativa mínima (%) – valor mais baixo registado, relevante para a perceção de secura do ar.
- Amplitude higrométrica – indicador da variabilidade da humidade relativa ao longo do dia.

A partir da combinação destas variáveis foi calculado o índice de calor, métrica que integra temperatura e humidade relativa para estimar a sensação térmica percebida pelos ocupantes. Este índice é particularmente relevante em contextos de elevada ocupação, como hotéis, uma vez que traduz de forma mais realista o impacto das condições ambientais no conforto e no consumo energético associado a sistemas de climatização. A inclusão destas variáveis no modelo explicativo permite relacionar picos de consumo energético com condições meteorológicas extremas, justificar desvios episódicos nos consumos diários através de fatores ambientais, apoiar a definição de limiares de alerta para situações críticas de conforto térmico. Também, reforçar a robustez dos modelos preditivos, ao integrar variáveis independentes que condicionam a requisição energética.

Assim, a utilização de indicadores meteorológicos e do índice de calor constitui uma etapa essencial para a compreensão integrada do desempenho energético, permitindo análises mais consistentes e alinhadas com a realidade operacional.

4.2 VALORES GLOBAIS

Em análise, conforme a recolha de dados disponíveis do edifício em estudo, sabemos diariamente a ocupação do edifício e reconhecemos o padrão do funcionamento, refletido pelos consumos dos recursos energéticos num passado equivalente em termos de ocupação. Assim, começa-se por definir as metas do custo diário de um cliente em variáveis como o kWh/dia (energia elétrica), o m³ /dia (água potável) e o kg/dia (gás propano) e posteriormente definir quais os valores que queremos atingir, nas variáveis kWh/dia/cliente, o m³ /dia/cliente e o kg/dia/cliente, por forma a melhor o desempenho energético.

Na Figura 4.1 apresenta-se os valores globais dos consumos do hotel, desde 2023, onde se verifica o período de operação de meados de abril a novembro, também, quais os valores durante o período de fecho e abertura da unidade. De acordo com o gestor do departamento de manutenção do hotel, ao longo do período em análise foram implementadas diversas medidas de melhoria, entre as quais se destacam: em janeiro/fevereiro de 2023, a substituição dos redutores de caudal nos quartos e a instalação de um sistema solar fotovoltaico com 216 módulos e cerca de 60 kWh de produção por dia. Em janeiro/fevereiro de 2024, a substituição do grupo de chillers, a renovação de 10% da iluminação nas zonas comuns, a instalação de novos redutores de caudal nas áreas comuns e a substituição de um túnel de lavagem na cozinha principal. No período de fecho entre 2024 e 2025, a substituição dos restantes 90% da iluminação nas zonas comuns e de 83% da iluminação nos quartos, bem como a renovação da tubagem de climatização em dois pisos de quartos. Todas estas intervenções encontram-se assinaladas na Figura 4.1, representadas por retângulos com tracejado verde.

Antes de se analisar as estatísticas globais de consumo do edifício, é fundamental compreender os valores parciais de consumo, identificando quais são os principais consumidores de energia e água. Esta análise permite avaliar se esses consumidores apresentam níveis de eficiência adequados, determinar quais necessitam de maior intervenção para otimização e compreender de que forma tais desempenhos se refletem em aumentos ou reduções nos consumos energéticos totais.

Para complementar o raciocínio, quando estamos no nível industrial 4.0, temos que ser muito *inteligentes*, isto é, ter acesso a toda a informação relativa a consumos e equipamentos, significa um grande volume de dados (*Big Data)*, por forma a esses dados revelem padrões de comportamento e que estes provem ser indicadores de desempenho. Por sua vez, indicadores estes que nos influenciem a tomar decisões. Demonstrando a necessidade de utilização de

ferramentas digitais, permitindo que estas ajudem na manutenção eficiente de um determinado ativo.

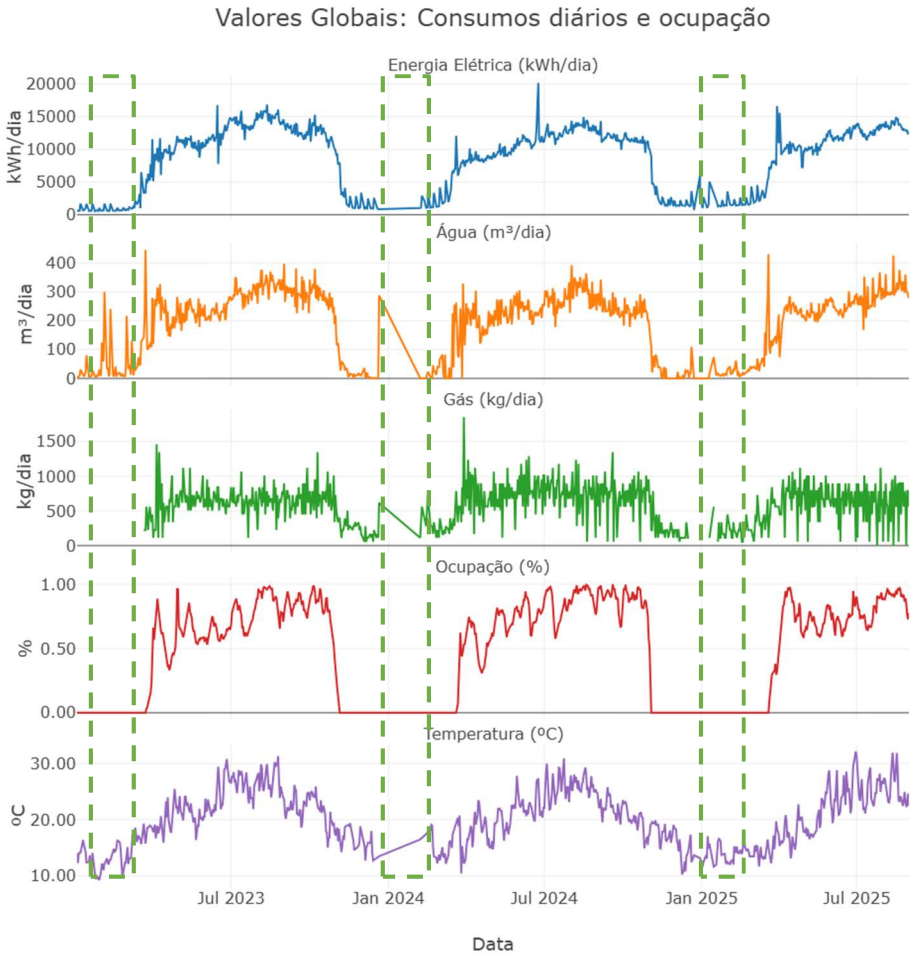


Figura 4.1- Valores globais das variáveis de consumos diários de energia elétrica, água potável, gás propano, assim com a taxa de ocupação diária e a temperatura ambiente média diária

Existem outras variáveis, que devem ser contempladas para aprofundar o estudo das medições realizadas, podemos de forma estatística analisar variáveis como a mediana, a média e quartis ao longo do nosso histórico de dados.

Conforme a Figura 4.2, é possível comparar e verificar se realmente existiram aumentos ou diminuições em termos de valores globais nos consumos de energia elétrica, água potável e gás propano longo dos anos de histórico de dados. No nosso caso de estudo, o hotel apresentou diminuições na média e na mediana dos consumos de energia elétrica e água, comparativamente ao ano de 2023, mesmo com um aumento na ocupação. Esta redução de consumos, estará certamente ligada às intervenções realizadas no ano de 2023, tais como, a

modificação dos redutores de caudal, melhorias aplicadas na iluminação, e a compra de novos equipamentos de AVAC.

Com a apresentação dos digramas de caixa, Figura 4.2, facilmente se compreende como foi a distribuição dos dados durante o período de 2023 e 2024. Neste âmbito estatístico, 2025 apresenta apenas contagens até ao mês de agosto, o que significa que de forma anual ainda não pode ser comparado com o restante histórico. Contudo se analisarmos os dados no formato de *séries temporais*, podemos observar tendências, ciclos, sazonalidade e anomalias comparando com um determinado histórico entre os anos 2023 e 2024.

Em termos de resultados, a diminuição do consumo de energia elétrica foi de 8,2% enquanto a ocupação aumentou 4%, referente aos valores da mediana. Na água, fica evidente a maior redução, representado uma diminuição de 12,9%, face à mediana de 2023. De forma divergente o gás registou um aumento de 16,7%.

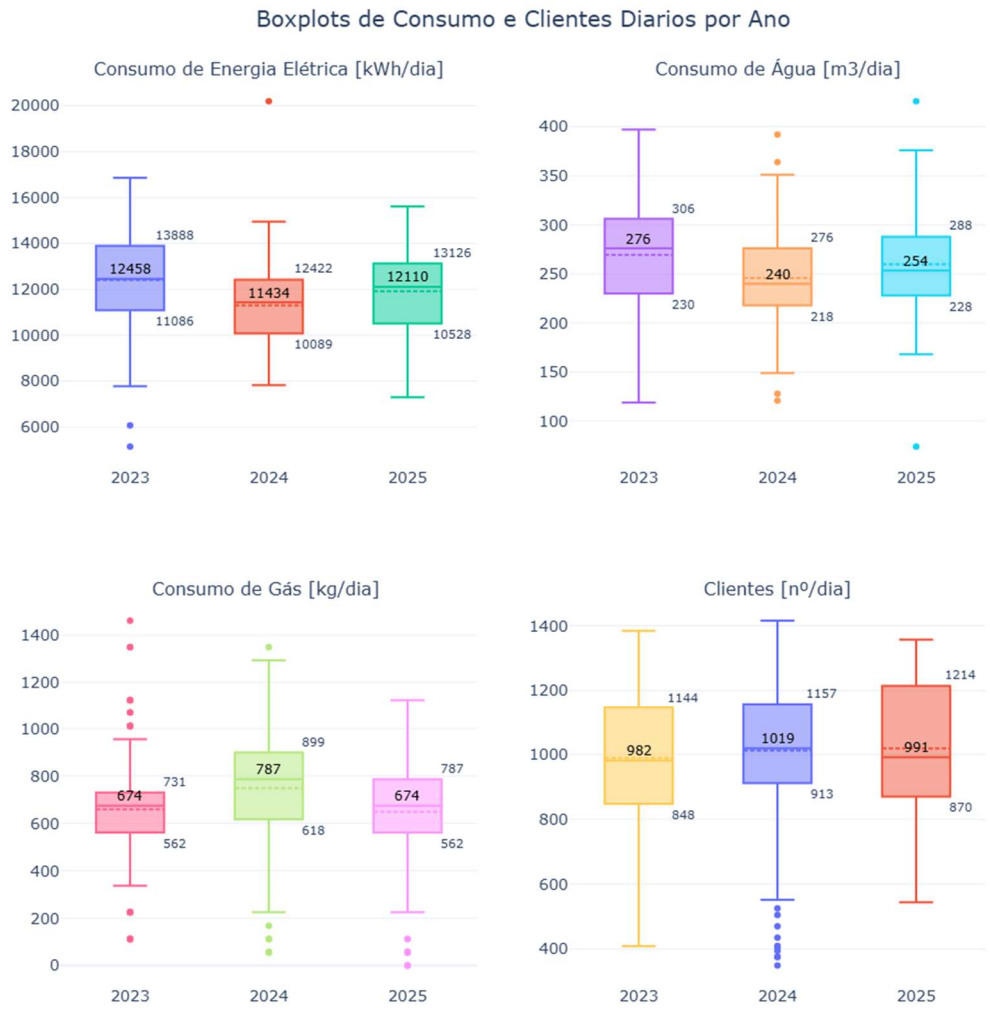


Figura 4.2 - Diagramas de Caixa sobre os consumos e clientes diário ao longo do 3 anos de histórico de dados

Ao considerar a visualização do grupo de dados por mês, dias da semana, semanas do ano, ou mesmo horas entre outros, deteta-se crescimentos ou declínios em valores de consumo, produção e ocupação ao longo do tempo. A análise de series temporais facilita o estudo da sazonalidade e padrões cíclicos ao reconhecer variações ao longo do tempo que se repetem quando garantidos certos valores em outras variáveis. Ao localizar e sermos alertados para valores fora do padrão de funcionamento podemos antecipar intervenções de manutenção, ou corrigir anomalias. A figura 4.3, reflete a diminuição dos consumos de energia e água, ao comparar 2024 com 2023 durante os diferentes meses de operação.

Figura 4.3 - Diagramas de caixa de consumos e ocupação diários por mês e ano, durante o período de ocupação (abril a outubro).

Definitivamente a sazonalidade da estação de verão fica refletida, a ocupação aumenta e por sua vez os consumos. Junho, julho e agosto mostraram um aumento de significativo na

ocupação e ainda assim garantiram reduções de consumos, em média de 9% para a energia elétrica e 4% nos consumos de água potável. Os melhores resultados mensais observados foram o de maio e julho que apresentaram a maior eficiência ao considerar a diferença entre a taxa de consumo por ocupante relativo a 2024 e 2023.

Pode-se comprovar na Tabela 4.1 e na Figura 4.3, que o consumo de água variou de forma positiva apenas nos meses de maio e junho combinando com os maiores aumentos percentuais da ocupação, relativamente às medianas de 2024 quando comparadas com 2023. Apesar da ocupação ter aumentado os consumos de energia elétrica, nos meses de maio junho e julho, apresentam uma redução considerável. será importante perceber como se distribuíram essas variações de forma desagregada por setores. Quando desagregados todos os consumidores parciais referentes ao valor global de um determinado recurso, podemos identificar as variáveis que se encontram desfavorecidas, sendo assim possível também quantificar a pegada de carbono equivalente, ou a hídrica.

Tabela 4.1 - Variação percentual das medianas dos valores diarios registados referente às varivaeis consumo de energia eletrica, água potavel, gas e o nº de clientes.

	ELTRICIDADE	**ÁGUA**	**GAS**	**CLIENTES**
mês	variação da mediana (%)			
1	-	-	-	-
2	93.77	41.67	-	-
3	193.09	-18.00	-18.75	-
4	-11.33	-19.78	20.00	-15.14
5	-13.54	3.10	8.33	12.50
6	-6.21	3.27	21.74	14.97
7	-10.10	-6.51	27.27	6.93
8	-9.64	-5.23	8.33	1.53
9	-6.73	-21.11	12.50	-1.46
10	-9.28	-19.50	7.69	-2.95
11	-19.93	50.00	-20.00	-
12	37.80	200.00	0.00	-

A decomposição das séries permite isolar componentes sazonais, cíclicos e de tendência, revelando padrões que, à primeira vista, poderiam passar despercebidos. Essas metodologias,

não apenas evidenciam a influência da sazonalidade, mas também possibilitam quantificar as reduções de consumo, mesmo em períodos de maior ocupação. Com dados organizados e modelados de forma adequada, é possível aplicar métricas objetivas que comprovaram reduções médias de consumo, podendo se destacar meses de maior eficiência. É importante compreender que, estas metodologias não se limitam a constatar variações, mas também a perceber o contexto que ocorrem e o que as influenciou.

4.3 PEGADA DE CARBONO

O setor da hotelaria e o consumo de energia e água que lhe está associado têm uma relação direta com a pegada de carbono. Os requisitos energéticos dos hotéis, quando não otimizados, contribuem para uma alta pegada de carbono.

Segundo o "*Guia neutralidade carbónica nos empreendimentos turísticos"* (Turismo de Portugal, 2021), para alcançar a neutralidade carbónica, os empreendimentos turísticos (ET) devem seguir um ciclo de planeamento e ação que envolve a quantificação, a redução e a compensação/neutralização das suas emissões de *gases com efeito de estuf*a (GEE), num horizonte de longo prazo, idealmente até 2050[37].

As emissões de carbono por quarto ocupado foram reportadas entre 11,9 e 26,4 kg CO_2e em Portugal (fonte: https://www.hotelfootprints.org), portanto, e conforme as referências, para conservar as ações de quantificação, redução e compensação/neutralização das GEE, seguem as seguintes orientações:

i. Quantificar as emissões de carbono: A quantificação das emissões de gases de efeito estufa representa o primeiro passo fundamental para a implementação de estratégias de mitigação ambiental em empreendimentos turísticos. Como tal, é essencial a realização de um inventário de emissões (*baseline*) que reflita a realidade operacional do empreendimento, adotando-se normas internacionais reconhecidas, como o Protocolo GHG (Greenhouse Gas Protocol)[38] ou a ISO 14064-1:2018.

ii. Reduzir as emissões de carbono: A mitigação das emissões deve concentrar-se na eficiência energética, em que a primeira linha de atuação inclui a substituição de equipamentos por tecnologias de baixo consumo, como lâmpadas LED e caldeiras eficientes, a implementação de práticas de gestão energética, como sensores de presença, reguladores de intensidade luminosa, adoção de medidas passivas, como isolamento térmico e sombreamento, que reduzem a necessidade de climatização.

A eletrificação dos sistemas é estratégica, dado o processo de descarbonização da rede elétrica nacional. A incorporação de fontes renováveis complementa as estratégias anteriores, como por exemplo produção local de energia por meio de painéis solares fotovoltaicos e térmicos.

iii. Compensar ou neutralizar as emissões residuais: A neutralização consiste no investimento em projetos que removem carbono da atmosfera, seja por vias biológicas (reflorestação, florestação, gestão florestal sustentável) ou tecnológicas (captura direta de CO_2 e armazenamento geológico). A compensação refere-se ao apoio a iniciativas externas à cadeia de valor do empreendimento turístico ET, que contribuem para a evitação ou redução de emissões em outras localizações, como projetos de energia renovável.

iv. Transparência e comunicação: Divulgar anualmente as emissões de GEE, o horizonte temporal para atingir o objetivo, os objetivos intermédios de redução e o investimento em projetos de compensação/neutralização. A comunicação deve ser rigorosa e específica, detalhando o âmbito das ações e o volume de emissões envolvidas. Importa sublinhar que as emissões compensadas ou neutralizadas não devem ser subtraídas do inventário anual de GEE, garantindo a integridade metodológica e a credibilidade do reporte.

No âmbito do presente estudo foram calculadas as GEE de cada recurso, em kg de CO_2 equivalente, onde os fatores de emissão aplicados foram de 0,167 kg CO_2e/kWh referente à energia elétrica, assim como 0,29 kg CO_2e/m^3 e 3,26 kg CO_2e/kg respetivamente para a água e o gás propano.

Na Figura 4.4 apresenta-se os consumos totais de cada mês por recurso, ao longo do período de dados recolhidos, onde fica evidenciado as toneladas de CO_2 equivalente. Destaca-se que o gás apresenta o maior impacto, representando uma média ponderada de 70 tCO_2e por mês, durante o período de operação. As GEE associadas à energia elétrica, ligeiramente abaixo com cerca de 60 tCO_2e por mês, enquanto a pegada da água potável representa cerca de 2 tCO_2e por mês.

Conforme a Figura 4.5, verifica-se que os valores estáticos a presenta a mesma redução percentual verificada anteriormente, no subcapitulo 4.1.2 Valores globais, em média de 9% para a energia elétrica e 4% nos consumos de água potável comparativamente aos anos 2024-2023. Aqui verifica-se que o valor mediano para a pegada de carbono, no referenciado ano de 2024, correspondente à energia elétrica é 1909 kg CO_2e/dia, relativa ao consumo diário de

11434 kWh/dia. Na água potável verifica-se para 240 m^3/dia uma pegada de carbono equivalente de 70 kg CO_2e/dia. No caso do gás e de forma contraria, a pegada de carbono foi superior em 2024 cerca de 360 kg CO_2e/dia.

Enquadrando o caso de estudo com os valores referenciados para o benchmarking ambiental (11,9-26,4 kg CO_2e/quarto/dia), foi calculado as emissões de carbono por quarto ocupado e é interessante refletir sobre os dados estatísticos apresentados na Figura 4.6. Dado que o intervalo interquartil da pegada de carbono por quarto ocupado no caso de estudo se situa entre 10 e 13 kg CO_2e/quarto/dia, este resultado posiciona o hotel dentro da faixa inferior das emissões reportadas para Portugal.

Esse resultado deve ser interpretado como indicador positivo de desempenho ambiental, sugerindo que o hotel em análise apresenta uma pegada de carbono por quarto ocupado inferior à mediana nacional, o que pode refletir boas práticas de eficiência energética, gestão hídrica e operacional. No entanto, os valores inferiores ao intervalo nacional também exigem análise crítica, considerando fatores como sazonalidade, tipologia de serviços oferecidos, grau de ocupação e abrangência das fontes de emissão contabilizadas, por exemplo, se foram contabilizadas todas as "outras" emissões diretas assim como indiretas[35]. Portanto, deve ser recomendado que este resultado seja refletido como evidência de desempenho ambiental competitivo, mas acompanhado de uma discussão metodológica transparente, que esclareça os limites da comparação e assegure a credibilidade do benchmarking.

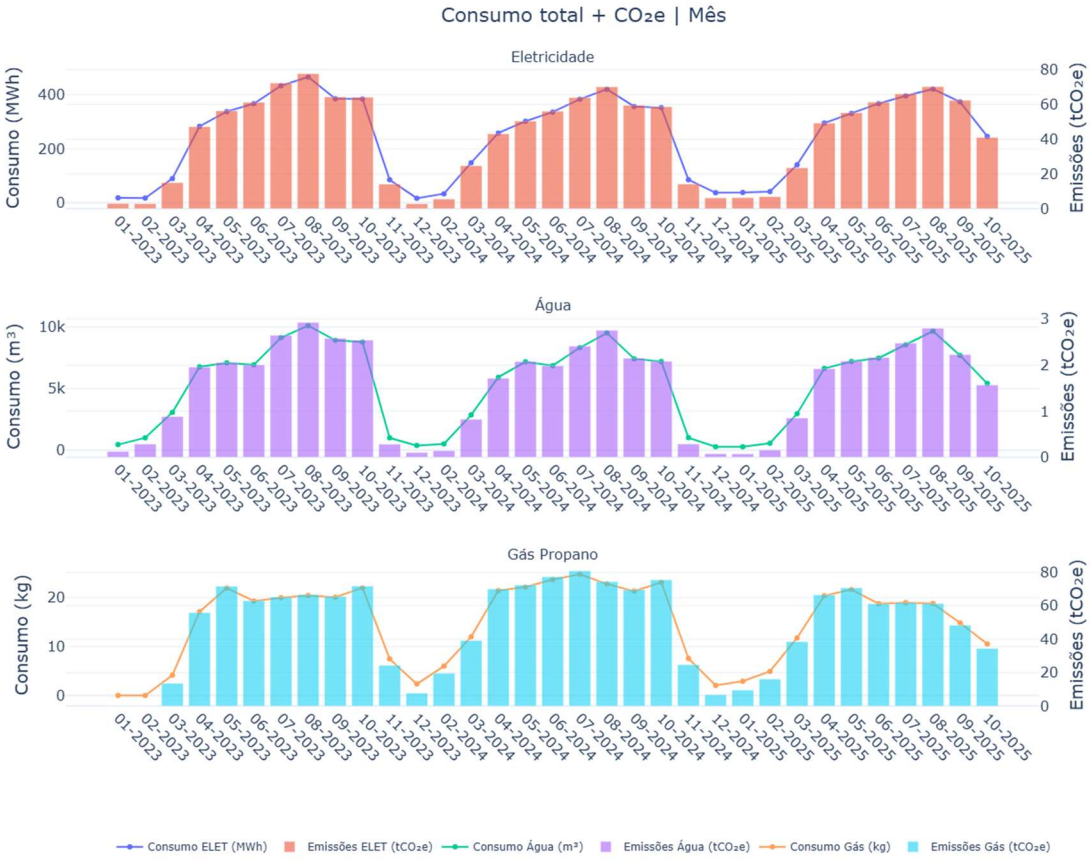


Figura 4.4 - Consumos e emissões mensais por recurso.

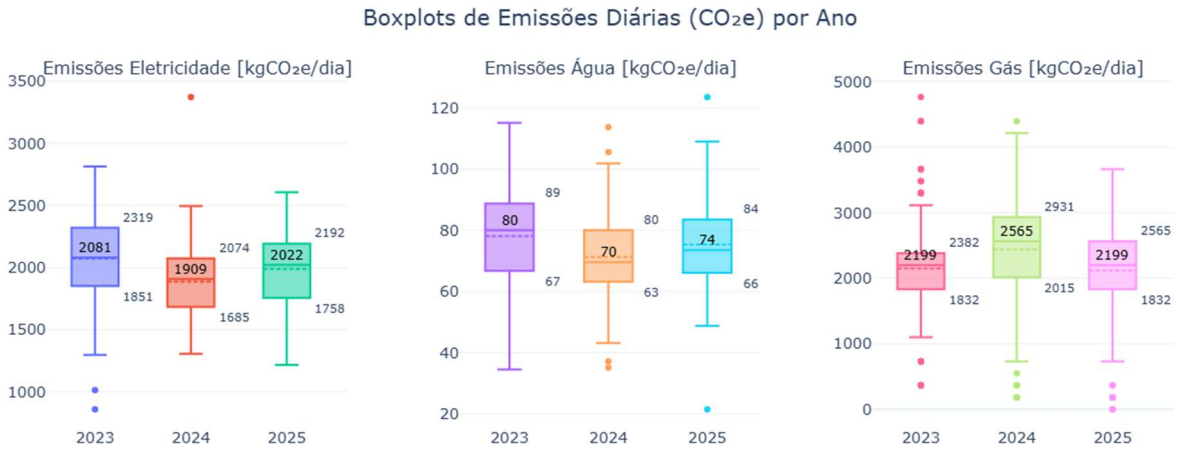


Figura 4.5 - Diagramas de caixa da distribuição estatística da pegada de carbono para os três recursos em analise, energia elétrica, água potável e gás propano

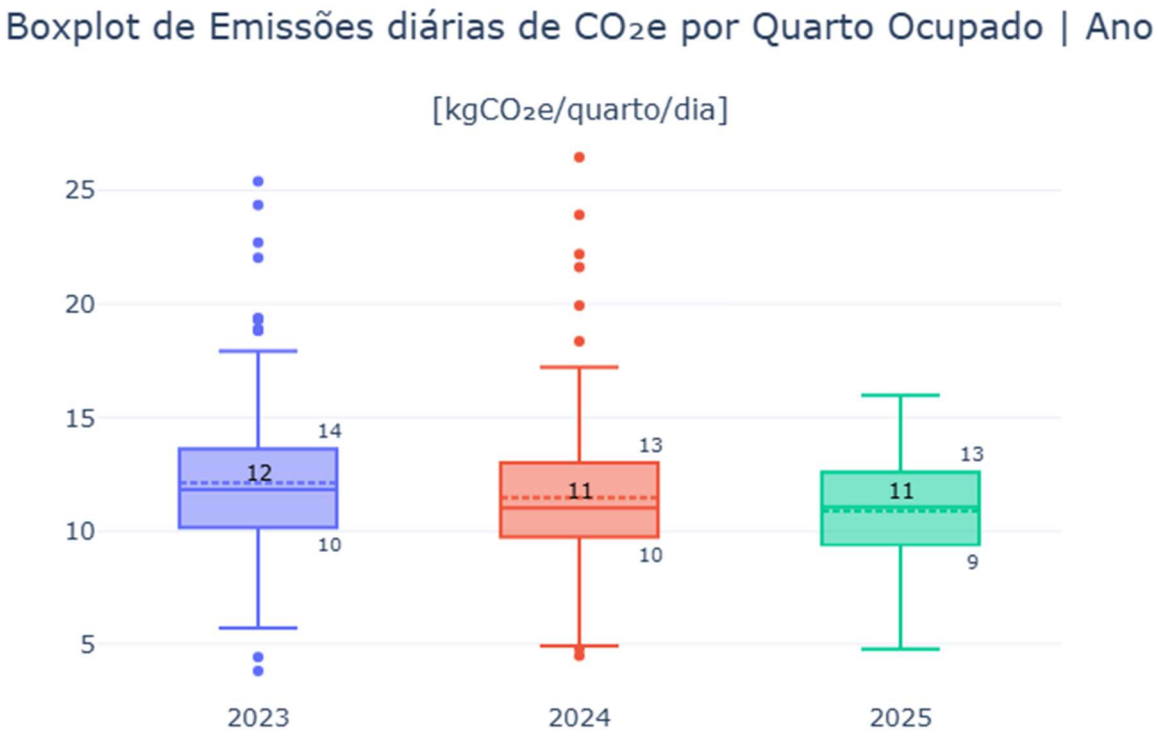


Figura 4.6 - Valores estatísticos do indicador emissões de carbono por quarto [kgCO₂e/quarto/dia]

4.4 VARIÁVEIS INDEPENDENTES

4.4.1 OCUPAÇÃO

Entre o conjunto de variáveis a considerar para uma boa monitorização de um edifício, as variáveis que não dependem diretamente de outras, mas que podem influenciar muitas delas, são as usadas como ponto de partida para explicar ou prever comportamentos referentes aos consumos de energia, água ou gás.

A análise da matriz de correlação apresentada na Figura 4.7 evidencia relações substanciais entre as variáveis consideradas, destacando-se as variáveis da ocupação (OCP – taxa de ocupação; Q_OCP – nº de quartos ocupados; CLTs – nº de clientes), que apresentam uma correlação forte de 0,93 com o consumo de energia elétrica (ELET_C), 0,90 com o consumo de água (AGUA_C) e uma correlação moderada de 0,60 com o consumo de gás (GAS_C).

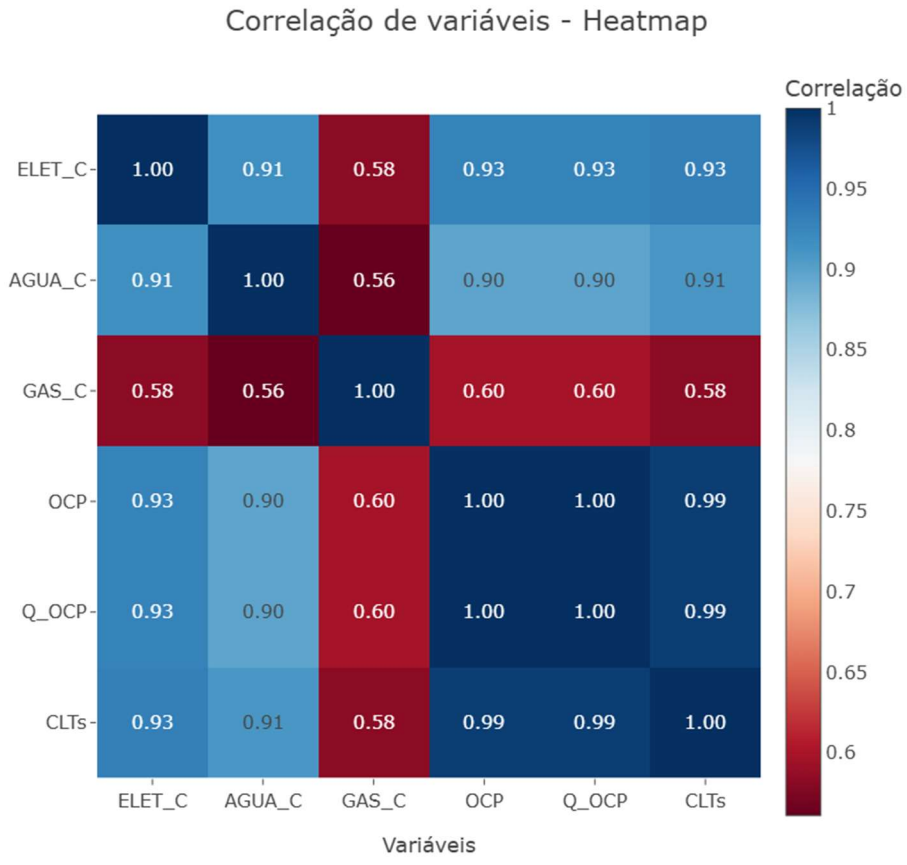


Figura 4.7 - Heatmap das correlações entre as variáveis de ocupação e as de consumos.

Entre o consumo de energia elétrica (ELET_C) e o consumo de água (AGUA_C), apresenta um coeficiente de 0,91. Essa correlação sugere uma possível interdependência entre esses dois recursos, possivelmente refletindo padrões operacionais comuns ou requisições simultâneas em sistemas integrados, onde o uso de água e eletricidade pode estar sincronizado por equipamentos ou processos compartilhados. Além disso, observa-se uma correlação moderada entre o consumo de gás (GAS_C) e o consumo de água (AGUA_C), com valor de 0,60, indicando uma relação menos intensa, mas ainda relevante. Essa relação pode indicar que determinados processos dependem tanto do consumo de gás, como de água, por exemplo, a produção de AQS.

Aprofundar o estudo destas correlações não só permite melhor compreensão dos fluxos de consumo em sistemas complexos, mas também fundamenta a elaboração de estratégias sustentáveis que considerem os vínculos entre energia, água e ocupação.

Na gestão de uma unidade hoteleira, a ocupação assume o papel de variável independente e constitui um verdadeiro driver operacional, uma vez que condiciona diretamente o uso dos recursos disponíveis em função do número e do perfil dos hóspedes. Do ponto de vista operacional, a ocupação define a intensidade de utilização dos serviços e equipamentos, influenciando a eficiência diária e a necessidade de ajustes nos processos internos. Sob a perspetiva financeira, o seu impacto é imediato, refletindo-se na receita por quarto ocupado e na rentabilidade global da operação. No domínio ambiental, a ocupação exerce influência direta sobre os consumos de energia e água, bem como sobre a pegada ecológica associada à atividade hoteleira. Paralelamente, a experiência do cliente é também moldada por esta variável, uma vez que flutuações na taxa de ocupação podem exigir alterações nos padrões de serviço, na gestão de espaços comuns e na disponibilidade de equipamentos, afetando a perceção de qualidade e satisfação do hóspede. Finalmente, numa dimensão estratégica, a ocupação fornece informação essencial para o planeamento de operações e para a definição de investimentos futuros, funcionando como um indicador-chave para alinhar a gestão de manutenção com os objetivos de eficiência, sustentabilidade e competitividade do setor hoteleiro.

4.4.2 SAZONALIDADE

A recolha dos dados disponíveis para analise foram os meses de maior atividade turística, entre abril e outubro, os aumentos da taxa de ocupação com valores superiores a 90% registaram-se em agosto. Considera-se que essa variação relaciona fatores como clima, ocupação e calendário turístico, que por sua vez influenciam o consumo de recursos, ao longo do ano.

Conforme se evidencia nas figuras apresentados no subcapítulo anterior, verificou-se um aumento de consumos, durante a estação do verão. Recorde-se que é no mês de agosto onde o consumo de energia elétrica apresenta maiores valores de mediana, valores entre os 13,5-15 MWh/dia.

Referente ao consumo de água potável, também no mês de agosto apresenta-se valores de mediana entre 300-325 m3/dia, onde ficam assinalados os picos máximos. Sobre o recurso gás propano, devido a este estar fortemente ligado à produção de água quente sanitária e aquecimento, (apenas com base no ano 2025) aos consumos de gás nas cozinhas e na lavandaria esta é a variável que apresenta menor variação ao longo do tempo de operação da unidade.

Essas variações refletem não apenas o número de hóspedes, mas também o tipo de operação do edifício, como maior uso de climatização, iluminação, lavandaria e serviços complementares.

A análise de séries temporais permite analisar essas variações sazonais ao longo do tempo. No tratamento dos dados, foram criadas variáveis como mês, trimestre, dia do ano e semana do ano para modelar essas flutuações. Isso é essencial para previsão de consumo futuro com base em padrões históricos, identificação de picos e quedas sazonais, facilitando o planeamento operacional dos serviços técnicos nos diferentes setores. A correlação com ocupação foi integrada como variável explicativa, revelando forte relação com os consumos, especialmente em períodos turísticos, como é o caso da estação de verão em unidades hoteleiras no Algarve. Incorporar análises de séries temporais é fundamental para entender o comportamento dos consumos e otimizar a gestão de recursos em unidades hoteleiras. Isso permite decisões mais informadas, otimização de custos e maior eficiência operacional.

4.5 DESAGREGAÇÃO DOS PRINCIPAIS CONSUMIDORES

A hotelaria é intensiva no uso de água e energia, face às necessidades da hospitalidade básica a prestar ao hospede, inclusive se os serviços complementares forem abundantes. Ao considerar custos de energia e água numa unidade, verifica-se quais os serviços oferecidos ao cliente e deve-se quantifica-los por forma a compreender o custo de operação e a pegada ambiental. Admite-se que a adoção de práticas sustentáveis pode fortalecer a imagem do hotel junto do hospede e contribuir para à resiliência das tendências modernas e ecológicas do setor turismo.

No âmbito da gestão da manutenção de edifícios, as intervenções realizadas em setores específicos são fundamentais para maximizar a eficiência e conforme este caso, em estudo, garantir a sustentabilidade de unidades hoteleiras.

Adiantando que neste caso estudo não foi possível adquirir dados em todos os setores, reflete-se que para acompanhar as tendências sazonais, os padrões de funcionamento, as ineficiências e as possíveis otimizações nos diferentes setores, a desagregação de consumos energéticos e hídricos deve apresentar os valores parciais nas seguintes áreas: *quartos, zonas comuns publicas, zonas de serviços a colaboradores, cozinhas, restaurantes e bares, sistemas de AVAC e AQS, lavandaria, elevadores* e *piscinas*, entre outros que também devem ser monitorizados por forma a se desenvolver os objetivos da sustentabilidade energética e da boa manutenção de ativos.

Quando se quer aplicar as funcionalidades das tecnologias digitais, alinhados aos princípios da i*ndústria 4.0*, requer que o parque de máquinas esteja equipado com sensores e contadores interligados a um sistema de GTC, pois este é o principal consumidor onde é possível desenvolver processos e aplicar melhorias que contribuam para o impacto sustentável na operação e no ambiente.

4.5.1 ÁGUA POTÁVEL

No contexto da gestão hídrica, garantir a monitorização dos consumos nos diferentes setores de uma unidade hoteleira, não importa só quanto se quer acrescentar valor aos demais centros de custos, mas também quando se pretende identificar perdas, consumos excessivos ou ineficiências e principalmente quando se pretende atingir metas de sustentabilidade hídrica.

No desenvolvimento da dissertação, após o tratamento dos dados disponíveis, verificou-se que os consumidores de água potável monitorizados são os contadores da distribuição de água referentes a: central de produção e distribuição de AQS, lavandaria e piscinas. Na Figura 4.8 verifica-se que os contadores que apresentam maior expressão em termos de proporção, são os referentes ao consumo AQS (apenas nos dados de 2025), ao consumo de água fria e ao de água quente na lavandaria, assim como o consumo de água na piscina grande. O consumo de AQS apresenta fazer parte de 24% do consumo total da unidade, na lavandaria os consumos de água fria e quente, respetivamente, representam 19% (2025), 14% (2024) e o de água quente 9% (2025), 7% (2024).

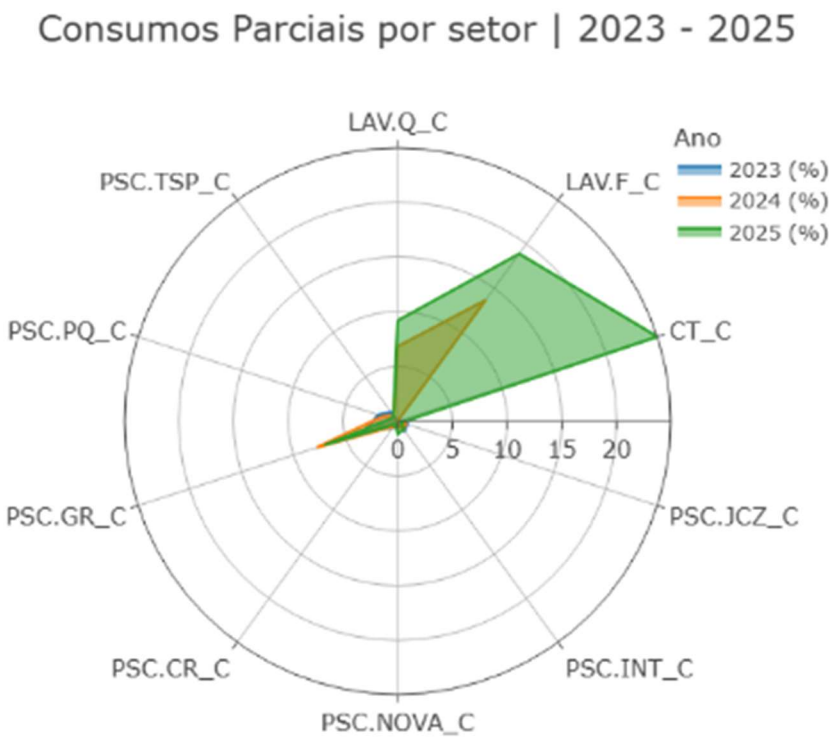


Figura 4.8 - Consumos parciais por setor

A piscina principal da unidade, com cerca de 1000 m³, mostrou grandes penalização ao longo do histórico, onde representou cerca de 3% do consumo total no de 2023, 8% em 2024 e 7% em 2025 (dados referentes a 01 de janeiro a 31 de agosto).

Ao analisar os dados estatísticos dos maiores consumidores referidos, pode-se perceber quais as diferenças nas tendências dos consumos ao longo dos meses de operação nos anos de 2023-2025. No seguimento do subcapítulo *4.1.1 Valores Globais* concluiu-se que o no ano de 2024 existiram consideráveis reduções no consumo, inclusive o consumo geral de água tinha sida a variável com maior expressão nessa redução (cerca de 13%). Contudo, e importante refletir, o quão importante desgregar os consumos nos diferentes setores, pois não se deve ficar satisfeito com uma análise global. É na análise exploratória dos consumidores parciais que é possível identificar falhas ou ineficiências de operação. Através deste caso em estudo é possível provar que mesmo em períodos de melhorias globais, determinados setores ou mesmo equipamentos podem estar a apresentar falhas na sua operação. Por meio de uma análise de diagramas de caixa, conforme Figura 4.9 é bastante percetível como a distribuição dos consumos de água na piscina principal (PSC.GR_C) ao longo dos meses de operação variou de forma atípica.

Figura 4.9 - Diagramas de caixa dos contadores parciais do recurso água potável

De realçar o mês de agosto onde sofreu-se um aumento de 485% referente aos valores medianos de 2024 comparativamente a 2023. O know-how técnico partilhado pelos responsáveis de manutenção da referida unidade, identificou as operações de manutenção à referida piscina foram desapropriadas, relativamente à falta de rigor nas limpezas dos sistemas de filtro assim como as respetivas renovações de água. As identificações destas operações ineficientes só são possíveis de justificar quando um registo de manutenções preventivas existe, assim como o demais registo de medições executadas neste determinado ativo. A Figura 4.10. ajuda a quantificar as diferenças do consumo total por mês nos diferentes anos de histórico de dados, com especial evidencia entre os meses de maio a agosto.

Neste caso da piscina, fica claro que o ano de referência foi o de 2023 e todos os ajustes de manutenção no presente ano 2025 devem visar atingir esses valores referenciados.

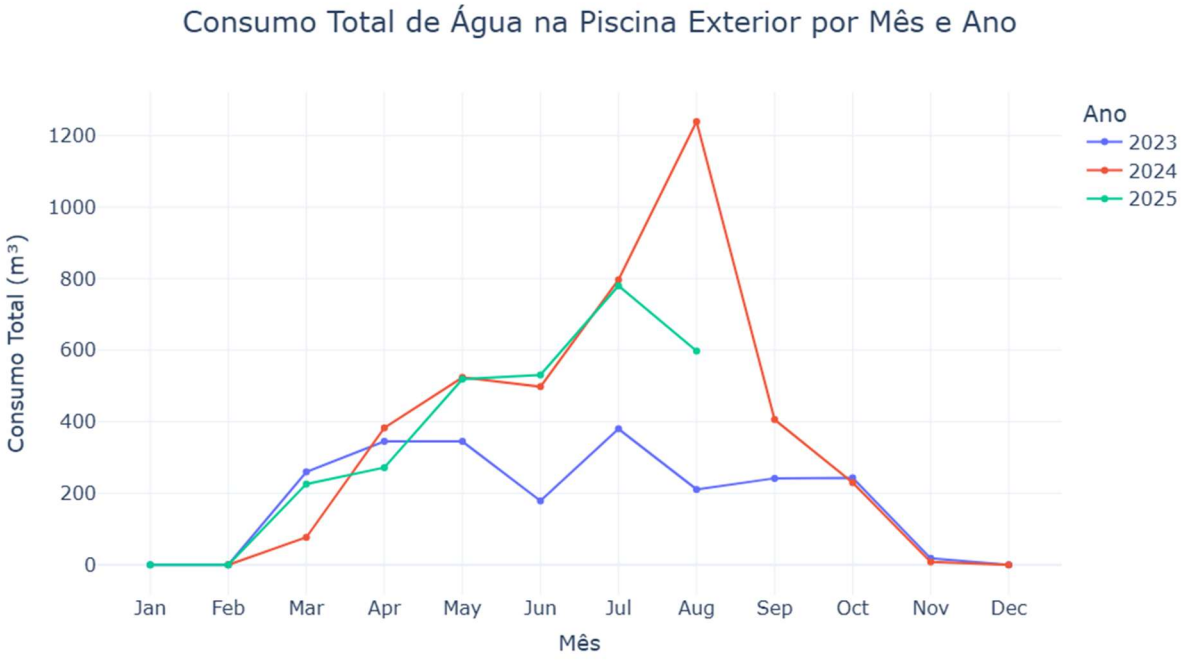


Figura 4.10 – Consumo de água por mês da piscina exterior, ao longo do período em analise

A análise dos consumos de água de rega, assim como os dados exportados da estação meteorológica do hotel, permitiu relacionar o consumo de água da piscina com variáveis ambientais, nomeadamente temperatura exterior, humidade relativa. Os resultados (Figura 4.11) demonstraram que o consumo de água da piscina (1000 m³) apresenta uma correlação moderada (0,4) com a temperatura exterior, uma correlação fraca e negativa (-0,22) com a humidade relativa e igualmente fraca (0,28) com o consumo de água da rega.

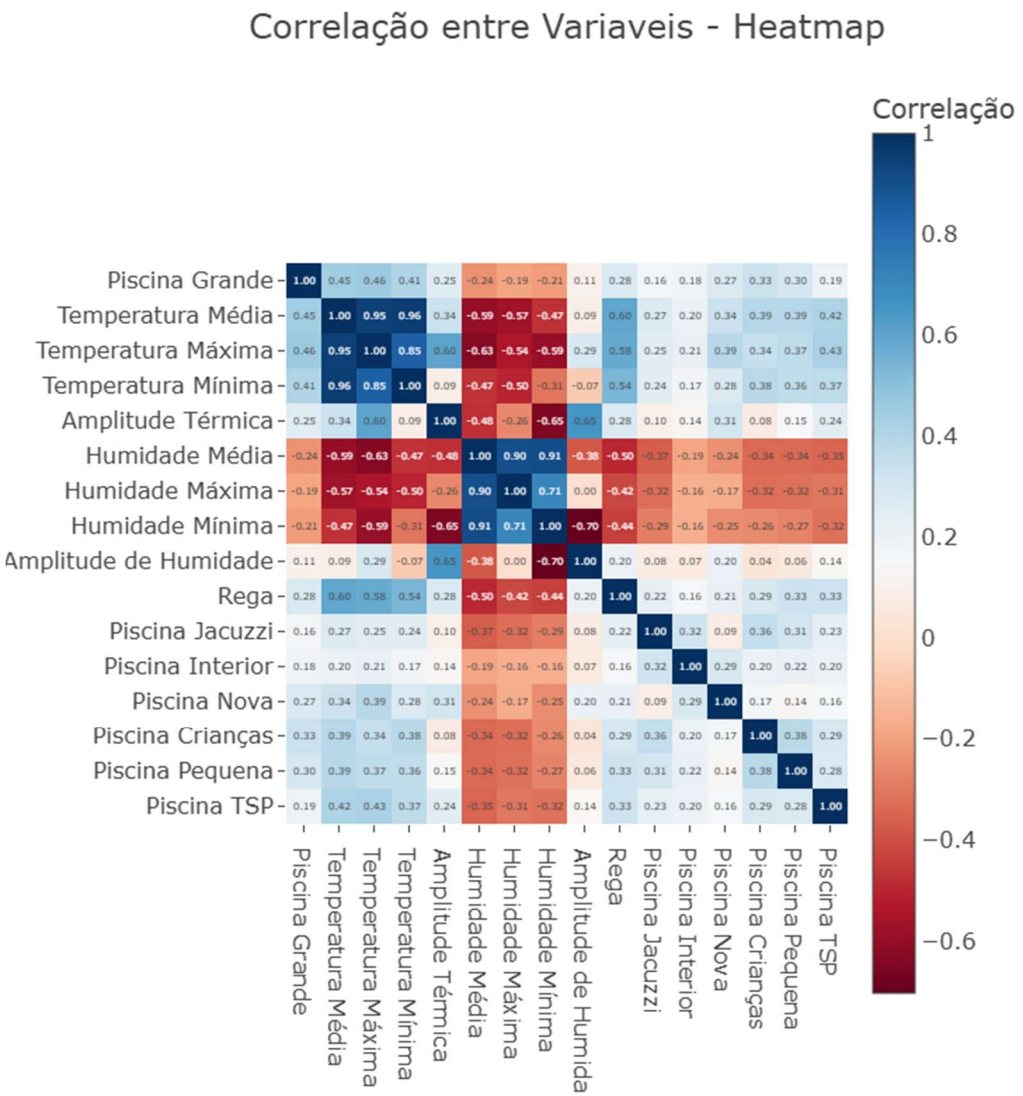


Figura 4.11 - Heatmap da correlação entre variáveis ambientais e consumo de água da rega com o consumo de água da piscina

Considerando que o consumo de água da piscina corresponde essencialmente à reposição de perdas, e excluindo a variável associada ao fluxo de ocupantes, conclui-se que as condições climáticas do ambiente exterior exercem influência direta sobre essas perdas. Em particular, períodos mais quentes e secos, caracterizados por temperaturas elevadas e valores reduzidos de humidade relativa, tendem a intensificar a taxa de evaporação[39]. A este efeito acrescem ainda fatores como o vento e a radiação solar, que, embora não tenham sido objeto de monitorização neste estudo, são reconhecidos na literatura como variáveis determinantes no processo de evaporação.

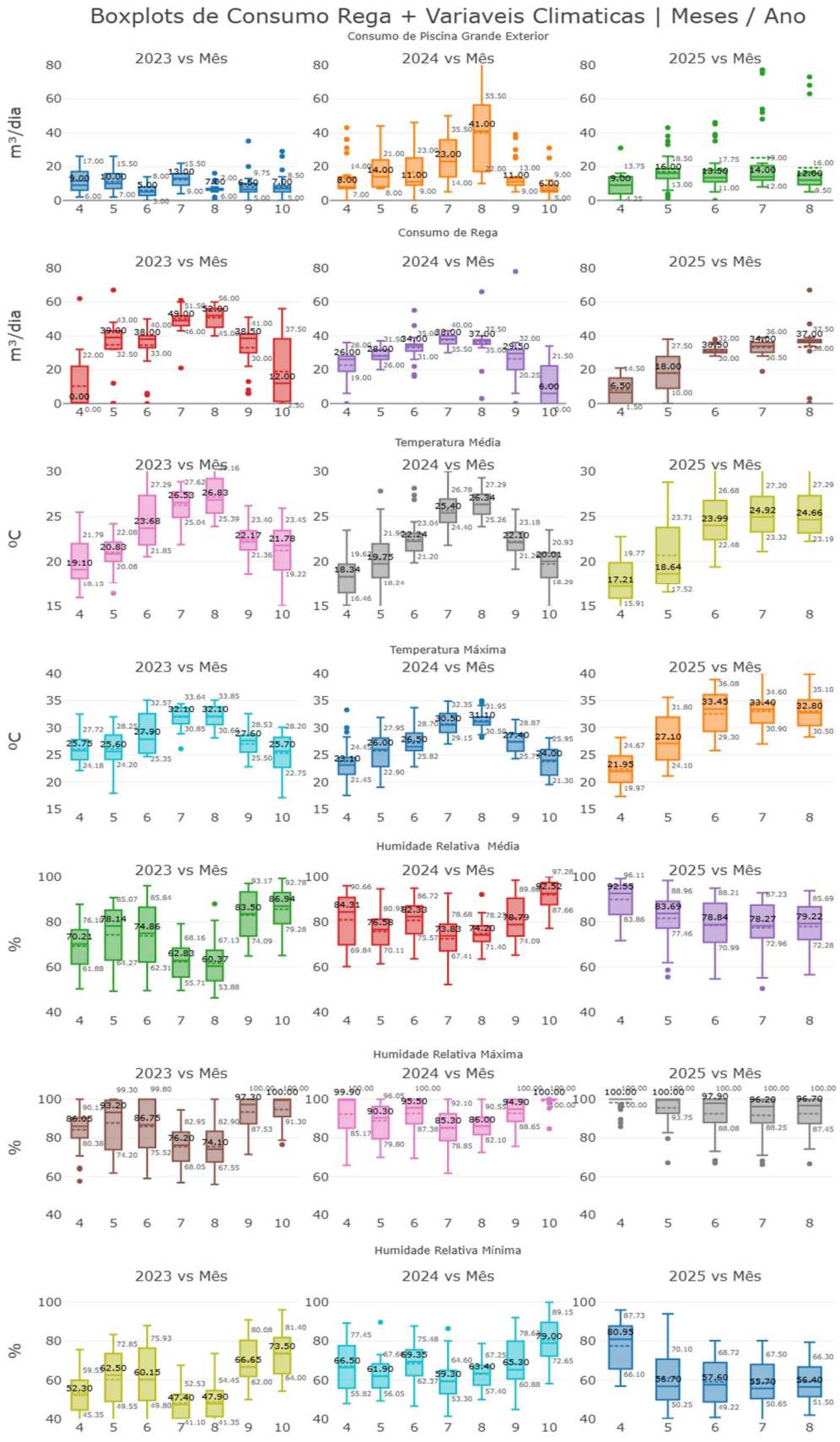


Figura 4.12 - Diagrama de caixa com distribuição estatística dos valores por mês e por ano, referente às varáveis climáticas e ao consumo de água para rega.

Com o intuito de aprofundar a análise, foram elaborados diagramas de caixa que permitiram observar a distribuição estatística dos valores por mês e por ano, conforme Figura 4.12. A comparação entre os anos de 2023 e 2024, durante os meses de maior consumo de água (julho e agosto), evidenciou que, em 2024, a temperatura exterior apresentou valores medianos mais baixos e intervalos interquartis mais reduzidos, enquanto a humidade relativa se mostrou mais elevada. Assim, o pressuposto de que um período mais seco e quente estaria associado a maiores perdas por evaporação não se confirmou neste caso. Adicionalmente, ao integrar a variável consumo de água da rega na análise, verificou-se que o ano de 2024 registou valores mais baixos em comparação com 2023. Este resultado reforça a conclusão de que, apesar da expectativa de maiores perdas em condições climáticas mais secas, os dados observados não corroboram essa relação direta no período em estudo.

O caso em estudo reforça a necessidade de implementar sistemas de monitorização contínua e de registo rigoroso das operações de manutenção preventiva, permitindo correlacionar consumos anómalos com falhas operacionais. A piscina principal, pela sua dimensão e impacto no consumo total, constitui um exemplo paradigmático de como a ausência de procedimentos técnicos adequados, como a limpeza regular dos filtros e a gestão das renovações de água, pode comprometer a eficiência hídrica da unidade. Conclui-se que a gestão sustentável da água em hotelaria não se define apenas em metas globais de eficiência, mas também a análise segmentada por equipamentos e setores críticos. A adoção de indicadores normalizados, associada a planos de manutenção preventiva bem estruturados, é essencial para garantir a resiliência operacional e a sustentabilidade ambiental da unidade[32].

4.5.2 ENERGIA ELÉTRICA

No âmbito da gestão de energia elétrica numa unidade hoteleira, a monitorização detalhada dos consumos por setor também é uma ferramenta essencial para identificar desperdícios, otimizar recursos e apoiar decisões estratégicas de eficiência energética. A análise global do consumo fornece uma visão geral, mas é na desagregação por pontos de medição que se revelam padrões de utilização, ineficiências e oportunidades de melhoria.

No desenvolvimento deste caso de estudo, após o tratamento dos dados disponíveis, verificou-se que os consumidores de energia elétrica monitorizados correspondem aos contadores parciais instalados em diferentes áreas operacionais da unidade, nomeadamente: o edifício principal, o edifício 50S, a cozinha principal, cozinha 50S, pastelaria e lavandaria. Estes dados permitiram avaliar o comportamento energético destas áreas específicas, evidenciando, que o

edifício principal representa cerca de 85% do consumo total da unidade, sucessivamente o edifício 50S (17%) e a cozinha principal (15%). A lavandaria e pastelaria representam individualmente cerca de 3% e a cozinha 50S cerca de 4,5% (Figura 4.13).

No seguimento do subcapítulo 4.1.1, concluiu-se que, no ano de 2024, existiram reduções consideráveis no consumo global de energia elétrica, sendo que essa conclusão também fica evidente no consumo parcial do edifício principal, cerca de 7% de redução relativamente variação de 2024 referente a 2023. Verifica-se esses resultados na Figura 4.14, onde se apresenta a distribuição mensal através de diagramas de caixa, salienta-se que a ocupação foi superior e que os dados estatísticos do edifício 50S posiciona-se num registo superior a 2023. Nomeadamente no comparativo de 2024 com 2023, verificou-se que nos meses de junho a agosto os intervalos interquartis foram superiores aos limites superiores do ano anterior.

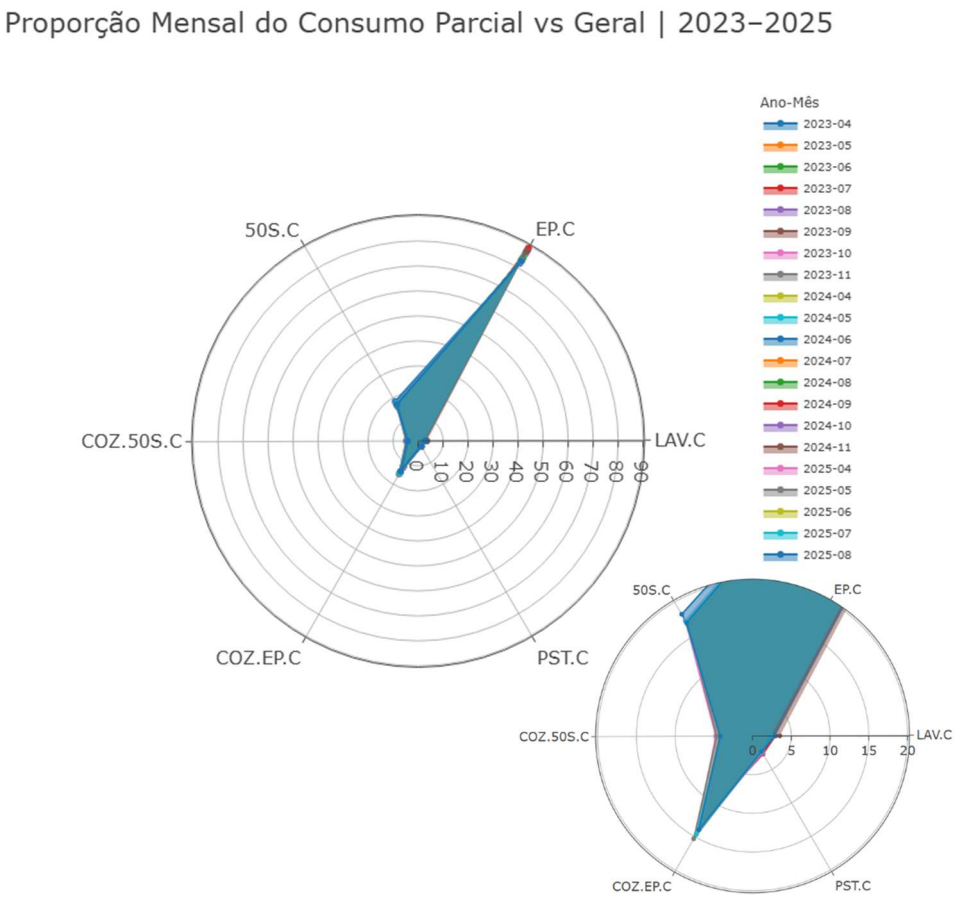


Figura 4.13 - Gráfico radar com a proporção dos consumos mensais dos contadores parciais

A Figura 4.15 explica os dados dos restantes contadores parciais, que apenas apresentaram um registo disponível desde julho de 2024, onde se verificou que os consumos, mantêm-se relativamente estáveis, ao longo da maioria dos meses analisados. Nomeadamente, serve de

exemplo a pastelaria e a lavandaria, onde em vários meses apresentaram resultados em que o intervalo interquartil é bastante pequeno ou nulo (ou seja, o primeiro quartil=mediana=terceiro quartil), demonstrando estabilidade operacional devido a ausência de variações diárias com grande amplitude, consequentemente indica eficiência e uma boa previsibilidade. Esse tipo de resultados pode ser interpretado como um sinal positivo de regularidade e eficiência, contudo deve também ser analisado criticamente com o intuito de despistar problemas de medição ou no registo dos dados. Neste sentido reforça-se a ideia que se deve sempre cruzar com outros indicadores, medições e inspeções para confirmar se a estabilidade é real ou apenas aparente. Na cozinha principal fica evidente que nos dois meses de verão, junho e agosto, a maior parte dos dados referentes a 2025 estiveram concentrados acima do limite máximo dos respetivos meses em 2024. Relativamente à mediana desses dois meses, em 2025 o consumo foi superior a 2024 cerca de 19%, relembrando que a ocupação foi inferior.

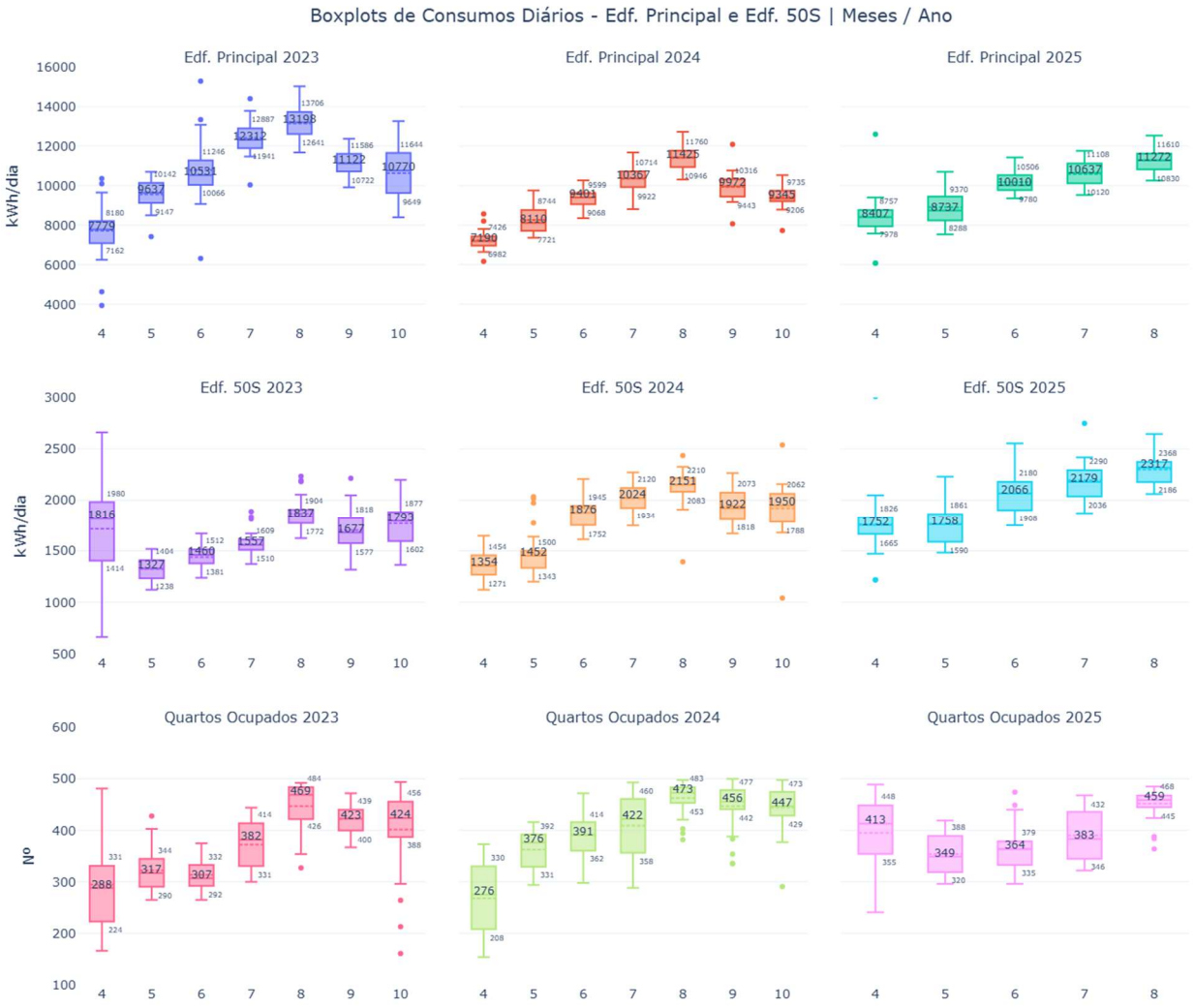


Figura 4.14 - Diagrama de caixa com distribuição estatística dos valores por mês e por ano, referente aos consumos parciais de cada edifício comparativamente com o nº de quartos ocupados.

Ao desagregar os consumos por setor, fica-se sensível à necessidade de obter maiores estruturas de dados. Isto é, quando se verifica reduções ou aumentos de consumos é justamente na observação dos valores registados em cada consumidor parcial que se pode interpretar as causas dessas respetivas variações. A título de exemplo, neste caso de estudo, ao se verificar que no edifício principal existiu uma redução de consumos 7% (2024-2023) mas que no mesmo período o edifício 50S teve um aumento de cerca e 18%. Devido ao insuficiente registo de consumos parciais não é possível analisar os motivos dessa divergência, não só medições nos consumidores parciais mas também, o registo de ocorrências de manutenção, registo de eventos especiais (com uma requisição extraordinária de recursos), de alterações operacionais relevantes (como mudanças de horários de funcionamento ou de ocupação), de substituição ou avaria de equipamentos, de intervenções de eficiência energética implementadas, bem como de condições ambientais atípicas (temperatura exterior, humidade, ondas de calor ou frio) que possam influenciar diretamente o consumo. A integração destes dados complementares permitiria correlacionar variações de consumo com causas específicas, possibilitando uma gestão energética mais precisa e orientada para a otimização.

Prende-se a necessidade que, para atingir objetivos de eficiência energética, é necessário interpretar o motivo das variações divergentes e reagir no sentido de melhorar a respetiva operação. Importante reforçar que uma análise apenas global pode mascarar comportamentos anómalos e que é na análise exploratória dos consumidores parciais que se identificam falhas ou ineficiências operacionais.

Portanto, para uma gestão energética verdadeiramente abrangente, seria fundamental integrar no sistema de monitorização outros pontos de consumo relevantes, tais como o AVAC (ar condicionado, ventilação e aquecimento), pois frequentemente um dos maiores consumidores de energia em unidades hoteleiras, assim como as demais áreas técnicas, nomeadamente as dedicadas a centrais de bombagem, centrais frigorificas, oficinas, entre outros setores relevantes. A iluminação, especialmente em zonas de grande permanência. Complementarmente, reforça-se a ideia que a monitorização individual de equipamentos de elevado consumo elétrico e impacto na operação geral do hotel, é importante no sentido das metodologias de otimização de recursos energéticos e operações de manutenção.

A inclusão destes contadores adicionais permitiria não só quantificar com maior precisão o peso de cada setor no consumo global, mas também detetar desvios operacionais e implementar medidas corretivas direcionadas. Assim, a desagregação detalhada dos consumos deixa de ser apenas uma ferramenta de auditoria e passa a ser um instrumento ativo de gestão, capaz de sustentar políticas de eficiência energética e de redução de custos.

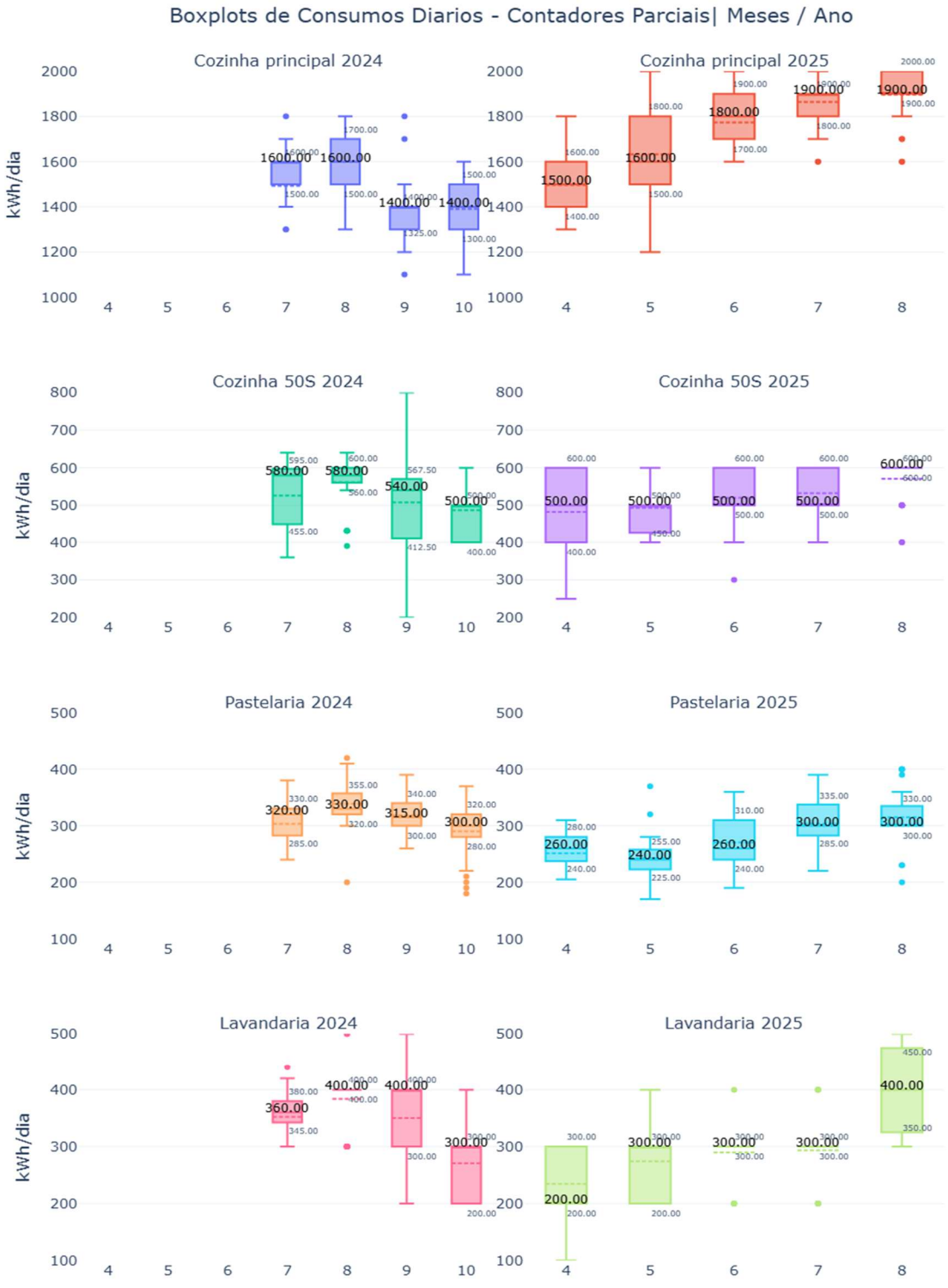


Figura 4.15 - Diagrama de caixa com distribuição estatística dos valores por mês e por ano, referente aos consumos parciais de cada setor, disponível no grupo de dados.

4.5.3 GÁS PROPANO

Relativamente aos consumos parciais de gás propano, neste caso em estudo, verificou-se que o único contador parcial existente para está instalado na lavandaria, o que permite conhecer com detalhe o perfil de consumo deste setor específico. Contudo, esta informação isolada é insuficiente para compreender o impacto global do gás propano na operação do hotel, uma vez que este combustível é utilizado em diferentes áreas funcionais. A ausência de contadores parciais em outros pontos de consumo limita a capacidade de identificar ineficiências, comparar desempenhos e implementar medidas de otimização direcionadas. Assim, para além da lavandaria, deveriam ser considerados os seguintes contadores adicionais:

a. Cozinhas: O gás propano é frequentemente utilizado em fogões, fornos e outros equipamentos de confeção alimentar. A instalação de contadores parciais permitiria avaliar a eficiência do processo de produção alimentar, identificar períodos de maior consumo (ex.: noites temáticas ou eventos extraordinários) e apoiar a implementação de medidas de racionalização, como a substituição de equipamentos por versões mais eficientes.
b. Produção de Água Quente Sanitária (AQS): Um contador dedicado permitiria monitorizar a eficiência dos sistemas de aquecimento, nomeadamente através de caldeiras, avaliar perdas energéticas e apoiar decisões de investimento em soluções renováveis (painéis solares térmicos, bombas de calor).
c. Piscinas e Spa: O aquecimento da água de piscinas interiores ou de spa representa um consumo significativo e contínuo. A instalação de contadores específicos permitiria quantificar este impacto e identificar oportunidades de otimização, como a melhoria do isolamento térmico ou a integração de sistemas híbridos.
d. Aquecimento: Este hotel utiliza caldeiras a gás propano para climatização na estação de aquecimento, a medição dedicada deste consumo é essencial para avaliar a eficiência do sistema e justificar eventuais substituições por soluções mais sustentáveis.

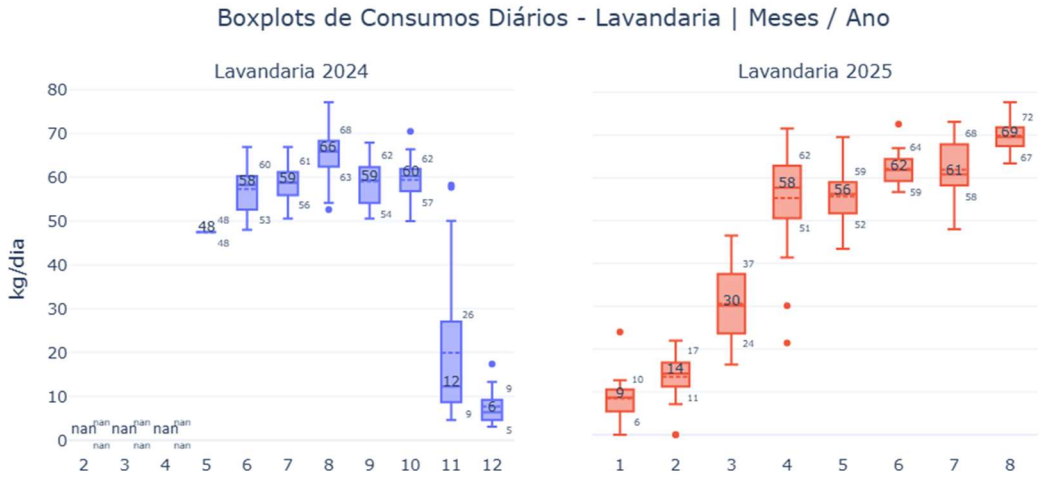


Figura 4.16 - Diagramas de caixa dos consumos diários do contador de gás propano da lavandaria ao longo dos meses de 2024 e 2025.

Conclusão, a existência de apenas um contador parcial na lavandaria fornece uma visão limitada do consumo de gás propano. Para uma gestão energética eficaz, é fundamental ampliar a rede de monitorização, instalando contadores adicionais nas cozinhas, na produção de AQS, nas piscinas/spa e nos sistemas de aquecimento ambiente. Esta desagregação permitirá não só uma análise mais rigorosa e detalhada, mas também a implementação de medidas de eficiência direcionadas, contribuindo para a redução de custos operacionais e para a sustentabilidade da unidade hoteleira.

4.6 Indicadores de Eficiência Operacional

4.6.1 Consumo Diário por Cliente

O indicador *Consumo Diário por Cliente* pode ser um KPI (*Key Performance Indicator*) extremamente útil para avaliar eficiência operacional, sustentabilidade e qualidade de serviço em setores como a hotelaria. Ele permite uma análise mais refinada do uso de recursos por unidade de atendimento, ajustando o consumo total à necessidade real.

Consumo Diário por Cliente é mais do que uma métrica operacional, uma ferramenta estratégica que permite alinhar eficiência energética, sustentabilidade e qualidade de serviço. A literatura científica reforça sua relevância ao mostrar que indicadores normalizados por cliente são mais sensíveis, comparáveis e acionáveis do que métricas agregadas.

No caso específico do ano de 2025, conforme ilustrado na Figura 4.17, observa-se um aumento significativo nos consumos por cliente em relação ao ano anterior de 2024. Este comportamento anómalo constitui um alerta para a necessidade de revisão dos processos de manutenção no ciclo anual subsequente, com vista à redução dos erros relativos e à reposição dos níveis de eficiência energética previamente alcançados. Os diagramas de caixa apresentam que o *Consumo Diário por Cliente* sofreu um aumento, de cerca de 6% relativamente à mediana, tanto nos consumos energia elétrica por cliente, assim como nos consumos de água. Em divergência o recurso gás propano, em termos de valores globais, apresentou uma variação negativa de 12%. Em termos de resultados comparativos entre 2025 e 2023, observa-se o oposto, uma variação negativa de cerca de 6.5% e 5%, respetivamente, para a energia elétrica e a água potável. O gás propano sofreu um aumento de 3%.

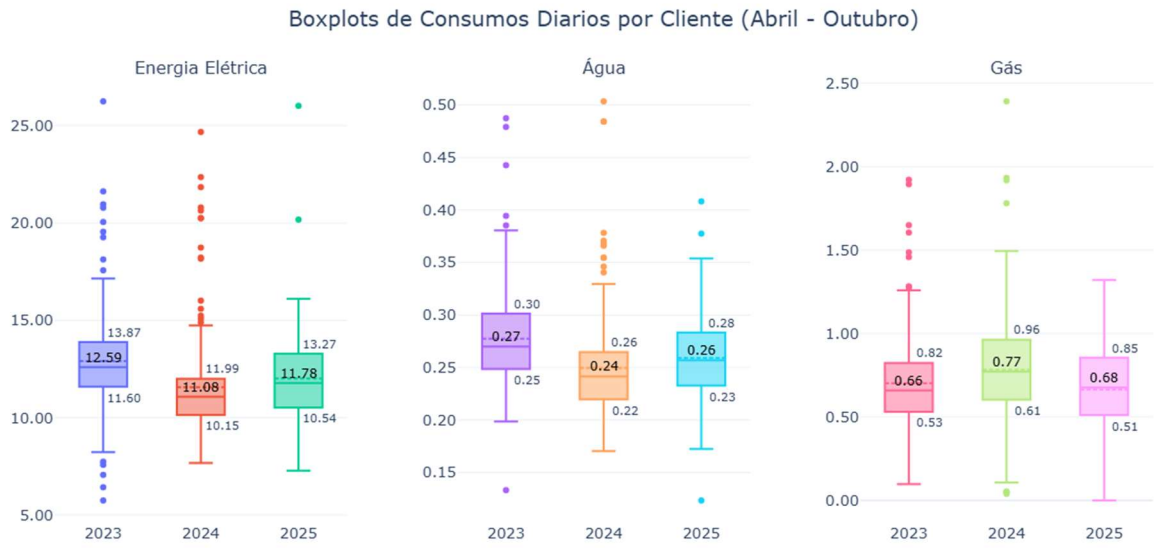


Figura 4.17 - Diagramas de caixa dos valores estáticos anuais durante o período de operação (abril a outubro)

Esta análise permite identificar padrões sazonais de consumo, particularmente associados ao período de verão, que podem servir como referência para a definição de metas futuras de desempenho energético. A utilização de indicadores normalizados por cliente, como o consumo diário por ocupante, constitui uma abordagem robusta para o benchmarking energético, conforme evidenciado por Bohdanowicz & Martinac (2007)[32] e Piscitelli et al. (2024)[40], que demonstram a eficácia de métricas ajustadas à ocupação na avaliação da eficiência operacional e na deteção de desvios significativos.

A titulo de exemplo, no caso de estudo, quando a análise do consumo diário de água por hóspede revela que o intervalo interquartil observado situa-se 220 e 300 litros por hóspede/dia, o que excede significativamente a média nacional de consumo doméstico individual, que se situa entre 180 e 190 litros por pessoa/dia, segundo dados da ERSAR (2024), contudo segundo relatório de monitorização do "Compromisso com a Eficiência Hídrica para o setor do turismo", elaborado pela ADENE em 2024, os dados reportados pelos empreendimentos revelaram ser compreendidos em torno de 345 litros por hospede/dia, em 2024. Espera-se uma discrepância de valores no contexto hoteleiro, dado a considerações de serviços associados à estadia, como lavandaria, restauração, piscinas e spa, assim como consumo rega e outros associados à operação de cada unidade. A comparação com os valores nacionais serve como referência crítica para avaliar o desempenho ambiental e identificar oportunidades de redução sem comprometer o conforto e a qualidade do serviço. Também importante, o objetivo de delimitar valores de alerta ou metas a serem cumpridas que promovam a eficiência hídrica, assim como gerir o desempenho hídrico em tempo real e visando um futuro sustentável com base no histórico de dados.

Adicionalmente, a variável "custo do recurso por ocupante" revela-se um indicador de desempenho estratégico, cuja monitorização contínua pode apoiar decisões técnicas em tempo real. Ao combinar gráficos de controlo que mostrem limites, valores de referência ou indicadores de desempenho com gráficos de variáveis de medição, pretende-se implementar ferramentas eficazes de monitorização e controlo em tempo real, com o objetivo de identificar prontamente desvios dos valores esperados, assim como para detetar anomalias e avaliar as tendências de melhoria ou agravamento do desempenho energético ao longo do tempo[4].

Ao comparar o ano 2025 com 2024 através da Figura 4.17, os diagramas de caixa apresentam que o *Consumo Diário por Cliente* sofreu um aumento de cerca de 16%, em média durante os meses de maio a julho, relativamente à eletricidade. Na água a variação do consumo por cliente foi cerca de 10% (valor médio) no mesmo período. No mês de agosto, para os recursos energia elétrica e água potável, os valores apresentam uma variação negativa menos expressiva, cerca de 5% e 1% respetivamente.

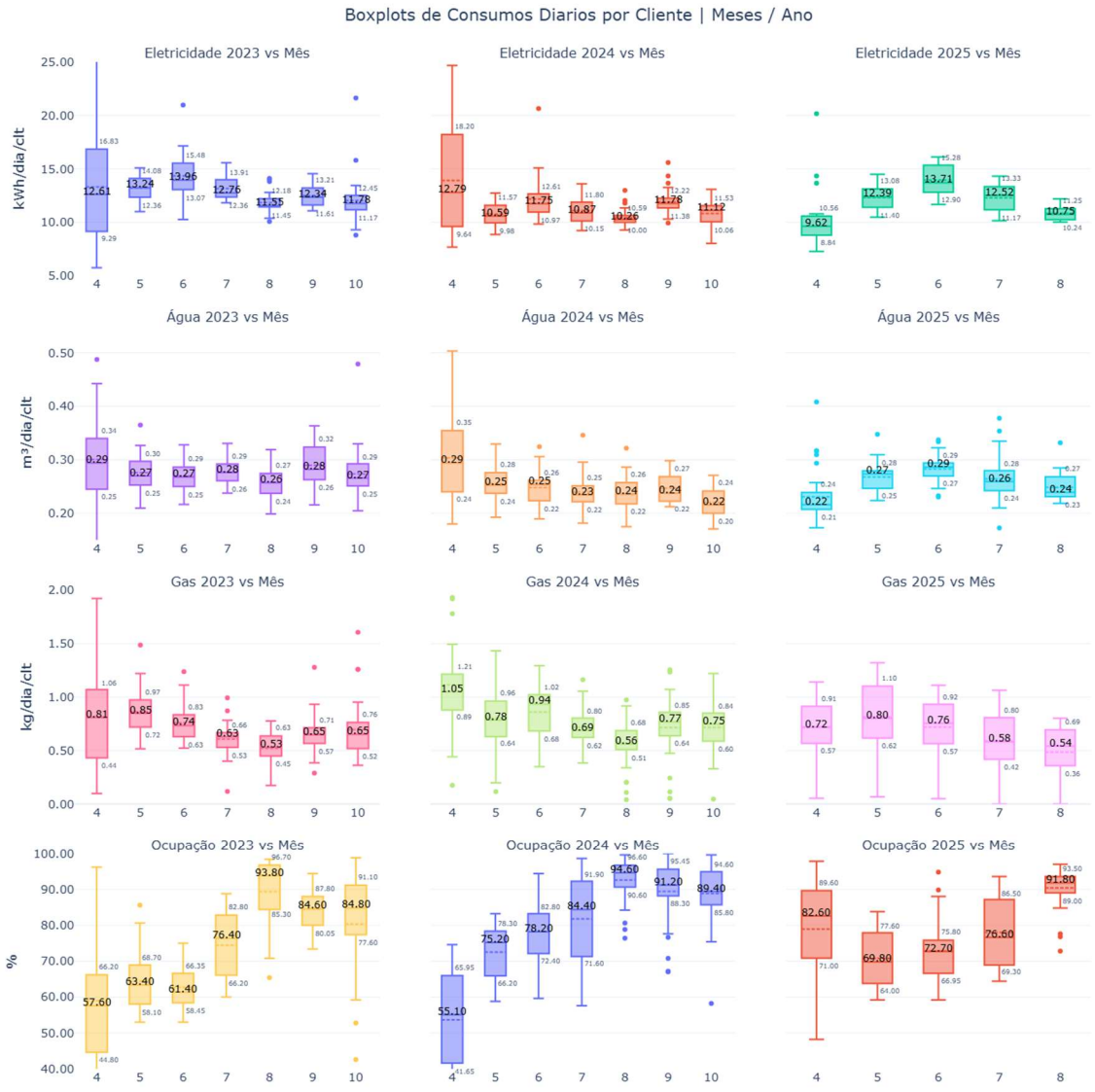


Figura 4.18 - Distribuição estatística do valores referente ao consumo diário por cliente, por mês e ano

A Figura 4.18 retrata a comparação do ano 2025 com período homologo de 2024. Admitindo que 2024 pode ser utilizado como período de referência, ao considerar o histórico de medições disponível, é interessante comparar como as variáveis se comportam diariamente no ano de 2025 e qual a sua variação relativa ao aos valores estáticos de 2024. Contudo é importante relembrar qual o contexto da operação, onde sabe-se que a ocupação variou negativamente cerca de 3% relativamente a 2025-2024. Portanto, pode-se refletir que se a ocupação diminuiu o consumo por cliente tende para aumentar, principalmente devido aos serviços disponibilizados, assim como as tipologias de quarto ocupado. Nem sempre pode ser possível justificar que, quanto menos clientes, menor será o consumo global. Como fica evidenciado,

nos diagramas de caixa por mês, é no mês de abril que o intervalo interquartil é maior, assim como a respetiva mediana.

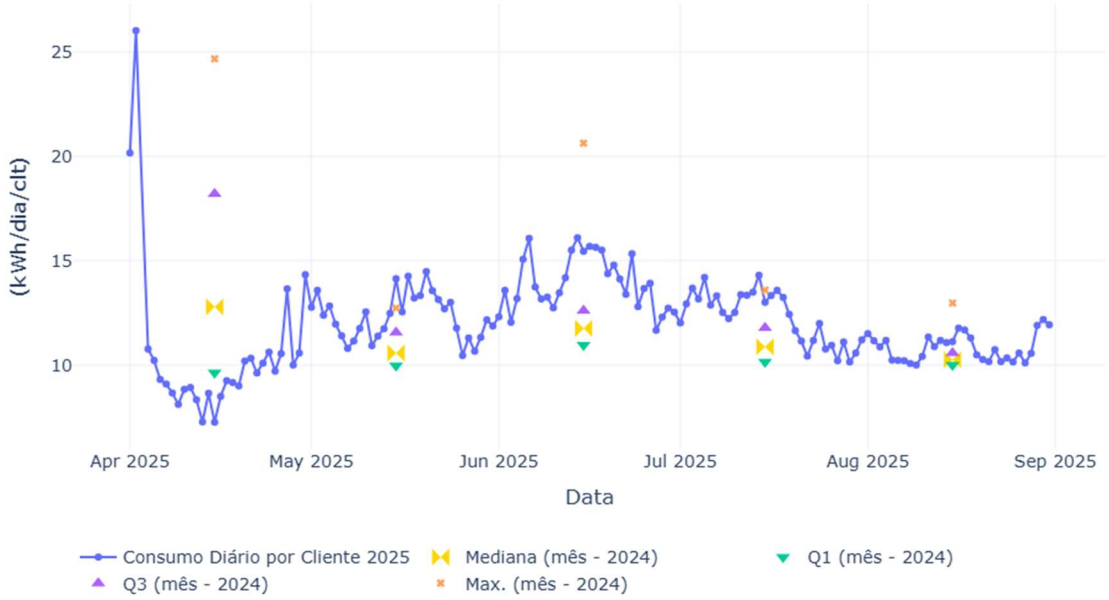


Figura 4.19- Comparação do consumo de energia elétrica por cliente, entre 2025 e o período homologo de 2024.

Quando se prende estudar como estivemos, como estamos e como queremos estar é importante definir valores de referência, limites operacionais, e objetivos a concretizar.

No seguimento da análise da Figura 4.19, as referidas estatísticas mensais do ano 2024 servem como marcos de avaliação, permitindo não apenas medir o desempenho passado e presente, mas também projetar cenários futuros de forma fundamentada.

A integração de métodos estatísticos, ao monitorizar variáveis em tempo real, pode ser um suporte à criação de limites a serem ajustados ou até mesmo para alertas para intervenções antecipadas. Ao analisar dados de forma sistemática, é possível definir limites dinâmicos, ajustáveis conforme as condições operacionais, e estabelecer indicadores de alerta que sinalizem a necessidade de intervenções de manutenção.

Mais uma vez, um método para contribuir na gestão de recursos energéticos, na prevenção de falhas e redução de desperdícios. Monitorizar uma variável de forma relativa a indicadores de eficiência operacional promove a capacidade de resposta a variações inesperadas, garantindo

maior resiliência e sustentabilidade do sistema. Prende-se a visão que transformar dados em informação acessível, melhora o desenvolvimento de estratégias de curto, médio e longo prazo. Para que isso seja possível, é importante definir as medidas para interpretar valores ótimos, tal como definir previamente as métricas e parâmetros de avaliação que permitam estabelecer limites de desempenho, faixas de tolerância e metas realistas.

4.6.2 Análise do Erro Relativo diário como Métrica de Diagnóstico Operacional

A análise do erro relativo entre o consumo energético e a ocupação diária reforça uma abordagem eficaz para identificar inconsistências operacionais, variações atípicas e potenciais ineficiências em ambientes hoteleiros. Esta métrica tem sido utilizada como complemento à modelagem preditiva, à gestão energética e ao controlo de qualidade. Conforme já referido, a relação entre ocupação e consumo é um dos principais determinantes da eficiência operacional em unidades hoteleiras, destacando a importância de indicadores proporcionais ao número de clientes [32][40]. A norma ISO 50001 (2018) estabelece diretrizes para gestão energética com ênfase em indicadores ajustados à ocupação e no controlo contínuo da performance, sendo o erro relativo diário um bom exemplo de métrica para esse fim.

O erro relativo é uma ferramenta essencial na análise de medições porque permite avaliar a precisão e a confiabilidade dos dados em relação ao valor real ou esperado. Esta métrica é especialmente útil quando se quer entender o impacto do erro em termos proporcionais, independentemente da escala da medição.

O erro relativo da variação diária, das variáveis em analise consumo e ocupação, é uma tentativa de desenvolver métricas que sejam indicadoras de condição, performance, ou apenas de um desvio inesperado no funcionamento padrão de um ativo. Ao aplicar esta métrica, tanto do consumo diário como da variação diária da ocupação, pode auxiliar diagnóstico para identificar falhas ou desvios de comportamento em sistemas complexos, comparando o valor observado com o valor esperado com base em variáveis correlacionadas.

O consumo de energia elétrica de um edifício apresenta uma mediana ao longo de um determinado tempo (ou nº de observações) e esta é relativa à necessidade imposta pela ocupação do hotel. Então, admite-se que a variação diária da ocupação deve ser equivalente à variação diária do consumo de energia. Em prática, se um número de clientes aumenta 10% é esperado que o consumo geral de eletricidade aumente cera de 10%. Se não, ou se o erro relativo ao dia anterior for muito distante entre as duas variáveis, deve-se perceber o porquê, assim como se essa pode descrever o funcionamento eficiente de um determinado ativo.

Ao se estudar a variável erro relativo diário ajuda a detetar desvios inesperados no comportamento padrão, tais como:

a. Desperdício de energia: o consumo aumentou sem justificação proporcional na ocupação.
b. Subutilização: um consumo mais baixo, mesmo com ocupação estável ou crescente.
c. Falha operacional: equipamentos ligados sem necessidade, climatização excessiva, etc.
d. Mudança no perfil de consumo: hóspedes com hábitos diferentes, eventos especiais, etc. — aumento no consumo sem aumento proporcional na ocupação.

No âmbito de uma monitorização eficaz de consumos de energia, ou outros recursos, considerar a variação proporcional da variável independente e comparar com as outras variáveis, pode indicar valores consumos atípicos, falhas operacionais ou comportamentos fora do padrão esperado.

A aplicação de técnicas de análise em séries temporais, utilizando filtros sazonais (por exemplo, o intervalo abril–outubro de 2023 e 2024), permite observar a distribuição e evolução do erro relativo diário entre consumo e ocupação. Esta segmentação não só revela padrões sazonais como também evidencia melhorias na eficiência energética por cliente ao longo do tempo, sobretudo quando associada a alterações estruturais ou operacionais.

Ao aplicar esta metodologia ao caso de estudo, compreende-se como o acompanhamento contínuo do erro relativo diário pode representar um instrumento decisivo para análises aprofundadas, diagnósticos energéticos mais robustos e intervenções orientadas à otimização do desempenho global das instalações monitorizadas.

A Figura 4.20, mostra como o acompanhamento desta variável pode ser pertinente em tempo real numa operação. Como se verificou em 2024, a 5 de setembro existiu uma variação relativa erros relativos diários com alguma expressão. Considerar o erro relativo como uma métrica indicadora de desempenho, torna-se pertinente definir limites operacionais que permitam classificar os níveis de alerta e antecipar condições críticas. A parametrização desses limites, por via estatística, permite estabelecer níveis de normalidade, atenção e alarme, por forma a facilitar a gestão energética baseada em dados históricos.

A Figura 4.21 mostra como o acompanhamento desta variável pode ser pertinente em tempo real numa operação. Como se verificou em 2024, destacam-se variações expressivas nos dias 16 de agosto, 30 de agosto, 5 de setembro e 24 de setembro, em que os erros relativos diários apresentaram picos significativos. A análise cruzada com os consumos parciais nos diferentes setores evidencia que estes desvios estão fortemente associados ao comportamento da Cozinha

EP e da Cozinha 50S, setores cuja evolução de consumo apresenta uma relação de proporcionalidade direta. Esta relação confirma que alterações operacionais nestes setores têm impacto imediato na métrica de erro relativo, reforçando a pertinência da sua monitorização contínua.

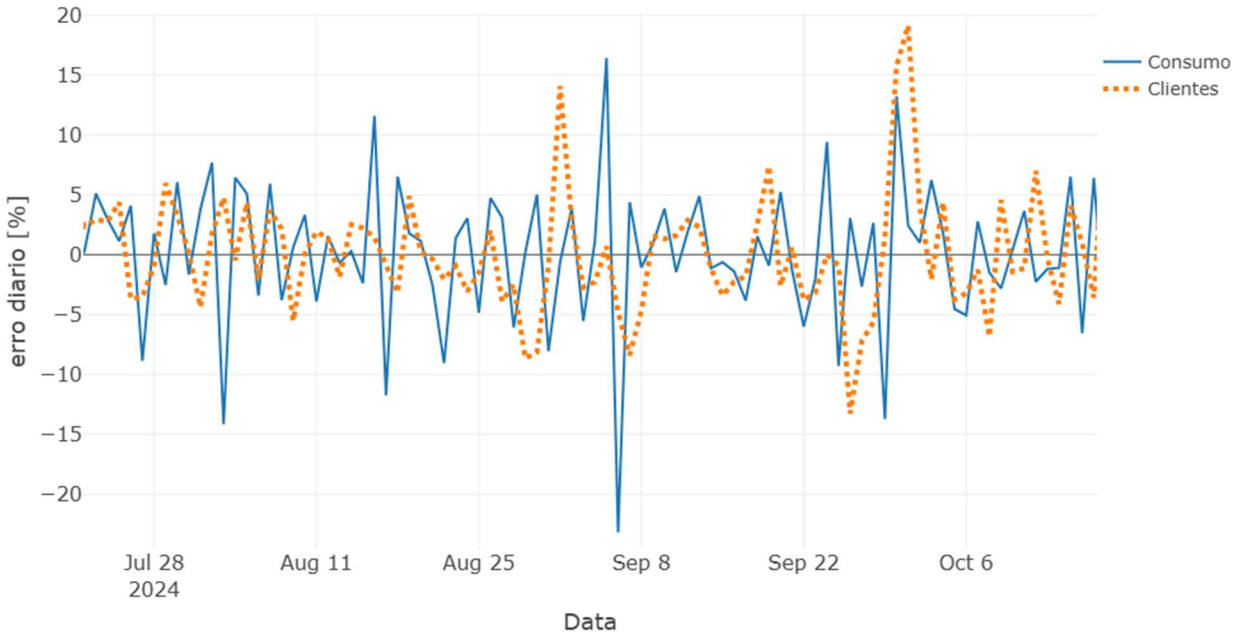


Figura 4.20 - Variação diária relativa ao dia anterior, para as variáveis Consumo e nº de Clientes

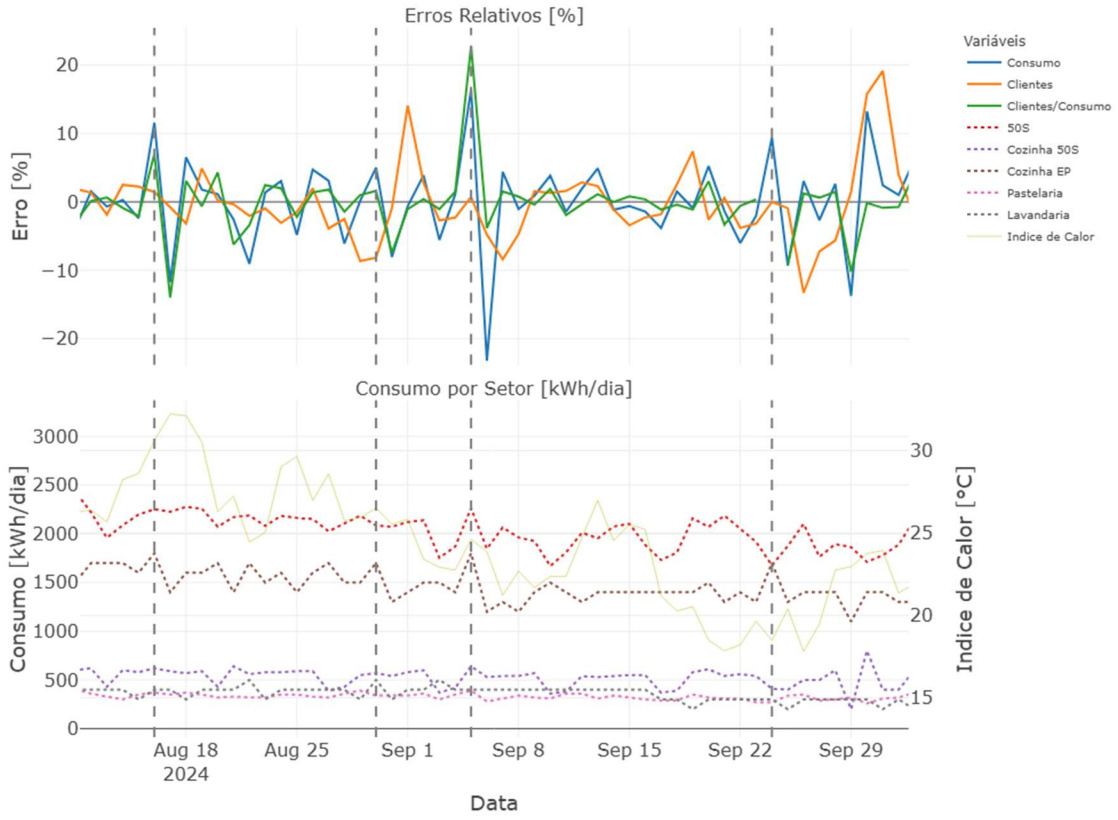


Figura 4.21 - Variação diária relativa ao dia anterior, clientes e consumo, cruzada com a analise dos consumos parciais nos diferentes setores e índice de calor diario.

