Avaliação da Regressão Harmônica Dinâmica com Boosting para Previsões de Temperatura em 365 Dias: Um Estudo de Caso em Chicago

Pedro Gabriel Moura
Departamento de Estatítica - Universidade de Brasília (UnB)
pedropgmoura@gmail.com

Abstract

This study evaluates a hybrid forecasting approach that integrates Dynamic Harmonic Regression (DHR) with a LightGBM model trained on the DHR residuals to predict Chicago's daily average temperature up to 365 days ahead. The dataset exhibits strong annual seasonality and seasonal heteroscedasticity, making long-term forecasting particularly challenging. After discarding Box–Cox transformations and identifying a single dominant seasonal cycle through wavelet analysis, the DHR was optimized and selected as the baseline model. A comprehensive feature-engineering pipeline generated lagged and differenced predictors for the machine-learning component, followed by multi-stage variable selection and hyperparameter optimization using Optuna and grid search.

Results show that the hybrid model achieves slightly better performance than DHR alone, particularly at longer forecast horizons, despite the unfavorable conditions for machine learning. These findings suggest that hybrid approaches may offer substantial benefits in richer data environments. Future work may explore larger datasets, contemporary exogenous variables, neural architectures, and probabilistic techniques to further improve long-range temperature forecasting.

Palavras-chave: Regressão Harmônica Dinâmica; Previsão de Séries Temporais; LightGBM; Modelo Híbrido; Temperatura.

1 Introdução

A previsão de temperatura é um tema de crucial importância em várias áreas, incluindo planejamento energético, agricultura, saúde pública e modelagem climática. Embora métodos estatísticos tradicionais, como a Regressão Harmônica Dinâmica (RHD) [Hyndman e Athanasopoulos 2021], continuem relevantes, especialmente em previsões de longo prazo que exigem modelos parcimoniosos, interpretáveis e capazes de capturar padrões sazonais bem definidos, a busca por maior precisão tem impulsionado o desenvolvimento de abordagens híbridas.

Paralelamente, o interesse por modelos híbridos, que combinam a solidez dos métodos estatísticos com a adaptabilidade dos algoritmos de aprendizado de máquina, tem crescido. A literatura [Camelo et al. 2017] aponta que a combinação de modelos estatísticos, responsáveis por capturar tendências, ciclos e sazonalidades, com modelos de aprendizado de máquina aplicados aos resíduos pode potencialmente melhorar o desempenho preditivo ao explorar padrões remanescentes não explicados pela componente estrutural. A expectativa é que essa integração resulte em previsões mais robustas e acuradas.

Este estudo avalia a eficácia de um modelo híbrido que combina modelos estatísticos de séries temporais com modelos de aprendizagem de máquina aplicados aos resíduos do modelo temporal. O objetivo é determinar se essa abordagem pode proporcionar ganhos significativos de acurácia em um cenário particularmente desafiador: a previsão da temperatura média diária com um horizonte de 365 passos à frente. Trata-se de um contexto marcado por forte sazonalidade anual e por covariáveis disponíveis apenas com longas defasagens, características que tendem a limitar o poder explicativo das variáveis independentes e, consequentemente, restringir o potencial preditivo de modelos de aprendizado de máquina.

Os resultados obtidos indicam que o modelo RHD tradicional apresenta desempenho superior ou, no mínimo, equivalente ao do modelo híbrido nos horizontes analisados. Essa evidência sugere que, em cenários onde a estrutura sazonal explica a maior parte da variabilidade e onde as covariáveis possuem baixo poder preditivo em defasagens longas, o ganho esperado com o método não se concretiza.

O presente artigo está organizado da seguinte maneira: a Seção 2 detalha os dados utilizados; a Seção 3 apresenta a metodologia e descreve os modelos empregados; a Seção 4 expõe os resultados empíricos; a Seção 5 oferece uma interpretação crítica desses achados; e a Seção 6 sintetiza as conclusões, discutindo limitações e propondo direções para futuras investigações.

2 Conjunto de Dados

Este estudo utiliza dados meteorológicos diários obtidos por meio da API do pacote Meteostat em Python, que disponibiliza uma ampla variedade de informações históricas e atuais provenientes de estações meteorológicas distribuídas globalmente. Especificamente, foram coletados dados referentes à cidade de Chicago, Illinois, cobrindo o período de 21 de novembro de 2014 a 09 de novembro de 2025.