Considerar o erro relativo como métrica indicadora de desempenho torna-se, assim, fundamental para definir limites operacionais que permitam classificar níveis de alerta e antecipar condições críticas. A parametrização desses limites, por via estatística, possibilita estabelecer zonas de normalidade, atenção e alarme, facilitando a gestão energética baseada em dados históricos.

A integração desta metodologia no sistema de monitorização acrescenta valor à análise de dados temporais, aumentando a capacidade preditiva e reforçando os mecanismos de decisão orientados à sustentabilidade energética e operacional.

Com base na análise desenvolvida, é possível delinear um conjunto de recomendações para aplicação prática em contextos de gestão energética. Tais como, medidas de implementação de sistemas de monitorização em tempo real, com integração de indicadores normalizados por cliente e alarmes visuais baseados em erro relativo. Estes sistemas devem permitir a deteção imediata de desvios e a ativação de protocolos de intervenção. Definição de limites operacionais dinâmicos, ajustados à sazonalidade e ao perfil de ocupação, com base em estatísticas históricas. A utilização de quartis e medianas como referência permite uma

abordagem estatisticamente robusta e adaptável. Integração dos indicadores com sistemas de manutenção técnica, permitindo a correlação entre desempenho energético e estado dos equipamentos. Esta integração deve apoiar a transição para modelos de manutenção preditiva, com base em dados reais de consumo. Formação das equipas operacionais na leitura e interpretação dos indicadores, promovendo uma cultura de eficiência e responsabilidade energética. A compreensão dos dados por parte dos técnicos é essencial para garantir a eficácia das ações corretivas. Revisão periódica dos indicadores e metas de desempenho, com base em análises anuais e benchmarking. Esta revisão deve considerar não apenas os valores absolutos, mas também a evolução dos padrões de consumo e a eficácia das medidas implementadas.

A adoção de ferramentas analíticas como as descritas neste capítulo representa um avanço significativo na transição para operações mais eficientes e ambientalmente responsáveis. Neste contexto, a aplicação de modelos de inteligência artificial, especialmente algoritmos de regressão e classificação supervisionada, pode funcionar como complemento à caracterização de rótulos operacionais. Métodos como *Decision Tree (DT), Random Forest (RF), Support Vector Machine*s (SVM) ou Redes Neuronais Artificiais (RNA) podem ser treinados para reconhecer padrões de desvios e associar amplitudes de erro a classes de comportamento (como por exemplo, normal, desvio moderado, desvio crítico). Esta metodologia contribui para a automatização da deteção de anomalias, assim como para p desenvolvimento de sistemas de alarme inteligentes que se ajustam dinamicamente em função da operação.

4.7 MEDIDAS ADOTADAS PARA INTERPRETAR VALORES ÓTIMOS

4.7.1 ALARMES BASEADOS EM VALORES ESTATÍSTICOS

A inclusão de um formato de alarme, baseado no erro relativo entre o indicador consumo diário por cliente e a mediana histórica, reforça a capacidade de resposta operacional e a tomada de decisão baseada em dados.

Ao comparar os valores observados com os valores históricos de referência, é possível calcular erros relativos diários, cuja magnitude pode indicar a necessidade de intervenções técnicas ou operacionais. Valores elevados de erro relativo sugerem potenciais ineficiências, podendo estar associados a fatores como desgaste de equipamentos, colmatações em sistemas hidráulicos ou falhas em processos de manutenção. A análise conjunta destes desvios com o histórico de manutenção preventiva e corretiva permite inferir correlações entre o estado dos equipamentos e o desempenho energético, reforçando a aplicabilidade do conceito de manutenção preditiva.

A Figura 4.22 evidencia estes desvios destacando os dias, entre 6 e 20 de julho, em que o erro relativo ultrapassa os limites de tolerância, hipoteticamente definidos. Os marcadores coloridos podem indicar uma gravidade do desvio, esta codificação visual permite uma leitura imediata da performance energética e a identificação de períodos críticos.

Admite-se, que posteriormente a se identificar valores menos esperados, encontrar a correlação entre os desvios de consumos e os registos de manutenção poderá sugerir que determinados aumentos de consumo podem estar associados a falhas técnicas não resolvidas, ou degradação de equipamentos com maior peso nos consumos, entre outras. Esses resultados reforçam a importância da manutenção preditiva e da monitorização contínua como ferramentas de gestão energética.

A utilização de indicadores normalizados por cliente, combinada com a análise de erro relativo e a visualização gráfica, constitui uma abordagem robusta para a gestão de recursos energético. Esta metodologia não permite apenas avaliar o desempenho atual, mas também antecipar necessidades de intervenção, definir metas realistas e acompanhar a eficácia das ações implementadas.

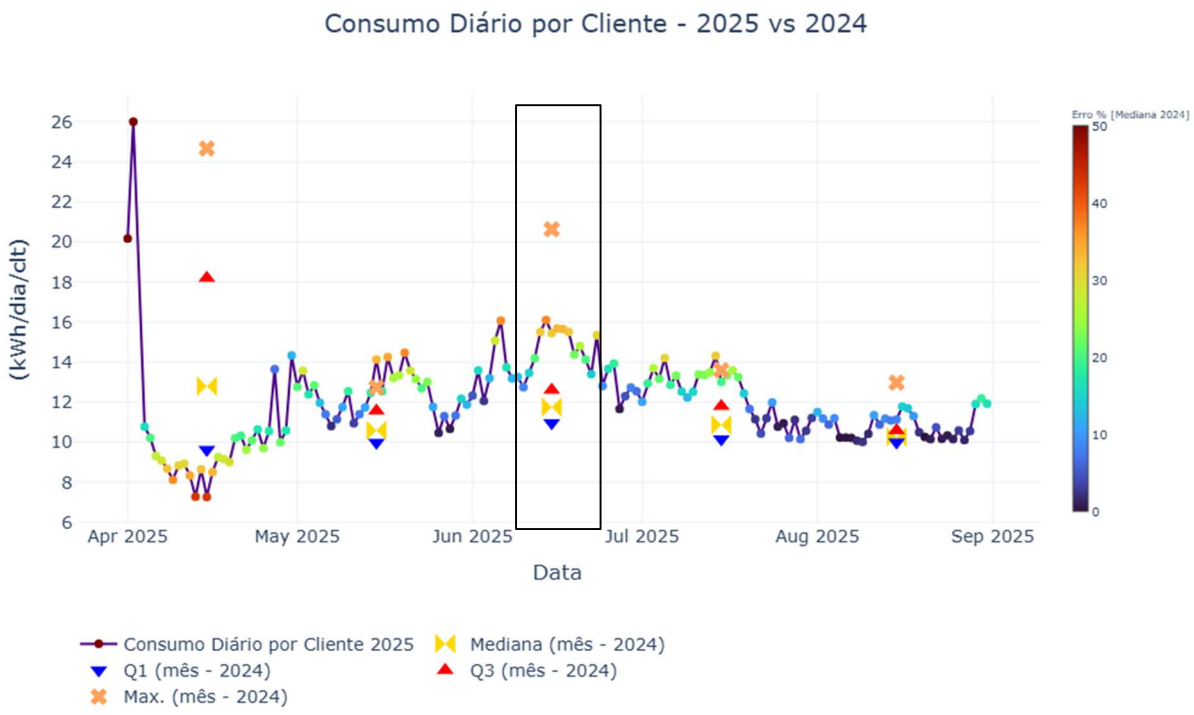


Figura 4.22 - Exemplo de grafico de controlo, com a utilização de indicadores normalizados por cliente, combinada com a análise de erro relativo e a visualização gráfica e limites.

4.7.2 MODELOS PREDITIVOS

A deteção e o diagnóstico de padrões atípicos podem também ser encarados como uma sofisticada adaptação de controlo de qualidade que incorpora algoritmos de inteligência artificial com o propósito de detetar com antecedência falhas e qual o seu grau de gravidade. O uso de técnicas de *machine learning* (ML) e de *deep learning* (DL), como *redes neurais recorrentes* (RNN), permite modelar e prever o consumo energético de edifícios, detetando valores ótimos de operação e pontos de ineficiência.

O reconhecimento de padrões refere-se à recolha de dados dos vários sensores e identificação de um padrão, algo que requer certos algoritmos e aplicações. Este estudo pretende utilizar os dados de aplicando diferentes técnicas e algoritmos, relacionando a consumos com a ocupação do edifício, assim como, com as series temporais que caracterizam o período de ocupação.

O aumento na disponibilidade de dados de consumo energético em edifícios e instalações industriais requer métodos avançados para análise e previsão. Modelos de ML oferecem variadas abordagens: enquanto *Decision Tree* e *Random Forest* focam em interpretabilidade e seleção de atributos, métodos como SVR e *Xtreme Gradient Boost* (XGB) garantem alta performance em conjuntos tabulares. Já redes neurais artificiais recorrentes (RNN), como GRU e LSTM, capturam padrões temporais, as arquiteturas convolucionais (CNN) extraem automaticamente representações espaciais. A combinação CNN-LSTM une ambas as forças, permitindo modelar séries temporais multivariadas com estrutura de alto nível.

A distinção entre Deep Learning (DL) e Machine Learning (ML) convencional reside em várias capacidades e desempenhos, conforme destacado pelas referências bibliograficas:

i. Categoria de Modelos: Tanto o *machine learni*ng quanto o *deep learning* são classificados como modelos "caixa preta", o que significa que são modelos orientados por dados que utilizam algoritmos para aprender a partir dos dados
ii. Desempenho na Previsão de Séries Temporais: Os modelos de Deep Learning (DL) frequentemente superam os modelos de Machine Learning (ML) na previsão de séries temporais.[41]
iii. Capacidade de Aprender Relações Complexas e Não Lineares: Uma das principais razões para o desempenho superior do DL é a sua capacidade avançada de aprender relações complexas e não lineares entre as características de entrada e saída. Esta é uma distinção crucial, pois as RNNs padrão (um tipo de DL) enfrentam desafios com dependências temporais de longo prazo devido a problemas de desvanecimento ou explosão de gradiente, que são resolvidos por LSTMs e GRUs.

iv. Lidar com Grandes Conjuntos de Dados: Os algoritmos de DL são mais eficazes no manuseio de grandes conjuntos de dados, o que lhes confere maior flexibilidade e escalabilidade para trabalhar com vários tipos de dados.

v. Adaptabilidade e Capacidades de Aprendizagem: Modelos "caixa preta" em geral (incluindo DL) são valorizados pelas suas altas precisões preditivas, requisitos limitados de dados ou informação prévia, capacidade de reconhecer e aprender relações complexas e não lineares dentro dos dados, escalabilidade, maior adaptabilidade e capacidades de aprendizagem.

vi. Enquanto os modelos estatísticos tradicionais têm limitações em lidar com grandes volumes de dados com não linearidade complexa, e os modelos de *machine learning* podem apresentar um problema de "atraso de tempo", os modelos de *deep learning* são notáveis pela sua forte capacidade de lidar com problemas não lineares completos, bem como pela sua precisão e robustez. [42]

O fluxo de trabalho típico envolve:

1. Preparar um conjunto de dados rotulado com exemplos de entrada e saída desejada.
2. Dividir em conjuntos de treinamento, validação e teste para ajustar e avaliar o modelo.
3. Escolher e treinar um algoritmo (por exemplo, regressão linear, árvores de decisão, SVM, redes neurais).
4. Medir o desempenho usando métricas como precisão para classificação ou erro quadrático médio para regressão.
5. Ajustar hiperparâmetros e validar até obter resultados satisfatórios.

Ao modelar para prever o próximo passo (próximo consumo/dia), podemos derivar curvas de consumo ótimas e ajustar restrições de conforto e custos energéticos. Simulação de *cenários de eficiência*, com base em séries temporais e modelos de regressão múltipla ou redes neurais recorrentes (RNN), conforme referenciado por Xiang S, et al. (2023)[43], promove um controlo inteligente, onde permitem aos utilizadores um controlo remoto em tempo real, rápido, económico e de baixo carbono dos sistemas de energia inteligentes.

4.7.3 APRENDIZAGEM DE MÁQUINA

Uma abordagem importante para sistemas de monitorização em unidades hoteleiras é recorrer à aprendizagem de máquina (*machine learning*) para modelar o comportamento normal de diferentes variáveis quando estas se encontram em estado saudável ou ideal. Idealmente, os

dados que representam esse funcionamento normal devem ser recolhidos em períodos em que a probabilidade de falha é reduzida, garantindo uma base de treino fiável para os modelos.
Com base nas entradas (variáveis independentes, ocupação), são construídos modelos de regressão que preveem a saída numérica (variáveis dependentes como consumo de energia elétrica) quando o componente a ser modelado assume o desempenho ideal. Ao modelar de forma independente essas variáveis, podemos avaliar melhor e prever o potencial de consumo por cliente num local específico, fazer melhores escolhas de intervenção técnicas, como monitorar e solucionar problemas de operação.
A avaliação quantitativa do desempenho de modelos de regressão é fundamental para aferir a sua capacidade preditiva e a apropriação ao problema em estudo. Entre as métricas mais utilizadas encontram-se o Erro Quadrático Médio (MSE), a Raiz do Erro Quadrático Médio (RMSE), o Erro Absoluto Médio (MAE) e o Coeficiente de Determinação (R^2). Cada uma destas métricas fornece uma perspetiva distinta sobre a qualidade das previsões, sendo frequentemente utilizadas de forma complementar.
O MSE é definido como a média dos quadrados das diferenças entre os valores observados e pretende quantificar o erro médio ao quadrado, penalizando de forma mais severa erros de maior magnitude. O RMSE é a raiz quadrada do MSE e fornecer uma medida de erro na mesma unidade da variável dependente, facilitando a interpretação prática pois está na escala original da variável. Também, valores mais baixos indicam previsões mais próximas dos valores reais.
O MAE é a média dos valores absolutos das diferenças entre valores observados e previstos. Isto é, mede o erro médio absoluto, trata todos os desvios de forma linear, sem penalização quadrática, representado assim o desvio absoluto, em média, entre a previsão e a realidade.
O R^2 mede a proporção da variabilidade da variável dependente explicada pelo modelo. Avalia o grau de ajuste do modelo aos dados observados. Varia entre $-\infty$ e 1, onde $R^2 = 1$ indica ajuste perfeito e $R^2 = 0$ indica que o modelo não explica melhor do que a média.
A escolha das métricas deve estar alinhada com os objetivos do estudo e a natureza dos dados. Em contextos onde erros grandes têm impacto crítico (por exemplo, previsão de consumo energético em larga escala), o RMSE pode ser mais relevante; já em cenários onde a robustez a valores extremos é desejada, o MAE pode ser preferível.

4.7.3.1 Decision Tree e Random Forest

As *Decision Trees* (árvores de decisão) são modelos supervisionados que estruturam a tomada de decisão em forma de árvore, dividindo iterativamente o espaço de dados com base em regras simples sobre as variáveis independentes. No contexto de monitorização hoteleira,

podem ser usadas para prever estados operacionais ou categorias de consumo energético a partir de variáveis como ocupação, temperatura ambiente ou horários de pico. O Random Forest expande este conceito ao criar múltiplas árvores de decisão treinadas sobre subconjuntos aleatórios dos dados e das variáveis, combinando os resultados por votação (classificação) ou média (regressão). Esta abordagem reduz o risco de overfitting e aumenta a robustez, sendo útil para lidar com dados heterogéneos e ruído, comuns em sistemas reais de operação hoteleira.

Neste caso em estudo foram realizadas várias iterações, por forma a encontrar o melhor modelo treinado, Tabela 4.3. Foram realizadas quatro iterações de treino a cada modelo, no qual as duas primeiras foram justamente para avaliar os respetivos modelos em função dos inputs, os históricos de dados usadas foi o conjunto de 01/04/2023 a 31/10/2023 e 01/04/2024 a 31/10/2024.

A terceira e quarta iteração foram treinadas com base no ano de referência, 2024, com o objetivo de simular um benchmarking. Isto é, o treino foi realizado no intervalo de dados 01/04/2024 a 31/10/2024 por forma a que a variável preditiva represente um comportamento de referência, e assim seja um objetivo a cumprir. Os inputs da terceira e quarta iteração acompanham, respetivamente, a sequência da segunda da primeira iteração.

Na 1ª Iteração:

Input = Temperatura média diária, Humidade Relativa média diária, Índice de calor, nº de total de clientes, nº de quartos ocupados, nº de adultos, nº de crianças, dia da semana, trimestre, mês, ano, dia do ano, dia do mês e semana do ano.

Output = Consumo diário de energia por cliente.

Na 2ª Iteração, admite-se os mesmos inputs que na primeira iteração mais os atributos criados a partir dos dados originais. A criação de variáveis derivadas pretende capturar dependências temporais, volatilidades ou estabilidades, suavizar flutuações de tendências, por forma a melhorar o desempenho dos respetivos modelos, conforme:

Input = Temperatura média diária, Humidade Relativa média diária, Índice de calor, nº de total de clientes, nº de quartos ocupados, nº de adultos, nº de crianças, dia da semana, trimestre, mês, ano, dia do ano, dia do mês e semana do ano, consumo diário de energia por cliente do dia anterior, consumo diário de energia por cliente da semana anterior, média movel de 7 dias do consumo diário de energia por cliente, mediana movel de 7 dias do consumo diário de energia por cliente, desvio padrão e 7 dias do consumo diário de energia por cliente, erro relativo diário do consumo diário de energia por cliente.

Output = Consumo diário de energia por cliente

A Tabela 4.3 mostra que os resultados das avaliações realizadas em cada modelo e a sua respetiva iteração. Como se verifica em ambos os modelos, tanto no conjunto de treino como de teste, as métricas consideradas apresentam valores mais baixos quando adicionado as "variáveis derivadas" indicando um melhor desempenho dos modelos (iteração 1 e 2).

Contudo, admitindo que se quer prever com base em comportamentos de referência, as iterações 3 e 4 servem para quantificar qual a magnitude do erro entre os valores reais em 2025 e a previsão baseado no comportamento do ano 2024.

As Figuras 4.22 e 4.23 apresentam o comparativo das previsões com os valores observados, onde fica evidente que na 2ª iteração do treino dos modelos, quando adicionadas variáveis derivadas do output 'consumo diário de energia elétrica por cliente', os modelos apresentam descidas consideráveis nas métricas do desempenho e um o coeficiente de determinação forte, relativamente à 1ª iteração, face aos valores observados de 2025. Para complementar o raciocínio, verifica-se nas Figuras 4.24 e 4.25, que apresentam a importância das variáveis nos modelos, as referidas variáveis derivadas tomaram os lugares de maior importância, comparativamente à 1ª iteração onde as variáveis de ocupação seguram as primeiras posições.

Ao longo da modelação de variáveis de previsão, é importante considerar qual a importância dos inputs considerados por forma a compreender o impacto nos modelos preditivos assim como se estes são adequadas ao grupo de dados considerados. É importante refletir que se deve enriquecer a informação disponível através de vários atributos, garantido a facilidade de aprendizagem de relações complexas entre as variáveis de input, assim como reduzir ruídos e aumentar o desempenho das previsões.

Tabela 4.2 – Quadro de metricas de avaliação de desempenho dos modelos de árvores de decisão e random forest, ordenadas por iteração e conjunto.

Modelo	Iteração	Conjunto	MSE	RMSE	MAE	R^2
Árvore de Decisão	1	Treino	4,334	2,080	0,709	0,81
Árvore de Decisão	1	Teste	1,492	1,220	0,944	0,47
Random Forest	1	Treino	1,019	1,010	0,355	0,95
Random Forest	1	Teste	0,935	0,970	0,766	0,65
Árvore de Decisão	2	Treino	3,796	1,930	0,429	0,83
Árvore de Decisão	2	Teste	0,273	0,520	0,344	0,86
Random Forest	2	Treino	0,704	0,840	0,196	0,97
Random Forest	2	Teste	0,082	0,290	0,178	0,97
Árvore de Decisão	3	Treino	0,304	0,550	0,277	0,95
Árvore de Decisão	3	Teste	1,172	1,080	0,744	0,51
Random Forest	3	Treino	0,065	0,250	0,131	0,99
Random Forest	3	Teste	0,458	0,680	0,502	0,81
Árvore de Decisão	4	Treino	0,553	0,740	0,412	0,91
Árvore de Decisão	4	Teste	1,162	1,080	0,890	0,51
Random Forest	4	Treino	0,119	0,340	0,201	0,98
Random Forest	4	Teste	1,240	1,110	0,898	0,48

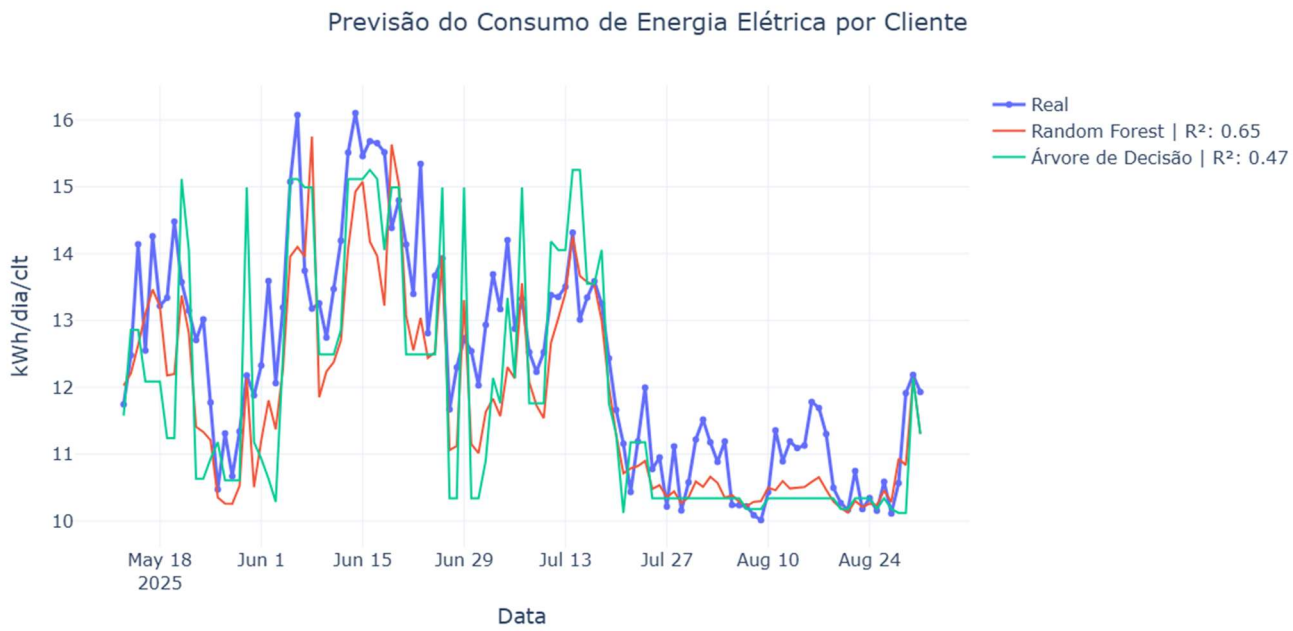


Figura 4.23 – Resultados comparativos entre os dados previstos e os dados observados da 1ª iteração au aplicar os modelos de arvore de decisão e random forest.

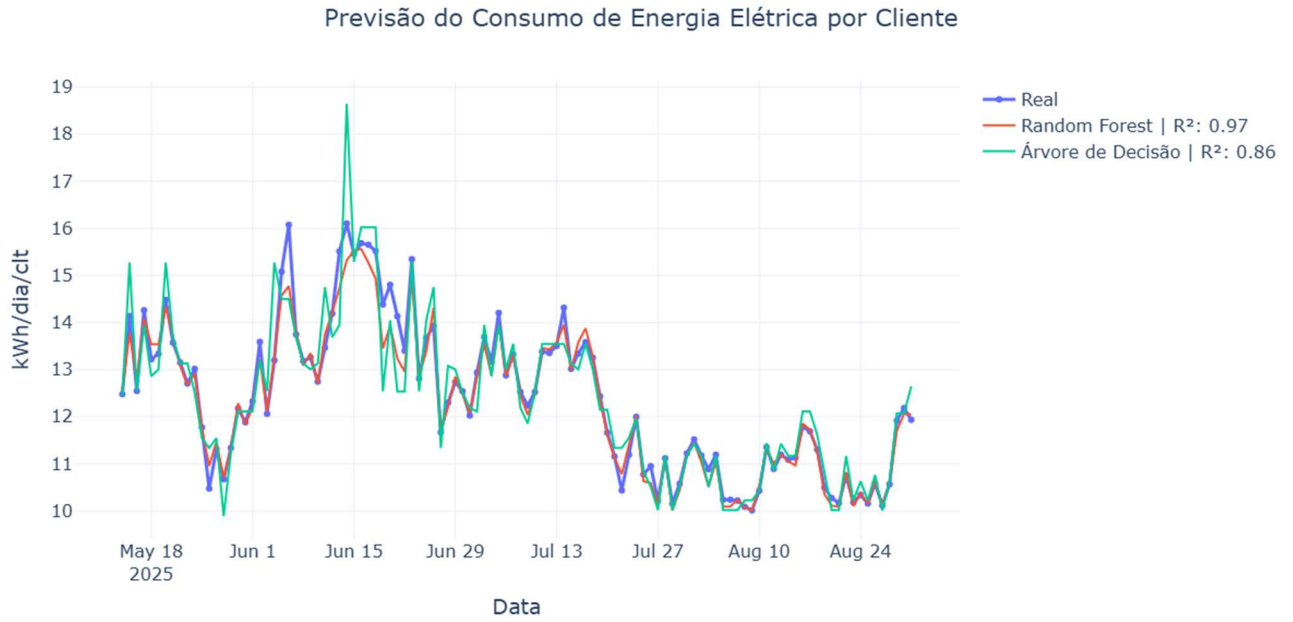


Figura 4.24 - Resultados comparativos entre os dados previstos e os dados observados da 2ª iteração au aplicar os modelos de arvore de decisão e random forest.

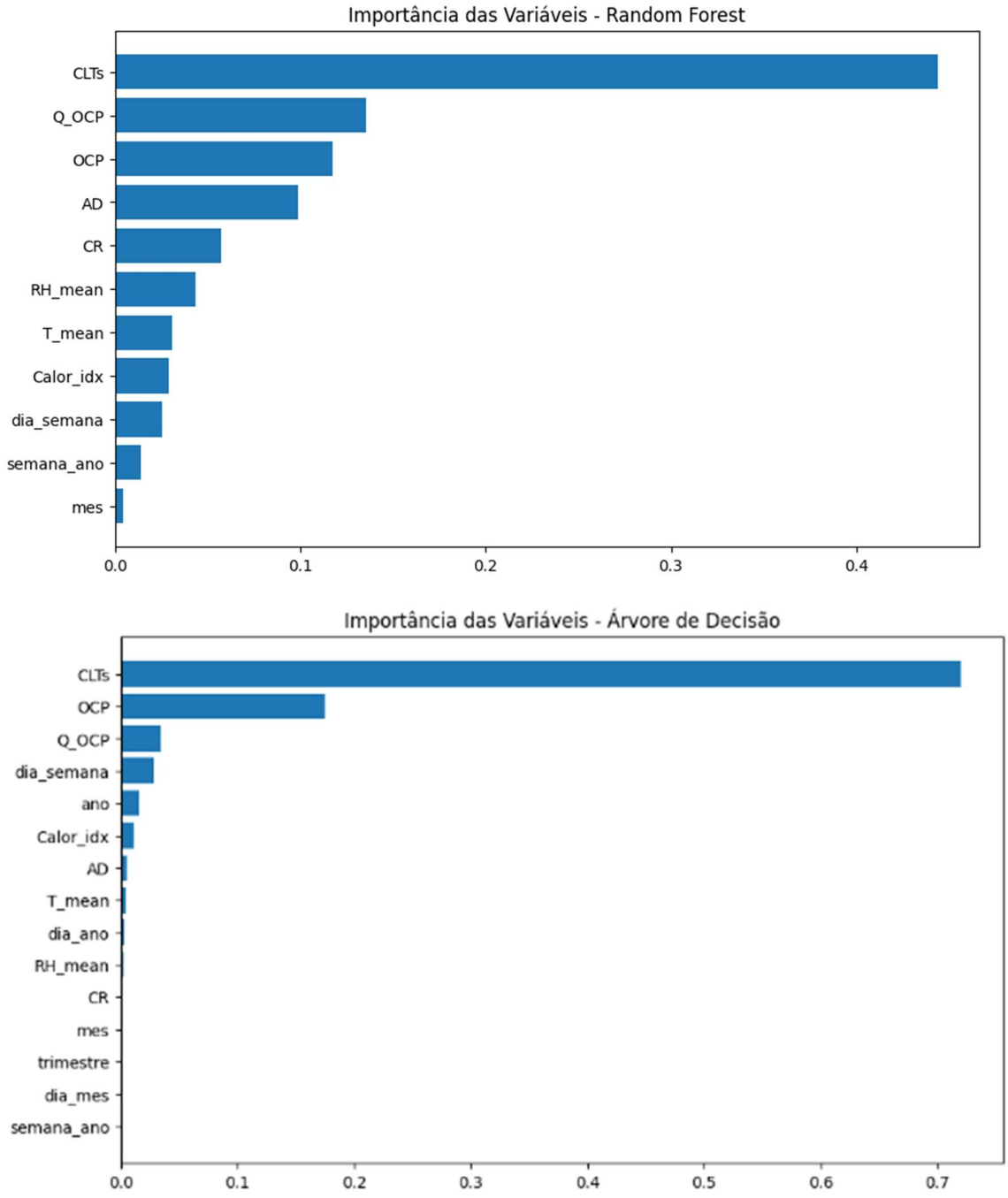


Figura 4.25 - Importância das variáveis input consideradas, comparação entre os dois modelos na 1ª interação.

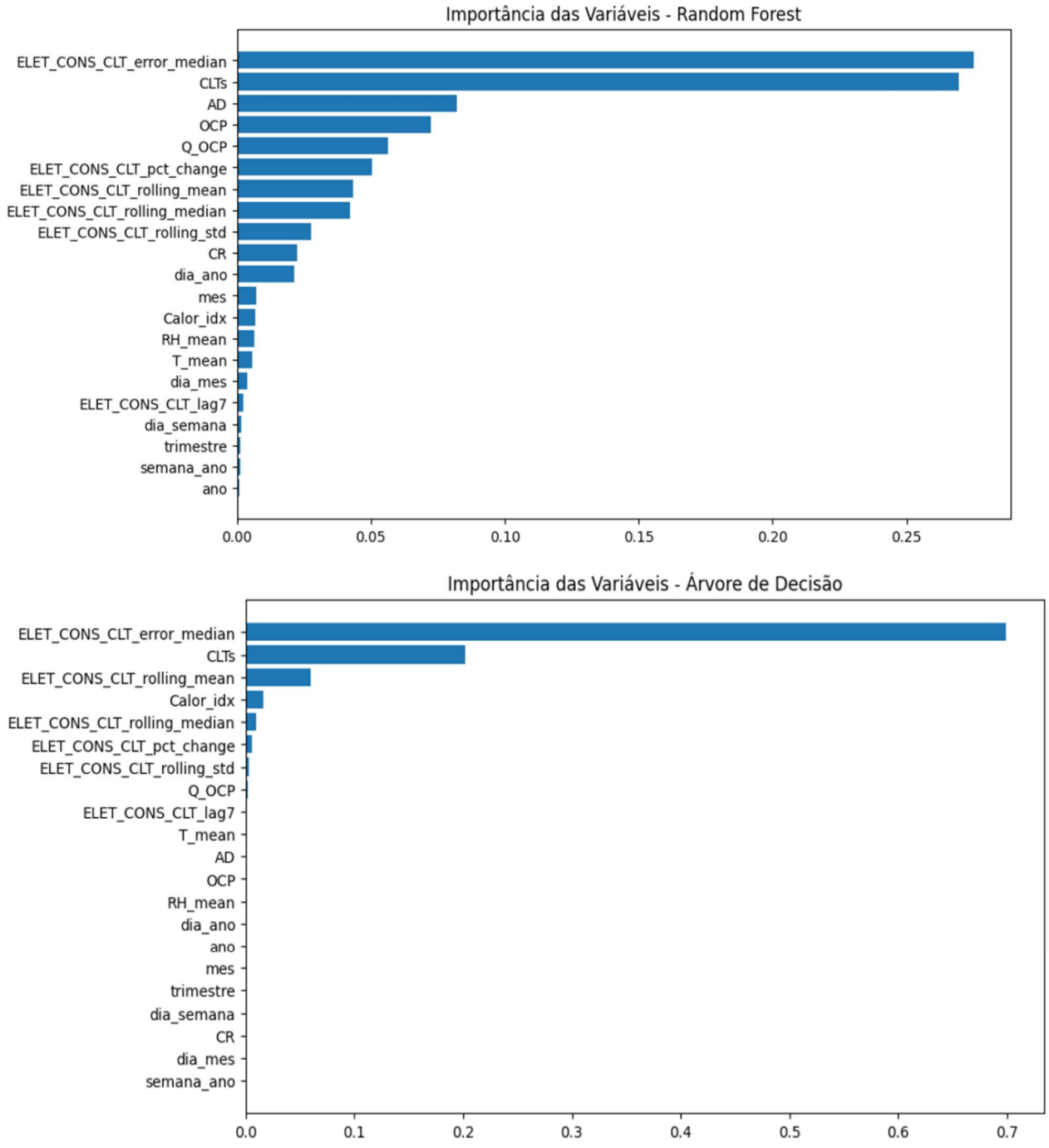


Figura 4.26 - Importância das variáveis de input consideradas, comparação entre os dois modelos na 2ª interação

4.7.3.2 Support Vector Regression (SVR) e Xtreme Gradient Boost (XGB)

O *Support Vector Regression* é uma adaptação do *Support Vector Machine* para problemas de regressão, procurando encontrar uma função que se afaste o mínimo possível dos valores reais, dentro de uma margem de tolerância. É particularmente eficaz quando a relação entre variáveis é não linear, podendo modelar, por exemplo, o consumo energético em função de padrões complexos de ocupação e climatização. O *Xtreme Gradient Boost* (XGB) é um algoritmo de boosting otimizado que constrói modelos de forma sequencial, onde cada novo modelo corrige os erros do anterior. A sua elevada capacidade de ajuste e eficiência computacional tornam-no adequado para prever falhas ou anomalias operacionais, integrando múltiplos fatores como sazonalidade, condições meteorológicas e padrões de utilização.

Com o objetivo de identificar o melhor modelo treinado (Tabela 4.4). Foram conduzidas quatro iterações de treino para cada modelo, sendo que as duas primeiras serviram para avaliar o desempenho em função dos inputs. O histórico de dados utilizado correspondeu aos períodos de 01/04/2023 a 31/10/2024.

As terceira e quarta iterações foram treinadas exclusivamente com base no ano de referência (2024), com o intuito de simular um benchmarking. Assim, o treino foi realizado no intervalo de 01/04/2024 a 31/10/2024, de modo a que a variável preditiva representasse um comportamento de referência, estabelecendo um objetivo a cumprir. Os inputs da terceira e quarta iteração acompanharam, respetivamente, a sequência da segunda e da primeira iteração.

1ª Iteração:

Inputs = Temperatura média diária, Humidade Relativa média diária, Índice de calor, nº total de clientes, nº de quartos ocupados, nº de adultos, nº de crianças, dia da semana, trimestre, mês, ano, dia do ano, dia do mês e semana do ano.

Output = Consumo diário de energia por cliente.

2ª Iteração: Foram mantidos os mesmos inputs da primeira iteração, acrescidos de variáveis derivadas criadas a partir dos dados originais. Estas variáveis adicionais capturam dependências temporais, volatilidades e tendências, suavizando flutuações e melhorando o desempenho dos modelos.

Inputs (adicionais) = Consumo diário de energia por cliente do dia anterior, da semana anterior, média móvel de 7 dias, mediana móvel de 7 dias, desvio padrão de 7 dias e erro relativo diário do consumo.

Output = Consumo diário de energia por cliente.

Considerando a Tabela 4.4 para se verificar as métricas de desempenho, sabe-se que em ambos os modelos (SVR e XGB), tanto no conjunto de treino como no de teste, as métricas de desempenho apresentaram, novamente, valores mais baixos quando adicionadas as variáveis derivadas (iterações 1 e 2), confirmando a sua relevância para a melhoria da previsão.

No entanto, com o objetivo de prever com base em comportamentos de referência, as iterações 3 e 4 permitiram quantificar a magnitude do erro entre os valores reais de 2025 e as previsões baseadas no comportamento de 2024. Isto é, já se verificou nos valores globais que as reduções foram importantes e expressivas em 2024, então admite-se que num formato de monitorização em tempo de real pode-se usar os algoritmos que melhor retratam o desvio entre os valores reais e os valores que se esperava ver, caso o comportamento for similar ao do ano de referência.

Tabela 4.3 - Quadro de metricas de avaliação de desenpenho dos modelos SVR e XGB, ordenadas por iteração e conjunto.

Modelo	Iteração	Conjunto	MSE	RMSE	MAE	R^2
SVR	1	Treino	5.3172	2.31	0.5493	0.76
SVR	1	Teste	0.7506	0.87	0.6945	0.70
SVR	2	Treino	6.1005	2.47	0.2629	0.72
SVR	2	Teste	0.0836	0.29	0.2376	0.97
SVR	3	Treino	0.0155	0.12	0.0900	1.00
SVR	3	Teste	0.1855	0.43	0.3345	0.92
SVR	4	Treino	0.4586	0.68	0.2897	0.92
SVR	4	Teste	0.9126	0.96	0.7488	0.62
XGB	1	Treino	1.5425	1.24	0.7065	0.92
XGB	1	Teste	0.7387	0.86	0.6388	0.69
XGB	2	Treino	0.0244	0.16	0.1082	1.00
XGB	2	Teste	0.0884	0.30	0.1921	0.96
XGB	3	Treino	0.0237	0.15	0.1113	1.00
XGB	3	Teste	0.0876	0.30	0.1909	0.97
XGB	4	Treino	0.3524	0.59	0.3203	0.94
XGB	4	Teste	1.1194	1.06	0.8464	0.53
XGB	5	Treino	0.0777	0.28	0.1577	0.99
XGB	5	Teste	0.2458	0.50	0.3545	0.90

As Figuras 4.26 e 4.27 representam as previsões realizadas com os modelos que melhor desempenho com as variáveis de entrada correspondentes apenas às variáveis independentes, portanto a ocupação, as variáveis climáticas e a variáveis time series, treino com dados do ano de referencia (2024). Esta metodologia, visa simular dados com características do comportamento de 2024, por forma que a segunda abordagem seja traçar limites nos valores do erro relativo, entre previsão e valor observado, e e desenvolver procedimento que fala convergir essas duas variaveis. Ao consultar as figuras percebe-se que os valores simulados posicionaram-se abaixo dos valores reais tanto em 2023 (Figura 4.27) com em 2025 (Figura 4.26) expressando mais os valores reduzidos conseguidos em 2024.

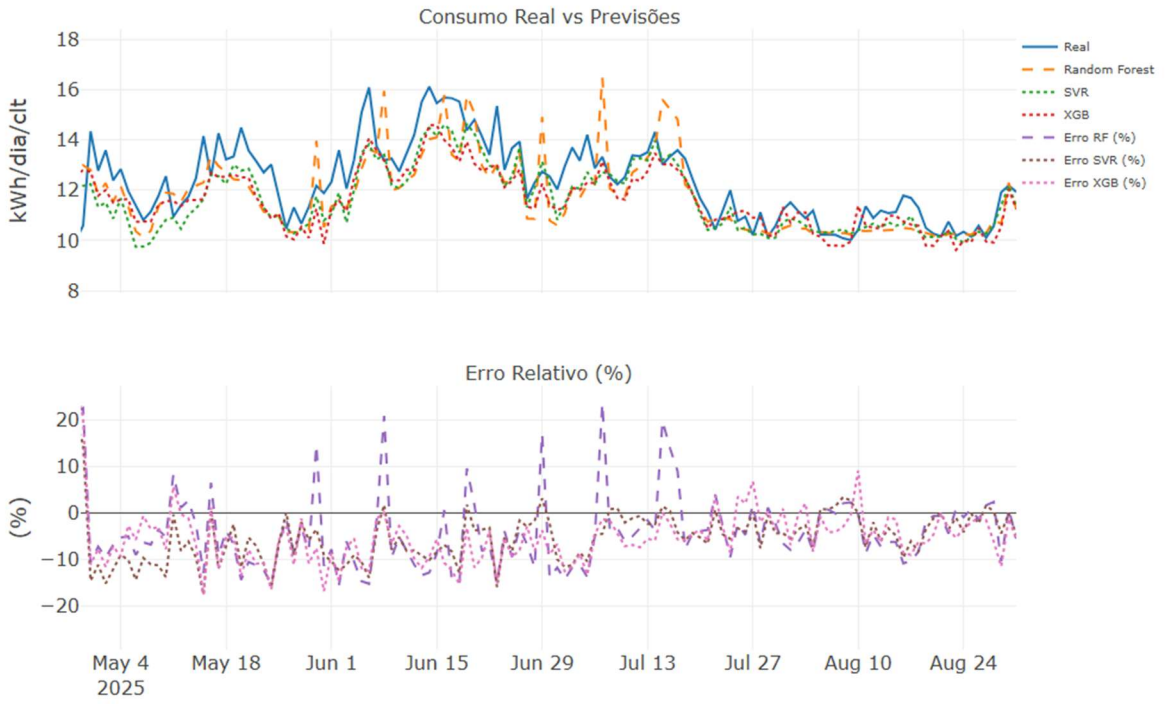


Figura 4.27 – Comparação de modelos treinados com base no histórico de dados dos período de operação (abril-outubro) de 2024. (correspondentes à 4ª interação de treino dos respetivos modelos), serie temporal de 2025.

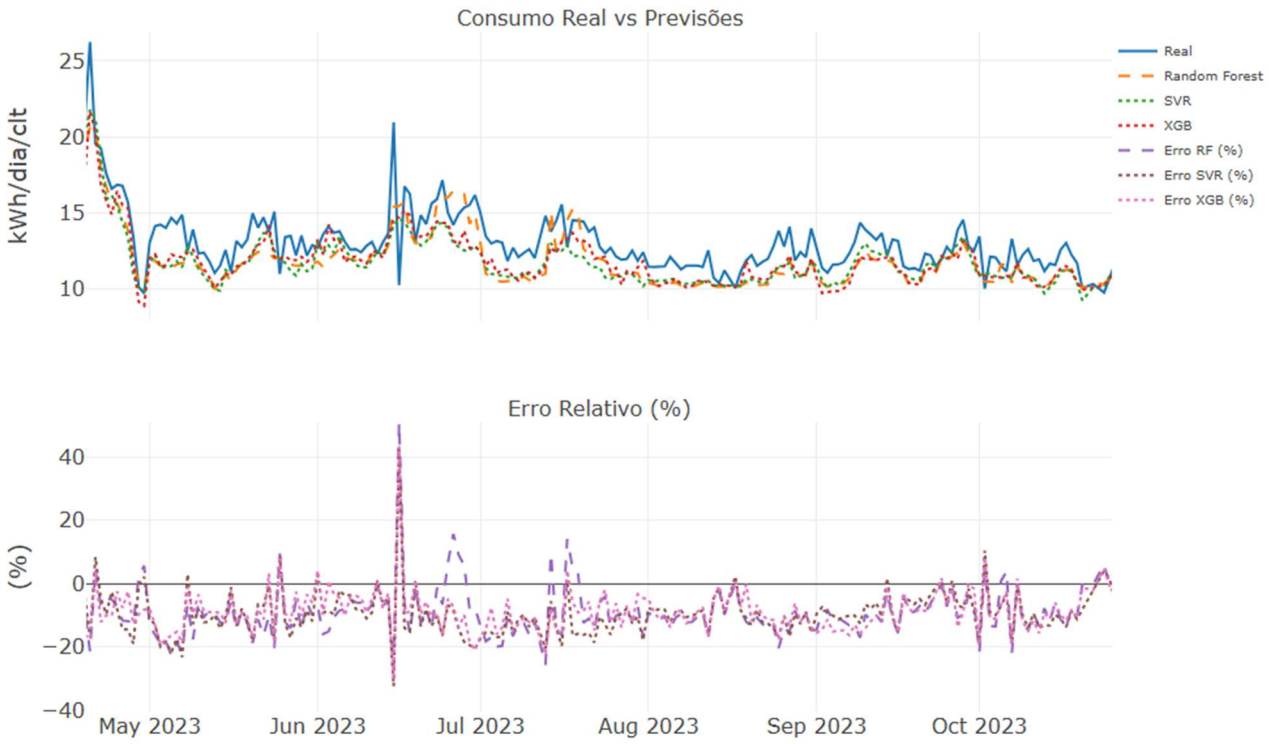


Figura 4.28 - Comparação de modelos treinados com base no histórico de dados dos períodos de operação (abril-outubro) de 2024. (correspondentes à 4ª interação de treino dos respetivos modelos), serie temporal de 2023

4.7.4 REDES NEURONAIS ARTIFICIAS

As redes neuronais artificiais (RNA) são modelos computacionais inspirados no funcionamento do cérebro humano. As RNA são compostas por unidades chamadas neurónios artificiais, organizadas em camadas (entrada, ocultas e saída), que processam informação de forma interligada.

As camadas definem como a informação entra, é transformada e sai de uma rede neuronal. Cada camada aplica operações matemáticas aos dados e passa as representações resultantes para a próxima, aprendendo pesos que minimizam um erro definido pela tarefa. Portanto, a camada de entrada é responsável por receber os dados no formato numérico (vetores, tensores, *embeddings*). A dimensão desta camada corresponde ao número de variáveis de entrada.

As camadas ocultas executam transformações não lineares sobre os dados, extraindo padrões e representações progressivamente mais abstratas. Cada neurónio aplica uma função de ativação sobre a soma ponderada das entradas, permitindo à rede modelar relações complexas.

No processo de treino, o conceito de *epoch* corresponde a uma passagem completa por todo o conjunto de dados, permitindo que a rede ajuste os seus pesos de forma iterativa até convergir para uma solução estável. Já o *batch size* (amostra) define quantos exemplos são processados de cada vez antes de atualizar os parâmetros do modelo: tamanhos menores introduzem mais

ruído, mas podem favorecer a generalização, enquanto tamanhos maiores tornam o treino mais estável, embora menos sensível a variações locais. A camada de saída transforma a última representação em predições específicas da tarefa (classificação, regressão, geração de sequências).[44] De forma simples: Cada neurónio recebe dados, aplica um cálculo matemático (função de ativação) e transmite o resultado para os neurónios seguintes. A rede aprende ajustando os pesos das conexões entre neurónios, de modo a melhorar a precisão das previsões ou classificações.

Apesar dos avanços significativos alcançados pelas redes neuronais artificiais, estas apresentam ainda um conjunto de limitações que importa considerar de forma crítica. Em primeiro lugar, o tempo de processamento tende a aumentar de forma exponencial à medida que cresce a complexidade do problema ou a dimensão dos dados de entrada. Este fenómeno decorre da elevada carga computacional associada ao treino de modelos profundos, o que pode comprometer a sua aplicabilidade em contextos que exigem respostas em tempo real ou em sistemas com recursos limitados. Em segundo lugar, o desempenho das redes neuronais é fortemente dependente da qualidade e do tipo de pré-processamento dos dados. Dados ruidosos, incompletos ou mal estruturados podem conduzir a resultados inconsistentes, enviesados ou de baixa robustez. Assim, a preparação criteriosa dos dados constitui uma etapa crítica para assegurar a fiabilidade dos modelos.

Por fim, uma das limitações mais discutidas na literatura é a falta de interpretabilidade dos resultados. As redes neuronais são frequentemente descritas como “caixas negras”, uma vez que as regras de operação internas e os mecanismos de decisão não são facilmente compreensíveis pelos utilizadores. Esta opacidade levanta desafios relevantes em termos de transparência, confiança e aceitação em domínios críticos, como a saúde, a justiça ou a gestão de recursos.[45]

4.7.4.1 Gated Recurrent Unit (GRU) e Long Short-Term Memory (LSTM)

As arquiteturas GRU e LSTM são redes neurais recorrentes projetadas para lidar com dependências temporais de longo prazo em séries temporais. No contexto hoteleiro, podem ser aplicadas para prever consumo energético ou desempenho de sistemas de climatização com base em históricos de dados, captando padrões sazonais e variações diárias. A GRU apresenta uma estrutura mais simples e computacionalmente eficiente, enquanto a LSTM possui mecanismos adicionais de controlo de memória, permitindo modelar relações temporais mais

complexas. Ambas são adequadas para dados sequenciais provenientes de sensores e sistemas de monitorização contínua.

Para aplicar ao caso de estudo, seguiu-se o mesmo princípio conforme no modelo de ML apresentadas. Foi realizado um grupo de iterações onde foram treinados os modelos com a adição de variáveis derivadas e o ajuste fino de hiperparâmetros por forma a melhorar o desempenho das GRU, reduzindo erros e aumentando a capacidade preditiva face às iterações iniciais.

Nos modelos GRU, os hiperparâmetros serviram como ajustes prévios que determinam a forma como a rede aprende a partir dos dados. Entre os mais relevantes está o número de unidades GRU em cada camada, que define a capacidade do modelo de capturar padrões temporais: quanto maior o número de unidades, maior a complexidade que pode ser aprendida, mas também maior o risco de *overfitting*. No presente estudo apenas foram usadas 100 ou 200 unidades (ou neurónios) na camada oculta.

Outro parâmetro utilizado foi a conhecida taxa de *dropout* de 20%, sendo este um mecanismo de regularização, desligando aleatoriamente algumas conexões durante o treino para evitar que o modelo memorize os dados em vez de generalizar. Em todas as iterações de treino foram usadas 100 épocas e 16 amostras. De forma geral, o objetivo é interpretar como as variáveis de entrada (inputs) nos diferentes treinos, assim como o tamanho do grupo de dados, tem impacto na sua performance assim com o tamanho do grupo de dados. No caso específico de séries temporais, como o caso em estudo, destaca-se ainda o número de passos temporais, que corresponde à quantidade de dias anteriores considerados pelo modelo para prever o valor seguinte.

Os resultados das iterações com o modelo GRU mostraram uma evolução interessante (Figura 4.28), considerando que a diferença desses modelos treinados foi, maioritariamente os dados de input e o número de passo temporais.

Nas 3 primeiras iterações, o desempenho em treino é consistente, com R^2 entre 0,92 e 0,93, e os erros (MSE, RMSE, MAE) vão diminuindo ligeiramente. No entanto, no conjunto de teste o R^2 mantém-se mais baixo (0,56–0,63), revelando que o modelo ainda não generaliza tão bem quanto seria desejável. Na 1ª e 2ª iteração os inputs são o conjunto das variáveis climáticas, ocupação e as variáveis adicionais, contudo existe a diferença do número de passos temporais para prever o dia seguinte. Respetivamente 7 dias e 15 dias, para a 1ª e 2ª iteração. Os dados para treino foi todo o conjunto de dados recolhidos de 01/01/2023 a 31/08/2025.

Considerando que, no contexto do *deep learning*, é fundamental disponibilizar ao modelo a maior quantidade possível de informação relevante, na 3ª iteração foram incluídas variáveis

dos consumos parciais de energia elétrica. Para além destas variáveis operacionais, foram igualmente considerados indicadores ambientais, nomeadamente as emissões de dióxido de carbono associadas à eletricidade, as emissões totais de dióxido de carbono equivalente e as emissões de dióxido de carbono equivalente por quarto ocupado (subcapítulo 4.1). Esta integração destas variáveis, visa reforçar que o modelo possa captar de forma mais abrangente os fatores que influenciam o consumo energético por cliente, reforçando a capacidade preditiva e a utilidade prática da análise.

Em termos de resultados, verificou-se uma otimização de 8% no conjunto de teste e 12% no conjunto de treino, relativamente ao RMSE.

A 4ª iteração é a combinação da 3ª, contudo os dados de treino foram comprometidos ao período de ocupação do ano de 2024. Nos resultados há uma queda clara no desempenho de treino ($R^2 = 0,77$), com aumento do erro, e no teste o R^2 cai para 0,52.

Na tentativa de melhorar o modelo, na sequência da antecessora, na 5ª iteração foram realizados ajustes nos parâmetros de treino, nomeadamente com 200 camadas de unidades na cada oculta (diferente das iterações, 100 camadas), assim como o acréscimo de 32 para 64 amostras (*batch size*) e foi utilizado 200 passagens completas no grupo de dados. Neste caso o modelo atingiu o melhor resultado até então (MSE = 0,32; $R^2 = 0,93$), mostrando que conseguiu aprender muito bem os padrões históricos. Contudo, no teste o desempenho degrada-se fortemente ($R^2 = 0,15$; RMSE = 1,41), evidenciando o *overfitting*, portanto o modelo memorizou o treino, mas perdeu capacidade de generalização.

Na Figura 4.29 representa-se graficamente os dois melhores resultados do modelo, a 3ª e 4ª iteração. Verifica-se que os dados de previsão acompanham de forma bastante próxima a linha do consumo real, sobretudo nas oscilações do erro relativo, consideravelmente moderadas.

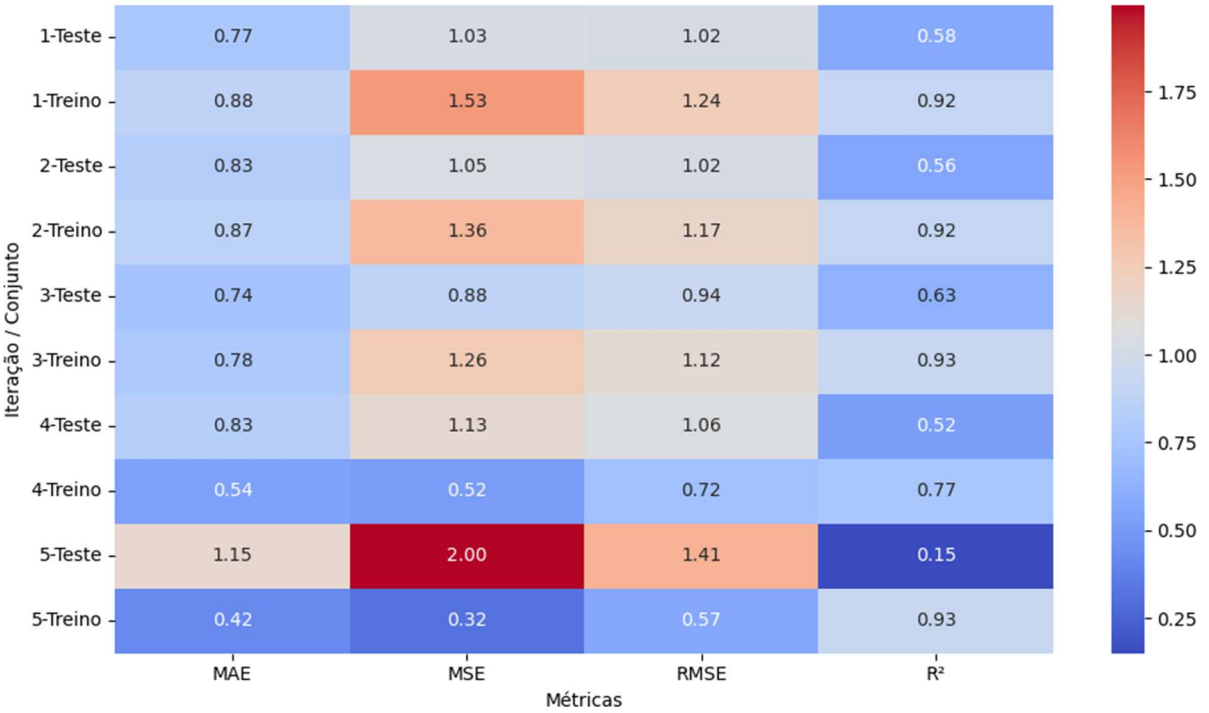


Figura 4.29 – Mapa de calor das métricas de avaliação do modelo GRU, ordenadas por iteração e conjunto

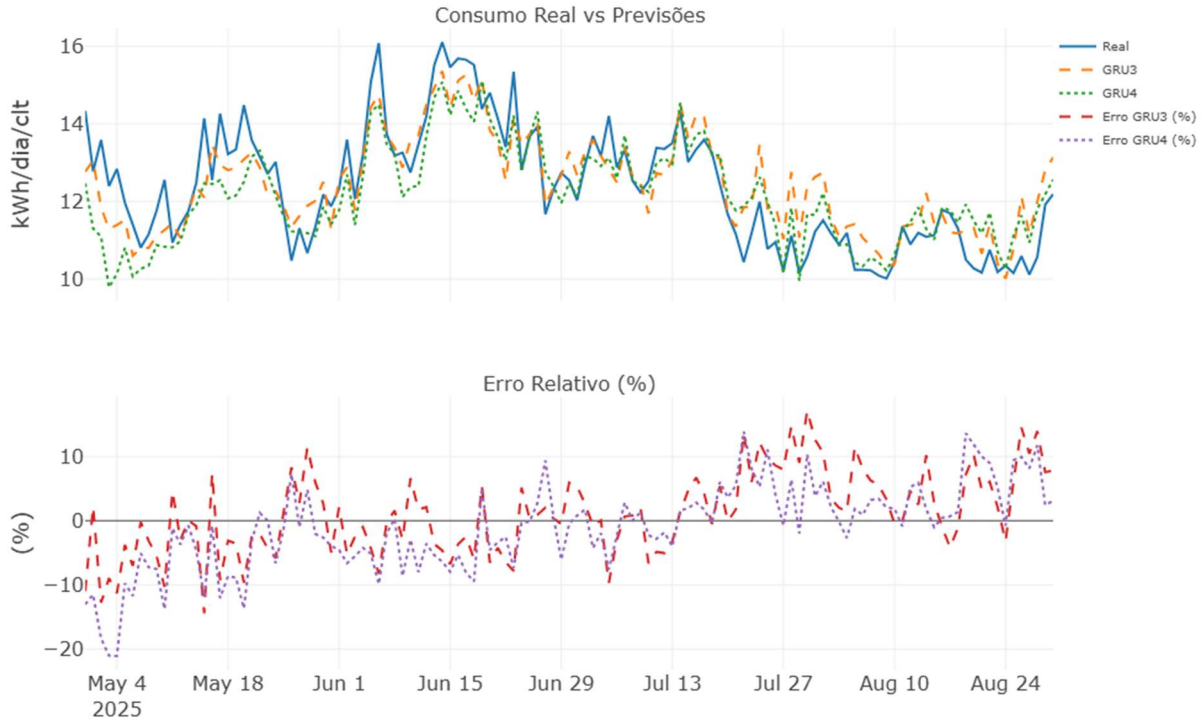


Figura 4.30 – Comparação das previsões da 3ª e 4º iteração, no conjunto teste, do modelo GRU

4.7.4.2 Convolutional Neural Networks (CNN)

As CNNs são redes neurais profundas constituem uma das arquiteturas mais relevantes no campo do *deep learning*, amplamente utilizadas para a extração automática de padrões complexos em dados estruturados e não estruturados. Originalmente desenvolvidas para o reconhecimento de imagens, as CNN destacam-se pela capacidade de identificar características locais e hierárquicas através da aplicação de filtros convolucionais, que captam desde padrões simples até representações mais abstratas. Esta propriedade torna-as particularmente eficazes em problemas que envolvem grandes volumes de informação e relações espaciais ou temporais, como séries temporais de consumo energético. Ao combinar camadas convolucionais, funções de ativação não lineares, camadas de *pooling* e mecanismos de regularização, as CNN conseguem aprender representações robustas e generalizáveis, oferecendo um elevado potencial para tarefas de previsão, classificação e deteção de anomalias em contextos complexos[46]. Embora amplamente usadas em visão computacional, também podem ser aplicadas a séries temporais convertidas em representações bidimensionais, permitindo identificar padrões espaciais e temporais no comportamento energético de diferentes áreas do hotel.

Para aplicação ao caso de estudo, seguiu-se o mesmo princípio adotado nos modelos de ML e GRU, realizando-se um conjunto de iterações com a rede CNN (Convolutional Neural Network). O objetivo foi avaliar o impacto da adição de variáveis derivadas e do ajuste fino de hiperparâmetros no desempenho preditivo, reduzindo erros e aumentando a capacidade de generalização face às iterações iniciais.

Nos modelos CNN, os hiperparâmetros assumem igualmente um papel determinante, destacando-se o número de filtros convolucionais, o tamanho do *kernel* e a profundidade das camadas, que influenciam diretamente a capacidade da rede em extrair padrões relevantes das séries temporais. Tal como nas GRU, foi utilizada uma taxa de dropout de 20% como mecanismo de regularização, bem como 100 épocas de treino e batch size de 16 amostras, assegurando consistência metodológica entre os diferentes modelos testados. Contudo na primeira iteração as métricas apresentaram consideravelmente fracas. Os resultados das iterações com as CNN encontram-se sintetizados no mapa de calor das métricas por iteração e conjunto (Figura 4.30).

Na 2ª iteração o modelo melhorou a sua performance devido aos ajustes nos hiperparâmetros. Na 1ª iteração usou-se 128 filtros com kernel de tamanho 5 e uma camada densa de 256 neurônios, ativação ReLU e dropout de 20%, o que confere maior capacidade de captura de padrões locais, mas aumenta o risco de overfitting. Na 2ª iteração, com 64 filtros e kernel

maior (7), ativação GeLU, camada densa menor (64 neurônios) e dropout mais suave de 10%. Também foi aumentado o nº de passagens para 200 com 64 amostras. Nos resultados, mais favoráveis, com valores de R^2 próximos de 0,95 e erros relativamente baixos (MAE, MSE e RMSE). Contudo, no conjunto de teste, os valores de R^2 mantêm-se mais modestos 0,72 revelando que a rede ainda não generaliza de forma totalmente satisfatória.

Nas 1ª e 2ª iterações, as variáveis de entrada incluíram dados climáticos e de ocupação, além de das variáveis de series temporais. Na 3ª iteração, foram adicionadas variáveis derivadas do consumo elétrico com as variáveis adicionais referentes às estatísticas. Na 4ª, adicionou-se os consumos parciais por setor e métricas ambientais, por forma a ampliar a granularidade e a capacidade explicativa da previsão. Todas as iterações usaram o conjunto dados completo referente ao período 2023-2025.

Na 4ª iteração, observa-se o melhor desempenho geral: RMSE de 0,71 no teste, MAE de 0,50 e R^2 de 0.80, com excelente ajuste no treino (R^2 = 0.96). Isso reflete o impacto positivo da inclusão dos consumos parciais por setor e métricas ambientais, que ampliaram a granularidade e a capacidade explicativa do modelo.

Já a 5ª iteração, embora tenha mantido os mesmos inputs das primeiras versões, apresentou melhorias moderadas devido ao ajuste de hiperparâmetros, o R^2 no treino subiu para 0.94, mas o teste caiu para 0,71, indicando possível sobreajuste. A 6ª iteração, com ajustes adicionais, conseguiu reduzir o erro no teste (RMSE = 0,4596) e manter boa generalização (R^2 = 0,81), embora com leve perda de precisão no treino.

Em síntese, a 4ª iteração se destaca como a mais equilibrada e explicativa, especialmente para aplicações com múltiplas fontes de consumo e indicadores ambientais.

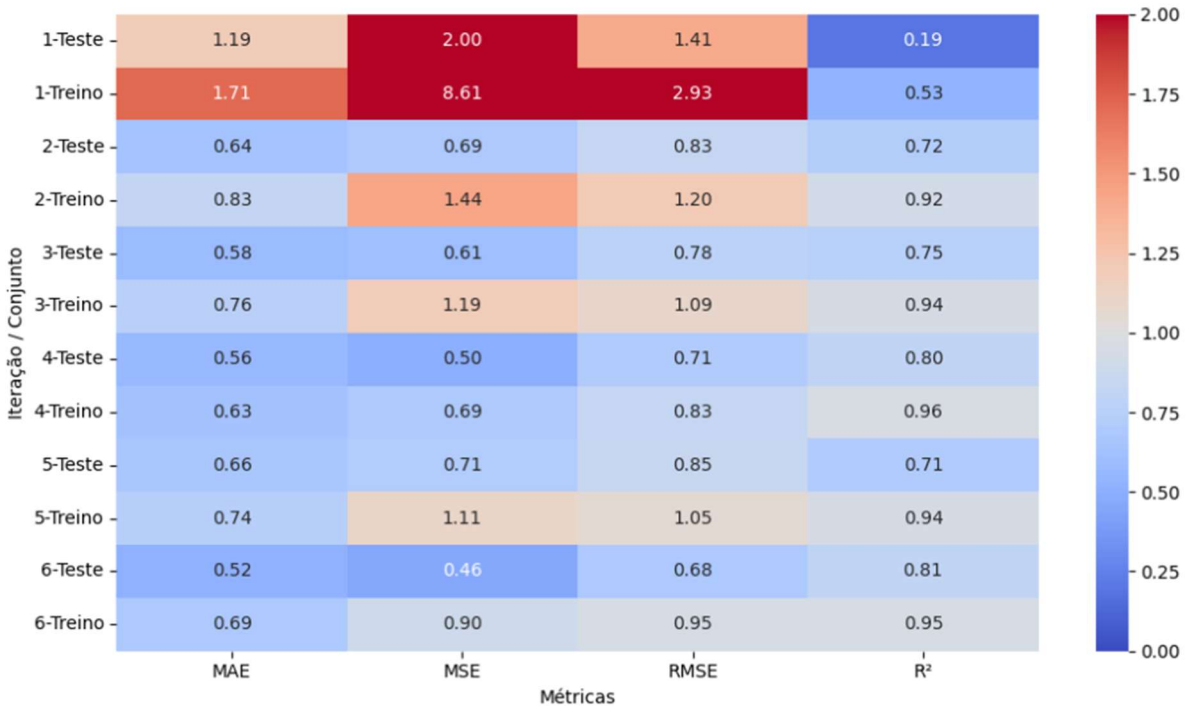


Figura 4.31 - Mapa de calor das métricas de avaliação do modelo CNN, ordenadas por iteração e conjunto

A Figura 4.31 apresenta uma comparação em relação às arquiteturas CNN e GRU no contexto da previsão de consumo energético por cliente. No gráfico superior, CNN4 e CNN6, apresentam desempenho competitivo e, em certos períodos, até superior em termos de estabilidade do erro relativo.

No gráfico inferior, onde se apresenta os modelos CNN mantêm os valores de erro consistentemente mais baixos ao longo do período analisado, com menor oscilação e maior estabilidade. Conclui-se que as CNNs, quando bem parametrizadas e alimentadas com variáveis contextuais e setoriais, como na iteração nº 4, oferecem excelente desempenho preditivo, com boa estabilidade e aderência ao consumo real. Isso reforça sua aplicabilidade em sistemas de monitorização energética com elevada granularidade e necessidade de resposta rápida a variações operacionais.

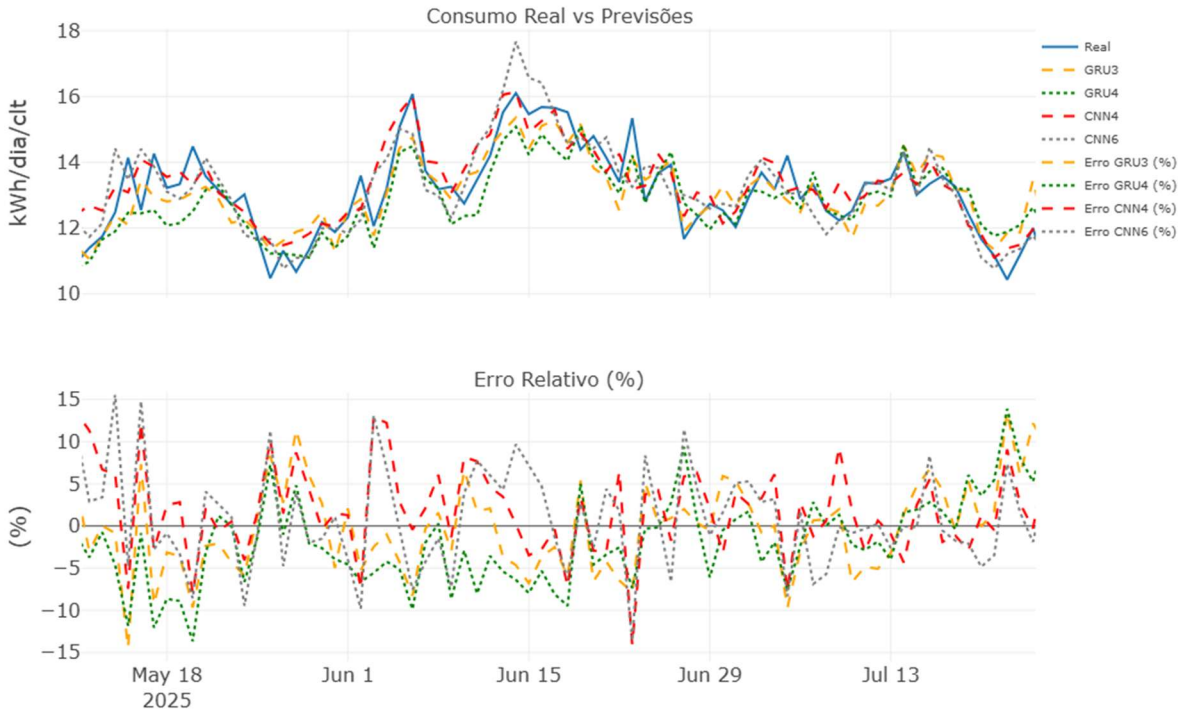


Figura 4.32 - Comparação abrangente entre os modelos GRU e CNN, considerando as iterações 3 e 4 do GRU e 4 e6 do CNN.

4.7.5 RESULTADOS

Com o objetivo de integrar ferramentas auxiliares ao sistema de monitorização e de gestão técnica centralizada, foram considerados modelos preditivos e relações estatísticas com valores de referência. Conforme apresentado no subcapítulo 4.3.1, as previsões adotadas, exemplificadas na Figura 4.21, permitem construir gráficos que incorporam o objetivo de delimitar os erros relativos entre os valores previstos e observados.

Neste contexto, a definição de valores-limite para o erro relativo constitui uma etapa crítica para transformar previsões em ações preditivas. Deve-se considerar o histórico de desempenho dos modelos, como as distribuições de erro nas iterações de teste realizados, também qual a sensibilidade operacional considerando o impacto das variações de consumo. A tolerância de falsos alarmes deves ser considerada, isto é, limites muitos baixos podem gerar alarmes excessivos, contudo também existem sistemas onde pequenas variações de consumo podem ter grande impacto e, portanto, erros relativos com os valores esperados devem ser mais restritivos.

Idealmente e quando disponíveis deve-se utilizar *benchmarks* ou padrões técnicos registados para ajustar os limites. Estes métodos devem ser garantidos por testes de validação operacional. A definição de valores-limite para erros relativos não deve ser estática.

Recomenda-se a reavaliação periódica destes parâmetros, incorporando novos dados e ajustando-os à evolução do desempenho dos modelos e das condições operacionais, garantido assim, um sistema de monitorização deixa de ser apenas reativo e passa a ter um papel proativo na gestão energética. A utilização de gráficos de controlo, aumenta a sensibilidade da monitorização, permitindo não só detetar anomalias em tempo real, mas também avaliar tendências de melhoria ou degradação do desempenho energético ao longo do tempo.

No seguimento do trabalho realizado e como resultado final, apresenta-se a Figura 4.32 com objetivo de espelhar um *dashboard* aplicável à monitorização continua recursos energéticos e hídricos. O trabalho desenvolvido na componente de avaliação de dos valores globais, da interpretação do seu comportamento ao longo de determinados momentos de operação, assim como os valores estatísticos considerados permitem apresentar soluções que simulem ou prevejam o comportamento de uma determinada variável dependente.

Neste caso e dando continuação à figura 4.21, a integração dos modelos preditivos mantem a pertinência que os valores registados entre 6 e 20 de julho foram consideravelmente superiores ao esperado, face à referência do período homologo de 2024. Neste período verifica-se que os erros relativos ultrapassam os -10%, de forma sistemática ao longo desse período de 14 dias, explicando assim um aumento de consumos inesperados, que por sua vez derivam para um indicador operacional, o consumo diário por cliente, também ele superior cerca de 10%. Nesta fase final e com o objetivo de interpretar a de delimitar valores de alarme a análise das anomalias numa janela de 30 dias é particularmente relevante porque introduz uma perspetiva temporal que permite distinguir desvios pontuais de padrões persistentes. Em vez de avaliar apenas erros diários isolados, esta metodologia acumula informação recente e evidencia tendências anómalas que podem estar associadas a problemas técnicos ou operacionais. Não só, também permite desenvolver know-how sobre quais os limites aceitáveis para que não se constitua falsos alarmes.

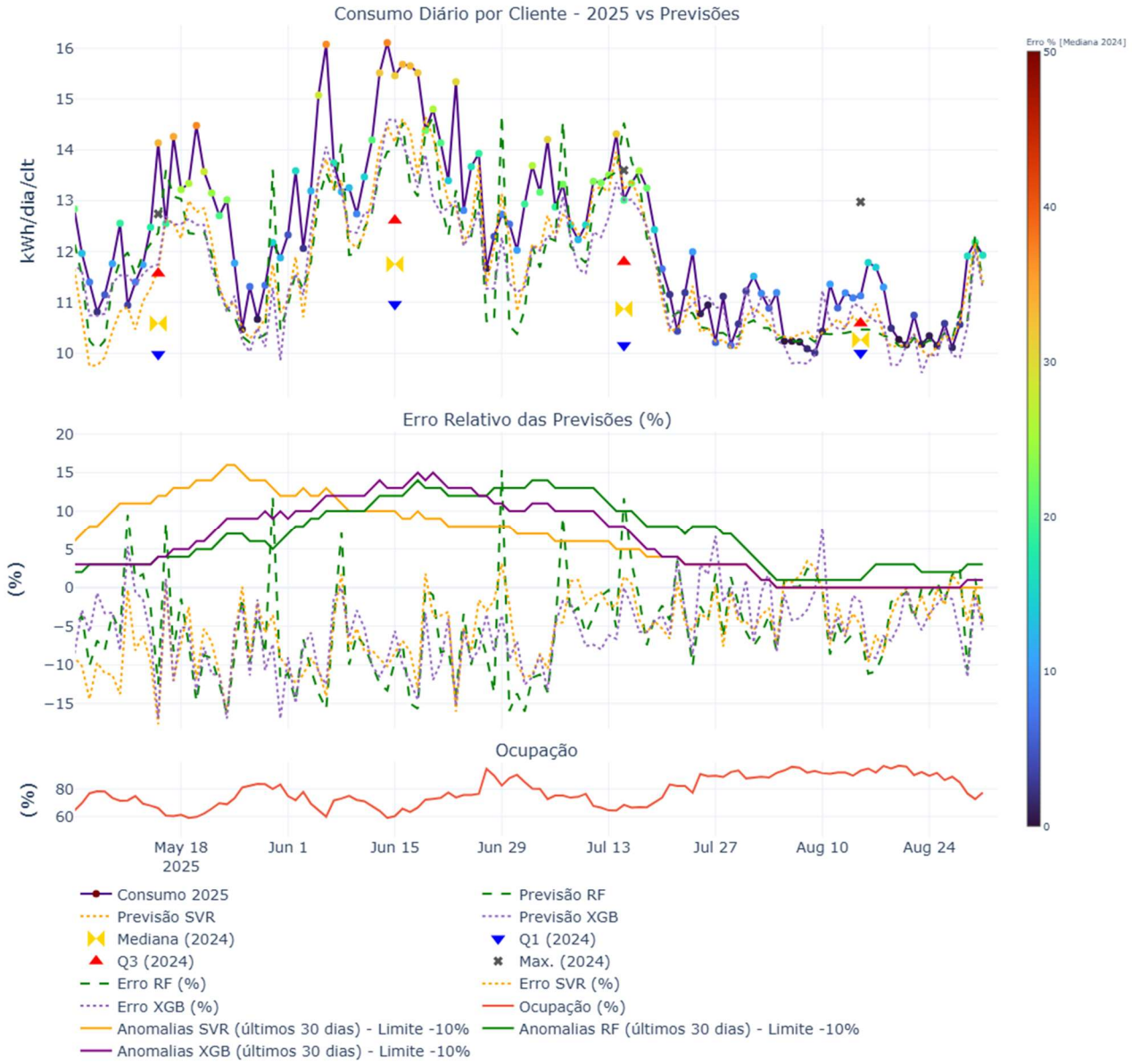


Figura 4.33 – Gráficos de controlo com previsões de ML, anomalias contabilizadas em 30 dias e erro relativo de variáveis estatísticas.

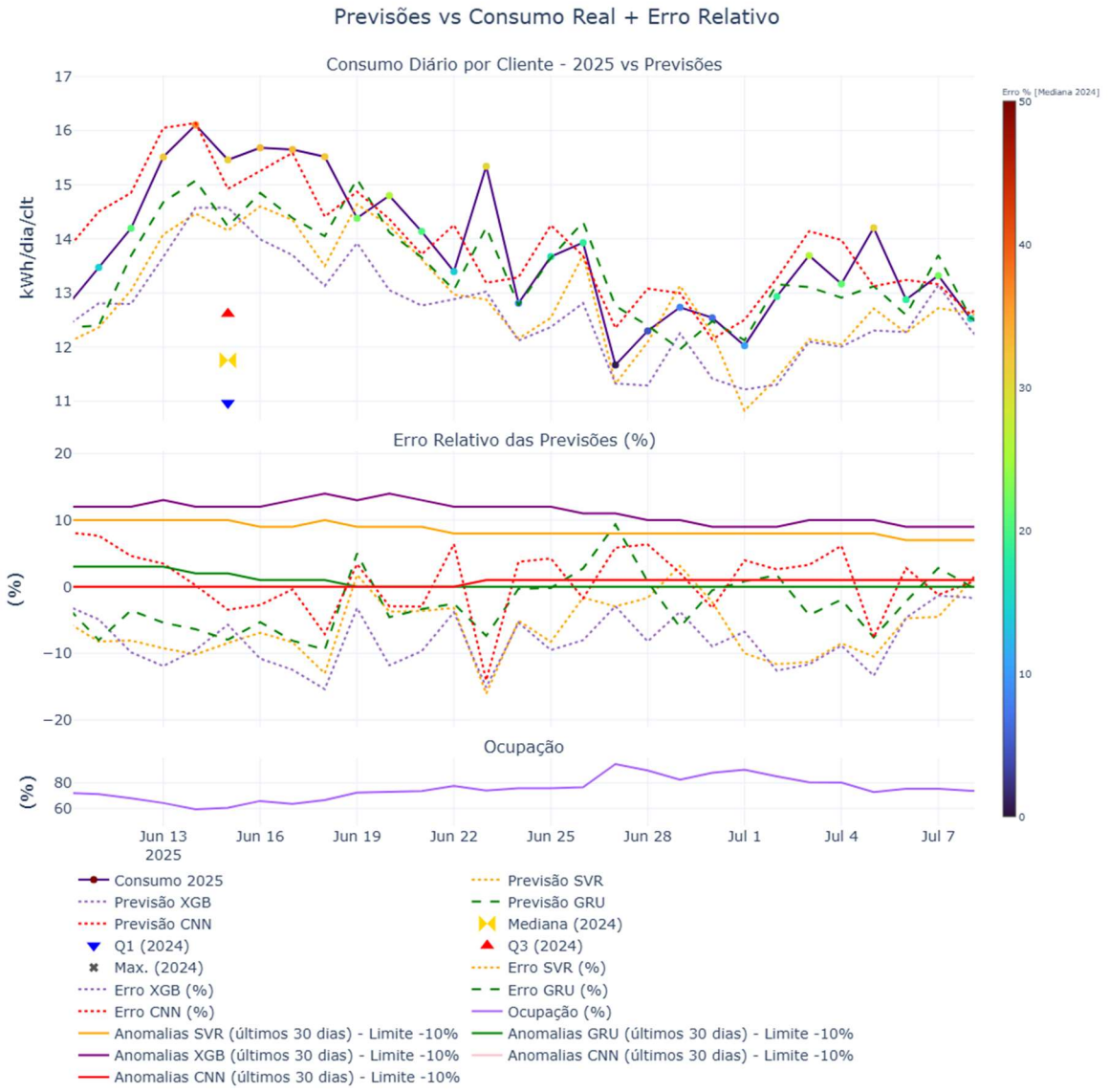


Figura 4.34 - Gráficos de controlo com previsões de ML e DL, com anomalias contabilizadas em 30 dias e erro relativo de variáveis estatísticas.

De forma conclusiva, a utilização da IA no contexto da gestão energética e hídrica em unidades hoteleiras representa um salto qualitativo na forma como os recursos são monitorizados, analisados e otimizados. Conforme se desenvolveu nesta dissertação a IA permite transformar grandes volumes de dados provenientes de consumos energéticos, padrões de ocupação, variáveis climáticas e indicadores ambientais, em conhecimento acionável, capaz de sustentar decisões técnicas e estratégicas.

Conforme representado na Figura 4.33, pode se considerar uma análise conjunta de vários modelos de previsão. Cada um deste com generalização especifica de um determinado grupo de dados treinado.

Neste contexto recomenda-se que, como o exemplo, pode-se manter redes neuronais artificiais a apreender constantemente ao longo dos últimos 7 ou 15 dias, enquanto modelos de ML treinados com objetivos específicos para interpretarem comportamentos de referência ou objetivos/metas a atingir.

Pode-se verificar na Figura 4.33 enquanto os modelos de DL mantem um desvio bastante pequeno da realidade registada, os de ML apresentam anomalias superiores aos desvios negativo de 10% na ordem de 14 ao longo de 30 dias.

A importância deste tipo de abordagem é permitir estruturar processos de gestão e manutenção de forma mais eficaz, uma vez que facilita a identificação das anomalias e as suas respetivas fontes. Assim, quando o consumo diário por cliente ultrapassa os limites previamente definidos como aceitáveis, não basta sinalizar o desvio: é necessário analisar também as restantes variáveis que influenciam esse indicador (como ocupação, condições climáticas e principalmente os setores que representam os consumos parciais da unidade hoteleira. Dessa forma, a intervenção de manutenção torna-se mais direcionada, objetiva e eficiente, evitando ações genéricas e permitindo atuar diretamente sobre a causa do problema.

Ao integrar modelos preditivos e gráficos de controlo a sistemas de GTC ou de gestão energética, metodologias baseadas em IA não se limitam a estimar consumos futuros, mas sim a criar mecanismos de deteção precoce de anomalias, definição dinâmica de limites de tolerância e correlação com fatores operacionais. Isto traduz-se numa gestão mais sensível e proativa, onde desvios de consumo podem ser rapidamente associados a falhas técnicas, degradação de equipamentos ou ineficiências nos processos.

4.7.6 INTEGRAÇÃO COM O DEPARTAMENTO DE OPERAÇÕES DE MANUTENÇÃO

A integração eficaz com o departamento de manutenção deve considerar a otimização tecnológica, a gestão inteligente de recursos e a implementação dos conceitos da I4.0. O foco principal é utilizar dados operacionais e históricos para suportar decisões estratégicas e proativas, garantindo a eficiência energética e a sustentabilidade operacional.

Com base no estudo desenvolvido, identificaram-se os principais aspetos a considerar na integração entre os sistemas de monitorização inteligente e o departamento de operações de manutenção. Estes elementos constituem fatores determinantes para assegurar a eficácia do diagnóstico operacional, a eficiência energética e a sustentabilidade da unidade hoteleira:

- Foco na manutenção preditiva e inteligente (manutenção 4.0): A integração deve apoiar a transição de práticas tradicionais para modelos de manutenção preditiva e

manutenção 4.0. Isto é impulsionado pela digitalização e pela interligação de sensores inteligentes.

- Disseminar conhecimento: O aspeto chave é a capacidade de criar novo conhecimento a partir do processamento de dados para desenvolver uma manutenção inteligente. Embora não seja escalável para grandes conjuntos de dados, a análise manual de casos individuais por especialistas é essencial para fornecer insights sobre os padrões associados a várias falhas, a sua distribuição temporal e o impacto nas variáveis de medição[41].
- Antecipar falhas: A utilização de tecnologias de IA e algoritmos de ML ou DL deve ser complementada pelas técnicas tradicionais de manutenção condicionada, como análises de vibrações, inspeções visuais e funcionais periódicas, analises termográficas, entre outras. Esta integração permite prever ocorrências de falha de forma mais robusta, combinando a capacidade preditiva dos modelos com a validação prática, obtida através da monitorização física dos equipamentos.
- Simulação e diagnóstico: A modelação e a simulação do comportamento dos sistemas, quando associadas a métodos clássicos de inspeção e análise de condição, constituem uma base sólida para o desenvolvimento de sistemas de produção inteligentes. Esta abordagem híbrida pode apoiar decisões mais objetivas sobre intervenções de manutenção, assegurando que os diagnósticos resultam tanto da análise de dados avançada como da observação direta do estado dos ativos.
- Redução de Custos: A implementação de técnicas de monitorização e planeamento dinâmico é crucial para a redução de custos de manutenção (estimada entre 20% a 30%).

Para que a manutenção seja eficiente, é necessário o acesso e o tratamento sistemático de dados, transformando-os em indicadores para a tomada de decisão. A monitorização deve incluir o registo rigoroso de manutenções preventivas e corretivas, registo de eventos especiais, substituição ou avaria de equipamentos e alterações operacionais relevantes. O histórico de dados, integrado em plataformas inteligentes (GTC e IoT), permite gerar alarmes dinâmicos e indicadores de condição, antecipando intervenções e otimizando o desempenho energético dos ativos. A análise de tendências de consumo transforma-se, assim, em indicadores de condição dos ativos, reforçando a necessidade de tecnologias que assegurem controlo e análise avançada

5

CONCLUSÕES E TRABALHOS FUTUROS

A análise exploratória e a modelação preditiva dos recursos energéticos e hídricos da unidade hoteleira em estudo permitiram caracterizar o desempenho atual, assim como delinear metodologias de diagnóstico operacional e de otimização com elevado potencial de aplicação prática.

Entre 2023 e 2024, verificaram-se melhorias significativas na eficiência operacional, traduzidas em reduções de 8,2% na mediana do consumo de energia elétrica e de 12,9% na mediana do consumo de água potável, apesar de um aumento de 4% na taxa de ocupação. Estas melhorias resultaram diretamente de intervenções técnicas, como a substituição de redutores de caudal, a modernização da iluminação e a aquisição de novos equipamentos de AVAC. A análise de séries temporais confirmou ainda o impacto da sazonalidade, sobretudo nos meses de verão, mas demonstrou que, mesmo em períodos de maior pressão, foi possível assegurar reduções médias de 9% no consumo elétrico e 4% no consumo de água.

As variáveis referentes à ocupação revelaram-se determinantes para explicar os consumos, apresentando correlações muito fortes com a energia elétrica (0,93) e a água (0,90). A interdependência entre consumos elétricos e hídricos (0,91) reforça a necessidade de abordagens integradas de gestão. Em termos de emissões de GEE, o gás propano destacou-se como o principal contribuinte (70 tCO_2e/mês), seguido da eletricidade (60 tCO_2e/mês). Ainda assim, o hotel demonstrou um desempenho ambiental consideravelmente competitivo, com uma pegada de carbono por quarto ocupado situada entre 10 e 13 kg CO_2e/dia, valores inferiores à média nacional reportada para o setor.

A análise global mostrou-se insuficiente para identificar ineficiências específicas, sendo a desagregação por setor essencial para o diagnóstico. No caso da água, a piscina principal registou um aumento anómalo de 485% no consumo mediano em agosto de 2024 face a 2023, explicado por práticas inadequadas de manutenção. Na eletricidade, verificaram-se divergências entre edifícios (redução de 7% no edifício principal e aumento de 18% no secundário), cuja análise foi limitada pela ausência de registos complementares. Quanto ao

gás propano, a monitorização revelou-se insuficiente devido à existência de apenas um contador parcial, recomendando-se a instalação de medição setorial em cozinhas, sistemas de AQS e aquecimento.

O consumo diário por cliente destacou-se como indicador-chave (KPI), permitindo normalizar a eficiência face à ocupação. Em 2025, o aumento de 6% na mediana deste indicador sinalizou a necessidade de revisão dos processos de manutenção. A análise do erro relativo diário entre consumo e ocupação mostrou-se uma ferramenta eficaz para identificar desperdícios e falhas operacionais, sobretudo quando cruzada com consumos parciais, permitindo associar desvios a setores específicos. A definição de limites estatísticos (normalidade, atenção, alarme) reforçou a capacidade de resposta proativa.

A aplicação de modelos de *machine learning* e *deep learning* demonstrou elevada eficácia na previsão do consumo energético. A inclusão de variáveis derivadas, como variações e médias e medianas móveis, melhorou significativamente o desempenho preditivo. Entre os modelos testados, as CNN destacaram-se pela estabilidade e precisão (R^2 = 0,80 na 4ª iteração), sobretudo quando alimentadas com variáveis contextuais e setoriais. A integração destes modelos com gráficos de controlo permitiu definir limites dinâmicos de tolerância para o erro relativo, transformando previsões em mecanismos de decisão.

Os resultados obtidos demonstram que a combinação de análise exploratória, desagregação de consumos, indicadores normalizados e modelação preditiva baseada em IA constitui uma abordagem robusta para a gestão inteligente dos recursos energéticos e hídricos em hotelaria. Esta metodologia não só melhora a eficiência e reduz a pegada carbónica, como também reforça a capacidade de diagnóstico operacional e de manutenção preditiva. Em última instância, a monitorização contínua e a integração de ferramentas de IA permitem que a gestão deixe de ser reativa e passe a assumir um carácter proativo, adaptativo e orientado para a otimização tecnológica, garantindo maior sustentabilidade e competitividade para unidades hoteleiras no Algarve.

Durante os últimos anos de avanço tecnológico, tornou-se evidente que, de forma orientada, não basta apenas substituir equipamentos por versões mais eficientes. Idealmente, quando se justifica o investimento em medidas destinadas a melhorar determinados indicadores de desempenho energético numa unidade hoteleira, deve-se assegurar que os ganhos obtidos sejam sustentados ao longo do tempo. Para que este tipo de intervenções se desenvolva de forma pragmática, é necessário definir claramente “o estado da operação no presente”, “como esteve no passado” e qual “o objetivo a atingir no futuro”.

A monitorização de uma unidade hoteleira pode ser exigente, mas quando aplicada com base em decisões conscientes e fundamentadas, permite demonstrar os benefícios alcançados ao longo do tempo. Para tal, é imprescindível conhecer e interpretar as variáveis-chave relacionadas com o desempenho energético, operacional e ambiental, garantindo uma análise contínua, estruturada e comparável dos dados.

De acordo com as boas práticas, não basta reduzir consumos através da substituição de lâmpadas, da troca de equipamentos ou da melhoria do isolamento. A verdadeira otimização energética depende da utilização eficaz dos sistemas de GTC), que constituem elementos estruturais das boas práticas de eficiência. A aplicação consistente destas práticas sustentáveis ao longo de todo o ciclo anual de operação permite reduzir de forma mensurável a despesa com recursos energéticos e melhorar o desempenho global de sustentabilidade da unidade.

A inteligência artificial deixa de ser apenas uma ferramenta de previsão e assume-se como um instrumento estratégico de gestão inteligente, capaz de transformar dados em ação, reduzir custos, aumentar a fiabilidade dos sistemas e reforçar a competitividade das unidades hoteleiras. Em última análise, a sua aplicação garante que a monitorização energética e hídrica evolua de um processo reativo para um modelo proativo, adaptativo e orientado para a otimização contínua. Uma das formas mais eficazes de reforçar esta capacidade analítica consiste no cálculo de valores esperados para diferentes características de funcionamento, permitindo identificar desvios e antecipar falhas. A comunidade científica tem defendido que a aplicação de algoritmos de ML e DL representa um avanço significativo nas metodologias de sustentabilidade energética em edifícios.

Assim, os objetivos principais desta dissertação foram alcançados ao demonstrar a necessidade e a viabilidade de desenvolver uma ferramenta de apoio à gestão de energia, baseada em modelos de previsão de consumos energéticos. Esta ferramenta visa fornecer um conjunto de respostas possíveis às situações técnico-operacionais, considerando o presente, o passado e o futuro, a partir de dados recolhidos pelos sistemas tradicionais de GTC ou de gestão de energia.

5.1 Trabalhos futuros

Com base nas conclusões alcançadas e nas lacunas identificadas na metodologia aplicada, os trabalhos futuros mais relevantes deverão centrar-se na expansão da monitorização, na validação prática dos modelos preditivos desenvolvidos e na integração total dos dados no sistema de gestão operacional. Um dos pontos fundamentais para a continuidade deste trabalho é o alargamento da rede de contadores parciais, uma vez que a ausência de medições desagregadas demonstrou poder mascarar comportamentos anómalos e limitar a interpretação

rigorosa das variações de consumo. Neste sentido, destaca-se a necessidade de integrar de forma sistemática a monitorização detalhada dos sistemas de AVAC, que representam, em regra, um dos maiores consumidores de energia elétrica em unidades hoteleiras. A recolha contínua e desagregada de dados destes equipamentos permitirá não apenas avaliar com maior precisão o seu desempenho energético, mas também identificar ineficiências operacionais, apoiar a manutenção preditiva e fornecer uma base sólida para a otimização tecnológica. Neste ponto, surgem várias motivações, principalmente pelo conjunto de oportunidades para desenvolver processos de otimização de recursos energéticos e hídricos. Toda a metodologia desenvolvida nesta dissertação deve ser transversal a manutenção de todos os ativos do parque de máquinas de um hotel. Assim, a expansão da monitorização, com especial atenção nos sistemas de AVAC, constitui um passo determinante para consolidar a gestão inteligente dos recursos e reforçar a sustentabilidade do setor hoteleiro.

No que respeita a integração de outros pontos de consumo relevantes, como as áreas técnicas (centrais de bombagem, centrais frigoríficas, oficinas) e a iluminação em zonas de grande permanência, de forma a obter uma visão mais abrangente do perfil energético da unidade.

Paralelamente, destaca-se a necessidade de um registo rigoroso de ocorrências de manutenção, tanto preventiva como corretiva, bem como de eventos especiais e alterações operacionais (por exemplo, mudanças de horários ou variações significativas na ocupação). Este histórico detalhado constitui uma base essencial para correlacionar variações de consumo com causas específicas, permitindo sustentar decisões de gestão energética mais fundamentadas. A aplicação de casos práticos, com recurso a dados de utilização futura, poderá ainda apoiar a previsão de comportamentos de consumo e o ajuste de padrões de funcionamento de equipamentos críticos, como caldeiras e chillers.

Outro vetor de desenvolvimento prende-se com a integração dos modelos preditivos e dos alarmes baseados em erro relativo com sistemas de GTC. Esta integração visa evoluir para um controlo inteligente, no qual as previsões se traduzem em ações operacionais concretas e preditivas. Neste contexto, a parametrização de alarmes inteligentes, baseada em estatísticas históricas (quartis e medianas), deve ser refinada para classificar níveis de alerta (normal, atenção, alarme), reforçando a capacidade de gestão energética proativa.

Assim, a aquisição de dados deve ser permanente, estruturada e integrada em plataformas de gestão inteligente e soluções baseadas em IoT. A transformação de observações dispersas em conhecimento estratégico é igualmente essencial, apoiando intervenções de manutenção e reposição de equipamentos em consonância com os princípios da Indústria 4.0.

REFERÊNCIAS

[1]W. Commission on Environment, "Report of the World Commission on Environment and Development: Our Common Future Towards Sustainable Development 2. Part II. Common Challenges Population and Human Resources 4."

[2]"MAKING TOURISM MORE SUSTAINABLE A Guide for Policy Makers Employment Quality Community Wellbeing Biological Diversity Economic Viability Local Control Physical Integrity Environmental Purity Local Prosperity Visitor Fulfillment Cultural Richness Resource Efficiency Social Equity." [Online]. Available: www.unep.frwww.world-tourism.org

[3]Turismo de Portugal, "| GUIA DE BOAS PRÁTICAS DE SUSTENTABILIDADE PARA A ANIMAÇÃO TURÍSTICA GUIA DE BOAS PRÁTICAS DE SUSTENTABILIDADE PARA A ANIMAÇÃO TURÍSTICA."

[4]A. Fichera, R. Volpe, and E. Cutore, "Energy performance measurement, monitoring and control for buildings of public organizations: Standardized practises compliant with the ISO 50001 and ISO 50006," *Developments in the Built Environment*, vol. 4, Nov. 2020, doi: 10.1016/j.dibe.2020.100024.

[5]Turismo de Portugal, "Relatório de Sustentabilidade | 2021."

[6]ENA - Agência de Energia e Ambiente da Arrábida, "Guia de Boas Práticas: 'Por um Turismo Sustentável' - Eficiência Energética no Setor Hoteleiro," 2020. [Online]. Available: www.porumturismosustentavel.pt

[7]C. Fan, F. Xiao, and S. Wang, "Development of prediction models for next-day building energy consumption and peak power demand using data mining techniques," *Appl Energy*, vol. 127, pp. 1–10, Aug. 2014, doi: 10.1016/j.apenergy.2014.04.016.

[8]S. Becken, "Water equity - Contrasting tourism water use with that of the local community," *Water Resour Ind*, vol. 7–8, pp. 9–22, Sep. 2014, doi: 10.1016/j.wri.2014.09.002.

[9]M. André Leonardo de Medeiros Melo, "Análise dos efeitos das variáveis independentes nos consumos energéticos para aplicações em modelos ESCO."

[10] L. Pérez-Lombard, J. Ortiz, and C. Pout, "A review on buildings energy consumption information," *Energy Build*, vol. 40, no. 3, pp. 394–398, 2008, doi: 10.1016/j.enbuild.2007.03.007.

[11] K. He *et al.*, “Predictive control optimization of chiller plants based on deep reinforcement learning,” *Journal of Building Engineering*, vol. 76, Oct. 2023, doi: 10.1016/j.jobe.2023.107158.

[12] H. Kagermann, W. Wahlster, and J. Helbig, “Recommendations for implementing the strategic initiative INDUSTRIE 4.0 April 2013 Securing the future of German manufacturing industry Final report of the Industrie 4.0 Working Group,” 2013.

[13] L. Silvestri, A. Forcina, V. Introna, A. Santolamazza, and V. Cesarotti, “Maintenance transformation through Industry 4.0 technologies: A systematic literature review,” *Comput Ind*, vol. 123, Dec. 2020, doi: 10.1016/j.compind.2020.103335.

[14] McKinsey & Company, “The future of maintenance for distributed fixed assets,” 2020.

[15] B. Salimian Rizi, G. Pavlak, V. Cushing, and M. Heidarinejad, “Predicting uncertainty of a chiller plant power consumption using quantile random forest: A commercial building case study,” *Energy*, vol. 283, Nov. 2023, doi: 10.1016/j.energy.2023.129112.

[16] F. A. Alenizi, S. Abbasi, A. Hussein Mohammed, and A. Masoud Rahmani, “The artificial intelligence technologies in Industry 4.0: A taxonomy, approaches, and future directions,” *Comput Ind Eng*, vol. 185, Nov. 2023, doi: 10.1016/j.cie.2023.109662.

[17] V. Rousopoulou *et al.*, “Cognitive analytics platform with AI solutions for anomaly detection,” *Comput Ind*, vol. 134, Jan. 2022, doi: 10.1016/j.compind.2021.103555.

[18] M. Alarcón, F. M. Martínez-García, and F. C. Gómez de León Hijes, “Energy and maintenance management systems in the context of industry 4.0. Implementation in a real case,” *Renewable and Sustainable Energy Reviews*, vol. 142, May 2021, doi: 10.1016/j.rser.2021.110841.

[19] R. Rai, M. K. Tiwari, D. Ivanov, and A. Dolgui, “Machine learning in manufacturing and industry 4.0 applications,” 2021, *Taylor and Francis Ltd.* doi: 10.1080/00207543.2021.1956675.

[20] T. M. . Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[21] S. Russell and P. Norvig, “Artificial Intelligence A Modern Approach Fourth Edition Global Edition.”

[22] J. Schmidhuber, “Deep Learning in neural networks: An overview,” Jan. 01, 2015, *Elsevier Ltd*. doi: 10.1016/j.neunet.2014.09.003.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks.” [Online]. Available: http://code.google.com/p/cuda-convnet/

[24] A. Vaswani *et al.*, “Attention Is All You Need.”

[25] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why Should I Trust You?' Explaining the Predictions of Any Classifier." [Online]. Available: https://github.

[26] Coelho J, "A inteligência artificial em hotelaria e o impacto da robótica do ponto de vista do consumidor."

[27] Valquir Correa, "INOVAÇÃO TECNOLÓGICA NO SETOR HOTELEIRO: ESTRATÉGIAS PARA GANHOS OPERACIONAIS COM AUTOMAÇÃO ROBÓTICA DE PROCESSOS E INTELIGÊNCIA ARTIFICIAL," *International Integralize Scientific. v 5, n 48*, 2025.

[28] D. E. Santiago, "Energy use in hotels: A case study in Gran Canaria," *International Journal of Low-Carbon Technologies*, vol. 16, no. 4, pp. 1264–1276, Dec. 2021, doi: 10.1093/ijlct/ctab048.

[29] J. António *et al.*, "Aplicação dos Conceitos da Indústria 4.0 na Manutenção de Edifícios de Consumo de Energia Quase Nula," 2021.

[30] G. Dall'O', S. Ferrari, E. Bruni, and L. Bramonti, "Effective implementation of ISO 50001: A case study on energy management for heating load reduction for a social building stock in Northern Italy," *Energy Build*, vol. 219, p. 110029, Jul. 2020, doi: 10.1016/J.ENBUILD.2020.110029.

[31] S. Becken and D. G. Simmons, "Understanding energy consumption patterns of tourist attractions and activities in New Zealand," *Tour Manag*, vol. 23, no. 4, pp. 343–354, Aug. 2002, doi: 10.1016/S0261-5177(01)00091-7.

[32] P. Bohdanowicz and I. Martinac, "Determinants and benchmarking of resource consumption in hotels—Case study of Hilton International and Scandic in Europe," *Energy Build*, vol. 39, no. 1, pp. 82–95, Jan. 2007, doi: 10.1016/J.ENBUILD.2006.05.005.

[33] ERSAR, E. Alexandra Costa, E. Ana Rita Catarino, E. Joana Cardoso, E. João Rosa, and E. Rute Rodrigues Engª Susana Rodrigues, "RELATÓRIO ANUAL DOS SERVIÇOS DE ÁGUAS E RESÍDUOS EM PORTUGAL," 2024. [Online]. Available: www.ersar.pt.

[34] APA, "FATOR DEEMISSÃO DA ELETRICIDADE - 2025," 2025.

[35] DGEG, "Trajetórias de Fatores de Conversão de Energiae de Emissão de GEE para Portugal," 2025.

[36] X. Wei, A. Zhao, X. Zhou, F. Sun, and Z. Feng, "A Hybrid Prediction Model for Hotel Cooling Load Considering Dynamic Occupancy Rate," *International Journal of Refrigeration*, Aug. 2025, doi: 10.1016/J.IJREFRIG.2025.08.023.

[37] I. P. Turismo de Portugal and T. and H. P. NOVA, “Guia neutralidade carbónica nos empreendimentos turísticos,” 2021.

[38] BCSD Portugal, “Edição portuguesa: O Protocolo de GEE.”

[39] M. E. Jensen, “Estimating evaporation from water surfaces ESTIMATING EVAPORATION FROM WATER SURFACES 1.” [Online]. Available: https://www.researchgate.net/publication/265749992

[40] M. S. Piscitelli, R. Giudice, and A. Capozzoli, “A holistic time series-based energy benchmarking framework for applications in large stocks of buildings,” *Appl Energy*, vol. 357, Mar. 2024, doi: 10.1016/j.apenergy.2023.122550.

[41] D. Leiria *et al.*, “Is it returning too hot? Time series segmentation and feature clustering of end-user substation faults in district heating systems,” *Appl Energy*, vol. 381, Mar. 2025, doi: 10.1016/j.apenergy.2024.125122.

[42] Y. Chen and Z. Fu, “Multi-Step Ahead Forecasting of the Energy Consumed by the Residential and Commercial Sectors in the United States Based on a Hybrid CNN-BiLSTM Model,” *Sustainability (Switzerland)*, vol. 15, no. 3, Feb. 2023, doi: 10.3390/su15031895.

[43] S. Xiang, C. Zhen, J. Peng, L. Zhang, and Z. Pu, “Power load prediction of smart grid based on deep learning,” in *Procedia Computer Science*, Elsevier B.V., 2023, pp. 762–773. doi: 10.1016/j.procs.2023.11.090.

[44] S. Seyedzadeh, “Modelling Non-Domestic Buildings Energy Performance Using Machine Learning Methods: A Case Study of the UK,” 2020.

[45] J. Veríssimo Sobreira, “Desafios para a Manutenção na perspetiva da Indústria 4.0,” 2018.

[46] L. Alzubaidi *et al.*, “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,” *J Big Data*, vol. 8, no. 1, Dec. 2021, doi: 10.1186/s40537-021-00444-8.

ANEXOS

Nos anexos encontra-se o *script* desenvolvido em linguagem Python, executado na plataforma Google Colab, utilizado para o tratamento e organização dos dados. A inclusão deste *script* tem como finalidade demonstrar a metodologia computacional aplicada e disponibilizar uma ferramenta prática que complementa a descrição teórica apresentada nos capítulos anteriores.

Importar Dados

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.io as pio
import matplotlib.patches as mpatches
from sklearn.metrics import mean_squared_error
```

```
meu_tema_layout = {
    'layout': {
        'title_x': 0.5,
        'width': 1000,
        'height': 600,
        'title': {'font': {'size': 24}},
        'xaxis': {'title': {'font': {'size': 18}}},
        'yaxis': {
            'title': {'font': {'size': 18}},
            'tickformat': ".0f"
        },
        'font': {'size': 16},
        'template': 'plotly_white',
        'legend': {
            'font': {'size': 10},
            'itemwidth': 30,
            'x': 1,
            'y': 1
        }
    }
}

import plotly.io as pio
pio.templates["meu_tema"] = meu_tema_layout
pio.templates.default = "meu_tema"
```

```
dados_E=pd.read_excel('/content/drive/MyDrive/Colab Notebooks/ELET_GUA/ELET2_GUA.xlsx')
dados_A=pd.read_excel('/content/drive/MyDrive/Colab Notebooks/AGUA_GUA/AGUA2_GUA.xlsx')
dados_G=pd.read_excel('//content/drive/MyDrive/Colab Notebooks/GAS_GUA/GAS_GUA.xlsx')

dados_A = dados_A.rename(columns={'GERAL_C': 'AGUA_C'})
```

```
dados_A = dados_A.rename(columns={'GERAL_C': 'AGUA_C'})
dados_E = dados_E.rename(columns={'GUA.C': 'ELET_C'})
dados_G = dados_G.rename(columns={'GUA.C': 'GAS_C'})
```

```
dados_E.columns
```

```
Index(['DATA', 'T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max',
       'RH_min', 'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'CONS_CLT', 'CONS_Q'],
      dtype='object')
```

```
dados_A.columns
```

```
Index(['DATA', 'T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max',
       'RH_min', 'RH_rg', 'calor_idx', 'AGUA_C', 'EP_C', '50S_C', 'LAV.Q_C',
       'LAV.F_C', 'CT_C', 'REGA_C', 'PSC.JCZ_C', 'PSC.INT_C', 'PSC.NOVA_C',
       'PSC.CR_C', 'PSC.GR_C', 'GR.BW_C', 'PSC.PQ_C', 'PQ.BW_C', 'PSC.TSP_C',
       'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR', 'CLTs', 'CONS_CLT', 'CONS_Q',
       'REGA_CLT'],
      dtype='object')
```

```
dados_G.columns
```

```
Index(['DATA', 'T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max',
       'RH_min', 'RH_rg', 'calor_idx', 'GAS_C', 'LAV.C', 'Q_OCP', 'OCP',
       'OCP[%]', 'AD', 'CR', 'CLTs', 'CONS_CLT', 'CONS_Q'],
      dtype='object')
```

```
dados_E = dados_E.set_index('DATA')
dados_E.index = pd.to_datetime(dados_E.index, format='%Y-%m-%d %H:%M:%S')
dados_E.tail()
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	PST.C	LAV.C	Q_OCP	OCP	OCP[%]	AD	CR	CLTs	CONS_CLT	CONS_Q
2025-10-16	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12713.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2025-10-17	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12668.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2025-10-18	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12138.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2025-10-19	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12453.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2025-10-20	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12044.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 24 columns

```
dados_E.tail()
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	PST.C	LAV.C	Q_OCP	OCP	OCP[%]	AD	CR	CLTs	CONS_CLT	CONS_Q
2025-10-16	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12713.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2025-10-17	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12668.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2025-10-18	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12138.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2025-10-19	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12453.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2025-10-20	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12044.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 24 columns

```
dados_A = dados_A.set_index('DATA')
dados_A.index = pd.to_datetime(dados_A.index, format='%Y-%m-%d')
dados_A.head()
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	calor_idx	AGUA_C	...	PSC.TSP_C	Q_OCP	OCP	OCP[%]	AD	CR	CLTs	CONS_CLT	CONS_Q	REGA_CLT
2023-01-02	14.050000	17.299999	11.7	5.599999	85.716667	100.000000	59.299999	40.700001	22.347228	7.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2023-01-03	12.300000	17.400000	8.6	8.799999	74.133333	87.099998	49.099998	38.000000	29.903723	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2023-01-04	13.850000	19.700001	10.0	9.700001	85.166667	99.000000	59.200001	39.799999	22.878928	8.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2023-01-05	14.257143	20.299999	11.1	9.199999	89.614285	99.199997	60.900002	38.299995	20.169687	10.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2023-01-06	14.040000	20.000000	9.8	10.200000	89.300000	100.000000	67.000000	33.000000	20.595821	30.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 34 columns

```
dados_G = dados_G.set_index('DATA')
dados_G.index = pd.to_datetime(dados_G.index, format='%Y-%m-%d')
dados_G.head()
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	calor_idx	GAS_C	LAV.C	Q_OCP	OCP	OCP[%]	AD	CR	CLTs	CONS_CLT	CONS_Q
2023-03-20	16.785714	23.000000	12.9	10.100000	80.242857	89.300003	68.900002	20.400002	21.796942	224.796	NaN	0.0	0.00	0.0	0.0	0.0	0.0	NaN	NaN
2023-03-21	19.300000	23.299999	15.6	7.699999	69.060001	77.300003	59.200001	18.100002	23.452173	224.796	NaN	0.0	0.00	0.0	0.0	0.0	0.0	NaN	NaN
2023-03-22	17.385714	23.900000	13.5	10.400000	78.785715	93.500000	59.000000	34.500000	21.899319	224.756	NaN	0.0	0.00	0.0	0.0	0.0	0.0	NaN	NaN
2023-03-23	15.750000	21.900000	11.3	10.599999	83.350001	96.800003	71.800003	25.000000	21.472725	224.796	NaN	0.0	0.00	0.0	0.0	0.0	0.0	NaN	NaN
2023-03-24	16.883333	21.400000	11.7	9.700000	87.233334	97.599998	76.400002	21.199997	19.157320	561.990	NaN	15.0	0.03	3.0	30.0	3.0	33.0	17.03	37.466

```
dados_E = pd.merge(dados_E, dados_A[['AGUA_C']], left_index=True, right_index=True, how='left')
dados_E = pd.merge(dados_E, dados_G[['GAS_C']], left_index=True, right_index=True, how='left')
```

```
dados_E[['ELET_C', 'AGUA_C', 'GAS_C']].tail()
```

DATA	ELET_C	AGUA_C	GAS_C
2025-10-16	12713.0	272.0	674.388
2025-10-17	12668.0	337.0	674.388
2025-10-18	12138.0	269.0	786.786
2025-10-19	12453.0	281.0	337.194
2025-10-20	12044.0	273.0	561.990

Criar variaveis de "time series"

dados['dia_semana'] = dados.index.dayofweek

dados['trimestre'] = dados.index.quarter

dados['mes'] = dados.index.month

dados['ano'] = dados.index.year

dados['dia_ano'] = dados.index.dayofyear

dados['dia_mes'] = dados.index.day

dados['semana_ano'] = dados.index.isocalendar().week

```
import holidays
```

```
def create_features(dados_E):
    """
    Criar as variaveis baseadas no "time series index".
    """

    dados_E = dados_E.copy()
    dados_E['dia_semana'] = dados_E.index.dayofweek
    dados_E['trimestre'] = dados_E.index.quarter
    dados_E['mes'] = dados_E.index.month
    dados_E['ano'] = dados_E.index.year
    dados_E['dia_ano'] = dados_E.index.dayofyear
    dados_E['dia_mes'] = dados_E.index.day
    dados_E['semana_ano'] = dados_E.index.isocalendar().week

    # Feriados nacionais de Portugal
    feriados_pt = holidays.Portugal()

    # Criar coluna booleana: True se for feriado
    dados_E['feriado'] = dados_E.index.to_series().apply(lambda x: x in feriados_pt)

    return dados_E

dados_E = create_features(dados_E)
```

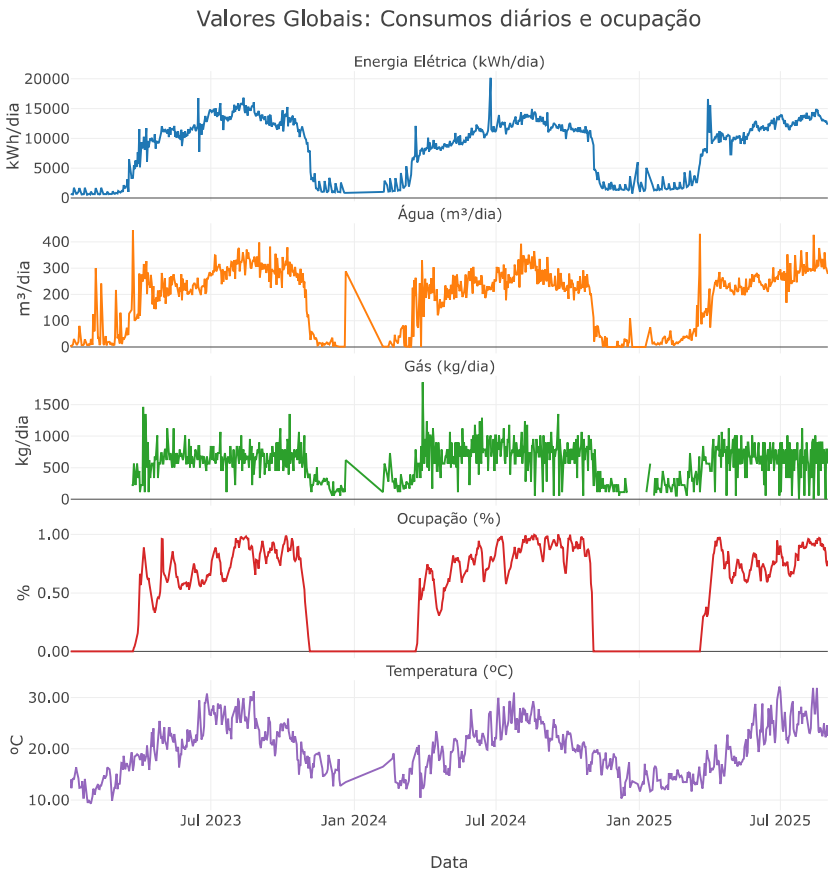
```
dados_E.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado'],
      dtype='object')
```

Analises do Consumo Geral

```
from plotly.subplots import make_subplots
import plotly.graph_objects as go

# Filtrar os dados
df = dados_E.loc[dados_E.index <= '2025-08-31', ['ELET_C', 'AGUA_C', 'GAS_C', 'OCP', 'T_mean']].reset_index()

# Criar subplots com 4 gráficos verticais
fig = make_subplots(
    rows=5, cols=1,
    shared_xaxes=True,
    vertical_spacing=0.03,
    subplot_titles=('Energia Elétrica (kWh/dia)', 'Água (m³/dia)', 'Gás (kg/dia)', 'Ocupação (%)', 'Temperatura (ºC)')
)

# Gráfico 1: Energia
fig.add_trace(go.Scatter(x=df['DATA'], y=df['ELET_C'], name='Energia'),
```

```
            row=1, col=1)

# Gráfico 2: Água
fig.add_trace(go.Scatter(x=df['DATA'], y=df['AGUA_C'], name='Água'),
              row=2, col=1)

# Gráfico 3: Gás
fig.add_trace(go.Scatter(x=df['DATA'], y=df['GAS_C'], name='Gás'),
              row=3, col=1)

# Gráfico 4: Ocupação
fig.add_trace(go.Scatter(x=df['DATA'], y=df['OCP'], name='Ocupação'),
              row=4, col=1)

# Gráfico 4: Temperatura média
fig.add_trace(go.Scatter(x=df['DATA'], y=df['T_mean'], name='Temperatura'),
              row=5, col=1)

# Layout geral
fig.update_layout(
    height=1000,
    width=1000,
    title='Valores Globais: Consumos diários e ocupação',
    title_font=dict(size=24),
    font=dict(size=16),
    showlegend=False
)

# Ajustes dos eixos
fig.update_xaxes(title_text='Data', row=5, col=1)
fig.update_yaxes(title_text='kWh/dia', row=1, col=1)
fig.update_yaxes(title_text='m³/dia', row=2, col=1)
fig.update_yaxes(title_text='kg/dia', row=3, col=1)
fig.update_yaxes(title_text='%', row=4, col=1, tickformat=".2f")
fig.update_yaxes(title_text='ºC', row=5, col=1, tickformat=".2f")

# Exibir o gráfico
fig.show()
#fig.show('notebook_connected')
```




```
#fig.write_html("/content/drive/MyDrive/Colab Notebooks/grafico_consumos_3GLOBAIS.html")
```

Correlação c/ Ocupação

```
dados_E.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado'],
      dtype='object')
```

```
dados_E.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado'],
      dtype='object')
```

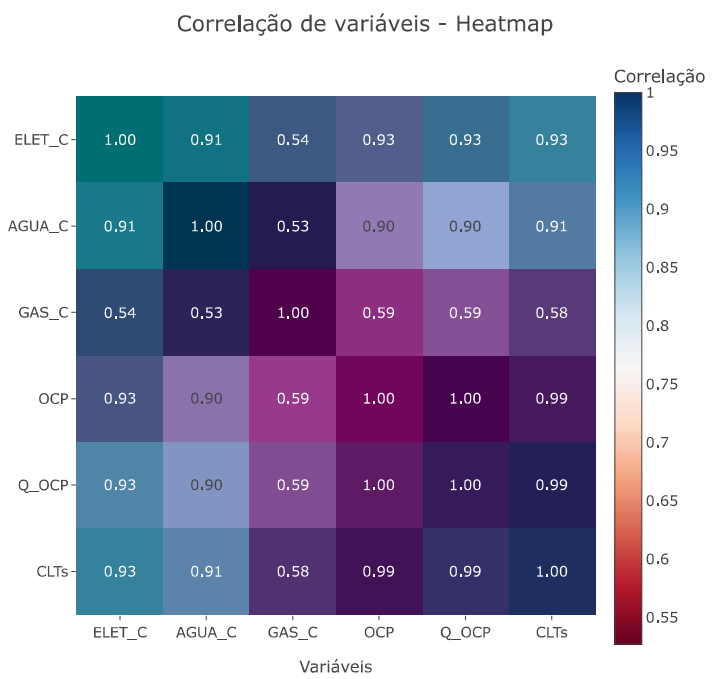
```
from statsmodels.api import OLS
import statsmodels.api as sm
modelo = OLS(dados_E['ELET_C'], sm.add_constant(dados_E['OCP'])).fit()
modelo.summary()
```

```
---------------------------------------------------------------------------
MissingDataError                          Traceback (most recent call last)
/tmp/ipython-input-1128323592.py in <cell line: 0>()
      1 from statsmodels.api import OLS
      2 import statsmodels.api as sm
----> 3 modelo = OLS(dados_E['ELET_C'], sm.add_constant(dados_E['OCP'])).fit()
      4 modelo.summary()

 8 frames
/usr/local/lib/python3.12/dist-packages/statsmodels/base/data.py in _handle_constant(self, hasconst)
    132             exog_max = np.max(self.exog, axis=0)
    133             if not np.isfinite(exog_max).all():
--> 134                 raise MissingDataError('exog contains inf or nans')
    135             exog_min = np.min(self.exog, axis=0)
    136             const_idx = np.where(exog_max == exog_min)[0].squeeze()

MissingDataError: exog contains inf or nans
```

```
dados_E.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado'],
      dtype='object')
```

```
# Calcular a matriz de correlação
corr_matrix = dados_E[['ELET_C', 'AGUA_C', 'GAS_C',  'OCP', 'Q_OCP',
       'CLTs',]].corr()

# Renomear a coluna temporariamente para a visualização
#corr_matrix.rename(columns={'ELET_C': 'Consumo Geral'}, index={'ELET_C': 'Consumo Geral', 'T_mean': 'Temperatura', 'RH_mean': 'Humidade'}, inplace=True)

# Criar o heatmap interativo
# Change 'coolwarm' to a valid plotly colorscale like 'RdBu'
fig = px.imshow(corr_matrix,
                labels={ "x": "Variáveis","y": " ", "color": "Correlação"},
                color_continuous_scale="RdBu", # Changed colorscale to a valid plotly colorscale
                text_auto=True)  # Exibe os valores diretamente no gráfico


# Ajustarcasas decimais
fig.update_traces(texttemplate="%{z:.2f}")
# Ajustar layout
fig.update_layout(title="Correlação de variáveis - Heatmap", title_x=0.5, width=800, height=800)
fig.update_layout(
    #title_font=dict(size=24),  # Tamanho do título
    xaxis_title_font=dict(size=18),  # Tamanho do rótulo do eixo X
    yaxis_title_font=dict(size=18),  # Tamanho do rótulo do eixo Y
    font=dict(size=16))
fig.show()
#fig.show('notebook_connected')
```



Consumo por Cliente

```
dados_E.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado'],
      dtype='object')
```

```
dados_A = dados_A.rename(columns={'CONS_CLT': 'AGUA_CONS_CLT'})
dados_E = dados_E.rename(columns={'CONS_CLT': 'ELET_CONS_CLT'})
dados_G = dados_G.rename(columns={'CONS_CLT': 'GAS_CONS_CLT'})
```

```
dados_E = pd.merge(dados_E, dados_A[['AGUA_CONS_CLT']], left_index=True, right_index=True, how='left')
dados_E = pd.merge(dados_E, dados_G[['GAS_CONS_CLT']], left_index=True, right_index=True, how='left')
```

```
dados_E.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT'],
      dtype='object')
```

```
df = dados_E.loc[dados_E.index <= '2025-08-31', ['ELET_CONS_CLT', 'AGUA_CONS_CLT', 'GAS_CONS_CLT']].reset_index()

fig = make_subplots(
    rows=3, cols=1,
    shared_xaxes=True,
    vertical_spacing=0.03,
    subplot_titles=('Energia Elétrica', 'Água', 'Gás')
)

# Gráfico 1: Energia
fig.add_trace(go.Scatter(x=df['DATA'], y=df['ELET_CONS_CLT'], name='Energia'),
              row=1, col=1)

# Gráfico 2: Água
fig.add_trace(go.Scatter(x=df['DATA'], y=df['AGUA_CONS_CLT'], name='Água'),
              row=2, col=1)

# Gráfico 3: Gás
fig.add_trace(go.Scatter(x=df['DATA'], y=df['GAS_CONS_CLT'], name='Gás'),
              row=3, col=1)

# Gráfico 4: Ocupação
#fig.add_trace(go.Scatter(x=df['DATA'], y=df['OCP'], name='Ocupação', line=dict(color='purple')),
          #    row=4, col=1)

# Layout geral
fig.update_layout(
    height=1000,
    width=1000,
    title='Consumos Diários por Cliente',
    title_font=dict(size=24),
    font=dict(size=16),
    showlegend=False
)

# Ajustes dos eixos
fig.update_xaxes(title_text='Data', row=4, col=1)
fig.update_yaxes(title_text='kWh/clt/dia', row=1, col=1, range=[0, 50])
fig.update_yaxes(title_text='m³/clt/dia', row=2, col=1, tickformat=".2f", range=[0, 1])
fig.update_yaxes(title_text='kg/clt/dia', row=3, col=1, tickformat=".2f", range=[0, 2])
#fig.update_yaxes(title_text='%', row=4, col=1, tickformat=".2f")

# Exibir o gráfico
fig.show()
#fig.show('notebook_connected')
```


```
df = dados_E.loc[(dados_E['mes'] >= 4) & (dados_E['mes'] < 11), ['ELET_CONS_CLT', 'AGUA_CONS_CLT', 'GAS_CONS_CLT']].reset_index()


fig = make_subplots(
    rows=3, cols=1,
    shared_xaxes=True,
    vertical_spacing=0.03,
    subplot_titles=('Energia Elétrica', 'Água', 'Gás')
)

# Gráfico 1: Energia
fig.add_trace(go.Scatter(x=df['DATA'], y=df['ELET_CONS_CLT'], name='Energia'),
              row=1, col=1)

# Gráfico 2: Água
fig.add_trace(go.Scatter(x=df['DATA'], y=df['AGUA_CONS_CLT'], name='Água'),
              row=2, col=1)

# Gráfico 3: Gás
fig.add_trace(go.Scatter(x=df['DATA'], y=df['GAS_CONS_CLT'], name='Gás'),
              row=3, col=1)

# Gráfico 4: Ocupação
#fig.add_trace(go.Scatter(x=df['DATA'], y=df['OCP'], name='Ocupação', line=dict(color='purple')),
          #    row=4, col=1)

# Layout geral
fig.update_layout(
```

```
    height=1000,
    width=1000,
    title='Consumos Diários por Cliente',
    title_font=dict(size=24),
    font=dict(size=16),
    showlegend=False
)

# Ajustes dos eixos
fig.update_xaxes(title_text='Data', row=4, col=1)
fig.update_yaxes(title_text='kWh/clt/dia', row=1, col=1, range=[0, 50])
fig.update_yaxes(title_text='m³/clt/dia', row=2, col=1, tickformat=".2f", range=[0, 1])
fig.update_yaxes(title_text='kg/clt/dia', row=3, col=1, tickformat=".2f", range=[0, 2])
#fig.update_yaxes(title_text='%', row=4, col=1, tickformat=".2f")

# Exibir o gráfico
fig.show()
#fig.show('notebook_connected')
```

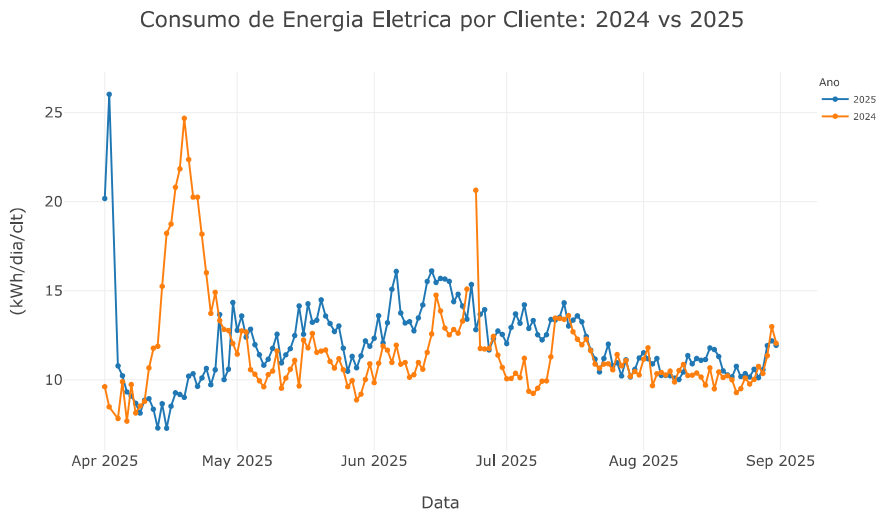

```
df_c = dados_E.loc['2025-04-01':'2025-08-31', ['ELET_CONS_CLT']]
df_c['DiaMes'] = df_c.index.strftime('%d-%m')

df_2024 = dados_E.loc['2024-04-01':'2024-08-31', ['ELET_CONS_CLT']]
df_2024['DiaMes'] = df_2024.index.strftime('%d-%m')

# Criar uma versão reduzida de df_2024 com 'DiaMes' como índice
df_2024_reduzido = df_2024[['ELET_CONS_CLT', 'DiaMes']].copy()
df_2024_reduzido.set_index('DiaMes', inplace=True)

# Mapear os valores de 2024 para df_c com base na coluna 'DiaMes'
df_c['cons_2024'] = df_c['DiaMes'].map(df_2024_reduzido['ELET_CONS_CLT'])
df_c
```

DATA	ELET_CONS_CLT	DiaMes	cons_2024
2025-04-01	20.167279	01-04	9.607102
2025-04-02	26.015000	02-04	8.474630
2025-04-04	10.777647	04-04	7.824859
2025-04-05	10.224138	05-04	9.891707
2025-04-06	9.313346	06-04	7.674190
...	...	...	...
2025-08-27	10.112764	27-08	10.745511
2025-08-28	10.565074	28-08	10.350681
2025-08-29	11.912844	29-08	11.349693
2025-08-30	12.184495	30-08	12.976145
2025-08-31	11.930569	31-08	12.045279

152 rows × 3 columns

```
df_c = dados_E.loc['2025-04-01':'2025-08-31', ['ELET_CONS_CLT']]
df_c['DiaMes'] = df_c.index.strftime('%d-%m')

df_2024 = dados_E.loc['2024-04-01':'2024-08-31', ['ELET_CONS_CLT']]
df_2024['DiaMes'] = df_2024.index.strftime('%d-%m')

# Criar uma versão reduzida de df_2024 com 'DiaMes' como índice
df_2024_reduzido = df_2024[['ELET_CONS_CLT', 'DiaMes']].copy()
df_2024_reduzido.set_index('DiaMes', inplace=True)

# Mapear os valores de 2024 para df_c com base na coluna 'DiaMes'
df_c['cons_2024'] = df_c['DiaMes'].map(df_2024_reduzido['ELET_CONS_CLT'])

fig = go.Figure()

fig.add_trace(go.Scatter(
    x=df_c.index,
    y=df_c['ELET_CONS_CLT'],
    mode='markers+lines',
```

```
    name='2025'
))

fig.add_trace(go.Scatter(
    x=df_c.index,
    y=df_c['cons_2024'],
    mode='markers+lines',
    name='2024'
))

fig.update_layout(
    title='Consumo de Energia Eletrica por Cliente: 2024 vs 2025',
    xaxis_title='Data',
    yaxis_title='(kWh/dia/clt)',
    legend_title='Ano',

)


fig.show()
fig.show('notebook_connected')
```




```
dados_E.columns
```

```
df = dados_E.loc[dados_E.index <= '2025-08-31', ['ELET_CONS_CLT', 'AGUA_CONS_CLT', 'GAS_CONS_CLT']].reset_index()

fig = go.Figure()

# Energia Elétrica no eixo primário
fig.add_trace(go.Scatter(
    x=df['DATA'], y=df['ELET_CONS_CLT'], name='Energia Elétrica (kWh/clt)',
    mode='lines'
))

# Água no eixo secundário
fig.add_trace(go.Scatter(
    x=df['DATA'], y=df['AGUA_CONS_CLT'], name='Água (m3/clt)',
    mode='lines',
    yaxis='y2'
))

# Gás no eixo secundário
fig.add_trace(go.Scatter(
    x=df['DATA'], y=df['GAS_CONS_CLT'], name='Gás (kg/clt)',
    mode='lines',
    yaxis='y2'
))

# Configuração dos eixos e layout
fig.update_layout(
    title='Consumo por cliente: Energia, Água e Gás',
    title_x=0.5,
    xaxis=dict(title='Data'),
    yaxis=dict(title='Energia Elétrica (kWh/clt)', titlefont=dict(size=18)),
    yaxis2=dict(title='Água e Gás', titlefont=dict(size=18),
                overlaying='y', side='right', showgrid=False, tickformat=".2f" ),
    width=1000,
    height=600,
    font=dict(size=16)
)

fig.show()
```

Comece a programar ou gerar com a IA.

```
dados_E.columns
```

Tipo de Cliente

```
from plotly.subplots import make_subplots
import plotly.graph_objects as go


df = dados_E.loc[dados_E.index <= '2025-08-31', ['ELET_C', 'AGUA_C', 'GAS_C', 'AD', 'CR']].reset_index()

# Criar subplots com 4 gráficos verticais
fig = make_subplots(
    rows=4, cols=1,
    shared_xaxes=True,
    vertical_spacing=0.03,
    subplot_titles=('Energia Elétrica (kWh/dia)', 'Água (m³/dia)', 'Gás (kg/dia)', 'Ocupação')
)

# Gráfico 1: Energia
fig.add_trace(go.Scatter(x=df['DATA'], y=df['ELET_C'], name='Energia'),
              row=1, col=1)

# Gráfico 2: Água
fig.add_trace(go.Scatter(x=df['DATA'], y=df['AGUA_C'], name='Água'),
              row=2, col=1)

# Gráfico 3: Gás
fig.add_trace(go.Scatter(x=df['DATA'], y=df['GAS_C'], name='Gás'),
              row=3, col=1)

# Gráfico 4: Ocupação com duas séries separadas
fig.add_trace(go.Scatter(
    x=df['DATA'], y=df['AD'], name='Adultos'),
    row=4, col=1
)

fig.add_trace(go.Scatter(
    x=df['DATA'], y=df['CR'], name='Crianças'),
    row=4, col=1
)


# Layout geral
fig.update_layout(
    height=1000,
    width=1000,
    title='Consumo Diário vs Ocupação',
    title_font=dict(size=24),
    font=dict(size=16),
    showlegend=True
)

# Ajustes dos eixos
fig.update_xaxes(title_text='Data', row=4, col=1)
fig.update_yaxes(title_text='kWh/dia', row=1, col=1)
fig.update_yaxes(title_text='m³/dia', row=2, col=1)
fig.update_yaxes(title_text='kg/dia', row=3, col=1)
fig.update_yaxes(title_text='Clientes', row=4, col=1, tickformat=".0f")


# Gerar 8 datas espaçadas ao longo do ano
tickvals = pd.date_range(start='2023-01-01', end='2025-08-31', periods=9)
ticktext = [d.strftime('%b %Y') for d in tickvals]  # Ex: 'Jan 2025', 'Mar 2025', ...


fig.update_xaxes(
    tickvals=tickvals,
    ticktext=ticktext,
    tickangle=0
)


fig.show()
#fig.show('notebook_connected')
```

```
df = dados_E.loc[dados_E.index <= '2025-08-31', ['ELET_C', 'AD', 'CR']].reset_index()


fig = make_subplots(rows=2, cols=1, shared_xaxes=True, row_heights=[0.85, 0.2])


fig.add_trace(
    go.Scatter(x=df['DATA'], y=df['ELET_C'], name='Consumo de Energia (kWh/dia)'),
    row=1, col=1
)


fig.add_trace(
    go.Scatter(x=df['DATA'], y=df['AD'], name='Adultos'),
    row=2, col=1
)

fig.add_trace(
    go.Scatter(x=df['DATA'], y=df['CR'], name='Crianças'),
    row=2, col=1
)



fig.update_layout(
    title='Consumo de Energia Elétrica - Geral',

    width=1400,
    height=800,
    title_x=0.5,
    font=dict(size=16)
)


fig.update_yaxes(title_text="kWh/dia", row=1, col=1)
fig.update_yaxes(title_text="nº pessoas", row=2, col=1)

fig.show()
fig.show('notebook_connected')
```

```
dados_E.columns
```

Estatisticas gerais

Pegada carbono

```
dados_E['CO2e_ELET'] = dados_E['ELET_C'] * 0.167 #corrigir fator de conversão
dados_E['CO2e_AGUA'] = dados_E['AGUA_C'] * 0.29 #corrigir fator de conversão
dados_E['CO2e_GAS'] = dados_E['GAS_C'] * 3.26 #corrigir fator de conversão
```

```
dados_E['Total_CO2e'] = dados_E['CO2e_ELET'] + dados_E['CO2e_AGUA'] + dados_E['CO2e_GAS']
```

```
dados_E['CO2e_QOCP'] = dados_E['Total_CO2e'] / dados_E['Q_OCP']
```

```
dados_E
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	dia_mes	semana_ano	feriado	AGUA_CONS_CLT	GAS_CONS_CLT	CO2e_ELET	CO2e_AGUA	CO2e_GAS	Total_CO2e
2023-01-02	14.050000	17.299999	11.7	5.599999	85.716667	100.000000	59.299999	40.700001	22.347228	691.0	...	2	1	False	0.0	NaN	115.397	2.03	NaN	NaN
2023-01-03	12.300000	17.400000	8.6	8.799999	74.133333	87.099998	49.099998	38.000000	29.903723	606.0	...	3	1	False	0.0	NaN	101.202	0.29	NaN	NaN
2023-01-04	13.850000	19.700001	10.0	9.700001	85.166667	99.000000	59.200001	39.799999	22.878928	585.0	...	4	1	False	0.0	NaN	97.695	2.32	NaN	NaN
2023-01-05	14.257143	20.299999	11.1	9.199999	89.614285	99.199997	60.900002	38.299995	20.169687	610.0	...	5	1	False	0.0	NaN	101.870	2.90	NaN	NaN
2023-01-06	14.040000	20.000000	9.8	10.200000	89.300000	100.000000	67.000000	33.000000	20.595821	1721.0	...	6	1	False	0.0	NaN	287.407	8.70	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2025-10-16	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12713.0	...	16	42	False	NaN	NaN	2123.071	78.88	2198.50488	4400.45588
2025-10-17	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12668.0	...	17	42	False	NaN	NaN	2115.556	97.73	2198.50488	4411.79088
2025-10-18	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12138.0	...	18	42	False	NaN	NaN	2027.046	78.01	2564.92236	4669.97836
2025-10-19	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12453.0	...	19	42	False	NaN	NaN	2079.651	81.49	1099.25244	3260.39344
2025-10-20	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12044.0	...	20	43	False	NaN	NaN	2011.348	79.17	1832.08740	3922.60540

903 rows × 41 columns

```
dados_E.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP'],
      dtype='object')
```

```
display( dados_E[['CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP']].loc[( dados_E['ano']==2023)].describe())
display( dados_E[['CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP']].loc[( dados_E['ano']==2024)].describe())
display( dados_E[['CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP']].loc[( dados_E['ano']==2025)].describe())
```

```
display( dados_E[['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']].loc[( dados_E['ano']==2023)].describe())
display( dados_E[['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']].loc[( dados_E['ano']==2024)].describe())
display( dados_E[['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']].loc[( dados_E['ano']==2025)].describe())
```

	ELET_C	AGUA_C	GAS_C	CLTs
count	324.000000	324.000000	264.000000	324.000000
mean	8895.033951	195.885802	587.321973	660.709877
std	5346.079135	119.230853	243.499743	491.458664
min	535.000000	1.000000	56.199000	0.000000
25%	2414.750000	57.000000	449.592000	0.000000
50%	11072.500000	233.000000	618.189000	843.000000
75%	13323.750000	289.000000	730.587000	1071.250000
max	16854.000000	445.000000	1461.174000	1384.000000

	ELET_C	AGUA_C	GAS_C	CLTs
count	307.000000	307.000000	301.000000	307.000000
mean	8830.540717	185.472313	621.923153	724.022801
std	4258.306892	105.753378	315.527417	484.965489
min	744.000000	0.000000	56.199000	0.000000
25%	6196.000000	82.000000	337.194000	0.000000
50%	10487.000000	224.000000	674.388000	934.000000
75%	11939.500000	259.500000	842.985000	1081.500000
max	20199.000000	392.000000	1854.567000	1416.000000

	ELET_C	AGUA_C	GAS_C	CLTs
count	272.000000	272.000000	265.000000	222.000000
mean	9740.566176	207.029412	542.479404	709.864865
std	4294.448498	103.358756	283.243762	487.341138
min	1029.000000	0.000000	0.000000	0.000000
25%	8075.750000	165.750000	337.194000	0.000000
50%	11664.500000	242.500000	561.990000	874.000000
75%	12771.750000	272.250000	786.786000	1099.500000
max	16623.000000	430.000000	1123.980000	1357.000000

```
de_2023 = dados_E.loc[
    (dados_E.index >= '2023-04-01') & (dados_E.index <= '2023-11-01'),
    ['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']
].describe()


display(de_2023)
```

```
de_2023 = dados_E.loc[
    (dados_E.index >= '2023-04-01') & (dados_E.index <= '2023-11-01'),
    ['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']
].describe()
de_2024 = dados_E.loc[
    (dados_E.index >= '2024-04-01') & (dados_E.index <= '2024-11-01'),
    ['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']
].describe()
de_2025 = dados_E.loc[
    (dados_E.index >= '2025-04-01') & (dados_E.index <= '2025-11-01'),
    ['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']
].describe()
```

```
# Exibe o resultado
display(de_2023)
display(de_2024)
```

	ELET_C	AGUA_C	GAS_C	CLTs
count	215.000000	215.000000	215.000000	215.000000
mean	12393.000000	268.720930	658.443167	985.646512
std	2063.800801	51.759926	203.126433	213.980730
min	5142.000000	119.000000	112.398000	352.000000
25%	11072.500000	229.500000	561.990000	848.000000
50%	12448.000000	275.000000	674.388000	979.000000
75%	13877.500000	305.500000	730.587000	1144.000000
max	16854.000000	397.000000	1461.174000	1384.000000

	ELET_C	AGUA_C	GAS_C	CLTs
count	214.000000	214.000000	214.000000	214.000000
mean	11294.780374	245.294393	748.182014	1010.537383
std	1767.764904	46.061791	240.167861	222.592358
min	7820.000000	121.000000	56.199000	349.000000
25%	10055.250000	218.000000	618.189000	909.250000
50%	11423.000000	239.500000	786.786000	1018.000000
75%	12414.250000	275.750000	899.184000	1157.000000
max	20199.000000	392.000000	1348.776000	1416.000000

```
relative_error_2024_vs_2023 = ((de_2024 - de_2023) / de_2023) * 100
relative_error_2025_vs_2024 = ((de_2025 - de_2024) / de_2024) * 100
relative_error_2025_vs_2023 = ((de_2025 - de_2023) / de_2023) * 100
print("Variação percentual (%) entre as estatisticas de 2024 e 2023")
display(relative_error_2024_vs_2023.round(2))
print("Variação percentual (%) entre as estatisticas de 2025 e 2024")
display(relative_error_2025_vs_2024.round(2))
print("Variação percentual (%) entre as estatisticas de 2025 e 2023")
display(relative_error_2025_vs_2023.round(2))
```

```
Variação percentual (%) entre as estatisticas de 2024 e 2023
```

	ELET_C	AGUA_C	GAS_C	CLTs
count	-0.47	-0.47	-0.47	-0.47
mean	-8.86	-8.72	13.63	2.53
std	-14.34	-11.01	18.24	4.02
min	52.08	1.68	-50.00	-0.85
25%	-9.19	-5.01	10.00	7.22
50%	-8.23	-12.91	16.67	3.98
75%	-10.54	-9.74	23.08	1.14
max	19.85	-1.26	-7.69	2.31

```
Variação percentual (%) entre as estatisticas de 2025 e 2024
```

	ELET_C	AGUA_C	GAS_C	CLTs
count	-5.61	-5.61	-5.61	-28.97
mean	6.47	6.07	-17.90	0.75
std	-20.30	-11.23	11.93	-11.14
min	-6.75	-38.84	-100.00	55.87
25%	11.96	7.45	-27.27	-4.26
50%	6.79	6.47	-14.29	-2.65
75%	4.84	2.63	-12.50	4.95
max	-22.72	8.67	-16.67	-4.17

```
Variação percentual (%) entre as estatisticas de 2025 e 2023
```

	ELET_C	AGUA_C	GAS_C	CLTs
count	-6.05	-6.05	-6.05	-29.30
mean	-2.96	-3.18	-6.71	3.30
std	-31.74	-21.00	32.34	-7.57
min	41.81	-37.82	-100.00	54.55
25%	1.68	2.07	-20.00	2.65
50%	-2.00	-7.27	0.00	1.23
75%	-6.21	-7.36	7.69	6.14
max	-7.39	7.30	-23.08	-1.95

```
# Agrupar média mensal por ano
mensal_2023 = dados_E[dados_E['ano'] == 2023].groupby('mes')[['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']].mean()
mensal_2024 = dados_E[dados_E['ano'] == 2024].groupby('mes')[['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']].mean()
# Calcular variação percentual mês a mês
diff_percentual_mensal = ((mensal_2024 - mensal_2023) / mensal_2023) * 100

# Arredondar os valores para melhor visualização
print("📊 Variação percentual mensal entre 2024 e 2023 (%)")
display(diff_percentual_mensal.round(2))
```

📊 Variação percentual mensal entre 2024 e 2023 (%)

mes	ELET_C	AGUA_C	GAS_C	CLTs
1	NaN	NaN	NaN	NaN
2	93.99	-50.00	NaN	NaN
3	71.45	-3.03	22.35	263.21
4	-8.74	-12.49	25.25	-11.96
5	-10.60	1.23	1.02	9.85
6	-5.29	2.46	26.90	12.13
7	-11.57	-8.94	23.88	5.76
8	-9.73	-5.90	11.54	2.46
9	-7.46	-16.73	6.15	-2.29
10	-8.49	-18.11	5.10	0.04
11	-3.39	-2.38	-2.40	11.53
12	53.75	-47.88	-11.90	NaN

```
dados_E.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP'],
      dtype='object')
```

```
# Calcular consumos anuais por mês
consumos_anuais = dados_E.groupby(['ano'])[['ELET_C','AGUA_C','GAS_C', 'CO2e_ELET', 'CO2e_AGUA', 'CO2e_GAS','Total_CO2e']].sum()
# Visualizar resultados arredondados
print("📊 Consumos anuais mensais")
display(consumos_anuais.round(2))
```

📊 Consumos anuais mensais

ano	ELET_C	AGUA_C	GAS_C	CO2e_ELET	CO2e_AGUA	CO2e_GAS	Total_CO2e
2023	2881991.0	63467.0	155053.00	481292.50	18405.43	505472.78	994026.36
2024	2710976.0	56940.0	187198.87	452732.99	16512.60	610268.31	1076828.99
2025	2649434.0	56312.0	143757.04	442455.48	16330.48	468647.96	925558.32

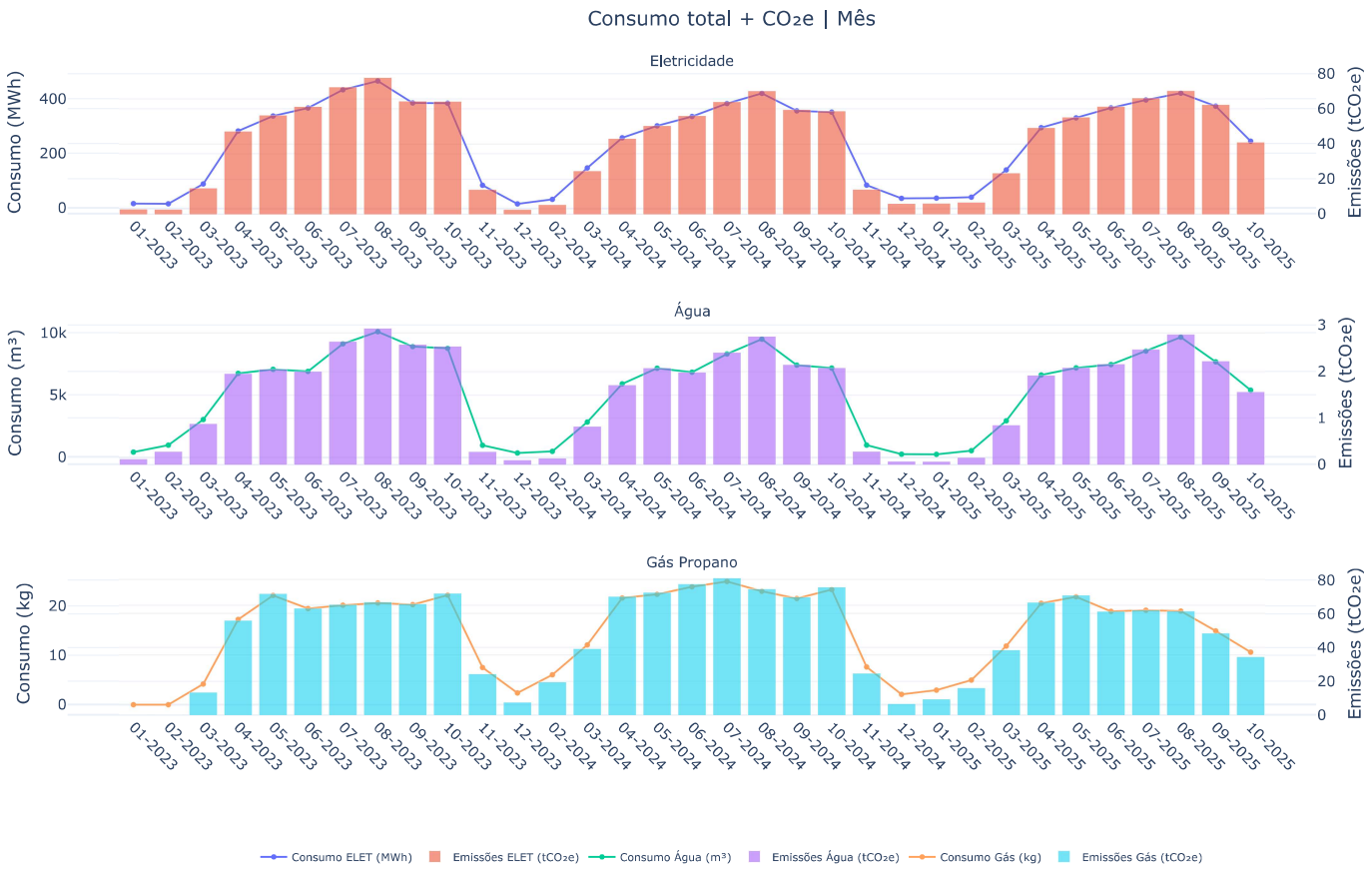
```
# Calcular consumos totais por mês
consumos_totais = dados_E.groupby(['ano','mes'])[['ELET_C','AGUA_C','GAS_C', 'CO2e_ELET', 'CO2e_AGUA', 'CO2e_GAS','Total_CO2e']].sum()
# Visualizar resultados arredondados
print("📊 Consumos totais mensais")
display(consumos_totais.round(2))
```

📊 Consumos totais mensais

ano	mes	ELET_C	AGUA_C	GAS_C	CO2e_ELET	CO2e_AGUA	CO2e_GAS	Total_CO2e
2023	1	17959.0	438.0	0.00	2999.15	127.02	0.00	0.00
	2	17230.0	994.0	0.00	2877.41	288.26	0.00	0.00
	3	89706.0	3050.0	4158.69	14980.90	884.50	13557.32	24570.21
	4	282822.0	6752.0	17140.70	47231.27	1958.08	55878.67	105068.02
	5	337540.0	7079.0	21973.81	56369.18	2052.91	71634.62	130056.71
	6	366747.0	6913.0	19332.46	61246.75	2004.77	63023.81	126275.33
	7	433337.0	9127.0	20006.84	72367.28	2646.83	65222.31	140236.42
	8	465433.0	10095.0	20456.44	77727.31	2927.55	66687.98	147342.84
	9	385157.0	8904.0	20119.24	64321.22	2582.16	65588.73	132492.11
	10	384300.0	8767.0	22030.01	64178.10	2542.43	71817.83	138538.36
	11	84866.0	984.0	7474.47	14172.62	285.36	24366.76	38824.74
	12	16894.0	364.0	2360.36	2821.30	105.56	7694.77	10621.63
2024	2	33425.0	497.0	6013.29	5581.98	144.13	19603.34	25329.44
	3	148107.0	2848.0	12026.59	24733.87	825.92	39206.67	64766.46
	4	258105.0	5909.0	21468.02	43103.54	1713.61	69985.74	114802.88
	5	301755.0	7166.0	22198.60	50393.09	2078.14	72367.45	124838.68
	6	335767.0	6847.0	23715.98	56073.09	1985.63	77314.09	135372.81
	7	383187.0	8311.0	24783.76	63992.23	2410.19	80795.05	147197.47
	8	420155.0	9499.0	22816.79	70165.89	2754.71	74382.75	147303.34
	9	356415.0	7414.0	21355.62	59521.30	2150.06	69619.32	131290.69
	10	351686.0	7179.0	23153.99	58731.56	2081.91	75482.00	136295.47
	11	85268.0	999.0	7586.86	14239.76	289.71	24733.18	39262.65
	12	37106.0	271.0	2079.36	6196.70	78.59	6778.72	10369.10
2025	1	37946.0	257.0	2922.35	6336.98	74.53	9526.85	14062.77
	2	41392.0	551.0	4945.51	6912.46	159.79	16122.37	23194.62
	3	140877.0	2948.0	11801.79	23526.46	854.92	38473.84	62855.21
	4	295058.0	6624.0	20400.24	49274.69	1920.96	66504.77	117700.42
	5	330789.0	7193.0	21692.81	55241.76	2085.97	70718.57	128046.31
	6	367188.0	7460.0	18770.47	61320.40	2163.40	61191.72	124675.52
	7	396143.0	8543.0	18995.26	66155.88	2477.47	61924.55	130557.91
	8	420908.0	9654.0	18826.66	70291.64	2799.66	61374.93	134466.22
	9	373471.0	7677.0	14836.54	62369.66	2226.33	48367.11	112963.09
	10	245662.0	5405.0	10565.41	41025.55	1567.45	34443.24	77036.25

```
import plotly.graph_objects as go
from plotly.subplots import make_subplots


consumos_totais_reset = consumos_totais.reset_index()


consumos_totais_reset['mes_ano'] = consumos_totais_reset['mes'].astype(str).str.zfill(2) + "-" + consumos_totais_reset['ano'].astype(str)

# Criar figura com 3 subplots (um por variável, com eixo secundário)
fig = make_subplots(
```

```
    rows=3, cols=1,
    subplot_titles=("Eletricidade", "Água", "Gás Propano"),
    specs=[[{"secondary_y": True}],
           [{"secondary_y": True}],
           [{"secondary_y": True}]]
)

# Subplot 1 - Eletricidade
fig.add_trace(
    go.Scatter(x=consumos_totais_reset['mes_ano'], y=consumos_totais_reset['ELET_C']/1000,
               mode='lines+markers', name="Consumo ELET (MWh)", ),
    row=1, col=1, secondary_y=False
)
fig.add_trace(
    go.Bar(x=consumos_totais_reset['mes_ano'], y=consumos_totais_reset['CO2e_ELET']/1000,
           name="Emissões ELET (tCO₂e)",  opacity=0.6),
    row=1, col=1, secondary_y=True
)

# Subplot 2 - Água
fig.add_trace(
    go.Scatter(x=consumos_totais_reset['mes_ano'], y=consumos_totais_reset['AGUA_C'],
               mode='lines+markers', name="Consumo Água (m³)",),
    row=2, col=1, secondary_y=False
)
fig.add_trace(
    go.Bar(x=consumos_totais_reset['mes_ano'], y=consumos_totais_reset['CO2e_AGUA']/1000,
           name="Emissões Água (tCO₂e)", opacity=0.6),
    row=2, col=1, secondary_y=True
)

# Subplot 3 - Gás
fig.add_trace(
    go.Scatter(x=consumos_totais_reset['mes_ano'], y=consumos_totais_reset['GAS_C']/1000,
               mode='lines+markers', name="Consumo Gás (kg)", ),
    row=3, col=1, secondary_y=False
)
fig.add_trace(
    go.Bar(x=consumos_totais_reset['mes_ano'], y=consumos_totais_reset['CO2e_GAS']/1000,
           name="Emissões Gás (tCO₂e)", opacity=0.6),
    row=3, col=1, secondary_y=True
)

# Layout
fig.update_layout(
    title="Consumo total + CO₂e | Mês",
    height=1000,
    font=dict(size=16),
    title_x=0.5,
    hovermode="x unified",
    template="plotly_white",

    # Configuração da legenda
    legend=dict(
        orientation="h",          # horizontal
        yanchor="top", y=-0.2,    # coloca abaixo do gráfico
        xanchor="center", x=0.5,  # centraliza
        font=dict(size=12)        # tamanho da fonte da legenda
    )

)
# Ajustar títulos dos eixos
fig.update_yaxes(title_text="Consumo (MWh)", row=1, col=1, secondary_y=False)
fig.update_yaxes(title_text="Emissões (tCO₂e)", row=1, col=1, secondary_y=True)

fig.update_yaxes(title_text="Consumo (m³)", row=2, col=1, secondary_y=False)
fig.update_yaxes(title_text="Emissões (tCO₂e)", row=2, col=1, secondary_y=True)

fig.update_yaxes(title_text="Consumo (kg)", row=3, col=1, secondary_y=False)
fig.update_yaxes(title_text="Emissões (tCO₂e)", row=3, col=1, secondary_y=True)
fig.update_xaxes(tickangle=45, row=1, col=1)
fig.update_xaxes(tickangle=45, row=2, col=1)
fig.update_xaxes(tickangle=45, row=3, col=1)

fig.show()
```



```
dados_E.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP'],
      dtype='object')
```

```
mensal_2023 = dados_E[dados_E['ano'] == 2023].groupby('mes')[['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']].median()
mensal_2024 = dados_E[dados_E['ano'] == 2024].groupby('mes')[['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']].median()
mensal_2025 = dados_E[dados_E['ano'] == 2025].groupby('mes')[['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']].median()
# Calcular variação percentual mês a mês
diff_percentual_mensal = ((mensal_2024 - mensal_2023) / mensal_2023) * 100
diff_percentual_mensal2025 = ((mensal_2025 - mensal_2024) / mensal_2024) * 100
diff_percentual_mensal2025_23 = ((mensal_2025 - mensal_2023) / mensal_2023) * 100
# Arredondar os valores para melhor visualização
print("📊 Variação percentual mensal entre 2024 e 2023 (%)")
display(diff_percentual_mensal.round(2))
print("📊 Variação percentual mensal entre 2025 e 2024 (%)")
display(diff_percentual_mensal2025.round(2))
print("📊 Variação percentual mensal entre 2025 e 2023 (%)")
display(diff_percentual_mensal2025_23.round(2))
```

📊 Variação percentual mensal entre 2024 e 2023 (%)

mes	ELET_C	AGUA_C	GAS_C	CLTs
1	NaN	NaN	NaN	NaN
2	93.77	41.67	NaN	NaN
3	193.09	-18.00	-18.75	NaN
4	-11.33	-19.78	20.00	-15.14
5	-13.54	3.10	8.33	12.50
6	-6.21	3.27	21.74	14.97
7	-10.10	-6.51	27.27	6.93
8	-9.64	-5.23	8.33	1.53
9	-6.73	-21.11	12.50	-1.46
10	-9.28	-19.50	7.69	-2.95
11	-19.93	50.00	-20.00	NaN
12	37.80	200.00	0.00	NaN

📊 Variação percentual mensal entre 2025 e 2024 (%)

mes	ELET_C	AGUA_C	GAS_C	CLTs
1	NaN	NaN	NaN	NaN
2	20.39	-21.57	14.29	NaN
3	-29.81	4.88	23.08	NaN
4	20.36	24.12	8.33	49.96
5	9.03	-2.15	-7.69	-9.15
6	8.26	4.22	-14.29	-9.50
7	3.23	2.56	-14.29	-13.21
8	0.24	-1.95	-7.69	-3.70
9	NaN	NaN	NaN	NaN
10	NaN	NaN	NaN	NaN
11	NaN	NaN	NaN	NaN
12	NaN	NaN	NaN	NaN

📊 Variação percentual mensal entre 2025 e 2023 (%)

mes	ELET_C	AGUA_C	GAS_C	CLTs
1	118.20	29.41	NaN	NaN
2	133.28	11.11	NaN	NaN
3	105.73	-14.00	-0.00	NaN
4	6.72	-0.43	30.00	27.26
5	-5.73	0.88	-0.00	2.21
6	1.54	7.63	4.35	4.05
7	-7.20	-4.11	9.09	-7.20
8	-9.43	-7.08	0.00	-2.22
9	NaN	NaN	NaN	NaN
10	NaN	NaN	NaN	NaN
11	NaN	NaN	NaN	NaN
12	NaN	NaN	NaN	NaN

```
dados_E.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP'],
      dtype='object')
```

```
mensal_2023 = dados_E[dados_E['ano'] == 2023].groupby('mes')[['ELET_CONS_CLT', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CLTs']].mean()
mensal_2024 = dados_E[dados_E['ano'] == 2024].groupby('mes')[['ELET_CONS_CLT', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CLTs']].mean()
# Calcular variação percentual mês a mês
diff_percentual_mensal = ((mensal_2024 - mensal_2023) / mensal_2023) * 100

# Arredondar os valores para melhor visualização
print("📊 Variação percentual mensal entre 2024 e 2023 (%)")
display(diff_percentual_mensal.round(2))
```

📊 Variação percentual mensal entre 2024 e 2023 (%)

mes	ELET_CONS_CLT	AGUA_CONS_CLT	GAS_CONS_CLT	CLTs
1	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN
3	-61.74	-76.36	-70.96	263.21
4	4.55	1.13	37.12	-11.96
5	-18.35	-7.63	-6.92	9.85
6	-15.01	-8.09	13.19	12.13
7	-15.77	-13.49	18.37	5.76
8	-11.98	-8.49	8.63	2.46
9	-4.82	-14.78	9.34	-2.29
10	-10.06	-19.43	2.82	0.04
11	-77.70	-74.67	-30.44	11.53
12	NaN	NaN	NaN	NaN

```
dados_E['ELET_CONS_CLT'] = dados_E['ELET_C'] / dados_E['CLTs']
dados_E['AGUA_CONS_CLT'] = dados_E['AGUA_C'] / dados_E['CLTs']
dados_E['GAS_CONS_CLT'] = dados_E['GAS_C'] / dados_E['CLTs']

dados_E[['ELET_CONS_CLT', 'AGUA_CONS_CLT', 'GAS_CONS_CLT']].tail()
```

DATA	ELET_CONS_CLT	AGUA_CONS_CLT	GAS_CONS_CLT
2025-08-27	10.112764	0.281911	0.528103
2025-08-28	10.565074	0.250412	0.000000
2025-08-29	11.912844	0.275229	0.721822
2025-08-30	12.184495	0.284593	0.772116
2025-08-31	11.930569	0.268081	0.541938

```
display( dados_E[['ELET_CONS_CLT', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CLTs']].loc[( dados_E['ano']==2023)].describe())
display( dados_E[['ELET_CONS_CLT', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CLTs']].loc[( dados_E['ano']==2024)].describe())
display( dados_E[['ELET_CONS_CLT', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CLTs']].loc[( dados_E['ano']==2025)].describe())
```

```
/usr/local/lib/python3.12/dist-packages/numpy/lib/_function_base_impl.py:4779: RuntimeWarning:

invalid value encountered in subtract

/usr/local/lib/python3.12/dist-packages/numpy/lib/_function_base_impl.py:4779: RuntimeWarning:

invalid value encountered in subtract
```

	ELET_CONS_CLT	AGUA_CONS_CLT	GAS_CONS_CLT	CLTs
count	324.000000	324.000000	264.000000	324.000000
mean	inf	inf	inf	660.709877
std	NaN	NaN	NaN	491.458664
min	5.751678	0.133110	0.098336	0.000000
25%	12.178365	0.262485	0.576072	0.000000
50%	13.904486	0.301605	0.714829	843.000000
75%	NaN	NaN	1.089964	1071.250000
max	inf	inf	inf	1384.000000

```
/usr/local/lib/python3.12/dist-packages/numpy/lib/_function_base_impl.py:4779: RuntimeWarning:

invalid value encountered in subtract

/usr/local/lib/python3.12/dist-packages/numpy/lib/_function_base_impl.py:4779: RuntimeWarning:

invalid value encountered in subtract
```

	ELET_CONS_CLT	AGUA_CONS_CLT	GAS_CONS_CLT	CLTs
count	307.000000	289.000000	301.000000	307.000000
mean	inf	inf	inf	724.022801
std	NaN	NaN	NaN	484.965489
min	7.674190	0.001698	0.039689	0.000000
25%	10.504352	0.223551	0.669036	0.000000
50%	11.796771	0.256303	0.907132	934.000000
75%	NaN	0.367594	NaN	1081.500000
max	inf	inf	inf	1416.000000

```
/usr/local/lib/python3.12/dist-packages/numpy/lib/_function_base_impl.py:4779: RuntimeWarning:

invalid value encountered in subtract
```

	ELET_CONS_CLT	AGUA_CONS_CLT	GAS_CONS_CLT	CLTs
count	222.000000	216.000000	215.000000	222.000000
mean	inf	inf	inf	709.864865
std	NaN	NaN	NaN	487.341138
min	7.278383	0.123333	0.000000	0.000000
25%	11.135371	0.240898	0.581169	0.000000
50%	13.207727	0.279004	0.804759	874.000000
75%	NaN	inf	3.234585	1099.500000
max	inf	inf	inf	1357.000000

```
de_2023 = dados_E.loc[
    (dados_E.index >= '2023-04-01') & (dados_E.index <= '2023-11-01'),
    ['ELET_CONS_CLT', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CLTs']
].describe()
de_2024 = dados_E.loc[
    (dados_E.index >= '2024-04-01') & (dados_E.index <= '2024-11-01'),
    ['ELET_CONS_CLT', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CLTs']
].describe()
de_2025 = dados_E.loc[
    (dados_E.index >= '2025-04-01') & (dados_E.index <= '2025-11-01'),
    ['ELET_CONS_CLT', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CLTs']
].describe()
```

```
relative_error_2024_vs_2023 = ((de_2024 - de_2023) / de_2023) * 100
relative_error_2025_vs_2023 = ((de_2025 - de_2023) / de_2023) * 100
```

```
relative_error_2025_vs_2024 = ((de_2025 - de_2024) / de_2024) * 100

print("Variação percentual (%) entre as estatisticas de 2024 e 2023")
display(relative_error_2024_vs_2023.round(2))

print("Variação percentual (%) entre as estatisticas de 2025 e 2023")
display(relative_error_2025_vs_2023.round(2))

print("Variação percentual (%) entre as estatisticas de 2025 e 2024")
display(relative_error_2025_vs_2024.round(2))
```

```
Variação percentual (%) entre as estatisticas de 2024 e 2023
```

	ELET_CONS_CLT	AGUA_CONS_CLT	GAS_CONS_CLT	CLTs
count	-0.47	-0.47	-0.47	-0.47
mean	-10.66	-10.23	11.14	2.53
std	-5.08	7.36	11.85	4.02
min	33.43	28.01	-59.64	-0.85
25%	-12.42	-11.67	13.84	7.22
50%	-11.84	-10.44	17.25	3.98
75%	-13.46	-12.35	16.12	1.14
max	-5.99	3.29	24.41	2.31

```
Variação percentual (%) entre as estatisticas de 2025 e 2023
```

	ELET_CONS_CLT	AGUA_CONS_CLT	GAS_CONS_CLT	CLTs
count	-29.30	-29.30	-29.30	-29.30
mean	-7.40	-6.81	-5.77	3.30
std	-13.28	-11.53	1.61	-7.57
min	26.54	-7.34	-100.00	54.55
25%	-9.13	-6.46	-3.95	2.65
50%	-6.48	-4.94	2.73	1.23
75%	-4.46	-6.13	2.94	6.14
max	-0.87	-16.27	-31.30	-1.95

```
Variação percentual (%) entre as estatisticas de 2025 e 2024
```

	ELET_CONS_CLT	AGUA_CONS_CLT	GAS_CONS_CLT	CLTs
count	-28.97	-28.97	-28.97	-28.97
mean	3.65	3.81	-15.21	0.75
std	-8.64	-17.59	-9.16	-11.14
min	-5.16	-27.62	-100.00	55.87
25%	3.76	5.90	-15.63	-4.26
50%	6.07	6.14	-12.39	-2.65
75%	10.40	7.10	-11.35	4.95
max	5.45	-18.94	-44.78	-4.17

```
# Agrupar mediana mensal por ano
mensal_2023 = dados_E[dados_E['ano'] == 2023].groupby('mes')[['ELET_CONS_CLT', 'AGUA_CONS_CLT', 'GAS_CONS_CLT']].median()
mensal_2024 = dados_E[dados_E['ano'] == 2024].groupby('mes')[['ELET_CONS_CLT', 'AGUA_CONS_CLT', 'GAS_CONS_CLT']].median()
mensal_2025 = dados_E[dados_E['ano'] == 2025].groupby('mes')[['ELET_CONS_CLT', 'AGUA_CONS_CLT', 'GAS_CONS_CLT']].median()
# Calcular variação percentual mês a mês
diff_percentual_mensal = ((mensal_2024 - mensal_2023) / mensal_2023) * 100
diff_percentual_mensal2025 = ((mensal_2025 - mensal_2024) / mensal_2024) * 100
diff_percentual_mensal2023 = ((mensal_2025 - mensal_2023) / mensal_2023) * 100
# Arredondar os valores para melhor visualização
print("📊 Variação percentual mensal entre 2024 e 2023 (%)")
display(diff_percentual_mensal.round(2))
print("📊 Variação percentual mensal entre 2025 e 2024 (%)")
display(diff_percentual_mensal2025.round(2))
print("📊 Variação percentual mensal entre 2025 e 2023 (%)")
display(diff_percentual_mensal2023.round(2))
```

📊 Variação percentual mensal entre 2024 e 2023 (%)

mes	ELET_CONS_CLT	AGUA_CONS_CLT	GAS_CONS_CLT
1	NaN	NaN	NaN
2	NaN	NaN	NaN
3	NaN	NaN	inf
4	1.45	0.01	29.73
5	-20.07	-6.22	-7.47
6	-15.78	-5.31	27.31
7	-14.79	-15.02	9.62
8	-11.13	-8.15	5.85
9	-4.56	-15.79	18.38
10	-5.54	-18.91	14.65
11	NaN	NaN	NaN
12	NaN	NaN	NaN

📊 Variação percentual mensal entre 2025 e 2024 (%)

mes	ELET_CONS_CLT	AGUA_CONS_CLT	GAS_CONS_CLT
1	NaN	NaN	NaN
2	NaN	NaN	NaN
3	NaN	NaN	NaN
4	-24.76	-26.51	-31.62
5	17.05	8.27	2.19
6	16.61	12.88	-19.25
7	15.17	10.51	-15.48
8	4.71	1.29	-3.64
9	NaN	NaN	NaN
10	NaN	NaN	NaN
11	NaN	NaN	NaN
12	NaN	NaN	NaN

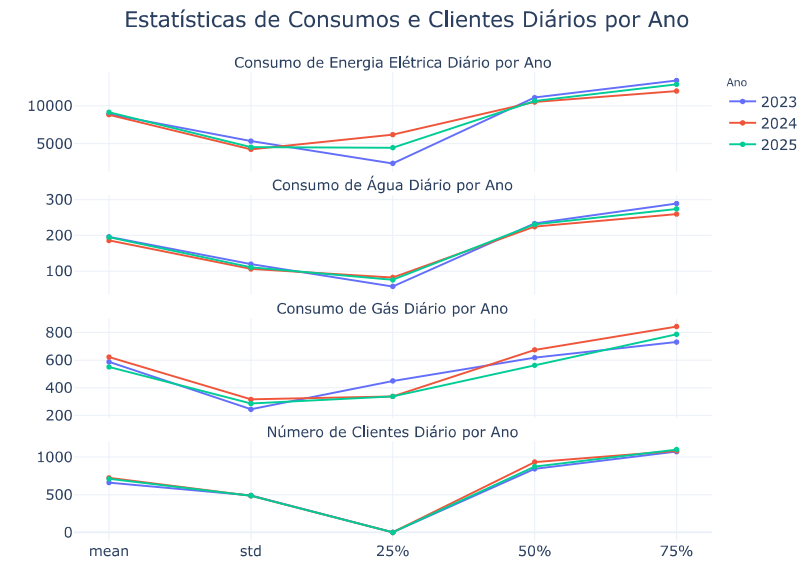
📊 Variação percentual mensal entre 2025 e 2023 (%)

mes	ELET_CONS_CLT	AGUA_CONS_CLT	GAS_CONS_CLT
1	NaN	NaN	NaN
2	NaN	NaN	NaN
3	NaN	NaN	inf
4	-23.67	-26.50	-11.29
5	-6.43	1.53	-5.44
6	-1.79	6.89	2.80
7	-1.87	-6.09	-7.35
8	-6.95	-6.97	2.00
9	NaN	NaN	NaN
10	NaN	NaN	NaN
11	NaN	NaN	NaN
12	NaN	NaN	NaN

```
de_2023=( dados_E[['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']].loc[( dados_E['ano']==2023)].describe())
de_2024=( dados_E[['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']].loc[( dados_E['ano']==2024)].describe())
de_2025=( dados_E[['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']].loc[( dados_E['ano']==2025)].describe())
```

```
import pandas as pd
import plotly.graph_objects as go
from plotly.subplots import make_subplots


df_2023 = de_2023[['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']].reset_index()
df_2024 = de_2024[['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']].reset_index()
df_2025 = de_2025[['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']].reset_index()


df_2023['Ano'] = '2023'
df_2024['Ano'] = '2024'
df_2025['Ano'] = '2025'


df = pd.concat([df_2023, df_2024, df_2025])

# Remover estatísticas agregadas
df = df[~df['index'].isin(['count', 'min', 'max'])]

# Criar subplots com espaçamento vertical
fig = make_subplots(
    rows=4, cols=1, shared_xaxes=True, vertical_spacing=0.05,
    subplot_titles=[
        'Consumo de Energia Elétrica Diário por Ano',
        'Consumo de Água Diário por Ano',
        'Consumo de Gás Diário por Ano',
        'Número de Clientes Diário por Ano'
    ]
)

# Cores por ano
cores_por_ano = {
    '2023': '#636EFA',  # azul
    '2024': '#EF553B',  # vermelho
    '2025': '#00CC96'   # verde
}

# Função modular para adicionar traços
def add_consumo_trace(fig, df, y_col, row):
    for ano in cores_por_ano.keys():
        df_ano = df[df['Ano'] == ano]
        fig.add_trace(go.Scatter(
            x=df_ano['index'],
            y=df_ano[y_col],
            mode='lines+markers',
            name=ano,
            legendgroup=y_col, # Use y_col as legendgroup for consistent legend
            line=dict(color=cores_por_ano[ano]),
            showlegend=(row == 1)
        ), row=row, col=1)
```

```
# Adicionar traços para cada variável
add_consumo_trace(fig, df, 'ELET_C', 1)
add_consumo_trace(fig, df, 'AGUA_C', 2)
add_consumo_trace(fig, df, 'GAS_C', 3)
add_consumo_trace(fig, df, 'CLTs', 4)

# Personalizar layout
fig.update_layout(
    title='Estatísticas de Consumos e Clientes Diários por Ano',
    title_x=0.5,
    width=900,
    height=700,
    font=dict(size=16),
    title_font=dict(size=24),
    legend_title=dict(text='Ano', font=dict(size=12)),
    template='plotly_white'
)

# Formatar eixo Y
fig.update_yaxes(tickformat=".0f")

# Exibir o gráfico
fig.show()
```




```
df_2023 = de_2023[['ELET_C', 'AGUA_C', 'GAS_C']]
df_2024 = de_2024[['ELET_C', 'AGUA_C', 'GAS_C']]
df_2025 = de_2025[['ELET_C', 'AGUA_C', 'GAS_C']]
display(df_2024)
```

	ELET_C	AGUA_C	GAS_C
count	307.000000	307.000000	301.000000
mean	8830.540717	185.472313	621.923153
std	4258.306892	105.753378	315.527417
min	744.000000	0.000000	56.199000
25%	6196.000000	82.000000	337.194000
50%	10487.000000	224.000000	674.388000
75%	11939.500000	259.500000	842.985000
max	20199.000000	392.000000	1854.567000

Quadros Resumo

```
dados_E.columns
```

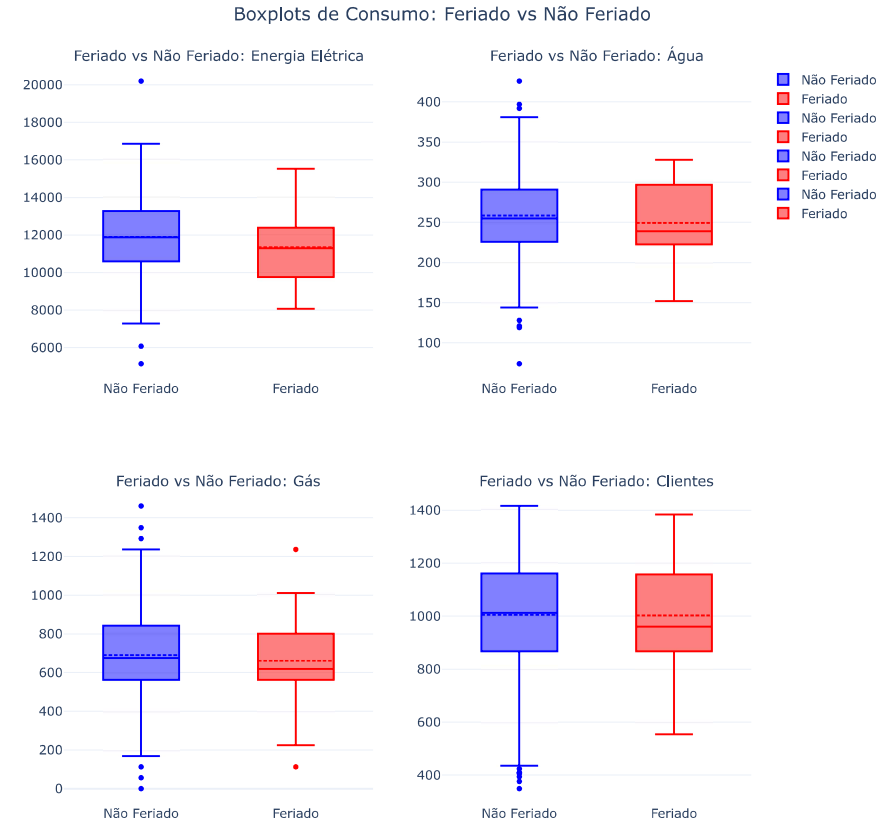
```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP'],
      dtype='object')
```

```
Comece a programar ou gerar com a IA.
```

Boxplots

```
from plotly.subplots import make_subplots
import plotly.graph_objects as go

# Variáveis e títulos
variaveis = ['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']
titulos = ['Feriado vs Não Feriado: Energia Elétrica',
           'Feriado vs Não Feriado: Água',
           'Feriado vs Não Feriado: Gás',
           'Feriado vs Não Feriado: Clientes']

# Criar figura com 2 linhas e 2 colunas
fig = make_subplots(rows=2, cols=2,
                    subplot_titles=titulos,
                    horizontal_spacing=0.1,
                    vertical_spacing=0.15)

# Posições dos subplots
posicoes = [(1, 1), (1, 2), (2, 1), (2, 2)]

# Filtrar meses de interesse (abril a outubro)
dados_filtrados = dados_E[(dados_E['mes'] >= 4) & (dados_E['mes'] < 11)]

# Loop pelas variáveis
for idx, var in enumerate(variaveis):
```

```
    row, col = posicoes[idx]

    # Boxplot para dias não feriados
    trace_nao_feriado = go.Box(
        y=dados_filtrados[dados_filtrados['feriado'] == False][var],
        name='Não Feriado',
        boxmean=True,
        marker=dict(color='blue')
    )

    # Boxplot para dias feriados
    trace_feriado = go.Box(
        y=dados_filtrados[dados_filtrados['feriado'] == True][var],
        name='Feriado',
        boxmean=True,
        marker=dict(color='red')
    )

    # Adicionar ambos os boxplots ao subplot
    fig.add_trace(trace_nao_feriado, row=row, col=col)
    fig.add_trace(trace_feriado, row=row, col=col)

# Layout final
fig.update_layout(height=1000, width=1000,
                  title_text="Boxplots de Consumo: Feriado vs Não Feriado",
                  title_x=0.5,
                  font=dict(size=14),
                  showlegend=True,
                  template="plotly_white")

fig.update_yaxes(tickformat=".0f")
fig.show()
```

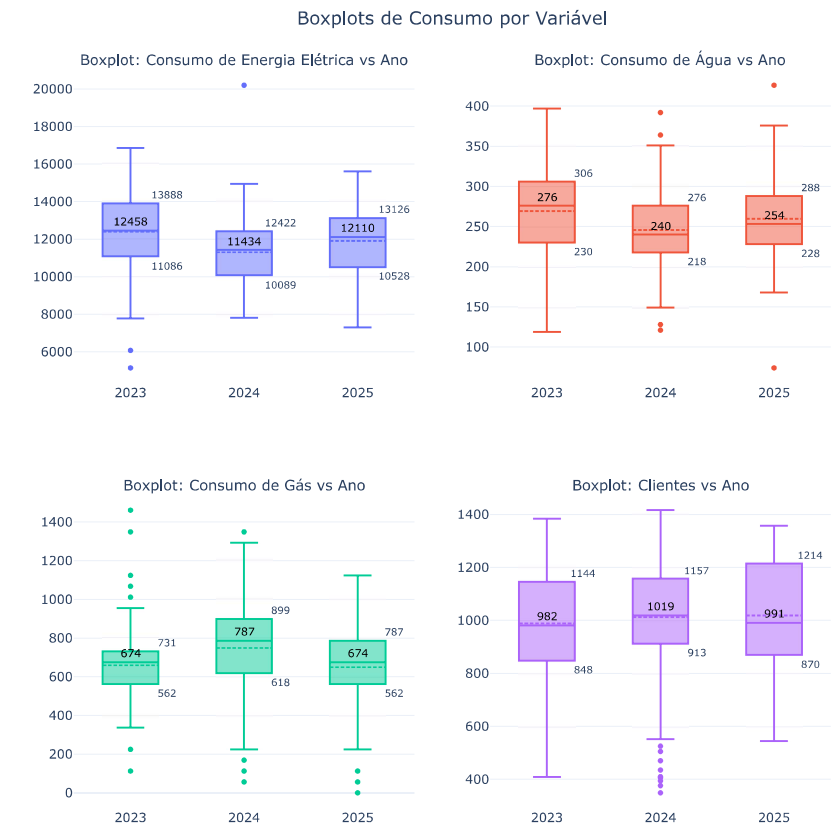



```
from plotly.subplots import make_subplots
import plotly.graph_objects as go

# Variáveis e títulos
variaveis = ['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']
anos = [2023, 2024, 2025]

# Criar figura com 3 linhas (anos) e 4 colunas (variáveis)
fig = make_subplots(
    rows=3, cols=4,
    subplot_titles=[f"{var} - {ano}" for ano in anos for var in variaveis],
    horizontal_spacing=0.05,
    vertical_spacing=0.1
)

# Loop por ano e variável
for i, ano in enumerate(anos):
    dados_ano = dados_E[
        (dados_E['ano'] == ano) &
        (dados_E['mes'] >= 4) &
        (dados_E['mes'] < 11)
    ]

    for j, var in enumerate(variaveis):
        # Boxplot para dias não feriados
        trace_nao_feriado = go.Box(
            y=dados_ano[dados_ano['feriado'] == False][var],
            name='Não Feriado',
            boxmean=True,
            marker=dict(color='blue'),
            legendgroup=f"{ano}-{var}",
            showlegend=(i == 0 and j == 0)  # Mostrar legenda apenas uma vez
        )

        # Boxplot para dias feriados
        trace_feriado = go.Box(
            y=dados_ano[dados_ano['feriado'] == True][var],
            name='Feriado',
            boxmean=True,
            marker=dict(color='red'),
            legendgroup=f"{ano}-{var}",
            showlegend=(i == 0 and j == 0)
```

```
        )

        # Adicionar ao subplot correspondente
        fig.add_trace(trace_nao_feriado, row=i+1, col=j+1)
        fig.add_trace(trace_feriado, row=i+1, col=j+1)

# Layout final
fig.update_layout(
    height=1200, width=1400,
    title_text="Boxplots de Consumo por Ano e Tipo de Dia (Feriado vs Não Feriado)",
    title_x=0.5,
    font=dict(size=13),
    template="plotly_white",
    showlegend=True
)

fig.update_yaxes(tickformat=".0f")
fig.show()
```

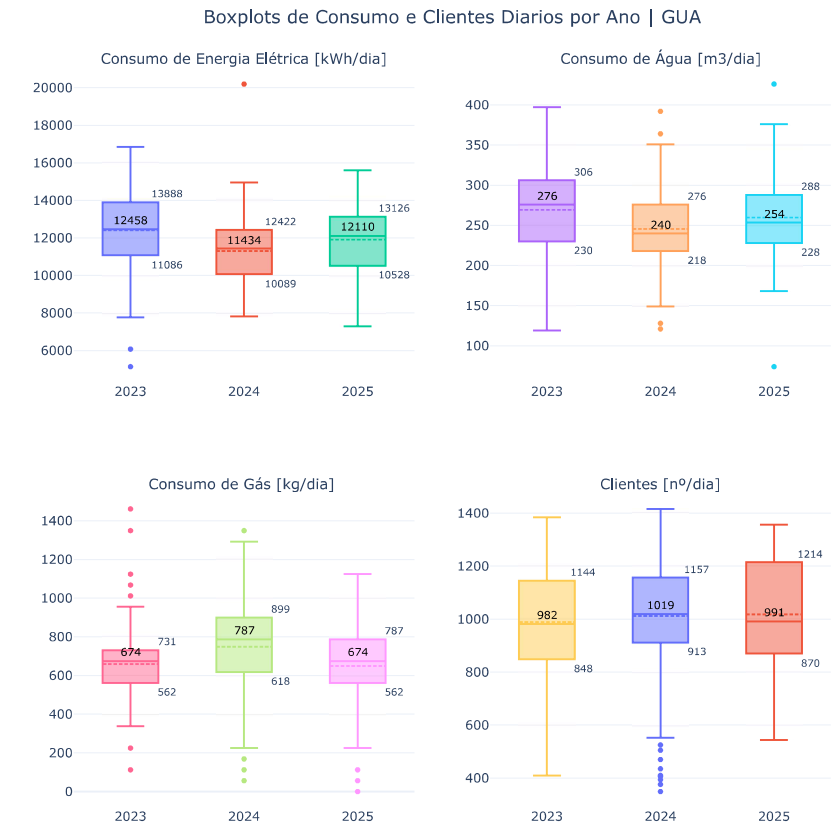

```
from plotly.subplots import make_subplots
import plotly.graph_objects as go

# Definir as variáveis e títulos dos subplots
variaveis = ['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']
titulos = ['Boxplot: Consumo de Energia Elétrica vs Ano',
           'Boxplot: Consumo de Água vs Ano',
           'Boxplot: Consumo de Gás vs Ano',
           'Boxplot: Clientes vs Ano']

# Criar a figura com 2 linhas e 2 colunas
fig = make_subplots(rows=2, cols=2,
                    subplot_titles=titulos,
                    horizontal_spacing=0.1,
                    vertical_spacing=0.15)

# Mapear posição dos plots
posicoes = [(1, 1), (1, 2), (2, 1), (2, 2)]


# Gerar gráficos individuais e adicionar à subplot
for idx, (var, titulo) in enumerate(zip(variaveis, titulos)):
    dados_filtrados = dados_E[(dados_E['mes'] >= 4) & (dados_E['mes'] < 11)]

    stats = dados_filtrados.groupby('ano')[var].describe().reset_index()

    trace = go.Box(x=dados_filtrados['ano'], y=dados_filtrados[var],
                   name=var, boxmean=True)

    row, col = posicoes[idx]
    fig.add_trace(trace, row=row, col=col)

    # Adicionar anotações de quartis
    for _, row_data in stats.iterrows():
        fig.add_annotation(x=row_data['ano'], y=row_data['50%'], text=f"{row_data['50%']:.0f}",
                           showarrow=False, font=dict(size=12, color="black"),
                           yshift=10, row=row, col=col)

        fig.add_annotation(x=row_data['ano'], y=row_data['25%'], text=f"{row_data['25%']:.0f}",
                           showarrow=False, font=dict(size=11), xshift=40, yshift=-10,
                           row=row, col=col)

        fig.add_annotation(x=row_data['ano'], y=row_data['75%'], text=f"{row_data['75%']:.0f}",
                           showarrow=False, font=dict(size=11), xshift=40, yshift=10,
                           row=row, col=col)
```

```
# Ajustar layout geral
fig.update_layout(height=1000, width=1000,
                  title_text="Boxplots de Consumo por Variável",
                  title_x=0.5,
                  font=dict(size=14),
                  showlegend=False,
                  template="plotly_white")

fig.update_yaxes(tickformat=".0f")
fig.show()
```




```
#fig.write_html("bx_diario_ano.html")
#from google.colab import files
#files.download("bx_diario_ano.html")
```

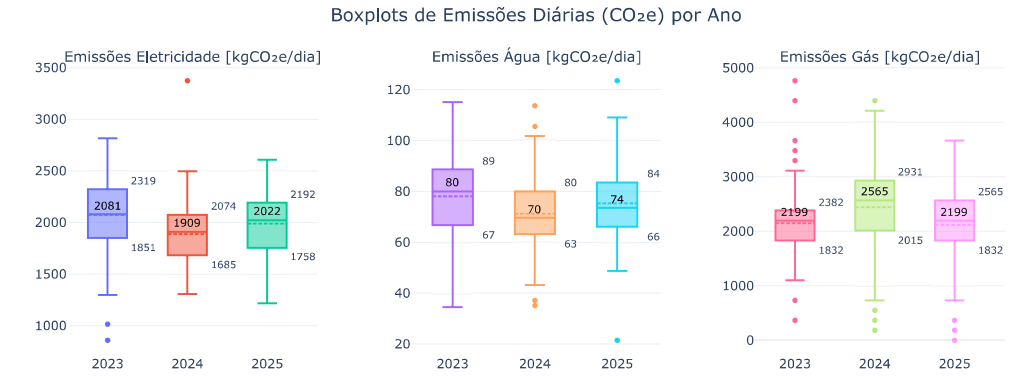
```
from plotly.subplots import make_subplots
import plotly.graph_objects as go


variaveis = ['ELET_C', 'AGUA_C', 'GAS_C', 'CLTs']
titulos = ['Consumo de Energia Elétrica [kWh/dia]',
           'Consumo de Água [m3/dia]',
           'Consumo de Gás [kg/dia] ',
           'Clientes [nº/dia]']

# Criar a figura com 2 linhas e 2 colunas
fig = make_subplots(rows=2, cols=2,
                    subplot_titles=titulos,
                    horizontal_spacing=0.1,
                    vertical_spacing=0.15)

# Mapear posição dos plots
posicoes = [(1, 1), (1, 2), (2, 1), (2, 2)]

# Gerar gráficos individuais e adicionar à subplot
for idx, (var, titulo) in enumerate(zip(variaveis, titulos)):
    dados_filtrados = dados_E[(dados_E['mes'] >= 4) & (dados_E['mes'] < 11)]

    # Calcular estatísticas por ano para anotações
    stats = dados_filtrados.groupby('ano')[var].describe().reset_index()

    # Adicionar boxplot para cada ano
    for ano in [2023, 2024, 2025]:
        dados_ano = dados_filtrados[dados_filtrados['ano'] == ano]
        trace = go.Box(x=[ano] * len(dados_ano), y=dados_ano[var],
                       name=str(ano), boxmean=True, legendgroup=var, showlegend=True if idx == 0 else False)

        row, col = posicoes[idx]
        fig.add_trace(trace, row=row, col=col)

    # Adicionar anotações de quartis para cada ano
    for _, row_data in stats.iterrows():
        fig.add_annotation(x=row_data['ano'], y=row_data['50%'], text=f"{row_data['50%']:.0f}",
                           showarrow=False, font=dict(size=12, color="black"),
                           yshift=10, row=row, col=col)

        fig.add_annotation(x=row_data['ano'], y=row_data['25%'], text=f"{row_data['25%']:.0f}",
                           showarrow=False, font=dict(size=11), xshift=40, yshift=-10,
                           row=row, col=col)

        fig.add_annotation(x=row_data['ano'], y=row_data['75%'], text=f"{row_data['75%']:.0f}",
                           showarrow=False, font=dict(size=11), xshift=40, yshift=10,
                           row=row, col=col)

# Ajustar layout geral
fig.update_layout(height=1000, width=1000,
                  title_text="Boxplots de Consumo e Clientes Diarios por Ano | GUA",
                  title_x=0.5,
                  font=dict(size=14),
                  showlegend=False, # Keep legend to differentiate years
                  template="plotly_white")

fig.update_yaxes(tickformat=".0f")
fig.show()
```



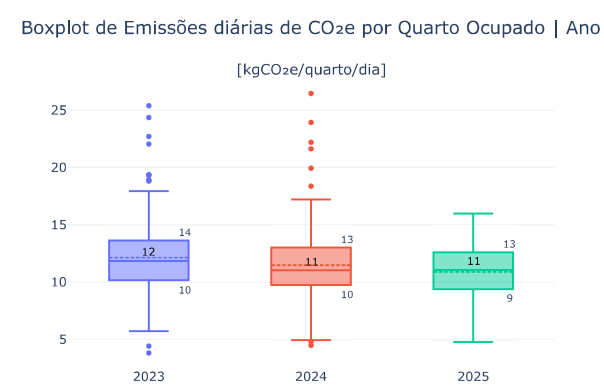

```
"fig.write_html("bx_diario_ano.html")
"from google.colab import files
"files.download("bx_diario_ano.html")
```

```
from plotly.subplots import make_subplots
import plotly.graph_objects as go


variaveis = ['CO2e_ELET', 'CO2e_AGUA', 'CO2e_GAS']
titulos = ['Emissões Eletricidade [kgCO₂e/dia]',
           'Emissões Água [kgCO₂e/dia]',
           'Emissões Gás [kgCO₂e/dia]']

# Criar a figura com 1 linha e 3 colunas
fig = make_subplots(rows=1, cols=3,
                    subplot_titles=titulos,
                    horizontal_spacing=0.1,
                    vertical_spacing=0.15)

# Mapear posição dos plots
posicoes = [(1, 1), (1, 2), (1, 3)]

# Gerar gráficos individuais e adicionar à subplot
for idx, (var, titulo) in enumerate(zip(variaveis, titulos)):
    dados_filtrados = dados_E[(dados_E['mes'] >= 4) & (dados_E['mes'] < 11)]

    # Calcular estatísticas por ano para anotações
    stats = dados_filtrados.groupby('ano')[var].describe().reset_index()

    # Adicionar boxplot para cada ano
    for ano in [2023, 2024, 2025]:
        dados_ano = dados_filtrados[dados_filtrados['ano'] == ano]
        trace = go.Box(
            x=[ano] * len(dados_ano),
            y=dados_ano[var],
            name=str(ano),
            boxmean=True,
            legendgroup=var,
            showlegend=False if idx == 0 else False
        )

        row, col = posicoes[idx]
        fig.add_trace(trace, row=row, col=col)

    # Adicionar anotações de quartis para cada ano
    for _, row_data in stats.iterrows():
        fig.add_annotation(
            x=row_data['ano'], y=row_data['50%'],
            text=f"{row_data['50%']:.0f}",
            showarrow=False, font=dict(size=12, color="black"),
            yshift=10, row=posicoes[idx][0], col=posicoes[idx][1]
        )

        fig.add_annotation(
            x=row_data['ano'], y=row_data['25%'],
            text=f"{row_data['25%']:.0f}",
            showarrow=False, font=dict(size=11),
            xshift=40, yshift=-10,
            row=posicoes[idx][0], col=posicoes[idx][1]
        )

        fig.add_annotation(
            x=row_data['ano'], y=row_data['75%'],
            text=f"{row_data['75%']:.0f}",
            showarrow=False, font=dict(size=11),
            xshift=40, yshift=10,
            row=posicoes[idx][0], col=posicoes[idx][1]
        )

# Ajustar layout geral
fig.update_layout(
    height=500, width=1200,
    title_text="Boxplots de Emissões Diárias (CO₂e) por Ano",
    title_x=0.5,
```

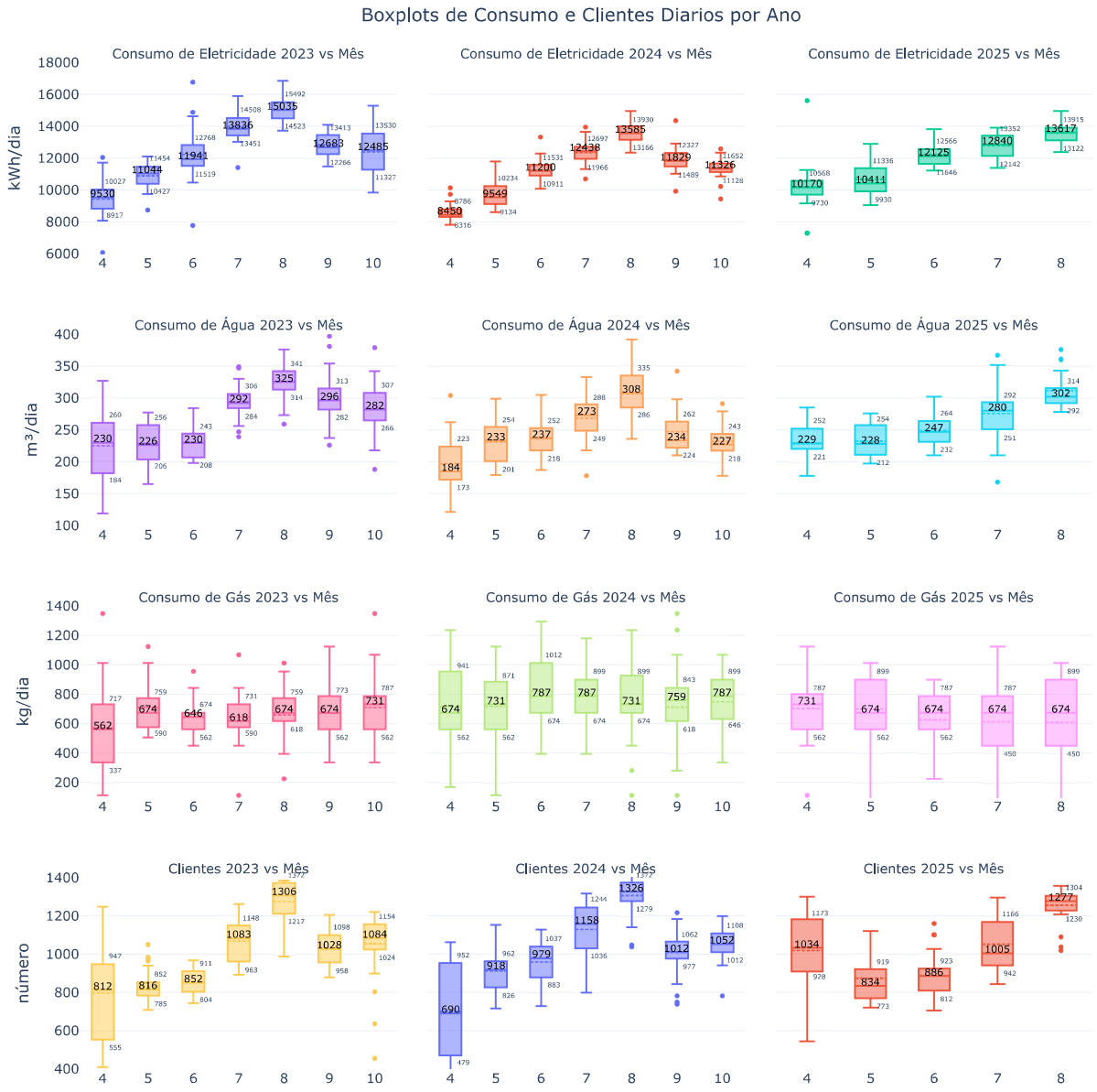
```
    font=dict(size=14),
    showlegend=True,
    template="plotly_white"
)

fig.update_yaxes(tickformat=".0f")
fig.show()
```




```
from plotly.subplots import make_subplots
import plotly.graph_objects as go


variavel = 'CO2e_QOCP'
titulo = '[kgCO₂e/quarto/dia]'


dados_filtrados = dados_E[(dados_E['mes'] >= 4) & (dados_E['mes'] < 11)]


fig = make_subplots(rows=1, cols=1,
                    subplot_titles=[titulo],
                    horizontal_spacing=0.1,
                    vertical_spacing=0.15)

# Calcular estatísticas por ano
stats = dados_filtrados.groupby('ano')[variavel].describe().reset_index()

# Adicionar boxplot por ano
for ano in [2023, 2024, 2025]:
    dados_ano = dados_filtrados[dados_filtrados['ano'] == ano]
    trace = go.Box(
        x=[ano] * len(dados_ano),
        y=dados_ano[variavel],
        name=str(ano),
        boxmean=True,
        legendgroup=variavel,
        showlegend=False
    )
    fig.add_trace(trace, row=1, col=1)

# Adicionar anotações de quartis
for _, row_data in stats.iterrows():
    fig.add_annotation(
        x=row_data['ano'], y=row_data['50%'],
        text=f"{row_data['50%']:.0f}",
        showarrow=False, font=dict(size=12, color="black"),
        yshift=10, row=1, col=1
    )
    fig.add_annotation(
        x=row_data['ano'], y=row_data['25%'],
        text=f"{row_data['25%']:.0f}",
        showarrow=False, font=dict(size=11),
        xshift=40, yshift=-10, row=1, col=1
    )
    fig.add_annotation(
        x=row_data['ano'], y=row_data['75%'],
        text=f"{row_data['75%']:.0f}",
        showarrow=False, font=dict(size=11),
        xshift=40, yshift=10, row=1, col=1
    )

# Layout final
fig.update_layout(
    height=500, width=700,
    title_text="Boxplot de Emissões diárias de CO₂e por Quarto Ocupado | Ano",
    title_x=0.5,
    font=dict(size=14),
    showlegend=True,
    template="plotly_white"
)

fig.update_yaxes(tickformat=".0f")
fig.show()
```