O conjunto de dados é composto por diversas variáveis meteorológicas relevantes para a modelagem e previsão de temperatura, conforme detalhado na Tabela 1.

Table 1: Informação dos dados

Variável	**Descrição**	**Unidade**	**Tipo**
time	Data da medição	Dia	Datetime64
tavg	Temperatura média diária	Kelvin	Float64
tmin	Temperatura mínima registrada no dia	Kelvin	Float64
tmax	Temperatura máxima registrada no dia	Kelvin	Float64
prcp	Quantidade total de precipitação diária	mm	Float64
snow	Profundidade acumulada de neve	mm	Float64
wspd	Velocidade média do vento	km/h	Float64
pres	Pressão atmosférica ao nível do mar	hPa	Float64

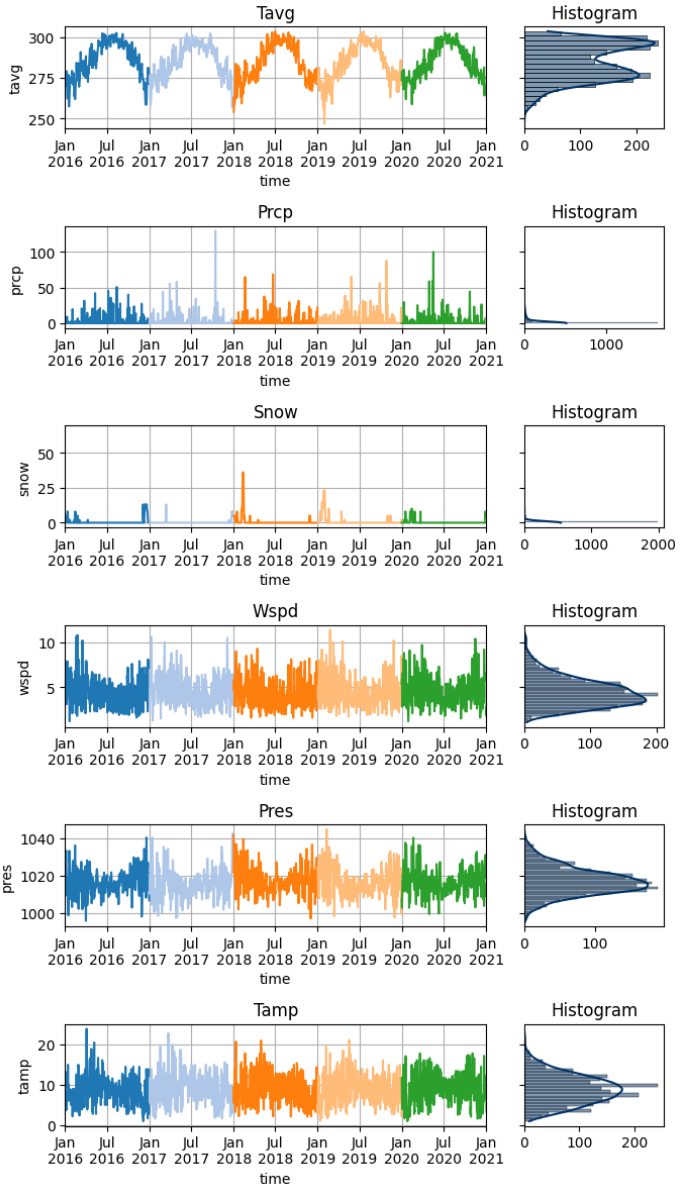
As variáveis *tsun*, *wpgt* e *wdir* foram excluídas da análise devido à insuficiência de dados, apresentando menos de 1% de preenchimento, além disso, as variáveis de temperatura foram originalmente fornecidas em graus Celsius, porém foram convertidas para Kelvin para padronização e consistência na análise. As demais variáveis foram mantidas por sua relevância potencial na previsão de temperatura e pela regularidade de sua disponibilidade ao longo do período estudado.

Antes da utilização nos modelos, foi conduzido um processo rigoroso de limpeza e preparação dos dados, assegurando sua qualidade e integridade para as etapas subsequentes. Os procedimentos específicos adotados nessa etapa são detalhados na seção de Metodologia.

A escolha de Chicago como local de estudo foi realizada de maneira arbitrária, sem um critério específico. Ainda assim, a cidade se destaca por sua ampla variabilidade climática, o que fornece um ambiente desafiador e apropriado para a investigação de métodos de previsão de longo prazo.