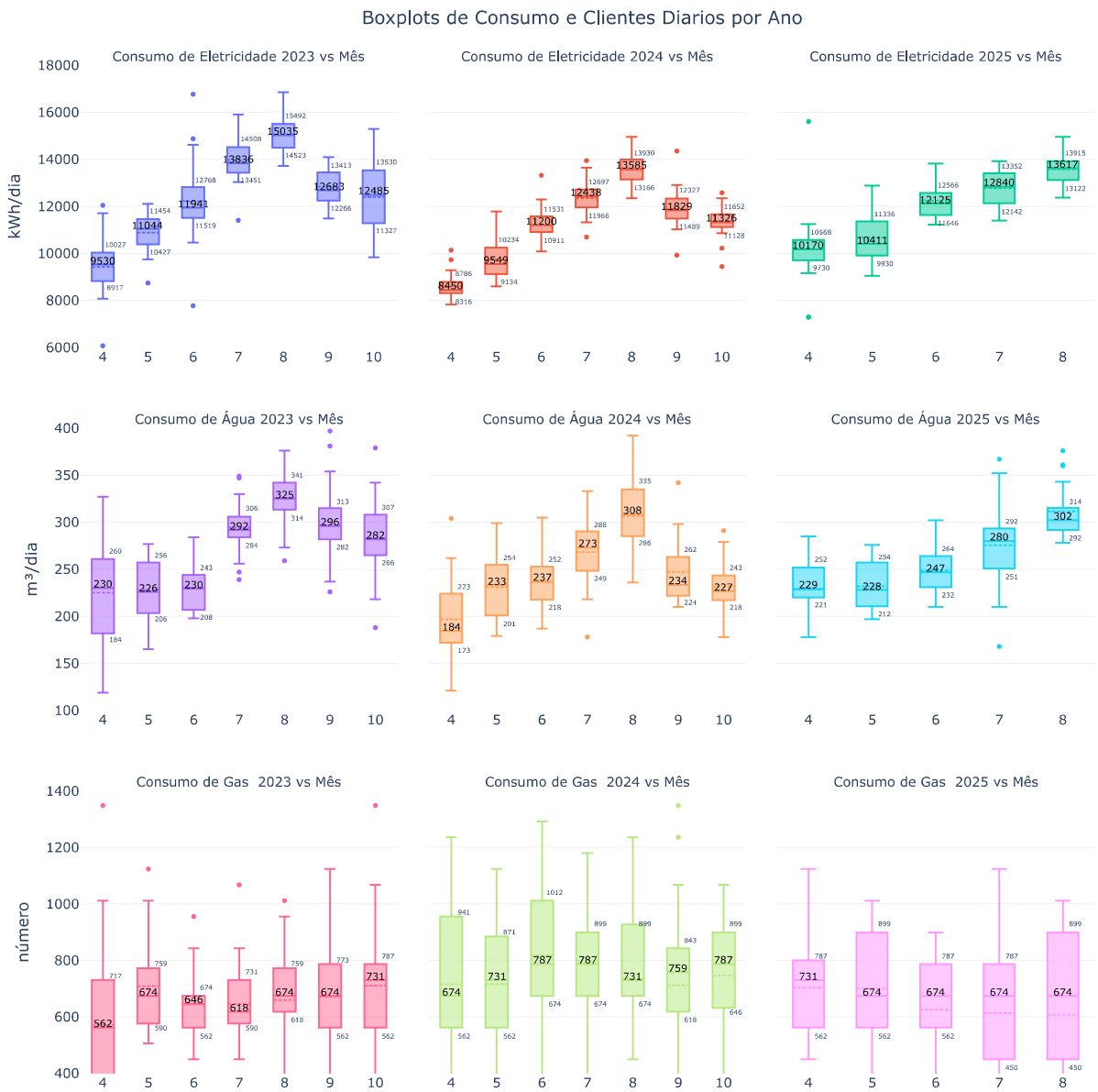
```
# Atualizar variáveis e posições para incluir o ano de 2025
variaveis = ['ELET_C', 'ELET_C', 'ELET_C',
             'AGUA_C', 'AGUA_C', 'AGUA_C',
             'GAS_C', 'GAS_C', 'GAS_C',
             'CLTs', 'CLTs', 'CLTs']

anos = [2023, 2024, 2025,
        2023, 2024, 2025,
        2023, 2024, 2025,
        2023, 2024, 2025]

titulos = ['Consumo de Eletricidade 2023 vs Mês',
           'Consumo de Eletricidade 2024 vs Mês',
           'Consumo de Eletricidade 2025 vs Mês',
           'Consumo de Água 2023 vs Mês',
           'Consumo de Água 2024 vs Mês',
           'Consumo de Água 2025 vs Mês',
           'Consumo de Gás 2023 vs Mês',
           'Consumo de Gás 2024 vs Mês',
           'Consumo de Gás 2025 vs Mês',
           'Clientes 2023 vs Mês',
           'Clientes 2024 vs Mês',
           'Clientes 2025 vs Mês']

# Define a grade 3x3 para suportar 9 gráficos (3 variáveis x 3 anos)
posicoes = [(1, 1), (1, 2), (1, 3),
            (2, 1), (2, 2), (2, 3),
            (3, 1), (3, 2), (3, 3),
            (4, 1), (4, 2), (4, 3)]

# Criar figura com 3 linhas e 3 colunas
fig = make_subplots(rows=4, cols=3,
                    subplot_titles=titulos,
                    horizontal_spacing=0.03,
                    vertical_spacing=0.08)

# Gerar gráficos e adicionar à subplot
for idx, (var, ano, (row, col)) in enumerate(zip(variaveis, anos, posicoes)):
    dados_filtrados = dados_E[(dados_E['mes'] >= 4) & (dados_E['mes'] < 11) & (dados_E['ano'] == ano)]

    trace = go.Box(x=dados_filtrados['mes'], y=dados_filtrados[var],
                   name=f"{var} {ano}", boxmean=True)

    fig.add_trace(trace, row=row, col=col)

    # Recalcular stats por mês usando os dados já filtrados
    for mes in sorted(dados_filtrados['mes'].unique()):
        stats_mes = dados_filtrados[dados_filtrados['mes'] == mes][var].describe()

        fig.add_annotation(x=mes, y=stats_mes['50%'], text=f"{stats_mes['50%']:.0f}",
                           showarrow=False, font=dict(size=12, color="black"),
                           yshift=5, row=row, col=col)

        fig.add_annotation(x=mes, y=stats_mes['25%'], text=f"{stats_mes['25%']:.0f}",
                           showarrow=False, font=dict(size=8), xshift=15, yshift=-10,
                           row=row, col=col)

        fig.add_annotation(x=mes, y=stats_mes['75%'], text=f"{stats_mes['75%']:.0f}",
                           showarrow=False, font=dict(size=8), xshift=15, yshift=10,
                           row=row, col=col)

# Layout final


# Atualizar o eixo Y apenas para os subplots de Eletricidade
fig.update_yaxes(range=[6000, 18000], row=1, col=1)
fig.update_yaxes(range=[6000, 18000], row=1, col=2)
fig.update_yaxes(range=[6000, 18000], row=1, col=3)
fig.update_yaxes(range=[100, 400], row=2, col=1)
fig.update_yaxes(range=[100, 400], row=2, col=2)
fig.update_yaxes(range=[100, 400], row=2, col=3)
fig.update_yaxes(range=[100, 1400], row=3, col=1)
fig.update_yaxes(range=[100, 1400], row=3, col=2)
fig.update_yaxes(range=[100, 1400], row=3, col=3)
fig.update_yaxes(range=[400, 1400], row=4, col=1)
fig.update_yaxes(range=[400, 1400], row=4, col=2)
fig.update_yaxes(range=[400, 1400], row=4, col=3)
fig.update_yaxes(showticklabels=False, row=1, col=2)
fig.update_yaxes(showticklabels=False, row=2, col=2)
fig.update_yaxes(showticklabels=False, row=3, col=2)
fig.update_yaxes(showticklabels=False, row=4, col=2)
fig.update_yaxes(showticklabels=False, row=1, col=3)
fig.update_yaxes(showticklabels=False, row=2, col=3)
fig.update_yaxes(showticklabels=False, row=3, col=3)
fig.update_yaxes(showticklabels=False, row=4, col=3)
fig.update_layout(height=1400, width=1400,
                  title_text="Boxplots de Consumo e Clientes Diarios por Ano",
                  title_x=0.5,
                  font=dict(size=16),
                  showlegend=False,
                  template="plotly_white")
# Garantir que os meses de 4 a 10 apareçam no eixo X de todos os subplots
fig.update_xaxes(tickvals=list(range(4, 11)), ticktext=[str(m) for m in range(4, 11)])
# Linha 1 (Energia): kWh/dia
fig.update_yaxes(title_text="kWh/dia", row=1, col=1)
#fig.update_yaxes(title_text="kWh/dia", row=1, col=2)
#fig.update_yaxes(title_text="kWh/dia", row=1, col=3)

# Linha 2 (Água): m³/dia
fig.update_yaxes(title_text="m³/dia", row=2, col=1)
#fig.update_yaxes(title_text="m³/dia", row=2, col=2)
#fig.update_yaxes(title_text="m³/dia", row=2, col=3)

# Linha 3 (Gás ou Clientes): kg/dia
fig.update_yaxes(title_text="kg/dia", row=3, col=1)
#fig.update_yaxes(title_text="kg/dia", row=3, col=2)
#fig.update_yaxes(title_text="kg/dia", row=3, col=3)

# Linha 3 (Clientes ou Ocupação): número
fig.update_yaxes(title_text="número", row=4, col=1)
# fig.update_yaxes(title_text="número", row=3, col=2)
# fig.update_yaxes(title_text="número", row=3, col=3)


fig.update_yaxes(tickformat=".0f")
fig.show()
```




```
fig.write_html("bx_diario_mes.html")
from google.colab import files
files.download("bx_diario_mes.html")
```

```
# Atualizar variáveis e posições para incluir o ano de 2025
variaveis = ['ELET_C', 'ELET_C', 'ELET_C',
             'AGUA_C', 'AGUA_C', 'AGUA_C',
             'GAS_C', 'GAS_C', 'GAS_C']

anos = [2023, 2024, 2025,
        2023, 2024, 2025,
        2023, 2024, 2025]

titulos = ['Consumo de Eletricidade 2023 vs Mês',
           'Consumo de Eletricidade 2024 vs Mês',
           'Consumo de Eletricidade 2025 vs Mês',
           'Consumo de Água 2023 vs Mês',
           'Consumo de Água 2024 vs Mês',
           'Consumo de Água 2025 vs Mês',
           'Consumo de Gas  2023 vs Mês',
           'Consumo de Gas  2024 vs Mês',
           'Consumo de Gas  2025 vs Mês']

# Define a grade 3x3 para suportar 9 gráficos (3 variáveis x 3 anos)
posicoes = [(1, 1), (1, 2), (1, 3),
            (2, 1), (2, 2), (2, 3),
            (3, 1), (3, 2), (3, 3)]

# Criar figura com 3 linhas e 3 colunas
fig = make_subplots(rows=3, cols=3,
                    subplot_titles=titulos,
                    horizontal_spacing=0.03,
                    vertical_spacing=0.08)

# Gerar gráficos e adicionar à subplot
for idx, (var, ano, (row, col)) in enumerate(zip(variaveis, anos, posicoes)):
    dados_filtrados = dados_E[(dados_E['mes'] >= 4) & (dados_E['mes'] < 11) & (dados_E['ano'] == ano)]

    trace = go.Box(x=dados_filtrados['mes'], y=dados_filtrados[var],
                   name=f"{var} {ano}", boxmean=True)

    fig.add_trace(trace, row=row, col=col)

    # Recalcular stats por mês usando os dados já filtrados
    for mes in sorted(dados_filtrados['mes'].unique()):
        stats_mes = dados_filtrados[dados_filtrados['mes'] == mes][var].describe()

        fig.add_annotation(x=mes, y=stats_mes['50%'], text=f"{stats_mes['50%']:.0f}",
                           showarrow=False, font=dict(size=12, color="black"),
                           yshift=5, row=row, col=col)

        fig.add_annotation(x=mes, y=stats_mes['25%'], text=f"{stats_mes['25%']:.0f}",
                           showarrow=False, font=dict(size=8), xshift=15, yshift=-10,
```

```
                        row=row, col=col)

        fig.add_annotation(x=mes, y=stats_mes['75%'], text=f"{stats_mes['75%']:.0f}",
                        showarrow=False, font=dict(size=8), xshift=15, yshift=10,
                        row=row, col=col)

# Layout final


# Atualizar o eixo Y apenas para os subplots de Eletricidade
fig.update_yaxes(range=[6000, 18000], row=1, col=1)
fig.update_yaxes(range=[6000, 18000], row=1, col=2)
fig.update_yaxes(range=[6000, 18000], row=1, col=3)
fig.update_yaxes(range=[100, 400], row=2, col=1)
fig.update_yaxes(range=[100, 400], row=2, col=2)
fig.update_yaxes(range=[100, 400], row=2, col=3)
fig.update_yaxes(range=[400, 1400], row=3, col=1)
fig.update_yaxes(range=[400, 1400], row=3, col=2)
fig.update_yaxes(range=[400, 1400], row=3, col=3)
fig.update_yaxes(showticklabels=False, row=1, col=2)
fig.update_yaxes(showticklabels=False, row=2, col=2)
fig.update_yaxes(showticklabels=False, row=3, col=2)
fig.update_yaxes(showticklabels=False, row=1, col=3)
fig.update_yaxes(showticklabels=False, row=2, col=3)
fig.update_yaxes(showticklabels=False, row=3, col=3)
fig.update_layout(height=1400, width=1400,
                  title_text="Boxplots de Consumo e Clientes Diarios por Ano",
                  title_x=0.5,
                  font=dict(size=16),
                  showlegend=False,
                  template="plotly_white")
# Garantir que os meses de 4 a 10 apareçam no eixo X de todos os subplots
fig.update_xaxes(tickvals=list(range(4, 11)), ticktext=[str(m) for m in range(4, 11)])
# Linha 1 (Energia): kWh/dia
fig.update_yaxes(title_text="kWh/dia", row=1, col=1)
#fig.update_yaxes(title_text="kWh/dia", row=1, col=2)
#fig.update_yaxes(title_text="kWh/dia", row=1, col=3)

# Linha 2 (Água): m³/dia
fig.update_yaxes(title_text="m³/dia", row=2, col=1)
#fig.update_yaxes(title_text="m³/dia", row=2, col=2)
#fig.update_yaxes(title_text="m³/dia", row=2, col=3)

# Linha 3 (Gás ou Clientes): kg/dia
#fig.update_yaxes(title_text="kg/dia", row=3, col=1)
#fig.update_yaxes(title_text="kg/dia", row=3, col=2)
#fig.update_yaxes(title_text="kg/dia", row=3, col=3)

# Linha 3 (Clientes ou Ocupação): número (caso venhas a adicionar)
fig.update_yaxes(title_text="número", row=3, col=1)
# fig.update_yaxes(title_text="número", row=3, col=2)
# fig.update_yaxes(title_text="número", row=3, col=3)


fig.update_yaxes(tickformat=".0f")
fig.show()
```



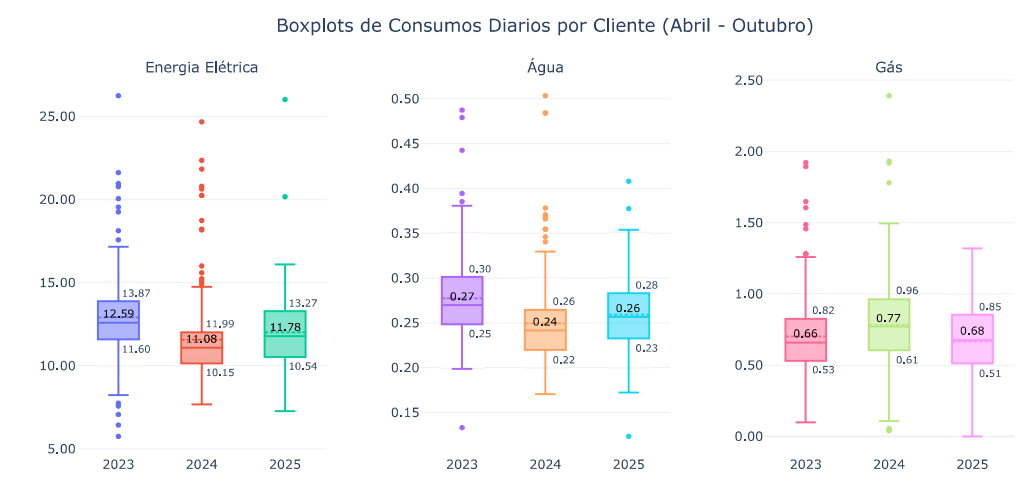
```
from plotly.subplots import make_subplots
import plotly.graph_objects as go

variaveis = ['ELET_C', 'ELET_C','AGUA_C', 'AGUA_C',  'CLTs', 'CLTs']
titulos = ['Boxplot: Consumo de Eletricidade 2023 vs Mês',
           'Boxplot: Consumo de Eletricidade 2024 vs Mês',
           'Boxplot: Consumo de Água 2023 vs Mês',
           'Boxplot: Consumo de Água 2024 vs Mês',

           'Boxplot: Clientes 2023 vs Mês',
           'Boxplot: Clientes 2024 vs Mês']


# Criar figura com 3 linhas e 2 colunas
fig = make_subplots(rows=3, cols=2,
                    subplot_titles=titulos,
                    horizontal_spacing=0.05,
                    vertical_spacing=0.12)

# Variáveis e posições específicas
anos = [2023, 2024, 2023, 2024, 2023, 2024]
posicoes = [(1, 1), (1, 2), (2, 1), (2, 2), (3, 1), (3, 2)]

# Gerar gráficos e adicionar à subplot
for idx, (var, ano, (row, col)) in enumerate(zip(variaveis, anos, posicoes)):
    dados_filtrados = dados_E[(dados_E['mes'] >= 4) & (dados_E['mes'] < 11) & (dados_E['ano'] == ano)]

    trace = go.Box(x=dados_filtrados['mes'], y=dados_filtrados[var],
                   name=f"{var} {ano}", boxmean=True)

    fig.add_trace(trace, row=row, col=col)

    # Recalcular stats por mês usando os dados já filtrados
    for mes in sorted(dados_filtrados['mes'].unique()):
        stats_mes = dados_filtrados[dados_filtrados['mes'] == mes][var].describe()

        fig.add_annotation(x=mes, y=stats_mes['50%'], text=f"{stats_mes['50%']:.0f}",
                           showarrow=False, font=dict(size=12, color="black"),
                           yshift=5, row=row, col=col)

        fig.add_annotation(x=mes, y=stats_mes['25%'], text=f"{stats_mes['25%']:.0f}",
                           showarrow=False, font=dict(size=8), xshift=15, yshift=-10,
                           row=row, col=col)

        fig.add_annotation(x=mes, y=stats_mes['75%'], text=f"{stats_mes['75%']:.0f}",
                           showarrow=False, font=dict(size=8), xshift=15, yshift=10,
                           row=row, col=col)

# Layout final

# Atualizar o eixo Y apenas para os subplots de Eletricidade
fig.update_yaxes(range=[6000, 18000], row=1, col=1)
fig.update_yaxes(range=[6000, 18000], row=1, col=2)
fig.update_yaxes(showticklabels=False, row=1, col=2)
fig.update_yaxes(showticklabels=False, row=2, col=2)
fig.update_yaxes(showticklabels=False, row=3, col=2)

fig.update_layout(height=1200, width=1200,
                  title_text="Boxplots de Consumo e Clientes Diarios por Ano",
                  title_x=0.5,
                  font=dict(size=16),
                  showlegend=False,
                  template="plotly_white")
# Garantir que os meses de 4 a 10 apareçam no eixo X de todos os subplots
fig.update_xaxes(tickvals=list(range(4, 11)), ticktext=[str(m) for m in range(4, 11)])

# Linha 1 (Energia): kWh/dia
fig.update_yaxes(title_text="kWh/dia", row=1, col=1)
#fig.update_yaxes(title_text="kWh/dia", row=1, col=2)
#fig.update_yaxes(title_text="kWh/dia", row=1, col=3)

# Linha 2 (Água): m³/dia
fig.update_yaxes(title_text="m³/dia", row=2, col=1)
#fig.update_yaxes(title_text="m³/dia", row=2, col=2)
#fig.update_yaxes(title_text="m³/dia", row=2, col=3)

# Linha 3 (Gás ou Clientes): kg/dia
#fig.update_yaxes(title_text="kg/dia", row=3, col=1)
#fig.update_yaxes(title_text="kg/dia", row=3, col=2)
#fig.update_yaxes(title_text="kg/dia", row=3, col=3)

# Linha 3 (Clientes ou Ocupação): número (caso venhas a adicionar)
fig.update_yaxes(title_text="número", row=3, col=1)
# fig.update_yaxes(title_text="número", row=3, col=2)
# fig.update_yaxes(title_text="número", row=3, col=3)
fig.update_yaxes(tickformat=".0f")
fig.show()
```


```
dados_E.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP'],
      dtype='object')
```

```
from plotly.subplots import make_subplots
import plotly.graph_objects as go


variaveis = ['ELET_CONS_CLT', 'AGUA_CONS_CLT', 'GAS_CONS_CLT']
titulos = ['Energia Elétrica',
           'Água',
           'Gás']

# Criar a figura com 1 linhas e 3 colunas
fig = make_subplots(rows=1, cols=3,
                    subplot_titles=titulos,
                    horizontal_spacing=0.1,
                    vertical_spacing=0.15)

# Mapear posição dos plots
posicoes = [(1, 1), (1, 2), (1, 3)]

# Gerar gráficos individuais e adicionar à subplot
for idx, (var, titulo) in enumerate(zip(variaveis, titulos)):
    dados_filtrados = dados_E[(dados_E['mes'] >= 4) & (dados_E['mes'] < 11)]

    # Calcular estatísticas por ano para anotações
    stats = dados_filtrados.groupby('ano')[var].describe().reset_index()

    # Adicionar boxplot para cada ano
    for ano in [2023, 2024, 2025]:
        dados_ano = dados_filtrados[dados_filtrados['ano'] == ano]
        trace = go.Box(x=[ano] * len(dados_ano), y=dados_ano[var],
                       name=str(ano), boxmean=True, legendgroup=var, showlegend=True if idx == 0 else False)

        row, col = posicoes[idx]
        fig.add_trace(trace, row=row, col=col)

    # Adicionar anotações de quartis para cada ano
    for _, row_data in stats.iterrows():
        fig.add_annotation(x=row_data['ano'], y=row_data['50%'], text=f"{row_data['50%']:.2f}",
                           showarrow=False, font=dict(size=12, color="black"),
                           yshift=10, row=row, col=col)

        fig.add_annotation(x=row_data['ano'], y=row_data['25%'], text=f"{row_data['25%']:.2f}",
                           showarrow=False, font=dict(size=11), xshift=20, yshift=-10,
                           row=row, col=col)

        fig.add_annotation(x=row_data['ano'], y=row_data['75%'], text=f"{row_data['75%']:.2f}",
                           showarrow=False, font=dict(size=11), xshift=20, yshift=10,
                           row=row, col=col)

# Ajustar layout geral
fig.update_layout(height=600, width=1200,
                  title_text="Boxplots de Consumos Diarios por Cliente (Abril - Outubro)",
                  title_x=0.5,
                  font=dict(size=14),
                  showlegend=False, # Keep legend to differentiate years
```

```
                  template="plotly_white")

fig.update_yaxes(tickformat=".2f")
fig.show()
```

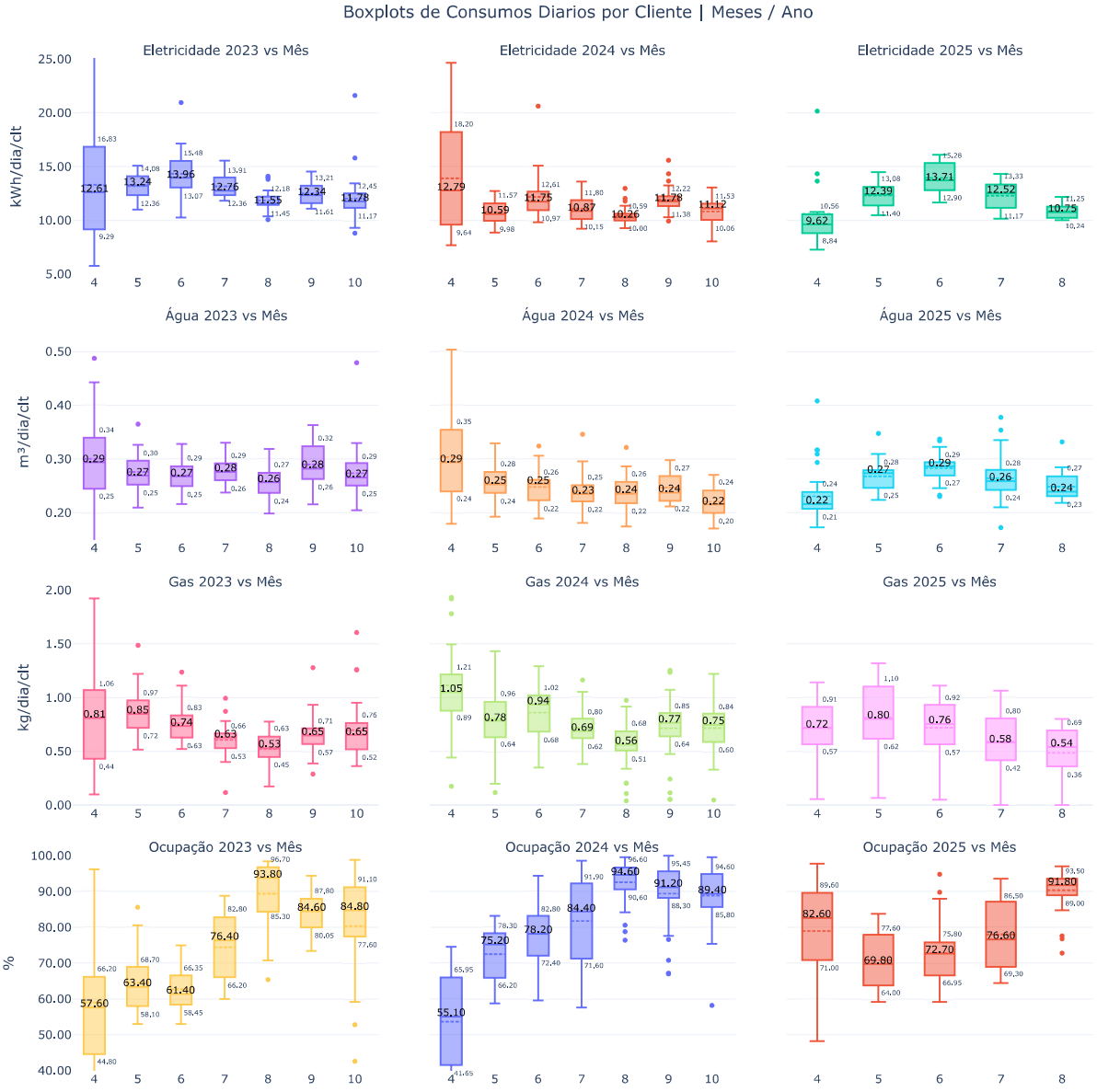



```
variaveis = ['ELET_CONS_CLT', 'ELET_CONS_CLT', 'ELET_CONS_CLT',
             'AGUA_CONS_CLT','AGUA_CONS_CLT','AGUA_CONS_CLT',
             'GAS_CONS_CLT', 'GAS_CONS_CLT', 'GAS_CONS_CLT']

anos = [2023, 2024, 2025,
        2023, 2024, 2025,
        2023, 2024, 2025]

titulos = ['Eletricidade 2023 vs Mês',
           'Eletricidade 2024 vs Mês',
           'Eletricidade 2025 vs Mês',
           'Água 2023 vs Mês',
           'Água 2024 vs Mês',
           'Água 2025 vs Mês',
           'Gas 2023 vs Mês',
           'Gas 2024 vs Mês',
           'Gas 2025 vs Mês']

# Define a grade 3x3 para suportar 9 gráficos (3 variáveis x 3 anos)
posicoes = [(1, 1), (1, 2), (1, 3),
            (2, 1), (2, 2), (2, 3),
            (3, 1), (3, 2), (3, 3)]

# Criar figura com 3 linhas e 3 colunas
fig = make_subplots(rows=3, cols=3,
                    subplot_titles=titulos,
                    horizontal_spacing=0.07,
                    vertical_spacing=0.12)

# Gerar gráficos e adicionar à subplot
for idx, (var, ano, (row, col)) in enumerate(zip(variaveis, anos, posicoes)):
    dados_filtrados = dados_E[(dados_E['mes'] >= 4) & (dados_E['mes'] < 11) & (dados_E['ano'] == ano)]

    trace = go.Box(x=dados_filtrados['mes'], y=dados_filtrados[var],
                   name=f"{var} {ano}", boxmean=True)

    fig.add_trace(trace, row=row, col=col)

    # Recalcular stats por mês usando os dados já filtrados
    for mes in sorted(dados_filtrados['mes'].unique()):
        stats_mes = dados_filtrados[dados_filtrados['mes'] == mes][var].describe()

        fig.add_annotation(x=mes, y=stats_mes['50%'], text=f"{stats_mes['50%']:.2f}",
                           showarrow=False, font=dict(size=10, color="black"),
                           yshift=5, row=row, col=col)

        fig.add_annotation(x=mes, y=stats_mes['25%'], text=f"{stats_mes['25%']:.2f}",
                           showarrow=False, font=dict(size=8), xshift=15, yshift=-10,
                           row=row, col=col)

        fig.add_annotation(x=mes, y=stats_mes['75%'], text=f"{stats_mes['75%']:.2f}",
                           showarrow=False, font=dict(size=8), xshift=15, yshift=10,
                           row=row, col=col)

# Layout final


# Atualizar o eixo Y apenas para os subplots de Eletricidade
"""fig.update_yaxes(range=[6000, 18000], row=1, col=1)
fig.update_yaxes(range=[6000, 18000], row=1, col=2)
fig.update_yaxes(range=[6000, 18000], row=1, col=3)
fig.update_yaxes(range=[100, 400], row=2, col=1)
fig.update_yaxes(range=[100, 400], row=2, col=2)
fig.update_yaxes(range=[100, 400], row=2, col=3)
fig.update_yaxes(range=[400, 1400], row=3, col=1)
fig.update_yaxes(range=[400, 1400], row=3, col=2)
fig.update_yaxes(range=[400, 1400], row=3, col=3)"""
fig.update_layout(height=1400, width=1400,
                  title_text="Boxplots de Consumos Diarios por Cliente | Meses / Ano",
                  title_x=0.5,
                  font=dict(size=14),
                  showlegend=False,
                  template="plotly_white")
# Garantir que os meses de 4 a 10 apareçam no eixo X de todos os subplots
fig.update_xaxes(tickvals=list(range(4, 11)), ticktext=[str(m) for m in range(4, 11)])
# Linha 1 (Energia): kWh/dia
fig.update_yaxes(title_text="kWh/dia", row=1, col=1)
#fig.update_yaxes(title_text="kWh/dia", row=1, col=2)
#fig.update_yaxes(title_text="kWh/dia", row=1, col=3)

# Linha 2 (Água): m³/dia
fig.update_yaxes(title_text="m³/dia", row=2, col=1)
#fig.update_yaxes(title_text="m³/dia", row=2, col=2)
#fig.update_yaxes(title_text="m³/dia", row=2, col=3)

# Linha 3 (Gás ou Gas): kg/dia
#fig.update_yaxes(title_text="kg/dia", row=3, col=1)
#fig.update_yaxes(title_text="kg/dia", row=3, col=2)
#fig.update_yaxes(title_text="kg/dia", row=3, col=3)

# Linha 3 (Gas ou Ocupação): número (caso venhas a adicionar)
fig.update_yaxes(title_text="número", row=3, col=1)
# fig.update_yaxes(title_text="número", row=3, col=2)
```

```
# fig.update_yaxes(title_text="número", row=3, col=3)


fig.update_yaxes(tickformat=".2f")
fig.show()
```

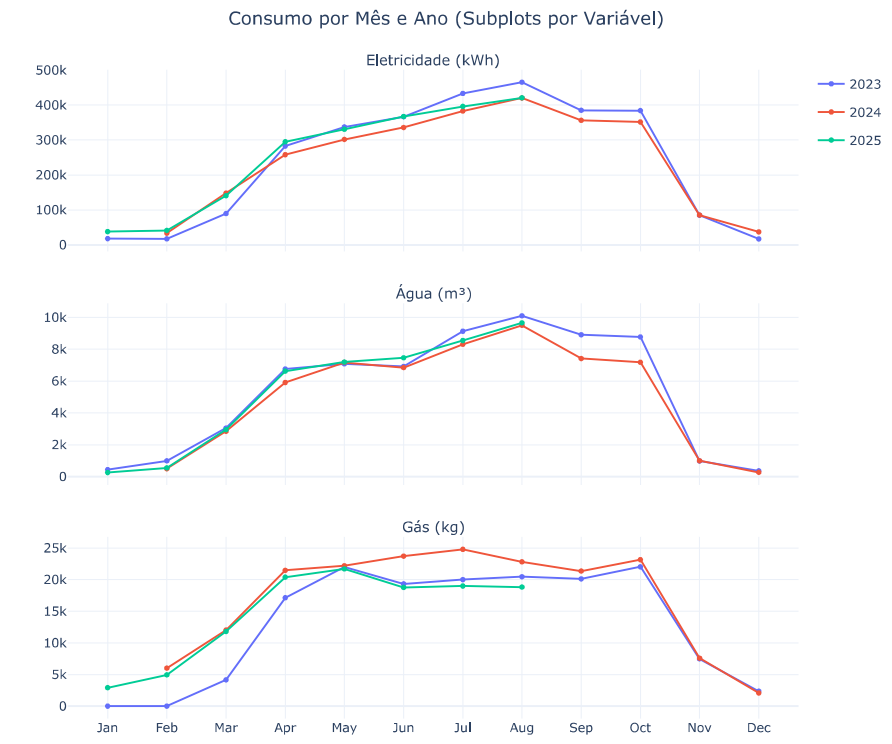

```
variaveis = ['ELET_CONS_CLT', 'ELET_CONS_CLT', 'ELET_CONS_CLT',
             'AGUA_CONS_CLT','AGUA_CONS_CLT','AGUA_CONS_CLT',
             'GAS_CONS_CLT', 'GAS_CONS_CLT', 'GAS_CONS_CLT',
             'OCP[%]', 'OCP[%]', 'OCP[%]']

anos = [2023, 2024, 2025,
        2023, 2024, 2025,
        2023, 2024, 2025,
        2023, 2024, 2025]

titulos = ['Eletricidade 2023 vs Mês',
           'Eletricidade 2024 vs Mês',
           'Eletricidade 2025 vs Mês',
           'Água 2023 vs Mês',
           'Água 2024 vs Mês',
           'Água 2025 vs Mês',
           'Gas 2023 vs Mês',
           'Gas 2024 vs Mês',
           'Gas 2025 vs Mês',
            'Ocupação 2023 vs Mês',
           'Ocupação 2024 vs Mês',
           'Ocupação 2025 vs Mês']

# Define a grade 3x3 para suportar 9 gráficos (3 variáveis x 3 anos)
posicoes = [(1, 1), (1, 2), (1, 3),
            (2, 1), (2, 2), (2, 3),
            (3, 1), (3, 2), (3, 3),
            (4, 1), (4, 2), (4, 3)]

# Criar figura com 3 linhas e 3 colunas
fig = make_subplots(rows=4, cols=3,
                    subplot_titles=titulos,
                    horizontal_spacing=0.05,
                    vertical_spacing=0.05)

# Gerar gráficos e adicionar à subplot
for idx, (var, ano, (row, col)) in enumerate(zip(variaveis, anos, posicoes)):
    dados_filtrados = dados_E[(dados_E['mes'] >= 4) & (dados_E['mes'] < 11) & (dados_E['ano'] == ano)]

    trace = go.Box(x=dados_filtrados['mes'], y=dados_filtrados[var],
                   name=f"{var} {ano}", boxmean=True)

    fig.add_trace(trace, row=row, col=col)

    # Recalcular stats por mês usando os dados já filtrados
    for mes in sorted(dados_filtrados['mes'].unique()):
        stats_mes = dados_filtrados[dados_filtrados['mes'] == mes][var].describe()

        fig.add_annotation(x=mes, y=stats_mes['50%'], text=f"{stats_mes['50%']:.2f}",
                           showarrow=False, font=dict(size=12, color="black"),
```

```
                        yshift=5, row=row, col=col)

        fig.add_annotation(x=mes, y=stats_mes['25%'], text=f"{stats_mes['25%']:.2f}",
                        showarrow=False, font=dict(size=8), xshift=15, yshift=-10,
                        row=row, col=col)

        fig.add_annotation(x=mes, y=stats_mes['75%'], text=f"{stats_mes['75%']:.2f}",
                        showarrow=False, font=dict(size=8), xshift=15, yshift=10,
                        row=row, col=col)

# Layout final


# Atualizar o eixo Y apenas para os subplots de Eletricidade
fig.update_yaxes(range=[5, 25], row=1, col=1)
fig.update_yaxes(range=[5, 25], row=1, col=2)
fig.update_yaxes(range=[5, 25], row=1, col=3)
fig.update_yaxes(range=[0.15, 0.55], row=2, col=1)
fig.update_yaxes(range=[0.15, 0.55], row=2, col=2)
fig.update_yaxes(range=[0.15, 0.55], row=2, col=3)
fig.update_yaxes(range=[0, 2], row=3, col=1)
fig.update_yaxes(range=[0, 2], row=3, col=2)
fig.update_yaxes(range=[0, 2], row=3, col=3)
fig.update_yaxes(range=[40, 100], row=4, col=1)
fig.update_yaxes(range=[40, 100], row=4, col=2)
fig.update_yaxes(range=[40, 100], row=4, col=3)
fig.update_yaxes(showticklabels=False, row=1, col=2)
fig.update_yaxes(showticklabels=False, row=2, col=2)
fig.update_yaxes(showticklabels=False, row=3, col=2)
fig.update_yaxes(showticklabels=False, row=1, col=3)
fig.update_yaxes(showticklabels=False, row=2, col=3)
fig.update_yaxes(showticklabels=False, row=3, col=3)
fig.update_yaxes(showticklabels=False, row=4, col=2)
fig.update_yaxes(showticklabels=False, row=4, col=3)
fig.update_layout(height=1400, width=1400,
                  title_text="Boxplots de Consumos Diarios por Cliente | Meses / Ano",
                  title_x=0.5,
                  font=dict(size=14),
                  showlegend=False,
                  template="plotly_white")
# Garantir que os meses de 4 a 10 apareçam no eixo X de todos os subplots
fig.update_xaxes(tickvals=list(range(4, 11)), ticktext=[str(m) for m in range(4, 11)])
# Linha 1 (Energia): kWh/dia/clt
fig.update_yaxes(title_text="kWh/dia/clt", row=1, col=1)
#fig.update_yaxes(title_text="kWh/dia/clt", row=1, col=2)
#fig.update_yaxes(title_text="kWh/dia/clt", row=1, col=3)

# Linha 2 (Água): m³/dia/clt
fig.update_yaxes(title_text="m³/dia/clt", row=2, col=1)
#fig.update_yaxes(title_text="m³/dia/clt", row=2, col=2)
#fig.update_yaxes(title_text="m³/dia/clt", row=2, col=3)

# Linha 3 (Gás ou Gas): kg/dia/clt
fig.update_yaxes(title_text="%", row=4, col=1)
#fig.update_yaxes(title_text="kg/dia/clt", row=3, col=2)
#fig.update_yaxes(title_text="kg/dia/clt", row=3, col=3)

# Linha 3 (Gas ou Ocupação): número (caso venhas a adicionar)
fig.update_yaxes(title_text="kg/dia/clt", row=3, col=1)
# fig.update_yaxes(title_text="número", row=3, col=2)
# fig.update_yaxes(title_text="número", row=3, col=3)


fig.update_yaxes(tickformat=".2f")
fig.show()
```




```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# 🔁 Mapeamento de nomes legíveis e unidades
nomes_legiveis = {
    "ELET_C": "Eletricidade",
    "AGUA_C": "Água",
    "GAS_C": "Gás"
}
unidades = {
    "ELET_C": "kWh",
    "AGUA_C": "m³",
    "GAS_C": "kg"
}

# 🎨 Definir cores fixas para os anos
cores_ano = {
    2023: '#636EFA',
    2024: '#EF553B',
    2025: '#00CC96'
}

# 📊 Agrupar por ano e mês
variaveis = list(nomes_legiveis.keys())
consumo_mensal = dados_E.groupby(['ano', 'mes'])[variaveis].sum().reset_index()
consumo_mensal['ano_mes'] = (
    consumo_mensal['ano'].astype(str) + '-' +
    consumo_mensal['mes'].astype(str).str.zfill(2)
)

# 📐 Criar subplots
fig = make_subplots(
    rows=len(variaveis),
    cols=1,
    shared_xaxes=True,
    vertical_spacing=0.08,
    subplot_titles=[f"{nomes_legiveis[var]} ({unidades[var]})" for var in variaveis]
)

# 🧠 Controlar exibição única da legenda por ano
anos_plotados = set()

# ➕ Adicionar traços por variável e ano com cores fixas
for i, var in enumerate(variaveis, start=1):
    for ano in sorted(consumo_mensal['ano'].unique()):
        df_filtrado = consumo_mensal[consumo_mensal['ano'] == ano]
        mostrar_legenda = ano not in anos_plotados
        fig.add_trace(
            go.Scatter(
                x=df_filtrado['mes'],
                y=df_filtrado[var],
                mode='lines+markers',
                name=str(ano),
                legendgroup=str(ano),
                showlegend=mostrar_legenda,
```

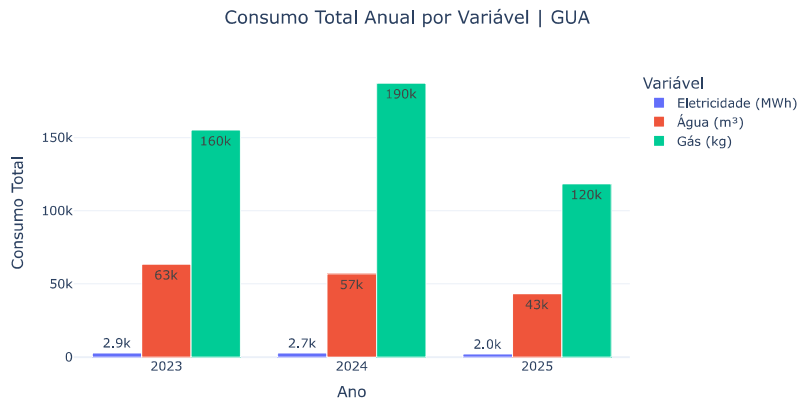
```
                marker=dict(color=cores_ano.get(ano, '#000000')),
                line=dict(color=cores_ano.get(ano, '#000000')),
                hovertemplate=f"Ano: {ano}<br>Mês: %{{x}}<br>{nomes_legiveis[var]}: %{{y:.0f}} {unidades[var]}"
            ),
            row=i,
            col=1
        )
        anos_plotados.add(ano)

# 🎨 Layout refinado
fig.update_layout(
    height=300 * len(variaveis),
    width=1000,
    title_text="Consumo por Mês e Ano (Subplots por Variável)",
    title_x=0.5,
    template='plotly_white',
    font=dict(size=14),
    hovermode='x unified'
)

fig.update_xaxes(
    tickmode='array',
    tickvals=list(range(1, 13)),
    ticktext=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
              'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
)

fig.show()
```




```
fig.write_html("cons_total_mes.html")
from google.colab import files
files.download("cons_total_mes.html")
```

```
import plotly.express as px

# 🔁 Mapeamento de nomes legíveis e unidades ajustadas
nomes_legiveis = {
    "ELET_C": "Eletricidade (MWh)",
    "AGUA_C": "Água (m³)",
    "GAS_C": "Gás (kg)"
}

# 📊 Agrupar por ano e somar consumo total
consumo_anual = dados_E.groupby('ano')[variaveis].sum().reset_index()

# 🔄 Converter Eletricidade de kWh para MWh
consumo_anual['ELET_C'] = consumo_anual['ELET_C'] / 1000

# 🔄 Transformar em formato longo para gráfico
df_long = consumo_anual.melt(
    id_vars='ano',
    value_vars=variaveis,
    var_name='variavel',
    value_name='consumo_total'
)

# 🧠 Aplicar nomes legíveis
df_long['variavel_legivel'] = df_long['variavel'].map(nomes_legiveis)

# 📊 Gráfico de barras comparativo
fig = px.bar(
    df_long,
    x='ano',
    y='consumo_total',
    color='variavel_legivel',
    barmode='group',
    text_auto='.2s',
    title='Consumo Total Anual por Variável | GUA',
    labels={'ano': 'Ano', 'consumo_total': 'Consumo Total', 'variavel_legivel': 'Variável'},
    color_discrete_map={
        "Eletricidade (MWh)": cores_ano.get(2023, '#636EFA'),
        "Água (m³)": cores_ano.get(2024, '#EF553B'),
        "Gás (kg)": cores_ano.get(2025, '#00CC96')
    },
    template='plotly_white'
)

fig.update_layout(
    title_x=0.5,
```

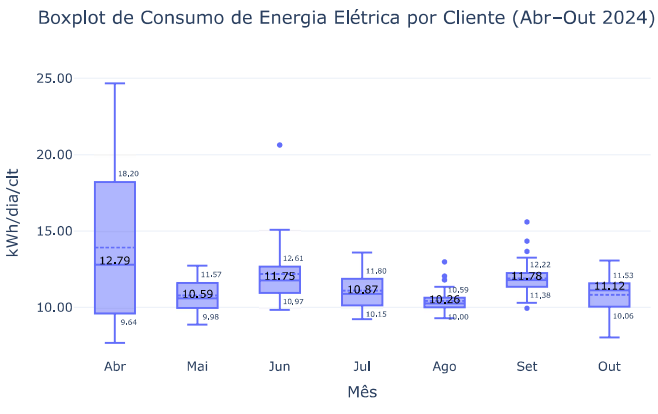
```
    width=900,
    height=500,
    font=dict(size=14),
    legend_title_text='Variável'
)

fig.update_traces(
    hovertemplate="Ano: %{x}<br>%{legend}: %{y:.2f}"
)

fig.show()
```



Aplicar ano de referência

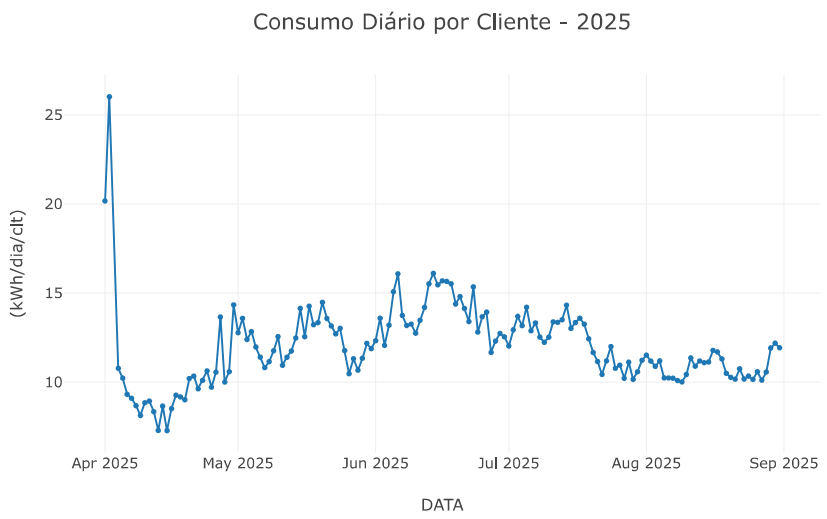
Energia Eletrica

```
import plotly.graph_objects as go

# Filtrar os dados para o ano de 2024 e meses de abril a outubro
dados_2024 = dados_E[(dados_E['ano'] == 2024) & (dados_E['mes'].between(4, 10))]

# Criar figura
fig = go.Figure()

# Adicionar boxplot
fig.add_trace(go.Box(
    x=dados_2024['mes'],
    y=dados_2024['ELET_CONS_CLT'],
    name='ELET_CONS_CLT 2024',
    boxmean=True))

 # Recalcular stats por mês usando os dados já filtrados
for mes in sorted(dados_2024['mes'].unique()):
    stats_mes = dados_2024[dados_2024['mes'] == mes]['ELET_CONS_CLT'].describe()

    fig.add_annotation(x=mes, y=stats_mes['50%'], text=f"{stats_mes['50%']:.2f}",
                       showarrow=False, font=dict(size=12, color="black"),
                       yshift=5)

    fig.add_annotation(x=mes, y=stats_mes['25%'], text=f"{stats_mes['25%']:.2f}",
                       showarrow=False, font=dict(size=8), xshift=15, yshift=-10)

    fig.add_annotation(x=mes, y=stats_mes['75%'], text=f"{stats_mes['75%']:.2f}",
                       showarrow=False, font=dict(size=8), xshift=15, yshift=10)



# Layout
fig.update_layout(
    title='Boxplot de Consumo de Energia Elétrica por Cliente (Abr–Out 2024)',
    xaxis_title='Mês',
    yaxis_title='kWh/dia/clt',
    xaxis=dict(tickvals=list(range(4, 11)), ticktext=['Abr', 'Mai', 'Jun', 'Jul', 'Ago', 'Set', 'Out']),
    template='plotly_white',
    height=500,
    width=800,
    font=dict(size=14)
)

fig.update_yaxes(tickformat=".2f")
fig.show()
```




```
# Filtrar apenas o ano de 2025
dados_2025 = dados_E.loc['2025-04-01':'2025-08-31', ['ELET_CONS_CLT']]

# Criar gráfico interativo com Plotly
fig = px.line(
    dados_2025,
    x=dados_2025.index,
```

```
    y='ELET_CONS_CLT',
    title='Consumo Diário por Cliente - 2025',
    labels={'x': 'Data', 'ELET_CONS_CLT': '(kWh/dia/clt)'}
)
fig.update_traces(mode='lines+markers')
fig.show()
```

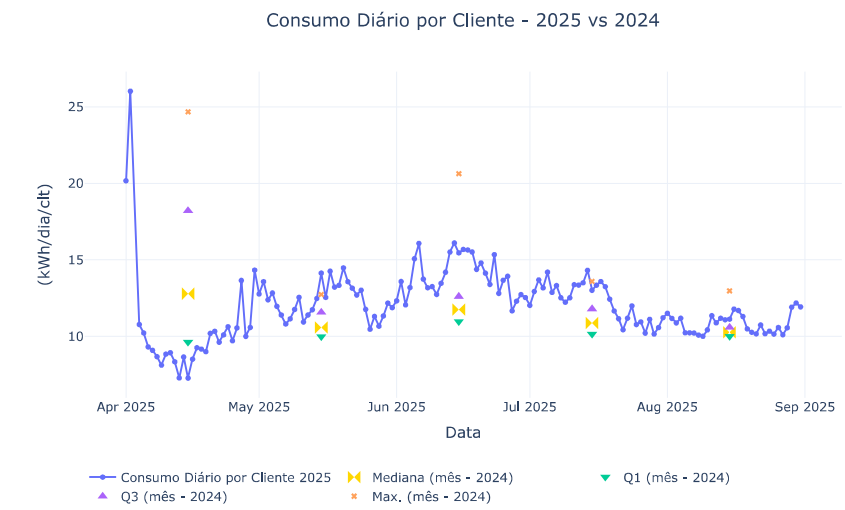



```
dados_E.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP'],
      dtype='object')
```

```
import plotly.express as px
import plotly.graph_objects as go
import pandas as pd

# Garantir que o índice é datetime
dados_E.index = pd.to_datetime(dados_E.index)

# Filtrar dados de 2025 entre abril e julho
dados_2025 = dados_E.loc['2025-04-01':'2025-08-31', ['ELET_CONS_CLT', 'mes']].copy()
dados_2024 = dados_E.loc['2024-04-01':'2024-08-31', ['ELET_CONS_CLT']].copy()
# Calcular estatísticas mensais
estatisticas = dados_2024.copy()
estatisticas['mes'] = estatisticas.index.month
estatisticas['ano'] = estatisticas.index.year

# Agrupar por mês e calcular estatísticas
resumo_mensal = estatisticas.groupby('mes')['ELET_CONS_CLT'].describe()[['25%', '50%', '75%', 'max']]

# Criar datas para posicionar os pontos no mês - 2024 de cada mês
datas_15 = [pd.Timestamp(year=2025, month=mes, day=15) for mes in resumo_mensal.index]

# Criar gráfico de linha com dados diários
fig = go.Figure()

fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['ELET_CONS_CLT'],
    mode='lines+markers',
    name='Consumo Diário por Cliente 2025',
    #line=dict(color='royalblue')
))

# Adicionar pontos estatísticos
fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['50%'],
    mode='markers',
    name='Mediana (mês - 2024)',
    marker=dict(color='gold', symbol='bowtie', size=14)
))

fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['25%'],
    mode='markers',
    name='Q1 (mês - 2024)',
    marker=dict( symbol='triangle-down', size=10)
))

fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['75%'],
    mode='markers',
    name='Q3 (mês - 2024)',
    marker=dict(symbol='triangle-up', size=10)
))
fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['max'],
    mode='markers',
    name='Max. (mês - 2024)',
    marker=dict(symbol='x', size=6)
))
# Layout final
fig.update_layout(

    title='Consumo Diário por Cliente - 2025 vs 2024',
    xaxis_title='Data',
    yaxis_title='(kWh/dia/clt)',
   template='plotly_white',
    height=600,
    width=1000,
    font=dict(size=14),title_x=0.5,
    legend=dict(orientation='h', y=-0.2)
)
```

```
fig.show()
```




```
# Certificar que dados_2025 tem a coluna 'mes'
dados_2025['mes'] = dados_2025.index.month

# Mapear a mediana de 2024 para cada linha de 2025 com base no mês
dados_2025['med2024'] = dados_2025['mes'].map(resumo_mensal['50%'])
dados_2025['erro_relativo'] = abs(dados_2025['ELET_CONS_CLT'] - dados_2025['med2024']) / dados_2025['med2024']
```

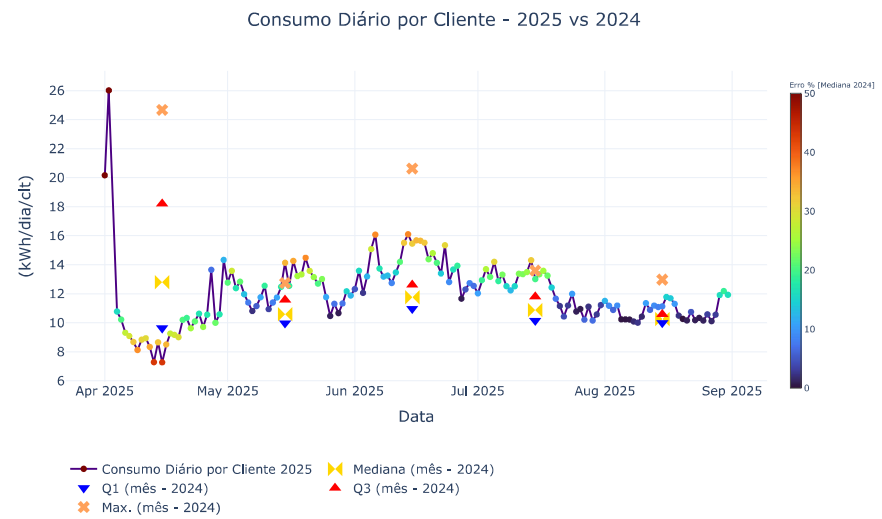
```
dados_2025
```

DATA	ELET_CONS_CLT	mes	med2024	erro_relativo
2025-04-01	20.167279	4	12.791762	0.576583
2025-04-02	26.015000	4	12.791762	1.033731
2025-04-04	10.777647	4	12.791762	0.157454
2025-04-05	10.224138	4	12.791762	0.200725
2025-04-06	9.313346	4	12.791762	0.271926
...	...	...	...	...
2025-08-27	10.112764	8	10.264049	0.014739
2025-08-28	10.565074	8	10.264049	0.029328
2025-08-29	11.912844	8	10.264049	0.160638
2025-08-30	12.184495	8	10.264049	0.187104
2025-08-31	11.930569	8	10.264049	0.162365

152 rows × 4 columns

```
# Certificar que dados_2025 tem a coluna 'mes'
dados_2025['mes'] = dados_2025.index.month

# Mapear a mediana de 2024 para cada linha de 2025 com base no mês
dados_2025['med2024'] = dados_2025['mes'].map(resumo_mensal['50%'])
dados_2025['erro_relativo'] = (abs(dados_2025['ELET_CONS_CLT'] - dados_2025['med2024']) / dados_2025['med2024'])*100

# Determinar cor dos marcadores
#cores = ['red' if erro > +0.60 else 'yellow' if erro > +0.30 else 'blue' for erro in dados_2025['erro_relativo']]


# Criar gráfico de linha com dados diários
fig = go.Figure()

fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['ELET_CONS_CLT'],
    mode='lines+markers',
    marker=dict(
    color=dados_2025['erro_relativo'],  # valores entre 0 e 1
    colorscale='Turbo',                  # ou 'Viridis', 'Plasma', etc.
    cmin=0,
    cmax=50,
    colorbar=dict(title='Erro % [Mediana 2024]', titlefont=dict(size=8),thickness=12, tickfont=dict(size=10)),
    size=7,),
    name='Consumo Diário por Cliente 2025',
    line=dict(color='indigo')
))

# Adicionar pontos estatísticos
fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['50%'],
    mode='markers',
    name='Mediana (mês - 2024)',
    marker=dict(color='gold', symbol='bowtie', size=16)
))

fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['25%'],
    mode='markers',
    name='Q1 (mês - 2024)',
    marker=dict(color='blue', symbol='triangle-down', size=12)
))

fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['75%'],
    mode='markers',
    name='Q3 (mês - 2024)',
    marker=dict(color='red',symbol='triangle-up', size=12)
))
fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['max'],
    mode='markers',
```

```
    name='Max. (mês - 2024)',
    marker=dict(symbol='x', size=12)
))
# Layout final
fig.update_layout(

    title='Consumo Diário por Cliente - 2025 vs 2024',
    xaxis_title='Data',
    yaxis_title='(kWh/dia/clt)',
   template='plotly_white',
    height=600,
    width=1000,
    font=dict(size=14),title_x=0.5,
    legend=dict(orientation='h', y=-0.2)
)
fig.update_yaxes(
    dtick=2,  # intervalo de 2 unidades
    title_text='(kWh/dia/clt)'
)

fig.show()
```



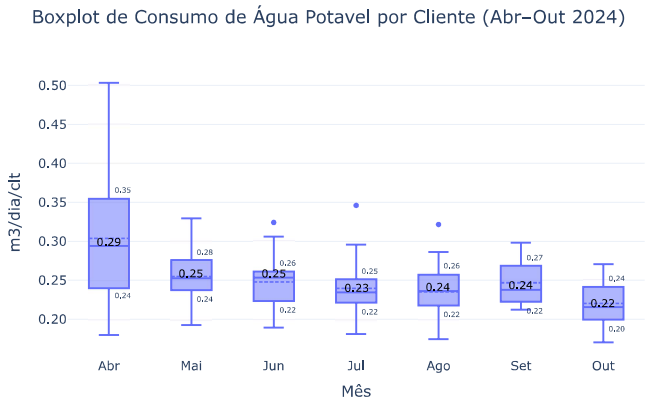

```
dados_2025
```

DATA	ELET_CONS_CLT	mes	med2024	erro_relativo
2025-04-01	20.167279	4	12.791762	57.658333
2025-04-02	26.015000	4	12.791762	103.373071
2025-04-04	10.777647	4	12.791762	15.745409
2025-04-05	10.224138	4	12.791762	20.072484
2025-04-06	9.313346	4	12.791762	27.192627
...	...	...	...	...
2025-08-27	10.112764	8	10.264049	1.473931
2025-08-28	10.565074	8	10.264049	2.932808
2025-08-29	11.912844	8	10.264049	16.063785
2025-08-30	12.184495	8	10.264049	18.710406
2025-08-31	11.930569	8	10.264049	16.236474

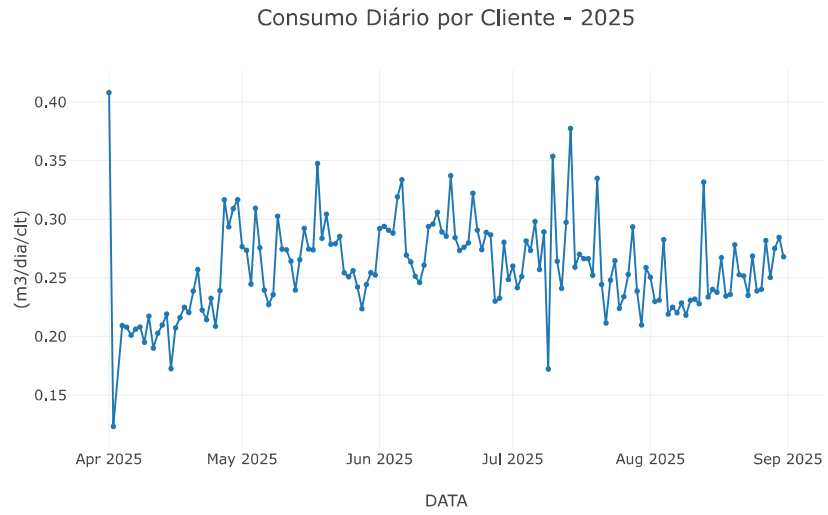
152 rows × 4 columns

Agua

```
import plotly.graph_objects as go

# Filtrar os dados para o ano de 2024 e meses de abril a outubro
dados_2024 = dados_E[(dados_E['ano'] == 2024) & (dados_E['mes'].between(4, 10))]

# Criar figura
fig = go.Figure()

# Adicionar boxplot
fig.add_trace(go.Box(
    x=dados_2024['mes'],
    y=dados_2024['AGUA_CONS_CLT'],
    name='AGUA_CONS_CLT 2024',
    boxmean=True))

 # Recalcular stats por mês usando os dados já filtrados
for mes in sorted(dados_2024['mes'].unique()):
    stats_mes = dados_2024[dados_2024['mes'] == mes]['AGUA_CONS_CLT'].describe()

    fig.add_annotation(x=mes, y=stats_mes['50%'], text=f"{stats_mes['50%']:.2f}",
                       showarrow=False, font=dict(size=12, color="black"),
                       yshift=5)

    fig.add_annotation(x=mes, y=stats_mes['25%'], text=f"{stats_mes['25%']:.2f}",
                       showarrow=False, font=dict(size=8), xshift=15, yshift=-10)

    fig.add_annotation(x=mes, y=stats_mes['75%'], text=f"{stats_mes['75%']:.2f}",
                       showarrow=False, font=dict(size=8), xshift=15, yshift=10)



# Layout
fig.update_layout(
    title='Boxplot de Consumo de Água Potavel por Cliente (Abr–Out 2024)',
    xaxis_title='Mês',
    yaxis_title='m3/dia/clt',
    xaxis=dict(tickvals=list(range(4, 11)), ticktext=['Abr', 'Mai', 'Jun', 'Jul', 'Ago', 'Set', 'Out']),
    template='plotly_white',
    height=500,
    width=800,
    font=dict(size=14)
)

fig.update_yaxes(tickformat=".2f")
fig.show()
```




```
# Filtrar apenas o ano de 2025
dados_2025 = dados_E.loc['2025-04-01':'2025-08-31', ['AGUA_CONS_CLT']]

# Criar gráfico interativo com Plotly
fig = px.line(
    dados_2025,
    x=dados_2025.index,
    y='AGUA_CONS_CLT',
    title='Consumo Diário por Cliente - 2025',
    labels={'x': 'Data', 'AGUA_CONS_CLT': '(m3/dia/clt)'}
)
fig.update_traces(mode='lines+markers')
fig.update_yaxes(tickformat=".2f")
fig.show()
```

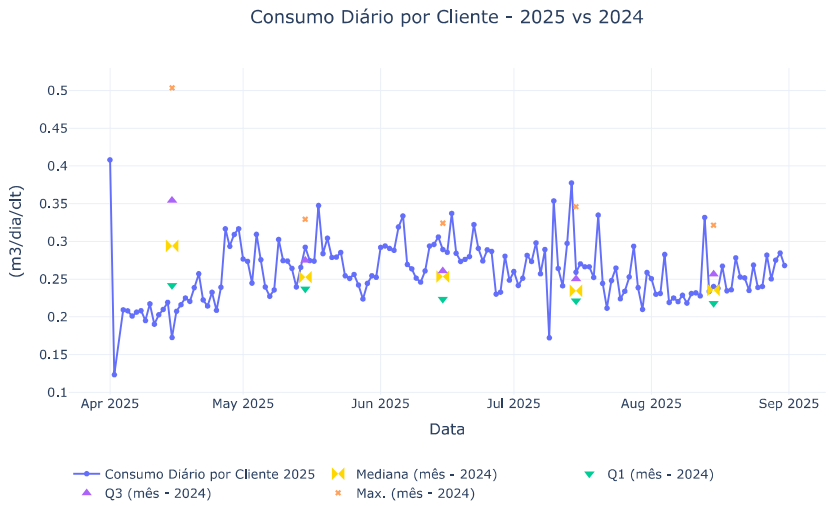



```
dados_E.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP'],
      dtype='object')
```

```
import plotly.express as px
import plotly.graph_objects as go
import pandas as pd

# Garantir que o índice é datetime
dados_E.index = pd.to_datetime(dados_E.index)

# Filtrar dados de 2025 entre abril e julho
dados_2025 = dados_E.loc['2025-04-01':'2025-08-31', ['AGUA_CONS_CLT', 'mes']].copy()
dados_2024 = dados_E.loc['2024-04-01':'2024-08-31', ['AGUA_CONS_CLT']].copy()
# Calcular estatísticas mensais
estatisticas = dados_2024.copy()
estatisticas['mes'] = estatisticas.index.month
estatisticas['ano'] = estatisticas.index.year

# Agrupar por mês e calcular estatísticas
resumo_mensal = estatisticas.groupby('mes')['AGUA_CONS_CLT'].describe()[['25%', '50%', '75%', 'max']]

# Criar datas para posicionar os pontos no mês - 2024 de cada mês
datas_15 = [pd.Timestamp(year=2025, month=mes, day=15) for mes in resumo_mensal.index]

# Criar gráfico de linha com dados diários
fig = go.Figure()

fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['AGUA_CONS_CLT'],
    mode='lines+markers',
    name='Consumo Diário por Cliente 2025',
    #line=dict(color='royalblue')
))

# Adicionar pontos estatísticos
fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['50%'],
    mode='markers',
    name='Mediana (mês - 2024)',
```

```
    marker=dict(color='gold', symbol='bowtie', size=14)
))

fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['25%'],
    mode='markers',
    name='Q1 (mês - 2024)',
    marker=dict( symbol='triangle-down', size=10)
))

fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['75%'],
    mode='markers',
    name='Q3 (mês - 2024)',
    marker=dict(symbol='triangle-up', size=10)
))
fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['max'],
    mode='markers',
    name='Max. (mês - 2024)',
    marker=dict(symbol='x', size=6)
))
# Layout final
fig.update_layout(

    title='Consumo Diário por Cliente - 2025 vs 2024',
    xaxis_title='Data',
    yaxis_title='(m3/dia/clt)',
   template='plotly_white',
    height=600,
    width=1000,
    font=dict(size=14),title_x=0.5,
    legend=dict(orientation='h', y=-0.2)
)

fig.show()
```




```
# Certificar que dados_2025 tem a coluna 'mes'
dados_2025['mes'] = dados_2025.index.month

# Mapear a mediana de 2024 para cada linha de 2025 com base no mês
dados_2025['med2024'] = dados_2025['mes'].map(resumo_mensal['50%'])
dados_2025['erro_relativo'] = abs(dados_2025['AGUA_CONS_CLT'] - dados_2025['med2024']) / dados_2025['med2024']
```

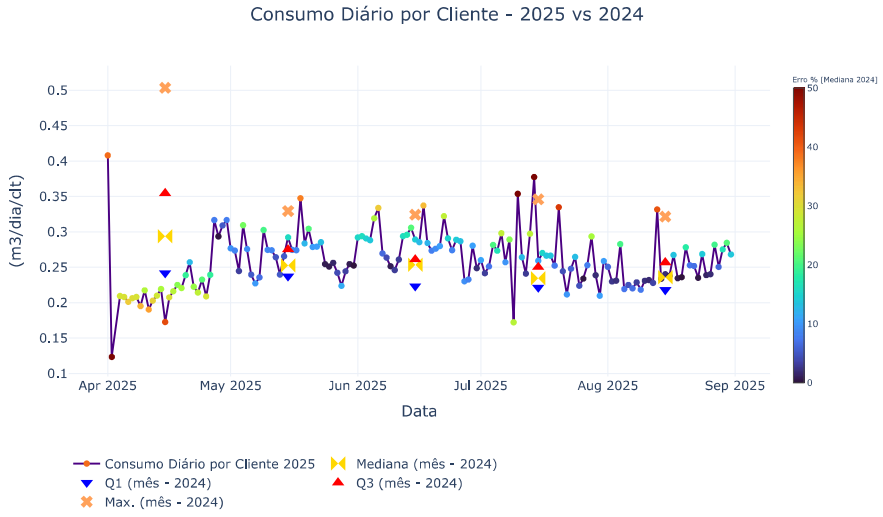
```
dados_2025
```

DATA	AGUA_CONS_CLT	mes	med2024	erro_relativo
2025-04-01	0.408088	4	0.293996	0.388073
2025-04-02	0.123333	4	0.293996	0.580494
2025-04-04	0.209412	4	0.293996	0.287706
2025-04-05	0.207974	4	0.293996	0.292596
2025-04-06	0.201161	4	0.293996	0.315772
...	...	...	...	...
2025-08-27	0.281911	8	0.235947	0.194807
2025-08-28	0.250412	8	0.235947	0.061307
2025-08-29	0.275229	8	0.235947	0.166489
2025-08-30	0.284593	8	0.235947	0.206174
2025-08-31	0.268081	8	0.235947	0.136193

152 rows × 4 columns

```
# Certificar que dados_2025 tem a coluna 'mes'
dados_2025['mes'] = dados_2025.index.month

# Mapear a mediana de 2024 para cada linha de 2025 com base no mês
dados_2025['med2024'] = dados_2025['mes'].map(resumo_mensal['50%'])
dados_2025['erro_relativo'] = (abs(dados_2025['AGUA_CONS_CLT'] - dados_2025['med2024']) / dados_2025['med2024'])*100

# Determinar cor dos marcadores
#cores = ['red' if erro > +0.60 else 'yellow' if erro > +0.30 else 'blue' for erro in dados_2025['erro_relativo']]


# Criar gráfico de linha com dados diários
fig = go.Figure()

fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['AGUA_CONS_CLT'],
    mode='lines+markers',
    marker=dict(
```

```
    color=dados_2025['erro_relativo'],  # valores entre 0 e 1
    colorscale='Turbo',                 # ou 'Viridis', 'Plasma', etc.
    cmin=0,
    cmax=50,
    colorbar=dict(title='Erro % [Mediana 2024]', titlefont=dict(size=8),thickness=12, tickfont=dict(size=10)),
    size=7),
    name='Consumo Diário por Cliente 2025',
    line=dict(color='indigo')
))

# Adicionar pontos estatísticos
fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['50%'],
    mode='markers',
    name='Mediana (mês - 2024)',
    marker=dict(color='gold', symbol='bowtie', size=16)
))

fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['25%'],
    mode='markers',
    name='Q1 (mês - 2024)',
    marker=dict(color='blue', symbol='triangle-down', size=12)
))

fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['75%'],
    mode='markers',
    name='Q3 (mês - 2024)',
    marker=dict(color='red',symbol='triangle-up', size=12)
))
fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['max'],
    mode='markers',
    name='Max. (mês - 2024)',
    marker=dict(symbol='x', size=12)
))
# Layout final
fig.update_layout(

    title='Consumo Diário por Cliente - 2025 vs 2024',
    xaxis_title='Data',
    yaxis_title='(m3/dia/clt)',
   template='plotly_white',
    height=600,
    width=1000,
    font=dict(size=14),title_x=0.5,
    legend=dict(orientation='h', y=-0.2)
)
fig.update_yaxes(
    #dtick=2,  # intervalo de 2 unidades
    title_text='(m3/dia/clt)'
)

fig.show()
```




```
dados_2025
```

DATA	AGUA_CONS_CLT	mes	med2024	erro_relativo
2025-04-01	0.408088	4	0.293996	38.807286
2025-04-02	0.123333	4	0.293996	58.049354
2025-04-04	0.209412	4	0.293996	28.770604
2025-04-05	0.207974	4	0.293996	29.259599
2025-04-06	0.201161	4	0.293996	31.577178
...	...	...	...	...
2025-08-27	0.281911	8	0.235947	19.480660
2025-08-28	0.250412	8	0.235947	6.130670
2025-08-29	0.275229	8	0.235947	16.648932
2025-08-30	0.284593	8	0.235947	20.617361
2025-08-31	0.268081	8	0.235947	13.619284

152 rows × 4 columns

```
resumo_mensal
```

mes	25%	50%	75%	max
4	0.241735	0.293996	0.354523	0.503448
5	0.237254	0.252677	0.275007	0.329396
6	0.223404	0.253521	0.260994	0.324176
7	0.221567	0.234538	0.250408	0.346021
8	0.218039	0.235947	0.256762	0.321575

```
resumo_mensal
```

mes	25%	50%	75%	max
4	0.241735	0.293996	0.354523	0.503448
5	0.237254	0.252677	0.275007	0.329396
6	0.223404	0.253521	0.260994	0.324176
7	0.221567	0.234538	0.250408	0.346021
8	0.218039	0.235947	0.256762	0.321575

Comece a programar ou gerar com a IA.

Aplicar ML

Filtrar dados

```
# Check for missing values
print(dados_E.isnull().sum())
```

Mostrar saída oculta

```
dados_E.columns
```

Mostrar saída oculta

```
df1=dados_E[['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP']]
```

```
df1.tail()
```

Mostrar saída oculta

Comece a programar ou gerar com a IA.

```
# Check for missing values
print(df1.isnull().sum())
```

Mostrar saída oculta

```
#print(df1.isnull().sum())
```

```
#df1 = df1[df1.index <= '2025-07-21']
```

```
df1.tail()
```

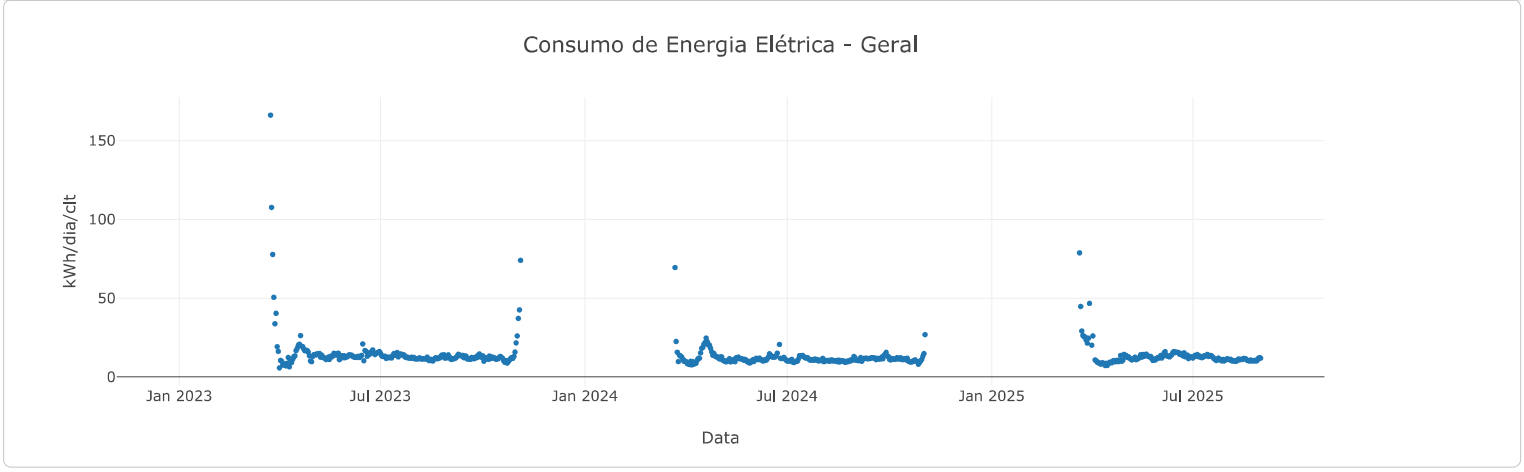
DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	dia_mes	semana_ano	feriado	AGUA_CONS_CLT	GAS_CONS_CLT	CO2e_ELET	CO2e_AGUA	CO2e_GAS	Total_CO2e	CO2e_QOCP
2025-08-27	23.41699	31.9	18.3	13.6	75.88219	89.0	51.5	37.5	23.98021	12914.0	...	27	35	False	0.281911	0.528103	2156.638	104.40	2198.50488	4459.54288	10.021445
2025-08-28	22.23625	30.2	18.3	11.9	80.14818	100.0	54.7	45.3	22.37211	12826.0	...	28	35	False	0.250412	0.000000	2141.942	88.16	0.00000	2230.10200	5.259675
2025-08-29	22.64507	34.0	17.5	16.5	79.59112	91.5	54.4	37.1	22.82476	12985.0	...	29	35	False	0.275229	0.721822	2168.495	87.00	2564.92236	4820.41736	12.553170
2025-08-30	24.65698	34.5	19.7	14.8	78.81356	92.0	56.4	35.6	25.36257	12416.0	...	30	35	False	0.284593	0.772116	2073.472	84.10	2564.92236	4722.49436	12.973886
2025-08-31	22.51888	30.1	18.7	11.4	83.64605	98.5	59.9	38.6	22.13280	12372.0	...	31	35	False	0.268081	0.541938	2066.124	80.62	1832.08740	3978.83140	10.254720

5 rows × 41 columns

```
df1.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP'],
      dtype='object')
```

```
fig = px.scatter(df1, x=df1.index, y='ELET_CONS_CLT',
                 title='Consumo de Energia Elétrica - Geral   ',
                labels={'DATA': 'Data', 'ELET_CONS_CLT': 'kWh/dia/clt'}
)

fig.update_layout(width=1500, height=500)


fig.show()
```



verificar e corrigir Outliers

```
def filtrar_outliers_iqr(df1, coluna):
    """
    Filtra outliers de uma coluna usando o método IQR.

    Args:
        df1: DataFrame.
        coluna: Nome da coluna a ser filtrada.

    Returns:
        DataFrame sem os outliers.
    """

    Q1 = df1[coluna].quantile(0.25)
    Q3 = df1[coluna].quantile(0.75)
    IQR = Q3 - Q1
    limite_inferior = Q1 - 1.5 * IQR
    limite_superior = Q3 + 1.5 * IQR
    df1_filtrado1 = df1[(df1[coluna] >= limite_inferior) & (df1[coluna] <= limite_superior)]
    return df1_filtrado1

# Aplicar a função na coluna 'ELET_CONS_CLT'
df1_filtrado1 = filtrar_outliers_iqr(df1, 'ELET_CONS_CLT')
```

```
/usr/local/lib/python3.12/dist-packages/numpy/lib/_function_base_impl.py:4779: RuntimeWarning:

invalid value encountered in subtract
```

```
import scipy.stats as stats

def filtrar_outliers_zscore(df1, coluna, threshold=3):
    """
    Filtra outliers de uma coluna usando o Z-score.

    Args:
        df1: DataFrame.
        coluna: Nome da coluna a ser filtrada.
        threshold: Limite do Z-score para considerar um dado como outlier.

    Returns:
        DataFrame sem os outliers.
    """

    z = np.abs(stats.zscore(df1[coluna]))
    df1_filtrado2 = df1[(z < threshold)]
    return df1_filtrado2

# Aplicar a função na coluna 'ELET_CONS_CLT' com threshold de 3
df1_filtrado2 = filtrar_outliers_zscore(df1, 'ELET_CONS_CLT', 3)
```

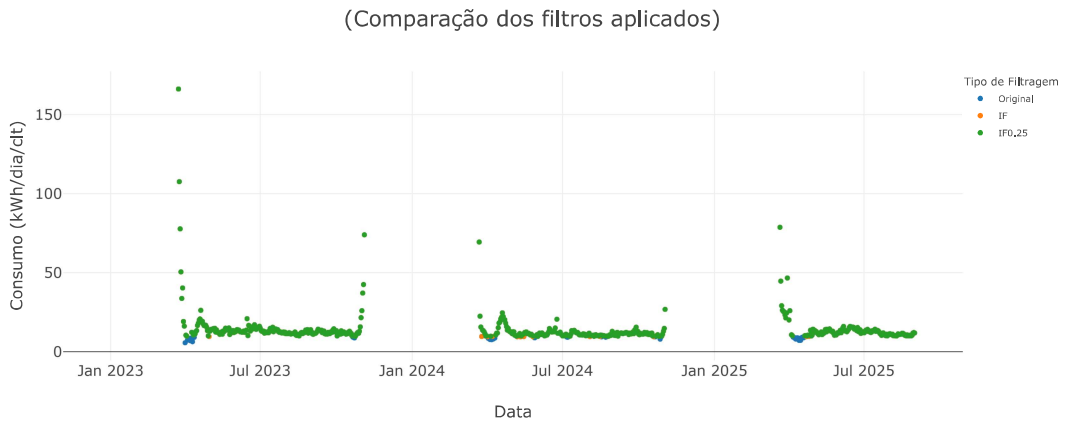
```
/usr/local/lib/python3.12/dist-packages/scipy/stats/_stats_py.py:1158: RuntimeWarning:

invalid value encountered in subtract
```

```
from sklearn.ensemble import IsolationForest

def filtrar_outliers_isolation_forest(df1, coluna, contamination=0.05):
    """
    Filtra outliers de uma coluna usando o Isolation Forest.

    Args:
        df: DataFrame.
        coluna: Nome da coluna a ser filtrada.
        contamination: Proporção de outliers esperada no dataset.

    Returns:
        DataFrame sem os outliers.
    """

    iso = IsolationForest(contamination=contamination)
    yhat = iso.fit_predict(df1[[coluna]])
    mask = yhat != -1
    df1_filtrado3 = df1[mask]
    return df1_filtrado3

# Aplicar a função na coluna 'ELET_CONS_CLT' com contamination de 5%
df1_filtrado3 = filtrar_outliers_isolation_forest(df1, 'ELET_CONS_CLT', 0.05)
```

```
from sklearn.ensemble import IsolationForest

def filtrar_outliers_isolation_forest(df1, coluna, contamination=0.05):
    """
    Filtra outliers de uma coluna usando o Isolation Forest.

    Args:
        df: DataFrame.
        coluna: Nome da coluna a ser filtrada.
        contamination: Proporção de outliers esperada no dataset.

    Returns:
        DataFrame sem os outliers.
    """

    iso = IsolationForest(contamination=contamination)
    yhat = iso.fit_predict(df1[[coluna]])
    mask = yhat != -1
    df1_filtrado4 = df1[mask]
```

```
    return df1_filtrado4

# Aplicar a função na coluna 'ELET_CONS_CLT' com contamination de 15%
df1_filtrado4 = filtrar_outliers_isolation_forest(df1, 'ELET_CONS_CLT', 0.15)
```

```
df_plot = pd.concat([
    df1[['ELET_CONS_CLT']].assign(Série='Original'),
    df1_filtrado1[['ELET_CONS_CLT']].assign(Série='IQR'),
    df1_filtrado2[['ELET_CONS_CLT']].assign(Série='Z-core'),
    df1_filtrado3[['ELET_CONS_CLT']].assign(Série='IF'),
    df1_filtrado4[['ELET_CONS_CLT']].assign(Série='IF0.25')
])


df_plot = df_plot.reset_index()
df_plot = df_plot.rename(columns={'index': 'DATA'})


# Criar o gráfico interativo
fig = px.scatter(df_plot, x='DATA', y='ELET_CONS_CLT', color='Série',
             title='(Comparação dos filtros aplicados)',
             labels={'DATA': 'Data', 'ELET_CONS_CLT': 'Consumo (kWh/dia/clt)', 'Série': 'Tipo de Filtragem'})

fig.update_layout(width=1200, height=500)
# Exibir o gráfico
fig.show()
fig.show('notebook_connected')
```



+ engenharia de dados

```
def create_features(df1_filtrado3):


    df1_filtrado3['ELET_CONS_CLT_lag1'] = df1_filtrado3['ELET_CONS_CLT'].shift(1)  # Consumo do dia anterior
    df1_filtrado3['ELET_CONS_CLT_lag7'] = df1_filtrado3['ELET_CONS_CLT'].shift(7)  # Consumo da semana anterior
    df1_filtrado3['ELET_CONS_CLT_rolling_mean'] = df1_filtrado3['ELET_CONS_CLT'].rolling(window=7).mean()  # Média móvel de 7 dias
    df1_filtrado3['ELET_CONS_CLT_rolling_median'] = df1_filtrado3['ELET_CONS_CLT'].rolling(window=7).median()  # Média móvel de 7 dias
    df1_filtrado3['ELET_CONS_CLT_rolling_std'] = df1_filtrado3['ELET_CONS_CLT'].rolling(window=7).std()  # desvio padrão móvel de 7 dias
    df1_filtrado3['ELET_CONS_CLT_error_median'] = (df1_filtrado3['ELET_CONS_CLT'] - df1_filtrado3['ELET_CONS_CLT_rolling_median']  )/ df1_filtrado3['ELET_CONS_CLT_rolling_median']
    df1_filtrado3['ELET_CONS_CLT_pct_change'] = df1_filtrado3['ELET_CONS_CLT'].pct_change()  # Variação percentual

    # Remover NaNs gerados pelo shift e rolling
    df1_filtrado3.dropna(inplace=True)

    return df1_filtrado3

df1_filtrado3 = create_features(df1_filtrado3)
```

Mostrar saída oculta

```
df1_filtrado3
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	CO2e_GAS	Total_CO2e	CO2e_QOCP	ELET_CONS_CLT_lag1	ELET_CONS_CLT_lag7	ELET_CONS_CLT_rolling_mean
2023-03-30	18.750000	24.900000	13.8	11.099999	77.516667	91.000000	63.299999	27.700001	21.755307	5679.0	...	1099.25244	2078.96544	18.077960	40.397906	inf	70.756545
2023-03-31	18.333333	24.400000	14.1	10.299999	85.416667	100.000000	70.000000	30.000000	19.473651	11569.0	...	1465.66992	3478.60292	13.226627	19.185811	166.121212	49.346171
2023-04-02	18.366667	22.100000	14.2	7.900001	74.466667	95.599998	44.400002	51.199997	22.624800	8496.0	...	732.83496	2217.78696	7.809109	16.248596	107.577778	35.480039
2023-04-03	19.140000	25.299999	15.0	10.299999	55.960000	75.500000	44.500000	31.000000	25.191946	9342.0	...	732.83496	2372.69896	7.168275	10.514851	77.728395	25.800285
2023-04-09	18.785715	26.700001	13.6	13.100000	74.428573	87.300003	59.900002	27.400002	22.498339	11715.0	...	2748.13110	4799.36610	14.721982	9.970117	50.528302	20.347327
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2025-08-27	23.416990	31.900000	18.3	13.600000	75.882190	89.000000	51.500000	37.500000	23.980210	12914.0	...	2198.50488	4459.54288	10.021445	10.585227	10.270170	10.325795
2025-08-28	22.236250	30.200000	18.3	11.900000	80.148180	100.000000	54.700000	45.300000	22.372110	12826.0	...	0.00000	2230.10200	5.259675	10.112764	10.165070	10.382938
2025-08-29	22.645070	34.000000	17.5	16.500000	79.591120	91.500000	54.400000	37.100000	22.824760	12985.0	...	2564.92236	4820.41736	12.553170	10.565074	10.747419	10.549428
2025-08-30	24.656980	34.500000	19.7	14.800000	78.813560	92.000000	56.400000	35.600000	25.362570	12416.0	...	2564.92236	4722.49436	12.973886	11.912844	10.176106	10.836340
2025-08-31	22.518880	30.100000	18.7	11.400000	83.646050	98.500000	59.900000	38.600000	22.132800	12372.0	...	1832.08740	3978.83140	10.254720	12.184495	10.341620	11.063333

552 rows × 48 columns

```
# Check for missing values
print(df1_filtrado3.isnull().sum())
```

```
T_mean                          0
T_max                           0
T_min                           0
T_rg                            0
RH_mean                         0
RH_max                          0
RH_min                          0
RH_rg                           0
Calor_idx                       0
ELET_C                          0
EP.C                            0
50S.C                           0
COZ.50S.C                       0
COZ.EP.C                        0
PST.C                           0
LAV.C                           0
Q_OCP                           0
OCP                             0
OCP[%]                          0
AD                              0
CR                              0
CLTs                            0
ELET_CONS_CLT                   0
CONS_Q                          0
AGUA_C                          0
GAS_C                           0
dia_semana                      0
trimestre                       0
mes                             0
ano                             0
dia_ano                         0
dia_mes                         0
semana_ano                      0
feriado                         0
AGUA_CONS_CLT                   0
GAS_CONS_CLT                    0
CO2e_ELET                       0
CO2e_AGUA                       0
CO2e_GAS                        0
Total_CO2e                      0
CO2e_QOCP                       0
ELET_CONS_CLT_lag1              0
ELET_CONS_CLT_lag7              0
ELET_CONS_CLT_rolling_mean      0
ELET_CONS_CLT_rolling_median    0
ELET_CONS_CLT_rolling_std       0
ELET_CONS_CLT_error_median      0
ELET_CONS_CLT_pct_change        0
dtype: int64
```

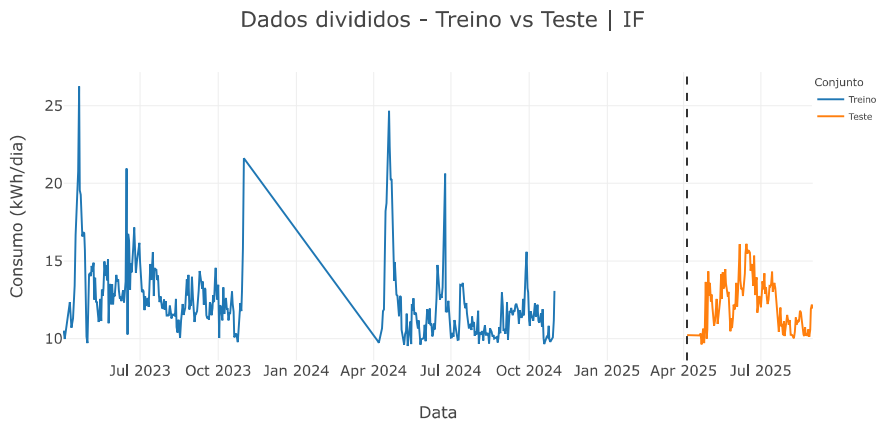
Definir dados para treino

```
"""treino1 = df1_filtrado1.loc[(df1_filtrado1.index > '2023-04-01') & (df1_filtrado1.index <= '2024-10-31')].reset_index()
teste1 = df1_filtrado1.loc[df1_filtrado1.index >= '2025-04-05'].reset_index()


treino1['Tipo'] = 'Treino1'
teste1['Tipo'] = 'Teste1'
df_plot = pd.concat([treino1, teste1])


fig = px.line(df_plot, x='DATA', y='ELET_CONS_CLT', color='Tipo',
              title='Dados divididos - Treino vs Teste | IQR',
              labels={'DATA': 'Data', 'ELET_CONS_CLT': 'Consumo (kWh/dia)', 'Tipo': 'Conjunto'})

#  a linha vertical para marcar a separação
fig.add_vline(x='2025-03-21', line_dash="dash", line_color="black")


fig.update_layout(width=1500, height=500)
# Exibir o gráfico
fig.show()
fig.show('notebook_connected')"""
```

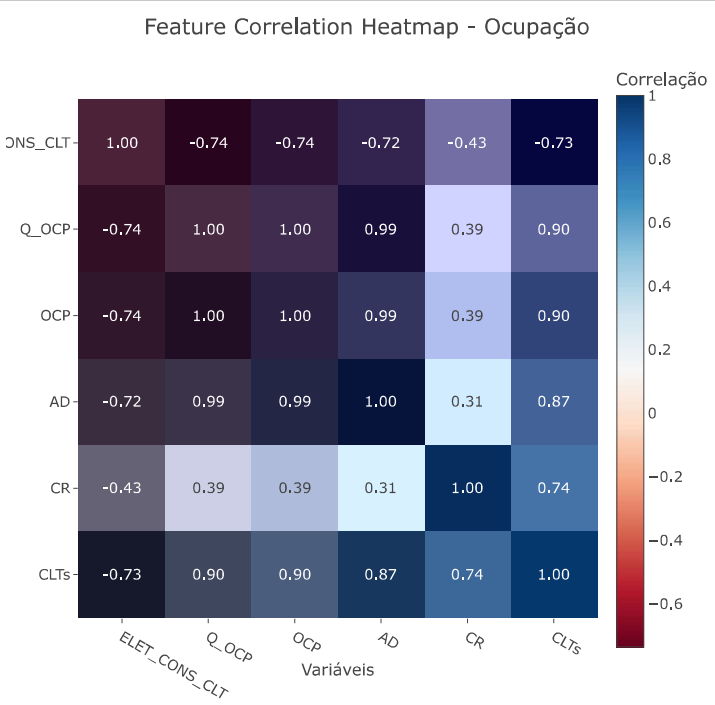
```
'treino1 = df1_filtrado1.loc[(df1_filtrado1.index > \'2023-04-01\') & (df1_filtrado1.index <= \'2024-10-31\')].reset_index()\nteste1 = df1_filtrado1.loc[df1_filtrado1.index >= \'2025-04-05\'].reset_index()\n\n\nt
reino1[\'Tipo\'] = \'Treino1\'\nteste1[\'Tipo\'] = \'Teste1\'\ndf_plot = pd.concat([treino1, teste1])\n\n\nfig = px.line(df_plot, x=\'DATA\', y=\'ELET_CONS_CLT\', color=\'Tipo\',\n              title=\'Dados divi
didos - Treino vs Teste | IQR\',\n              labels={\'DATA\': \'Data\', \'ELET_CONS_CLT\': \'Consumo (kWh/dia)\', \'Tipo\': \'Conjunto\'})\n\n#  a linha vertical para marcar a separação\nfig.add_vline(x=\'202
5-03-21\', line_dash="dash", line_color="black")\n\n\nfig.update_layout(width=1500, height=500)\n# Exibir o gráfico\nfig.show()\nfig.show(\'notebook_connected\')'
```

```
treino = df1_filtrado3.loc[
    ((df1_filtrado3.index > '2023-04-01') & (df1_filtrado3.index <= '2023-10-31')) |
    ((df1_filtrado3.index > '2024-04-01') & (df1_filtrado3.index <= '2024-10-31'))
].reset_index()


#teste = df1_filtrado3.loc[df1_filtrado3.index >= '2025-04-05'].reset_index()

teste = df1_filtrado3.loc[df1_filtrado3.index >= '2025-04-05'].reset_index()


treino['Tipo'] = 'Treino'
teste['Tipo'] = 'Teste'
df_plot = pd.concat([treino, teste])
```

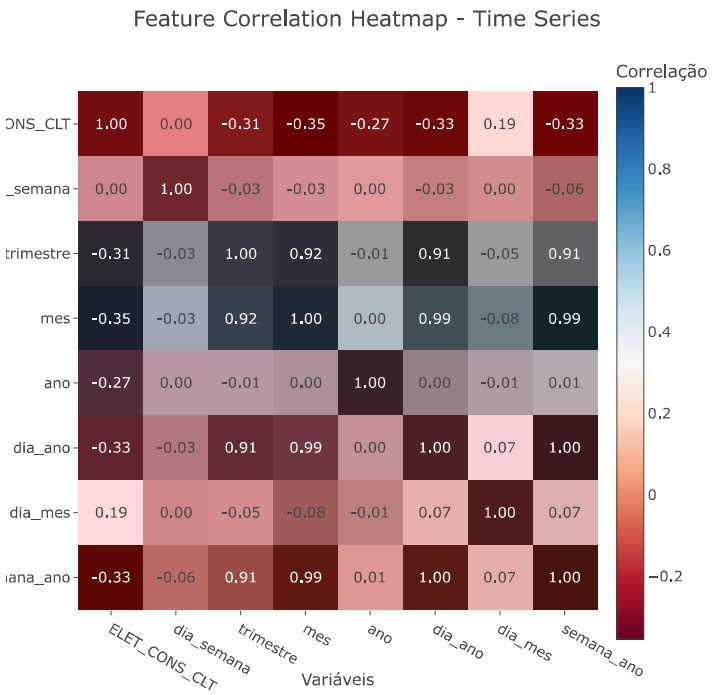
```
fig = px.line(df_plot, x='DATA', y='ELET_CONS_CLT', color='Tipo',
              title='Dados divididos - Treino vs Teste | IF',
              labels={'DATA': 'Data', 'ELET_CONS_CLT': 'Consumo (kWh/dia)', 'Tipo': 'Conjunto'})


fig.add_vline(x='2025-04-05', line_dash="dash", line_color="black")


fig.update_layout(width=1000, height=500)

fig.show()
fig.show('notebook_connected')
```



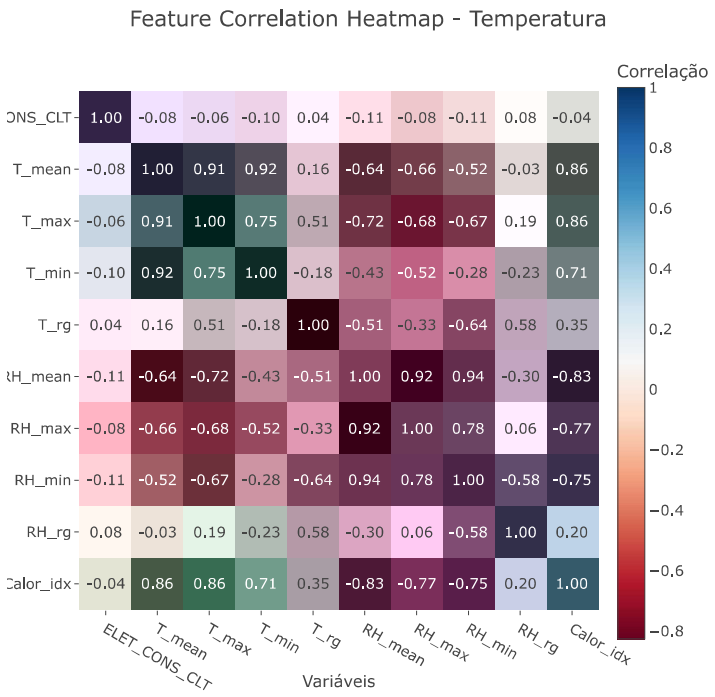

```
# Calcular a matriz de correlação
corr_matrix = treino[['ELET_CONS_CLT', 'Q_OCP', 'OCP', 'AD', 'CR', 'CLTs',
      ]].corr()

# Criar o heatmap interativo
# Change 'coolwarm' to a valid plotly colorscale like 'RdBu'
fig = px.imshow(corr_matrix,
                labels={ "x": "Variáveis","y": " ", "color": "Correlação"},
                color_continuous_scale="RdBu", # Changed colorscale to a valid plotly colorscale
                text_auto=True)  # Exibe os valores diretamente no gráfico


# Ajustarcasas decimais
fig.update_traces(texttemplate="%{z:.2f}")
# Ajustar layout
fig.update_layout(title="Feature Correlation Heatmap - Ocupação", title_x=0.5, width=800, height=800)
fig.update_layout(
    #title_font=dict(size=24),  # Tamanho do título
    xaxis_title_font=dict(size=18),  # Tamanho do rótulo do eixo X
    yaxis_title_font=dict(size=18),  # Tamanho do rótulo do eixo Y
    font=dict(size=16))
fig.show()
fig.show('notebook_connected')
```




```
# Calcular a matriz de correlação
corr_matrix = treino[['ELET_CONS_CLT', 'dia_semana', 'trimestre', 'mes', 'ano',
       'dia_ano', 'dia_mes', 'semana_ano']].corr()

# Criar o heatmap interativo
# Change 'coolwarm' to a valid plotly colorscale like 'RdBu'
fig = px.imshow(corr_matrix,
                labels={ "x": "Variáveis","y": " ", "color": "Correlação"},
                color_continuous_scale="RdBu", # Changed colorscale to a valid plotly colorscale
                text_auto=True)  # Exibe os valores diretamente no gráfico


# Ajustarcasas decimais
fig.update_traces(texttemplate="%{z:.2f}")
# Ajustar layout
fig.update_layout(title="Feature Correlation Heatmap - Time Series", title_x=0.5, width=800, height=800)
fig.update_layout(
    #title_font=dict(size=24),  # Tamanho do título
    xaxis_title_font=dict(size=18),  # Tamanho do rótulo do eixo X
    yaxis_title_font=dict(size=18),  # Tamanho do rótulo do eixo Y
    font=dict(size=16))
fig.show()
fig.show('notebook_connected')
```




```
# Calcular a matriz de correlação
corr_matrix = treino[['ELET_CONS_CLT', 'T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx']].corr()

# Criar o heatmap interativo
# Change 'coolwarm' to a valid plotly colorscale like 'RdBu'
fig = px.imshow(corr_matrix,
                labels={ "x": "Variáveis","y": " ", "color": "Correlação"},
                color_continuous_scale="RdBu", # Changed colorscale to a valid plotly colorscale
                text_auto=True)  # Exibe os valores diretamente no gráfico


# Ajustarcasas decimais
fig.update_traces(texttemplate="%{z:.2f}")
# Ajustar layout
fig.update_layout(title="Feature Correlation Heatmap - Temperatura", title_x=0.5, width=800, height=800)
fig.update_layout(
    #title_font=dict(size=24),  # Tamanho do título
    xaxis_title_font=dict(size=18),  # Tamanho do rótulo do eixo X
    yaxis_title_font=dict(size=18),  # Tamanho do rótulo do eixo Y
    font=dict(size=16))
fig.show()
fig.show('notebook_connected')
```



Escolher inputs

ML supervisionado

1 iteração

```
df1_filtrado3.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change'],
      dtype='object')
```

```
print(df1_filtrado3.isnull().sum())
```

Mostrar saída oculta

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Selecionar variáveis
features = ['T_mean', 'RH_mean', 'Calor_idx', 'CLTs', 'OCP', 'Q_OCP', 'AD', 'CR', 'dia_semana', 'trimestre', 'mes', 'ano', 'dia_ano',
       'dia_mes', 'semana_ano']
target = 'ELET_CONS_CLT'

df = df1_filtrado3.dropna(subset=features + [target])
X = df[features]
y = df[target]

# Normalizar os dados para SVR
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Dividir em treino e teste
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, shuffle=False)
```

```
import numpy as np

# Remover valores infinitos ou muito grandes
df1_filtrado3 = df1_filtrado3.replace([np.inf, -np.inf], np.nan)
df1_filtrado3 = df1_filtrado3.dropna(subset=['ELET_CONS_CLT'])
```

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score

# Inicializar modelos
tree_model = DecisionTreeRegressor(max_depth=10, min_samples_leaf= 4, min_samples_split= 10)
rf_model = RandomForestRegressor(n_estimators=100, max_depth=None, bootstrap= True)
svr_model = SVR(kernel='rbf', C=10, epsilon=0.1, gamma= 'scale')

# Treinar
tree_model.fit(X_train, y_train)
rf_model.fit(X_train, y_train)
svr_model.fit(X_train, y_train)

# Prever
y_pred_tree = tree_model.predict(X_test)
y_pred_rf = rf_model.predict(X_test)
y_pred_svr = svr_model.predict(X_test)
```

```
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score


resultados_acumulados = []

# Função para avaliar e guardar resultados de treino e teste
def avaliar_modelo(modelo, X_train, y_train, X_test, y_test, modelo_nome, iteracao):
    # Avaliação no treino
    y_pred_train = modelo.predict(X_train)
    mse_train = mean_squared_error(y_train, y_pred_train)
    rmse_train = np.sqrt(mse_train)
    mae_train = mean_absolute_error(y_train, y_pred_train)
    r2_train = r2_score(y_train, y_pred_train)

    print(f'{modelo_nome} (Iteração {iteracao}) - TREINO → MSE: {mse_train:.4f} | RMSE: {rmse_train:.2f} | MAE: {mae_train:.4f} | R²: {r2_train:.2f}')
    resultados_acumulados.append({
        'Modelo': modelo_nome,
        'Iteração': iteracao,
        'Conjunto': 'Treino',
        'MSE': round(mse_train, 4),
        'RMSE': round(rmse_train, 2),
        'MAE': round(mae_train, 4),
        'R²': round(r2_train, 2)
    })

    # Avaliação no teste
    y_pred_test = modelo.predict(X_test)
    mse_test = mean_squared_error(y_test, y_pred_test)
    rmse_test = np.sqrt(mse_test)
    mae_test = mean_absolute_error(y_test, y_pred_test)
    r2_test = r2_score(y_test, y_pred_test)

    print(f'{modelo_nome} (Iteração {iteracao}) - TESTE  → MSE: {mse_test:.4f} | RMSE: {rmse_test:.2f} | MAE: {mae_test:.4f} | R²: {r2_test:.2f}')
    resultados_acumulados.append({
        'Modelo': modelo_nome,
        'Iteração': iteracao,
        'Conjunto': 'Teste',
        'MSE': round(mse_test, 4),
        'RMSE': round(rmse_test, 2),
        'MAE': round(mae_test, 4),
        'R²': round(r2_test, 2)
    })
```

```
# =========================
# Chamar a função para cada modelo
# =========================
avaliar_modelo(tree_model, X_train, y_train, X_test, y_test, 'Árvore de Decisão', 1)
avaliar_modelo(rf_model, X_train, y_train, X_test, y_test, 'Random Forest', 1)
avaliar_modelo(svr_model, X_train, y_train, X_test, y_test, 'SVR', 1)  # SVR com dados escalados

# Criar DataFrame com todos os resultados
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nTabela de Avaliação:")
print(df_resultados)
```

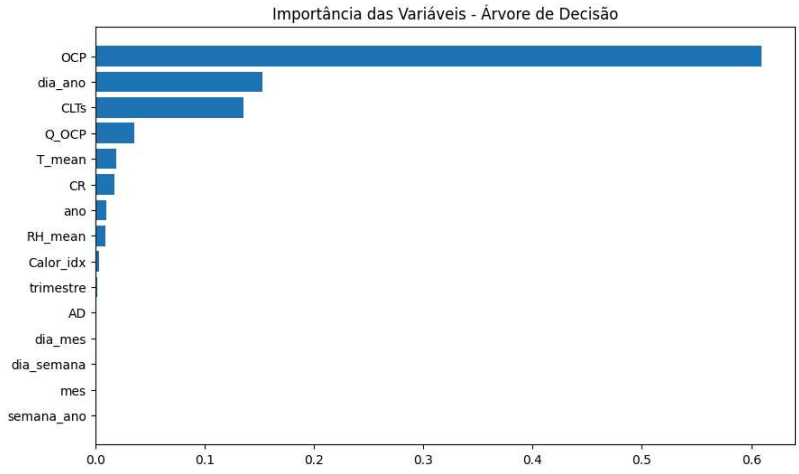
```
Árvore de Decisão (Iteração 1) - TREINO → MSE: 4.3337 | RMSE: 2.08 | MAE: 0.7089 | R²: 0.81
Árvore de Decisão (Iteração 1) - TESTE  → MSE: 1.4757 | RMSE: 1.21 | MAE: 0.9388 | R²: 0.41
Random Forest (Iteração 1) - TREINO → MSE: 0.9935 | RMSE: 1.00 | MAE: 0.3682 | R²: 0.96
Random Forest (Iteração 1) - TESTE  → MSE: 0.8964 | RMSE: 0.95 | MAE: 0.7417 | R²: 0.64
SVR (Iteração 1) - TREINO → MSE: 5.3172 | RMSE: 2.31 | MAE: 0.5493 | R²: 0.76
SVR (Iteração 1) - TESTE  → MSE: 0.7506 | RMSE: 0.87 | MAE: 0.6945 | R²: 0.70

Tabela de Avaliação:
             Modelo  Iteração Conjunto     MSE  RMSE     MAE    R²
0  Árvore de Decisão         1   Treino  4.3337  2.08  0.7089  0.81
1  Árvore de Decisão         1    Teste  1.4757  1.21  0.9388  0.41
2      Random Forest         1   Treino  0.9935  1.00  0.3682  0.96
3      Random Forest         1    Teste  0.8964  0.95  0.7417  0.64
4                SVR         1   Treino  5.3172  2.31  0.5493  0.76
5                SVR         1    Teste  0.7506  0.87  0.6945  0.70
```

```
df_resultados
```

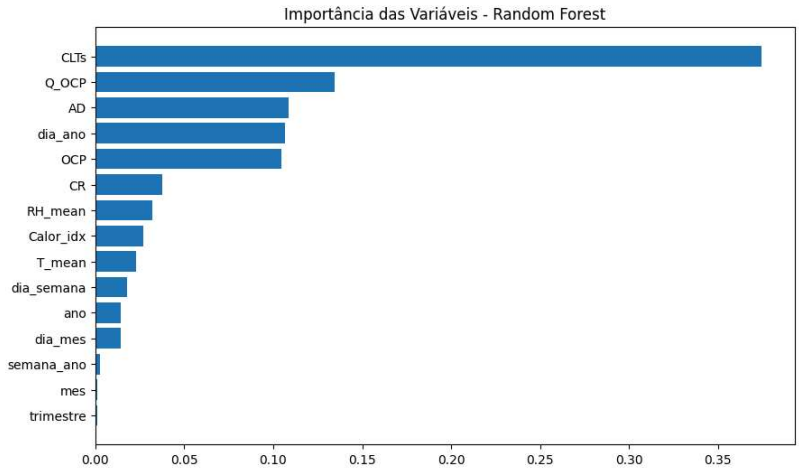
	Modelo	Iteração	Conjunto	MSE	RMSE	MAE	R²
0	Árvore de Decisão	1	Treino	4.3337	2.08	0.7089	0.81
1	Árvore de Decisão	1	Teste	1.4757	1.21	0.9388	0.41
2	Random Forest	1	Treino	0.9935	1.00	0.3682	0.96
3	Random Forest	1	Teste	0.8964	0.95	0.7417	0.64
4	SVR	1	Treino	5.3172	2.31	0.5493	0.76
5	SVR	1	Teste	0.7506	0.87	0.6945	0.70

```
import pandas as pd
import matplotlib.pyplot as plt

# Importância das features
importancias_tree = tree_model.feature_importances_
```

```
# Visualizar em gráfico
features = X.columns  # ou X_train.columns, dependendo do que usaste
df_importancia_tree = pd.DataFrame({'Feature': features, 'Importância': importancias_tree})
df_importancia_tree = df_importancia_tree.sort_values(by='Importância', ascending=False)

# Plot
plt.figure(figsize=(10,6))
plt.barh(df_importancia_tree['Feature'], df_importancia_tree['Importância'])
plt.gca().invert_yaxis()
plt.title('Importância das Variáveis - Árvore de Decisão')
plt.show()
```

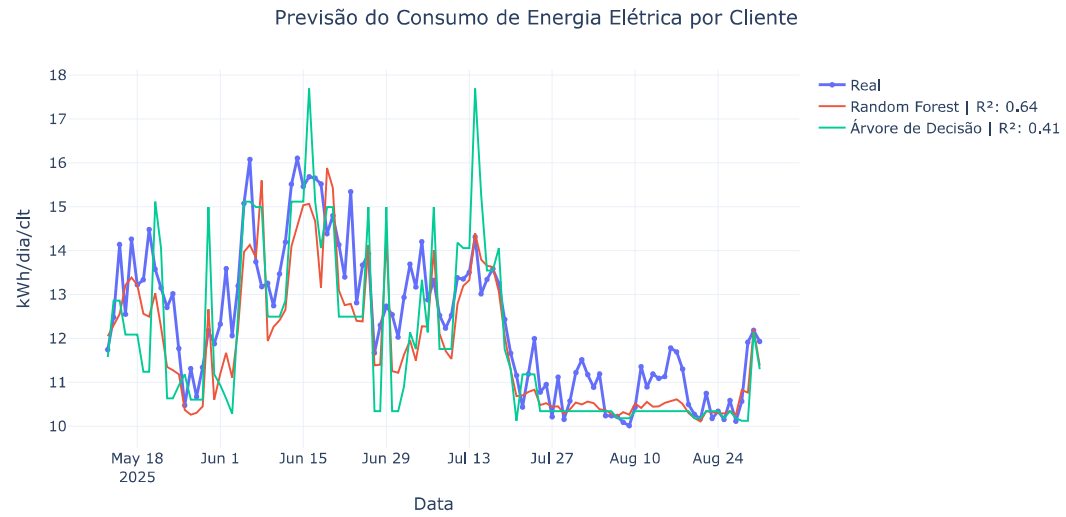



```
importancias_rf = rf_model.feature_importances_

df_importancia_rf = pd.DataFrame({'Feature': features, 'Importância': importancias_rf})
df_importancia_rf = df_importancia_rf.sort_values(by='Importância', ascending=False)

plt.figure(figsize=(10,6))
plt.barh(df_importancia_rf['Feature'], df_importancia_rf['Importância'])
plt.gca().invert_yaxis()
plt.title('Importância das Variáveis - Random Forest')
plt.show()
```

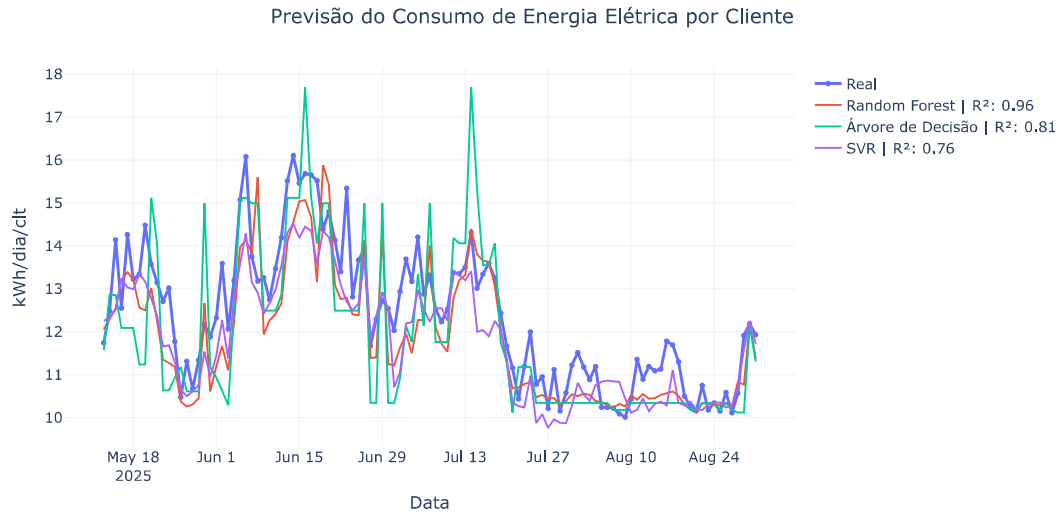



```
import plotly.graph_objects as go

fig = go.Figure()

# Série real
fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_test.values,
    mode='lines+markers',
    name='Real',
    line=dict( width=3)
))

# Previsões
r2_rf = df_resultados.loc[(df_resultados['Modelo'] == 'Random Forest') &
    (df_resultados['Iteração'] == 1) &
    (df_resultados['Conjunto'] == 'Teste'),
    'R²'].values[0]

fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_pred_rf,
    mode='lines',
    name=f'Random Forest | R²: {r2_rf:.2f}'
))

r2_tree = df_resultados.loc[
    (df_resultados['Modelo'] == 'Árvore de Decisão') &
    (df_resultados['Iteração'] == 1) &
    (df_resultados['Conjunto'] == 'Teste'),
    'R²'].values[0]

fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_pred_tree,
    mode='lines',
    name=f'Árvore de Decisão | R²: {r2_tree:.2f}'
))
"""
r2_svr = df_resultados.loc[(df_resultados['Modelo'] == 'SVR')  &
    (df_resultados['Iteração'] == 1) &
    (df_resultados['Conjunto'] == 'Teste'),
    'R²'].values[0]

fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_pred_svr,
```

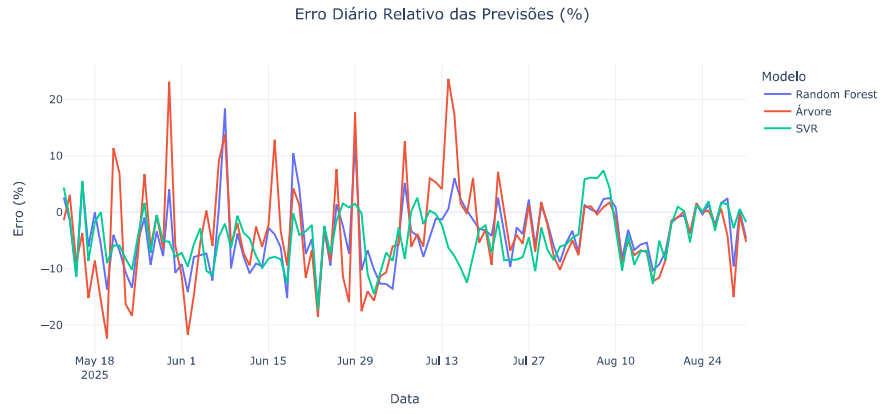
```
    mode='lines',
    name=f'SVR | R²: {r2_svr:.2f}'
))"""

# Layout
fig.update_layout(
   title='Previsão do Consumo de Energia Elétrica por Cliente',
    xaxis_title='Data',
    yaxis_title='kWh/dia/clt',
    width=1200,
    height=600,
    title_x=0.5,
    font=dict(size=16),
   # legend=dict(tickfont=dict(size=10)),
    template='plotly_white'
)

fig.show()
```




```
import plotly.graph_objects as go

fig = go.Figure()

# Série real
fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_test.values,
    mode='lines+markers',
    name='Real',
    line=dict( width=3)
))

# Previsões
r2_rf = df_resultados.loc[(df_resultados['Modelo'] == 'Random Forest') &
    (df_resultados['Iteração'] == 1),
    'R²'].values[0]

fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_pred_rf,
    mode='lines',
    name=f'Random Forest | R²: {r2_rf:.2f}'
))

r2_tree = df_resultados.loc[
    (df_resultados['Modelo'] == 'Árvore de Decisão') &
    (df_resultados['Iteração'] == 1),
    'R²'].values[0]

fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_pred_tree,
    mode='lines',
    name=f'Árvore de Decisão | R²: {r2_tree:.2f}'
))

r2_svr = df_resultados.loc[(df_resultados['Modelo'] == 'SVR')  &
    (df_resultados['Iteração'] == 1),
    'R²'].values[0]

fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_pred_svr,
    mode='lines',
    name=f'SVR | R²: {r2_svr:.2f}'
))

# Layout
fig.update_layout(
   title='Previsão do Consumo de Energia Elétrica por Cliente',
    xaxis_title='Data',
    yaxis_title='kWh/dia/clt',
    width=1200,
    height=600,
    title_x=0.5,
    font=dict(size=16),
   # legend=dict(tickfont=dict(size=10)),
    template='plotly_white'
)

fig.show()
```




```
import pandas as pd
import plotly.express as px

# Calcular erro relativo (%) para cada modelo
erro_rf = ((y_pred_rf - y_test.values) / y_test.values) * 100
erro_tree = ((y_pred_tree - y_test.values) / y_test.values) * 100
erro_svr = ((y_pred_svr - y_test.values) / y_test.values) * 100

# Criar DataFrame com datas e erros
df_erro = pd.DataFrame({
    'Data': y_test.index,
    'Random Forest': erro_rf,
    'Árvore': erro_tree,
    'SVR': erro_svr
})

# Transformar para formato longo (tidy)
df_erro_melt = df_erro.melt(id_vars='Data', var_name='Modelo', value_name='Erro [%]')

# Criar gráfico interativo
fig = px.line(df_erro_melt, x='Data', y='Erro [%]', color='Modelo',
              title='Erro Diário Relativo das Previsões (%)',
              labels={'Erro [%]': 'Erro (%)', 'Data': 'Data'},
              template='plotly_white')

fig.update_layout(width=1000, height=500, title_x=0.5)
fig.show()
```




```
df1_filtrado3.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change'],
      dtype='object')
```

2º iteração

```
print(df1_filtrado3.isnull().sum())
```

```
T_mean                      0
T_max                       0
T_min                       0
T_rg                        0
RH_mean                     0
RH_max                      0
RH_min                      0
RH_rg                       0
Calor_idx                   0
ELET_C                      0
EP.C                        0
50S.C                       0
COZ.50S.C                   0
COZ.EP.C                    0
PST.C                       0
LAV.C                       0
Q_OCP                       0
OCP                         0
OCP[%]                      0
AD                          0
CR                          0
CLTs                        0
ELET_CONS_CLT               0
```

```
CONS_Q                          0
AGUA_C                          0
GAS_C                           0
dia_semana                      0
trimestre                       0
mes                             0
ano                             0
dia_ano                         0
dia_mes                         0
semana_ano                      0
feriado                         0
AGUA_CONS_CLT                   0
GAS_CONS_CLT                    0
CO2e_ELET                       0
CO2e_AGUA                       0
CO2e_GAS                        0
Total_CO2e                      0
CO2e_QOCP                       0
ELET_CONS_CLT_lag1              0
ELET_CONS_CLT_lag7              3
ELET_CONS_CLT_rolling_mean      0
ELET_CONS_CLT_rolling_median    0
ELET_CONS_CLT_rolling_std       0
ELET_CONS_CLT_error_median      0
ELET_CONS_CLT_pct_change        0
dtype: int64
```

```
df1_filtrado3.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change'],
      dtype='object')
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Selecionar variáveis
#features = ['T_mean', 'RH_mean', 'Calor_idx', 'OCP', 'CLTs', 'CONS_CLT', 'dia_semana', 'mes']
features = ['T_mean', 'RH_mean', 'Calor_idx', 'CLTs', 'OCP', 'Q_OCP', 'AD', 'CR', 'dia_semana', 'trimestre', 'mes', 'ano', 'dia_ano',
       'dia_mes', 'semana_ano','ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
        'ELET_CONS_CLT_rolling_median','ELET_CONS_CLT_rolling_std', 'ELET_CONS_CLT_error_median',  'ELET_CONS_CLT_pct_change' ]

target = 'ELET_CONS_CLT'

df = df1_filtrado3.dropna(subset=features + [target])
X = df[features]
y = df[target]

# Normalizar os dados para SVR
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Dividir em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, shuffle=False)
```

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score

# Inicializar modelos
tree_model = DecisionTreeRegressor(max_depth=10, min_samples_leaf= 4, min_samples_split= 10)
rf_model = RandomForestRegressor(n_estimators=100, max_depth=None, bootstrap= True)
svr_model = SVR(kernel='rbf', C=10, epsilon=0.1, gamma= 'scale')

# Treinar
tree_model.fit(X_train, y_train)
rf_model.fit(X_train, y_train)
svr_model.fit(X_train, y_train)

# Prever
y_pred_tree = tree_model.predict(X_test)
y_pred_rf = rf_model.predict(X_test)
y_pred_svr = svr_model.predict(X_test)
```

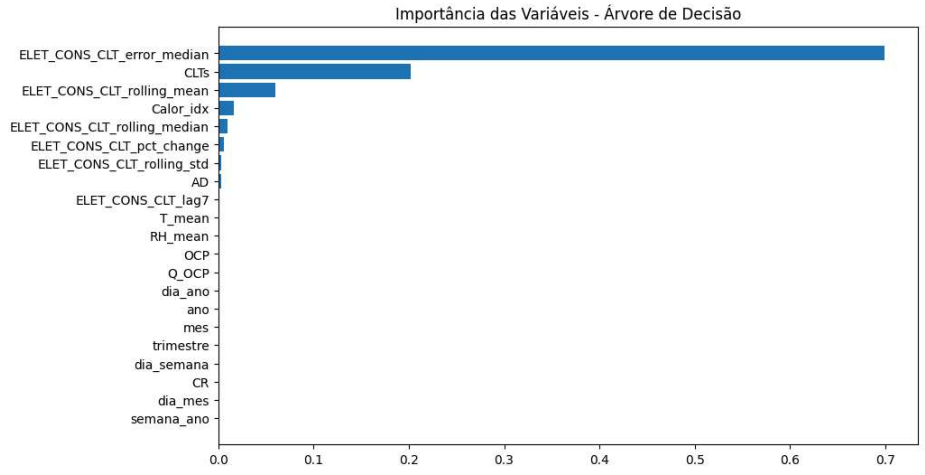
```
# ========================
# Chamando a função para cada modelo
# ========================
avaliar_modelo(tree_model, X_train, y_train, X_test, y_test, 'Árvore de Decisão', 2)
avaliar_modelo(rf_model, X_train, y_train, X_test, y_test, 'Random Forest', 2)
avaliar_modelo(svr_model, X_train, y_train, X_test, y_test, 'SVR', 2)  # SVR com dados escalados

# Criar DataFrame com todos os resultados
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nTabela de Avaliação:")
print(df_resultados)
```

```
Árvore de Decisão (Iteração 2) - TREINO → MSE: 3.7421 | RMSE: 1.93 | MAE: 0.4133 | R²: 0.83
Árvore de Decisão (Iteração 2) - TESTE  → MSE: 0.3438 | RMSE: 0.59 | MAE: 0.3803 | R²: 0.86
Random Forest (Iteração 2) - TREINO → MSE: 0.7137 | RMSE: 0.84 | MAE: 0.1967 | R²: 0.97
Random Forest (Iteração 2) - TESTE  → MSE: 0.0818 | RMSE: 0.29 | MAE: 0.1801 | R²: 0.97
SVR (Iteração 2) - TREINO → MSE: 6.1005 | RMSE: 2.47 | MAE: 0.2629 | R²: 0.72
SVR (Iteração 2) - TESTE  → MSE: 0.0836 | RMSE: 0.29 | MAE: 0.2376 | R²: 0.97

Tabela de Avaliação:
              Modelo  Iteração Conjunto     MSE  RMSE     MAE    R²
0   Árvore de Decisão         1   Treino  4.3337  2.08  0.7089  0.81
1   Árvore de Decisão         1    Teste  1.4757  1.21  0.9388  0.41
2       Random Forest         1   Treino  0.9935  1.00  0.3682  0.96
3       Random Forest         1    Teste  0.8964  0.95  0.7417  0.64
4                 SVR         1   Treino  5.3172  2.31  0.5493  0.76
5                 SVR         1    Teste  0.7506  0.87  0.6945  0.70
6   Árvore de Decisão         2   Treino  3.7421  1.93  0.4133  0.83
7   Árvore de Decisão         2    Teste  0.3438  0.59  0.3803  0.86
8       Random Forest         2   Treino  0.7137  0.84  0.1967  0.97
9       Random Forest         2    Teste  0.0818  0.29  0.1801  0.97
10                SVR         2   Treino  6.1005  2.47  0.2629  0.72
11                SVR         2    Teste  0.0836  0.29  0.2376  0.97
```

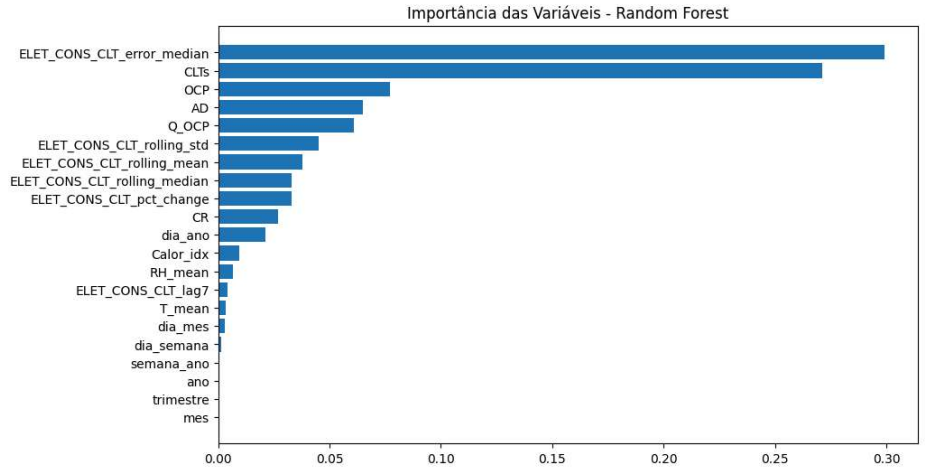
Comece a programar ou gerar com a IA.

```
import pandas as pd
import matplotlib.pyplot as plt

# Importância das features
importancias_tree = tree_model.feature_importances_

# Visualizar em gráfico
features = X.columns  # ou X_train.columns, dependendo do que usaste
```

```
df_importancia_tree = pd.DataFrame({'Feature': features, 'Importância': importancias_tree})
df_importancia_tree = df_importancia_tree.sort_values(by='Importância', ascending=False)

# Plot
plt.figure(figsize=(10,6))
plt.barh(df_importancia_tree['Feature'], df_importancia_tree['Importância'])
plt.gca().invert_yaxis()
plt.title('Importância das Variáveis - Árvore de Decisão')
plt.show()
```

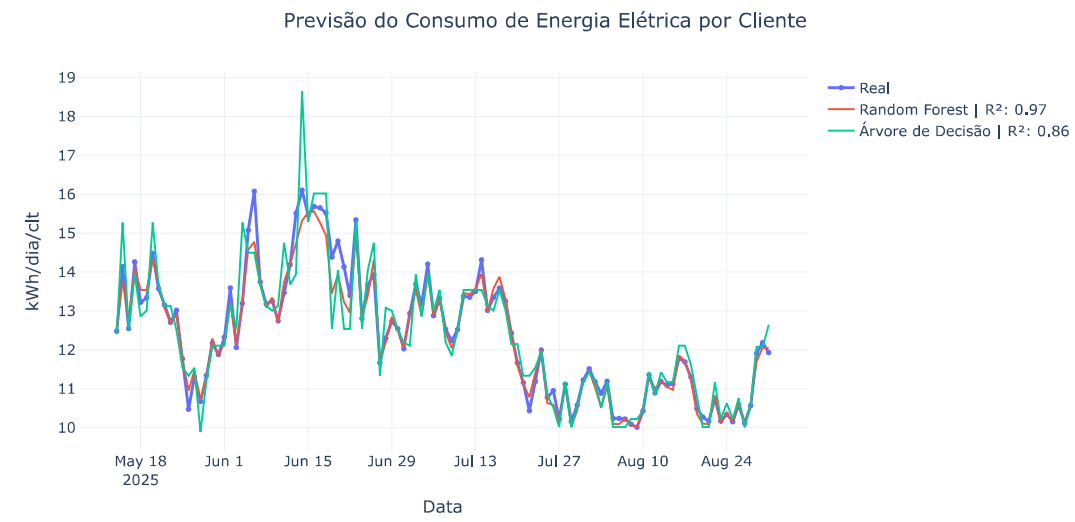



```
importancias_rf = rf_model.feature_importances_

df_importancia_rf = pd.DataFrame({'Feature': features, 'Importância': importancias_rf})
df_importancia_rf = df_importancia_rf.sort_values(by='Importância', ascending=False)

plt.figure(figsize=(10,6))
plt.barh(df_importancia_rf['Feature'], df_importancia_rf['Importância'])
plt.gca().invert_yaxis()
plt.title('Importância das Variáveis - Random Forest')
plt.show()
```

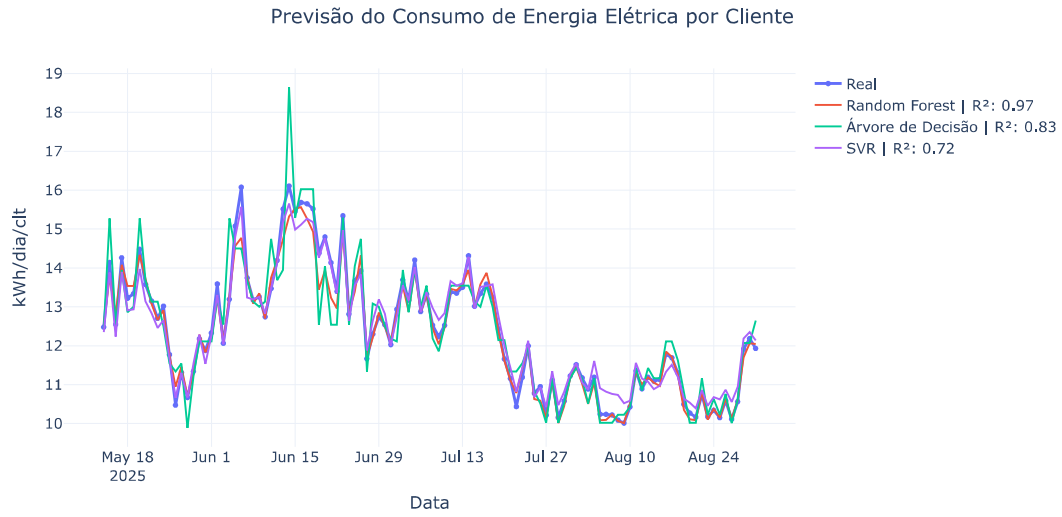



```
import plotly.graph_objects as go

fig = go.Figure()

# Série real
fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_test.values,
    mode='lines+markers',
    name='Real',
    line=dict( width=3)
))

# Previsões
r2_rf = df_resultados.loc[(df_resultados['Modelo'] == 'Random Forest') &
    (df_resultados['Iteração'] == 2) &
    (df_resultados['Conjunto'] == 'Teste'),
    'R²'].values[0]

fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_pred_rf,
    mode='lines',
    name=f'Random Forest | R²: {r2_rf:.2f}'
))

r2_tree = df_resultados.loc[
    (df_resultados['Modelo'] == 'Árvore de Decisão') &
    (df_resultados['Iteração'] == 2) &
    (df_resultados['Conjunto'] == 'Teste'),
    'R²'].values[0]

fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_pred_tree,
    mode='lines',
    name=f'Árvore de Decisão | R²: {r2_tree:.2f}'
))
"""
r2_svr = df_resultados.loc[(df_resultados['Modelo'] == 'SVR')  &
    (df_resultados['Iteração'] == 1) &
    (df_resultados['Conjunto'] == 'Teste'),
    'R²'].values[0]

fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_pred_svr,
    mode='lines',
```

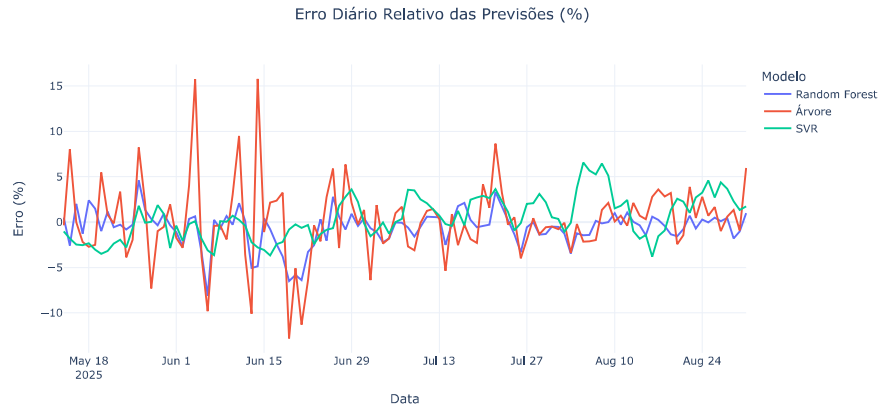
```
    name=f'SVR | R²: {r2_svr:.2f}'
))"""

# Layout
fig.update_layout(
   title='Previsão do Consumo de Energia Elétrica por Cliente',
    xaxis_title='Data',
    yaxis_title='kWh/dia/clt',
    width=1200,
    height=600,
    title_x=0.5,
    font=dict(size=16),
   # legend=dict(tickfont=dict(size=10)),
    template='plotly_white'
)

fig.show()
```




```
import plotly.graph_objects as go

fig = go.Figure()

# Série real
fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_test.values,
    mode='lines+markers',
    name='Real',
    line=dict( width=3)
))

# Previsões
r2_rf = df_resultados.loc[(df_resultados['Modelo'] == 'Random Forest') &
    (df_resultados['Iteração'] == 2),
    'R²'].values[0]

fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_pred_rf,
    mode='lines',
    name=f'Random Forest | R²: {r2_rf:.2f}'
))

r2_tree = df_resultados.loc[
    (df_resultados['Modelo'] == 'Árvore de Decisão') &
    (df_resultados['Iteração'] == 2),
    'R²'].values[0]

fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_pred_tree,
    mode='lines',
    name=f'Árvore de Decisão | R²: {r2_tree:.2f}'
))

r2_svr = df_resultados.loc[(df_resultados['Modelo'] == 'SVR')  &
    (df_resultados['Iteração'] == 2),
    'R²'].values[0]

fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_pred_svr,
    mode='lines',
    name=f'SVR | R²: {r2_svr:.2f}'
))

# Layout
fig.update_layout(
   title='Previsão do Consumo de Energia Elétrica por Cliente',
    xaxis_title='Data',
    yaxis_title='kWh/dia/clt',
    width=1200,
    height=600,
    title_x=0.5,
    font=dict(size=16),
   # legend=dict(tickfont=dict(size=10)),
    template='plotly_white'
)

fig.show()
```




```
import pandas as pd
import plotly.express as px

# Calcular erro relativo (%) para cada modelo
erro_rf = ((y_pred_rf - y_test.values) / y_test.values) * 100
erro_tree = ((y_pred_tree - y_test.values) / y_test.values) * 100
erro_svr = ((y_pred_svr - y_test.values) / y_test.values) * 100

# Criar DataFrame com datas e erros
df_erro = pd.DataFrame({
    'Data': y_test.index,
    'Random Forest': erro_rf,
    'Árvore': erro_tree,
    'SVR': erro_svr
})

# Transformar para formato longo (tidy)
df_erro_melt = df_erro.melt(id_vars='Data', var_name='Modelo', value_name='Erro [%]')

# Criar gráfico interativo
fig = px.line(df_erro_melt, x='Data', y='Erro [%]', color='Modelo',
              title='Erro Diário Relativo das Previsões (%)',
              labels={'Erro [%]': 'Erro (%)', 'Data': 'Data'},
              template='plotly_white')

fig.update_layout(width=1000, height=500, title_x=0.5)
fig.show()
```



3ª iteração

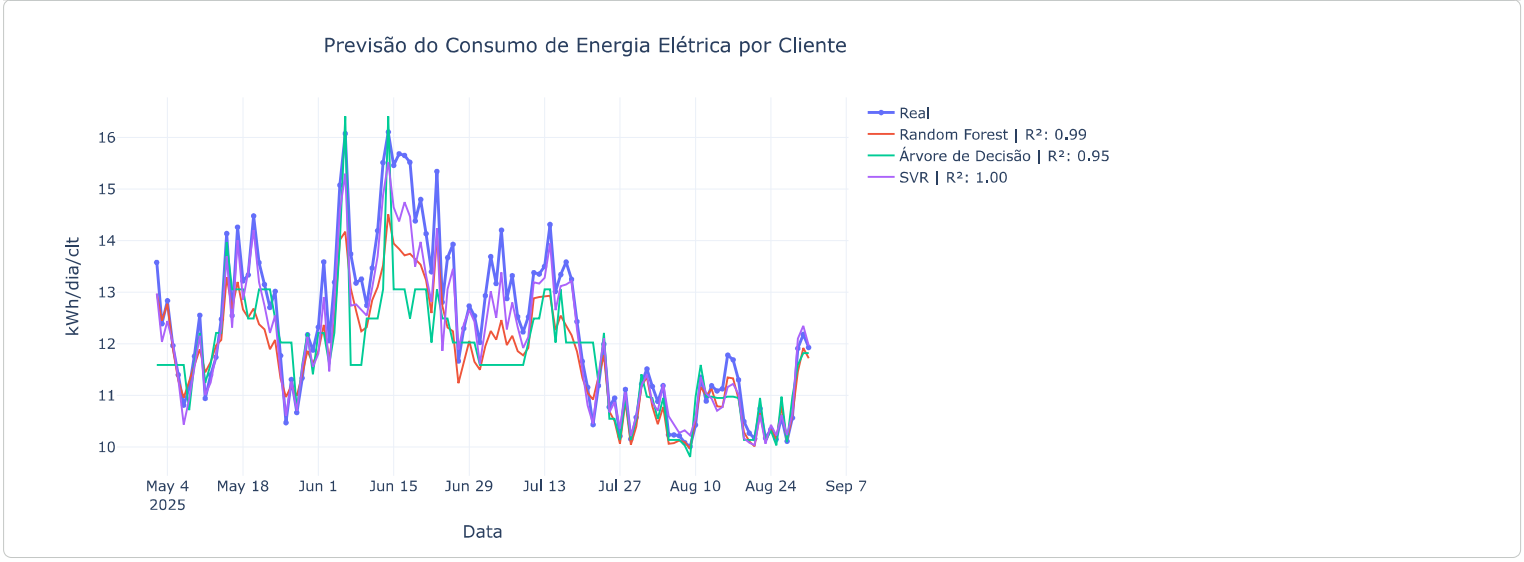
```
from sklearn.preprocessing import StandardScaler


df1_filtrado3.index = pd.to_datetime(df1_filtrado3.index)

# Separar treino e teste por datas
df1_filtrado3_treino = df1_filtrado3.loc[
    #(df1_filtrado3.index > '2023-04-01') &
    #(df1_filtrado3.index <= '2023-10-31') |
    (df1_filtrado3.index > '2024-04-01') &
    (df1_filtrado3.index <= '2024-10-31')
]

df1_filtrado3_teste = df1_filtrado3.loc[df1_filtrado3.index > '2025-05-01']

# Selecionar features e target
features = ['T_mean', 'RH_mean', 'Calor_idx', 'CLTs', 'OCP', 'Q_OCP', 'AD', 'CR', 'dia_semana', 'trimestre', 'mes', 'ano', 'dia_ano',
       'dia_mes', 'semana_ano','ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
        'ELET_CONS_CLT_rolling_median','ELET_CONS_CLT_rolling_std', 'ELET_CONS_CLT_error_median',  'ELET_CONS_CLT_pct_change' ]

target = 'ELET_CONS_CLT'

X_train = df1_filtrado3_treino[features]
y_train = df1_filtrado3_treino[target]

X_test = df1_filtrado3_teste[features]
y_test = df1_filtrado3_teste[target]

# Normalizar os dados com base apenas no treino
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score

# Inicializar modelos
tree_model = DecisionTreeRegressor(max_depth=10, min_samples_leaf= 4, min_samples_split= 10)
rf_model = RandomForestRegressor(n_estimators=100, max_depth=None, bootstrap= True)
svr_model = SVR(kernel='rbf', C=10, epsilon=0.1, gamma= 'scale')

# Treinar
tree_model.fit(X_train_scaled, y_train)
rf_model.fit(X_train_scaled, y_train)
svr_model.fit(X_train_scaled, y_train)

# Prever
y_pred_tree = tree_model.predict(X_test_scaled)
y_pred_rf = rf_model.predict(X_test_scaled)
y_pred_svr = svr_model.predict(X_test_scaled)
```

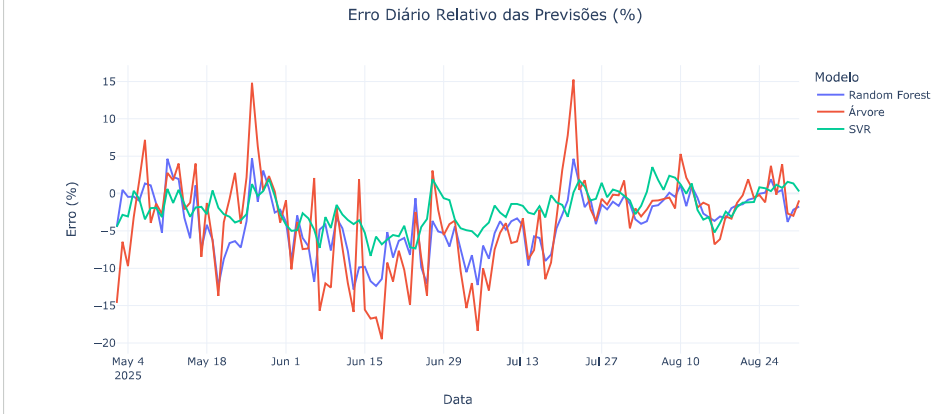
```
# ========================
# Chamar a função para cada modelo
# ========================
avaliar_modelo(tree_model, X_train_scaled, y_train, X_test_scaled, y_test, 'Árvore de Decisão', 3)
avaliar_modelo(rf_model, X_train_scaled, y_train, X_test_scaled, y_test, 'Random Forest', 3)
avaliar_modelo(svr_model, X_train_scaled, y_train, X_test_scaled, y_test, 'SVR', 3)

# Criar DataFrame com todos os resultados
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nTabela de Avaliação:")
print(df_resultados)
```

```
Árvore de Decisão (Iteração 3) - TREINO → MSE: 0.3084 | RMSE: 0.56 | MAE: 0.2801 | R²: 0.95
Árvore de Decisão (Iteração 3) - TESTE  → MSE: 1.0447 | RMSE: 1.02 | MAE: 0.7326 | R²: 0.56
Random Forest (Iteração 3) - TREINO → MSE: 0.0769 | RMSE: 0.28 | MAE: 0.1294 | R²: 0.99
Random Forest (Iteração 3) - TESTE  → MSE: 0.6032 | RMSE: 0.78 | MAE: 0.5853 | R²: 0.75
SVR (Iteração 3) - TREINO → MSE: 0.0155 | RMSE: 0.12 | MAE: 0.0900 | R²: 1.00
SVR (Iteração 3) - TESTE  → MSE: 0.1855 | RMSE: 0.43 | MAE: 0.3345 | R²: 0.92

Tabela de Avaliação:
              Modelo  Iteração Conjunto     MSE  RMSE     MAE    R²
0   Árvore de Decisão         1   Treino  4.3337  2.08  0.7089  0.81
1   Árvore de Decisão         1    Teste  1.4757  1.21  0.9388  0.41
2       Random Forest         1   Treino  0.9935  1.00  0.3682  0.96
3       Random Forest         1    Teste  0.8964  0.95  0.7417  0.64
4                 SVR         1   Treino  5.3172  2.31  0.5493  0.76
5                 SVR         1    Teste  0.7506  0.87  0.6945  0.70
6   Árvore de Decisão         2   Treino  3.7421  1.93  0.4133  0.83
7   Árvore de Decisão         2    Teste  0.3438  0.59  0.3803  0.86
8       Random Forest         2   Treino  0.7137  0.84  0.1967  0.97
9       Random Forest         2    Teste  0.0818  0.29  0.1801  0.97
10                SVR         2   Treino  6.1005  2.47  0.2629  0.72
11                SVR         2    Teste  0.0836  0.29  0.2376  0.97
12  Árvore de Decisão         3   Treino  0.3084  0.56  0.2801  0.95
13  Árvore de Decisão         3    Teste  1.0447  1.02  0.7326  0.56
14      Random Forest         3   Treino  0.0769  0.28  0.1294  0.99
15      Random Forest         3    Teste  0.6032  0.78  0.5853  0.75
16                SVR         3   Treino  0.0155  0.12  0.0900  1.00
17                SVR         3    Teste  0.1855  0.43  0.3345  0.92
```

```
import plotly.graph_objects as go

fig = go.Figure()

# Série real
fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_test.values,
    mode='lines+markers',
    name='Real',
    line=dict( width=3)
))

# Previsões
r2_rf = df_resultados.loc[(df_resultados['Modelo'] == 'Random Forest') &
    (df_resultados['Iteração'] == 3),
    'R²'].values[0]

fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_pred_rf,
    mode='lines',
    name=f'Random Forest | R²: {r2_rf:.2f}'
))

r2_tree = df_resultados.loc[
    (df_resultados['Modelo'] == 'Árvore de Decisão') &
    (df_resultados['Iteração'] == 3),
    'R²'].values[0]

fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_pred_tree,
    mode='lines',
    name=f'Árvore de Decisão | R²: {r2_tree:.2f}'
))

r2_svr = df_resultados.loc[(df_resultados['Modelo'] == 'SVR')  &
    (df_resultados['Iteração'] == 3),
    'R²'].values[0]

fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_pred_svr,
    mode='lines',
    name=f'SVR | R²: {r2_svr:.2f}'
))

# Layout
fig.update_layout(
   title='Previsão do Consumo de Energia Elétrica por Cliente',
    xaxis_title='Data',
    yaxis_title='kWh/dia/clt',
    width=1200,
    height=600,
    title_x=0.5,
    font=dict(size=16),
   # legend=dict(tickfont=dict(size=10)),
    template='plotly_white'
)

fig.show()
```




```
# Calcular erro relativo (%) para cada modelo
erro_rf = ((y_pred_rf - y_test.values) / y_test.values) * 100
erro_tree = ((y_pred_tree - y_test.values) / y_test.values) * 100
erro_svr = ((y_pred_svr - y_test.values) / y_test.values) * 100

# Criar DataFrame com datas e erros
df_erro = pd.DataFrame({
    'Data': y_test.index,
    'Random Forest': erro_rf,
    'Árvore': erro_tree,
    'SVR': erro_svr
})

# Transformar para formato longo (tidy)
df_erro_melt = df_erro.melt(id_vars='Data', var_name='Modelo', value_name='Erro [%]')

# Criar gráfico interativo
fig = px.line(df_erro_melt, x='Data', y='Erro [%]', color='Modelo',
              title='Erro Diário Relativo das Previsões (%)',
              labels={'Erro [%]': 'Erro (%)', 'Data': 'Data'},
              template='plotly_white')

fig.update_layout(width=1000, height=500, title_x=0.5)
fig.show()
```




```
df1_filtrado3.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change'],
      dtype='object')
```

4ª iteração

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Selecionar variáveis preditoras
features = ['T_mean', 'CLTs', 'OCP', 'Q_OCP', 'AD', 'CR', 'dia_semana',  'dia_ano',
       'dia_mes', 'semana_ano'] #com variaveis nº1
#features = ['OCP', 'CLTs', 'mes', 'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       # 'ELET_CONS_CLT_rolling_median',
     #  'ELET_CONS_CLT_rolling_std', 'ELET_CONS_CLT_error_median',
      # 'ELET_CONS_CLT_pct_change' ] #com variaveis nº2
target = 'ELET_CONS_CLT'

#df = df1_filtrado3.dropna(subset=features + [target])
df1_filtrado3_treino = df1_filtrado3.loc[
    #(df1_filtrado3.index > '2023-04-01') &
    #(df1_filtrado3.index <= '2023-10-31') |
    (df1_filtrado3.index > '2024-04-01') &
    (df1_filtrado3.index <= '2024-10-31')
]

df1_filtrado3_teste = df1_filtrado3.loc[df1_filtrado3.index > '2025-05-01']

X_train = df1_filtrado3_treino[features]
y_train = df1_filtrado3_treino[target]

X_test = df1_filtrado3_teste[features]
y_test = df1_filtrado3_teste[target]
```

```
# Normalizar os dados com base apenas no treino
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Comece a programar ou gerar com a IA.

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score

# Inicializar modelos
tree_model = DecisionTreeRegressor(max_depth=10, min_samples_leaf= 4, min_samples_split= 10)
rf_model = RandomForestRegressor(n_estimators=100, max_depth=None, bootstrap= True)
svr_model = SVR(kernel='rbf', C=10, epsilon=0.1, gamma= 'scale')

# Treinar
tree_model.fit(X_train_scaled, y_train)
rf_model.fit(X_train_scaled, y_train)
svr_model.fit(X_train_scaled, y_train)

# Prever
y_pred_tree = tree_model.predict(X_test_scaled)
y_pred_rf = rf_model.predict(X_test_scaled)
y_pred_svr = svr_model.predict(X_test_scaled)
```

```
# =========================
# Chamando a função para cada modelo
# =========================
avaliar_modelo(tree_model, X_train_scaled, y_train, X_test_scaled, y_test, 'Árvore de Decisão', 4)
avaliar_modelo(rf_model, X_train_scaled, y_train, X_test_scaled, y_test, 'Random Forest', 4)
avaliar_modelo(svr_model, X_train_scaled, y_train, X_test_scaled, y_test, 'SVR', 4)

# Criar DataFrame com todos os resultados
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nTabela de Avaliação:")
print(df_resultados)
```

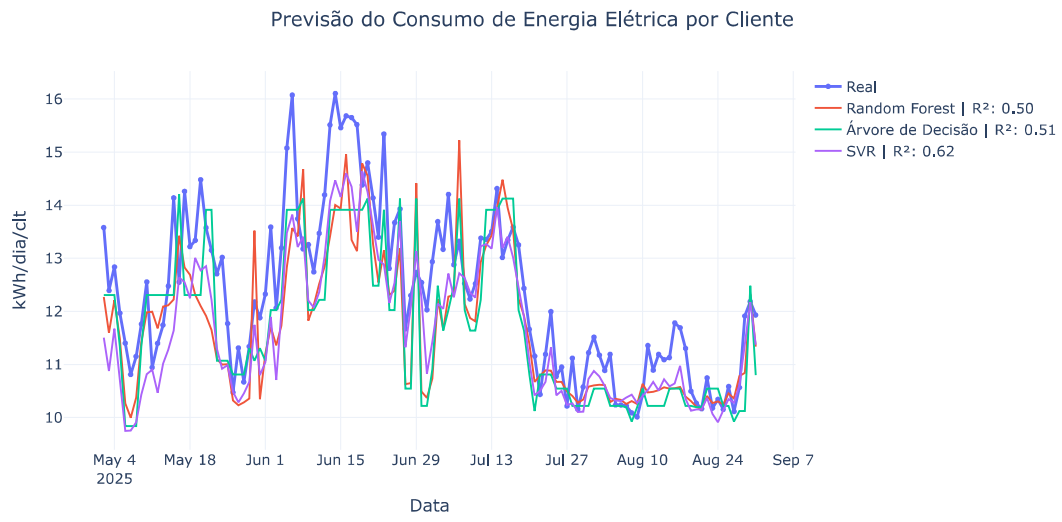
```
Árvore de Decisão (Iteração 4) - TREINO → MSE: 0.5531 | RMSE: 0.74 | MAE: 0.4124 | R²: 0.91
Árvore de Decisão (Iteração 4) - TESTE  → MSE: 1.1623 | RMSE: 1.08 | MAE: 0.8901 | R²: 0.51
Random Forest (Iteração 4) - TREINO → MSE: 0.1244 | RMSE: 0.35 | MAE: 0.1925 | R²: 0.98
Random Forest (Iteração 4) - TESTE  → MSE: 1.1921 | RMSE: 1.09 | MAE: 0.8701 | R²: 0.50
SVR (Iteração 4) - TREINO → MSE: 0.4586 | RMSE: 0.68 | MAE: 0.2897 | R²: 0.92
SVR (Iteração 4) - TESTE  → MSE: 0.9126 | RMSE: 0.96 | MAE: 0.7488 | R²: 0.62

Tabela de Avaliação:
              Modelo  Iteração Conjunto     MSE  RMSE     MAE    R²
0   Árvore de Decisão         1   Treino  4.3337  2.08  0.7089  0.81
1   Árvore de Decisão         1    Teste  1.4757  1.21  0.9388  0.41
2       Random Forest         1   Treino  0.9935  1.00  0.3682  0.96
3       Random Forest         1    Teste  0.8964  0.95  0.7417  0.64
4                 SVR         1   Treino  5.3172  2.31  0.5493  0.76
5                 SVR         1    Teste  0.7506  0.87  0.6945  0.70
6   Árvore de Decisão         2   Treino  3.7421  1.93  0.4133  0.83
7   Árvore de Decisão         2    Teste  0.3438  0.59  0.3803  0.86
8       Random Forest         2   Treino  0.7137  0.84  0.1967  0.97
9       Random Forest         2    Teste  0.0818  0.29  0.1801  0.97
10                SVR         2   Treino  6.1005  2.47  0.2629  0.72
11                SVR         2    Teste  0.0836  0.29  0.2376  0.97
12  Árvore de Decisão         3   Treino  0.3084  0.56  0.2801  0.95
13  Árvore de Decisão         3    Teste  1.0447  1.02  0.7326  0.56
14      Random Forest         3   Treino  0.0769  0.28  0.1294  0.99
15      Random Forest         3    Teste  0.6032  0.78  0.5853  0.75
16                SVR         3   Treino  0.0155  0.12  0.0900  1.00
17                SVR         3    Teste  0.1855  0.43  0.3345  0.92
18  Árvore de Decisão         4   Treino  0.5531  0.74  0.4124  0.91
19  Árvore de Decisão         4    Teste  1.1623  1.08  0.8901  0.51
20      Random Forest         4   Treino  0.1244  0.35  0.1925  0.98
21      Random Forest         4    Teste  1.1921  1.09  0.8701  0.50
22                SVR         4   Treino  0.4586  0.68  0.2897  0.92
23                SVR         4    Teste  0.9126  0.96  0.7488  0.62
```

```
df_resultados
```

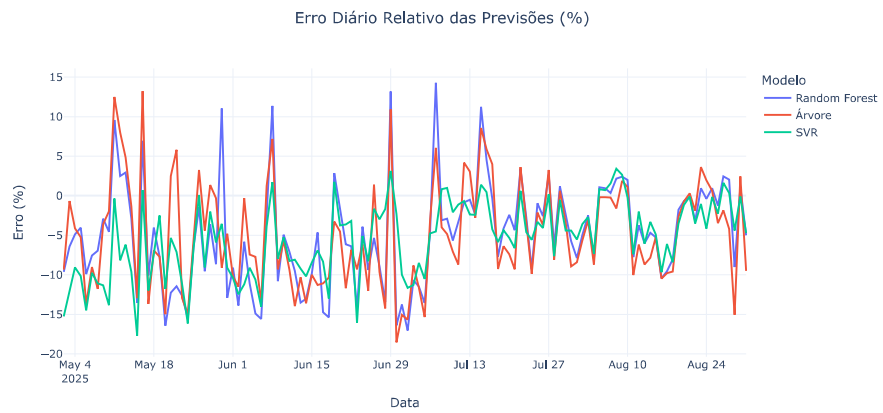
	Modelo	Iteração	Conjunto	MSE	RMSE	MAE	R²
0	Árvore de Decisão	1	Treino	4.3337	2.08	0.7089	0.81
1	Árvore de Decisão	1	Teste	1.4757	1.21	0.9388	0.41
2	Random Forest	1	Treino	0.9935	1.00	0.3682	0.96
3	Random Forest	1	Teste	0.8964	0.95	0.7417	0.64
4	SVR	1	Treino	5.3172	2.31	0.5493	0.76
5	SVR	1	Teste	0.7506	0.87	0.6945	0.70
6	Árvore de Decisão	2	Treino	3.7421	1.93	0.4133	0.83
7	Árvore de Decisão	2	Teste	0.3438	0.59	0.3803	0.86
8	Random Forest	2	Treino	0.7137	0.84	0.1967	0.97
9	Random Forest	2	Teste	0.0818	0.29	0.1801	0.97
10	SVR	2	Treino	6.1005	2.47	0.2629	0.72
11	SVR	2	Teste	0.0836	0.29	0.2376	0.97
12	Árvore de Decisão	3	Treino	0.3084	0.56	0.2801	0.95
13	Árvore de Decisão	3	Teste	1.0447	1.02	0.7326	0.56
14	Random Forest	3	Treino	0.0769	0.28	0.1294	0.99
15	Random Forest	3	Teste	0.6032	0.78	0.5853	0.75
16	SVR	3	Treino	0.0155	0.12	0.0900	1.00
17	SVR	3	Teste	0.1855	0.43	0.3345	0.92
18	Árvore de Decisão	4	Treino	0.5531	0.74	0.4124	0.91
19	Árvore de Decisão	4	Teste	1.1623	1.08	0.8901	0.51
20	Random Forest	4	Treino	0.1244	0.35	0.1925	0.98
21	Random Forest	4	Teste	1.1921	1.09	0.8701	0.50
22	SVR	4	Treino	0.4586	0.68	0.2897	0.92
23	SVR	4	Teste	0.9126	0.96	0.7488	0.62

```
import plotly.graph_objects as go

fig = go.Figure()

# Série real
fig.add_trace(go.Scatter(
```

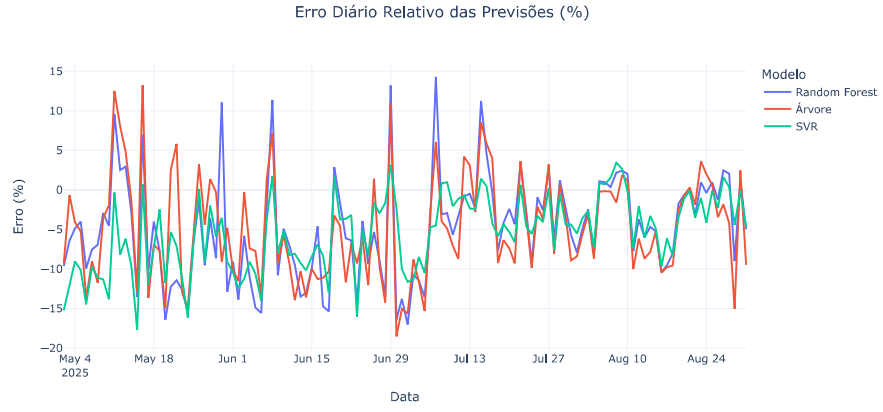
```
    x=y_test.index,
    y=y_test.values,
    mode='lines+markers',
    name='Real',
    line=dict( width=3)
))

# Previsões
r2_rf = df_resultados.loc[(df_resultados['Modelo'] == 'Random Forest') &
    (df_resultados['Iteração'] == 4) & (df_resultados['Conjunto'] == 'Teste'),
    'R²'].values[0]

fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_pred_rf,
    mode='lines',
    name=f'Random Forest | R²: {r2_rf:.2f}'
))

r2_tree = df_resultados.loc[
    (df_resultados['Modelo'] == 'Árvore de Decisão') &
    (df_resultados['Iteração'] == 4) & (df_resultados['Conjunto'] == 'Teste'),
    'R²'].values[0]

fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_pred_tree,
    mode='lines',
    name=f'Árvore de Decisão | R²: {r2_tree:.2f}'
))

r2_svr = df_resultados.loc[(df_resultados['Modelo'] == 'SVR')  &
    (df_resultados['Iteração'] == 4) & (df_resultados['Conjunto'] == 'Teste'),
    'R²'].values[0]

fig.add_trace(go.Scatter(
    x=y_test.index,
    y=y_pred_svr,
    mode='lines',
    name=f'SVR | R²: {r2_svr:.2f}'
))

# Layout
fig.update_layout(
   title='Previsão do Consumo de Energia Elétrica por Cliente',
    xaxis_title='Data',
    yaxis_title='kWh/dia/clt',
    width=1200,
    height=600,
    title_x=0.5,
    font=dict(size=16),
   # legend=dict(tickfont=dict(size=10)),
    template='plotly_white'
)

fig.show()
```




```
# Calcular erro relativo (%) para cada modelo
erro_rf = ((y_pred_rf - y_test.values) / y_test.values) * 100
erro_tree = ((y_pred_tree - y_test.values) / y_test.values) * 100
erro_svr = ((y_pred_svr - y_test.values) / y_test.values) * 100

# Criar DataFrame com datas e erros
df_erro = pd.DataFrame({
    'Data': y_test.index,
    'Random Forest': erro_rf,
    'Árvore': erro_tree,
    'SVR': erro_svr
})

# Transformar para formato longo (tidy)
df_erro_melt = df_erro.melt(id_vars='Data', var_name='Modelo', value_name='Erro [%]')

# Criar gráfico interativo
fig = px.line(df_erro_melt, x='Data', y='Erro [%]', color='Modelo',
              title='Erro Diário Relativo das Previsões (%)',
              labels={'Erro [%]': 'Erro (%)', 'Data': 'Data'},
              template='plotly_white')

fig.update_layout(width=1000, height=500, title_x=0.5)
fig.show()
```




```
import pandas as pd
import plotly.express as px

# Calcular erro relativo (%) para cada modelo
erro_rf = ((y_pred_rf - y_test.values) / y_test.values) * 100
erro_tree = ((y_pred_tree - y_test.values) / y_test.values) * 100
erro_svr = ((y_pred_svr - y_test.values) / y_test.values) * 100

# Criar DataFrame com datas e erros
df_erro = pd.DataFrame({
    'Data': y_test.index,
    'Random Forest': erro_rf,
    'Árvore': erro_tree,
    'SVR': erro_svr
})

# Transformar para formato longo (tidy)
df_erro_melt = df_erro.melt(id_vars='Data', var_name='Modelo', value_name='Erro [%]')

# Criar gráfico interativo
fig = px.line(df_erro_melt, x='Data', y='Erro [%]', color='Modelo',
              title='Erro Diário Relativo das Previsões (%)',
              labels={'Erro [%]': 'Erro (%)', 'Data': 'Data'},
              template='plotly_white')

fig.update_layout(width=1000, height=500, title_x=0.5)
fig.show()
```



Final

```
# Selecionar as mesmas features usadas no treino
features =  ['T_mean', 'CLTs', 'OCP', 'Q_OCP', 'AD', 'CR', 'dia_semana',  'dia_ano',
       'dia_mes', 'semana_ano' ]


df_pred = df1_filtrado3.dropna(subset=features + ['ELET_CONS_CLT'])

# Separar X e y
X_pred = df_pred[features]
y_real = df_pred['ELET_CONS_CLT']

# Normalizar com o mesmo scaler usado no treino
X_pred_scaled = scaler.transform(X_pred)

#Prever
y_pred_tree = tree_model.predict(X_pred_scaled)
y_pred_rf = rf_model.predict(X_pred_scaled)
y_pred_svr = svr_model.predict(X_pred_scaled)
```

```
# o DataFrame usado para gerar as previsões
df_svr_pred = pd.DataFrame(y_pred_svr, index=df_pred.index, columns=['previsão_svr'])
df_rf_pred = pd.DataFrame(y_pred_rf, index=df_pred.index, columns=['previsão_rf'])
df_tree_pred = pd.DataFrame(y_pred_tree, index=df_pred.index, columns=['previsão_tree'])

# fazer o merge com df_pred
df_pred = df_pred\
    .merge(df_svr_pred, left_index=True, right_index=True)\
    .merge(df_rf_pred, left_index=True, right_index=True)\
    .merge(df_tree_pred, left_index=True, right_index=True)
```

```
df_pred.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
```

```
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change',
       'previsão_svr', 'previsão_rf', 'previsão_tree'],
      dtype='object')
```

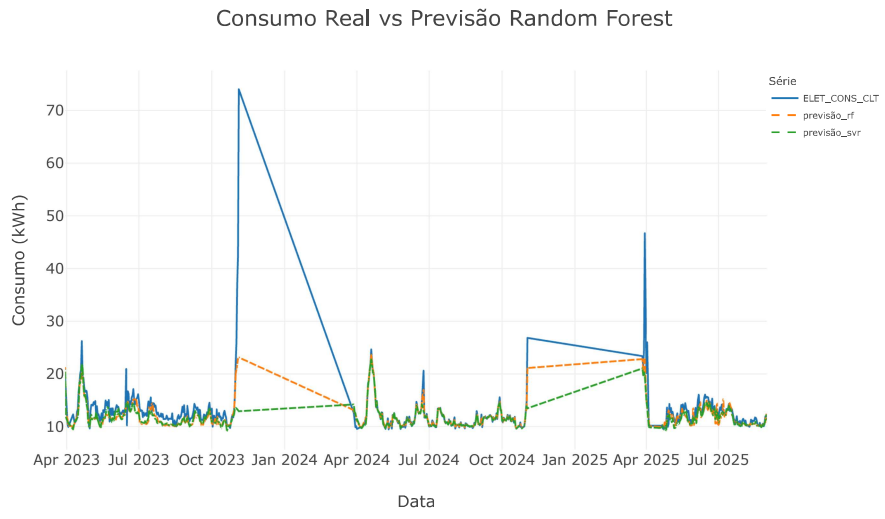
Comece a programar ou gerar com a IA.

```
df_pred.tail()
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	ELET_CONS_CLT_lag1	ELET_CONS_CLT_lag7	ELET_CONS_CLT_rolling_mean	ELET_CONS_CLT_rolling_median	ELET_CONS_CLT_rolling_s
2025-08-27	23.41699	31.9	18.3	13.6	75.88219	89.0	51.5	37.5	23.98021	12914.0	...	10.585227	10.270170	10.325795	10.176106	0.2480
2025-08-28	22.23625	30.2	18.3	11.9	80.14818	100.0	54.7	45.3	22.37211	12826.0	...	10.112764	10.165070	10.382938	10.341620	0.2508
2025-08-29	22.64507	34.0	17.5	16.5	79.59112	91.5	54.4	37.1	22.82476	12985.0	...	10.565074	10.747419	10.549428	10.341620	0.6313
2025-08-30	24.65698	34.5	19.7	14.8	78.81356	92.0	56.4	35.6	25.36257	12416.0	...	11.912844	10.176106	10.836340	10.565074	0.8513
2025-08-31	22.51888	30.1	18.7	11.4	83.64605	98.5	59.9	38.6	22.13280	12372.0	...	12.184495	10.341620	11.063333	10.585227	0.9074

5 rows × 51 columns

```
import plotly.express as px
import pandas as pd

# Selecionar os dados
df_plot = df_pred[['ELET_CONS_CLT', 'previsão_rf', 'previsão_svr']].dropna().copy()

# Resetar índice e renomear para 'Data'
df_plot = df_plot.reset_index().rename(columns={df_plot.index.name or 'index': 'Data'})

# Transformar para formato longo
df_plot_melt = df_plot.melt(id_vars='Data',
                            value_vars=['ELET_CONS_CLT', 'previsão_rf', 'previsão_svr'],
                            var_name='Tipo', value_name='Consumo')

# Criar gráfico com estilo de linha definido manualmente
fig = px.line(df_plot_melt, x='Data', y='Consumo', color='Tipo',
              line_dash='Tipo',
              line_dash_map={'ELET_CONS_CLT': 'solid', 'previsão_rf': 'dash'},
              title='Consumo Real vs Previsão Random Forest',
              labels={'Data': 'Data', 'Consumo': 'Consumo (kWh)', 'Tipo': 'Série'})

# Atualizar layout para legenda limpa
fig.update_traces(showlegend=True)
fig.update_layout( hovermode='x unified')
fig.show()
```

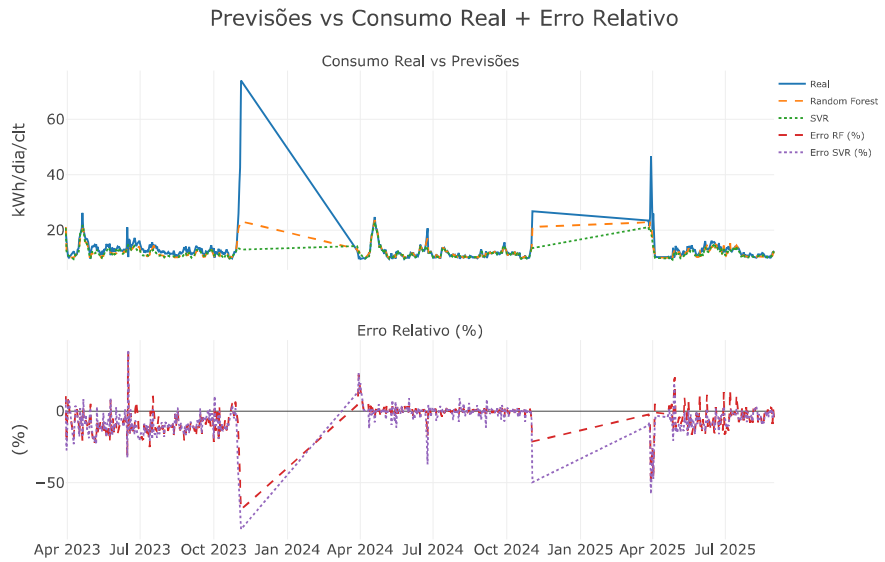



```
df_pred.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change',
       'previsão_svr', 'previsão_rf', 'previsão_tree'],
      dtype='object')
```

```
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import pandas as pd

# Calcular erro relativo (%)
df_pred['erro_relativo_rf'] = ((df_pred['previsão_rf'] - df_pred['ELET_CONS_CLT']) / df_pred['ELET_CONS_CLT']) * 100
df_pred['erro_relativo_svr'] = ((df_pred['previsão_svr'] - df_pred['ELET_CONS_CLT']) / df_pred['ELET_CONS_CLT']) * 100

# Selecionar os dados
df_plot = df_pred[['ELET_CONS_CLT', 'previsão_rf', 'previsão_svr', 'erro_relativo_rf', 'erro_relativo_svr']].dropna().copy()

# Resetar índice e renomear para 'Data'
df_plot = df_plot.reset_index().rename(columns={df_plot.index.name or 'index': 'Data'})

# Criar subplots
fig = make_subplots(rows=2, cols=1, shared_xaxes=True, vertical_spacing=0.15,
                    subplot_titles=("Consumo Real vs Previsões", "Erro Relativo (%)"))

# Subplot 1: Consumo real e previsões
fig.add_trace(go.Scatter(x=df_plot['Data'], y=df_plot['ELET_CONS_CLT'],
                         mode='lines', name='Real', line=dict( dash='solid')), row=1, col=1)

fig.add_trace(go.Scatter(x=df_plot['Data'], y=df_plot['previsão_rf'],
                         mode='lines', name='Random Forest', line=dict( dash='dash')), row=1, col=1)
```

```
fig.add_trace(go.Scatter(x=df_plot['Data'], y=df_plot['previsão_svr'],
                         mode='lines', name='SVR', line=dict( dash='dot')), row=1, col=1)

# Subplot 2: Erro relativo
fig.add_trace(go.Scatter(x=df_plot['Data'], y=df_plot['erro_relativo_rf'],
                         mode='lines', name='Erro RF (%)', line=dict( dash='dash')), row=2, col=1)

fig.add_trace(go.Scatter(x=df_plot['Data'], y=df_plot['erro_relativo_svr'],
                         mode='lines', name='Erro SVR (%)', line=dict( dash='dot')), row=2, col=1)

# Layout final
fig.update_layout(height=700, width=1000,
                  title_text="Previsões vs Consumo Real + Erro Relativo",
                  yaxis=dict(title='kWh/dia/clt'),
                  yaxis2=dict(title='(%)'),
                  hovermode='x unified')

fig.show()
```



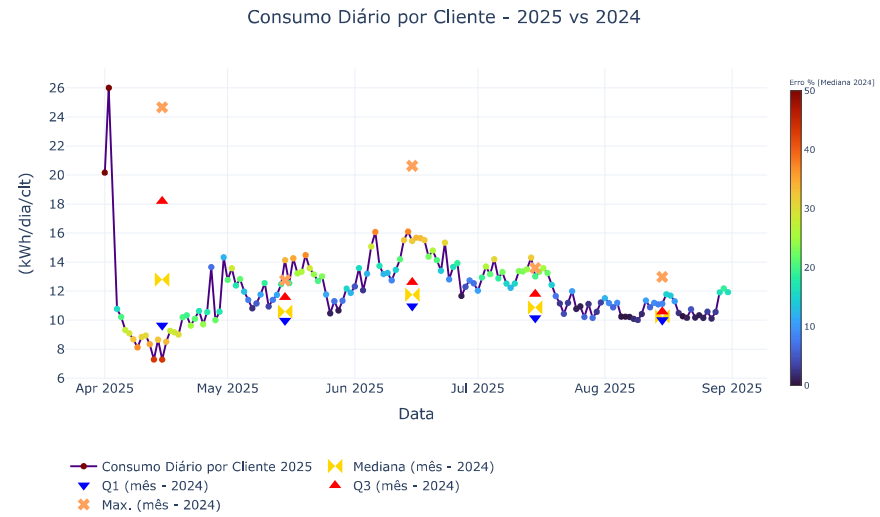

```
dados_2025
```

DATA	ELET_CONS_CLT	mes	med2024	erro_relativo	previsão_rf	previsão_svr	erro_relativo_rf	erro_relativo_svr	anomalia_negativa
2025-04-01	20.167279	4	12.791762	57.658333	14.855425	14.965814	-26.338976	-25.791605	True
2025-04-02	26.015000	4	12.791762	103.373071	13.895164	13.729358	-46.587875	-47.225225	True
2025-04-04	10.777647	4	12.791762	15.745409	11.175130	9.979468	3.688033	-7.405878	False
2025-04-05	10.224138	4	12.791762	20.072484	10.216189	9.898363	-0.077746	-3.186332	False
2025-04-06	9.313346	4	12.791762	27.192627	NaN	NaN	NaN	NaN	False
...	...	...	...	...	...	...	...	...	...
2025-08-27	10.112764	8	10.264049	1.473931	10.280448	10.275543	1.658136	1.609637	False
2025-08-28	10.565074	8	10.264049	2.932808	10.830714	10.598425	2.514322	0.315668	False
2025-08-29	11.912844	8	10.264049	16.063785	10.639877	11.385425	-10.685673	-4.427316	True
2025-08-30	12.184495	8	10.264049	18.710406	12.325657	12.174222	1.158541	-0.084308	False
2025-08-31	11.930569	8	10.264049	16.236474	11.410585	11.372169	-4.358417	-4.680411	False

152 rows × 9 columns

```
dados_2025['mes'] = dados_2025.index.month

# Mapear a mediana de 2024 para cada linha de 2025 com base no mês
dados_2025['med2024'] = dados_2025['mes'].map(resumo_mensal['50%'])
dados_2025['erro_relativo'] = (abs(dados_2025['ELET_CONS_CLT'] - dados_2025['med2024']) / dados_2025['med2024'])*100

# Determinar cor dos marcadores
#cores = ['red' if erro > +0.60 else 'yellow' if erro > +0.30 else 'blue' for erro in dados_2025['erro_relativo']]


# Criar gráfico de linha com dados diários
fig = go.Figure()

fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['ELET_CONS_CLT'],
    mode='lines+markers',
    marker=dict(
    color=dados_2025['erro_relativo'],  # valores entre 0 e 1
    colorscale='Turbo',                  # ou 'Viridis', 'Plasma', etc.
    cmin=0,
    cmax=50,
    colorbar=dict(title='Erro % [Mediana 2024]', titlefont=dict(size=8),thickness=12, tickfont=dict(size=10)),
    size=7,),
    name='Consumo Diário por Cliente 2025',
    line=dict(color='indigo')
))

# Adicionar pontos estatísticos
fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['50%'],
    mode='markers',
    name='Mediana (mês - 2024)',
    marker=dict(color='gold', symbol='bowtie', size=16)
))

fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['25%'],
    mode='markers',
    name='Q1 (mês - 2024)',
    marker=dict(color='blue', symbol='triangle-down', size=12)
))
```

```
fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['75%'],
    mode='markers',
    name='Q3 (mês - 2024)',
    marker=dict(color='red',symbol='triangle-up', size=12)
))
fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['max'],
    mode='markers',
    name='Max. (mês - 2024)',
    marker=dict(symbol='x', size=12)
))
# Layout final
fig.update_layout(

    title='Consumo Diário por Cliente - 2025 vs 2024',
    xaxis_title='Data',
    yaxis_title='(kWh/dia/clt)',
   template='plotly_white',
    height=600,
    width=1000,
    font=dict(size=14),title_x=0.5,
    legend=dict(orientation='h', y=-0.2)
)
fig.update_yaxes(
    dtick=2,  # intervalo de 2 unidades
    title_text='(kWh/dia/clt)'
)

fig.show()
```




```
dados_2025.columns
```

```
# Certifique-se de que ambos os índices estão no tipo datetime
dados_2025.index = pd.to_datetime(dados_2025.index)
df_pred.index = pd.to_datetime(df_pred.index)

# Selecionar apenas as colunas desejadas de df_pred
df_pred_selecionado = df_pred[['previsão_rf', 'previsão_svr', 'erro_relativo_rf', 'erro_relativo_svr']]

# Realizar o merge com base no índice
dados_2025 = dados_2025.merge(df_pred_selecionado, left_index=True, right_index=True, how='left')
dados_2025
```

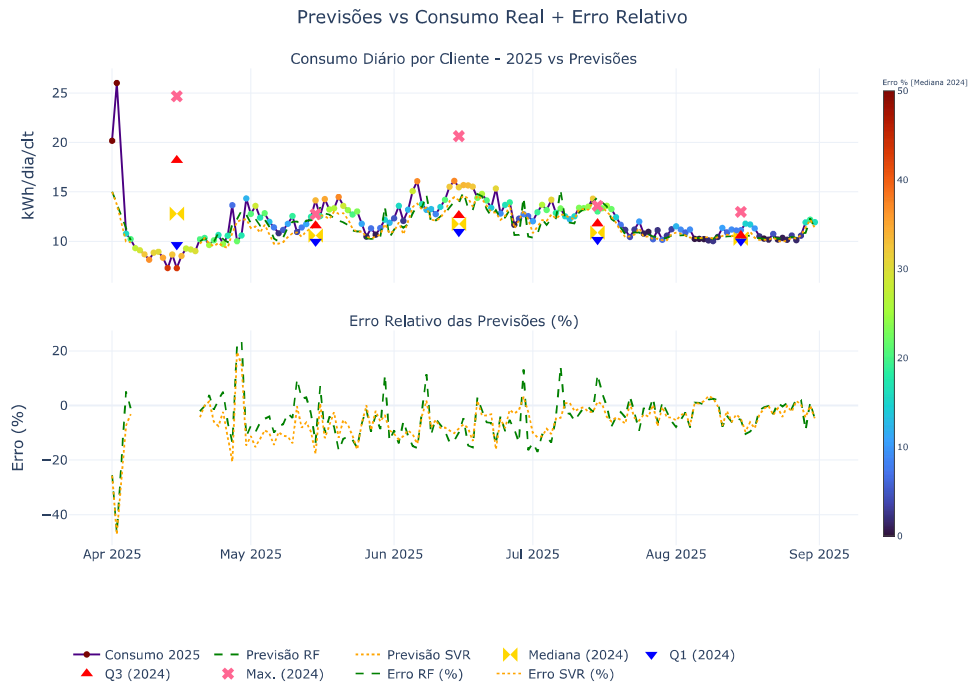
DATA	ELET_CONS_CLT	mes	med2024	erro_relativo	previsão_rf	previsão_svr	erro_relativo_rf	erro_relativo_svr
2025-04-01	20.167279	4	12.791762	57.658333	15.017605	14.971816	25.534801	25.761843
2025-04-02	26.015000	4	12.791762	103.373071	13.820930	13.731936	46.873228	47.215314
2025-04-04	10.777647	4	12.791762	15.745409	11.327024	9.976873	5.097373	7.429948
2025-04-05	10.224138	4	12.791762	20.072484	10.120539	9.896231	1.013282	3.207189
2025-04-06	9.313346	4	12.791762	27.192627	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...
2025-08-27	10.112764	8	10.264049	1.473931	10.359583	10.276879	2.440664	1.622845
2025-08-28	10.565074	8	10.264049	2.932808	10.780303	10.600844	2.037177	0.338569
2025-08-29	11.912844	8	10.264049	16.063785	10.840260	11.387050	9.003590	4.413674
2025-08-30	12.184495	8	10.264049	18.710406	12.321556	12.172142	1.124885	0.101380
2025-08-31	11.930569	8	10.264049	16.236474	11.334702	11.369584	4.994458	4.702080

152 rows × 8 columns

```
dados_2025.columns
```

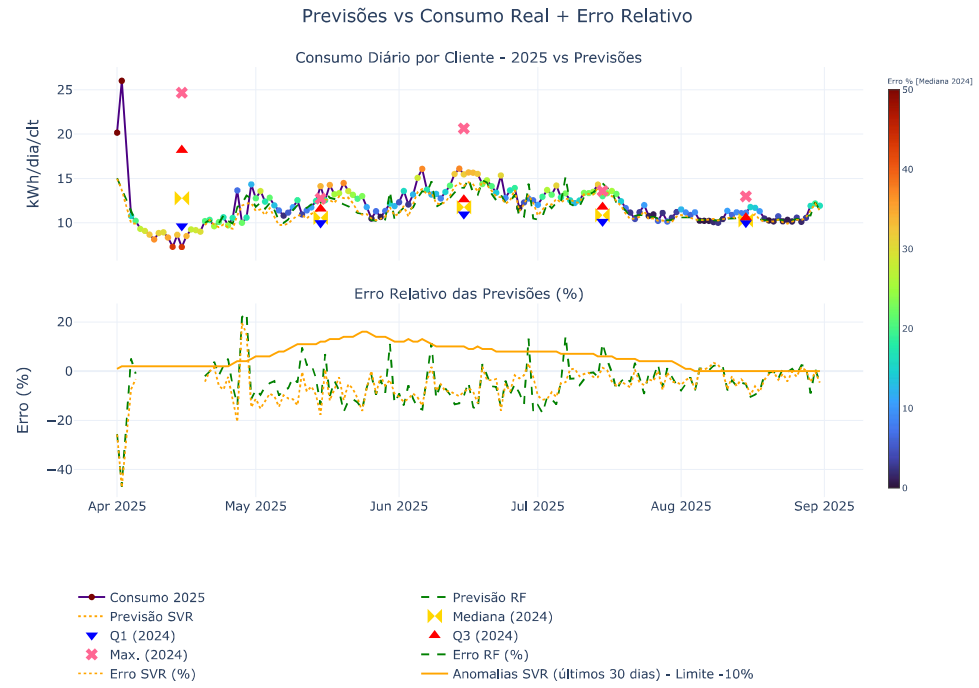
```
from plotly.subplots import make_subplots
import plotly.graph_objects as go

# Criar subplots: 2 linhas, 1 coluna
fig = make_subplots(
    rows=2, cols=1,
    shared_xaxes=True,
    vertical_spacing=0.1,
    subplot_titles=("Consumo Diário por Cliente - 2025 vs Previsões", "Erro Relativo das Previsões (%)")
)

# ——————————————————————————————
# Subplot 1: Consumo real + previsões
fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['ELET_CONS_CLT'],
    mode='lines+markers',
    name='Consumo 2025',
    line=dict(color='indigo'),
    marker=dict(
        color=dados_2025['erro_relativo'],
        colorscale='Turbo',
        cmin=0,
        cmax=50,
```

```
        colorbar=dict(title='Erro % [Mediana 2024]', titlefont=dict(size=8), thickness=12, tickfont=dict(size=10)),
        size=7
    )
), row=1, col=1)

fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['previsão_rf'],
    mode='lines',
    name='Previsão RF',
    line=dict(dash='dash', color='green')
), row=1, col=1)

fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['previsão_svr'],
    mode='lines',
    name='Previsão SVR',
    line=dict(dash='dot', color='orange')
), row=1, col=1)

# Estatísticas mensais de 2024
fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['50%'],
    mode='markers',
    name='Mediana (2024)',
    marker=dict(color='gold', symbol='bowtie', size=16)
), row=1, col=1)

fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['25%'],
    mode='markers',
    name='Q1 (2024)',
    marker=dict(color='blue', symbol='triangle-down', size=12)
), row=1, col=1)

fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['75%'],
    mode='markers',
    name='Q3 (2024)',
    marker=dict(color='red', symbol='triangle-up', size=12)
), row=1, col=1)

fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['max'],
    mode='markers',
    name='Max. (2024)',
    marker=dict(symbol='x', size=12)
), row=1, col=1)

# ——————————————————————————————————————
# Subplot 2: Erro relativo
fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['erro_relativo_rf'],
    mode='lines',
    name='Erro RF (%)',
    line=dict(dash='dash', color='green')
), row=2, col=1)

fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['erro_relativo_svr'],
    mode='lines',
    name='Erro SVR (%)',
    line=dict(dash='dot', color='orange')
), row=2, col=1)

# ——————————————————————————————————————
# Layout final
fig.update_layout(
    height=800,
    width=1100,
    title_text="Previsões vs Consumo Real + Erro Relativo",
    template='plotly_white',
    font=dict(size=14),
    title_x=0.5,
    legend=dict(orientation='h', y=-0.2),
    hovermode='x unified'
)

fig.update_yaxes(title_text='kWh/dia/clt', row=1, col=1)
fig.update_yaxes(title_text='Erro (%)', row=2, col=1)

fig.show()
```



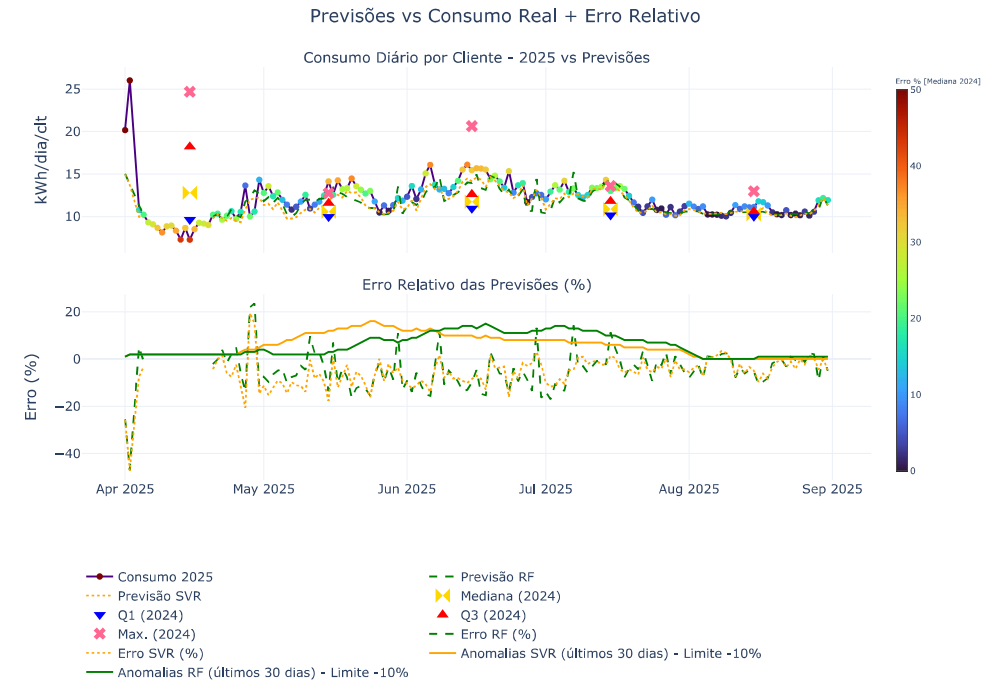

```
import pandas as pd

# Definir o limite negativo de erro relativo (%)
limite_negativo = -10  # pode alterar para -15, etc.

# Selecionar a coluna de erro relativo do modelo a avaliar
coluna_erro = 'erro_relativo_svr'  # ou 'erro_relativo_svr', 'erro_relativo_xgb_f'

# Criar coluna booleana: True apenas quando o erro relativo é menor que o limite negativo
dados_2025['anomalia_negativa'] = dados_2025[coluna_erro] < limite_negativo

# Contar anomalias por janela móvel de 30 dias
anomalias_neg_30d = (
    dados_2025['anomalia_negativa']
    .rolling(window=30, min_periods=1)
    .sum()
)

# Número total de anomalias negativas no período completo
total_anomalias_neg = dados_2025['anomalia_negativa'].sum()

print(f"Limite de erro relativo negativo: {limite_negativo}%")
print(f"Número total de anomalias negativas detetadas: {total_anomalias_neg}")
print("\nAnomalias negativas detetadas em cada janela de 30 dias:")
print(anomalias_neg_30d)

# (Opcional) Adicionar ao gráfico no subplot 2
fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=anomalias_neg_30d,
    mode='lines',
    name=f'Anomalias SVR (últimos 30 dias) - Limite {limite_negativo}%',
    line=dict(color='orange', width=2)
), row=2, col=1)

fig.show()
```

```
Limite de erro relativo negativo: -10%
Número total de anomalias negativas detetadas: 30

Anomalias negativas detetadas em cada janela de 30 dias:
DATA
2025-04-01    1.0
2025-04-02    2.0
2025-04-04    2.0
2025-04-05    2.0
2025-04-06    2.0
              ...
2025-08-27    0.0
2025-08-28    0.0
2025-08-29    0.0
2025-08-30    0.0
2025-08-31    0.0
Name: anomalia_negativa, Length: 152, dtype: float64
```




```
import pandas as pd

# Definir o limite negativo de erro relativo (%)
limite_negativo = -10  # pode alterar para -15, etc.

# Selecionar a coluna de erro relativo do modelo a avaliar
coluna_erro = 'erro_relativo_rf'  # ou 'erro_relativo_svr', 'erro_relativo_xgb_f'

# Criar coluna booleana: True apenas quando o erro relativo é menor que o limite negativo
dados_2025['anomalia_negativa'] = dados_2025[coluna_erro] < limite_negativo

# Contar anomalias por janela móvel de 30 dias
anomalias_neg_30d = (
    dados_2025['anomalia_negativa']
    .rolling(window=30, min_periods=1)
    .sum()
)

# Número total de anomalias negativas no período completo
total_anomalias_neg = dados_2025['anomalia_negativa'].sum()

print(f"Limite de erro relativo negativo: {limite_negativo}%")
print(f"Número total de anomalias negativas detetadas: {total_anomalias_neg}")
print("\nAnomalias negativas detetadas em cada janela de 30 dias:")
print(anomalias_neg_30d)

# (Opcional) Adicionar ao gráfico no subplot 2
fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=anomalias_neg_30d,
    mode='lines',
    name=f'Anomalias RF (últimos 30 dias) - Limite {limite_negativo}%',
    line=dict(color='green', width=2)
), row=2, col=1)

fig.show()
```

```
Limite de erro relativo negativo: -10%
Número total de anomalias negativas detetadas: 30

Anomalias negativas detetadas em cada janela de 30 dias:
DATA
2025-04-01    1.0
2025-04-02    2.0
2025-04-04    2.0
2025-04-05    2.0
2025-04-06    2.0
              ...
2025-08-27    1.0
2025-08-28    1.0
2025-08-29    1.0
2025-08-30    1.0
2025-08-31    1.0
Name: anomalia_negativa, Length: 152, dtype: float64
```



EXPORTAR

↳ 4 células ocultas

XGB

```
df1_filtrado3
```

```
treino1 = df1_filtrado3.loc[(df1_filtrado3.index > '2023-04-01') & (df1_filtrado3.index <= '2024-10-31')].reset_index()
teste1 = df1_filtrado3.loc[df1_filtrado3.index >= '2025-04-05'].reset_index()


treino1['Tipo'] = 'Treino1'
teste1['Tipo'] = 'Teste1'
df_plot = pd.concat([treino1, teste1])


fig = px.line(df_plot, x='DATA', y='ELET_CONS_CLT', color='Tipo',
              title='Dados divididos - Treino vs Teste | IF',
              labels={'DATA': 'Data', 'ELET_CONS_CLT': 'Consumo (kWh/dia)', 'Tipo': 'Conjunto'})

#  a linha vertical para marcar a separação
fig.add_vline(x='2025-03-21', line_dash="dash", line_color="black")


fig.update_layout(width=1500, height=500)
# Exibir o gráfico
fig.show()
fig.show('notebook_connected')
```

1ª itereção

```
##treino1 = df1_filtrado3.loc[(df1_filtrado3.index > '2023-04-01') & (df1_filtrado3.index <= '2024-10-31')].reset_index()
#teste1 = df1_filtrado3.loc[df1_filtrado3.index >= '2025-04-05'].reset_index()

treino1 = df1_filtrado3.loc[(df1_filtrado3.index > '2023-04-01') & (df1_filtrado3.index <= '2024-10-31')].copy()
teste1 = df1_filtrado3.loc[df1_filtrado3.index >= '2025-04-05'].copy()




treino1 = create_features(treino1)
teste1 = create_features(teste1)

# em função do nivel de correlação, foram escolhidos os inputs
input = ['T_mean', 'T_max', 'T_min', 'RH_mean', 'RH_max', 'RH_min', 'Calor_idx',

         'Q_OCP', 'OCP', 'AD', 'CR', 'CLTs',
#'delta_ELET_CONS_CLT', 'delta_CLTs',
         'dia_semana', 'trimestre', 'mes',
        'dia_ano', 'dia_mes', 'semana_ano']

output = 'ELET_CONS_CLT'

x_treino1 = treino1[input]
y_treino1 = treino1[output]

x_teste1= teste1[input]
y_teste1= teste1[output]
```

Treinar o modelo XGB

XGBoost (Extreme Gradient Boosting)

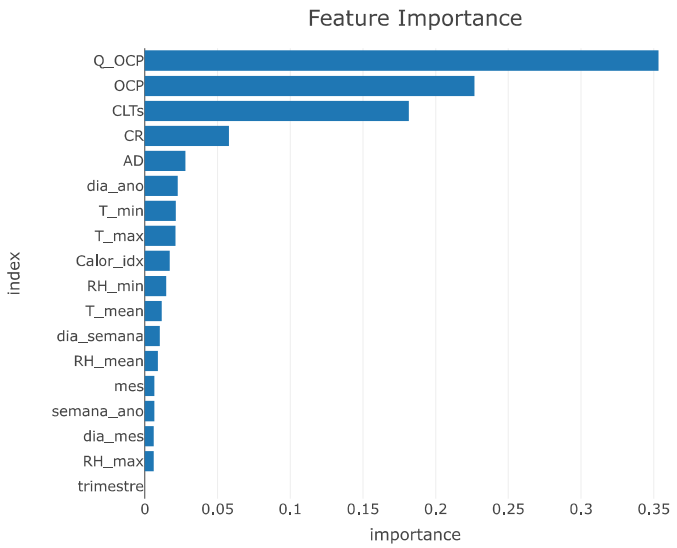
O XGBoost é um modelo baseado em árvores de decisão que utiliza um algoritmo de boosting para melhorar a previsão. Ele é especialmente eficaz para dados tabulares e séries temporais.Em vez de treinar uma única árvore de decisão, o XGBoost cria várias árvores sequencialmente, onde cada nova árvore corrige os erros da anterior.

```
import xgboost as xgb

reg1 = xgb.XGBRegressor(base_score=0.01, booster='gbtree',
                        n_estimators=100000,
                       early_stopping_rounds=8000,
                       objective='reg:squarederror',
                         reg_lambda=5,
                         reg_alpha=0.5,
                       max_depth=1,
                         min_child_weight=3,
                         subsample=0.95,
                         colsample_bytree=0.85,
                         gamma=0.05,

                       learning_rate=0.05)

reg1.fit(x_treino1, y_treino1,
        eval_set=[(x_treino1, y_treino1), (x_teste1, y_teste1)],
        verbose=1000)
```

```
[0] validation_0-rmse:12.95805   validation_1-rmse:11.81549
[1000]  validation_0-rmse:1.27780   validation_1-rmse:0.86669
[2000]  validation_0-rmse:1.04176   validation_1-rmse:0.89012
[3000]  validation_0-rmse:0.90026   validation_1-rmse:0.92147
[4000]  validation_0-rmse:0.80144   validation_1-rmse:0.95921
[5000]  validation_0-rmse:0.72835   validation_1-rmse:0.99935
[6000]  validation_0-rmse:0.66996   validation_1-rmse:1.03537
[7000]  validation_0-rmse:0.62274   validation_1-rmse:1.06158
[8000]  validation_0-rmse:0.58320   validation_1-rmse:1.08691
[9000]  validation_0-rmse:0.54885   validation_1-rmse:1.10811
[9112]  validation_0-rmse:0.54515   validation_1-rmse:1.11147
```

```
XGBRegressor(base_score=0.01, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.85, device=None, early_stopping_rounds=8000,
             enable_categorical=False, eval_metric=None, feature_types=None,
             feature_weights=None, gamma=0.05, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=0.05, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=1,
             max_leaves=None, min_child_weight=3, missing=nan,
             monotone_constraints=None, multi_strategy=None,
             n_estimators=100000, n_jobs=None, num_parallel_tree=None, ...)
```

```
fi = pd.DataFrame(data=reg1.feature_importances_,
                  index=reg1.feature_names_in_,
                  columns=['importance'])

fi = fi.sort_values('importance')

# o gráfico de barras horizontal interativo
fig = px.bar(fi, x='importance', y=fi.index, orientation='h',
             title='Feature Importance')
fig.update_layout(
    yaxis_title_standoff=30  # Aumente o valor para afastar mais
)
fig.update_layout(
    margin=dict(l=200, r=200, t=50, b=50)  # Aumenta a margem direita (r) para mover o gráfico
)
fig.show()
fig.show('notebook_connected')
```



Resultados

```
df_pred.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change'],
      dtype='object')
```

```
teste1['previsão_xgb1'] = reg1.predict(x_teste1)
df_pred = df_pred.merge(teste1[['previsão_xgb1']], how='left', left_index=True, right_index=True)
```

```
df_pred.tail()
```

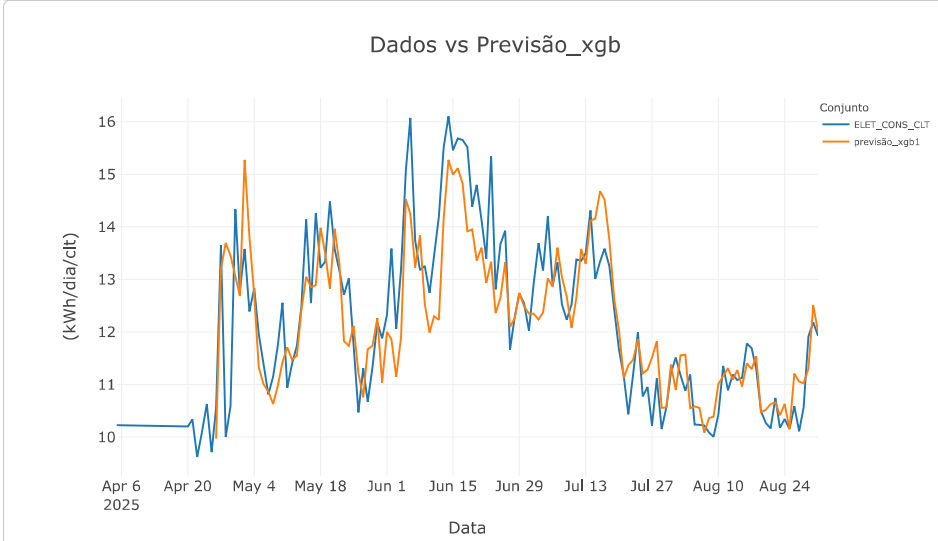
DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	Total_CO2e	CO2e_QOCP	ELET_CONS_CLT_lag1	ELET_CONS_CLT_lag7	ELET_CONS_CLT_rolling_mean	ELET_CONS_CLT_rolling_median	E
2025-08-27	23.41699	31.9	18.3	13.6	75.88219	89.0	51.5	37.5	23.98021	12914.0	...	4459.54288	10.021445	10.585227	10.270170	10.325795	10.176106	
2025-08-28	22.23625	30.2	18.3	11.9	80.14818	100.0	54.7	45.3	22.37211	12826.0	...	2230.10200	5.259675	10.112764	10.165070	10.382938	10.341620	
2025-08-29	22.64507	34.0	17.5	16.5	79.59112	91.5	54.4	37.1	22.82476	12985.0	...	4820.41736	12.553170	10.565074	10.747419	10.549428	10.341620	
2025-08-30	24.65698	34.5	19.7	14.8	78.81356	92.0	56.4	35.6	25.36257	12416.0	...	4722.49436	12.973886	11.912844	10.176106	10.836340	10.565074	
2025-08-31	22.51888	30.1	18.7	11.4	83.64605	98.5	59.9	38.6	22.13280	12372.0	...	3978.83140	10.254720	12.184495	10.341620	11.063333	10.585227	

5 rows × 49 columns

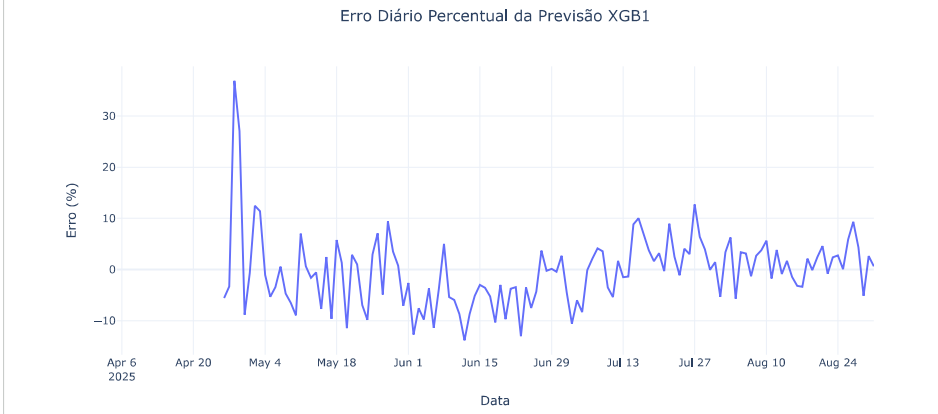
```
# =========================
# Chamar a função para cada modelo
# =========================
avaliar_modelo(reg1, x_treino1, y_treino1, x_teste1, y_teste1, 'XGB', 1)

# Criar DataFrame com todos os resultados
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nTabela de Avaliação:")
print(df_resultados)
```

```
XGB (Iteração 1) - TREINO → MSE: 1.5425 | RMSE: 1.24 | MAE: 0.7065 | R²: 0.92
XGB (Iteração 1) - TESTE  → MSE: 0.7387 | RMSE: 0.86 | MAE: 0.6388 | R²: 0.69

Tabela de Avaliação:
             Modelo  Iteração Conjunto     MSE  RMSE     MAE    R²
0   Árvore de Decisão         1   Treino  4.3337  2.08  0.7089  0.81
1   Árvore de Decisão         1    Teste  1.4757  1.21  0.9388  0.41
2       Random Forest         1   Treino  0.9935  1.00  0.3682  0.96
3       Random Forest         1    Teste  0.8964  0.95  0.7417  0.64
4                 SVR         1   Treino  5.3172  2.31  0.5493  0.76
5                 SVR         1    Teste  0.7506  0.87  0.6945  0.70
6   Árvore de Decisão         2   Treino  3.7421  1.93  0.4133  0.83
7   Árvore de Decisão         2    Teste  0.3438  0.59  0.3803  0.86
8       Random Forest         2   Treino  0.7137  0.84  0.1967  0.97
9       Random Forest         2    Teste  0.0818  0.29  0.1801  0.97
10                SVR         2   Treino  6.1005  2.47  0.2629  0.72
11                SVR         2    Teste  0.0836  0.29  0.2376  0.97
12  Árvore de Decisão         3   Treino  0.3084  0.56  0.2801  0.95
13  Árvore de Decisão         3    Teste  1.0447  1.02  0.7326  0.56
14      Random Forest         3   Treino  0.0769  0.28  0.1294  0.99
15      Random Forest         3    Teste  0.6032  0.78  0.5853  0.75
16                SVR         3   Treino  0.0155  0.12  0.0900  1.00
17                SVR         3    Teste  0.1855  0.43  0.3345  0.92
18  Árvore de Decisão         4   Treino  0.5531  0.74  0.4124  0.91
19  Árvore de Decisão         4    Teste  1.1623  1.08  0.8901  0.51
20      Random Forest         4   Treino  0.1244  0.35  0.1925  0.98
21      Random Forest         4    Teste  1.1921  1.09  0.8701  0.50
22                SVR         4   Treino  0.4586  0.68  0.2897  0.92
23                SVR         4    Teste  0.9126  0.96  0.7488  0.62
24                XGB         1   Treino  1.5425  1.24  0.7065  0.92
25                XGB         1    Teste  0.7387  0.86  0.6388  0.69
```

```
import pandas as pd

df_plot = df_pred.loc[df_pred.index >= pd.to_datetime('2025-04-05')].reset_index()


# Criando um DataFrame para diferenciar os conjuntos
df_plot = df_plot.melt(id_vars=['DATA'], value_vars=['ELET_CONS_CLT', 'previsão_xgb1'], var_name='Tipo', value_name='Valor')

# Criando o gráfico interativo
fig = px.line(df_plot, x='DATA', y='Valor', color='Tipo',
              title='Dados vs Previsão_xgb',
              labels={'DATA': 'Data', 'Valor': '(kWh/dia/clt)', 'Tipo': 'Conjunto'})

fig.show()
#fig.show('notebook_connected')
```




```
df_erro = df_pred.loc[df_pred.index >= pd.to_datetime('2025-04-05')].copy()

# Calcular erro percentual diretamente no DataFrame erro
df_erro['Erro [%]'] = ((df_erro['previsão_xgb1'] - df_erro['ELET_CONS_CLT']) / df_erro['ELET_CONS_CLT']) * 100

# Resetar índice para usar 'DATA' como coluna
df_plot = df_erro.reset_index()

# Criar gráfico interativo
fig = px.line(df_plot, x='DATA', y='Erro [%]',
              title='Erro Diário Percentual da Previsão XGB1',
              labels={'DATA': 'Data', 'Erro [%]': 'Erro (%)'},
              template='plotly_white')

fig.update_layout(width=1000, height=500, title_x=0.5)
fig.show()
```

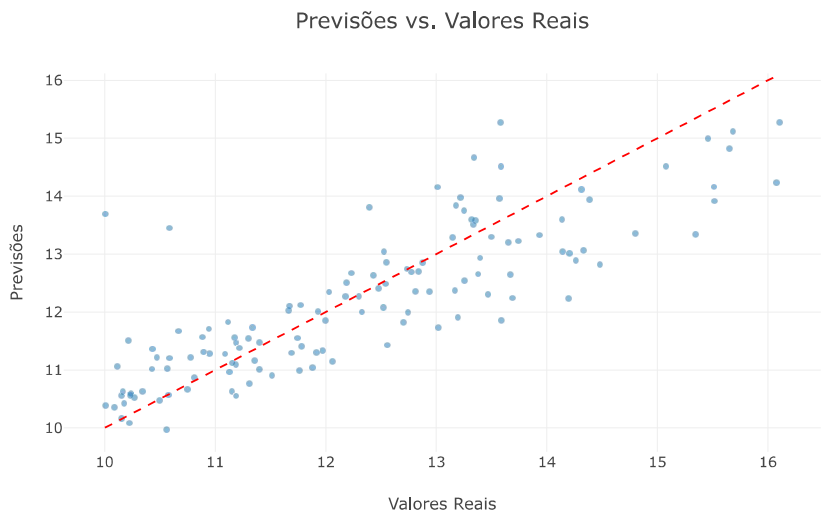


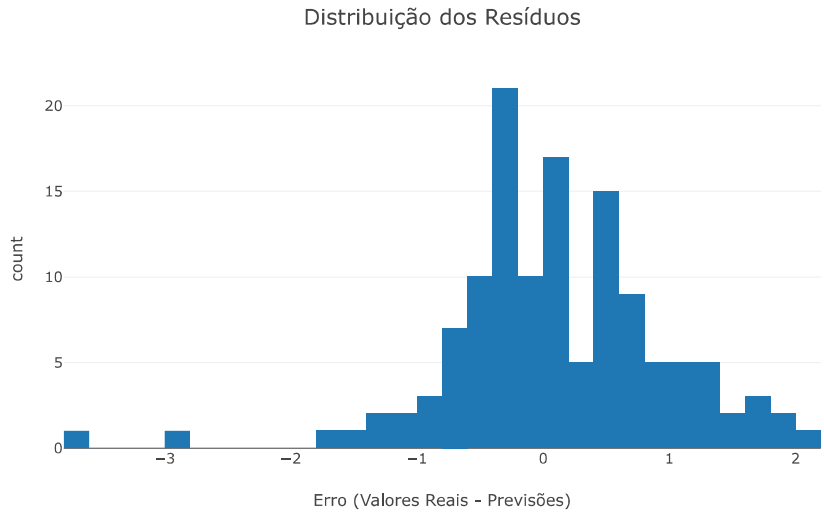

```
df1 = df1.merge(teste1['previsão_xgb1'], left_index=True, right_index=True, how='left')
#df1.rename(columns={'Previsto': 'Previsões GRU2'}, inplace=True)
df1.tail()
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	semana_ano	feriado	AGUA_CONS_CLT	GAS_CONS_CLT	CO2e_ELET	CO2e_AGUA	CO2e_GAS	Total_CO2e	CO2e_QOCP	previsão_xgb1
2025-08-27	23.41699	31.9	18.3	13.6	75.88219	89.0	51.5	37.5	23.98021	12914.0	...	35	False	0.281911	0.528103	2156.638	104.40	2198.50488	4459.54288	10.021445	11.054025
2025-08-28	22.23625	30.2	18.3	11.9	80.14818	100.0	54.7	45.3	22.37211	12826.0	...	35	False	0.250412	0.000000	2141.942	88.16	0.00000	2230.10200	5.259675	11.024541
2025-08-29	22.64507	34.0	17.5	16.5	79.59112	91.5	54.4	37.1	22.82476	12985.0	...	35	False	0.275229	0.721822	2168.495	87.00	2564.92236	4820.41736	12.553170	11.302096
2025-08-30	24.65698	34.5	19.7	14.8	78.81356	92.0	56.4	35.6	25.36257	12416.0	...	35	False	0.284593	0.772116	2073.472	84.10	2564.92236	4722.49436	12.973886	12.505833
2025-08-31	22.51888	30.1	18.7	11.4	83.64605	98.5	59.9	38.6	22.13280	12372.0	...	35	False	0.268081	0.541938	2066.124	80.62	1832.08740	3978.83140	10.254720	12.004293

5 rows × 42 columns

```
df_plot1 = teste1[['ELET_CONS_CLT', 'previsão_xgb1']].reset_index()

#  o gráfico de dispersão interativo
fig = px.scatter(df_plot1, x='ELET_CONS_CLT', y='previsão_xgb1',
                 title="Previsões vs. Valores Reais",
                 labels={'ELET_CONS_CLT': 'Valores Reais', 'previsão_xgb1': 'Previsões'},
                 opacity=0.5)

# Adiciona a linha de referência
fig.add_shape(type="line", x0=df_plot1['ELET_CONS_CLT'].min(), x1=df_plot1['ELET_CONS_CLT'].max(),
              y0=df_plot1['ELET_CONS_CLT'].min(), y1=df_plot1['ELET_CONS_CLT'].max(),
              line=dict(color="red", dash="dash"))

fig.show()
fig.show('notebook_connected')
# Cria o gráfico de resíduos
df_plot1['Resíduos'] = df_plot1['ELET_CONS_CLT'] - df_plot1['previsão_xgb1']
fig_residuos = px.histogram(df_plot1, x='Resíduos', nbins=50,
                            title="Distribuição dos Resíduos",
                            labels={'Resíduos': 'Erro (Valores Reais - Previsões)'})

fig_residuos.show()
fig_residuos.show('notebook_connected')
```


Previsões vs. Valores Reais
Previsões
Valores Reais
16
15
14
13
12
11
10
10
11
12
13
14
15
16


Distribuição dos Resíduos
count
Erro (Valores Reais - Previsões)
20
15
10
5
0
−3
−2
−1
0
1
2

2ª iteração

```
df1_filtrado3.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change'],
      dtype='object')
```

```
treino2 = df1_filtrado3.loc[
    ((df1_filtrado3.index > '2023-04-01') & (df1_filtrado3.index <= '2023-10-31')) |
    ((df1_filtrado3.index > '2024-04-01') & (df1_filtrado3.index <= '2024-10-31'))
]

# Seleção do conjunto de teste
teste2 = df1_filtrado3.loc[df1_filtrado3.index >= '2025-04-05']

# Aplicar função de engenharia de features
treino2 = create_features(treino2)
teste2 = create_features(teste2)

# Definir variáveis de entrada e saída
input = ['ELET_CONS_CLT_lag1', 'ELET_CONS_CLT_lag7',
       'ELET_CONS_CLT_rolling_mean', 'ELET_CONS_CLT_rolling_median',
       'ELET_CONS_CLT_rolling_std', 'ELET_CONS_CLT_error_median',
       'ELET_CONS_CLT_pct_change']
output = 'ELET_CONS_CLT'

x_treino2 = treino2[input]
y_treino2 = treino2[output]

x_teste2 = teste2[input]
y_teste2 = teste2[output]
```

Mostrar saída oculta

```
treino2
```

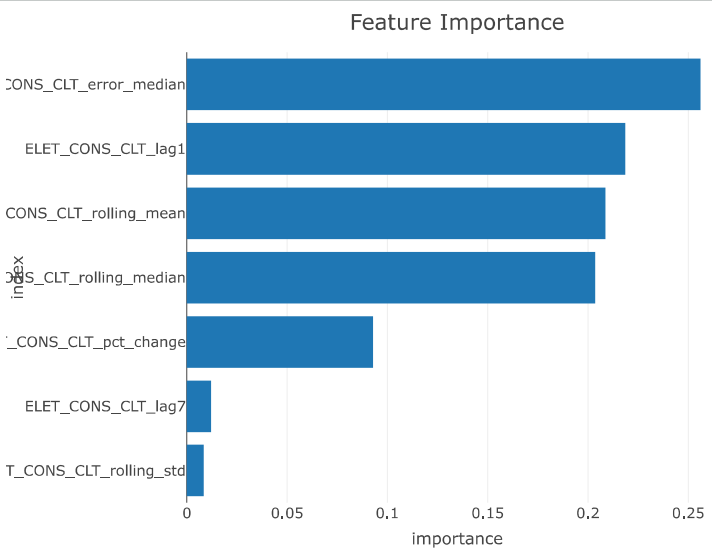
DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	CO2e_GAS	Total_CO2e	CO2e_QOCP	ELET_CONS_CLT_lag1	ELET_CONS_CLT_lag7	ELET_CONS_CLT_rolling_mean	ELE
2023-04-16	23.260000	28.6	19.4	9.200001	54.360001	64.400002	37.599998	26.800003	25.058527	10108.0	...	1832.08740	3589.43340	16.096114	13.397222	10.514851	12.464950	
2023-04-17	20.400000	25.9	15.1	10.799999	68.149999	84.699997	54.000000	30.699997	23.453397	9514.0	...	1832.08740	3467.90540	17.167849	16.679868	9.970117	13.629491	
2023-04-18	24.000000	27.5	20.5	7.000000	50.299999	57.599998	43.000000	14.599998	25.392125	9545.0	...	1832.08740	3493.38240	19.300455	18.121905	12.357595	14.728766	
2023-04-19	20.000000	26.1	15.7	10.400001	66.816668	82.000000	49.299999	32.700001	23.677519	9473.0	...	1099.25244	2723.58344	14.882970	20.052521	10.696767	16.168388	
2023-04-20	18.583333	24.4	14.2	10.200000	78.683334	87.599998	65.300003	22.299995	21.481411	10733.0	...	2198.50488	4043.40588	24.357867	20.774123	11.292824	18.303993	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2024-10-25	17.860000	20.5	15.6	4.900000	85.900002	100.000000	71.300003	28.699997	19.363441	10862.0	...	2564.92236	4444.70636	10.974584	10.826283	10.768186	10.366977	
2024-10-26	14.928572	21.6	11.0	10.600000	92.014285	99.800003	75.699997	24.100006	18.224307	11462.0	...	2748.13110	4742.32510	11.003074	9.956004	11.885312	10.067392	
2024-10-29	16.983333	21.1	14.7	6.400001	84.416667	95.300003	71.000000	24.300003	20.204971	11157.0	...	2931.33984	4860.38884	11.912718	9.788215	10.041516	10.071389	
2024-10-30	15.542857	19.5	13.2	6.300000	94.757143	100.000000	81.500000	18.500000	16.309118	11328.0	...	2564.92236	4513.82836	11.973020	10.069495	9.645635	10.289384	
2024-10-31	17.316667	20.1	13.9	6.200001	99.816667	100.000000	98.900002	1.099998	13.311978	10224.0	...	3114.54858	4880.82658	16.772600	11.171598	10.128298	10.710222	

390 rows × 48 columns

```
reg2 = xgb.XGBRegressor(base_score=0.01, booster='gbtree',
                        n_estimators=100000,
                       early_stopping_rounds=8000,
                       objective='reg:squarederror',
                         reg_lambda=5,
                         reg_alpha=0.5,
                       max_depth=1,
                         min_child_weight=3,
                         subsample=0.95,
                         colsample_bytree=0.85,
                         gamma=0.05,

                       learning_rate=0.05)

reg2.fit(x_treino2, y_treino2,
        eval_set=[(x_treino2, y_treino2), (x_teste2, y_teste2)],
        verbose=1000)
```

Mostrar saída oculta

```
fi = pd.DataFrame(data=reg2.feature_importances_,
                  index=reg2.feature_names_in_,
                  columns=['importance'])

fi = fi.sort_values('importance')

# o gráfico de barras horizontal interativo
fig = px.bar(fi, x='importance', y=fi.index, orientation='h',
             title='Feature Importance')
fig.update_layout(
    yaxis_title_standoff=30  # Aumente o valor para afastar mais
)
fig.update_layout(
    margin=dict(l=200, r=200, t=50, b=50)  # Aumenta a margem direita (r) para mover o gráfico
)
fig.show()
fig.show('notebook_connected')
```



Resultados

```
#df1_filtrado1 = df1_filtrado1.drop(columns=['previsão_xgb1_x', 'previsão_xgb1_y'])
```

```
teste2['previsão_xgb2'] = reg2.predict(x_teste2)
df_pred = df_pred.merge(teste2[['previsão_xgb2']], how='left', left_index=True, right_index=True)
```

Mostrar saída oculta

```
#df1 = df1.merge(teste2[['previsão_xgb2']], how='left', left_index=True, right_index=True)
```

```
df_pred.tail()
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	CO2e_QOCP	ELET_CONS_CLT_lag1	ELET_CONS_CLT_lag7	ELET_CONS_CLT_rolling_mean	ELET_CONS_CLT_rolling_median	ELET_CONS_CLT
2025-08-27	23.41699	31.9	18.3	13.6	75.88219	89.0	51.5	37.5	23.98021	12914.0	...	10.021445	10.585227	10.270170	10.325795	10.176106	
2025-08-28	22.23625	30.2	18.3	11.9	80.14818	100.0	54.7	45.3	22.37211	12826.0	...	5.259675	10.112764	10.165070	10.382938	10.341620	
2025-08-29	22.64507	34.0	17.5	16.5	79.59112	91.5	54.4	37.1	22.82476	12985.0	...	12.553170	10.565074	10.747419	10.549428	10.341620	
2025-08-30	24.65698	34.5	19.7	14.8	78.81356	92.0	56.4	35.6	25.36257	12416.0	...	12.973886	11.912844	10.176106	10.836340	10.565074	
2025-08-31	22.51888	30.1	18.7	11.4	83.64605	98.5	59.9	38.6	22.13280	12372.0	...	10.254720	12.184495	10.341620	11.063333	10.585227	

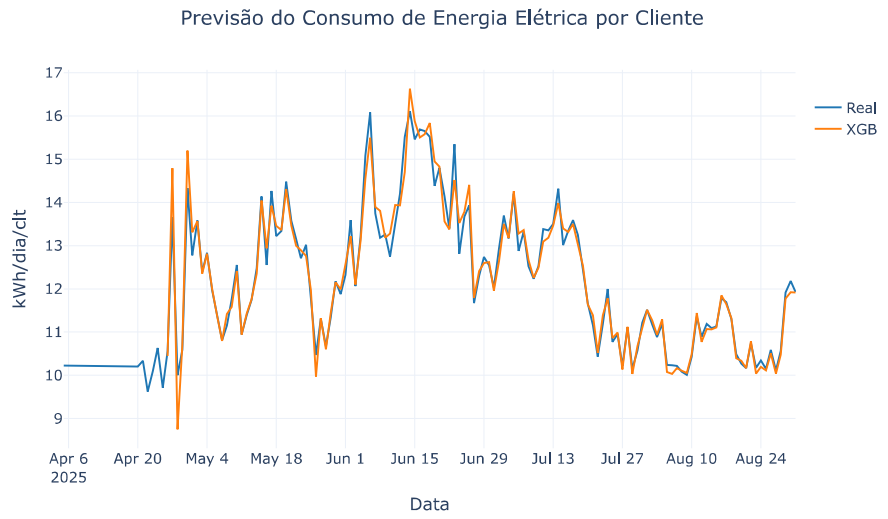
5 rows × 50 columns

```
# ========================
# Chamando a função para cada modelo
# ========================
avaliar_modelo(reg2, x_treino2, y_treino2, x_teste2, y_teste2, 'XGB', 2)

# Criar DataFrame com todos os resultados
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nTabela de Avaliação:")
print(df_resultados)
```

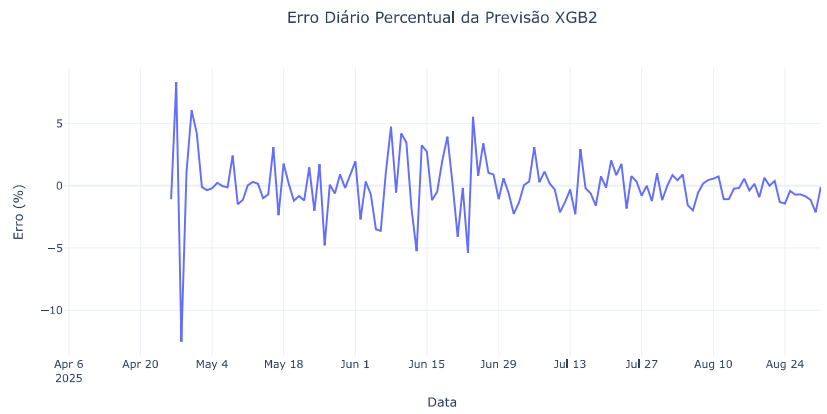
```
XGB (Iteração 2) - TREINO → MSE: 0.0244 | RMSE: 0.16 | MAE: 0.1082 | R²: 1.00
XGB (Iteração 2) - TESTE  → MSE: 0.0884 | RMSE: 0.30 | MAE: 0.1921 | R²: 0.96

Tabela de Avaliação:
              Modelo  Iteração Conjunto     MSE  RMSE     MAE    R²
0   Árvore de Decisão         1   Treino  4.3337  2.08  0.7089  0.81
1   Árvore de Decisão         1    Teste  1.4757  1.21  0.9388  0.41
2       Random Forest         1   Treino  0.9935  1.00  0.3682  0.96
3       Random Forest         1    Teste  0.8964  0.95  0.7417  0.64
4                 SVR         1   Treino  5.3172  2.31  0.5493  0.76
5                 SVR         1    Teste  0.7506  0.87  0.6945  0.70
6   Árvore de Decisão         2   Treino  3.7421  1.93  0.4133  0.83
7   Árvore de Decisão         2    Teste  0.3438  0.59  0.3803  0.86
8       Random Forest         2   Treino  0.7137  0.84  0.1967  0.97
9       Random Forest         2    Teste  0.0818  0.29  0.1801  0.97
10                SVR         2   Treino  6.1005  2.47  0.2629  0.72
11                SVR         2    Teste  0.0836  0.29  0.2376  0.97
12  Árvore de Decisão         3   Treino  0.3084  0.56  0.2801  0.95
13  Árvore de Decisão         3    Teste  1.0447  1.02  0.7326  0.56
```

14	Random Forest	3	Treino	0.0769	0.28	0.1294	0.99
15	Random Forest	3	Teste	0.6032	0.78	0.5853	0.75
16	SVR	3	Treino	0.0155	0.12	0.0900	1.00
17	SVR	3	Teste	0.1855	0.43	0.3345	0.92
18	Árvore de Decisão	4	Treino	0.5531	0.74	0.4124	0.91
19	Árvore de Decisão	4	Teste	1.1623	1.08	0.8901	0.51
20	Random Forest	4	Treino	0.1244	0.35	0.1925	0.98
21	Random Forest	4	Teste	1.1921	1.09	0.8701	0.50
22	SVR	4	Treino	0.4586	0.68	0.2897	0.92
23	SVR	4	Teste	0.9126	0.96	0.7488	0.62
24	XGB	1	Treino	1.5425	1.24	0.7065	0.92
25	XGB	1	Teste	0.7387	0.86	0.6388	0.69
26	XGB	2	Treino	0.0244	0.16	0.1082	1.00
27	XGB	2	Teste	0.0884	0.30	0.1921	0.96

```
import pandas as pd
import plotly.express as px


df_plot = df_pred.loc[df_pred.index >= pd.to_datetime('2025-04-05')].reset_index()

df_plot = df_plot.rename(columns={'ELET_CONS_CLT': 'Real', 'previsão_xgb2': 'XGB'})

# Transformar em formato longo para o gráfico
df_plot = df_plot.melt(id_vars=['DATA'], value_vars=['Real', 'XGB'], var_name='Tipo', value_name='Valor')

# Criar gráfico interativo
fig = px.line(df_plot, x='DATA', y='Valor', color='Tipo',
              title='Dados vs Previsão XGB',
              labels={'DATA': 'Data', 'Valor': 'Consumo (kWh/dia)', 'Tipo': ' '})
fig.update_layout(
    title='Previsão do Consumo de Energia Elétrica por Cliente',
    xaxis_title='Data',
    yaxis_title='kWh/dia/clt',
    width=1000,
    height=600,
    title_x=0.5,
    font=dict(size=16),
    template='plotly_white'
)

fig.show()
```



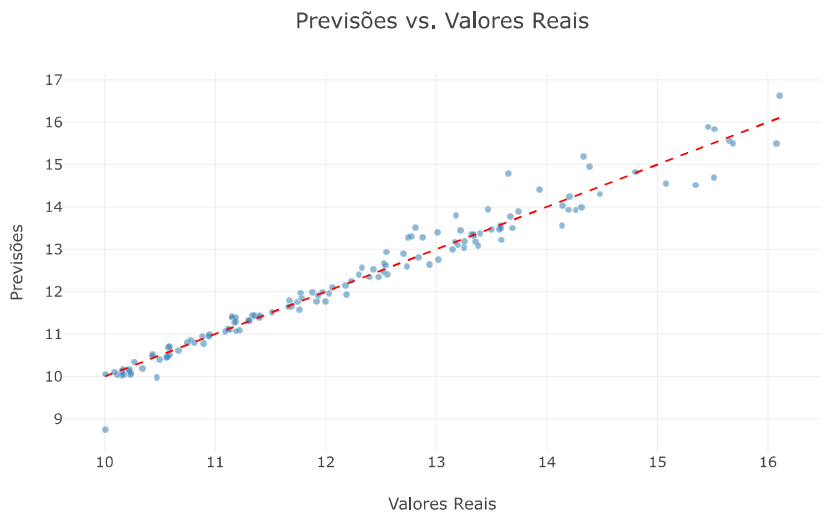

```
# Filtrar os dados a partir de 2025-04-05
df_erro = df_pred.loc[df_pred.index >= pd.to_datetime('2025-04-05')].copy()

# Calcular erro percentual diretamente no DataFrame erro
df_erro['Erro [%]'] = ((df_erro['previsão_xgb2'] - df_erro['ELET_CONS_CLT']) / df_erro['ELET_CONS_CLT']) * 100

# Resetar índice para usar 'DATA' como coluna
df_plot = df_erro.reset_index()

# Criar gráfico interativo
fig = px.line(df_plot, x='DATA', y='Erro [%]',
              title='Erro Diário Percentual da Previsão XGB2',
              labels={'DATA': 'Data', 'Erro [%]': 'Erro (%)'},
              template='plotly_white')

fig.update_layout(width=1000, height=500, title_x=0.5)
fig.show()
```

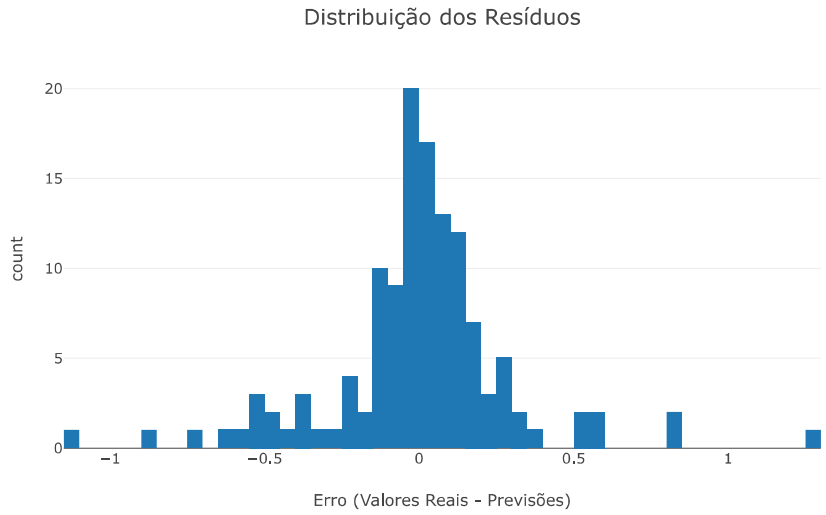



```
#df1 = df1.merge(teste2['previsão_xgb2'], left_index=True, right_index=True, how='left')
#df1.rename(columns={'Previsto': 'Previsões GRU2'}, inplace=True)
#df1.tail()
```

```
df_plot1 = teste2[['ELET_CONS_CLT', 'previsão_xgb2']].reset_index()
```

```
#  o gráfico de dispersão interativo
fig = px.scatter(df_plot1, x='ELET_CONS_CLT', y='previsão_xgb2',
                 title="Previsões vs. Valores Reais",
                 labels={'ELET_CONS_CLT': 'Valores Reais', 'previsão_xgb2': 'Previsões'},
                 opacity=0.5)

# Adiciona a linha de referência
fig.add_shape(type="line", x0=df_plot1['ELET_CONS_CLT'].min(), x1=df_plot1['ELET_CONS_CLT'].max(),
              y0=df_plot1['ELET_CONS_CLT'].min(), y1=df_plot1['ELET_CONS_CLT'].max(),
              line=dict(color="red", dash="dash"))

fig.show()
fig.show('notebook_connected')
# Cria o gráfico de resíduos
df_plot1['Resíduos'] = df_plot1['ELET_CONS_CLT'] - df_plot1['previsão_xgb2']
fig_resíduos = px.histogram(df_plot1, x='Resíduos', nbins=50,
                            title="Distribuição dos Resíduos",
                            labels={'Resíduos': 'Erro (Valores Reais - Previsões)'})

fig_resíduos.show()
fig_resíduos.show('notebook_connected')
```


Previsões vs. Valores Reais
Previsões
Valores Reais
17
16
15
14
13
12
11
10
9
10
11
12
13
14
15
16


Distribuição dos Resíduos
count
Erro (Valores Reais - Previsões)
20
15
10
5
0
−1
−0.5
0
0.5
1

3ª iteração

```
df1_filtrado3.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change'],
      dtype='object')
```

```
df1_filtrado3
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	CO2e_GAS	Total_CO2e	CO2e_QOCP	ELET_CONS_CLT_lag1	ELET_CONS_CLT_lag7	ELET_CONS_CLT_rolling_mean
2023-03-30	18.750000	24.900000	13.8	11.099999	77.516667	91.000000	63.299999	27.700001	21.755307	5679.0	...	1099.25244	2078.96544	18.077960	40.397906	NaN	70.756545
2023-03-31	18.333333	24.400000	14.1	10.299999	85.416667	100.000000	70.000000	30.000000	19.473651	11569.0	...	1465.66992	3478.60292	13.226627	19.185811	166.121212	49.346171
2023-04-02	18.366667	22.100000	14.2	7.900001	74.466667	95.599998	44.400002	51.199997	22.624800	8496.0	...	732.83496	2217.78696	7.809109	16.248596	107.577778	35.480039
2023-04-03	19.140000	25.299999	15.0	10.299999	55.960000	75.500000	44.500000	31.000000	25.191946	9342.0	...	732.83496	2372.69896	7.168275	10.514851	77.728395	25.800285
2023-04-09	18.785715	26.700001	13.6	13.100000	74.428573	87.300003	59.900002	27.400002	22.498339	11715.0	...	2748.13110	4799.36610	14.721982	9.970117	50.528302	20.347327
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2025-08-27	23.416990	31.900000	18.3	13.600000	75.882190	89.000000	51.500000	37.500000	23.980210	12914.0	...	2198.50488	4459.54288	10.021445	10.585227	10.270170	10.325795
2025-08-28	22.236250	30.200000	18.3	11.900000	80.148180	100.000000	54.700000	45.300000	22.372110	12826.0	...	0.00000	2230.10200	5.259675	10.112764	10.165070	10.382938
2025-08-29	22.645070	34.000000	17.5	16.500000	79.591120	91.500000	54.400000	37.100000	22.824760	12985.0	...	2564.92236	4820.41736	12.553170	10.565074	10.747419	10.549428
2025-08-30	24.656980	34.500000	19.7	14.800000	78.813560	92.000000	56.400000	35.600000	25.362570	12416.0	...	2564.92236	4722.49436	12.973886	11.912844	10.176106	10.836340
2025-08-31	22.518880	30.100000	18.7	11.400000	83.646050	98.500000	59.900000	38.600000	22.132800	12372.0	...	1832.08740	3978.83140	10.254720	12.184495	10.341620	11.063333

552 rows × 48 columns

```
df1_filtrado3.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change'],
      dtype='object')
```

```
treino3 = df1_filtrado3.loc[
    ((df1_filtrado3.index > '2023-04-01') & (df1_filtrado3.index <= '2023-10-31')) |
    ((df1_filtrado3.index > '2024-04-01') & (df1_filtrado3.index <= '2024-10-31'))
].copy()


teste3 = df1_filtrado3.loc[df1_filtrado3.index >= '2025-04-05'].copy ()

# Aplicar função
#treino3 = create_features(treino3)
#teste3 = create_features(teste3)

# Definir variáveis de entrada e saída
input = ['OCP', 'Q_OCP', 'CLTs', 'T_mean', 'Calor_idx', 'RH_mean',
       'dia_semana', 'trimestre', 'mes', 'ano',
       'dia_ano', 'dia_mes', 'semana_ano', 'ELET_CONS_CLT_lag1', 'ELET_CONS_CLT_lag7',
       'ELET_CONS_CLT_rolling_mean', 'ELET_CONS_CLT_rolling_median',
       'ELET_CONS_CLT_rolling_std', 'ELET_CONS_CLT_error_median',
       'ELET_CONS_CLT_pct_change'

           ]

output = 'ELET_CONS_CLT'

x_treino3 = treino3[input]
y_treino3 = treino3[output]

x_teste3 = teste3[input]
y_teste3 = teste3[output]

treino3
```

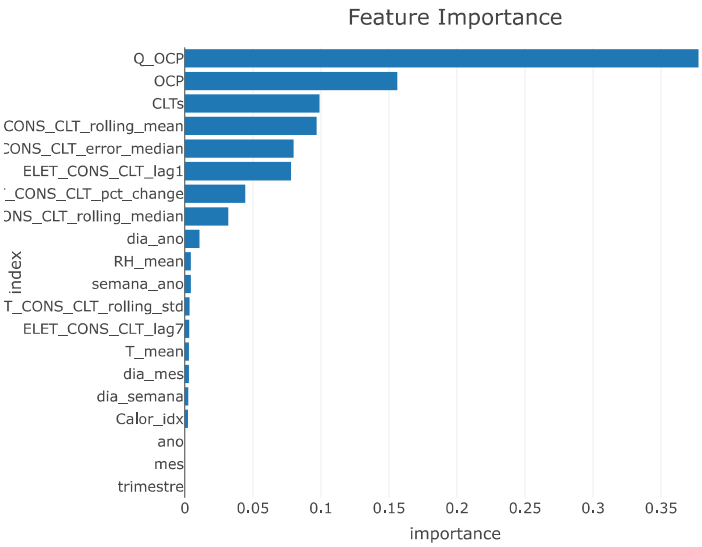
DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	CO2e_GAS	Total_CO2e	CO2e_QOCP	ELET_CONS_CLT_lag1	ELET_CONS_CLT_lag7	ELET_CONS_CLT_rolling_mean
2023-04-02	18.366667	22.100000	14.2	7.900001	74.466667	95.599998	44.400002	51.199997	22.624800	8496.0	...	732.83496	2217.78696	7.809109	16.248596	107.577778	35.480039
2023-04-03	19.140000	25.299999	15.0	10.299999	55.960000	75.500000	44.500000	31.000000	25.191946	9342.0	...	732.83496	2372.69896	7.168275	10.514851	77.728395	25.800285
2023-04-09	18.785715	26.700001	13.6	13.100000	74.428573	87.300003	59.900002	27.400002	22.498339	11715.0	...	2748.13110	4799.36610	14.721982	9.970117	50.528302	20.347327
2023-04-11	19.066667	25.500000	14.1	11.400000	75.066667	86.500000	58.500000	28.000000	22.289491	9595.0	...	2931.33984	4615.77484	15.133688	12.357595	33.756410	17.053092
2023-04-13	15.916667	24.500000	11.1	13.400000	66.066668	82.400002	40.700001	41.700001	26.361804	9757.0	...	2015.29614	3714.89514	12.722244	10.696767	40.397906	12.895223
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2024-10-25	17.860000	20.500000	15.6	4.900000	85.900002	100.000000	71.300003	28.699997	19.363441	10862.0	...	2564.92236	4444.70636	10.974584	10.826283	10.768186	10.366977
2024-10-26	14.928572	21.600000	11.0	10.600000	92.014285	99.800003	75.699997	24.100006	18.224307	11462.0	...	2748.13110	4742.32510	11.003074	9.956004	11.885312	10.067392
2024-10-29	16.983333	21.100000	14.7	6.400001	84.416667	95.300003	71.000000	24.300003	20.204971	11157.0	...	2931.33984	4860.38884	11.912718	9.788215	10.041516	10.071389
2024-10-30	15.542857	19.500000	13.2	6.300000	94.757143	100.000000	81.500000	18.500000	16.309118	11328.0	...	2564.92236	4513.82836	11.973020	10.069495	9.645635	10.289384
2024-10-31	17.316667	20.100000	13.9	6.200001	99.816667	100.000000	98.900002	1.099998	13.311978	10224.0	...	3114.54858	4880.82658	16.772600	11.171598	10.128298	10.710222

397 rows × 48 columns

```
reg3 = xgb.XGBRegressor(base_score=0.01, booster='gbtree',
                        n_estimators=100000,
                       early_stopping_rounds=8000,
                       objective='reg:squarederror',
                         reg_lambda=5,
                         reg_alpha=0.5,
                       max_depth=1,
                         min_child_weight=3,
                         subsample=0.95,
                         colsample_bytree=0.85,
                         gamma=0.05,

                       learning_rate=0.05)

reg3.fit(x_treino3, y_treino3,
        eval_set=[(x_treino3, y_treino3), (x_teste3, y_teste3)],
        verbose=1000)
```

Mostrar saída oculta

```
fi = pd.DataFrame(data=reg3.feature_importances_,
                  index=reg3.feature_names_in_,
                  columns=['importance'])

fi = fi.sort_values('importance')

# o gráfico de barras horizontal interativo
fig = px.bar(fi, x='importance', y=fi.index, orientation='h',
             title='Feature Importance')
fig.update_layout(
    yaxis_title_standoff=30  # Aumente o valor para afastar mais
)
fig.update_layout(
    margin=dict(l=200, r=200, t=50, b=50)  # Aumenta a margem direita (r) para mover o gráfico
)
fig.show()
fig.show('notebook_connected')
```



Resultados

```
#df1_filtrado3 = df1_filtrado3.drop(columns=['previsão_xgb1_x', 'previsão_xgb1_y'])
#df1_filtrado3 = df1_filtrado3.drop(columns=['previsão_xgb1'])
```

```
teste3['previsão_xgb3'] = reg3.predict(x_teste3)
df_pred = df_pred.merge(teste3[['previsão_xgb3']], how='left', left_index=True, right_index=True)
```

```
#df1 = df1.merge(teste3[['previsão_xgb3']], how='left', left_index=True, right_index=True)
```

```
df_pred.tail()
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	ELET_CONS_CLT_lag1	ELET_CONS_CLT_lag7	ELET_CONS_CLT_rolling_mean	ELET_CONS_CLT_rolling_median	ELET_CONS_CLT_rolling_s
2025-08-27	23.41699	31.9	18.3	13.6	75.88219	89.0	51.5	37.5	23.98021	12914.0	...	10.585227	10.270170	10.325795	10.176106	0.2480
2025-08-28	22.23625	30.2	18.3	11.9	80.14818	100.0	54.7	45.3	22.37211	12826.0	...	10.112764	10.165070	10.382938	10.341620	0.2508
2025-08-29	22.64507	34.0	17.5	16.5	79.59112	91.5	54.4	37.1	22.82476	12985.0	...	10.565074	10.747419	10.549428	10.341620	0.6313
2025-08-30	24.65698	34.5	19.7	14.8	78.81356	92.0	56.4	35.6	25.36257	12416.0	...	11.912844	10.176106	10.836340	10.565074	0.8513
2025-08-31	22.51888	30.1	18.7	11.4	83.64605	98.5	59.9	38.6	22.13280	12372.0	...	12.184495	10.341620	11.063333	10.585227	0.9074

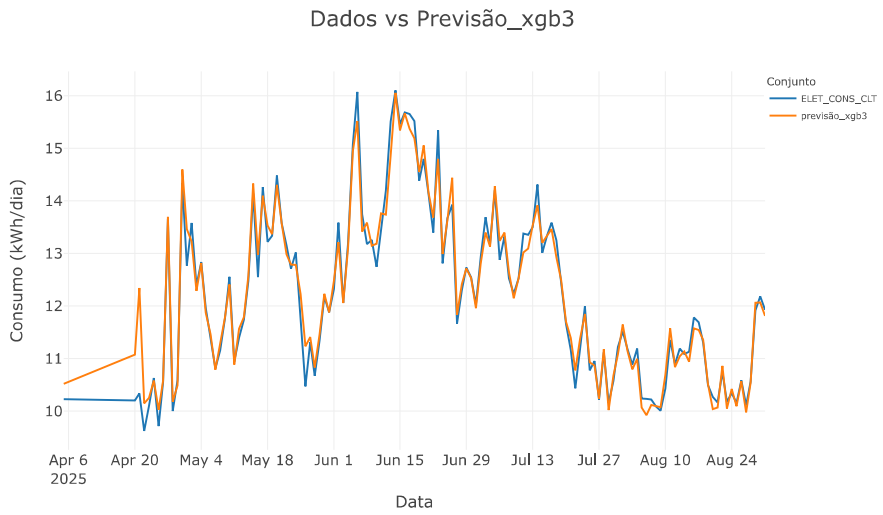
5 rows × 51 columns

Comece a programar ou gerar com a IA.

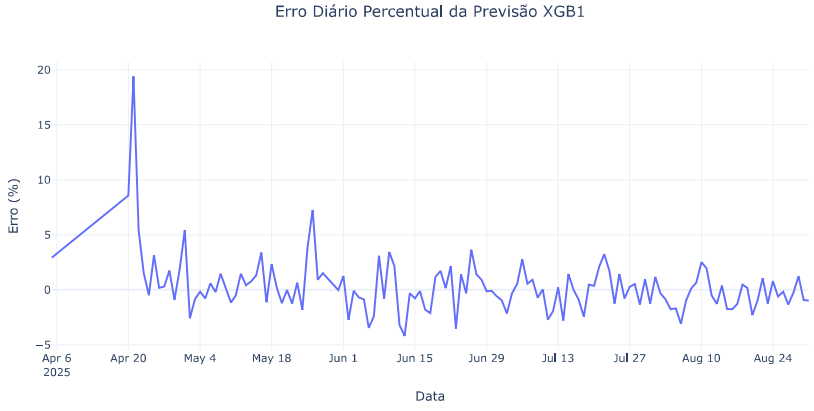
```
# ==========================
# Chamar a função para cada modelo
# ==========================
avaliar_modelo(reg3, x_treino3, y_treino3, x_teste3, y_teste3, 'XGB', 3)

# Criar DataFrame com todos os resultados
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nTabela de Avaliação:")
print(df_resultados)
```

```
XGB (Iteração 3) - TREINO → MSE: 0.0237 | RMSE: 0.15 | MAE: 0.1113 | R²: 1.00
XGB (Iteração 3) - TESTE  → MSE: 0.0876 | RMSE: 0.30 | MAE: 0.1909 | R²: 0.97

Tabela de Avaliação:
              Modelo  Iteração Conjunto     MSE  RMSE     MAE    R²
0   Árvore de Decisão         1   Treino  4.3337  2.08  0.7089  0.81
1   Árvore de Decisão         1    Teste  1.4757  1.21  0.9388  0.41
2       Random Forest         1   Treino  0.9935  1.00  0.3682  0.96
3       Random Forest         1    Teste  0.8964  0.95  0.7417  0.64
4                 SVR         1   Treino  5.3172  2.31  0.5493  0.76
5                 SVR         1    Teste  0.7506  0.87  0.6945  0.70
6   Árvore de Decisão         2   Treino  3.7421  1.93  0.4133  0.83
7   Árvore de Decisão         2    Teste  0.3438  0.59  0.3803  0.86
8       Random Forest         2   Treino  0.7137  0.84  0.1967  0.97
9       Random Forest         2    Teste  0.0818  0.29  0.1801  0.97
10                SVR         2   Treino  6.1005  2.47  0.2629  0.72
11                SVR         2    Teste  0.0836  0.29  0.2376  0.97
12  Árvore de Decisão         3   Treino  0.3084  0.56  0.2801  0.95
13  Árvore de Decisão         3    Teste  1.0447  1.02  0.7326  0.56
14      Random Forest         3   Treino  0.0769  0.28  0.1294  0.99
15      Random Forest         3    Teste  0.6032  0.78  0.5853  0.75
16                SVR         3   Treino  0.0155  0.12  0.0900  1.00
17                SVR         3    Teste  0.1855  0.43  0.3345  0.92
18  Árvore de Decisão         4   Treino  0.5531  0.74  0.4124  0.91
19  Árvore de Decisão         4    Teste  1.1623  1.08  0.8901  0.51
20      Random Forest         4   Treino  0.1244  0.35  0.1925  0.98
21      Random Forest         4    Teste  1.1921  1.09  0.8701  0.50
22                SVR         4   Treino  0.4586  0.68  0.2897  0.92
23                SVR         4    Teste  0.9126  0.96  0.7488  0.62
24                XGB         1   Treino  1.5425  1.24  0.7065  0.92
25                XGB         1    Teste  0.7387  0.86  0.6388  0.69
26                XGB         2   Treino  0.0244  0.16  0.1082  1.00
27                XGB         2    Teste  0.0884  0.30  0.1921  0.96
28                XGB         3   Treino  0.0237  0.15  0.1113  1.00
29                XGB         3    Teste  0.0876  0.30  0.1909  0.97
```

```
df_plot = df_pred.loc[df_pred.index >= '2025-04-05'].reset_index()

# Criando um DataFrame para diferenciar os conjuntos
df_plot = df_plot.melt(id_vars=['DATA'], value_vars=['ELET_CONS_CLT', 'previsão_xgb3'], var_name='Tipo', value_name='Valor')

# Criando o gráfico interativo
fig = px.line(df_plot, x='DATA', y='Valor', color='Tipo',
              title='Dados vs Previsão_xgb3',
              labels={'DATA': 'Data', 'Valor': 'Consumo (kWh/dia)', 'Tipo': 'Conjunto'})

fig.show()
fig.show('notebook_connected')
```

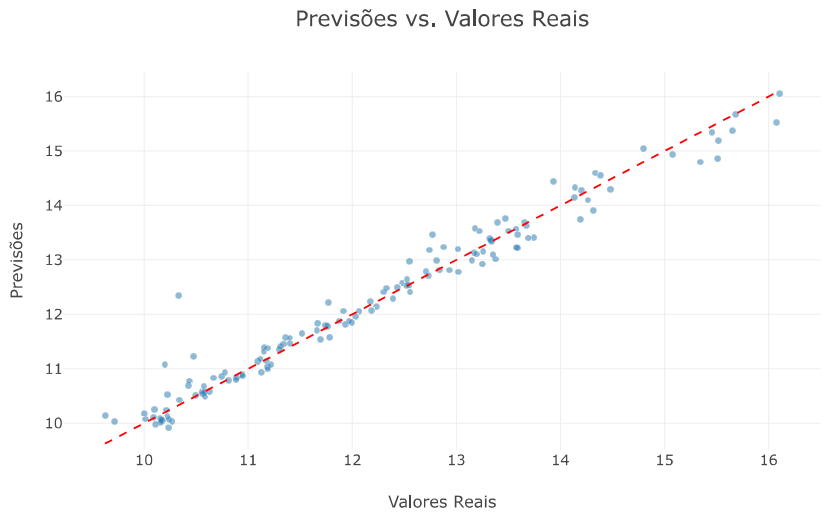


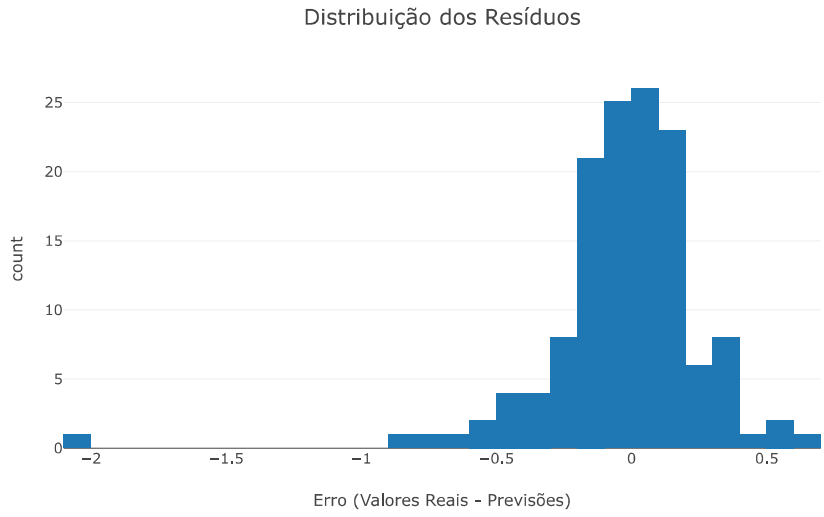

```
df_erro = df_pred.loc[df_pred.index >= pd.to_datetime('2025-04-05')].copy()

# Calcular erro percentual diretamente no DataFrame erro
df_erro['Erro [%]'] = ((df_erro['previsão_xgb3'] - df_erro['ELET_CONS_CLT']) / df_erro['ELET_CONS_CLT']) * 100

# Resetar índice para usar 'DATA' como coluna
df_plot = df_erro.reset_index()

# Criar gráfico interativo
fig = px.line(df_plot, x='DATA', y='Erro [%]',
              title='Erro Diário Percentual da Previsão XGB1',
              labels={'DATA': 'Data', 'Erro [%]': 'Erro (%)'},
              template='plotly_white')

fig.update_layout(width=1000, height=500, title_x=0.5)
fig.show()
```




```
df_plot3 = teste3[['ELET_CONS_CLT', 'previsão_xgb3']].reset_index()

#  o gráfico de dispersão interativo
fig = px.scatter(df_plot3, x='ELET_CONS_CLT', y='previsão_xgb3',
                 title="Previsões vs. Valores Reais",
                 labels={'ELET_CONS_CLT': 'Valores Reais', 'previsão_xgb3': 'Previsões'},
                 opacity=0.5)

# Adiciona a linha de referência
fig.add_shape(type="line", x0=df_plot3['ELET_CONS_CLT'].min(), x1=df_plot3['ELET_CONS_CLT'].max(),
              y0=df_plot3['ELET_CONS_CLT'].min(), y1=df_plot3['ELET_CONS_CLT'].max(),
              line=dict(color="red", dash="dash"))

fig.show()
fig.show('notebook_connected')
# Cria o gráfico de resíduos
df_plot3['Resíduos'] = df_plot3['ELET_CONS_CLT'] - df_plot3['previsão_xgb3']
fig_residuos = px.histogram(df_plot3, x='Resíduos', nbins=50,
```

```
                          title="Distribuição dos Resíduos",
                          labels={'Resíduos': 'Erro (Valores Reais - Previsões)'})

fig_residuos.show()
fig_residuos.show('notebook_connected')
```






```
##df1 = df1.merge(teste3['previsão_xgb3'], left_index=True, right_index=True, how='left')
#df1.rename(columns={'Previsto': 'Previsões GRU2'}, inplace=True)
```

```
#df1.tail()
```

4ª itereção

```
treino4 = df1_filtrado3.loc[(df1_filtrado3.index > '2024-04-01') & (df1_filtrado3.index <= '2024-10-31')].copy()
teste4 = df1_filtrado3.loc[df1_filtrado3.index >= '2025-04-05'].copy()
```

```
treino4
```

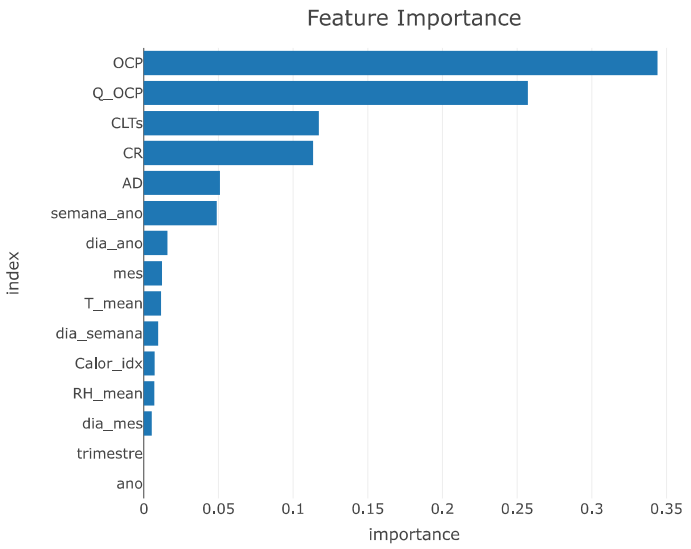
Mostrar saída oculta

```
#treino4 = df1_filtrado3.loc[(df1_filtrado3.index > '2024-04-01') & (df1_filtrado3.index <= '2024-10-31')].copy()
#teste4 = df1_filtrado3.loc[df1_filtrado3.index >= '2025-04-05'].copy()

treino4 = create_features(treino4)
teste4 = create_features(teste4)

# em função do nivel de correlação, foram escolhidos os inputos
input = ['Q_OCP', 'OCP', 'AD', 'CR', 'CLTs', 'T_mean', 'Calor_idx', 'RH_mean',
       'dia_semana', 'trimestre', 'mes', 'ano',
       'dia_ano', 'dia_mes', 'semana_ano'
           ]

output = ['ELET_CONS_CLT']

x_treino4 = treino4[input]
y_treino4 = treino4[output]

x_teste4= teste4[input]
y_teste4= teste4[output]
```

```
reg4 = xgb.XGBRegressor(base_score=0.01, booster='gbtree',
                        n_estimators=100000,
                       early_stopping_rounds=8000,
                       objective='reg:squarederror',
                         reg_lambda=5,
                         reg_alpha=0.5,
                       max_depth=1,
                         min_child_weight=3,
                         subsample=0.95,
                         colsample_bytree=0.85,
                         gamma=0.05,

                       learning_rate=0.05)

reg4.fit(x_treino4, y_treino4,
        eval_set=[(x_treino4, y_treino4), (x_teste4, y_teste4)],
        verbose=1000)
```

Mostrar saída oculta

```
fi = pd.DataFrame(data=reg4.feature_importances_,
                  index=reg4.feature_names_in_,
                  columns=['importance'])

fi = fi.sort_values('importance')

# o gráfico de barras horizontal interativo
fig = px.bar(fi, x='importance', y=fi.index, orientation='h',
             title='Feature Importance')
fig.update_layout(
    yaxis_title_standoff=30  # Aumente o valor para afastar mais
)
fig.update_layout(
    margin=dict(l=200, r=200, t=50, b=50)  # Aumenta a margem direita (r) para mover o gráfico
)
fig.show()
#fig.show('notebook_connected')
```




```
Comece a programar ou gerar com a IA.
```

Resultados

```
teste4
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	CO2e_GAS	Total_CO2e	CO2e_QOCP	ELET_CONS_CLT_lag1	ELET_CONS_CLT_lag7	ELET_CONS_CLT_rolling_m
2025-04-26	20.000000	26.799999	13.400000	13.400000	78.880000	95.900002	65.699997	30.200005	21.422470	10548.0	...	3664.17480	5495.00080	13.567903	9.715201	10.224138	10.166
2025-04-27	20.414286	25.600000	15.800000	9.800000	79.342858	87.800003	59.900002	27.900002	21.427752	10392.0	...	1465.66992	3271.02392	10.384203	10.558559	10.200209	10.659
2025-04-28	20.420000	24.299999	16.299999	8.000000	80.360000	100.000000	60.700001	39.299999	21.207170	7292.0	...	2381.71362	3661.53762	11.849636	13.655716	10.334694	10.612
2025-04-29	19.450000	23.200001	17.600000	5.600000	91.933334	97.300003	83.699997	13.600006	17.526266	7294.0	...	2564.92236	3844.79036	12.989157	10.002743	9.624135	10.749
2025-04-30	17.742857	23.299999	13.700000	9.599999	89.528571	100.000000	73.900002	26.099998	18.012499	9818.0	...	2381.71362	4084.24962	14.035222	10.586357	10.097963	11.354
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2025-08-27	23.416990	31.900000	18.300000	13.600000	75.882190	89.000000	51.500000	37.500000	23.980210	12914.0	...	2198.50488	4459.54288	10.021445	10.585227	10.270170	10.325
2025-08-28	22.236250	30.200000	18.300000	11.900000	80.148180	100.000000	54.700000	45.300000	22.372110	12826.0	...	0.00000	2230.10200	5.259675	10.112764	10.165070	10.382
2025-08-29	22.645070	34.000000	17.500000	16.500000	79.591120	91.500000	54.400000	37.100000	22.824760	12985.0	...	2564.92236	4820.41736	12.553170	10.565074	10.747419	10.549
2025-08-30	24.656980	34.500000	19.700000	14.800000	78.813560	92.000000	56.400000	35.600000	25.362570	12416.0	...	2564.92236	4722.49436	12.973886	11.912844	10.176106	10.836
2025-08-31	22.518880	30.100000	18.700000	11.400000	83.646050	98.500000	59.900000	38.600000	22.132800	12372.0	...	1832.08740	3978.83140	10.254720	12.184495	10.341620	11.063

128 rows × 48 columns

```
df_pred.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change',
       'previsão_svr', 'previsão_rf', 'previsão_tree', 'erro_relativo_rf',
       'erro_relativo_svr', 'previsão_xgb1', 'previsão_xgb2', 'previsão_xgb3'],
      dtype='object')
```

```
teste4['previsão_xgb4'] = reg4.predict(x_teste4)
df_pred = df_pred.merge(teste4[['previsão_xgb4']], how='left', left_index=True, right_index=True)
```

```
df_pred.tail()
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	ELET_CONS_CLT_lag7	ELET_CONS_CLT_rolling_mean	ELET_CONS_CLT_rolling_median	ELET_CONS_CLT_rolling_std	ELET_CONS_CLT_er
2025-08-27	23.41699	31.9	18.3	13.6	75.88219	89.0	51.5	37.5	23.98021	12914.0	...	10.270170	10.325795	10.176106	0.248000	
2025-08-28	22.23625	30.2	18.3	11.9	80.14818	100.0	54.7	45.3	22.37211	12826.0	...	10.165070	10.382938	10.341620	0.250861	
2025-08-29	22.64507	34.0	17.5	16.5	79.59112	91.5	54.4	37.1	22.82476	12985.0	...	10.747419	10.549428	10.341620	0.631311	
2025-08-30	24.65698	34.5	19.7	14.8	78.81356	92.0	56.4	35.6	25.36257	12416.0	...	10.176106	10.836340	10.565074	0.851388	
2025-08-31	22.51888	30.1	18.7	11.4	83.64605	98.5	59.9	38.6	22.13280	12372.0	...	10.341620	11.063333	10.585227	0.907476	

5 rows × 52 columns

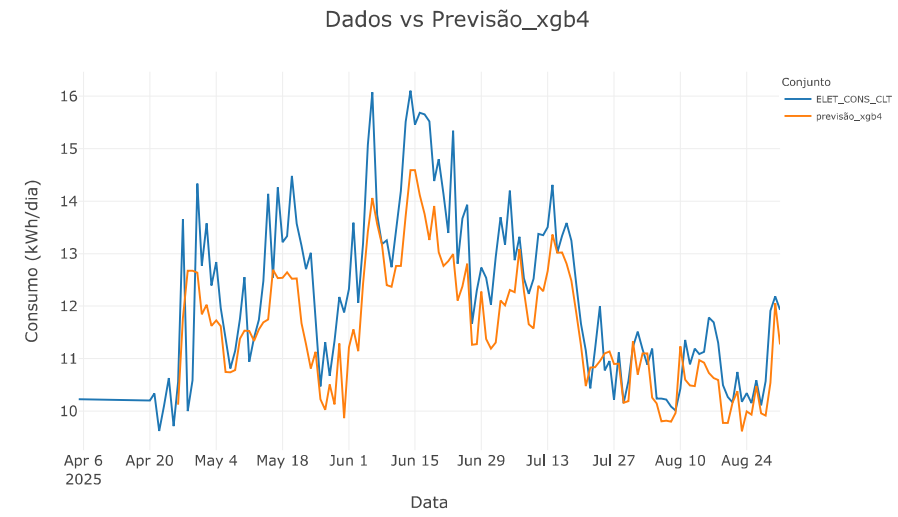
```
# =========================
# Chamando a função para cada modelo
# =========================
avaliar_modelo(reg4, x_treino4, y_treino4, x_teste4, y_teste4, 'XGB', 4)

# Criar DataFrame com todos os resultados
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nTabela de Avaliação:")
print(df_resultados)
```

```
XGB (Iteração 4) - TREINO → MSE: 0.3524 | RMSE: 0.59 | MAE: 0.3203 | R²: 0.94
XGB (Iteração 4) - TESTE  → MSE: 1.1194 | RMSE: 1.06 | MAE: 0.8464 | R²: 0.53

Tabela de Avaliação:
              Modelo  Iteração Conjunto     MSE  RMSE     MAE    R²
0   Árvore de Decisão         1   Treino  4.3337  2.08  0.7089  0.81
1   Árvore de Decisão         1    Teste  1.4757  1.21  0.9388  0.41
2       Random Forest         1   Treino  0.9935  1.00  0.3682  0.96
3       Random Forest         1    Teste  0.8964  0.95  0.7417  0.64
4                 SVR         1   Treino  5.3172  2.31  0.5493  0.76
5                 SVR         1    Teste  0.7506  0.87  0.6945  0.70
6   Árvore de Decisão         2   Treino  3.7421  1.93  0.4133  0.83
7   Árvore de Decisão         2    Teste  0.3438  0.59  0.3803  0.86
8       Random Forest         2   Treino  0.7137  0.84  0.1967  0.97
9       Random Forest         2    Teste  0.0818  0.29  0.1801  0.97
10                SVR         2   Treino  6.1005  2.47  0.2629  0.72
11                SVR         2    Teste  0.0836  0.29  0.2376  0.97
12  Árvore de Decisão         3   Treino  0.3084  0.56  0.2801  0.95
13  Árvore de Decisão         3    Teste  1.0447  1.02  0.7326  0.56
14      Random Forest         3   Treino  0.0769  0.28  0.1294  0.99
15      Random Forest         3    Teste  0.6032  0.78  0.5853  0.75
16                SVR         3   Treino  0.0155  0.12  0.0900  1.00
17                SVR         3    Teste  0.1855  0.43  0.3345  0.92
18  Árvore de Decisão         4   Treino  0.5531  0.74  0.4124  0.91
19  Árvore de Decisão         4    Teste  1.1623  1.08  0.8901  0.51
20      Random Forest         4   Treino  0.1244  0.35  0.1925  0.98
21      Random Forest         4    Teste  1.1921  1.09  0.8701  0.50
22                SVR         4   Treino  0.4586  0.68  0.2897  0.92
23                SVR         4    Teste  0.9126  0.96  0.7488  0.62
24                XGB         1   Treino  1.5425  1.24  0.7065  0.92
25                XGB         1    Teste  0.7387  0.86  0.6388  0.69
26                XGB         2   Treino  0.0244  0.16  0.1082  1.00
27                XGB         2    Teste  0.0884  0.30  0.1921  0.96
28                XGB         3   Treino  0.0237  0.15  0.1113  1.00
29                XGB         3    Teste  0.0876  0.30  0.1909  0.97
30                XGB         4   Treino  0.3524  0.59  0.3203  0.94
31                XGB         4    Teste  1.1194  1.06  0.8464  0.53
```

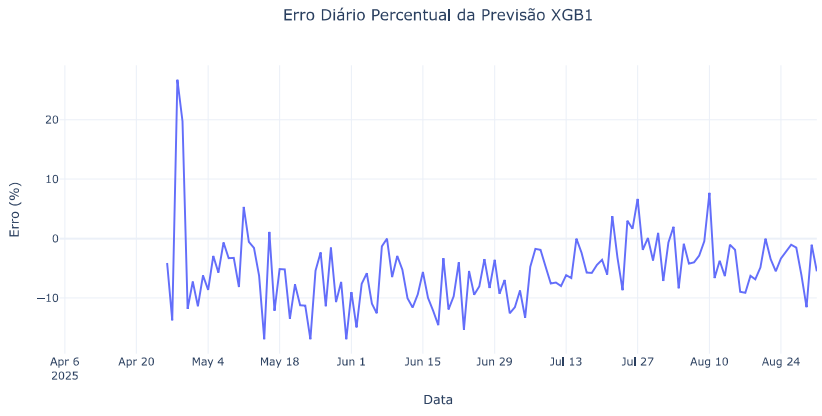
```
df_resultados
```

Mostrar saída oculta

```
df_plot = df_pred.loc[df_pred.index >= '2025-04-05'].reset_index()

# Criando um DataFrame para diferenciar os conjuntos
df_plot = df_plot.melt(id_vars=['DATA'], value_vars=['ELET_CONS_CLT', 'previsão_xgb4'], var_name='Tipo', value_name='Valor')

# Criando o gráfico interativo
fig = px.line(df_plot, x='DATA', y='Valor', color='Tipo',
              title='Dados vs Previsão_xgb4',
              labels={'DATA': 'Data', 'Valor': 'Consumo (kWh/dia)', 'Tipo': 'Conjunto'})

fig.show()
fig.show('notebook_connected')
```




```
# Filtrar os dados a partir de 2025-04-05
df_erro = df_pred.loc[df_pred.index >= pd.to_datetime('2025-04-05')].copy()

# Calcular erro percentual diretamente no DataFrame erro
df_erro['Erro [%]'] = ((df_erro['previsão_xgb4'] - df_erro['ELET_CONS_CLT']) / df_erro['ELET_CONS_CLT']) * 100

# Resetar índice para usar 'DATA' como coluna
df_plot = df_erro.reset_index()

# Criar gráfico interativo
fig = px.line(df_plot, x='DATA', y='Erro [%]',
              title='Erro Diário Percentual da Previsão XGB1',
              labels={'DATA': 'Data', 'Erro [%]': 'Erro (%)'},
              template='plotly_white')

fig.update_layout(width=1000, height=500, title_x=0.5)
fig.show()
```



reduzir os inputs

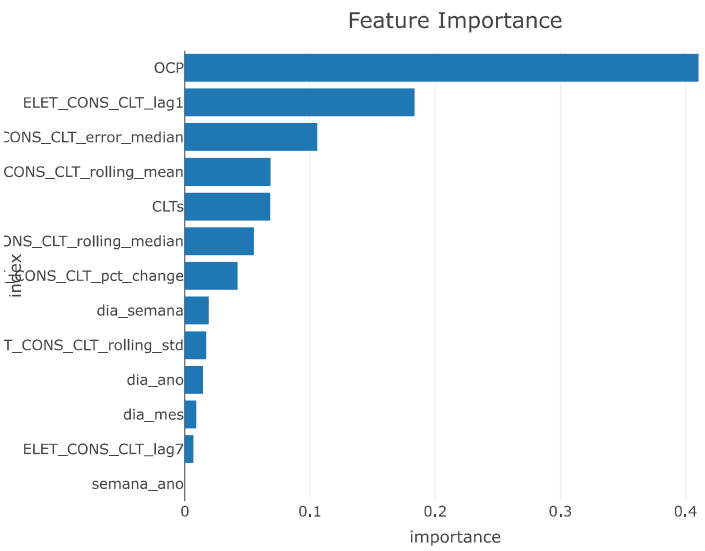
```
treino4 = df1_filtrado3.loc[(df1_filtrado3.index > '2024-04-01') & (df1_filtrado3.index <= '2024-10-31')].copy()
teste4 = df1_filtrado3.loc[df1_filtrado3.index >= '2025-04-05'].copy()
```

```
#treino4 = create_features(treino4)
#teste4 = create_features(teste4)

# em função do nivel de correlação, foram escolhidos os inputos
input = ['OCP', 'CLTs',
       'dia_semana',
       'dia_ano', 'dia_mes', 'semana_ano', 'ELET_CONS_CLT_lag1', 'ELET_CONS_CLT_lag7',
       'ELET_CONS_CLT_rolling_mean', 'ELET_CONS_CLT_rolling_median',
       'ELET_CONS_CLT_rolling_std', 'ELET_CONS_CLT_error_median',
       'ELET_CONS_CLT_pct_change'
            ]

output = 'ELET_CONS_CLT'

x_treino4 = treino4[input]
y_treino4 = treino4[output]

x_teste4= teste4[input]
y_teste4= teste4[output]
```

```
reg4 = xgb.XGBRegressor(base_score=0.01, booster='gbtree',
                        n_estimators=100000,
                       early_stopping_rounds=8000,
                       objective='reg:squarederror',
                         reg_lambda=5,
                         reg_alpha=0.5,
                       max_depth=1,
                         min_child_weight=3,
                         subsample=0.95,
                         colsample_bytree=0.85,
                         gamma=0.05,

                       learning_rate=0.05)

reg4.fit(x_treino4, y_treino4,
        eval_set=[(x_treino4, y_treino4), (x_teste4, y_teste4)],
        verbose=1000)
```

Mostrar saída oculta

```
fi = pd.DataFrame(data=reg4.feature_importances_,
                  index=reg4.feature_names_in_,
                  columns=['importance'])

fi = fi.sort_values('importance')

# o gráfico de barras horizontal interativo
fig = px.bar(fi, x='importance', y=fi.index, orientation='h',
             title='Feature Importance')
fig.update_layout(
    yaxis_title_standoff=30  # Aumente o valor para afastar mais
)
fig.update_layout(
    margin=dict(l=200, r=200, t=50, b=50)  # Aumenta a margem direita (r) para mover o gráfico
)
fig.show()
#fig.show('notebook_connected')
```



Resultados

```
#df_pred = df_pred.drop(columns=['previsão_xgb1_x', 'previsão_xgb1_y'])
#df_pred = df_pred.drop(columns=['previsão_xgb5'])
```

```
df_pred.tail()
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	ELET_CONS_CLT_pct_change	previsão_svr	previsão_rf	previsão_tree	erro_relativo_rf	erro_relativo_svr	previsão_xgb1	pr
2025-08-27	23.41699	31.9	18.3	13.6	75.88219	89.0	51.5	37.5	23.98021	12914.0	...	-0.044634	10.275543	10.280448	9.926178	1.658136	1.609637	11.066648	
2025-08-28	22.23625	30.2	18.3	11.9	80.14818	100.0	54.7	45.3	22.37211	12826.0	...	0.044727	10.598425	10.830714	10.120344	2.514322	0.315668	11.039792	
2025-08-29	22.64507	34.0	17.5	16.5	79.59112	91.5	54.4	37.1	22.82476	12985.0	...	0.127568	11.385425	10.639877	10.120344	-10.685673	-4.427316	11.288419	
2025-08-30	24.65698	34.5	19.7	14.8	78.81356	92.0	56.4	35.6	25.36257	12416.0	...	0.022803	12.174222	12.325657	12.483464	1.158541	-0.084308	12.505721	
2025-08-31	22.51888	30.1	18.7	11.4	83.64605	98.5	59.9	38.6	22.13280	12372.0	...	-0.020840	11.372169	11.410585	10.797468	-4.358417	-4.680411	12.032969	

5 rows × 57 columns

```
teste4['previsão_xgb5'] = reg4.predict(x_teste4)
df_pred = df_pred.merge(teste4[['previsão_xgb5']], how='left', left_index=True, right_index=True)
```

```
df_pred.tail()
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	previsão_svr	previsão_rf	previsão_tree	erro_relativo_rf	erro_relativo_svr	previsão_xgb1	previsão_xgb2	previsão_xgb3
2025-08-27	23.41699	31.9	18.3	13.6	75.88219	89.0	51.5	37.5	23.98021	12914.0	...	10.275543	10.280448	9.926178	1.658136	1.609637	11.066648	10.090389	10.097376
2025-08-28	22.23625	30.2	18.3	11.9	80.14818	100.0	54.7	45.3	22.37211	12826.0	...	10.598425	10.830714	10.120344	2.514322	0.315668	11.039792	10.437948	10.516932
2025-08-29	22.64507	34.0	17.5	16.5	79.59112	91.5	54.4	37.1	22.82476	12985.0	...	11.385425	10.639877	10.120344	-10.685673	-4.427316	11.288419	11.749044	12.104768
2025-08-30	24.65698	34.5	19.7	14.8	78.81356	92.0	56.4	35.6	25.36257	12416.0	...	12.174222	12.325657	12.483464	1.158541	-0.084308	12.505721	11.900819	12.119337
2025-08-31	22.51888	30.1	18.7	11.4	83.64605	98.5	59.9	38.6	22.13280	12372.0	...	11.372169	11.410585	10.797468	-4.358417	-4.680411	12.032969	11.876614	11.801851

5 rows × 58 columns

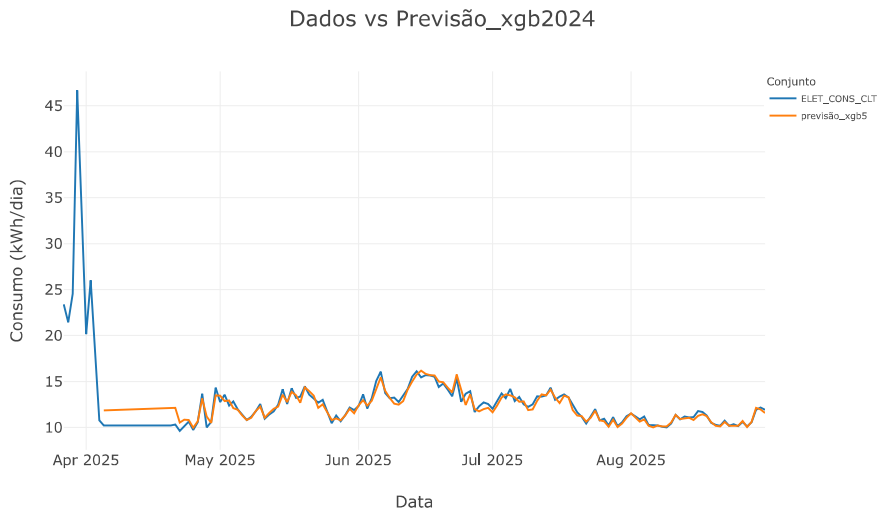
```
# ========================
# Chamando a função para cada modelo
# ========================
avaliar_modelo(reg4, x_treino4, y_treino4, x_teste4, y_teste4, 'XGB', 5)

# Criar DataFrame com todos os resultados
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nTabela de Avaliação:")
print(df_resultados)
```

```
XGB (Iteração 5) - TREINO → MSE: 0.0779 | RMSE: 0.28 | MAE: 0.1593 | R²: 0.99
XGB (Iteração 5) - TESTE  → MSE: 0.2341 | RMSE: 0.48 | MAE: 0.3431 | R²: 0.91

Tabela de Avaliação:
              Modelo  Iteração Conjunto     MSE  RMSE     MAE    R²
0   Árvore de Decisão         1   Treino  4.3240  2.08  0.7078  0.81
1   Árvore de Decisão         1    Teste  1.3907  1.18  0.9151  0.45
2       Random Forest         1   Treino  1.0877  1.04  0.3705  0.95
3       Random Forest         1    Teste  0.9618  0.98  0.7710  0.62
4                 SVR         1   Treino  5.3051  2.30  0.5483  0.76
5                 SVR         1    Teste  0.7486  0.87  0.6934  0.70
6   Árvore de Decisão         2   Treino  3.7339  1.93  0.4133  0.83
7   Árvore de Decisão         2    Teste  0.3538  0.59  0.3926  0.86
8       Random Forest         2   Treino  0.7114  0.84  0.1896  0.97
9       Random Forest         2    Teste  0.0760  0.28  0.1788  0.97
10                SVR         2   Treino  6.0878  2.47  0.2625  0.72
11                SVR         2    Teste  0.0834  0.29  0.2368  0.97
12  Árvore de Decisão         3   Treino  0.3084  0.56  0.2833  0.95
13  Árvore de Decisão         3    Teste  1.0638  1.03  0.7619  0.55
14      Random Forest         3   Treino  0.0778  0.28  0.1311  0.99
15      Random Forest         3    Teste  0.6197  0.79  0.5837  0.74
16                SVR         3   Treino  0.0155  0.12  0.0906  1.00
17                SVR         3    Teste  0.1841  0.43  0.3334  0.92
18  Árvore de Decisão         4   Treino  0.5509  0.74  0.4123  0.91
19  Árvore de Decisão         4    Teste  1.2960  1.14  0.9236  0.45
20      Random Forest         4   Treino  0.1322  0.36  0.1966  0.98
21      Random Forest         4    Teste  1.1817  1.09  0.8754  0.50
22                SVR         4   Treino  0.4558  0.68  0.2887  0.92
23                SVR         4    Teste  0.9121  0.96  0.7485  0.62
24                XGB         1   Treino  1.5318  1.24  0.7046  0.92
25                XGB         1    Teste  0.7402  0.86  0.6376  0.69
26                XGB         2   Treino  0.0441  0.21  0.1383  0.99
27                XGB         2    Teste  0.0917  0.30  0.1882  0.96
28                XGB         3   Treino  0.0244  0.16  0.1111  1.00
29                XGB         3    Teste  0.0902  0.30  0.1931  0.96
30                XGB         4   Treino  0.3629  0.60  0.3244  0.94
31                XGB         4    Teste  1.1050  1.05  0.8444  0.54
32                XGB         5   Treino  0.0779  0.28  0.1593  0.99
33                XGB         5    Teste  0.2341  0.48  0.3431  0.91
```

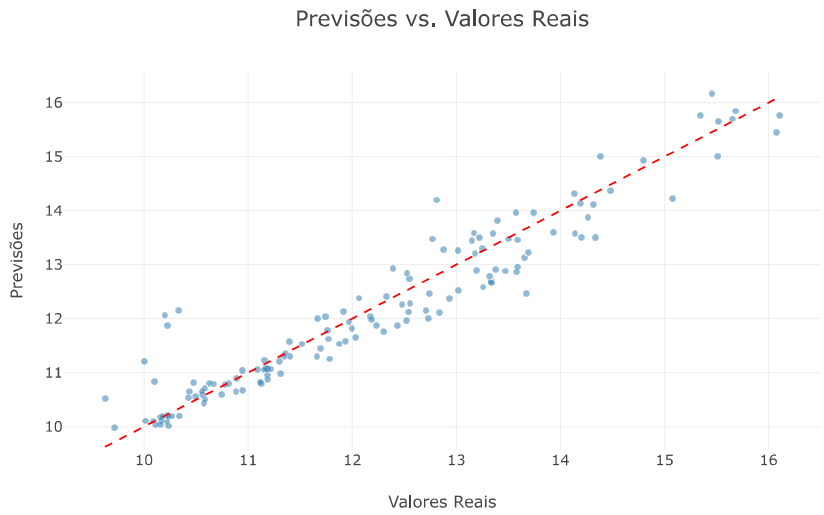
```
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nResultados acumulados:")
print(df_resultados)
```

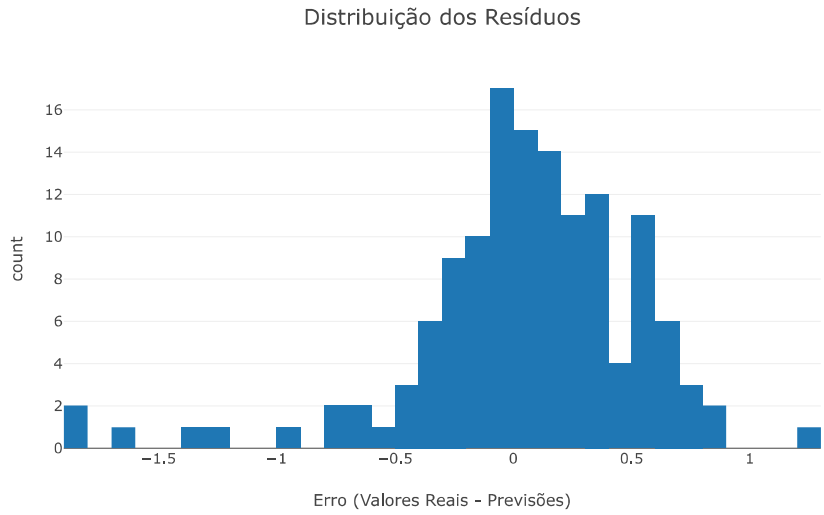
Mostrar saída oculta

```
df_plot = df_pred.loc[df_pred.index >= '2025-03-21'].reset_index()


df_plot = df_plot.melt(id_vars=['DATA'], value_vars=['ELET_CONS_CLT', 'previsão_xgb5'], var_name='Tipo', value_name='Valor')


fig = px.line(df_plot, x='DATA', y='Valor', color='Tipo',
              title='Dados vs Previsão_xgb2024',
              labels={'DATA': 'Data', 'Valor': 'Consumo (kWh/dia)', 'Tipo': 'Conjunto'})

fig.show()
fig.show('notebook_connected')
```




```
df_plot3 = teste4[['ELET_CONS_CLT', 'previsão_xgb5']].reset_index()

#  o gráfico de dispersão interativo
fig = px.scatter(df_plot3, x='ELET_CONS_CLT', y='previsão_xgb5',
                 title="Previsões vs. Valores Reais",
                 labels={'ELET_CONS_CLT': 'Valores Reais', 'previsão_xgb5': 'Previsões'},
                 opacity=0.5)

# Adiciona a linha de referência
fig.add_shape(type="line", x0=df_plot3['ELET_CONS_CLT'].min(), x1=df_plot3['ELET_CONS_CLT'].max(),
              y0=df_plot3['ELET_CONS_CLT'].min(), y1=df_plot3['ELET_CONS_CLT'].max(),
              line=dict(color="red", dash="dash"))

fig.show()
fig.show('notebook_connected')
# Cria o gráfico de resíduos
df_plot3['Resíduos'] = df_plot3['ELET_CONS_CLT'] - df_plot3['previsão_xgb5']
fig_residuos = px.histogram(df_plot3, x='Resíduos', nbins=50,
                            title="Distribuição dos Resíduos",
                            labels={'Resíduos': 'Erro (Valores Reais - Previsões)'})

fig_residuos.show()
fig_residuos.show('notebook_connected')
```





aplicar o treino4 à previsão para o ano 2025

```
teste5 = df_pred.loc[(df_pred.index >= '2025-04-01')]
x_teste5= teste5[input]
y_teste5= teste5[output]

teste5['previsão_xgb6'] = reg4.predict(x_teste5)
df_pred = df_pred.merge(teste5[['previsão_xgb6']], how='left', left_index=True, right_index=True)
```

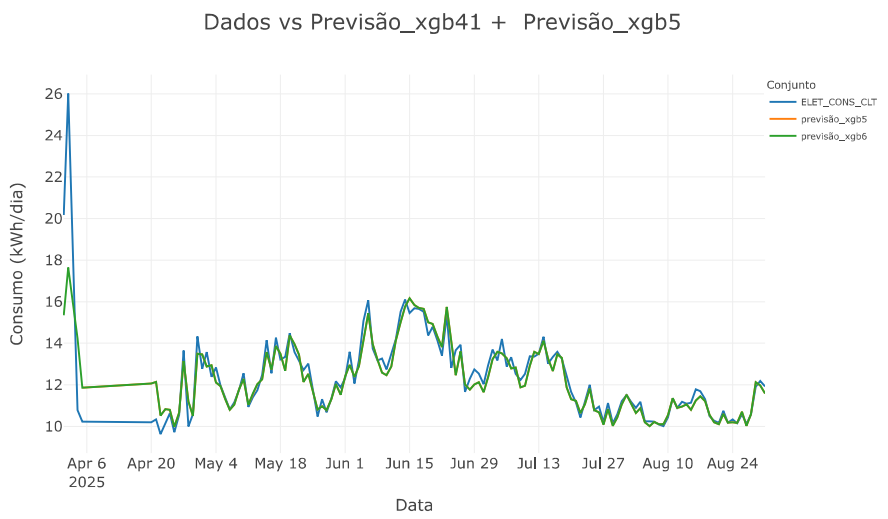
Mostrar saída oculta

```
df_pred.tail()
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	previsão_rf	previsão_tree	erro_relativo_rf	erro_relativo_svr	previsão_xgb1	previsão_xgb2	previsão_xgb3	previsão_xgb
2025-08-27	23.41699	31.9	18.3	13.6	75.88219	89.0	51.5	37.5	23.98021	12914.0	...	10.280448	9.926178	1.658136	1.609637	11.066648	10.090389	10.097376	9.95877
2025-08-28	22.23625	30.2	18.3	11.9	80.14818	100.0	54.7	45.3	22.37211	12826.0	...	10.830714	10.120344	2.514322	0.315668	11.039792	10.437948	10.516932	9.91510
2025-08-29	22.64507	34.0	17.5	16.5	79.59112	91.5	54.4	37.1	22.82476	12985.0	...	10.639877	10.120344	-10.685673	-4.427316	11.288419	11.749044	12.104768	10.53827
2025-08-30	24.65698	34.5	19.7	14.8	78.81356	92.0	56.4	35.6	25.36257	12416.0	...	12.325657	12.483464	1.158541	-0.084308	12.505721	11.900819	12.119337	12.06102
2025-08-31	22.51888	30.1	18.7	11.4	83.64605	98.5	59.9	38.6	22.13280	12372.0	...	11.410585	10.797468	-4.358417	-4.680411	12.032969	11.876614	11.801851	11.27329

5 rows × 59 columns

```
df_plot = df_pred.loc[df_pred.index >= '2025-04-01'].reset_index()

# Criando um DataFrame para diferenciar os conjuntos
df_plot = df_plot.melt(id_vars=['DATA'], value_vars=['ELET_CONS_CLT', 'previsão_xgb5',  'previsão_xgb6'], var_name='Tipo', value_name='Valor')

# Criando o gráfico interativo
fig = px.line(df_plot, x='DATA', y='Valor', color='Tipo',
              title='Dados vs Previsão_xgb41 +  Previsão_xgb5',
              labels={'DATA': 'Data', 'Valor': 'Consumo (kWh/dia)', 'Tipo': 'Conjunto'})

fig.show()
fig.show('notebook_connected')
```



Comece a programar ou gerar com a IA.

```
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nResultados acumulados:")
print(df_resultados)
```

```
df_plot3 = teste5[['ELET_CONS_CLT', 'previsão_xgb5']].reset_index()

#  o gráfico de dispersão interativo
fig = px.scatter(df_plot3, x='ELET_CONS_CLT', y='previsão_xgb5',
                 title="Previsões vs. Valores Reais",
                 labels={'ELET_CONS_CLT': 'Valores Reais', 'previsão_xgb5': 'Previsões'},
                 opacity=0.5)

# Adiciona a linha de referência
fig.add_shape(type="line", x0=df_plot3['ELET_CONS_CLT'].min(), x1=df_plot3['ELET_CONS_CLT'].max(),
```

```
            y0=df_plot3['ELET_CONS_CLT'].min(), y1=df_plot3['ELET_CONS_CLT'].max(),
            line=dict(color="red", dash="dash"))

fig.show()
fig.show('notebook_connected')
# Cria o gráfico de resíduos
df_plot3['Resíduos'] = df_plot3['ELET_CONS_CLT'] - df_plot3['previsão_xgb5']
fig_residuos = px.histogram(df_plot3, x='Resíduos', nbins=50,
                            title="Distribuição dos Resíduos",
                            labels={'Resíduos': 'Erro (Valores Reais - Previsões)'})

fig_residuos.show()
fig_residuos.show('notebook_connected')
```

outro treino para comparar 41 com o 5

```
treino41 = df_pred.loc[(df_pred.index > '2024-04-01') & (df_pred.index <= '2024-10-31')]
teste41 = df_pred.loc[(df_pred.index >= '2025-05-01') ]




#treino41 = create_features(treino41)
#teste41 = create_features(teste41)

# em função do nivel de correlação, foram escolhidos os inputos
input = ['ELET_CONS_CLT_lag1','semana_ano','ano','dia_ano','T_mean','CLTs'
           ]


output = 'ELET_CONS_CLT'

x_treino41 = treino41[input]
y_treino41 = treino41[output]

x_teste41= teste41[input]
y_teste41= teste41[output]
```

```
reg41 = xgb.XGBRegressor(base_score=0.01, booster='gbtree',
                         n_estimators=100000,
                        early_stopping_rounds=8000,
                        objective='reg:squarederror',
                          reg_lambda=5,
                          reg_alpha=0.5,
                        max_depth=1,
                          min_child_weight=3,
                          subsample=0.95,
                          colsample_bytree=0.85,
                          gamma=0.05,

                        learning_rate=0.05)

reg41.fit(x_treino41, y_treino41,
        eval_set=[(x_treino41, y_treino41), (x_teste41, y_teste41)],
        verbose=1000)
```

```
fi = pd.DataFrame(data=reg41.feature_importances_,
                  index=reg41.feature_names_in_,
                  columns=['importance'])

fi = fi.sort_values('importance')

# o gráfico de barras horizontal interativo
fig = px.bar(fi, x='importance', y=fi.index, orientation='h',
             title='Feature Importance')
fig.update_layout(
    yaxis_title_standoff=30  # Aumente o valor para afastar mais
)
fig.update_layout(
    margin=dict(l=200, r=200, t=50, b=50)  # Aumenta a margem direita (r) para mover o gráfico
)
fig.show()
#fig.show('notebook_connected')
```

```
df1_filtrado1.tail()
```

```
teste41['previsão_xgb41'] = reg4.predict(x_teste41)
df1_filtrado1 = df1_filtrado1.merge(teste41[['previsão_xgb41']], how='left', left_index=True, right_index=True)
```

```
df1_filtrado1.tail()
```

```
df_plot = df1_filtrado1.loc[df1_filtrado1.index >= '2025-04-01'].reset_index()

# Criando um DataFrame para diferenciar os conjuntos
df_plot = df_plot.melt(id_vars=['DATA'], value_vars=['ELET_CONS_CLT', 'previsão_xgb41'], var_name='Tipo', value_name='Valor')

# Criando o gráfico interativo
fig = px.line(df_plot, x='DATA', y='Valor', color='Tipo',
              title='Dados vs Previsão_xgb41',
              labels={'DATA': 'Data', 'Valor': 'Consumo (kWh/dia)', 'Tipo': 'Conjunto'})

fig.show()
fig.show('notebook_connected')
```

```
Comece a programar ou gerar com a IA.
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score


mse = mean_squared_error(teste41['ELET_CONS_CLT'], teste41['previsão_xgb41'])
rmse = np.sqrt(mse)
mae = mean_absolute_error(teste41['ELET_CONS_CLT'], teste41['previsão_xgb41'])
r2 = r2_score(teste41['ELET_CONS_CLT'], teste41['previsão_xgb41'])

# Imprimir as métricas
print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"MAE: {mae:.4f}")
print(f"R-squared: {r2:.4f}")
```

```
df_plot3 = teste41[['ELET_CONS_CLT', 'previsão_xgb41']].reset_index()

#  o gráfico de dispersão interativo
fig = px.scatter(df_plot3, x='ELET_CONS_CLT', y='previsão_xgb41',
                 title="Previsões vs. Valores Reais",
                 labels={'ELET_CONS_CLT': 'Valores Reais', 'previsão_xgb41': 'Previsões'},
                 opacity=0.5)

# Adiciona a linha de referência
fig.add_shape(type="line", x0=df_plot3['ELET_CONS_CLT'].min(), x1=df_plot3['ELET_CONS_CLT'].max(),
```

```
            y0=df_plot3['ELET_CONS_CLT'].min(), y1=df_plot3['ELET_CONS_CLT'].max(),
            line=dict(color="red", dash="dash"))

fig.show()
fig.show('notebook_connected')
# Cria o gráfico de resíduos
df_plot3['Resíduos'] = df_plot3['ELET_CONS_CLT'] - df_plot3['previsão_xgb41']
fig_residuos = px.histogram(df_plot3, x='Resíduos', nbins=50,
                            title="Distribuição dos Resíduos",
                            labels={'Resíduos': 'Erro (Valores Reais - Previsões)'})

fig_residuos.show()
fig_residuos.show('notebook_connected')
```

Final

```
teste4 = df1_filtrado3.copy()

#treino4 = create_features(treino4)
#teste4 = create_features(teste4)

# em função do nivel de correlação, foram escolhidos os inputos
input = ['Q_OCP', 'OCP', 'AD', 'CR', 'CLTs', 'T_mean', 'Calor_idx', 'RH_mean',
       'dia_semana', 'trimestre', 'mes', 'ano',
       'dia_ano', 'dia_mes', 'semana_ano'
           ]

output = ['ELET_CONS_CLT']


x_teste4= teste4[input]
y_teste4= teste4[output]
```

```
x_teste4.columns
```

```
Index(['Q_OCP', 'OCP', 'AD', 'CR', 'CLTs', 'T_mean', 'Calor_idx', 'RH_mean',
       'dia_semana', 'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes',
       'semana_ano'],
      dtype='object')
```

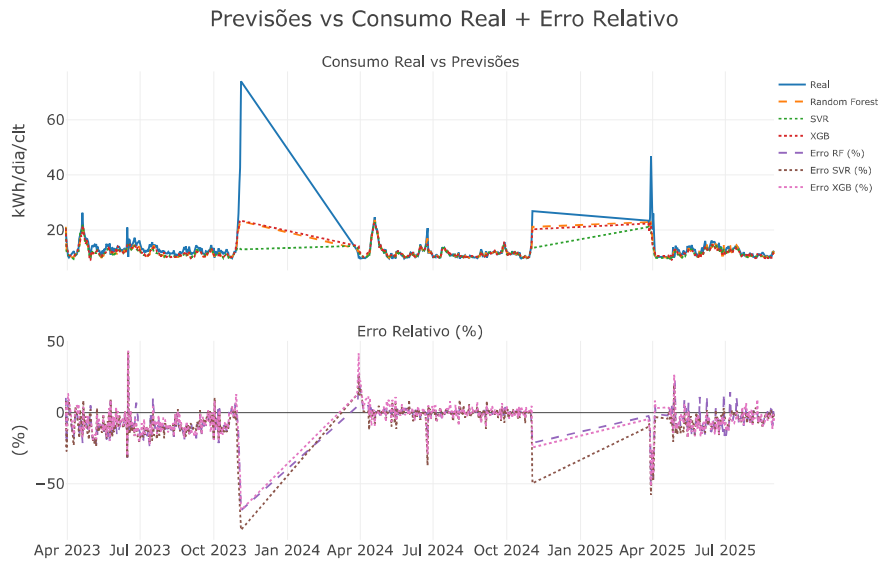
```
teste4['previsão_xgb_f'] = reg4.predict(x_teste4)
df_pred = df_pred.merge(teste4[['previsão_xgb_f']], how='left', left_index=True, right_index=True)
```

```
df_pred.tail()
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	ELET_CONS_CLT_rolling_median	ELET_CONS_CLT_rolling_std	ELET_CONS_CLT_error_median	ELET_CONS_CLT_pct_change	previsão_s
2025-08-27	23.41699	31.9	18.3	13.6	75.88219	89.0	51.5	37.5	23.98021	12914.0	...	10.176106	0.248000	-0.006225	-0.044634	10.2768
2025-08-28	22.23625	30.2	18.3	11.9	80.14818	100.0	54.7	45.3	22.37211	12826.0	...	10.341620	0.250861	0.021607	0.044727	10.6008
2025-08-29	22.64507	34.0	17.5	16.5	79.59112	91.5	54.4	37.1	22.82476	12985.0	...	10.341620	0.631311	0.151932	0.127568	11.3870
2025-08-30	24.65698	34.5	19.7	14.8	78.81356	92.0	56.4	35.6	25.36257	12416.0	...	10.565074	0.851388	0.153281	0.022803	12.1721
2025-08-31	22.51888	30.1	18.7	11.4	83.64605	98.5	59.9	38.6	22.13280	12372.0	...	10.585227	0.907476	0.127096	-0.020840	11.3695

5 rows × 54 columns

```
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import pandas as pd

# Calcular erro relativo (%)
df_pred['erro_relativo_xgb_f'] = ((df_pred['previsão_xgb_f'] - df_pred['ELET_CONS_CLT']) / df_pred['ELET_CONS_CLT']) * 100
#df_pred['erro_relativo_svr'] = ((df_pred['previsão_svr'] - df_pred['ELET_CONS_CLT']) / df_pred['ELET_CONS_CLT']) * 100

# Selecionar os dados
df_plot = df_pred[['ELET_CONS_CLT', 'previsão_rf', 'previsão_svr', 'erro_relativo_rf', 'erro_relativo_svr','previsão_xgb_f','erro_relativo_xgb_f']].dropna().copy()

# Resetar índice e renomear para 'Data'
df_plot = df_plot.reset_index().rename(columns={df_plot.index.name or 'index': 'Data'})

# Criar subplots
fig = make_subplots(rows=2, cols=1, shared_xaxes=True, vertical_spacing=0.15,
                    subplot_titles=("Consumo Real vs Previsões", "Erro Relativo (%)"))

# Subplot 1: Consumo real e previsões
fig.add_trace(go.Scatter(x=df_plot['Data'], y=df_plot['ELET_CONS_CLT'],
                         mode='lines', name='Real', line=dict( dash='solid')), row=1, col=1)

fig.add_trace(go.Scatter(x=df_plot['Data'], y=df_plot['previsão_rf'],
                         mode='lines', name='Random Forest', line=dict( dash='dash')), row=1, col=1)

fig.add_trace(go.Scatter(x=df_plot['Data'], y=df_plot['previsão_svr'],
                         mode='lines', name='SVR', line=dict( dash='dot')), row=1, col=1)

fig.add_trace(go.Scatter(x=df_plot['Data'], y=df_plot['previsão_xgb_f'],
                         mode='lines', name='XGB', line=dict( dash='dot')), row=1, col=1)

# Subplot 2: Erro relativo
fig.add_trace(go.Scatter(x=df_plot['Data'], y=df_plot['erro_relativo_rf'],
                         mode='lines', name='Erro RF (%)', line=dict( dash='dash')), row=2, col=1)

fig.add_trace(go.Scatter(x=df_plot['Data'], y=df_plot['erro_relativo_svr'],
                         mode='lines', name='Erro SVR (%)', line=dict( dash='dot')), row=2, col=1)

fig.add_trace(go.Scatter(x=df_plot['Data'], y=df_plot['erro_relativo_xgb_f'],
                         mode='lines', name='Erro XGB (%)', line=dict( dash='dot')), row=2, col=1)

# Layout final
fig.update_layout(height=700, width=1000,
                  title_text="Previsões vs Consumo Real + Erro Relativo",
                  yaxis=dict(title='kWh/dia/clt'),
                  yaxis2=dict(title='(%)'),
                  hovermode='x unified')

fig.show()
```




```
df_pred.columns
```

```
dados_2025 = dados_2025.merge(df_pred['OCP[%]'], left_index=True, right_index=True, how='left')
```

```
# Certifique-se de que ambos os índices estão no tipo datetime
dados_2025.index = pd.to_datetime(dados_2025.index)
df_pred.index = pd.to_datetime(df_pred.index)

# Selecionar apenas as colunas desejadas de df_pred
df_pred_selecionado = df_pred[['previsão_xgb_f', 'erro_relativo_xgb_f' ]]

# Realizar o merge com base no índice
dados_2025 = dados_2025.merge(df_pred_selecionado, left_index=True, right_index=True, how='left')
dados_2025
```

DATA	ELET_CONS_CLT	mes	med2024	erro_relativo	previsão_rf	previsão_svr	erro_relativo_rf	erro_relativo_svr	anomalia_negativa	previsão_xgb_f	erro_relativo_xgb_f
2025-04-01	20.167279	4	12.791762	57.658333	15.017605	14.971816	-25.534801	-25.761843	True	14.777119	-26.727258
2025-04-02	26.015000	4	12.791762	103.373071	13.820930	13.731936	-46.873228	-47.215314	True	14.777119	-43.197699
2025-04-04	10.777647	4	12.791762	15.745409	11.327024	9.976873	5.097373	-7.429948	False	11.680786	8.379743
2025-04-05	10.224138	4	12.791762	20.072484	10.120539	9.896231	-1.013282	-3.207189	False	10.548176	3.169342
2025-04-06	9.313346	4	12.791762	27.192627	NaN	NaN	NaN	NaN	False	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...
2025-08-27	10.112764	8	10.264049	1.473931	10.359583	10.276879	2.440664	1.622845	False	9.953657	-1.573330
2025-08-28	10.565074	8	10.264049	2.932808	10.780303	10.600844	2.037177	0.338569	False	9.920411	-6.101832
2025-08-29	11.912844	8	10.264049	16.063785	10.840260	11.387050	-9.003590	-4.413674	False	10.535409	-11.562605
2025-08-30	12.184495	8	10.264049	18.710406	12.321556	12.172142	1.124885	-0.101380	False	12.037998	-1.202318
2025-08-31	11.930569	8	10.264049	16.236474	11.334702	11.369584	-4.994458	-4.702080	False	11.249270	-5.710528

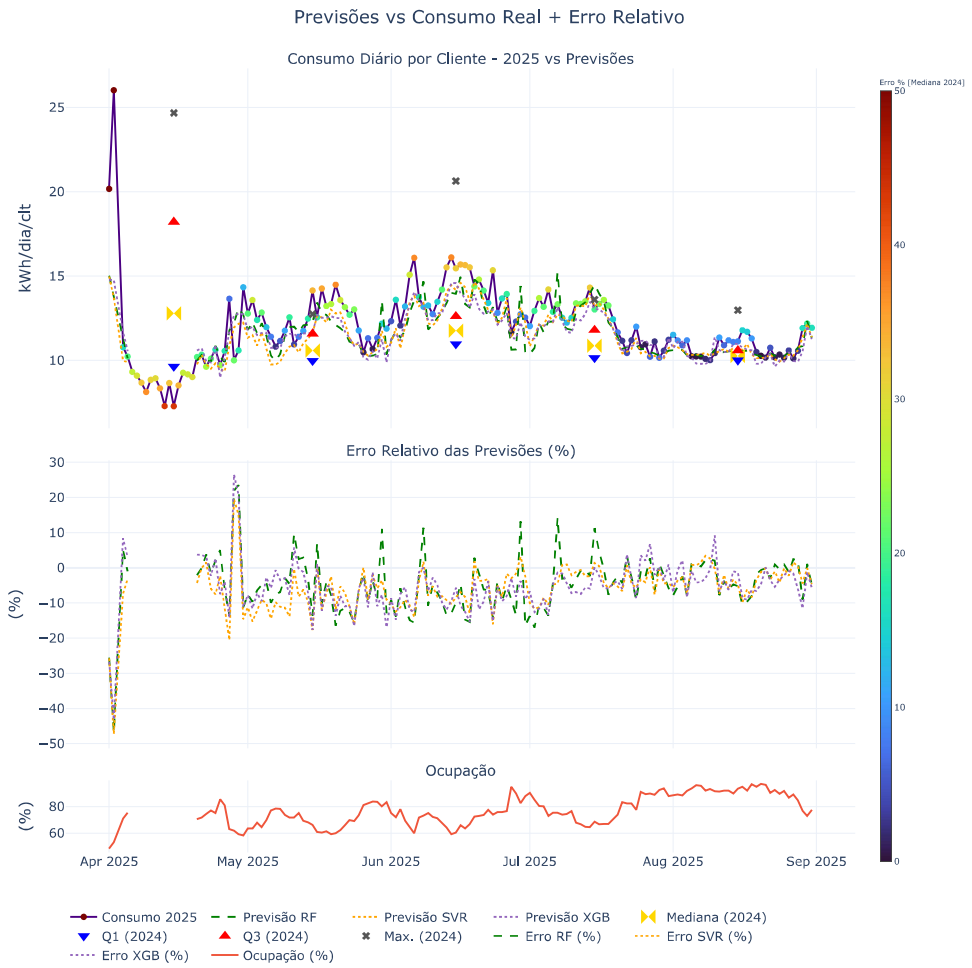
152 rows × 11 columns

```
# Remove todas as linhas que contêm qualquer valor NaN
#dados_2025 = dados_2025_merged.dropna()
dados_2025
```

```
dados_2025.columns
```

```
from plotly.subplots import make_subplots
import plotly.graph_objects as go

# Criar subplots: 2 linhas, 1 coluna
fig = make_subplots(
    rows=3, cols=1,
    shared_xaxes=True,
    vertical_spacing=0.041,
    row_heights=[0.5, 0.4, 0.1],
    subplot_titles=("Consumo Diário por Cliente - 2025 vs Previsões", "Erro Relativo das Previsões (%)", "Ocupação")
)

# ——————————————————————————————————————
# Subplot 1: Consumo real + previsões
fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['ELET_CONS_CLT'],
    mode='lines+markers',
    name='Consumo 2025',
    line=dict(color='indigo'),
    marker=dict(
        color=dados_2025['erro_relativo'],
        colorscale='Turbo',
        cmin=0,
        cmax=50,
        colorbar=dict(title='Erro % [Mediana 2024]', titlefont=dict(size=8), thickness=12, tickfont=dict(size=10)),
        size=7
    )
), row=1, col=1)

fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['previsão_rf'],
    mode='lines',
    name='Previsão RF',
    line=dict(dash='dash', color='green')
), row=1, col=1)

fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['previsão_svr'],
    mode='lines',
    name='Previsão SVR',
```

```
    line=dict(dash='dot', color='orange')
), row=1, col=1)

fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['previsão_xgb_f'],
    mode='lines',
    name='Previsão XGB',
    line=dict(dash='dot', color='#9467BD')
), row=1, col=1)



# Estatísticas mensais de 2024
fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['50%'],
    mode='markers',
    name='Mediana (2024)',
    marker=dict(color='gold', symbol='bowtie', size=16)
), row=1, col=1)

fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['25%'],
    mode='markers',
    name='Q1 (2024)',
    marker=dict(color='blue', symbol='triangle-down', size=12)
), row=1, col=1)

fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['75%'],
    mode='markers',
    name='Q3 (2024)',
    marker=dict(color='red', symbol='triangle-up', size=12)
), row=1, col=1)

fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['max'],
    mode='markers',
    name='Max. (2024)',
    marker=dict(color='#565656',symbol='x', size=8)
), row=1, col=1)

# ──────────────────────────────────────────
# Subplot 2: Erro relativo
fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['erro_relativo_rf'],
    mode='lines',
    name='Erro RF (%)',
    line=dict(dash='dash', color='green')
), row=2, col=1)

fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['erro_relativo_svr'],
    mode='lines',
    name='Erro SVR (%)',
    line=dict(dash='dot', color='orange')
), row=2, col=1)

fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['erro_relativo_xgb_f'],
    mode='lines',
    name='Erro XGB (%)',
    line=dict(dash='dot', color='#9467BD')
), row=2, col=1)

# Subplot 3: OCP
fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['OCP[%]'],
    mode='lines',
    name='Ocupação (%)',
    #line=dict(dash='dash', color='green')
), row=3, col=1)
# ──────────────────────────────────────────
# Layout final
fig.update_layout(
    height=1100,
    width=1100,
    title_text="Previsões vs Consumo Real + Erro Relativo",
    template='plotly_white',
    font=dict(size=14),
    title_x=0.5,
    legend=dict(orientation='h', y=-0.05),
    hovermode='x unified'
)

fig.update_yaxes(title_text='kWh/dia/clt', row=1, col=1)
fig.update_yaxes(title_text='(%)', row=2, col=1)
fig.update_yaxes(title_text='(%)', row=3, col=1)

fig.show()
```

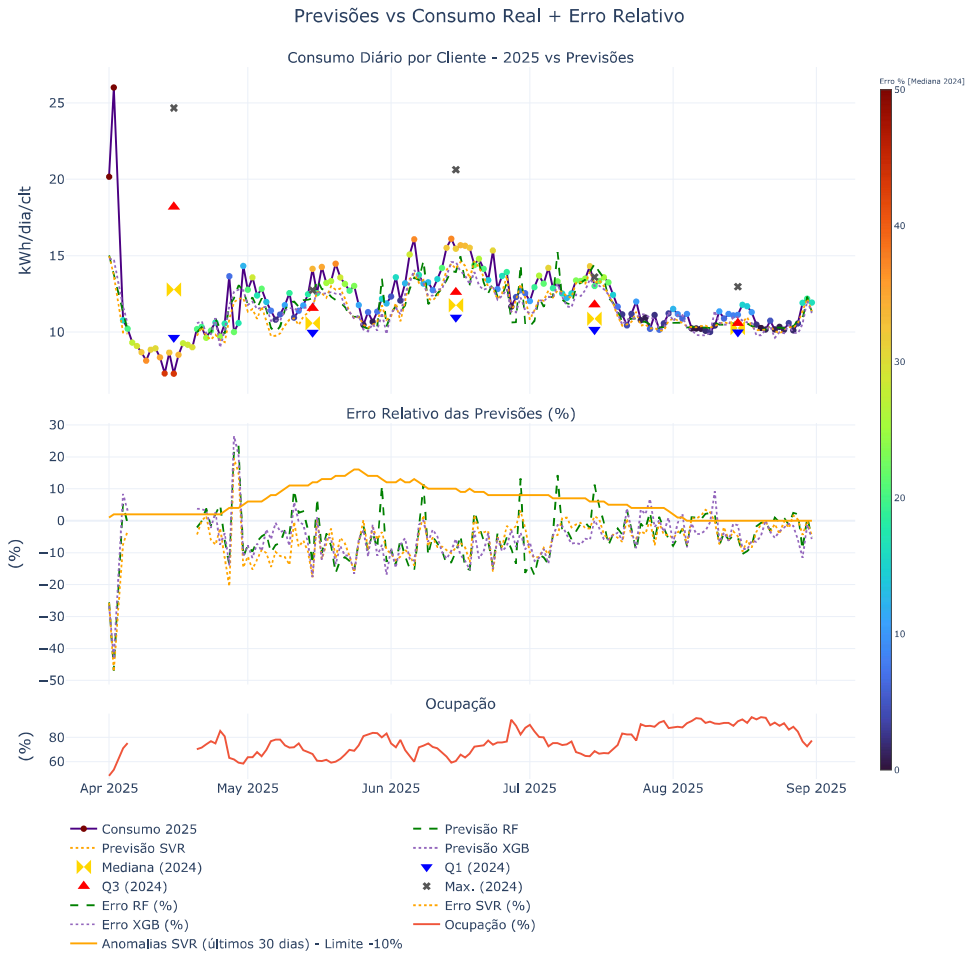



```
df_pred.columns
```

Comece a programar ou gerar com a IA.

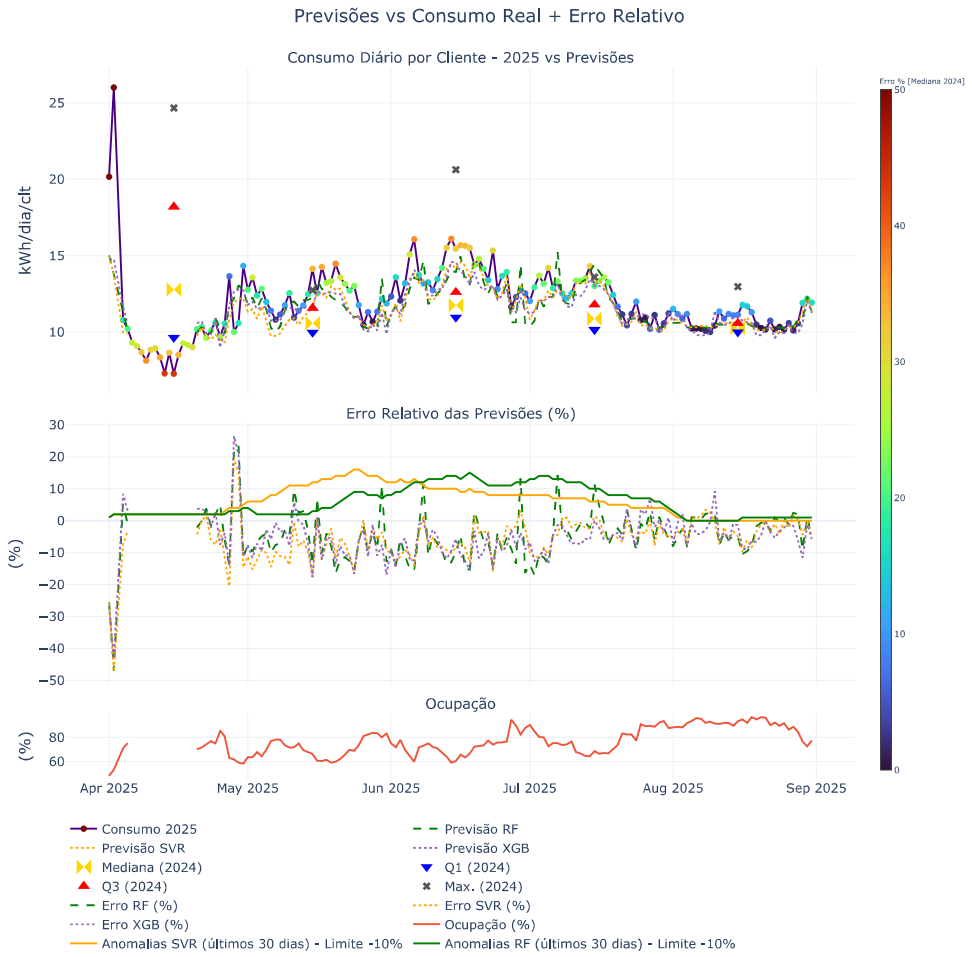
```
import pandas as pd

# Definir o limite negativo de erro relativo (%)
limite_negativo = -10  # pode alterar para -15, etc.

# Selecionar a coluna de erro relativo do modelo a avaliar
coluna_erro = 'erro_relativo_svr'  # ou 'erro_relativo_svr', 'erro_relativo_xgb_f'

# Criar coluna booleana: True apenas quando o erro relativo é menor que o limite negativo
dados_2025['anomalia_negativa'] = dados_2025[coluna_erro] < limite_negativo

# Contar anomalias por janela móvel de 30 dias
anomalias_neg_30d = (
    dados_2025['anomalia_negativa']
    .rolling(window=30, min_periods=1)
    .sum()
)

# Número total de anomalias negativas no período completo
total_anomalias_neg = dados_2025['anomalia_negativa'].sum()

print(f"Limite de erro relativo negativo: {limite_negativo}%")
print(f"Número total de anomalias negativas detetadas: {total_anomalias_neg}")
print("\nAnomalias negativas detetadas em cada janela de 30 dias:")
print(anomalias_neg_30d)

# (Opcional) Adicionar ao gráfico no subplot 2
fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=anomalias_neg_30d,
    mode='lines',
    name=f'Anomalias SVR (últimos 30 dias) - Limite {limite_negativo}%',
    line=dict(color='orange', width=2)
), row=2, col=1)

fig.show()
```

```
Limite de erro relativo negativo: -10%
Número total de anomalias negativas detetadas: 30

Anomalias negativas detetadas em cada janela de 30 dias:
DATA
2025-04-01    1.0
2025-04-02    2.0
2025-04-04    2.0
2025-04-05    2.0
2025-04-06    2.0
              ...
2025-08-27    0.0
2025-08-28    0.0
2025-08-29    0.0
2025-08-30    0.0
2025-08-31    0.0
Name: anomalia_negativa, Length: 152, dtype: float64
```



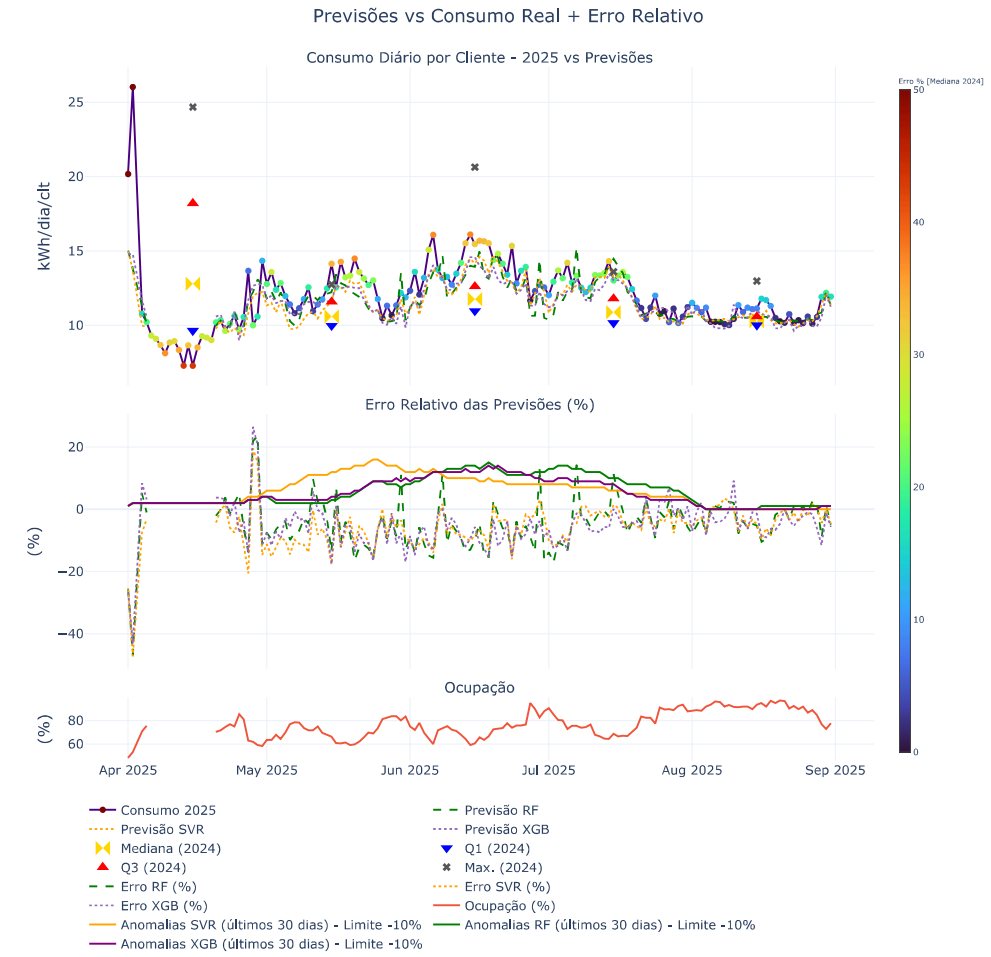

```
import pandas as pd

# Definir o limite negativo de erro relativo (%)
limite_negativo = -10  # pode alterar para -15, etc.

# Selecionar a coluna de erro relativo do modelo a avaliar
coluna_erro = 'erro_relativo_rf'  # ou 'erro_relativo_svr', 'erro_relativo_xgb_f'

# Criar coluna booleana: True apenas quando o erro relativo é menor que o limite negativo
dados_2025['anomalia_negativa'] = dados_2025[coluna_erro] < limite_negativo

# Contar anomalias por janela móvel de 30 dias
anomalias_neg_30d = (
    dados_2025['anomalia_negativa']
    .rolling(window=30, min_periods=1)
    .sum()
)

# Número total de anomalias negativas no período completo
total_anomalias_neg = dados_2025['anomalia_negativa'].sum()

print(f"Limite de erro relativo negativo: {limite_negativo}%")
print(f"Número total de anomalias negativas detetadas: {total_anomalias_neg}")
print("\nAnomalias negativas detetadas em cada janela de 30 dias:")
print(anomalias_neg_30d)

# (Opcional) Adicionar ao gráfico no subplot 2
fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=anomalias_neg_30d,
    mode='lines',
    name=f'Anomalias RF (últimos 30 dias) - Limite {limite_negativo}%',
    line=dict(color='green', width=2)
), row=2, col=1)

fig.show()
```

```
Limite de erro relativo negativo: -10%
Número total de anomalias negativas detetadas: 30

Anomalias negativas detetadas em cada janela de 30 dias:
DATA
2025-04-01    1.0
2025-04-02    2.0
2025-04-04    2.0
2025-04-05    2.0
2025-04-06    2.0
              ...
2025-08-27    1.0
2025-08-28    1.0
2025-08-29    1.0
2025-08-30    1.0
2025-08-31    1.0
Name: anomalia_negativa, Length: 152, dtype: float64
```




```
import pandas as pd

# Definir o limite negativo de erro relativo (%)
limite_negativo = -10  # pode alterar para -15, etc.

# Selecionar a coluna de erro relativo do modelo a avaliar
coluna_erro = 'erro_relativo_xgb_f'  # ou 'erro_relativo_svr', 'erro_relativo_xgb_f'

# Criar coluna booleana: True apenas quando o erro relativo é menor que o limite negativo
dados_2025['anomalia_negativa'] = dados_2025[coluna_erro] < limite_negativo

# Contar anomalias por janela móvel de 30 dias
anomalias_neg_30d = (
    dados_2025['anomalia_negativa']
    .rolling(window=30, min_periods=1)
    .sum()
)

# Número total de anomalias negativas no período completo
total_anomalias_neg = dados_2025['anomalia_negativa'].sum()

print(f"Limite de erro relativo negativo: {limite_negativo}%")
print(f"Número total de anomalias negativas detetadas: {total_anomalias_neg}")
print("\nAnomalias negativas detetadas em cada janela de 30 dias:")
print(anomalias_neg_30d)

# (Opcional) Adicionar ao gráfico no subplot 2
fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=anomalias_neg_30d,
    mode='lines',
    name=f'Anomalias XGB (últimos 30 dias) - Limite {limite_negativo}%',
    line=dict(color='purple', width=2)
), row=2, col=1)

fig.show()
```

```
Limite de erro relativo negativo: -10%
Número total de anomalias negativas detetadas: 27

Anomalias negativas detetadas em cada janela de 30 dias:
DATA
2025-04-01    1.0
2025-04-02    2.0
2025-04-04    2.0
2025-04-05    2.0
2025-04-06    2.0
              ...
2025-08-27    0.0
2025-08-28    0.0
2025-08-29    1.0
2025-08-30    1.0
2025-08-31    1.0
Name: anomalia_negativa, Length: 152, dtype: float64
```



GRU

```
Comece a programar ou gerar com a IA.
```

O GRU é um tipo de rede neural recorrente (RNN) que é mais eficiente do que LSTM para aprender padrões temporais em séries de tempo. Portas de atualização e esquecimento → Diferente do LSTM, o GRU tem apenas duas portas que controlam a informação que entra e sai da memória, tornando-o mais simples e rápido.

aplicar GRU

1ª iteração

```
# 🔍 Selecionar variáveis de entrada
features = ['T_mean', 'T_max', 'T_min', 'RH_mean', 'RH_max', 'RH_min', 'Calor_idx',
         'Q_OCP', 'OCP', 'AD', 'CR', 'CLTs',

         'dia_semana', 'trimestre', 'mes', 'dia_ano', 'dia_mes', 'semana_ano',
            'ELET_CONS_CLT_lag1', 'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean', 'ELET_CONS_CLT_pct_change',

            'ELET_CONS_CLT'] # Aqui, o script seleciona um conjunto de variáveis de entrada — e a última coluna escolhida é ELET_CONS_CLT, o valor que se deseja prever.
df1_gru = df1_filtrado3[features]
```

```
df1_gru = df1_gru.replace([np.inf, -np.inf], np.nan)

# Opção 1: Preencher com a média de cada coluna
#df1_gru = df1_gru.fillna(df1_gru.mean())

# Opção 2: Remover linhas com NaN (se não forem muitas)
df1_gru = df1_gru.dropna()
```

```
df1_gru = df1_gru.astype('float64')
```

```
import numpy as np
import pandas as pd

# 1. Garante que é um DataFrame
df1_gru = pd.DataFrame(df1_gru)

# 2. Substitui valores infinitos por NaN
df1_gru.replace([np.inf, -np.inf], np.nan, inplace=True)

# 3. Verifica se ainda há NaNs
print("Valores nulos por coluna:")
print(df1_gru.isnull().sum())
```

```
# 4. Preenche os NaNs com a média de cada coluna
df1_gru.fillna(df1_gru.mean(), inplace=True)

# 5. Converte tudo para float64 (se ainda não estiver)
df1_gru = df1_gru.astype('float64')

# 6. Aplica o MinMaxScaler
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
df1_normalizados = scaler.fit_transform(df1_gru)
```

```
Valores nulos por coluna:
T_mean                        0
T_max                         0
T_min                         0
RH_mean                       0
RH_max                        0
RH_min                        0
Calor_idx                     0
Q_OCP                         0
OCP                           0
AD                            0
CR                            0
CLTs                          0
dia_semana                    0
trimestre                     0
mes                           0
dia_ano                       0
dia_mes                       0
semana_ano                    0
ELET_CONS_CLT_lag1            0
ELET_CONS_CLT_lag7            0
ELET_CONS_CLT_rolling_mean    0
ELET_CONS_CLT_pct_change      0
ELET_CONS_CLT                 0
dtype: int64
```

```
from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, GRU, Dense, Dropout
from tensorflow.keras.optimizers import Adam
import tensorflow as tf



# 🔄 Criar sequências temporais para treinamento do GRU
def criar_sequencias(df1_normalizados, n_passo):
    X, y = [], []
    for i in range(len(df1_normalizados) - n_passo):
        X.append(df1_normalizados[i:i+n_passo, :-1])  # Variáveis climáticas e temporais
        y.append(df1_normalizados[i+n_passo, -1])  # Consumo elétrico
    return np.array(X), np.array(y)

# Definir tamanho da janela de tempo
n_passo = 7  # Exemplo: usar 30 dias para prever o próximo dia
X_train, y_train = criar_sequencias(df1_normalizados, n_passo)

# 🎯 Criar modelo GRU otimizado
model1_gru = Sequential([
    GRU(100, activation='tanh', input_shape=(X_train.shape[1], X_train.shape[2])),
    Dropout(0.2),  # Reduz overfitting
    Dense(1)  # Camada de saída
])

# ⏳ Compilar modelo com otimizador Adam
model1_gru.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), loss='mse')

# 🚀 Treinar modelo
history = model1_gru.fit(X_train, y_train, epochs=100, batch_size=16, validation_split=0.2)

# 🔮 Fazer previsões


previsoes1 = model1_gru.predict(X_train)

# 🚀 Reverter normalização para obter valores reais
previsoes1_reais = scaler.inverse_transform(np.concatenate((X_train[:, -1, :], previsoes1), axis=1))[:, -1]
# Criar DataFrame com previsões e valores reais
previsoes1_df = pd.DataFrame({
    'Previsto': previsoes1_reais,
    'Real': df1_gru['ELET_CONS_CLT'].iloc[n_passo:].values
}, index=df1_gru.index[n_passo:])

# 📊 Visualizar previsões
import plotly.graph_objects as go

# Criar figura
fig = go.Figure()

# Adicionar traço Real
fig.add_trace(go.Scatter(
    x=df1_gru.index,
    y=df1_gru['ELET_CONS_CLT'],
    mode='lines',
    name='Real',
    line=dict(color='blue')
))

# Adicionar traço Previsto
fig.add_trace(go.Scatter(
    x=previsoes1_df.index,
    y=previsoes1_df['Previsto'],
    mode='lines',
    name='Previsto (GRU)',
    line=dict(color='red', dash='dash')
))

# Personalizar layout
fig.update_layout(
    title='Comparação: Consumo Real vs. Previsto (GRU)',
    xaxis_title='Data',
    yaxis_title='Consumo Elétrico',
    legend=dict(x=0, y=1),
    font=dict(size=14),
    title_x=0.5,
    width=1000,
    height=500
)

# Exibir gráfico
fig.show()
```

Mostrar saída oculta

Validar

```
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Lista global para acumular resultados
resultados_acumulados = []

# Função para avaliar e guardar resultados de treino e teste (GRU)
def avaliar_gru(modelo, X_train, y_train, X_test, y_test, scaler, df_index, n_passo, modelo_nome, iteracao):
    # ========================
    # Avaliação no TREINO
    # ========================
    y_pred_train = modelo.predict(X_train).flatten()

    # Reverter normalização (concatena features + alvo)
    y_pred_train_reais = scaler.inverse_transform(
        np.concatenate((X_train[:, -1, :], y_pred_train.reshape(-1,1)), axis=1)
    )[:, -1]

    y_train_reais = scaler.inverse_transform(
        np.concatenate((X_train[:, -1, :], y_train.reshape(-1,1)), axis=1)
    )[:, -1]

    mse_train = mean_squared_error(y_train_reais, y_pred_train_reais)
    rmse_train = np.sqrt(mse_train)
    mae_train = mean_absolute_error(y_train_reais, y_pred_train_reais)
    r2_train = r2_score(y_train_reais, y_pred_train_reais)

    print(f'{modelo_nome} (Iteração {iteracao}) - TREINO → MSE: {mse_train:.4f} | RMSE: {rmse_train:.2f} | MAE: {mae_train:.4f} | R²: {r2_train:.2f}')
    resultados_acumulados.append({
        'Modelo': modelo_nome,
        'Iteração': iteracao,
        'Conjunto': 'Treino',
        'MSE': round(mse_train, 4),
        'RMSE': round(rmse_train, 2),
        'MAE': round(mae_train, 4),
        'R²': round(r2_train, 2)
    })

    # ========================
    # Avaliação no TESTE
    # ========================
    y_pred_test = modelo.predict(X_test).flatten()

    y_pred_test_reais = scaler.inverse_transform(
        np.concatenate((X_test[:, -1, :], y_pred_test.reshape(-1,1)), axis=1)
    )[:, -1]

    y_test_reais = scaler.inverse_transform(
        np.concatenate((X_test[:, -1, :], y_test.reshape(-1,1)), axis=1)
    )[:, -1]

    mse_test = mean_squared_error(y_test_reais, y_pred_test_reais)
    rmse_test = np.sqrt(mse_test)
    mae_test = mean_absolute_error(y_test_reais, y_pred_test_reais)
    r2_test = r2_score(y_test_reais, y_pred_test_reais)

    print(f'{modelo_nome} (Iteração {iteracao}) - TESTE  → MSE: {mse_test:.4f} | RMSE: {rmse_test:.2f} | MAE: {mae_test:.4f} | R²: {r2_test:.2f}')
    resultados_acumulados.append({
        'Modelo': modelo_nome,
        'Iteração': iteracao,
        'Conjunto': 'Teste',
        'MSE': round(mse_test, 4),
        'RMSE': round(rmse_test, 2),
        'MAE': round(mae_test, 4),
        'R²': round(r2_test, 2)
    })

    # Retornar DataFrame com previsões alinhadas (opcional)
    df_result = pd.DataFrame({
        'Real': y_test_reais,
        'Previsto': y_pred_test_reais
    }, index=df_index[n_passo: n_passo+len(y_test_reais)])

    return df_result
```

```
# Garantir que o índice é datetime
df1_gru.index = pd.to_datetime(df1_gru.index)

# Selecionar subconjunto
df_previsao = df1_gru.loc['2025-04-01':'2025-08-31']
df_previsao
```

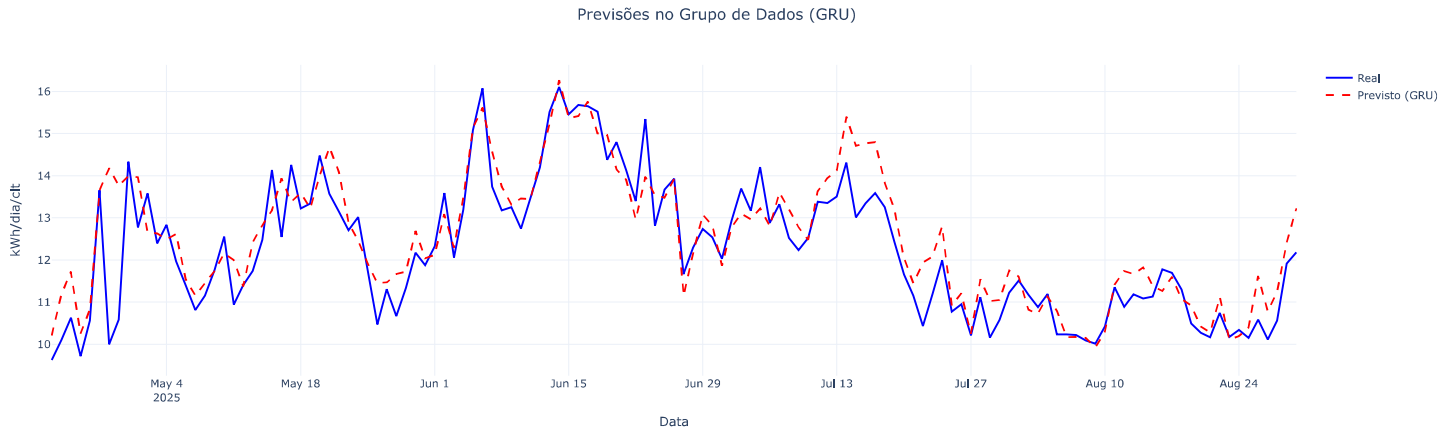
DATA	T_mean	T_max	T_min	RH_mean	RH_max	RH_min	Calor_idx	Q_OCP	OCP	AD	...	trimestre	mes	dia_ano	dia_mes	semana_ano	ELET_CONS_CLT_lag1	ELET_CONS_CLT_lag7	ELET_CONS_CLT_rolling_mean	E
2025-04-01	17.383333	21.299999	14.9	96.116666	100.0	87.500000	15.150681	241.0	0.482	445.0	...	2.0	4.0	91.0	1.0	14.0	46.693820	26.358306	26.742124	
2025-04-02	15.333333	17.700001	12.4	95.716667	100.0	88.500000	15.909849	266.0	0.532	493.0	...	2.0	4.0	92.0	2.0	14.0	20.167279	25.563694	26.806597	
2025-04-04	16.266667	19.900000	13.1	96.699999	100.0	90.099998	14.960232	355.0	0.710	653.0	...	2.0	4.0	94.0	4.0	14.0	26.015000	25.381098	24.720389	
2025-04-05	15.800000	20.700001	12.3	96.871429	100.0	87.800003	15.026587	377.0	0.754	701.0	...	2.0	4.0	95.0	5.0	14.0	10.777647	23.400000	22.838123	
2025-04-20	14.650000	20.799999	11.5	90.950001	100.0	76.400002	19.046753	352.0	0.704	656.0	...	2.0	4.0	110.0	20.0	16.0	10.224138	21.435233	21.233120	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2025-08-27	23.416990	31.900000	18.3	75.882190	89.0	51.500000	23.980210	445.0	0.890	834.0	...	3.0	8.0	239.0	27.0	35.0	10.585227	10.270170	10.325795	
2025-08-28	22.236250	30.200000	18.3	80.148180	100.0	54.700000	22.372110	424.0	0.848	791.0	...	3.0	8.0	240.0	28.0	35.0	10.112764	10.165070	10.382938	
2025-08-29	22.645070	34.000000	17.5	79.591120	91.5	54.400000	22.824760	384.0	0.768	717.0	...	3.0	8.0	241.0	29.0	35.0	10.565074	10.747419	10.549428	
2025-08-30	24.656980	34.500000	19.7	78.813560	92.0	56.400000	25.362570	364.0	0.728	677.0	...	3.0	8.0	242.0	30.0	35.0	11.912844	10.176106	10.836340	
2025-08-31	22.518880	30.100000	18.7	83.646050	98.5	59.900000	22.132800	388.0	0.776	724.0	...	3.0	8.0	243.0	31.0	35.0	12.184495	10.341620	11.063333	

138 rows × 23 columns

```
# Garantir que o índice é datetime
df1_gru.index = pd.to_datetime(df1_gru.index)

# Selecionar subconjunto
df_previsao = df1_gru.loc['2025-04-01':'2025-08-31']


# Normalizar com o mesmo scaler usado no treino
df_previsao_norm = scaler.transform(df_previsao.values)

# Criar sequências para este subconjunto
```

```
X_test, y_test = criar_sequencias(df_previsao_norm, n_passo)

# Fazer previsões com o modelo já treinado
y_pred = model1_gru.predict(X_test)

# Reverter normalização para valores reais
y_pred_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_pred), axis=1)
)[:, -1]

y_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_test.reshape(-1,1)), axis=1)
)[:, -1]

# ============================================================
# 6. Visualizar resultados
# ============================================================

# Gerar índice correspondente ao último dia da janela usada para cada previsão
index_alinhado = df_previsao.index[n_passo - 1 : n_passo - 1 + len(y_pred_reais)]

# Garantir que os valores reais estejam alinhados com esse índice
real_values = df_previsao['ELET_CONS_CLT'].iloc[n_passo - 1 : n_passo - 1 + len(y_pred_reais)].values

# Plot
fig = go.Figure()
fig.add_trace(go.Scatter(
    x=index_alinhado, y=real_values,
    mode='lines', name='Real', line=dict(color='blue')
))
fig.add_trace(go.Scatter(
    x=index_alinhado, y=y_pred_reais,
    mode='lines', name='Previsto (GRU)', line=dict(color='red', dash='dash')
))
fig.update_layout(
    title="Previsões no Grupo de Dados (GRU)",
    xaxis_title="Data", yaxis_title="kWh/dia/clt",
    template="plotly_white", title_x=0.5
)
fig.show()
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2739: UserWarning:

X does not have valid feature names, but MinMaxScaler was fitted with feature names
```




```
# Avaliar modelo
df_resultados = avaliar_gru(
    modelo=model1_gru,
    X_train=X_train, y_train=y_train,   # podes manter os originais
    X_test=X_test, y_test=y_test,       # aqui vai o subconjunto abril-agosto
    scaler=scaler,
    df_index=df_pred.index,             # índice do subconjunto
    n_passo=n_passo,
    modelo_nome="GRU",
    iteracao=1
)
```

Mostrar saída oculta

```
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nResultados acumulados:")
print(df_resultados)
```

```
Resultados acumulados:
  Modelo  Iteração Conjunto     MSE  RMSE     MAE    R²
0    GRU         1   Treino  1.5272  1.24  0.8796  0.92
1    GRU         1    Teste  1.0344  1.02  0.7701  0.58
```

2ª iteração

```
#2ª iteração
```

```
df1_filtrado3.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change'],
      dtype='object')
```

```
# 🔍 Selecionar variáveis de entrada
features = ['T_mean', 'T_max', 'T_min', 'RH_mean', 'RH_max', 'RH_min', 'Calor_idx',
         'Q_OCP', 'OCP', 'AD', 'CR', 'CLTs',

         'dia_semana', 'trimestre', 'mes', 'dia_ano', 'dia_mes', 'semana_ano',
            'ELET_CONS_CLT_lag1', 'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean', 'ELET_CONS_CLT_pct_change',
```

```
            'ELET_CONS_CLT'] # Aqui, o script seleciona um conjunto de variáveis de entrada — e a última coluna escolhida é ELET_CONS_CLT, o valor que se deseja prever.
df1_gru2 = df1_filtrado3[features]
```

```
df1_gru2 = df1_gru2.replace([np.inf, -np.inf], np.nan)

# Opção 1: Preencher com a média de cada coluna
#df1_gru = df1_gru2.fillna(df1_gru.mean())

# Opção 2: Remover linhas com NaN (se não forem muitas)
df1_gru2 = df1_gru2.dropna()
```

```
df1_gru2 = df1_gru2.astype('float64')
```

```
import numpy as np
import pandas as pd

# 1. Garante que é um DataFrame
df1_gru2 = pd.DataFrame(df1_gru2)

# 2. Substitui valores infinitos por NaN
df1_gru2.replace([np.inf, -np.inf], np.nan, inplace=True)

# 3. Verifica se ainda há NaNs
print("Valores nulos por coluna:")
print(df1_gru2.isnull().sum())

# 4. Preenche os NaNs com a média de cada coluna
df1_gru2.fillna(df1_gru2.mean(), inplace=True)

# 5. Converte tudo para float64 (se ainda não estiver)
df1_gru2 = df1_gru2.astype('float64')

# 6. Aplica o MinMaxScaler
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
df1_normalizados2 = scaler.fit_transform(df1_gru2)
```

```
Valores nulos por coluna:
T_mean                        0
T_max                         0
T_min                         0
RH_mean                       0
RH_max                        0
RH_min                        0
Calor_idx                     0
Q_OCP                         0
OCP                           0
AD                            0
CR                            0
CLTs                          0
dia_semana                    0
trimestre                     0
mes                           0
dia_ano                       0
dia_mes                       0
semana_ano                    0
ELET_CONS_CLT_lag1            0
ELET_CONS_CLT_lag7            0
ELET_CONS_CLT_rolling_mean    0
ELET_CONS_CLT_pct_change      0
ELET_CONS_CLT                 0
dtype: int64
```

```
#2º itereção



# 🔄 Criar sequências temporais para treinamento do GRU
def criar_sequencias(df1_normalizados2, n_passo):
    X, y = [], []
    for i in range(len(df1_normalizados2) - n_passo):
        X.append(df1_normalizados2[i:i+n_passo, :-1])  # Variáveis climáticas e temporais
        y.append(df1_normalizados2[i+n_passo, -1])  # Consumo elétrico
    return np.array(X), np.array(y)

# Definir tamanho da janela de tempo
n_passo = 15 # Exemplo: usar 30 dias para prever o próximo dia
X_train, y_train = criar_sequencias(df1_normalizados2, n_passo)

# 🎯 Criar modelo GRU otimizado
model2_gru = Sequential([
    GRU(100, activation='tanh', input_shape=(X_train.shape[1], X_train.shape[2])),
    Dropout(0.2),  # Reduz overfitting
    Dense(1)  # Camada de saída
])

# ⏳ Compilar modelo com otimizador Adam
model2_gru.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), loss='mse')

# 🚀 Treinar modelo
history2 = model2_gru.fit(X_train, y_train, epochs=100, batch_size=16, validation_split=0.2)

# 🌑 Fazer previsões


previsoes2 = model2_gru.predict(X_train)

# 🚀 Reverter normalização para obter valores reais
previsoes2_reais = scaler.inverse_transform(np.concatenate((X_train[:, -1, :], previsoes2), axis=1))[:, -1]
# Criar DataFrame com previsões e valores reais
previsoes2_df = pd.DataFrame({
    'Previsto': previsoes2_reais,
    'Real': df1_gru2['ELET_CONS_CLT'].iloc[n_passo:].values
}, index=df1_gru2.index[n_passo:])

# 📊 Visualizar previsões
import plotly.graph_objects as go

# Criar figura
fig = go.Figure()

# Adicionar traço Real
fig.add_trace(go.Scatter(
    x=df1_gru2.index,
    y=df1_gru2['ELET_CONS_CLT'],
    mode='lines',
    name='Real',
    line=dict(color='blue')
))

# Adicionar traço Previsto
fig.add_trace(go.Scatter(
    x=previsoes2_df.index,
    y=previsoes2_df['Previsto'],
    mode='lines',
    name='Previsto (GRU)',
```

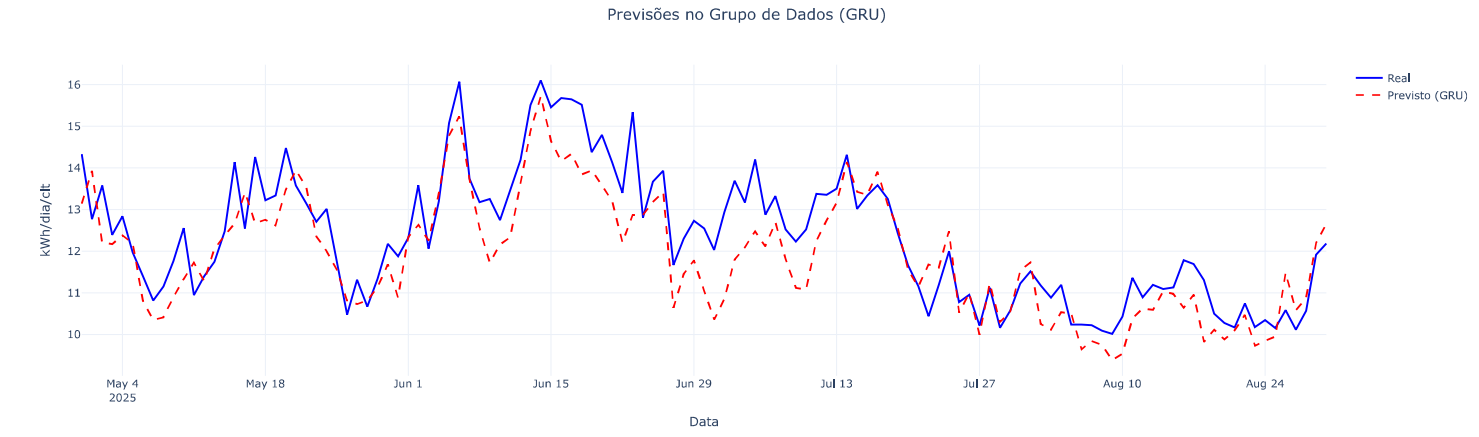
```
    line=dict(color='red', dash='dash')
))

# Personalizar layout
fig.update_layout(
    title='Comparação: Consumo Real vs. Previsto (GRU)',
    xaxis_title='Data',
    yaxis_title='Consumo Elétrico',
    legend=dict(x=0, y=1),
    font=dict(size=14),
    title_x=0.5,
    width=1000,
    height=500
)

# Exibir gráfico
fig.show()
```

Mostrar saída oculta

Validar

```
# Garantir que o índice é datetime
df1_gru2.index = pd.to_datetime(df1_gru2.index)

# Selecionar subconjunto
df_previsao = df1_gru2.loc['2025-04-01':'2025-08-31']


# Normalizar com o mesmo scaler usado no treino
df_previsao_norm = scaler.transform(df_previsao.values)

# Criar sequências para este subconjunto
X_test, y_test = criar_sequencias(df_previsao_norm, n_passo)

# Fazer previsões com o modelo já treinado
y_pred = model2_gru.predict(X_test)

# Reverter normalização para valores reais
y_pred_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_pred), axis=1)
)[:, -1]

y_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_test.reshape(-1,1)), axis=1)
)[:, -1]

# ============================================================
# 6. Visualizar resultados
# ============================================================

# Gerar índice correspondente ao último dia da janela usada para cada previsão
index_alinhado = df_previsao.index[n_passo - 1 : n_passo - 1 + len(y_pred_reais)]

# Garantir que os valores reais estejam alinhados com esse índice
real_values = df_previsao['ELET_CONS_CLT'].iloc[n_passo - 1 : n_passo - 1 + len(y_pred_reais)].values

# Plot
fig = go.Figure()
fig.add_trace(go.Scatter(
    x=index_alinhado, y=real_values,
    mode='lines', name='Real', line=dict(color='blue')
))
fig.add_trace(go.Scatter(
    x=index_alinhado, y=y_pred_reais,
    mode='lines', name='Previsto (GRU)', line=dict(color='red', dash='dash')
))
fig.update_layout(
    title="Previsões no Grupo de Dados (GRU)",
    xaxis_title="Data", yaxis_title="kWh/dia/clt",
    template="plotly_white", title_x=0.5
)
fig.show()
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2739: UserWarning:

X does not have valid feature names, but MinMaxScaler was fitted with feature names
```




```
# Avaliar modelo
df_resultados = avaliar_gru(
    modelo=model2_gru,
    X_train=X_train, y_train=y_train,   # podes manter os originais
    X_test=X_test, y_test=y_test,       # aqui vai o subconjunto abril–agosto
    scaler=scaler,
    df_index=df_pred.index,             # índice do subconjunto
    n_passo=n_passo,
    modelo_nome="GRU",
    iteracao=2
)
```

Mostrar saída oculta

```
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nResultados acumulados:")
```

```
print(df_resultados)
```

Mostrar saída oculta

3ª iteração

```
df1_filtrado3.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change'],
      dtype='object')
```

```
# 🔍 Selecionar variáveis de entrada
features = ['T_mean', 'T_max', 'T_min', 'RH_mean', 'RH_max', 'RH_min', 'Calor_idx',
         'Q_OCP', 'OCP', 'AD', 'CR', 'CLTs',
            'EP.C', '50S.C', 'COZ.50S.C', 'COZ.EP.C', 'PST.C', 'LAV.C',

         'dia_semana', 'trimestre', 'mes', 'dia_ano', 'dia_mes', 'semana_ano',

            'CO2e_ELET','Total_CO2e', 'CO2e_QOCP',

           'ELET_CONS_CLT_lag1', 'ELET_CONS_CLT_lag7',
       'ELET_CONS_CLT_rolling_mean', 'ELET_CONS_CLT_rolling_median',
       'ELET_CONS_CLT_rolling_std', 'ELET_CONS_CLT_error_median',
       'ELET_CONS_CLT_pct_change',

            'ELET_CONS_CLT'] # Aqui, o script seleciona um conjunto de variáveis de entrada — e a última coluna escolhida é ELET_CONS_CLT, o valor que se deseja prever.
df1_gru3 = df1_filtrado3[features]
```

```
df1_gru3 = df1_gru3.replace([np.inf, -np.inf], np.nan)

# Opção 1: Preencher com a média de cada coluna
#df1_gru = df1_gru3.fillna(df1_gru.mean())

# Opção 2: Remover linhas com NaN (se não forem muitas)
df1_gru3 = df1_gru3.dropna()
```

```
df1_gru3 = df1_gru3.astype('float64')
```

```
import numpy as np
import pandas as pd

# 1. Garante que é um DataFrame
df1_gru3 = pd.DataFrame(df1_gru3)

# 2. Substitui valores infinitos por NaN
df1_gru3.replace([np.inf, -np.inf], np.nan, inplace=True)

# 3. Verifica se ainda há NaNs
print("Valores nulos por coluna:")
print(df1_gru3.isnull().sum())

# 4. Preenche os NaNs com a média de cada coluna
df1_gru3.fillna(df1_gru3.mean(), inplace=True)

# 5. Converte tudo para float64 (se ainda não estiver)
df1_gru3 = df1_gru3.astype('float64')

# 6. Aplica o MinMaxScaler
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
df1_normalizados3 = scaler.fit_transform(df1_gru3)
```

```
Valores nulos por coluna:
T_mean                      0
T_max                       0
T_min                       0
RH_mean                     0
RH_max                      0
RH_min                      0
Calor_idx                   0
Q_OCP                       0
OCP                         0
AD                          0
CR                          0
CLTs                        0
EP.C                        0
50S.C                       0
COZ.50S.C                   0
COZ.EP.C                    0
PST.C                       0
LAV.C                       0
dia_semana                  0
trimestre                   0
mes                         0
dia_ano                     0
dia_mes                     0
semana_ano                  0
CO2e_ELET                   0
Total_CO2e                  0
CO2e_QOCP                   0
ELET_CONS_CLT_lag1          0
ELET_CONS_CLT_lag7          0
ELET_CONS_CLT_rolling_mean  0
ELET_CONS_CLT_rolling_median  0
ELET_CONS_CLT_rolling_std   0
ELET_CONS_CLT_error_median  0
ELET_CONS_CLT_pct_change    0
ELET_CONS_CLT               0
dtype: int64
```

```
#3º itereção

# 🔄 Criar sequências temporais para treinamento do GRU
def criar_sequencias(df1_normalizados3, n_passo):
    X, y = [], []
    for i in range(len(df1_normalizados3) - n_passo):
        X.append(df1_normalizados3[i:i+n_passo, :-1])  # Variáveis climáticas e temporais
        y.append(df1_normalizados3[i+n_passo, -1])  # Consumo elétrico
    return np.array(X), np.array(y)

# Definir tamanho da janela de tempo
n_passo = 15 # Exemplo: usar 30 dias para prever o próximo dia
X_train, y_train = criar_sequencias(df1_normalizados3, n_passo)

# 🎯 Criar modelo GRU otimizado
model3_gru = Sequential([
```

```
    GRU(100, activation='tanh', input_shape=(X_train.shape[1], X_train.shape[2])),
    Dropout(0.2),  # Reduz overfitting
    Dense(1)  # Camada de saída
])

# ⏳ Compilar modelo com otimizador Adam
model3_gru.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), loss='mse')

# 🚀 Treinar modelo
history3 = model3_gru.fit(X_train, y_train, epochs=100, batch_size=16, validation_split=0.2)

# 🔮 Fazer previsões


previsoes3 = model3_gru.predict(X_train)

# 🚀 Reverter normalização para obter valores reais
previsoes3_reais = scaler.inverse_transform(np.concatenate((X_train[:, -1, :], previsoes3), axis=1))[:, -1]
# Criar DataFrame com previsões e valores reais
previsoes3_df = pd.DataFrame({
    'Previsto': previsoes2_reais,
    'Real': df1_gru2['ELET_CONS_CLT'].iloc[n_passo:].values
}, index=df1_gru2.index[n_passo:])

# 📊 Visualizar previsões
import plotly.graph_objects as go

# Criar figura
fig = go.Figure()

# Adicionar traço Real
fig.add_trace(go.Scatter(
    x=df1_gru3.index,
    y=df1_gru3['ELET_CONS_CLT'],
    mode='lines',
    name='Real',
    line=dict(color='blue')
))

# Adicionar traço Previsto
fig.add_trace(go.Scatter(
    x=previsoes3_df.index,
    y=previsoes3_df['Previsto'],
    mode='lines',
    name='Previsto (GRU)',
    line=dict(color='red', dash='dash')
))

# Personalizar layout
fig.update_layout(
    title='Comparação: Consumo Real vs. Previsto (GRU)',
    xaxis_title='Data',
    yaxis_title='kWh/dia/clt',
    legend=dict(x=0, y=1),
    font=dict(size=14),
    title_x=0.5,
    width=1000,
    height=500
)

# Exibir gráfico
fig.show()
```

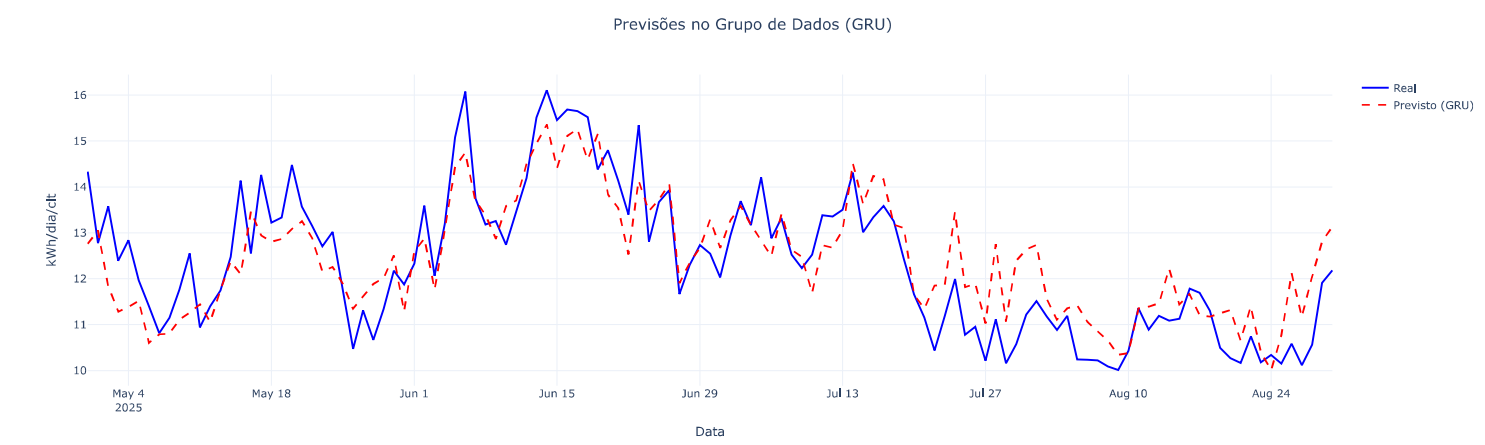
Mostrar saída oculta

Comece a programar ou gerar com a IA.

Validar

```
# Garantir que o índice é datetime
df1_gru3.index = pd.to_datetime(df1_gru3.index)

# Selecionar subconjunto
df_previsao = df1_gru3.loc['2025-04-01':'2025-08-31']


# Normalizar com o mesmo scaler usado no treino
df_previsao_norm = scaler.transform(df_previsao.values)

# Criar sequências para este subconjunto
X_test, y_test = criar_sequencias(df_previsao_norm, n_passo)

# Fazer previsões com o modelo já treinado
y_pred = model3_gru.predict(X_test)

# Reverter normalização para valores reais
y_pred_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_pred), axis=1)
)[:, -1]

y_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_test.reshape(-1,1)), axis=1)
)[:, -1]

# ============================================================
# 6. Visualizar resultados
# ============================================================

# Gerar índice correspondente ao último dia da janela usada para cada previsão
index_alinhado = df_previsao.index[n_passo - 1 : n_passo - 1 + len(y_pred_reais)]

# Garantir que os valores reais estejam alinhados com esse índice
real_values = df_previsao['ELET_CONS_CLT'].iloc[n_passo - 1 : n_passo - 1 + len(y_pred_reais)].values

# Plot
fig = go.Figure()
fig.add_trace(go.Scatter(
    x=index_alinhado, y=real_values,
    mode='lines', name='Real', line=dict(color='blue')
))
fig.add_trace(go.Scatter(
    x=index_alinhado, y=y_pred_reais,
    mode='lines', name='Previsto (GRU)', line=dict(color='red', dash='dash')
))
fig.update_layout(
    title="Previsões no Grupo de Dados (GRU)",
    xaxis_title="Data", yaxis_title="kWh/dia/clt",
    template="plotly_white", title_x=0.5
)
fig.show()
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2739: UserWarning:

X does not have valid feature names, but MinMaxScaler was fitted with feature names

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
```




```
# Avaliar modelo
df_resultados = avaliar_gru(
    modelo=model3_gru,
    X_train=X_train, y_train=y_train,   # podes manter os originais
    X_test=X_test, y_test=y_test,       # aqui vai o subconjunto abril-agosto
    scaler=scaler,
    df_index=df_pred.index,             # índice do subconjunto
    n_passo=n_passo,
    modelo_nome="GRU",
    iteracao=3
)
```

Mostrar saída oculta

```
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nResultados acumulados:")
print(df_resultados)
```

```
Resultados acumulados:
  Modelo  Iteração Conjunto     MSE  RMSE     MAE    R²
0    GRU         1   Treino  1.5272  1.24  0.8796  0.92
1    GRU         1    Teste  1.0344  1.02  0.7701  0.58
2    GRU         2   Treino  1.3627  1.17  0.8658  0.92
3    GRU         2    Teste  1.0475  1.02  0.8255  0.56
4    GRU         3   Treino  1.2581  1.12  0.7787  0.93
5    GRU         3    Teste  0.8839  0.94  0.7369  0.63
```

```
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nResultados acumulados:")
print(df_resultados)
```

Mostrar saída oculta

```
# Adicionar coluna de previsões (apenas nas datas alinhadas)
df_pred.loc[index_alinhado, "Previsto_GRU3"] = y_pred_reais

# Opcional: também podes guardar o erro
df_pred.loc[index_alinhado, "Erro_GRU3"] = ((df_pred["Previsto_GRU3"] - df_pred["ELET_CONS_CLT"])/df_pred["ELET_CONS_CLT"])*100
```

4ª iteração

```
#4ª iteração
```

```
df1_filtrado3.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change'],
      dtype='object')
```

```
df.index = pd.to_datetime(df.index)
# 🔍 Selecionar variáveis de entrada
features = ['T_mean', 'T_max', 'T_min', 'RH_mean', 'RH_max', 'RH_min', 'Calor_idx',
         'Q_OCP', 'OCP', 'AD', 'CR', 'CLTs',
            'EP.C', '50S.C', 'COZ.50S.C', 'COZ.EP.C', 'PST.C', 'LAV.C',

         'dia_semana', 'trimestre', 'mes', 'dia_ano', 'dia_mes', 'semana_ano',

            'CO2e_ELET','Total_CO2e', 'CO2e_QOCP',

          'ELET_CONS_CLT_lag1', 'ELET_CONS_CLT_lag7',
       'ELET_CONS_CLT_rolling_mean', 'ELET_CONS_CLT_rolling_median',
       'ELET_CONS_CLT_rolling_std', 'ELET_CONS_CLT_error_median',
       'ELET_CONS_CLT_pct_change',

            'ELET_CONS_CLT']

# Dados de treino: apenas 2024
df_gru_24 = df1_filtrado3.loc['2024-04-01':'2024-10-31', features]

# Dados de teste/previsão: apenas 2025
df_gru_25 = df1_filtrado3.loc['2025-04-01':'2025-08-31',features]
```

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
df_gru_24_scaled = scaler.fit_transform(df_gru_24)
```

```
df_gru_25_scaled = scaler.transform(df_gru_25)
```

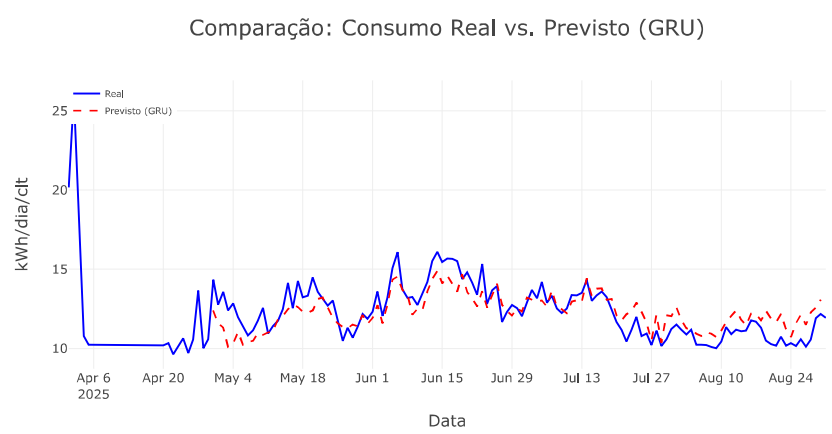
```
def criar_sequencias(array, n_passo):
    X, y = [], []
    for i in range(len(array) - n_passo):
        X.append(array[i:i+n_passo, :-1])  # todas as features exceto a última
        y.append(array[i+n_passo, -1])     # última coluna = target
    return np.array(X), np.array(y)

# Criar sequências de treino (2024)
n_passo = 15
X_train, y_train = criar_sequencias(df_gru_24_scaled, n_passo)

# Criar sequências de teste (2025)
X_test, y_test = criar_sequencias(df_gru_25_scaled, n_passo)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GRU, Dropout, Dense

model4_gru = Sequential([
    GRU(100, activation='tanh', input_shape=(X_train.shape[1], X_train.shape[2])),
    Dropout(0.2),
    Dense(1)
])

model4_gru.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005), loss='mse')
history4 = model4_gru.fit(X_train, y_train, epochs=100, batch_size=16, validation_split=0.2)
```

Mostrar saída oculta

```
# 🔮 Fazer previsões


previsoes4 = model4_gru.predict(X_test)

# 🚀 Reverter normalização para obter valores reais
previsoes4_reais = scaler.inverse_transform(np.concatenate((X_test[:, -1, :], previsoes4), axis=1))[:, -1]

# Gerar índice correspondente ao último dia da janela usada para cada previsão
index_anterior = df_gru_25.index[n_passo - 1 : n_passo - 1 + len(previsoes4_reais)]

# Garantir que os valores reais estejam alinhados com esse índice
real_values = df_gru_25['ELET_CONS_CLT'].iloc[n_passo - 1 : n_passo - 1 + len(previsoes4_reais)].values

# Criar DataFrame com previsões associadas ao dia anterior
previsoes4_df = pd.DataFrame({
    'Previsto': previsoes4_reais,
    'Real': real_values
}, index=index_anterior)


# 📊 Visualizar previsões
import plotly.graph_objects as go

# Criar figura
fig = go.Figure()

# Adicionar traço Real
fig.add_trace(go.Scatter(
    x=df_gru_25.index,
    y=df_gru_25['ELET_CONS_CLT'],
    mode='lines',
    name='Real',
    line=dict(color='blue')
))

# Adicionar traço Previsto
fig.add_trace(go.Scatter(
    x=previsoes4_df.index,
    y=previsoes4_df['Previsto'],
    mode='lines',
    name='Previsto (GRU)',
    line=dict(color='red', dash='dash')
))

# Personalizar layout
fig.update_layout(
    title='Comparação: Consumo Real vs. Previsto (GRU)',
    xaxis_title='Data',
    yaxis_title='kWh/dia/clt',
    legend=dict(x=0, y=1),
    font=dict(size=14),
    title_x=0.5,
    width=1000,
    height=500
)

# Exibir gráfico
fig.show()
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
```



Quando se define uma janela de entrada com n_passo = 15, por exemplo, o modelo está a aprender a prever o próximo valor com base nos 15 dias anteriores. Ou seja:

Entrada: dados de 03/06 até 17/06

Saída (target): valor de 18/06

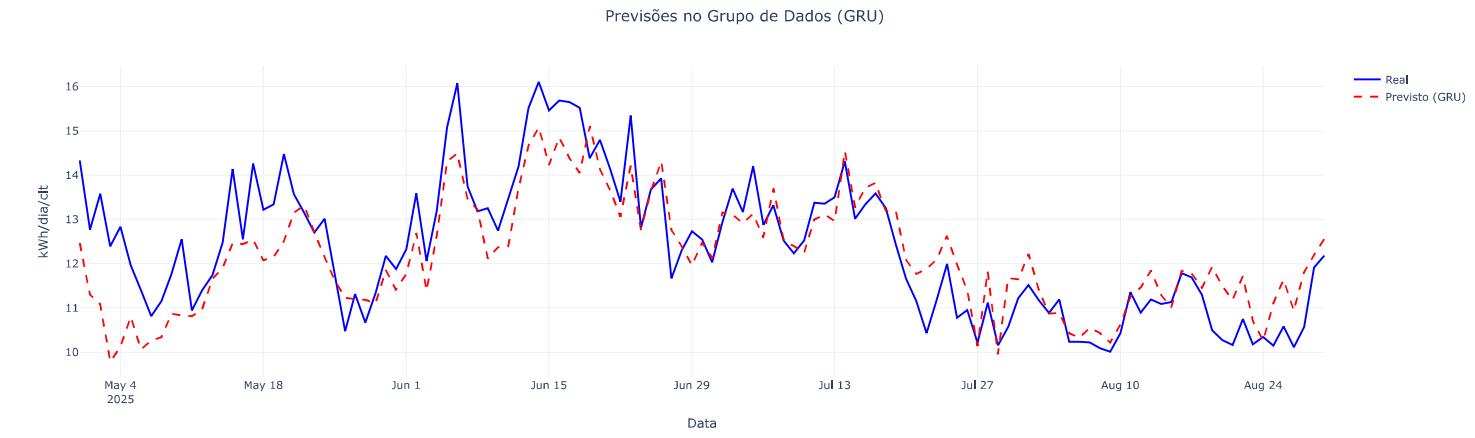
Então, quando se alimenta o modelo com os dados de 03/06 a 17/06, ele gera uma previsão para 18/06 — não para o último dia da janela.

Onde o desalinhamento acontece: Se não ajustar o índice das previsões corretamente, pode acabar por associar a previsão ao último dia da janela (17/06) em vez do dia seguinte (18/06). Isso faz parecer que a previsão está "um dia à frente" ou "um dia atrasada", dependendo de como se monta o DataFrame.

Validar

```
# Garantir que o índice é datetime
df_gru_25.index = pd.to_datetime(df_gru_25.index)

# Selecionar subconjunto
df_previsao = df_gru_25.loc['2025-04-01':'2025-08-31']


# Normalizar com o mesmo scaler usado no treino
df_previsao_norm = scaler.transform(df_previsao.values)

# Criar sequências para este subconjunto
X_test, y_test = criar_sequencias(df_previsao_norm, n_passo)

# Fazer previsões com o modelo já treinado
y_pred = model4_gru.predict(X_test)

# Reverter normalização para valores reais
y_pred_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_pred), axis=1)
)[:, -1]

y_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_test.reshape(-1,1)), axis=1)
)[:, -1]

# ============================================================
# 6. Visualizar resultados
# ============================================================

# Gerar índice correspondente ao último dia da janela usada para cada previsão
index_alinhado = df_previsao.index[n_passo - 1 : n_passo - 1 + len(y_pred_reais)]

# Garantir que os valores reais estejam alinhados com esse índice
real_values = df_previsao['ELET_CONS_CLT'].iloc[n_passo - 1 : n_passo - 1 + len(y_pred_reais)].values

# Plot
fig = go.Figure()
fig.add_trace(go.Scatter(
    x=index_alinhado, y=real_values,
    mode='lines', name='Real', line=dict(color='blue')
))
fig.add_trace(go.Scatter(
    x=index_alinhado, y=y_pred_reais,
    mode='lines', name='Previsto (GRU)', line=dict(color='red', dash='dash')
))
fig.update_layout(
    title="Previsões no Grupo de Dados (GRU)",
    xaxis_title="Data", yaxis_title="kWh/dia/clt",
    template="plotly_white", title_x=0.5
)
fig.show()
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2739: UserWarning:

X does not have valid feature names, but MinMaxScaler was fitted with feature names
```



Comece a programar ou gerar com a IA.

```
# Avaliar modelo
df_resultados = avaliar_gru(
    modelo=model4_gru,
    X_train=X_train, y_train=y_train,   # podes manter os originais
    X_test=X_test, y_test=y_test,       # aqui vai o subconjunto abril–agosto
    scaler=scaler,
    df_index=df_pred.index,             # índice do subconjunto
    n_passo=n_passo,
    modelo_nome="GRU",
    iteracao=4
)
```

Mostrar saída oculta

```
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nResultados acumulados:")
print(df_resultados)
```

Mostrar saída oculta

Comece a programar ou gerar com a IA.

```
# Adicionar coluna de previsões (apenas nas datas alinhadas)
df_pred.loc[index_alinhado, "Previsto_GRU4"] = y_pred_reais
```

```
# Opcional: também podes guardar o erro
df_pred.loc[index_alinhado, "Erro_GRU4"] = ((df_pred["Previsto_GRU4"] - df_pred["ELET_CONS_CLT"])/df_pred["ELET_CONS_CLT"])*100
```

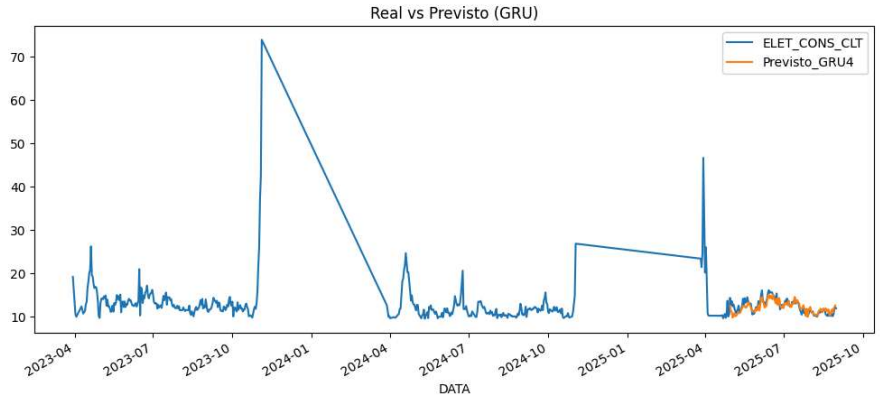
```
df_pred.tail()
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	ELET_CONS_CLT_pct_change	previsão_svr	previsão_rf	previsão_tree	erro_relativo_rf	erro_relativo_svr	previsão_xgb_f	e
2025-08-27	23.41699	31.9	18.3	13.6	75.88219	89.0	51.5	37.5	23.98021	12914.0	...	-0.044634	10.276879	10.359583	9.926178	2.440664	1.622845	9.953657	
2025-08-28	22.23625	30.2	18.3	11.9	80.14818	100.0	54.7	45.3	22.37211	12826.0	...	0.044727	10.600844	10.780303	10.120344	2.037177	0.338569	9.920411	
2025-08-29	22.64507	34.0	17.5	16.5	79.59112	91.5	54.4	37.1	22.82476	12985.0	...	0.127568	11.387050	10.840260	10.120344	-9.003590	-4.413674	10.535409	
2025-08-30	24.65698	34.5	19.7	14.8	78.81356	92.0	56.4	35.6	25.36257	12416.0	...	0.022803	12.172142	12.321556	12.483464	1.124885	-0.101380	12.037998	
2025-08-31	22.51888	30.1	18.7	11.4	83.64605	98.5	59.9	38.6	22.13280	12372.0	...	-0.020840	11.369584	11.334702	10.797468	-4.994458	-4.702080	11.249270	

5 rows × 57 columns

```
df_pred[["ELET_CONS_CLT", "Previsto_GRU4"]].plot(figsize=(12,5), title="Real vs Previsto (GRU)")
```

```
<Axes: title={'center': 'Real vs Previsto (GRU)'}, xlabel='DATA'>
```



5ª iteração

```
#5ª iteração
```

```
df1_filtrado3.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change'],
      dtype='object')
```

```
df.index = pd.to_datetime(df.index)
# 🔍 Selecionar variáveis de entrada
features = ['T_mean', 'T_max', 'T_min', 'RH_mean', 'RH_max', 'RH_min', 'Calor_idx',
         'Q_OCP', 'OCP', 'AD', 'CR', 'CLTs',
            'EP.C', '50S.C', 'COZ.50S.C', 'COZ.EP.C', 'PST.C', 'LAV.C',

         'dia_semana', 'trimestre', 'mes', 'dia_ano', 'dia_mes', 'semana_ano',

            'CO2e_ELET','Total_CO2e', 'CO2e_QOCP',

           'ELET_CONS_CLT_lag1', 'ELET_CONS_CLT_lag7',
       'ELET_CONS_CLT_rolling_mean', 'ELET_CONS_CLT_rolling_median',
       'ELET_CONS_CLT_rolling_std', 'ELET_CONS_CLT_error_median',
       'ELET_CONS_CLT_pct_change',

            'ELET_CONS_CLT']

# Dados de treino: apenas 2024
# Selecionar intervalo de 2023
df_gru_23 = df1_filtrado3.loc['2023-04-01':'2023-10-31', features]

# Selecionar intervalo de 2024
df_gru_24 = df1_filtrado3.loc['2024-04-01':'2024-10-31', features]

# Concatenar os dois períodos no mesmo DataFrame
df_gru_24 = pd.concat([df_gru_23, df_gru_24])

# Dados de teste/previsão: apenas 2025
df_gru_25 = df1_filtrado3.loc['2025-04-1':'2025-08-31',features]
```

```
Comece a programar ou gerar com a IA.
```

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
df_gru_24_scaled = scaler.fit_transform(df_gru_24)
df_gru_25_scaled = scaler.transform(df_gru_25)
```

```
def criar_sequencias(array, n_passo):
    X, y = [], []
    for i in range(len(array) - n_passo):
        X.append(array[i:i+n_passo, :-1])  # todas as features exceto a última
        y.append(array[i+n_passo, -1])     # última coluna = target
    return np.array(X), np.array(y)

# Criar sequências de treino (2024)
n_passo = 15
X_train, y_train = criar_sequencias(df_gru_24_scaled, n_passo)

# Criar sequências de teste (2025)
```

```
X_test, y_test = criar_sequencias(df_gru_25_scaled, n_passo)
```

```
"""# Definir tamanho da janela de tempo
n_passo = 30 # Exemplo: usar 30 dias para prever o próximo dia
X_train, y_train = criar_sequencias(df1_normalizados3, n_passo)

# 🎯 Criar modelo GRU otimizado
model3_gru = Sequential([
    GRU(500, activation='tanh', input_shape=(X_train.shape[1], X_train.shape[2])),
    Dropout(0.1),  # Reduz overfitting
    Dense(1)  # Camada de saída
])

# ⏳ Compilar modelo com otimizador Adam
model3_gru.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005), loss='mse')"""
```

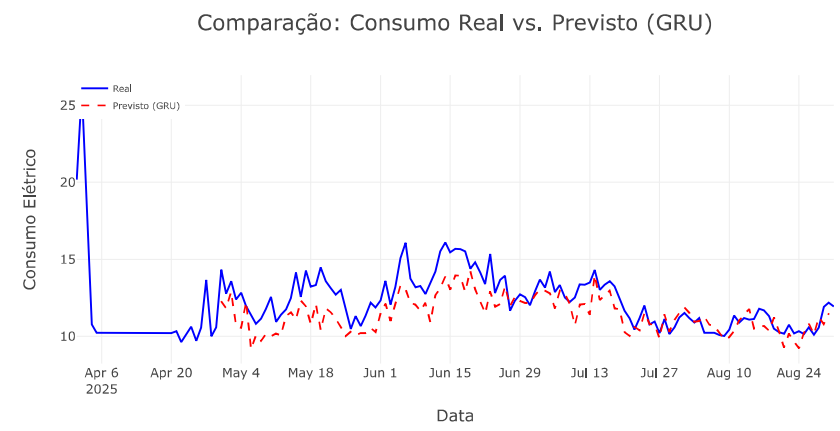
```
'# Definir tamanho da janela de tempo\nn_passo = 30 # Exemplo: usar 30 dias para prever o próximo dia\nX_train, y_train = criar_sequencias(df1_normalizados3, n_passo)\n\n# 🎯 Criar modelo GRU otimizado\nmodel3_gr
u = Sequential([\n    GRU(500, activation='tanh', input_shape=(X_train.shape[1], X_train.shape[2])),\n    Dropout(0.1),  # Reduz overfitting\n    Dense(1)  # Camada de saída\n])\n\n# ⏳ Compilar modelo com otimiz
ador Adam\nmodel3_gru.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005), loss='mse')'
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GRU, Dropout, Dense

model5_gru = Sequential([
    GRU(200, activation='tanh', input_shape=(X_train.shape[1], X_train.shape[2])),
    Dropout(0.1),
    Dense(1)
])

model5_gru.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.005), loss='mse')
history5 = model5_gru.fit(X_train, y_train, epochs=200, batch_size=64, validation_split=0.1)
```

Mostrar saída oculta

```
# 🌑 Fazer previsões


previsoes5 = model5_gru.predict(X_test)

# 🚀 Reverter normalização para obter valores reais
previsoes5_reais = scaler.inverse_transform(np.concatenate((X_test[:, -1, :], previsoes5), axis=1))[:, -1]

# Gerar índice correspondente ao último dia da janela usada para cada previsão
index_anterior = df_gru_25.index[n_passo - 1 : n_passo - 1 + len(previsoes5_reais)]

# Garantir que os valores reais estejam alinhados com esse índice
real_values = df_gru_25['ELET_CONS_CLT'].iloc[n_passo - 1 : n_passo - 1 + len(previsoes5_reais)].values

# Criar DataFrame com previsões associadas ao dia anterior
previsoes5_df = pd.DataFrame({
    'Previsto': previsoes5_reais,
    'Real': real_values
}, index=index_anterior)


# 📊 Visualizar previsões
import plotly.graph_objects as go

# Criar figura
fig = go.Figure()

# Adicionar traço Real
fig.add_trace(go.Scatter(
    x=df_gru_25.index,
    y=df_gru_25['ELET_CONS_CLT'],
    mode='lines',
    name='Real',
    line=dict(color='blue')
))

# Adicionar traço Previsto
fig.add_trace(go.Scatter(
    x=previsoes5_df.index,
    y=previsoes5_df['Previsto'],
    mode='lines',
    name='Previsto (GRU)',
    line=dict(color='red', dash='dash')
))

# Personalizar layout
fig.update_layout(
    title='Comparação: Consumo Real vs. Previsto (GRU)',
    xaxis_title='Data',
    yaxis_title='Consumo Elétrico',
    legend=dict(x=0, y=1),
    font=dict(size=14),
    title_x=0.5,
    width=1000,
    height=500
)

# Exibir gráfico
fig.show()
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 114ms/step
```

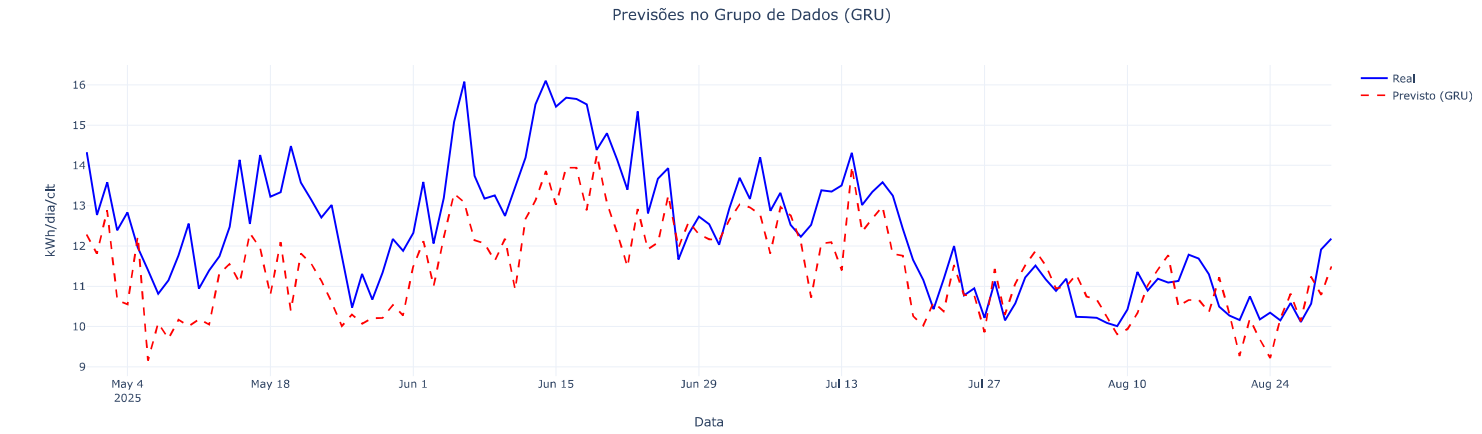


Comece a programar ou gerar com a IA.

Validar

```
# Garantir que o índice é datetime
df_gru_25.index = pd.to_datetime(df_gru_25.index)

# Selecionar subconjunto
df_previsao = df_gru_25.loc['2025-04-01':'2025-08-31']


# Normalizar com o mesmo scaler usado no treino
df_previsao_norm = scaler.transform(df_previsao.values)

# Criar sequências para este subconjunto
X_test, y_test = criar_sequencias(df_previsao_norm, n_passo)

# Fazer previsões com o modelo já treinado
y_pred = model5_gru.predict(X_test)

# Reverter normalização para valores reais
y_pred_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_pred), axis=1)
)[:, -1]

y_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_test.reshape(-1,1)), axis=1)
)[:, -1]

# ============================================================
# 6. Visualizar resultados
# ============================================================

# Gerar índice correspondente ao último dia da janela usada para cada previsão
index_alinhado = df_previsao.index[n_passo - 1 : n_passo - 1 + len(y_pred_reais)]

# Garantir que os valores reais estejam alinhados com esse índice
real_values = df_previsao['ELET_CONS_CLT'].iloc[n_passo - 1 : n_passo - 1 + len(y_pred_reais)].values

# Plot
fig = go.Figure()
fig.add_trace(go.Scatter(
    x=index_alinhado, y=real_values,
    mode='lines', name='Real', line=dict(color='blue')
))
fig.add_trace(go.Scatter(
    x=index_alinhado, y=y_pred_reais,
    mode='lines', name='Previsto (GRU)', line=dict(color='red', dash='dash')
))
fig.update_layout(
    title="Previsões no Grupo de Dados (GRU)",
    xaxis_title="Data", yaxis_title="kWh/dia/clt",
    template="plotly_white", title_x=0.5
)
fig.show()
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2739: UserWarning:

X does not have valid feature names, but MinMaxScaler was fitted with feature names
```




```
# Avaliar modelo
df_resultados = avaliar_gru(
    modelo=model5_gru,
    X_train=X_train, y_train=y_train,   # podes manter os originais
    X_test=X_test, y_test=y_test,       # aqui vai o subconjunto abril–agosto
    scaler=scaler,
    df_index=df_pred.index,             # índice do subconjunto
    n_passo=n_passo,
    modelo_nome="GRU",
    iteracao=5
)
```

Mostrar saída oculta

```
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nResultados acumulados:")
print(df_resultados)
```

```
Resultados acumulados:
  Modelo  Iteração Conjunto     MSE  RMSE     MAE    R²
0    GRU         1   Treino  1.5272  1.24  0.8796  0.92
1    GRU         1    Teste  1.0344  1.02  0.7701  0.58
2    GRU         2   Treino  1.3627  1.17  0.8658  0.92
3    GRU         2    Teste  1.0475  1.02  0.8255  0.56
4    GRU         3   Treino  1.2581  1.12  0.7787  0.93
5    GRU         3    Teste  0.8839  0.94  0.7369  0.63
6    GRU         4   Treino  0.5170  0.72  0.5368  0.77
7    GRU         4    Teste  1.1309  1.06  0.8346  0.52
8    GRU         5   Treino  0.3215  0.57  0.4216  0.93
9    GRU         5    Teste  1.9958  1.41  1.1462  0.15
```

Conclusões: Variabilidade entre iterações → mostra que o modelo é sensível à inicialização dos pesos, divisão treino/teste ou hiperparâmetros.

Overfitting frequente → R^2 muito alto no treino (0.9+) mas baixo no teste (0.3–0.4).

Melhor iteração → Iteração 5, com R^2 = 0.93 (treino) e R^2 = 0.60 (teste), equilíbrio razoável.

Próximos passos para melhorar:

Aumentar dropout ou regularização L2 para reduzir overfitting.

Testar early stopping para parar antes de sobreajustar.

Usar mais dados de treino (ex.: juntar 2023+2024).

Afinar hiperparâmetros (número de unidades GRU, taxa de aprendizagem, batch size).

A iteração 5 mostra que é possível atingir um equilíbrio melhor, mas ainda há espaço para otimização.

Quando se define uma janela de entrada com n_passo = 15, por exemplo, o modelo está a aprender a prever o próximo valor com base nos 15 dias anteriores. Ou seja:

Entrada: dados de 03/06 até 17/06

Saída (target): valor de 18/06

Então, quando se alimenta o modelo com os dados de 03/06 a 17/06, ele gera uma previsão para 18/06 — não para o último dia da janela.

Onde o desalinhamento acontece: Se não ajustar o índice das previsões corretamente, pode acabar por associar a previsão ao último dia da janela (17/06) em vez do dia seguinte (18/06). Isso faz parecer que a previsão está "um dia à frente" ou "um dia atrasada", dependendo de como se monta o DataFrame.

```
# Adicionar coluna de previsões (apenas nas datas alinhadas)
df_pred.loc[index_alinhado, "Previsto_GRU5"] = y_pred_reais

# Opcional: também podes guardar o erro
df_pred.loc[index_alinhado, "Erro_GRU5"] = ((df_pred["Previsto_GRU5"] - df_pred["ELET_CONS_CLT"])/df_pred["ELET_CONS_CLT"])*100
```

Final

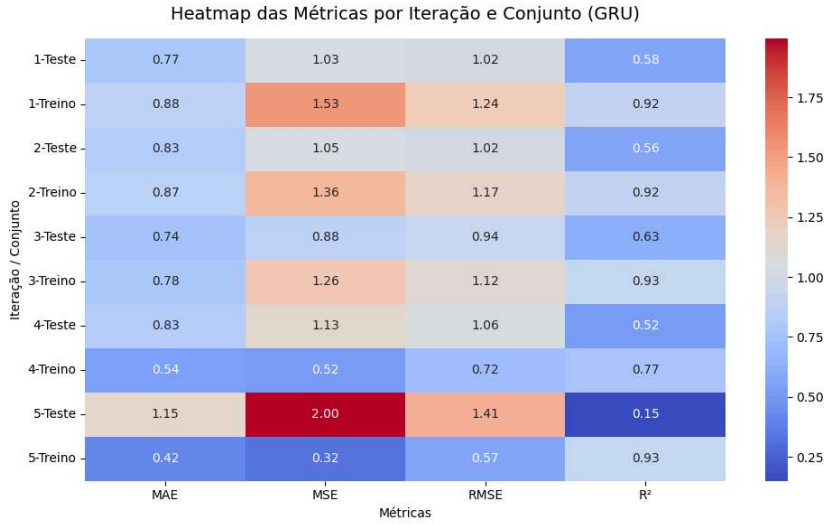
```
df_resultados
```

	Modelo	Iteração	Conjunto	MSE	RMSE	MAE	R²
0	GRU	1	Treino	1.5272	1.24	0.8796	0.92
1	GRU	1	Teste	1.0344	1.02	0.7701	0.58
2	GRU	2	Treino	1.3627	1.17	0.8658	0.92
3	GRU	2	Teste	1.0475	1.02	0.8255	0.56
4	GRU	3	Treino	1.2581	1.12	0.7787	0.93
5	GRU	3	Teste	0.8839	0.94	0.7369	0.63
6	GRU	4	Treino	0.5170	0.72	0.5368	0.77
7	GRU	4	Teste	1.1309	1.06	0.8346	0.52
8	GRU	5	Treino	0.3215	0.57	0.4216	0.93
9	GRU	5	Teste	1.9958	1.41	1.1462	0.15

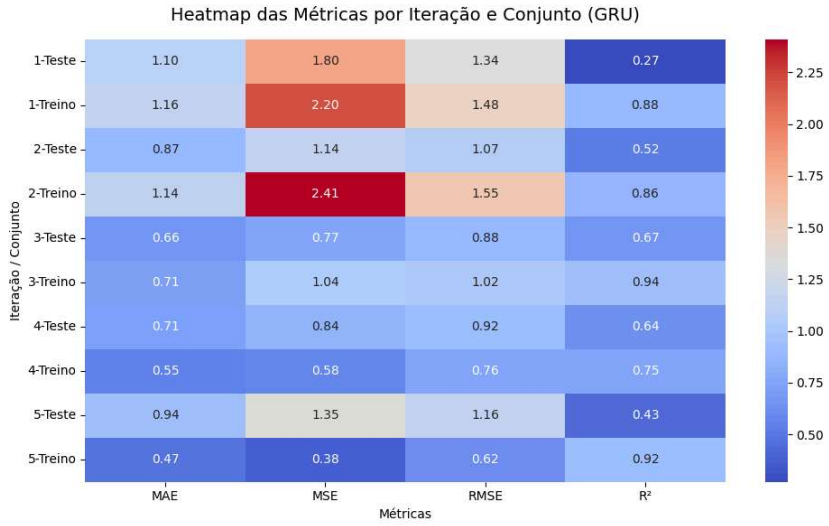
```
df_resultados
```

	Modelo	Iteração	Conjunto	MSE	RMSE	MAE	R²
0	GRU	1	Treino	2.1999	1.48	1.1620	0.88
1	GRU	1	Teste	1.7964	1.34	1.1035	0.27
2	GRU	2	Treino	2.4090	1.55	1.1357	0.86
3	GRU	2	Teste	1.1397	1.07	0.8739	0.52
4	GRU	3	Treino	1.0409	1.02	0.7101	0.94
5	GRU	3	Teste	0.7704	0.88	0.6577	0.67
6	GRU	4	Treino	0.5751	0.76	0.5525	0.75
7	GRU	4	Teste	0.8391	0.92	0.7145	0.64
8	GRU	5	Treino	0.3824	0.62	0.4652	0.92
9	GRU	5	Teste	1.3511	1.16	0.9385	0.43

```
import seaborn as sns
import matplotlib.pyplot as plt

# DataFrame com os resultados acumulados (já tens)
# df_resultados = pd.DataFrame(resultados_acumulados)

# Pivotar para formato adequado ao heatmap
df_heatmap = df_resultados.pivot_table(
    index=["Iteração", "Conjunto"],
    values=["MSE", "RMSE", "MAE", "R²"]
)

# Criar heatmap
plt.figure(figsize=(10,6))
sns.heatmap(df_heatmap, annot=True, fmt=".2f", cmap="coolwarm", cbar=True)

plt.title("Heatmap das Métricas por Iteração e Conjunto (GRU)", fontsize=14, pad=15)
plt.ylabel("Iteração / Conjunto")
plt.xlabel("Métricas")
plt.tight_layout()
plt.show()
```




```
import seaborn as sns
import matplotlib.pyplot as plt

# DataFrame com os resultados acumulados (já tens)
# df_resultados = pd.DataFrame(resultados_acumulados)

# Pivotar para formato adequado ao heatmap
df_heatmap = df_resultados.pivot_table(
    index=["Iteração", "Conjunto"],
    values=["MSE", "RMSE", "MAE", "R²"]
)

# Criar heatmap
plt.figure(figsize=(10,6))
sns.heatmap(df_heatmap, annot=True, fmt=".2f", cmap="coolwarm", cbar=True)

plt.title("Heatmap das Métricas por Iteração e Conjunto (GRU)", fontsize=14, pad=15)
plt.ylabel("Iteração / Conjunto")
plt.xlabel("Métricas")
plt.tight_layout()
plt.show()
```



Métricas apresentadas MSE (Mean Squared Error) → erro médio quadrático (quanto menor, melhor).

RMSE (Root Mean Squared Error) → raiz do MSE, mais interpretável na mesma escala da variável (kWh/dia/clt).

MAE (Mean Absolute Error) → erro médio absoluto (quanto menor, melhor).

R^2 (Coeficiente de Determinação) → mede a proporção da variância explicada (quanto mais próximo de 1, melhor).

```
df_pred
```

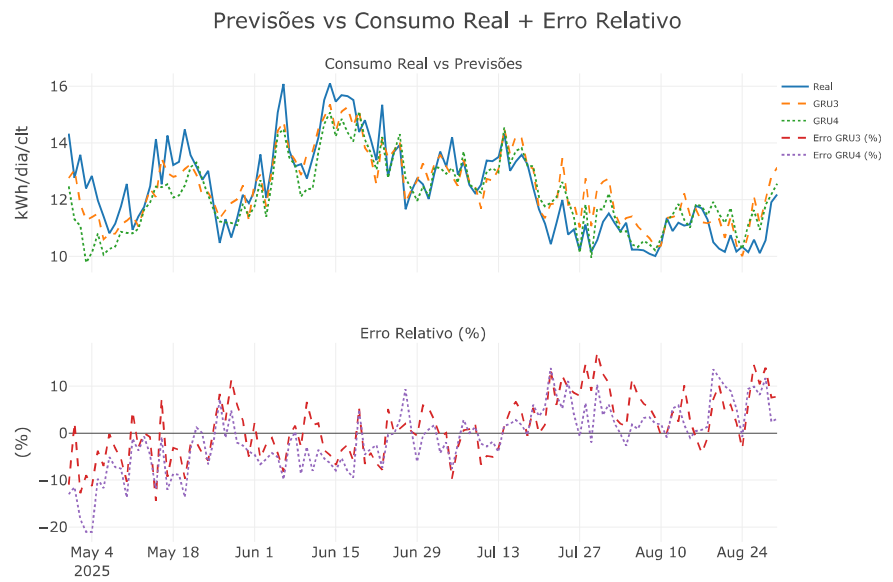
DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	previsão_rf	previsão_tree	erro_relativo_rf	erro_relativo_svr	previsão_xgb_f	erro_relativo_xgb_f
2023-03-30	18.750000	24.900000	13.8	11.099999	77.516667	91.000000	63.299999	27.700001	21.755307	5679.0	...	21.171876	21.691240	10.351742	8.128962	20.598866	7.365103
2023-03-31	18.333333	24.400000	14.1	10.299999	85.416667	100.000000	70.000000	30.000000	19.473651	11569.0	...	12.724047	11.338827	-21.691405	-27.276629	14.225427	-12.451346
2023-04-02	18.366667	22.100000	14.2	7.900001	74.466667	95.599998	44.400002	51.199997	22.624800	8496.0	...	11.474113	11.338827	9.122916	4.165229	11.925946	13.420016
2023-04-03	19.140000	25.299999	15.0	10.299999	55.960000	75.500000	44.500000	31.000000	25.191946	9342.0	...	10.217535	9.836980	2.481590	8.224761	10.838500	8.709854
2023-04-09	18.785715	26.700001	13.6	13.100000	74.428573	87.300003	59.900002	27.400002	22.498339	11715.0	...	10.075294	9.836980	-18.468815	-22.997806	10.455238	-15.394230
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2025-08-27	23.416990	31.900000	18.3	13.600000	75.882190	89.000000	51.500000	37.500000	23.980210	12914.0	...	10.359583	9.926178	2.440664	1.622845	9.953657	-1.573330
2025-08-28	22.236250	30.200000	18.3	11.900000	80.148180	100.000000	54.700000	45.300000	22.372110	12826.0	...	10.780303	10.120344	2.037177	0.338569	9.920411	-6.101832
2025-08-29	22.645070	34.000000	17.5	16.500000	79.591120	91.500000	54.400000	37.100000	22.824760	12985.0	...	10.840260	10.120344	-9.003590	-4.413674	10.535409	-11.562605
2025-08-30	24.656980	34.500000	19.7	14.800000	78.813560	92.000000	56.400000	35.600000	25.362570	12416.0	...	12.321556	12.483464	1.124885	-0.101380	12.037998	-1.202318
2025-08-31	22.518880	30.100000	18.7	11.400000	83.646050	98.500000	59.900000	38.600000	22.132800	12372.0	...	11.334702	10.797468	-4.994458	-4.702080	11.249270	-5.710528

552 rows × 59 columns

```
df_pred.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change',
       'previsão_svr', 'previsão_rf', 'previsão_tree', 'erro_relativo_rf',
       'erro_relativo_svr', 'previsão_xgb1', 'previsão_xgb2', 'previsão_xgb3',
       'previsão_xgb_f', 'erro_relativo_xgb_f', 'Previsto_GRU4', 'Erro_GRU4',
       'Previsto_GRU5', 'Erro_GRU5'],
      dtype='object')
```

```
# Selecionar os dados
df_plot = df_pred[['ELET_CONS_CLT', 'Previsto_GRU3', 'Erro_GRU3', 'Previsto_GRU4',
       'Erro_GRU4']].dropna().copy()

# Resetar índice e renomear para 'Data'
df_plot = df_plot.reset_index().rename(columns={df_plot.index.name or 'index': 'Data'})

# Criar subplots
fig = make_subplots(rows=2, cols=1, shared_xaxes=True, vertical_spacing=0.15,
                    subplot_titles=("Consumo Real vs Previsões", "Erro Relativo (%)"))

# Subplot 1: Consumo real e previsões
fig.add_trace(go.Scatter(x=df_plot['Data'], y=df_plot['ELET_CONS_CLT'],
                         mode='lines', name='Real', line=dict( dash='solid')), row=1, col=1)

fig.add_trace(go.Scatter(x=df_plot['Data'], y=df_plot['Previsto_GRU3'],
                         mode='lines', name='GRU3', line=dict( dash='dash')), row=1, col=1)

fig.add_trace(go.Scatter(x=df_plot['Data'], y=df_plot['Previsto_GRU4'],
                         mode='lines', name='GRU4', line=dict( dash='dot')), row=1, col=1)



# Subplot 2: Erro relativo
fig.add_trace(go.Scatter(x=df_plot['Data'], y=df_plot['Erro_GRU3'],
                         mode='lines', name='Erro GRU3 (%)', line=dict( dash='dash')), row=2, col=1)

fig.add_trace(go.Scatter(x=df_plot['Data'], y=df_plot['Erro_GRU4'],
                         mode='lines', name='Erro GRU4 (%)', line=dict( dash='dot')), row=2, col=1)



# Layout final
fig.update_layout(height=700, width=1000,
                  title_text="Previsões vs Consumo Real + Erro Relativo",
                  yaxis=dict(title='kWh/dia/clt'),
                  yaxis2=dict(title='(%)'),
                  hovermode='x unified')

fig.show()
```




```
df_pred.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change',
       'previsão_svr', 'previsão_rf', 'previsão_tree', 'erro_relativo_rf',
       'erro_relativo_svr', 'previsão_xgb1', 'previsão_xgb2', 'previsão_xgb3',
       'previsão_xgb_f', 'erro_relativo_xgb_f', 'Previsto_GRU4', 'Erro_GRU4',
       'Previsto_GRU5', 'Erro_GRU5'],
      dtype='object')
```

```
# Certifique-se de que ambos os índices estão no tipo datetime
dados_2025.index = pd.to_datetime(dados_2025.index)
df_pred.index = pd.to_datetime(df_pred.index)

# Selecionar apenas as colunas desejadas de df_pred
df_pred_selecionado = df_pred[['Previsto_GRU4', 'Erro_GRU4']]

# Realizar o merge com base no índice
dados_2025 = dados_2025.merge(df_pred_selecionado, left_index=True, right_index=True, how='left')
dados_2025
```

DATA	ELET_CONS_CLT	mes	med2024	erro_relativo	previsão_rf	previsão_svr	erro_relativo_rf	erro_relativo_svr	anomalia_negativa	previsão_xgb_f	erro_relativo_xgb_f	OCP[%]	GRU	Erro GRU	Previsto_GRU4
2025-04-01	20.167279	4	12.791762	57.658333	15.017605	14.971816	-25.534801	-25.761843	True	14.777119	-26.727258	48.2	NaN	NaN	NaN
2025-04-02	26.015000	4	12.791762	103.373071	13.820930	13.731936	-46.873228	-47.215314	True	14.777119	-43.197699	53.2	NaN	NaN	NaN
2025-04-04	10.777647	4	12.791762	15.745409	11.327024	9.976873	5.097373	-7.429948	False	11.680786	8.379743	71.0	NaN	NaN	NaN
2025-04-05	10.224138	4	12.791762	20.072484	10.120539	9.896231	-1.013282	-3.207189	False	10.548176	3.169342	75.4	NaN	NaN	NaN
2025-04-06	9.313346	4	12.791762	27.192627	NaN	NaN	NaN	NaN	False	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2025-08-27	10.112764	8	10.264049	1.473931	10.359583	10.276879	2.440664	1.622845	False	9.953657	-1.573330	89.0	10.936665	8.147133	10.936665
2025-08-28	10.565074	8	10.264049	2.932808	10.780303	10.600844	2.037177	0.338569	False	9.920411	-6.101832	84.8	11.804976	11.735856	11.804976
2025-08-29	11.912844	8	10.264049	16.063785	10.840260	11.387050	-9.003590	-4.413674	True	10.535409	-11.562605	76.8	12.204058	2.444539	12.204058
2025-08-30	12.184495	8	10.264049	18.710406	12.321556	12.172142	1.124885	-0.101380	False	12.037998	-1.202318	72.8	12.560123	3.082843	12.560123
2025-08-31	11.930569	8	10.264049	16.236474	11.334702	11.369584	-4.994458	-4.702080	False	11.249270	-5.710528	77.6	NaN	NaN	NaN

152 rows × 16 columns

```
#dados_2025 = dados_2025.merge(df1_filtrado3['OCP[%]'], left_index=True, right_index=True, how='left')
```

```
#dados_2025 = dados_2025.drop[]
```

```
dados_2025.columns
```

```
Index(['ELET_CONS_CLT', 'mes', 'med2024', 'erro_relativo', 'previsão_rf',
       'previsão_svr', 'erro_relativo_rf', 'erro_relativo_svr',
       'anomalia_negativa', 'previsão_xgb_f', 'erro_relativo_xgb_f', 'OCP[%]',
       'Previsto_GRU4', 'Erro_GRU4'],
      dtype='object')
```

```
#dados_2025 = dados_2025.drop(['OCP[%]_y'], axis=1)
```

```
#dados_2025 = dados_2025.rename(columns={'Previsto_GRU4': 'GRU', 'Erro_GRU4': 'Erro GRU'})
```

```
dados_2025.columns
```

```
Index(['ELET_CONS_CLT', 'mes', 'med2024', 'erro_relativo', 'previsão_rf',
       'previsão_svr', 'erro_relativo_rf', 'erro_relativo_svr',
       'anomalia_negativa', 'previsão_xgb_f', 'erro_relativo_xgb_f', 'OCP[%]',
       'GRU', 'Erro GRU'],
      dtype='object')
```

```
# Remove todas as linhas que contêm qualquer valor NaN
#dados_2025 = dados_2025_merged.dropna()
```

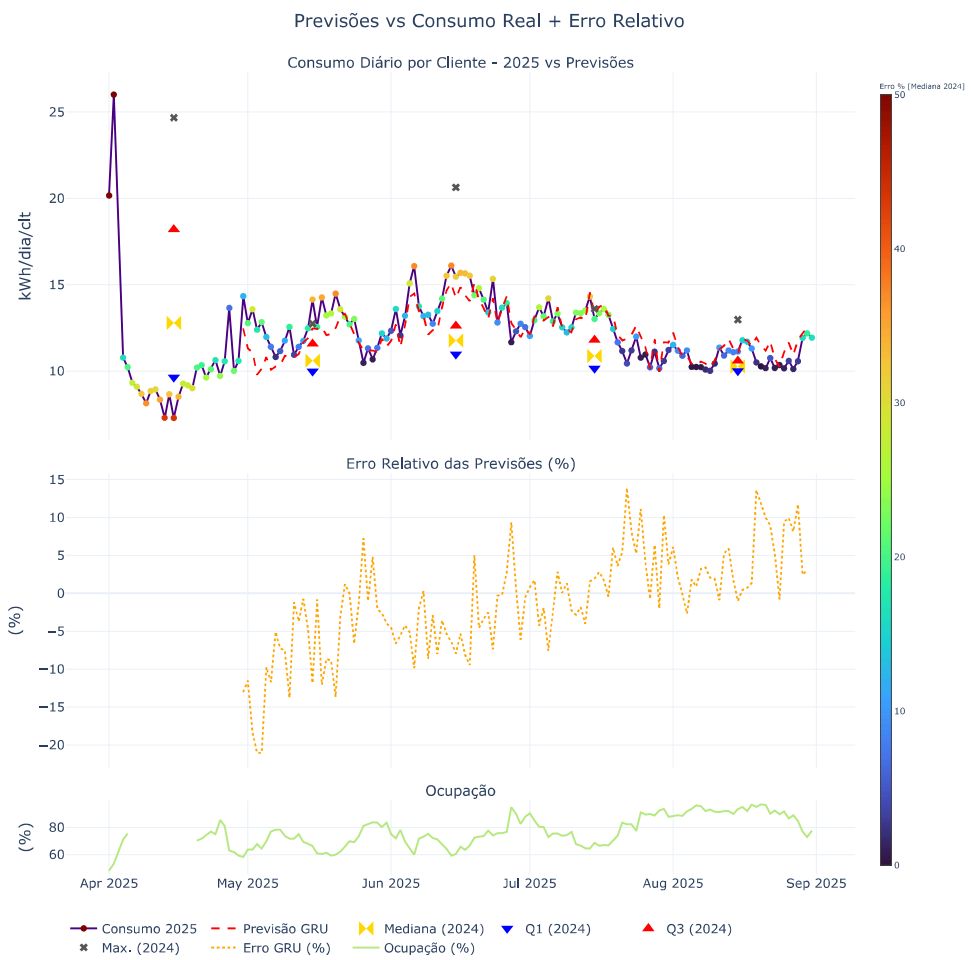
dados_2025

DATA	ELET_CONS_CLT	mes	med2024	erro_relativo	previsão_rf	previsão_svr	erro_relativo_rf	erro_relativo_svr	anomalia_negativa	previsão_xgb_f	erro_relativo_xgb_f	OCP[%]	GRU	Erro GRU
2025-04-01	20.167279	4	12.791762	57.658333	15.017605	14.971816	-25.534801	-25.761843	True	14.777119	-26.727258	48.2	NaN	NaN
2025-04-02	26.015000	4	12.791762	103.373071	13.820930	13.731936	-46.873228	-47.215314	True	14.777119	-43.197699	53.2	NaN	NaN
2025-04-04	10.777647	4	12.791762	15.745409	11.327024	9.976873	5.097373	-7.429948	False	11.680786	8.379743	71.0	NaN	NaN
2025-04-05	10.224138	4	12.791762	20.072484	10.120539	9.896231	-1.013282	-3.207189	False	10.548176	3.169342	75.4	NaN	NaN
2025-04-06	9.313346	4	12.791762	27.192627	NaN	NaN	NaN	NaN	False	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2025-08-27	10.112764	8	10.264049	1.473931	10.359583	10.276879	2.440664	1.622845	False	9.953657	-1.573330	89.0	10.936665	8.147133
2025-08-28	10.565074	8	10.264049	2.932808	10.780303	10.600844	2.037177	0.338569	False	9.920411	-6.101832	84.8	11.804976	11.735856
2025-08-29	11.912844	8	10.264049	16.063785	10.840260	11.387050	-9.003590	-4.413674	True	10.535409	-11.562605	76.8	12.204058	2.444539
2025-08-30	12.184495	8	10.264049	18.710406	12.321556	12.172142	1.124885	-0.101380	False	12.037998	-1.202318	72.8	12.560123	3.082843
2025-08-31	11.930569	8	10.264049	16.236474	11.334702	11.369584	-4.994458	-4.702080	False	11.249270	-5.710528	77.6	NaN	NaN

152 rows × 14 columns

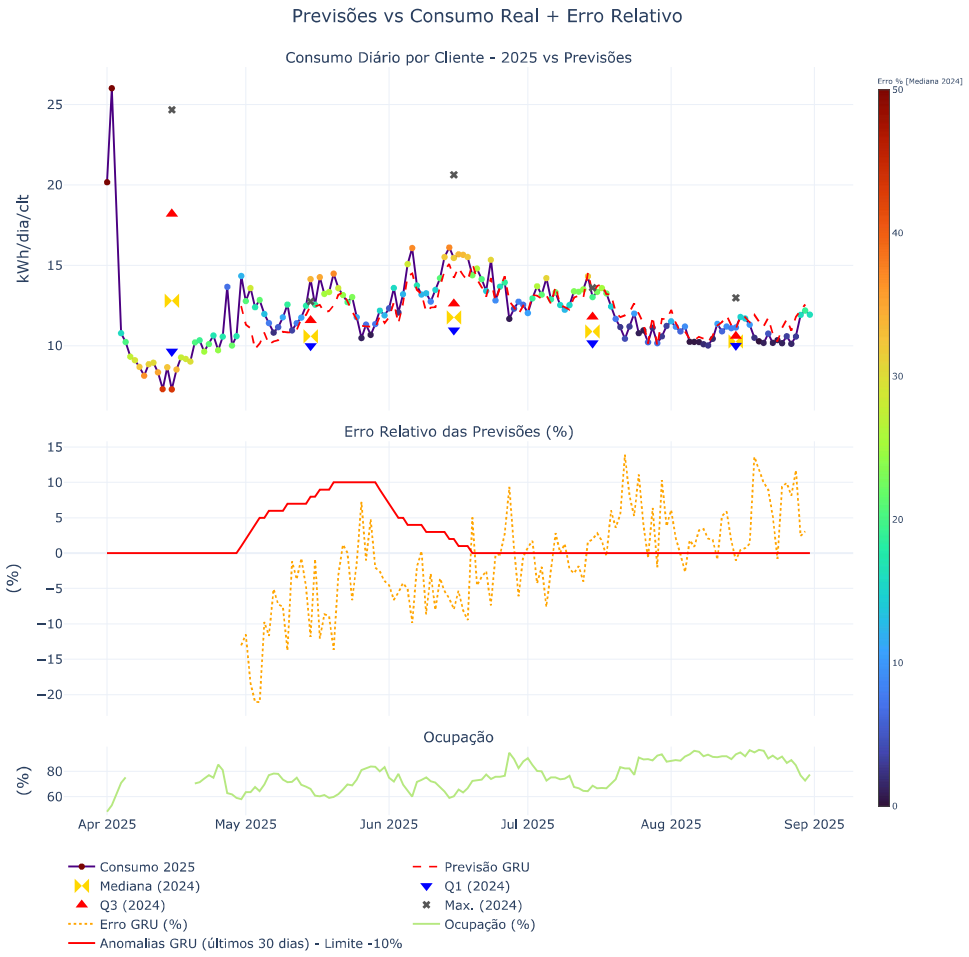
```
from plotly.subplots import make_subplots
import plotly.graph_objects as go

# Criar subplots: 2 linhas, 1 coluna
fig = make_subplots(
    rows=3, cols=1,
    shared_xaxes=True,
    vertical_spacing=0.041,
    row_heights=[0.5, 0.4, 0.1],
    subplot_titles=("Consumo Diário por Cliente - 2025 vs Previsões", "Erro Relativo das Previsões (%)", "Ocupação")
)

# ——————————————————————————————————————
# Subplot 1: Consumo real + previsões
fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['ELET_CONS_CLT'],
    mode='lines+markers',
    name='Consumo 2025',
    line=dict(color='indigo'),
    marker=dict(
        color=dados_2025['erro_relativo'],
        colorscale='Turbo',
        cmin=0,
        cmax=50,
        colorbar=dict(title='Erro % [Mediana 2024]', titlefont=dict(size=8), thickness=12, tickfont=dict(size=10)),
        size=7
    )
), row=1, col=1)

fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['Previsto_GRU4'],
    mode='lines',
    name='Previsão GRU',
    line=dict(dash='dash', color='red')
), row=1, col=1)




# Estatísticas mensais de 2024
fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['50%'],
    mode='markers',
    name='Mediana (2024)',
    marker=dict(color='gold', symbol='bowtie', size=16)
), row=1, col=1)

fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['25%'],
    mode='markers',
    name='Q1 (2024)',
    marker=dict(color='blue', symbol='triangle-down', size=12)
), row=1, col=1)

fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['75%'],
    mode='markers',
    name='Q3 (2024)',
    marker=dict(color='red', symbol='triangle-up', size=12)
), row=1, col=1)

fig.add_trace(go.Scatter(
    x=datas_15,
    y=resumo_mensal['max'],
    mode='markers',
    name='Max. (2024)',
    marker=dict(color='#565656',symbol='x', size=8)
), row=1, col=1)

# ——————————————————————————————————————
# Subplot 2: Erro relativo
fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['Erro_GRU4'],
    mode='lines',
    name='Erro GRU (%)',
    line=dict(dash='dot', color='orange')
), row=2, col=1)

# Subplot 3: OCP
fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=dados_2025['OCP[%]'],
    mode='lines',
    name='Ocupação (%)',
    #line=dict(dash='dash', color='green')
), row=3, col=1)
# ——————————————————————————————————————
# Layout final
fig.update_layout(
    height=1100,
    width=1100,
    title_text="Previsões vs Consumo Real + Erro Relativo",
    template='plotly_white',
    font=dict(size=14),
    title_x=0.5,
    legend=dict(orientation='h', y=-0.05),
    hovermode='x unified'
)
```

```
fig.update_yaxes(title_text='kWh/dia/clt', row=1, col=1)
fig.update_yaxes(title_text='(%)', row=2, col=1)
fig.update_yaxes(title_text='(%)', row=3, col=1)

fig.show()
```




```
import pandas as pd

# Definir o limite negativo de erro relativo (%)
limite_negativo = -10  # pode alterar para -15, etc.

# Selecionar a coluna de erro relativo do modelo a avaliar
coluna_erro = 'Erro_GRU4'  # ou 'erro_relativo_svr', 'erro_relativo_xgb_f'

# Criar coluna booleana: True apenas quando o erro relativo é menor que o limite negativo
dados_2025['anomalia_negativa'] = dados_2025[coluna_erro] < limite_negativo

# Contar anomalias por janela móvel de 30 dias
anomalias_neg_30d = (
    dados_2025['anomalia_negativa']
    .rolling(window=30, min_periods=1)
    .sum()
)

# Número total de anomalias negativas no período completo
total_anomalias_neg = dados_2025['anomalia_negativa'].sum()

print(f"Limite de erro relativo negativo: {limite_negativo}%")
print(f"Número total de anomalias negativas detetadas: {total_anomalias_neg}")
print("\nAnomalias negativas detetadas em cada janela de 30 dias:")
print(anomalias_neg_30d)

# (Opcional) Adicionar ao gráfico no subplot 2
fig.add_trace(go.Scatter(
    x=dados_2025.index,
    y=anomalias_neg_30d,
    mode='lines',
    name=f'Anomalias GRU (últimos 30 dias) - Limite {limite_negativo}%',
    line=dict(color='red', width=2)
), row=2, col=1)

fig.show()
```

```
Limite de erro relativo negativo: -10%
Número total de anomalias negativas detetadas: 10

Anomalias negativas detetadas em cada janela de 30 dias:
DATA
2025-04-01    0.0
2025-04-02    0.0
2025-04-04    0.0
2025-04-05    0.0
2025-04-06    0.0
              ...
2025-08-27    0.0
2025-08-28    0.0
2025-08-29    0.0
2025-08-30    0.0
2025-08-31    0.0
Name: anomalia_negativa, Length: 152, dtype: float64
```



Comece a programar ou gerar com a IA.

CNN

Comece a programar ou gerar com a IA.

1ª iteração

```
df1_filtrado3.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change'],
      dtype='object')
```

```
# 🔍 Selecionar variáveis de entrada
features = [ 'T_mean', 'RH_mean', 'Calor_idx',
         'Q_OCP', 'OCP', 'AD', 'CR', 'CLTs',

         'dia_semana', 'trimestre', 'mes', 'dia_ano', 'dia_mes', 'semana_ano',


            'ELET_CONS_CLT'] # Aqui, o script seleciona um conjunto de variáveis de entrada — e a última coluna escolhida é ELET_CONS_CLT, o valor que se deseja prever.
df1_cnn = df1_filtrado3[features]
```

```
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Conv1D, Dense, Dropout, Flatten
import tensorflow as tf
import numpy as np
import pandas as pd
import plotly.graph_objects as go

# ============================================================
# 1. Normalizar dados (X + y juntos)
# ============================================================
scaler = MinMaxScaler()
df1_normalizados = scaler.fit_transform(df1_cnn.values)

# ============================================================
# 2. Criar sequências temporais
# ============================================================
def criar_sequencias(df1_normalizados, n_passo):
    X, y = [], []
```

```
    for i in range(len(df1_normalizados) - n_passo):
        X.append(df1_normalizados[i:i+n_passo, :-1])  # features
        y.append(df1_normalizados[i+n_passo, -1])     # alvo (consumo)
    return np.array(X), np.array(y)

n_passo = 7
X_train, y_train = criar_sequencias(df1_normalizados, n_passo)

# ============================================================
# 3. Criar modelo CNN
# ============================================================
model1_cnn = Sequential([
    Input(shape=(X_train.shape[1], X_train.shape[2])),
    Conv1D(filters=128, kernel_size=5, activation='relu'),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(1)
])

# Compilar
model1_cnn.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.005), loss='mse')

# Treinar
history_cnn = model1_cnn.fit(X_train, y_train, epochs=100, batch_size=16, validation_split=0.2)

# ============================================================
# 4. Fazer previsões
# ============================================================
previsoes_cnn = model1_cnn.predict(X_train)

# ============================================================
# 5. Reverter normalização
# ============================================================
previsoes_cnn_reais = scaler.inverse_transform(
    np.concatenate((X_train[:, -1, :], previsoes_cnn), axis=1)
)[:, -1]

# Criar DataFrame com previsões e valores reais
previsoes_cnn_df = pd.DataFrame({
    'Previsto': previsoes_cnn_reais,
    'Real': df1_cnn['ELET_CONS_CLT'].iloc[n_passo:].values
}, index=df1_cnn.index[n_passo:])

# ============================================================
# 6. Visualizar previsões
# ============================================================
fig = go.Figure()

# Real
fig.add_trace(go.Scatter(
    x=df1_cnn.index,
    y=df1_cnn['ELET_CONS_CLT'],
    mode='lines',
    name='Real',
    line=dict(color='blue')
))

# Previsto
fig.add_trace(go.Scatter(
    x=previsoes_cnn_df.index,
    y=previsoes_cnn_df['Previsto'],
    mode='lines',
    name='Previsto (CNN)',
    line=dict(color='red', dash='dash')
))

# Layout
fig.update_layout(
    title='Comparação: Consumo Real vs. Previsto (CNN)',
    xaxis_title='Data',
    yaxis_title='kWh/dia/clt',
    legend=dict(x=0, y=1),
    font=dict(size=14),
    title_x=0.5,
    width=1000,
    height=500
)

fig.show()
```

Mostrar saída oculta

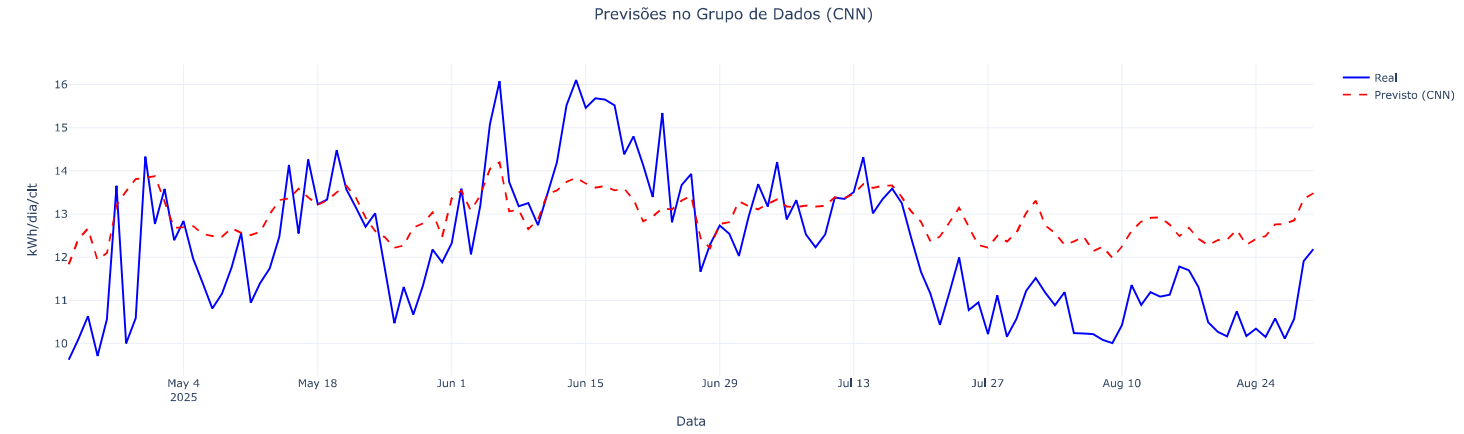
```
"""
# Criar modelo
model1_cnn = Sequential([
    Input(shape=(X_train.shape[1], X_train.shape[2])),
    Conv1D(filters=128, kernel_size=5, activation='relu'),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(1)
])

# Compilar (sempre antes do fit)
model1_cnn.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),
    loss='mse'
)

# Definir EarlyStopping
from tensorflow.keras.callbacks import EarlyStopping
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)

# Treinar
history_cnn = model1_cnn.fit(
    X_train, y_train,
    epochs=100,
    batch_size=16,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)
"""
```

```
'\n# Criar modelo\nmodel1_cnn = Sequential([\n    Input(shape=(X_train.shape[1], X_train.shape[2])),\n    Conv1D(filters=128, kernel_size=5, activation='rel
u'),\n    Dropout(0.2),\n    Dense(1)\n])\n\n# Compilar (sempre antes do fit)\nmodel1_cnn.compile(\n    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),\n    loss='mse'\n)\n\n# Definir EarlyStopping\nfro
m tensorflow.keras.callbacks import EarlyStopping\nearly_stop = EarlyStopping(\n    monitor='val_loss',\n    patience=10,\n    restore_best_weights=True\n)\n\n# Treinar\nhistory_cnn = model1_cnn.fit(\n    X_trai
n, y_train,\n    epochs=100,\n    batch_size=16,\n    validation_split=0.2,\n    callbacks=[early_stop],\n    verbose=1\n)\n'
```

```
# Garantir que o índice é datetime
df1_cnn.index = pd.to_datetime(df1_cnn.index)

# Selecionar subconjunto
df_previsao = df1_cnn.loc['2025-04-01':'2025-08-31']
```

```
# Normalizar com o mesmo scaler usado no treino
df_previsao_norm = scaler.transform(df_previsao.values)

# Criar sequências para este subconjunto
X_test, y_test = criar_sequencias(df_previsao_norm, n_passo)

# Fazer previsões com o modelo já treinado
y_pred = model1_cnn.predict(X_test)

# Reverter normalização para valores reais
y_pred_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_pred), axis=1)
)[:, -1]

y_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_test.reshape(-1,1)), axis=1)
)[:, -1]

# ============================================================
# 6. Visualizar resultados
# ============================================================

# Gerar índice correspondente ao último dia da janela usada para cada previsão
index_alinhado = df_previsao.index[n_passo - 1 : n_passo - 1 + len(y_pred_reais)]

# Garantir que os valores reais estejam alinhados com esse índice
real_values = df_previsao['ELET_CONS_CLT'].iloc[n_passo - 1 : n_passo - 1 + len(y_pred_reais)].values

# Plot
fig = go.Figure()
fig.add_trace(go.Scatter(
    x=index_alinhado, y=real_values,
    mode='lines', name='Real', line=dict(color='blue')
))
fig.add_trace(go.Scatter(
    x=index_alinhado, y=y_pred_reais,
    mode='lines', name='Previsto (CNN)', line=dict(color='red', dash='dash')
))
fig.update_layout(
    title="Previsões no Grupo de Dados (CNN)",
    xaxis_title="Data", yaxis_title="kWh/dia/clt",
    template="plotly_white", title_x=0.5
)
fig.show()
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
```



Validar previsões

```
# Avaliar modelo
df_resultados = avaliar_gru(
    modelo=model1_cnn,
    X_train=X_train, y_train=y_train,   # podes manter os originais
    X_test=X_test, y_test=y_test,       # aqui vai o subconjunto abril–agosto
    scaler=scaler,
    df_index=df_previsao.index,             # índice do subconjunto
    n_passo=n_passo,
    modelo_nome="CNN",
    iteracao=1
)
```

```
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
CNN (Iteração 1) - TREINO → MSE: 8.6086 | RMSE: 2.93 | MAE: 1.7113 | R²: 0.53
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
CNN (Iteração 1) - TESTE  → MSE: 1.9981 | RMSE: 1.41 | MAE: 1.1889 | R²: 0.19
```

```
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nResultados acumulados:")
print(df_resultados)
```

```
Resultados acumulados:
   Modelo  Iteração Conjunto     MSE  RMSE     MAE    R²
0     GRU         1   Treino  1.5272  1.24  0.8796  0.92
1     GRU         1    Teste  1.0344  1.02  0.7701  0.58
2     GRU         2   Treino  1.3627  1.17  0.8658  0.92
3     GRU         2    Teste  1.0475  1.02  0.8255  0.56
4     GRU         3   Treino  1.2581  1.12  0.7787  0.93
5     GRU         3    Teste  0.8839  0.94  0.7369  0.63
6     GRU         4   Treino  0.5170  0.72  0.5368  0.77
7     GRU         4    Teste  1.1309  1.06  0.8346  0.52
8     GRU         5   Treino  0.3215  0.57  0.4216  0.93
9     GRU         5    Teste  1.9958  1.41  1.1462  0.15
10    GRU         3   Treino  1.3724  1.17  0.8605  0.92
11    GRU         3    Teste  0.9088  0.95  0.7393  0.61
12    CNN         1   Treino  8.6086  2.93  1.7113  0.53
13    CNN         1    Teste  1.9981  1.41  1.1889  0.19
```

```
# Adicionar coluna de previsões (apenas nas datas alinhadas)
df_pred.loc[index_alinhado, "Previsto_CNN1"] = y_pred_reais

# Opcional: também podes guardar o erro
df_pred.loc[index_alinhado, "Erro_CNN1"] = ((df_pred["Previsto_CNN1"] - df_pred["ELET_CONS_CLT"])/df_pred["ELET_CONS_CLT"])*100
```

```
df_pred
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	previsão_xgb_f	erro_relativo_xgb_f	Previsto_GRU4	Erro_GRU4	Previsto_GRU5	Erro_GRU5	Previsto_GRU3
2023-03-30	18.750000	24.900000	13.8	11.099999	77.516667	91.000000	63.299999	27.700001	21.755307	5679.0	...	20.598866	7.365103	NaN	NaN	NaN	NaN	NaN
2023-03-31	18.333333	24.400000	14.1	10.299999	85.416667	100.000000	70.000000	30.000000	19.473651	11569.0	...	14.225427	-12.451346	NaN	NaN	NaN	NaN	NaN
2023-04-02	18.366667	22.100000	14.2	7.900001	74.466667	95.599998	44.400002	51.199997	22.624800	8496.0	...	11.925946	13.420016	NaN	NaN	NaN	NaN	NaN
2023-04-03	19.140000	25.299999	15.0	10.299999	55.960000	75.500000	44.500000	31.000000	25.191946	9342.0	...	10.838500	8.709854	NaN	NaN	NaN	NaN	NaN
2023-04-09	18.785715	26.700001	13.6	13.100000	74.428573	87.300003	59.900002	27.400002	22.498339	11715.0	...	10.455238	-15.394230	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2025-08-27	23.416990	31.900000	18.3	13.600000	75.882190	89.000000	51.500000	37.500000	23.980210	12914.0	...	9.953657	-1.573330	10.936665	8.147133	10.132778	0.197910	11.167550
2025-08-28	22.236250	30.200000	18.3	11.900000	80.148180	100.000000	54.700000	45.300000	22.372110	12826.0	...	9.920411	-6.101832	11.804976	11.735856	11.239166	6.380376	12.037146
2025-08-29	22.645070	34.000000	17.5	16.500000	79.591120	91.500000	54.400000	37.100000	22.824760	12985.0	...	10.535409	-11.562605	12.204058	2.444539	10.789237	-9.431897	12.813358
2025-08-30	24.656980	34.500000	19.7	14.800000	78.813560	92.000000	56.400000	35.600000	25.362570	12416.0	...	12.037998	-1.202318	12.560123	3.082843	11.493860	-5.668139	13.134220
2025-08-31	22.518880	30.100000	18.7	11.400000	83.646050	98.500000	59.900000	38.600000	22.132800	12372.0	...	11.249270	-5.710528	NaN	NaN	NaN	NaN	NaN

552 rows × 63 columns

Comece a programar ou gerar com a IA.

2ª iteração

```
df1_filtrado3.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change'],
      dtype='object')
```

```
# 🔍 Selecionar variáveis de entrada
features = [ 'T_mean', 'RH_mean', 'Calor_idx',
         'Q_OCP', 'OCP', 'AD', 'CR', 'CLTs',

         'dia_semana', 'trimestre', 'mes', 'dia_ano', 'dia_mes', 'semana_ano',


            'ELET_CONS_CLT'] # Aqui, o script seleciona um conjunto de variáveis de entrada — e a última coluna escolhida é ELET_CONS_CLT, o valor que se deseja prever.
df2_cnn = df1_filtrado3[features]
```

```
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Conv1D, Dense, Dropout, Flatten,BatchNormalization
import tensorflow as tf
import numpy as np
import pandas as pd

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
# ============================================================
# 1. Normalizar dados (X + y juntos)
# ============================================================
scaler = MinMaxScaler()
df2_normalizados = scaler.fit_transform(df2_cnn.values)

# ============================================================
# 2. Criar sequências temporais
# ============================================================
def criar_sequencias(df2_normalizados, n_passo):
    X, y = [], []
    for i in range(len(df2_normalizados) - n_passo):
        X.append(df2_normalizados[i:i+n_passo, :-1])  # features
        y.append(df2_normalizados[i+n_passo, -1])     # alvo (consumo)
    return np.array(X), np.array(y)

n_passo = 7
X_train, y_train = criar_sequencias(df2_normalizados, n_passo)

# ============================================================
# 3. Criar modelo CNN
# ============================================================
# Modelo CNN otimizado
model2_cnn = Sequential([
    Input(shape=(X_train.shape[1], X_train.shape[2])),

    Conv1D(filters=64, kernel_size=7, activation='gelu'),
    Flatten(),
    Dense(64, activation='gelu'),
    Dropout(0.1),
    Dense(1)
])

# 🔹 Compilar e treinar o modelo
model2_cnn.compile(optimizer='adam', loss='mse')
history2_cnn = model2_cnn.fit(X_train, y_train, epochs=200, batch_size=64)

# ============================================================
# 4. Fazer previsões
# ============================================================
previsoes2_cnn = model2_cnn.predict(X_train)

# ============================================================
# 5. Reverter normalização
# ============================================================
previsoes2_cnn_reais = scaler.inverse_transform(
    np.concatenate((X_train[:, -1, :], previsoes2_cnn), axis=1)
)[:, -1]

# Criar DataFrame com previsões e valores reais
previsoes2_cnn_df = pd.DataFrame({
    'Previsto': previsoes2_cnn_reais,
    'Real': df2_cnn['ELET_CONS_CLT'].iloc[n_passo:].values
}, index=df2_cnn.index[n_passo:])

# ============================================================
# 6. Visualizar previsões
```

```
# ============================================================
fig = go.Figure()

# Real
fig.add_trace(go.Scatter(
    x=df2_cnn.index,
    y=df2_cnn['ELET_CONS_CLT'],
    mode='lines',
    name='Real',
    line=dict(color='blue')
))

# Previsto
fig.add_trace(go.Scatter(
    x=previsoes2_cnn_df.index,
    y=previsoes2_cnn_df['Previsto'],
    mode='lines',
    name='Previsto (CNN)',
    line=dict(color='red', dash='dash')
))

# Layout
fig.update_layout(
    title='Comparação: Consumo Real vs. Previsto (CNN)',
    xaxis_title='Data',
    yaxis_title='kWh/dia/clt',
    legend=dict(x=0, y=1),
    font=dict(size=14),
    title_x=0.5,
    width=1000,
    height=500
)

fig.show()
```

Mostrar saída oculta

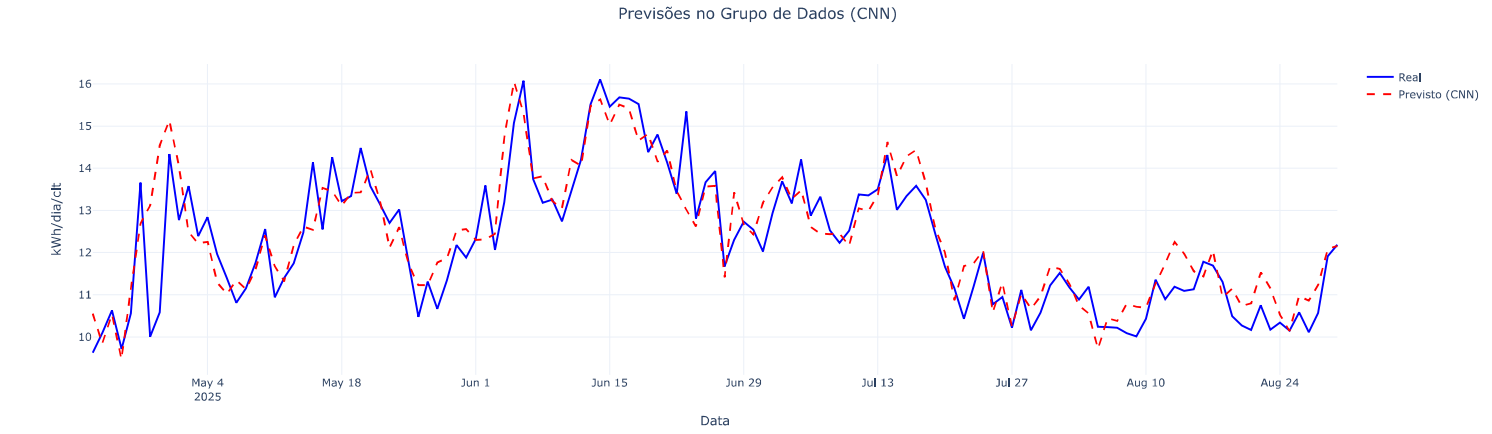
```
"""
# Criar modelo
model1_cnn = Sequential([
    Input(shape=(X_train.shape[1], X_train.shape[2])),
    Conv1D(filters=128, kernel_size=5, activation='relu'),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(1)
])

# Compilar (sempre antes do fit)
model1_cnn.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),
    loss='mse'
)

# Definir EarlyStopping
from tensorflow.keras.callbacks import EarlyStopping
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)

# Treinar
history_cnn = model1_cnn.fit(
    X_train, y_train,
    epochs=100,
    batch_size=16,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)
"""
```

```
'\n# Criar modelo\nmodel1_cnn = Sequential([\n    Input(shape=(X_train.shape[1], X_train.shape[2])),\n    Conv1D(filters=128, kernel_size=5, activation='relu'),\n    Flatten(),\n    Dense(256, activation='rel
u'),\n    Dropout(0.2),\n    Dense(1)\n])\n\n# Compilar (sempre antes do fit)\nmodel1_cnn.compile(\n    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),\n    loss='mse'\n)\n\n# Definir EarlyStopping\nfro
m tensorflow.keras.callbacks import EarlyStopping\nearly_stop = EarlyStopping(\n    monitor='val_loss',\n    patience=10,\n    restore_best_weights=True\n)\n\n# Treinar\nhistory_cnn = model1_cnn.fit(\n    X_trai
n, y_train,\n    epochs=100,\n    batch_size=16,\n    validation_split=0.2,\n    callbacks=[early_stop],\n    verbose=1\n)\n'
```

```
# Garantir que o índice é datetime
df2_cnn.index = pd.to_datetime(df2_cnn.index)

# Selecionar subconjunto
df_previsao2 = df2_cnn.loc['2025-04-01':'2025-08-31']


# Normalizar com o mesmo scaler usado no treino
df_previsao2_norm = scaler.transform(df_previsao2.values)

# Criar sequências para este subconjunto
X_test, y_test = criar_sequencias(df_previsao2_norm, n_passo)

# Fazer previsões com o modelo já treinado
y_pred = model2_cnn.predict(X_test)

# Reverter normalização para valores reais
y_pred_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_pred), axis=1)
)[:, -1]

y_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_test.reshape(-1,1)), axis=1)
)[:, -1]

# ============================================================
# 6. Visualizar resultados
# ============================================================

# Gerar índice correspondente ao último dia da janela usada para cada previsão
index_alinhado = df_previsao2.index[n_passo - 1 : n_passo - 1 + len(y_pred_reais)]

# Garantir que os valores reais estejam alinhados com esse índice
real_values = df_previsao2['ELET_CONS_CLT'].iloc[n_passo - 1 : n_passo - 1 + len(y_pred_reais)].values

# Plot
fig = go.Figure()
fig.add_trace(go.Scatter(
    x=index_alinhado, y=real_values,
    mode='lines', name='Real', line=dict(color='blue')
))
fig.add_trace(go.Scatter(
    x=index_alinhado, y=y_pred_reais,
    mode='lines', name='Previsto (CNN)', line=dict(color='red', dash='dash')
))
fig.update_layout(
    title="Previsões no Grupo de Dados (CNN)",
    xaxis_title="Data", yaxis_title="kWh/dia/clt",
    template="plotly_white", title_x=0.5
)
```

```
fig.show()
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
```



Validar previsões

```
# Avaliar modelo
df_resultados = avaliar_gru(
    modelo=model2_cnn,
    X_train=X_train, y_train=y_train,   # podes manter os originais
    X_test=X_test, y_test=y_test,       # aqui vai o subconjunto abril–agosto
    scaler=scaler,
    df_index=df_previsao2.index,             # índice do subconjunto
    n_passo=n_passo,
    modelo_nome="CNN",
    iteracao=2
)
```

```
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
CNN (Iteração 2) - TREINO → MSE: 1.4365 | RMSE: 1.20 | MAE: 0.8266 | R²: 0.92
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
CNN (Iteração 2) - TESTE  → MSE: 0.6896 | RMSE: 0.83 | MAE: 0.6368 | R²: 0.72
```

```
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nResultados acumulados:")
print(df_resultados)
```

```
Resultados acumulados:
   Modelo  Iteração Conjunto     MSE  RMSE     MAE    R²
0     GRU         1   Treino  1.5272  1.24  0.8796  0.92
1     GRU         1    Teste  1.0344  1.02  0.7701  0.58
2     GRU         2   Treino  1.3627  1.17  0.8658  0.92
3     GRU         2    Teste  1.0475  1.02  0.8255  0.56
4     GRU         3   Treino  1.2581  1.12  0.7787  0.93
5     GRU         3    Teste  0.8839  0.94  0.7369  0.63
6     GRU         4   Treino  0.5170  0.72  0.5368  0.77
7     GRU         4    Teste  1.1309  1.06  0.8346  0.52
8     GRU         5   Treino  0.3215  0.57  0.4216  0.93
9     GRU         5    Teste  1.9958  1.41  1.1462  0.15
10    GRU         3   Treino  1.3724  1.17  0.8605  0.92
11    GRU         3    Teste  0.9088  0.95  0.7393  0.61
12    CNN         1   Treino  8.6086  2.93  1.7113  0.53
13    CNN         1    Teste  1.9981  1.41  1.1889  0.19
14    CNN         2   Treino  1.4365  1.20  0.8266  0.92
15    CNN         2    Teste  0.6896  0.83  0.6368  0.72
```

```
# Adicionar coluna de previsões (apenas nas datas alinhadas)
df_pred.loc[index_alinhado, "Previsto_CNN2"] = y_pred_reais

# Opcional: também podes guardar o erro
df_pred.loc[index_alinhado, "Erro_CNN2"] = ((df_pred["Previsto_CNN2"] - df_pred["ELET_CONS_CLT"])/df_pred["ELET_CONS_CLT"])*100
```

```
df_pred
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	Previsto_GRU4	Erro_GRU4	Previsto_GRU5	Erro_GRU5	Previsto_GRU3	Erro_GRU3	Previsto_CNN1	Erro_CNN1
2023-03-30	18.750000	24.900000	13.8	11.099999	77.516667	91.000000	63.299999	27.700001	21.755307	5679.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2023-03-31	18.333333	24.400000	14.1	10.299999	85.416667	100.000000	70.000000	30.000000	19.473651	11569.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2023-04-02	18.366667	22.100000	14.2	7.900001	74.466667	95.599998	44.400002	51.199997	22.624800	8496.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2023-04-03	19.140000	25.299999	15.0	10.299999	55.960000	75.500000	44.500000	31.000000	25.191946	9342.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2023-04-09	18.785715	26.700001	13.6	13.100000	74.428573	87.300003	59.900002	27.400002	22.498339	11715.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2025-08-27	23.416990	31.900000	18.3	13.600000	75.882190	89.000000	51.500000	37.500000	23.980210	12914.0	...	10.936665	8.147133	10.132778	0.197910	11.167550	10.430246	12.768323	26.259473
2025-08-28	22.236250	30.200000	18.3	11.900000	80.148180	100.000000	54.700000	45.300000	22.372110	12826.0	...	11.804976	11.735856	11.239166	6.380376	12.037146	13.933380	12.854179	21.666723
2025-08-29	22.645070	34.000000	17.5	16.500000	79.591120	91.500000	54.400000	37.100000	22.824760	12985.0	...	12.204058	2.444539	10.789237	-9.431897	12.813358	7.559188	13.347230	12.040670
2025-08-30	24.656980	34.500000	19.7	14.800000	78.813560	92.000000	56.400000	35.600000	25.362570	12416.0	...	12.560123	3.082843	11.493860	-5.668139	13.134220	7.794544	13.482678	10.654392
2025-08-31	22.518880	30.100000	18.7	11.400000	83.646050	98.500000	59.900000	38.600000	22.132800	12372.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

552 rows × 65 columns

```
Comece a programar ou gerar com a IA.
```

3ª iteração

```
df1_filtrado3.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
```

```
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change'],
      dtype='object')
```

```
# 🔍 Selecionar variáveis de entrada
features = [ 'T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx',
         'Q_OCP', 'OCP', 'AD', 'CR', 'CLTs',

         'dia_semana', 'trimestre', 'mes', 'dia_ano', 'dia_mes', 'semana_ano',
             'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change',


           'ELET_CONS_CLT'] # Aqui, o script seleciona um conjunto de variáveis de entrada — e a última coluna escolhida é ELET_CONS_CLT, o valor que se deseja prever.
df3_cnn = df1_filtrado3[features]
```

```
Comece a programar ou gerar com a IA.
```

```
df3_cnn = df3_cnn.replace([np.inf, -np.inf], np.nan)

# Opção 1: Preencher com a média de cada coluna
#df3_cnn = df3_cnn.fillna(df3_cnn.mean())

# Opção 2: Remover linhas com NaN (se não forem muitas)
df3_cnn = df3_cnn.dropna()
```

```
df3_cnn = df3_cnn.astype('float64')
```

```
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Conv1D, Dense, Dropout, Flatten,BatchNormalization
import tensorflow as tf
import numpy as np
import pandas as pd

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
# ============================================================
# 1. Normalizar dados (X + y juntos)
# ============================================================
scaler = MinMaxScaler()
df3_normalizados = scaler.fit_transform(df3_cnn.values)

# ============================================================
# 2. Criar sequências temporais
# ============================================================
def criar_sequencias(df3_normalizados, n_passo):
    X, y = [], []
    for i in range(len(df3_normalizados) - n_passo):
        X.append(df3_normalizados[i:i+n_passo, :-1])  # features
        y.append(df3_normalizados[i+n_passo, -1])     # alvo (consumo)
    return np.array(X), np.array(y)

n_passo = 7
X_train, y_train = criar_sequencias(df3_normalizados, n_passo)

# ============================================================
# 3. Criar modelo CNN
# ============================================================
# Modelo CNN otimizado
model3_cnn = Sequential([
    Input(shape=(X_train.shape[1], X_train.shape[2])),

    Conv1D(filters=64, kernel_size=7, activation='gelu'),
    Flatten(),
    Dense(64, activation='gelu'),
    Dropout(0.1),
    Dense(1)
])

# 🔹 Compilar e treinar o modelo
model3_cnn.compile(optimizer='adam', loss='mse')
history3_cnn = model3_cnn.fit(X_train, y_train, epochs=200, batch_size=64)

# ============================================================
# 4. Fazer previsões
# ============================================================
previsoes3_cnn = model3_cnn.predict(X_train)

# ============================================================
# 5. Reverter normalização
# ============================================================
previsoes3_cnn_reais = scaler.inverse_transform(
    np.concatenate((X_train[:, -1, :], previsoes3_cnn), axis=1)
)[:, -1]

# Criar DataFrame com previsões e valores reais
previsoes3_cnn_df = pd.DataFrame({
    'Previsto': previsoes3_cnn_reais,
    'Real': df3_cnn['ELET_CONS_CLT'].iloc[n_passo:].values
}, index=df3_cnn.index[n_passo:])

# ============================================================
# 6. Visualizar previsões
# ============================================================
fig = go.Figure()

# Real
fig.add_trace(go.Scatter(
    x=df3_cnn.index,
    y=df3_cnn['ELET_CONS_CLT'],
    mode='lines',
    name='Real',
    line=dict(color='blue')
))

# Previsto
fig.add_trace(go.Scatter(
    x=previsoes3_cnn_df.index,
    y=previsoes3_cnn_df['Previsto'],
    mode='lines',
    name='Previsto (CNN)',
    line=dict(color='red', dash='dash')
))

# Layout
fig.update_layout(
    title='Comparação: Consumo Real vs. Previsto (CNN)',
    yaxis_title='Data'
```

```
    yaxis_title='kWh/dia/clt',
    legend=dict(x=0, y=1),
    font=dict(size=14),
    title_x=0.5,
    width=1000,
    height=500
)

fig.show()
```

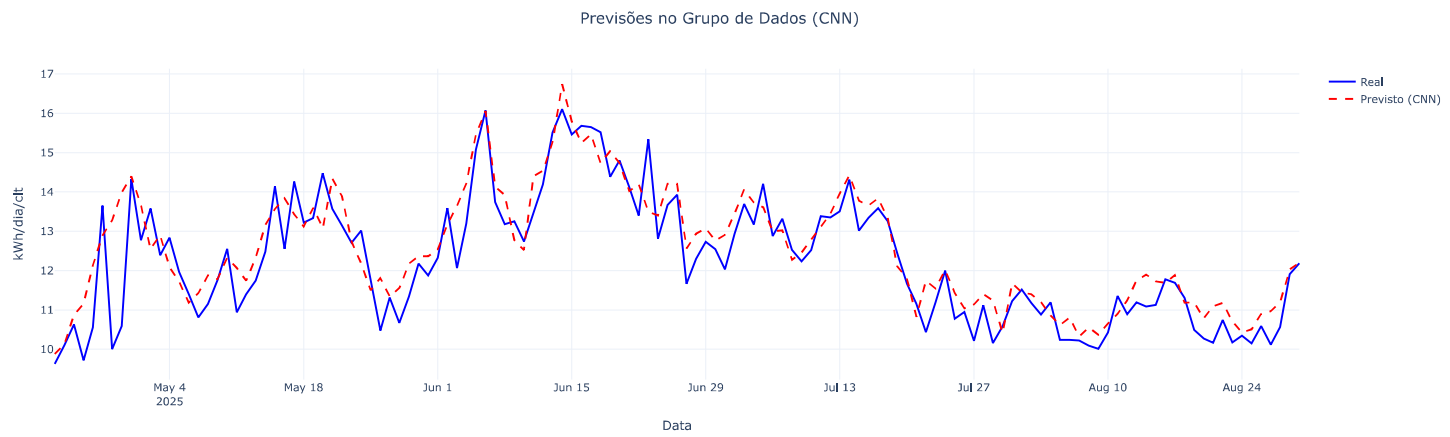
Mostrar saída oculta

```
"""
# Criar modelo
model1_cnn = Sequential([
    Input(shape=(X_train.shape[1], X_train.shape[2])),
    Conv1D(filters=128, kernel_size=5, activation='relu'),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(1)
])

# Compilar (sempre antes do fit)
model1_cnn.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),
    loss='mse'
)

# Definir EarlyStopping
from tensorflow.keras.callbacks import EarlyStopping
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)

# Treinar
history_cnn = model1_cnn.fit(
    X_train, y_train,
    epochs=100,
    batch_size=16,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)
"""
```

```
'\n# Criar modelo\nmodel1_cnn = Sequential([\n    Input(shape=(X_train.shape[1], X_train.shape[2])),\n    Conv1D(filters=128, kernel_size=5, activation='relu'),\n    Flatten(),\n    Dense(256, activation='rel
u'),\n    Dropout(0.2),\n    Dense(1)\n])\n\n# Compilar (sempre antes do fit)\nmodel1_cnn.compile(\n    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),\n    loss='mse'\n)\n\n# Definir EarlyStopping\nfro
m tensorflow.keras.callbacks import EarlyStopping\nearly_stop = EarlyStopping(\n    monitor='val_loss',\n    patience=10,\n    restore_best_weights=True\n)\n\n# Treinar\nhistory_cnn = model1_cnn.fit(\n    X_trai
n, y_train,\n    epochs=100,\n    batch_size=16,\n    validation_split=0.2,\n    callbacks=[early_stop],\n    verbose=1\n)\n'
```

```
# Garantir que o índice é datetime
df3_cnn.index = pd.to_datetime(df3_cnn.index)

# Selecionar subconjunto
df_previsao3 = df3_cnn.loc['2025-04-01':'2025-08-31']


# Normalizar com o mesmo scaler usado no treino
df_previsao3_norm = scaler.transform(df_previsao3.values)

# Criar sequências para este subconjunto
X_test, y_test = criar_sequencias(df_previsao3_norm, n_passo)

# Fazer previsões com o modelo já treinado
y_pred = model3_cnn.predict(X_test)

# Reverter normalização para valores reais
y_pred_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_pred), axis=1)
)[:, -1]

y_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_test.reshape(-1,1)), axis=1)
)[:, -1]

# ============================================================
# 6. Visualizar resultados
# ============================================================

# Gerar índice correspondente ao último dia da janela usada para cada previsão
index_alinhado = df_previsao3.index[n_passo - 1 : n_passo - 1 + len(y_pred_reais)]

# Garantir que os valores reais estejam alinhados com esse índice
real_values = df_previsao3['ELET_CONS_CLT'].iloc[n_passo - 1 : n_passo - 1 + len(y_pred_reais)].values

# Plot
fig = go.Figure()
fig.add_trace(go.Scatter(
    x=index_alinhado, y=real_values,
    mode='lines', name='Real', line=dict(color='blue')
))
fig.add_trace(go.Scatter(
    x=index_alinhado, y=y_pred_reais,
    mode='lines', name='Previsto (CNN)', line=dict(color='red', dash='dash')
))
fig.update_layout(
    title="Previsões no Grupo de Dados (CNN)",
    xaxis_title="Data", yaxis_title="kWh/dia/clt",
    template="plotly_white", title_x=0.5
)
fig.show()
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
```



Validar previsões

```
# Avaliar modelo
df_resultados = avaliar_gru(
    modelo=model3_cnn,
    X_train=X_train, y_train=y_train,   # podes manter os originais
    X_test=X_test, y_test=y_test,       # aqui vai o subconjunto abril-agosto
    scaler=scaler,
    df_index=df_previsao3.index,             # índice do subconjunto
    n_passo=n_passo,
    modelo_nome="CNN",
    iteracao=3
)
```

```
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
CNN (Iteração 3) - TREINO → MSE: 1.1868 | RMSE: 1.09 | MAE: 0.7599 | R²: 0.94
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
CNN (Iteração 3) - TESTE  → MSE: 0.6066 | RMSE: 0.78 | MAE: 0.5809 | R²: 0.75
```

```
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nResultados acumulados:")
print(df_resultados)
```

```
Resultados acumulados:
   Modelo  Iteração Conjunto     MSE  RMSE     MAE    R²
0     GRU         1   Treino  1.5272  1.24  0.8796  0.92
1     GRU         1    Teste  1.0344  1.02  0.7701  0.58
2     GRU         2   Treino  1.3627  1.17  0.8658  0.92
3     GRU         2    Teste  1.0475  1.02  0.8255  0.56
4     GRU         3   Treino  1.2581  1.12  0.7787  0.93
5     GRU         3    Teste  0.8839  0.94  0.7369  0.63
6     GRU         4   Treino  0.5170  0.72  0.5368  0.77
7     GRU         4    Teste  1.1309  1.06  0.8346  0.52
8     GRU         5   Treino  0.3215  0.57  0.4216  0.93
9     GRU         5    Teste  1.9958  1.41  1.1462  0.15
10    GRU         3   Treino  1.3724  1.17  0.8605  0.92
11    GRU         3    Teste  0.9088  0.95  0.7393  0.61
12    CNN         1   Treino  8.6086  2.93  1.7113  0.53
13    CNN         1    Teste  1.9981  1.41  1.1889  0.19
14    CNN         2   Treino  1.4365  1.20  0.8266  0.92
15    CNN         2    Teste  0.6896  0.83  0.6368  0.72
16    CNN         3   Treino  1.1868  1.09  0.7599  0.94
17    CNN         3    Teste  0.6066  0.78  0.5809  0.75
```

```
# Adicionar coluna de previsões (apenas nas datas alinhadas)
df_pred.loc[index_alinhado, "Previsto_CNN3"] = y_pred_reais

# Opcional: também podes guardar o erro
df_pred.loc[index_alinhado, "Erro_CNN3"] = ((df_pred["Previsto_CNN3"] - df_pred["ELET_CONS_CLT"])/df_pred["ELET_CONS_CLT"])*100
```

```
df_pred
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	Previsto_GRU5	Erro_GRU5	Previsto_GRU3	Erro_GRU3	Previsto_CNN1	Erro_CNN1	Previsto_CNN2	Erro_CNN2
2023-03-30	18.750000	24.900000	13.8	11.099999	77.516667	91.000000	63.299999	27.700001	21.755307	5679.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2023-03-31	18.333333	24.400000	14.1	10.299999	85.416667	100.000000	70.000000	30.000000	19.473651	11569.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2023-04-02	18.366667	22.100000	14.2	7.900001	74.466667	95.599998	44.400002	51.199997	22.624800	8496.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2023-04-03	19.140000	25.299999	15.0	10.299999	55.960000	75.500000	44.500000	31.000000	25.191946	9342.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2023-04-09	18.785715	26.700001	13.6	13.100000	74.428573	87.300003	59.900002	27.400002	22.498339	11715.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2025-08-27	23.416990	31.900000	18.3	13.600000	75.882190	89.000000	51.500000	37.500000	23.980210	12914.0	...	10.132778	0.197910	11.167550	10.430246	12.768323	26.259473	10.860293	7.391934
2025-08-28	22.236250	30.200000	18.3	11.900000	80.148180	100.000000	54.700000	45.300000	22.372110	12826.0	...	11.239166	6.380376	12.037146	13.933380	12.854179	21.666723	11.236564	6.355749
2025-08-29	22.645070	34.000000	17.5	16.500000	79.591120	91.500000	54.400000	37.100000	22.824760	12985.0	...	10.789237	-9.431897	12.813358	7.559188	13.347230	12.040670	12.092545	1.508461
2025-08-30	24.656980	34.500000	19.7	14.800000	78.813560	92.000000	56.400000	35.600000	25.362570	12416.0	...	11.493860	-5.668139	13.134220	7.794544	13.482678	10.654392	12.155922	-0.234499
2025-08-31	22.518880	30.100000	18.7	11.400000	83.646050	98.500000	59.900000	38.600000	22.132800	12372.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

552 rows × 67 columns

```
Comece a programar ou gerar com a IA.
```

4ª iteração

```
df1_filtrado3.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
```

```
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change'],
      dtype='object')
```

```
# 🔍 Selecionar variáveis de entrada
features = [ 'T_mean', 'T_max', 'T_min', 'RH_mean', 'RH_max', 'RH_min', 'Calor_idx',
         'Q_OCP', 'OCP', 'AD', 'CR', 'CLTs',
            'EP.C', '50S.C', 'COZ.50S.C', 'COZ.EP.C', 'PST.C', 'LAV.C',

         'dia_semana', 'trimestre', 'mes', 'dia_ano', 'dia_mes', 'semana_ano',

            'CO2e_ELET','Total_CO2e', 'CO2e_QOCP',

           'ELET_CONS_CLT_lag1', 'ELET_CONS_CLT_lag7',
       'ELET_CONS_CLT_rolling_mean', 'ELET_CONS_CLT_rolling_median',
       'ELET_CONS_CLT_rolling_std', 'ELET_CONS_CLT_error_median',
       'ELET_CONS_CLT_pct_change',

            'ELET_CONS_CLT'] # Aqui, o script seleciona um conjunto de variáveis de entrada — e a última coluna escolhida é ELET_CONS_CLT, o valor que se deseja prever.
df4_cnn = df1_filtrado3[features]
```

```
Comece a programar ou gerar com a IA.
```

```
df4_cnn = df4_cnn.replace([np.inf, -np.inf], np.nan)

# Opção 1: Preencher com a média de cada coluna
#df4_cnn = df4_cnn.fillna(df4_cnn.mean())

# Opção 2: Remover linhas com NaN (se não forem muitas)
df4_cnn = df4_cnn.dropna()
```

```
df4_cnn = df4_cnn.astype('float64')
```

```
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Conv1D, Dense, Dropout, Flatten,BatchNormalization
import tensorflow as tf
import numpy as np
import pandas as pd

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
# ============================================================
# 1. Normalizar dados (X + y juntos)
# ============================================================
scaler = MinMaxScaler()
df4_normalizados = scaler.fit_transform(df4_cnn.values)

# ============================================================
# 2. Criar sequências temporais
# ============================================================
def criar_sequencias(df4_normalizados, n_passo):
    X, y = [], []
    for i in range(len(df4_normalizados) - n_passo):
        X.append(df4_normalizados[i:i+n_passo, :-1])  # features
        y.append(df4_normalizados[i+n_passo, -1])     # alvo (consumo)
    return np.array(X), np.array(y)

n_passo = 7
X_train, y_train = criar_sequencias(df4_normalizados, n_passo)

# ============================================================
# 3. Criar modelo CNN
# ============================================================
# Modelo CNN otimizado
model4_cnn = Sequential([
    Input(shape=(X_train.shape[1], X_train.shape[2])),

    Conv1D(filters=64, kernel_size=7, activation='gelu'),
    Flatten(),
    Dense(64, activation='gelu'),
    Dropout(0.1),
    Dense(1)
])

# 🔹 Compilar e treinar o modelo
model4_cnn.compile(optimizer='adam', loss='mse')
history4_cnn = model4_cnn.fit(X_train, y_train, epochs=200, batch_size=64)

# ============================================================
# 4. Fazer previsões
# ============================================================
previsoes4_cnn = model4_cnn.predict(X_train)

# ============================================================
# 5. Reverter normalização
# ============================================================
previsoes4_cnn_reais = scaler.inverse_transform(
    np.concatenate((X_train[:, -1, :], previsoes4_cnn), axis=1)
)[:, -1]

# Criar DataFrame com previsões e valores reais
previsoes4_cnn_df = pd.DataFrame({
    'Previsto': previsoes4_cnn_reais,
    'Real': df4_cnn['ELET_CONS_CLT'].iloc[n_passo:].values
}, index=df4_cnn.index[n_passo:])

# ============================================================
# 6. Visualizar previsões
# ============================================================
fig = go.Figure()

# Real
fig.add_trace(go.Scatter(
    x=df4_cnn.index,
    y=df4_cnn['ELET_CONS_CLT'],
    mode='lines',
    name='Real',
    line=dict(color='blue')
))

# Previsto
fig.add_trace(go.Scatter(
    x=previsoes4_cnn_df.index,
    y=previsoes4_cnn_df['Previsto'],
    mode='lines',
    name='Previsto (CNN)',
    line=dict(color='red', dash='dash')
))

# Layout
fig.update_layout(
```

```
    title='Comparação: Consumo Real vs. Previsto (CNN)',
    xaxis_title='Data',
    yaxis_title='kWh/dia/clt',
    legend=dict(x=0, y=1),
    font=dict(size=14),
    title_x=0.5,
    width=1000,
    height=500
)

fig.show()
```

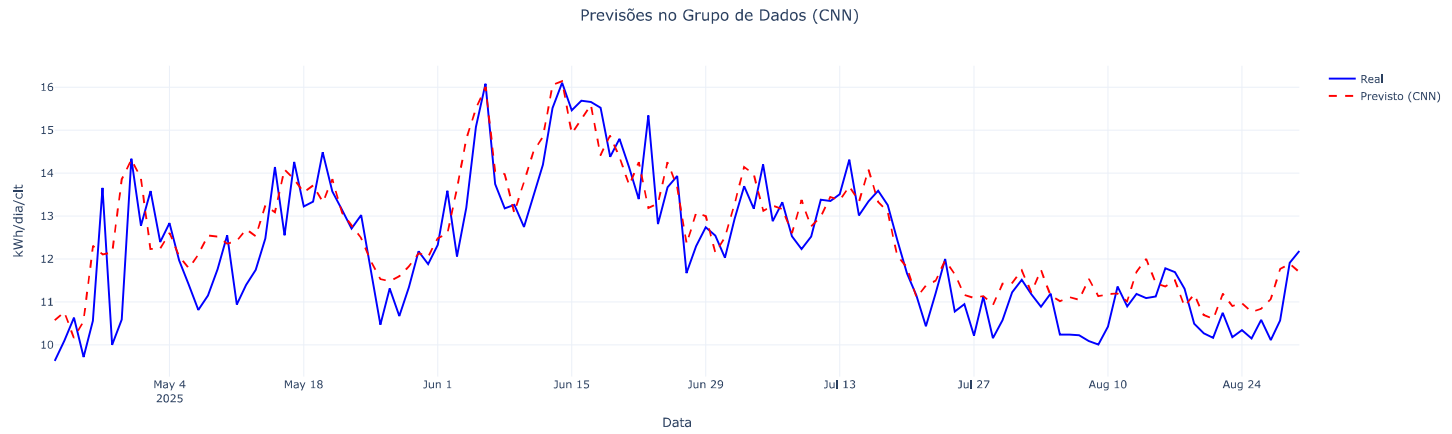
Mostrar saída oculta

```
"""
# Criar modelo
model1_cnn = Sequential([
    Input(shape=(X_train.shape[1], X_train.shape[2])),
    Conv1D(filters=128, kernel_size=5, activation='relu'),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(1)
])

# Compilar (sempre antes do fit)
model1_cnn.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),
    loss='mse'
)

# Definir EarlyStopping
from tensorflow.keras.callbacks import EarlyStopping
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)

# Treinar
history_cnn = model1_cnn.fit(
    X_train, y_train,
    epochs=100,
    batch_size=16,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)
"""
```

```
'\n# Criar modelo\nmodel1_cnn = Sequential([\n    Input(shape=(X_train.shape[1], X_train.shape[2])),\n    Conv1D(filters=128, kernel_size=5, activation='relu'),\n    Flatten(),\n    Dense(256, activation='rel
u'),\n    Dropout(0.2),\n    Dense(1)\n])\n\n# Compilar (sempre antes do fit)\nmodel1_cnn.compile(\n    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),\n    loss='mse'\n)\n\n# Definir EarlyStopping\nfro
m tensorflow.keras.callbacks import EarlyStopping\nearly_stop = EarlyStopping(\n    monitor='val_loss',\n    patience=10,\n    restore_best_weights=True\n)\n\n# Treinar\nhistory_cnn = model1_cnn.fit(\n    X_trai
n, y_train,\n    epochs=100,\n    batch_size=16,\n    validation_split=0.2,\n    callbacks=[early_stop],\n    verbose=1\n)\n'
```

```
# Garantir que o índice é datetime
df4_cnn.index = pd.to_datetime(df4_cnn.index)

# Selecionar subconjunto
df_previsao4 = df4_cnn.loc['2025-04-01':'2025-08-31']


# Normalizar com o mesmo scaler usado no treino
df_previsao4_norm = scaler.transform(df_previsao4.values)

# Criar sequências para este subconjunto
X_test, y_test = criar_sequencias(df_previsao4_norm, n_passo)

# Fazer previsões com o modelo já treinado
y_pred = model4_cnn.predict(X_test)

# Reverter normalização para valores reais
y_pred_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_pred), axis=1)
)[:, -1]

y_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_test.reshape(-1,1)), axis=1)
)[:, -1]

# ============================================================
# 6. Visualizar resultados
# ============================================================

# Gerar índice correspondente ao último dia da janela usada para cada previsão
index_alinhado = df_previsao4.index[n_passo - 1 : n_passo - 1 + len(y_pred_reais)]

# Garantir que os valores reais estejam alinhados com esse índice
real_values = df_previsao4['ELET_CONS_CLT'].iloc[n_passo - 1 : n_passo - 1 + len(y_pred_reais)].values

# Plot
fig = go.Figure()
fig.add_trace(go.Scatter(
    x=index_alinhado, y=real_values,
    mode='lines', name='Real', line=dict(color='blue')
))
fig.add_trace(go.Scatter(
    x=index_alinhado, y=y_pred_reais,
    mode='lines', name='Previsto (CNN)', line=dict(color='red', dash='dash')
))
fig.update_layout(
    title="Previsões no Grupo de Dados (CNN)",
    xaxis_title="Data", yaxis_title="kWh/dia/clt",
    template="plotly_white", title_x=0.5
)
fig.show()
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```



Validar previsões

```
# Avaliar modelo
df_resultados = avaliar_gru(
    modelo=model4_cnn,
    X_train=X_train, y_train=y_train,   # podes manter os originais
    X_test=X_test, y_test=y_test,       # aqui vai o subconjunto abril–agosto
    scaler=scaler,
    df_index=df_previsao4.index,             # índice do subconjunto
    n_passo=n_passo,
    modelo_nome="CNN",
    iteracao=4
)
```

```
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
CNN (Iteração 4) - TREINO → MSE: 0.6948 | RMSE: 0.83 | MAE: 0.6288 | R²: 0.96
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
CNN (Iteração 4) - TESTE  → MSE: 0.5022 | RMSE: 0.71 | MAE: 0.5626 | R²: 0.80
```

```
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nResultados acumulados:")
print(df_resultados)
```

Mostrar saída oculta

```
# Adicionar coluna de previsões (apenas nas datas alinhadas)
df_pred.loc[index_alinhado, "Previsto_CNN4"] = y_pred_reais

# Opcional: também podes guardar o erro
df_pred.loc[index_alinhado, "Erro_CNN4"] = ((df_pred["Previsto_CNN4"] - df_pred["ELET_CONS_CLT"])/df_pred["ELET_CONS_CLT"])*100
```

```
df_pred
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	Previsto_GRU3	Erro_GRU3	Previsto_CNN1	Erro_CNN1	Previsto_CNN2	Erro_CNN2	Previsto_CNN3	Erro_CNN3
2023-03-30	18.750000	24.900000	13.8	11.099999	77.516667	91.000000	63.299999	27.700001	21.755307	5679.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2023-03-31	18.333333	24.400000	14.1	10.299999	85.416667	100.000000	70.000000	30.000000	19.473651	11569.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2023-04-02	18.366667	22.100000	14.2	7.900001	74.466667	95.599998	44.400002	51.199997	22.624800	8496.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2023-04-03	19.140000	25.299999	15.0	10.299999	55.960000	75.500000	44.500000	31.000000	25.191946	9342.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2023-04-09	18.785715	26.700001	13.6	13.100000	74.428573	87.300003	59.900002	27.400002	22.498339	11715.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2025-08-27	23.416990	31.900000	18.3	13.600000	75.882190	89.000000	51.500000	37.500000	23.980210	12914.0	...	11.167550	10.430246	12.768323	26.259473	10.860293	7.391934	10.964200	8.419413
2025-08-28	22.236250	30.200000	18.3	11.900000	80.148180	100.000000	54.700000	45.300000	22.372110	12826.0	...	12.037146	13.933380	12.854179	21.666723	11.236564	6.355749	11.207939	6.084811
2025-08-29	22.645070	34.000000	17.5	16.500000	79.591120	91.500000	54.400000	37.100000	22.824760	12985.0	...	12.813358	7.559188	13.347230	12.040670	12.092545	1.508461	12.030809	0.990235
2025-08-30	24.656980	34.500000	19.7	14.800000	78.813560	92.000000	56.400000	35.600000	25.362570	12416.0	...	13.134220	7.794544	13.482678	10.654392	12.155922	-0.234499	12.203667	0.157351
2025-08-31	22.518880	30.100000	18.7	11.400000	83.646050	98.500000	59.900000	38.600000	22.132800	12372.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

552 rows × 69 columns

Comece a programar ou gerar com a IA.

5ª iteração

```
df1_filtrado3.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change'],
      dtype='object')
```

```
# 🔍 Selecionar variáveis de entrada
features = [ 'T_mean', 'T_max', 'T_min', 'RH_mean', 'RH_max', 'RH_min', 'Calor_idx',
             'Q_OCP', 'OCP', 'AD', 'CR', 'CLTs',


        'dia_semana', 'trimestre', 'mes', 'dia_ano', 'dia_mes', 'semana_ano',
```

```
            'ELET_CONS_CLT'] # Aqui, o script seleciona um conjunto de variáveis de entrada — e a última coluna escolhida é ELET_CONS_CLT, o valor que se deseja prever.
df5_cnn = df1_filtrado3[features]
```

Comece a programar ou gerar com a IA.

```
df5_cnn = df5_cnn.replace([np.inf, -np.inf], np.nan)

# Opção 1: Preencher com a média de cada coluna
#df5_cnn = df5_cnn.fillna(df5_cnn.mean())

# Opção 2: Remover linhas com NaN (se não forem muitas)
df5_cnn = df5_cnn.dropna()
```

```
df5_cnn = df5_cnn.astype('float64')
```

```
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Conv1D, Dense, Dropout, Flatten,BatchNormalization
import tensorflow as tf
import numpy as np
import pandas as pd

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
# ============================================================
# 1. Normalizar dados (X + y juntos)
# ============================================================
scaler = MinMaxScaler()
df5_normalizados = scaler.fit_transform(df5_cnn.values)

# ============================================================
# 2. Criar sequências temporais
# ============================================================
def criar_sequencias(df5_normalizados, n_passo):
    X, y = [], []
    for i in range(len(df5_normalizados) - n_passo):
        X.append(df5_normalizados[i:i+n_passo, :-1])  # features
        y.append(df5_normalizados[i+n_passo, -1])     # alvo (consumo)
    return np.array(X), np.array(y)

n_passo = 7
X_train, y_train = criar_sequencias(df5_normalizados, n_passo)

# ============================================================
# 3. Criar modelo CNN
# ============================================================
# Modelo CNN otimizado
model5_cnn = Sequential([
    Input(shape=(X_train.shape[1], X_train.shape[2])),

    Conv1D(filters=64, kernel_size=7, activation='gelu'),
    Flatten(),
    Dense(64, activation='gelu'),
    Dropout(0.1),
    Dense(1)
])

# 🔹 Compilar e treinar o modelo
model5_cnn.compile(optimizer='adam', loss='mse')
history5_cnn = model5_cnn.fit(X_train, y_train, epochs=200, batch_size=64)

# ============================================================
# 4. Fazer previsões
# ============================================================
previsoes5_cnn = model5_cnn.predict(X_train)

# ============================================================
# 5. Reverter normalização
# ============================================================
previsoes5_cnn_reais = scaler.inverse_transform(
    np.concatenate((X_train[:, -1, :], previsoes5_cnn), axis=1)
)[:, -1]

# Criar DataFrame com previsões e valores reais
previsoes5_cnn_df = pd.DataFrame({
    'Previsto': previsoes5_cnn_reais,
    'Real': df5_cnn['ELET_CONS_CLT'].iloc[n_passo:].values
}, index=df5_cnn.index[n_passo:])

# ============================================================
# 6. Visualizar previsões
# ============================================================
fig = go.Figure()

# Real
fig.add_trace(go.Scatter(
    x=df5_cnn.index,
    y=df5_cnn['ELET_CONS_CLT'],
    mode='lines',
    name='Real',
    line=dict(color='blue')
))

# Previsto
fig.add_trace(go.Scatter(
    x=previsoes5_cnn_df.index,
    y=previsoes5_cnn_df['Previsto'],
    mode='lines',
    name='Previsto (CNN)',
    line=dict(color='red', dash='dash')
))

# Layout
fig.update_layout(
    title='Comparação: Consumo Real vs. Previsto (CNN)',
    xaxis_title='Data',
    yaxis_title='kWh/dia/clt',
    legend=dict(x=0, y=1),
    font=dict(size=14),
    title_x=0.5,
    width=1000,
    height=500
)

fig.show()
```

Mostrar saída oculta

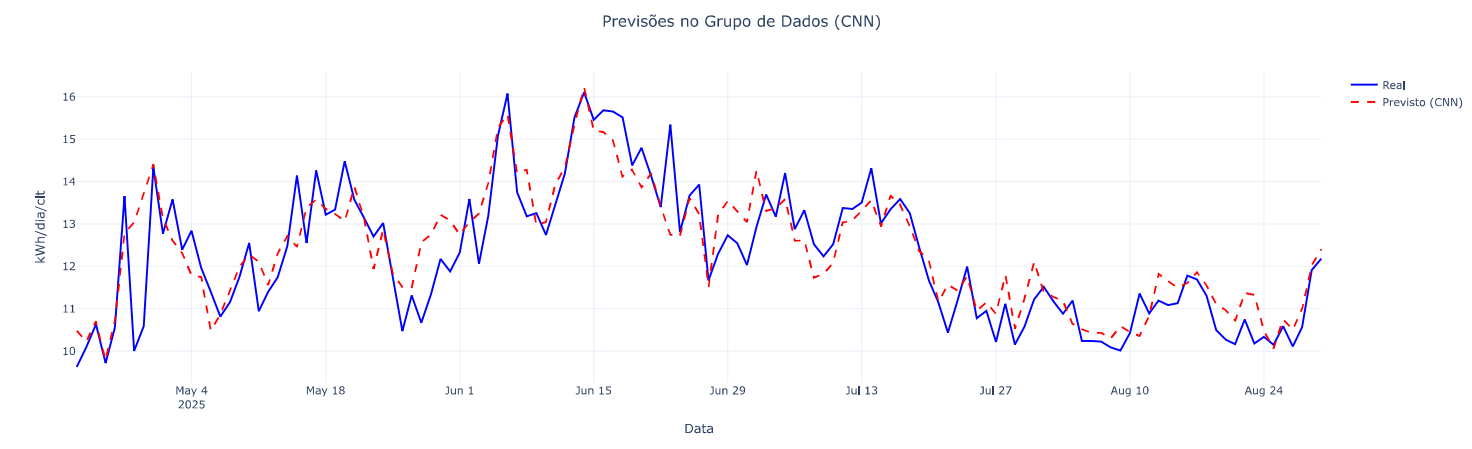
```
"""
# Criar modelo
model1_cnn = Sequential([
    Input(shape=(X_train.shape[1], X_train.shape[2])),
    Conv1D(filters=128, kernel_size=5, activation='relu'),
    Flatten(),
```

```
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(1)
])

# Compilar (sempre antes do fit)
model1_cnn.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),
    loss='mse'
)

# Definir EarlyStopping
from tensorflow.keras.callbacks import EarlyStopping
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)

# Treinar
history_cnn = model1_cnn.fit(
    X_train, y_train,
    epochs=100,
    batch_size=16,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)
"""
```

```
'\n# Criar modelo\nmodel1_cnn = Sequential([\n    Input(shape=(X_train.shape[1], X_train.shape[2])),\n    Conv1D(filters=128, kernel_size=5, activation='relu'),\n    Flatten(),\n    Dense(256, activation='rel
u'),\n    Dropout(0.2),\n    Dense(1)\n])\n\n# Compilar (sempre antes do fit)\nmodel1_cnn.compile(\n    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),\n    loss='mse'\n)\n\n# Definir EarlyStopping\nfro
m tensorflow.keras.callbacks import EarlyStopping\nearly_stop = EarlyStopping(\n    monitor='val_loss',\n    patience=10,\n    restore_best_weights=True\n)\n\n# Treinar\nhistory_cnn = model1_cnn.fit(\n    X_trai
n, y_train,\n    epochs=100,\n    batch_size=16,\n    validation_split=0.2,\n    callbacks=[early_stop],\n    verbose=1\n)\n'
```

```
# Garantir que o índice é datetime
df5_cnn.index = pd.to_datetime(df5_cnn.index)

# Selecionar subconjunto
df_previsao5 = df5_cnn.loc['2025-04-01':'2025-08-31']


# Normalizar com o mesmo scaler usado no treino
df_previsao5_norm = scaler.transform(df_previsao5.values)

# Criar sequências para este subconjunto
X_test, y_test = criar_sequencias(df_previsao5_norm, n_passo)

# Fazer previsões com o modelo já treinado
y_pred = model5_cnn.predict(X_test)

# Reverter normalização para valores reais
y_pred_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_pred), axis=1)
)[:, -1]

y_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_test.reshape(-1,1)), axis=1)
)[:, -1]

# ============================================================
# 6. Visualizar resultados
# ============================================================

# Gerar índice correspondente ao último dia da janela usada para cada previsão
index_alinhado = df_previsao5.index[n_passo - 1 : n_passo - 1 + len(y_pred_reais)]

# Garantir que os valores reais estejam alinhados com esse índice
real_values = df_previsao5['ELET_CONS_CLT'].iloc[n_passo - 1 : n_passo - 1 + len(y_pred_reais)].values

# Plot
fig = go.Figure()
fig.add_trace(go.Scatter(
    x=index_alinhado, y=real_values,
    mode='lines', name='Real', line=dict(color='blue')
))
fig.add_trace(go.Scatter(
    x=index_alinhado, y=y_pred_reais,
    mode='lines', name='Previsto (CNN)', line=dict(color='red', dash='dash')
))
fig.update_layout(
    title="Previsões no Grupo de Dados (CNN)",
    xaxis_title="Data", yaxis_title="kWh/dia/clt",
    template="plotly_white", title_x=0.5
)
fig.show()
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
```



Validar previsões

```
# Avaliar modelo
df_resultados = avaliar_gru(
    modelo=model5_cnn,
    X_train=X_train, y_train=y_train,   # podes manter os originais
    X_test=X_test, y_test=y_test,       # aqui vai o subconjunto abril-agosto
    scaler=scaler,
    df_index=df_previsao5.index,             # índice do subconjunto
    n_passo=n_passo,
    modelo_nome="CNN",
```

```
    iteracao=5
)
```

```
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
CNN (Iteração 5) - TREINO → MSE: 1.1067 | RMSE: 1.05 | MAE: 0.7428 | R²: 0.94
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
CNN (Iteração 5) - TESTE  → MSE: 0.7146 | RMSE: 0.85 | MAE: 0.6640 | R²: 0.71
```

```
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nResultados acumulados:")
print(df_resultados)
```

```
Resultados acumulados:
   Modelo  Iteração Conjunto     MSE  RMSE     MAE    R²
0     GRU         1   Treino  1.5272  1.24  0.8796  0.92
1     GRU         1    Teste  1.0344  1.02  0.7701  0.58
2     GRU         2   Treino  1.3627  1.17  0.8658  0.92
3     GRU         2    Teste  1.0475  1.02  0.8255  0.56
4     GRU         3   Treino  1.2581  1.12  0.7787  0.93
5     GRU         3    Teste  0.8839  0.94  0.7369  0.63
6     GRU         4   Treino  0.5170  0.72  0.5368  0.77
7     GRU         4    Teste  1.1309  1.06  0.8346  0.52
8     GRU         5   Treino  0.3215  0.57  0.4216  0.93
9     GRU         5    Teste  1.9958  1.41  1.1462  0.15
10    GRU         3   Treino  1.3724  1.17  0.8605  0.92
11    GRU         3    Teste  0.9088  0.95  0.7393  0.61
12    CNN         1   Treino  8.6086  2.93  1.7113  0.53
13    CNN         1    Teste  1.9981  1.41  1.1889  0.19
14    CNN         2   Treino  1.4365  1.20  0.8266  0.92
15    CNN         2    Teste  0.6896  0.83  0.6368  0.72
16    CNN         3   Treino  1.1868  1.09  0.7599  0.94
17    CNN         3    Teste  0.6066  0.78  0.5809  0.75
18    CNN         4   Treino  0.6948  0.83  0.6288  0.96
19    CNN         4    Teste  0.5022  0.71  0.5626  0.80
20    CNN         5   Treino  1.1067  1.05  0.7428  0.94
21    CNN         5    Teste  0.7146  0.85  0.6640  0.71
```

```
# Adicionar coluna de previsões (apenas nas datas alinhadas)
df_pred.loc[index_alinhado, "Previsto_CNN5"] = y_pred_reais

# Opcional: também podes guardar o erro
df_pred.loc[index_alinhado, "Erro_CNN5"] = ((df_pred["Previsto_CNN5"] - df_pred["ELET_CONS_CLT"])/df_pred["ELET_CONS_CLT"])*100
```

```
df_pred
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	Previsto_CNN1	Erro_CNN1	Previsto_CNN2	Erro_CNN2	Previsto_CNN3	Erro_CNN3	Previsto_CNN4	Erro_CNN4
2023-03-30	18.750000	24.900000	13.8	11.099999	77.516667	91.000000	63.299999	27.700001	21.755307	5679.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2023-03-31	18.333333	24.400000	14.1	10.299999	85.416667	100.000000	70.000000	30.000000	19.473651	11569.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2023-04-02	18.366667	22.100000	14.2	7.900001	74.466667	95.599998	44.400002	51.199997	22.624800	8496.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2023-04-03	19.140000	25.299999	15.0	10.299999	55.960000	75.500000	44.500000	31.000000	25.191946	9342.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2023-04-09	18.785715	26.700001	13.6	13.100000	74.428573	87.300003	59.900002	27.400002	22.498339	11715.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
2025-08-27	23.416990	31.900000	18.3	13.600000	75.882190	89.000000	51.500000	37.500000	23.980210	12914.0	...	12.768323	26.259473	10.860293	7.391934	10.964200	8.419413	11.061909	9.385609
2025-08-28	22.236250	30.200000	18.3	11.900000	80.148180	100.000000	54.700000	45.300000	22.372110	12826.0	...	12.854179	21.666723	11.236564	6.355749	11.207939	6.084811	11.770431	11.408879
2025-08-29	22.645070	34.000000	17.5	16.500000	79.591120	91.500000	54.400000	37.100000	22.824760	12985.0	...	13.347230	12.040670	12.092545	1.508461	12.030809	0.990235	11.888964	-0.200453
2025-08-30	24.656980	34.500000	19.7	14.800000	78.813560	92.000000	56.400000	35.600000	25.362570	12416.0	...	13.482678	10.654392	12.155922	-0.234499	12.203667	0.157351	11.690350	-4.055518
2025-08-31	22.518880	30.100000	18.7	11.400000	83.646050	98.500000	59.900000	38.600000	22.132800	12372.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

552 rows × 71 columns

```
Comece a programar ou gerar com a IA.
```

6ª iteração

```
df1_filtrado3.columns
```

```
Index(['T_mean', 'T_max', 'T_min', 'T_rg', 'RH_mean', 'RH_max', 'RH_min',
       'RH_rg', 'Calor_idx', 'ELET_C', 'EP.C', '50S.C', 'COZ.50S.C',
       'COZ.EP.C', 'PST.C', 'LAV.C', 'Q_OCP', 'OCP', 'OCP[%]', 'AD', 'CR',
       'CLTs', 'ELET_CONS_CLT', 'CONS_Q', 'AGUA_C', 'GAS_C', 'dia_semana',
       'trimestre', 'mes', 'ano', 'dia_ano', 'dia_mes', 'semana_ano',
       'feriado', 'AGUA_CONS_CLT', 'GAS_CONS_CLT', 'CO2e_ELET', 'CO2e_AGUA',
       'CO2e_GAS', 'Total_CO2e', 'CO2e_QOCP', 'ELET_CONS_CLT_lag1',
       'ELET_CONS_CLT_lag7', 'ELET_CONS_CLT_rolling_mean',
       'ELET_CONS_CLT_rolling_median', 'ELET_CONS_CLT_rolling_std',
       'ELET_CONS_CLT_error_median', 'ELET_CONS_CLT_pct_change'],
      dtype='object')
```

```
# 🔍 Selecionar variáveis de entrada
features = [ 'T_mean', 'T_max', 'T_min', 'RH_mean', 'RH_max', 'RH_min', 'Calor_idx',
             'Q_OCP', 'OCP', 'AD', 'CR', 'CLTs',


         'dia_semana', 'trimestre', 'mes', 'dia_ano', 'dia_mes', 'semana_ano',




            'ELET_CONS_CLT'] # Aqui, o script seleciona um conjunto de variáveis de entrada — e a última coluna escolhida é ELET_CONS_CLT, o valor que se deseja prever.
df6_cnn = df1_filtrado3[features]
```

```
Comece a programar ou gerar com a IA.
```

```
df6_cnn = df6_cnn.replace([np.inf, -np.inf], np.nan)

# Opção 1: Preencher com a média de cada coluna
#df6_cnn = df6_cnn.fillna(df6_cnn.mean())

# Opção 2: Remover linhas com NaN (se não forem muitas)
df6_cnn = df6_cnn.dropna()
```

```
df6_cnn = df6_cnn.astype('float64')
```

```
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Conv1D, Dense, Dropout, Flatten,BatchNormalization
import tensorflow as tf
import numpy as np
import pandas as pd

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
# ============================================================
# 1. Normalizar dados (X + y juntos)
# ============================================================
scaler = MinMaxScaler()
df6_normalizados = scaler.fit_transform(df6_cnn.values)

# ============================================================
# 2. Criar sequências temporais
# ============================================================
def criar_sequencias(df6_normalizados, n_passo):
    X, y = [], []
    for i in range(len(df6_normalizados) - n_passo):
        X.append(df6_normalizados[i:i+n_passo, :-1])  # features
        y.append(df6_normalizados[i+n_passo, -1])     # alvo (consumo)
    return np.array(X), np.array(y)

n_passo = 15
X_train, y_train = criar_sequencias(df6_normalizados, n_passo)

# ============================================================
# 3. Criar modelo CNN
# ============================================================
# Modelo CNN otimizado
model6_cnn = Sequential([
    Input(shape=(X_train.shape[1], X_train.shape[2])),

    Conv1D(filters=64, kernel_size=15, activation='gelu'),
    Flatten(),
    Dense(64, activation='gelu'),
    Dropout(0.1),
    Dense(1)
])

# 🔹 Compilar e treinar o modelo
model6_cnn.compile(optimizer='adam', loss='mse')
history6_cnn = model6_cnn.fit(X_train, y_train, epochs=200, batch_size=64)

# ============================================================
# 4. Fazer previsões
# ============================================================
previsoes6_cnn = model6_cnn.predict(X_train)

# ============================================================
# 5. Reverter normalização
# ============================================================
previsoes6_cnn_reais = scaler.inverse_transform(
    np.concatenate((X_train[:, -1, :], previsoes6_cnn), axis=1)
)[:, -1]

# Criar DataFrame com previsões e valores reais
previsoes6_cnn_df = pd.DataFrame({
    'Previsto': previsoes6_cnn_reais,
    'Real': df6_cnn['ELET_CONS_CLT'].iloc[n_passo:].values
}, index=df6_cnn.index[n_passo:])

# ============================================================
# 6. Visualizar previsões
# ============================================================
fig = go.Figure()

# Real
fig.add_trace(go.Scatter(
    x=df6_cnn.index,
    y=df6_cnn['ELET_CONS_CLT'],
    mode='lines',
    name='Real',
    line=dict(color='blue')
))

# Previsto
fig.add_trace(go.Scatter(
    x=previsoes6_cnn_df.index,
    y=previsoes6_cnn_df['Previsto'],
    mode='lines',
    name='Previsto (CNN)',
    line=dict(color='red', dash='dash')
))

# Layout
fig.update_layout(
    title='Comparação: Consumo Real vs. Previsto (CNN)',
    xaxis_title='Data',
    yaxis_title='kWh/dia/clt',
    legend=dict(x=0, y=1),
    font=dict(size=14),
    title_x=0.5,
    width=1000,
    height=500
)

fig.show()
```

Mostrar saída oculta

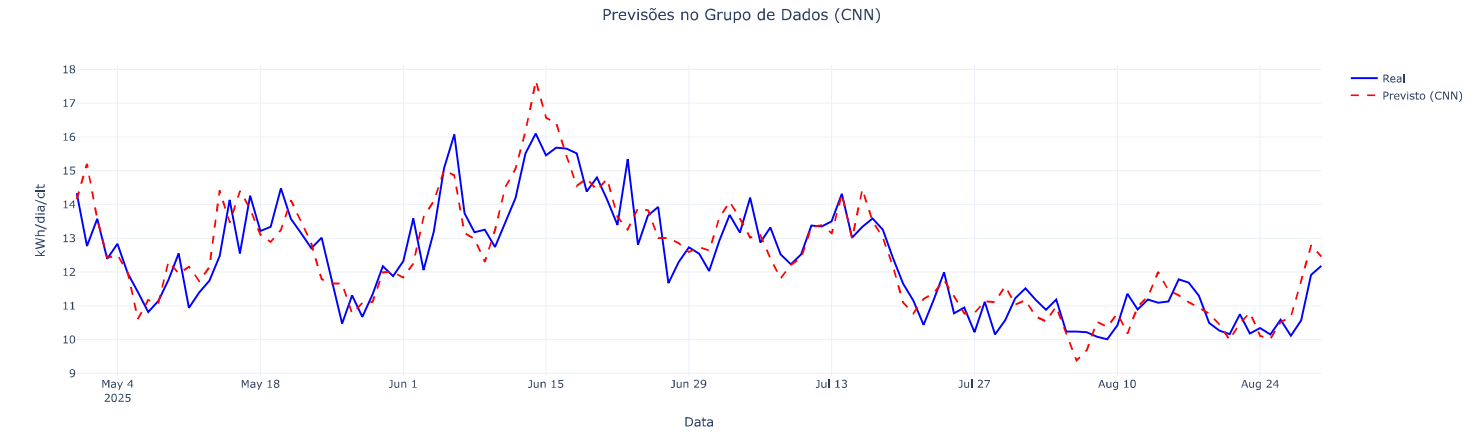
```
"""
# Criar modelo
model1_cnn = Sequential([
    Input(shape=(X_train.shape[1], X_train.shape[2])),
    Conv1D(filters=128, kernel_size=5, activation='relu'),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(1)
])

# Compilar (sempre antes do fit)
model1_cnn.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),
    loss='mse'
)

# Definir EarlyStopping
from tensorflow.keras.callbacks import EarlyStopping
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)

# Treinar
history_cnn = model1_cnn.fit(
```

```
    X_train, y_train,
    epochs=100,
    batch_size=16,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)
"""
```

```
'\n# Criar modelo\nmodel1_cnn = Sequential([\n    Input(shape=(X_train.shape[1], X_train.shape[2])),\n    Conv1D(filters=128, kernel_size=5, activation='relu'),\n    Flatten(),\n    Dense(256, activation='rel
u'),\n    Dropout(0.2),\n    Dense(1)\n])\n\n# Compilar (sempre antes do fit)\nmodel1_cnn.compile(\n    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),\n    loss='mse'\n)\n\n# Definir EarlyStopping\nfro
m tensorflow.keras.callbacks import EarlyStopping\nearly_stop = EarlyStopping(\n    monitor='val_loss',\n    patience=10,\n    restore_best_weights=True\n)\n\n# Treinar\nhistory_cnn = model1_cnn.fit(\n    X_trai
n, y_train,\n    epochs=100,\n    batch_size=16,\n    validation_split=0.2,\n    callbacks=[early_stop],\n    verbose=1\n)\n'
```

```
# Garantir que o índice é datetime
df6_cnn.index = pd.to_datetime(df6_cnn.index)

# Selecionar subconjunto
df_previsao6 = df6_cnn.loc['2025-04-01':'2025-08-31']


# Normalizar com o mesmo scaler usado no treino
df_previsao6_norm = scaler.transform(df_previsao6.values)

# Criar sequências para este subconjunto
X_test, y_test = criar_sequencias(df_previsao6_norm, n_passo)

# Fazer previsões com o modelo já treinado
y_pred = model6_cnn.predict(X_test)

# Reverter normalização para valores reais
y_pred_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_pred), axis=1)
)[:, -1]

y_reais = scaler.inverse_transform(
    np.concatenate((X_test[:, -1, :], y_test.reshape(-1,1)), axis=1)
)[:, -1]

# ============================================================
# 6. Visualizar resultados
# ============================================================

# Gerar índice correspondente ao último dia da janela usada para cada previsão
index_alinhado = df_previsao6.index[n_passo - 1 : n_passo - 1 + len(y_pred_reais)]

# Garantir que os valores reais estejam alinhados com esse índice
real_values = df_previsao6['ELET_CONS_CLT'].iloc[n_passo - 1 : n_passo - 1 + len(y_pred_reais)].values

# Plot
fig = go.Figure()
fig.add_trace(go.Scatter(
    x=index_alinhado, y=real_values,
    mode='lines', name='Real', line=dict(color='blue')
))
fig.add_trace(go.Scatter(
    x=index_alinhado, y=y_pred_reais,
    mode='lines', name='Previsto (CNN)', line=dict(color='red', dash='dash')
))
fig.update_layout(
    title="Previsões no Grupo de Dados (CNN)",
    xaxis_title="Data", yaxis_title="kWh/dia/clt",
    template="plotly_white", title_x=0.5
)
fig.show()
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
```



Validar previsões

```
# Avaliar modelo
df_resultados = avaliar_gru(
    modelo=model6_cnn,
    X_train=X_train, y_train=y_train,   # podes manter os originais
    X_test=X_test, y_test=y_test,       # aqui vai o subconjunto abril–agosto
    scaler=scaler,
    df_index=df_previsao5.index,            # índice do subconjunto
    n_passo=n_passo,
    modelo_nome="CNN",
    iteracao=6
)
```

```
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
CNN (Iteração 6) - TREINO → MSE: 0.8958 | RMSE: 0.95 | MAE: 0.6862 | R²: 0.95
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
CNN (Iteração 6) - TESTE  → MSE: 0.4596 | RMSE: 0.68 | MAE: 0.5199 | R²: 0.81
```

```
df_resultados = pd.DataFrame(resultados_acumulados)

print("\nResultados acumulados:")
print(df_resultados)
```

```

Resultados acumulados:
   Modelo  Iteração Conjunto     MSE  RMSE     MAE    R²
0     GRU         1   Treino  1.5272  1.24  0.8796  0.92
1     GRU         1    Teste  1.0344  1.02  0.7701  0.58
2     GRU         2   Treino  1.3627  1.17  0.8658  0.92
3     GRU         2    Teste  1.0475  1.02  0.8255  0.56
4     GRU         3   Treino  1.2581  1.12  0.7787  0.93
5     GRU         3    Teste  0.8839  0.94  0.7369  0.63
```

```
6     GRU          4   Treino  0.5170  0.72  0.5368  0.77
7     GRU          4    Teste  1.1309  1.06  0.8346  0.52
8     GRU          5   Treino  0.3215  0.57  0.4216  0.93
9     GRU          5    Teste  1.9958  1.41  1.1462  0.15
10    GRU          3   Treino  1.3724  1.17  0.8605  0.92
11    GRU          3    Teste  0.9088  0.95  0.7393  0.61
12    CNN          1   Treino  8.6086  2.93  1.7113  0.53
13    CNN          1    Teste  1.9981  1.41  1.1889  0.19
14    CNN          2   Treino  1.4365  1.20  0.8266  0.92
15    CNN          2    Teste  0.6896  0.83  0.6368  0.72
16    CNN          3   Treino  1.1868  1.09  0.7599  0.94
17    CNN          3    Teste  0.6066  0.78  0.5809  0.75
18    CNN          4   Treino  0.6948  0.83  0.6288  0.96
19    CNN          4    Teste  0.5022  0.71  0.5626  0.80
20    CNN          5   Treino  1.1067  1.05  0.7428  0.94
21    CNN          5    Teste  0.7146  0.85  0.6640  0.71
22    CNN          6   Treino  0.8958  0.95  0.6862  0.95
23    CNN          6    Teste  0.4596  0.68  0.5199  0.81
```

```
# Adicionar coluna de previsões (apenas nas datas alinhadas)
df_pred.loc[index_alinhado, "Previsto_CNN6"] = y_pred_reais

# Opcional: também podes guardar o erro
df_pred.loc[index_alinhado, "Erro_CNN6"] = ((df_pred["Previsto_CNN6"] - df_pred["ELET_CONS_CLT"])/df_pred["ELET_CONS_CLT"])*100
```

```
df_pred
```

DATA	T_mean	T_max	T_min	T_rg	RH_mean	RH_max	RH_min	RH_rg	Calor_idx	ELET_C	...	Previsto_CNN2	Erro_CNN2	Previsto_CNN3	Erro_CNN3	Previsto_CNN4	Erro_CNN4	Previsto_CNN5	Erro_CNN5
2023-03-30	18.750000	24.900000	13.8	11.099999	77.516667	91.000000	63.299999	27.700001	21.755307	5679.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2023-03-31	18.333333	24.400000	14.1	10.299999	85.416667	100.000000	70.000000	30.000000	19.473651	11569.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2023-04-02	18.366667	22.100000	14.2	7.900001	74.466667	95.599998	44.400002	51.199997	22.624800	8496.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2023-04-03	19.140000	25.299999	15.0	10.299999	55.960000	75.500000	44.500000	31.000000	25.191946	9342.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2023-04-09	18.785715	26.700001	13.6	13.100000	74.428573	87.300003	59.900002	27.400002	22.498339	11715.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
2025-08-27	23.416990	31.900000	18.3	13.600000	75.882190	89.000000	51.500000	37.500000	23.980210	12914.0	...	10.860293	7.391934	10.964200	8.419413	11.061909	9.385609	10.505621	3.884757
2025-08-28	22.236250	30.200000	18.3	11.900000	80.148180	100.000000	54.700000	45.300000	22.372110	12826.0	...	11.236564	6.355749	11.207939	6.084811	11.770431	11.408879	11.009132	4.203077
2025-08-29	22.645070	34.000000	17.5	16.500000	79.591120	91.500000	54.400000	37.100000	22.824760	12985.0	...	12.092545	1.508461	12.030809	0.990235	11.888964	-0.200453	12.029396	0.978370
2025-08-30	24.656980	34.500000	19.7	14.800000	78.813560	92.000000	56.400000	35.600000	25.362570	12416.0	...	12.155922	-0.234499	12.203667	0.157351	11.690350	-4.055518	12.404818	1.808228
2025-08-31	22.518880	30.100000	18.7	11.400000	83.646050	98.500000	59.900000	38.600000	22.132800	12372.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

552 rows × 73 columns

```
df_resultados
```

	Modelo	Iteração	Conjunto	MSE	RMSE	MAE	R²
0	GRU	1	Treino	1.5272	1.24	0.8796	0.92
1	GRU	1	Teste	1.0344	1.02	0.7701	0.58
2	GRU	2	Treino	1.3627	1.17	0.8658	0.92
3	GRU	2	Teste	1.0475	1.02	0.8255	0.56
4	GRU	3	Treino	1.2581	1.12	0.7787	0.93
5	GRU	3	Teste	0.8839	0.94	0.7369	0.63
6	GRU	4	Treino	0.5170	0.72	0.5368	0.77
7	GRU	4	Teste	1.1309	1.06	0.8346	0.52
8	GRU	5	Treino	0.3215	0.57	0.4216	0.93
9	GRU	5	Teste	1.9958	1.41	1.1462	0.15
10	GRU	3	Treino	1.3724	1.17	0.8605	0.92
11	GRU	3	Teste	0.9088	0.95	0.7393	0.61
12	CNN	1	Treino	8.6086	2.93	1.7113	0.53
13	CNN	1	Teste	1.9981	1.41	1.1889	0.19
14	CNN	2	Treino	1.4365	1.20	0.8266	0.92
15	CNN	2	Teste	0.6896	0.83	0.6368	0.72
16	CNN	3	Treino	1.1868	1.09	0.7599	0.94
17	CNN	3	Teste	0.6066	0.78	0.5809	0.75
18	CNN	4	Treino	0.6948	0.83	0.6288	0.96
19	CNN	4	Teste	0.5022	0.71	0.5626	0.80
20	CNN	5	Treino	1.1067	1.05	0.7428	0.94
21	CNN	5	Teste	0.7146	0.85	0.6640	0.71
22	CNN	6	Treino	0.8958	0.95	0.6862	0.95
23	CNN	6	Teste	0.4596	0.68	0.5199	0.81

```
df_resultados_cnn = df_resultados[df_resultados['Modelo'] == 'CNN'].copy()

df_resultados_cnn
```

```
    Modelo  Iteração  Conjunto     MSE  RMSE     MAE    R²
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# DataFrame com os resultados acumulados (já tens)
# df_resultados = pd.DataFrame(resultados_acumulados)
```