A Figura 1 apresenta a distribuição das variáveis incluídas na análise, ilustrando a heterogeneidade das características meteorológicas consideradas. As unidades de medida foram padronizadas de acordo com práticas internacionalmente adotadas, facilitando futuras comparações e possíveis integrações com outros estudos ou bases de dados.

Figure 1: Distribuição dos dados, Fonte: elaboração própria.



3 Metodologia

O presente estudo foi organizado em quatro fases principais. Na primeira fase, denominada Sanitização, realizou-se a compreensão inicial e a validação do conjunto de dados, assegurando a qualidade necessária para as análises subsequentes. A segunda fase, Análise Exploratória, concentrou-se na investigação de padrões e características da série temporal de interesse, *tavg*, bem como na definição das estratégias de modelagem. A terceira fase, Modelagem, envolveu o treinamento, a avaliação e o refinamento dos modelos propostos, com o objetivo de otimizar o desempenho preditivo. Por fim, na seção seguinte, Resultados, são apresentadas as descobertas obtidas ao longo do processo e demais informações relevantes.

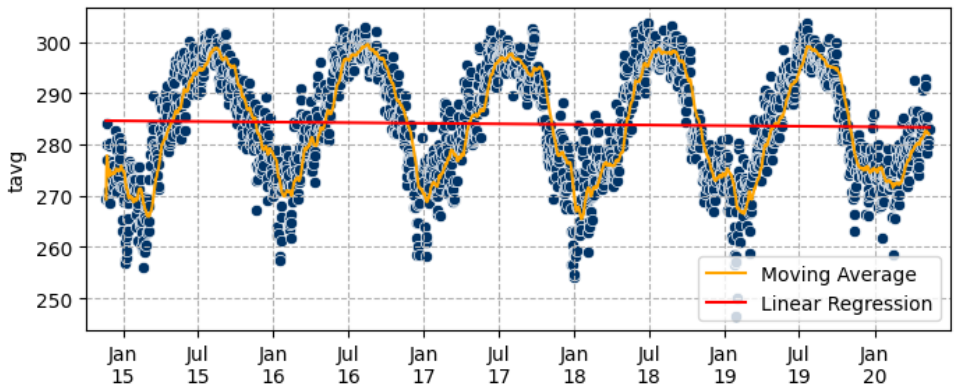
3.1 Sanitização

A etapa de sanitização envolveu um conjunto de verificações destinadas a assegurar a consistência e a validade do conjunto de dados. Inicialmente, procedeu-se à análise de integridade, confirmando-se a ausência de registros duplicados e a continuidade da frequência diária das observações. No entanto, verificou-se que as variáveis *wdir* e *tsun* estavam completamente vazias ao longo de todo o período analisado, enquanto a variável *wpgt* apresentava apenas 0,02% de preenchimento. Dada a extrema insuficiência de informações, essas três variáveis foram excluídas da análise.

A variável *snow* apresentou oito valores ausentes, concentrados nos últimos 11 dias da amostra, entre 10 e 20 de novembro de 2025. Considerando a localização temporal dessas ausências e visando manter a continuidade e a estabilidade da série, optou-se pela remoção desses dias do conjunto de dados.

A série temporal de temperatura média diária apresentou valores entre 246,6 e 306 Kelvin, compatíveis com o regime climático da cidade de Chicago. Contudo, uma inspeção mais detalhada evidenciou padrões importantes para a etapa de modelagem. Para facilitar a visualização, as figuras a seguir foram construídas utilizando apenas os 2.000 primeiros dias da série, evitando a sobrecarga gráfica que prejudicaria a interpretação.

Figure 2: Visualização de padrões temporais — *tavg*. Fonte: elaboração própria.



A Figura 2 sugere a presença de uma sazonalidade anual pronunciada, sem indicação clara de tendência. Já a Figura 3, que apresenta a sobreposição das séries anuais, evidencia maior variabilidade das temperaturas no início e no final do ano, em contraste com o período intermediário. Esse comportamento sugere a presença de heteroscedasticidade, posteriormente confirmada pelo gráfico da variância móvel com janela de 180 dias (Figura 4), o qual revela flutuações sazonais na variância da série temporal.

Figure 3: Sobreposição anual — *tavg*. Fonte: elaboração própria.

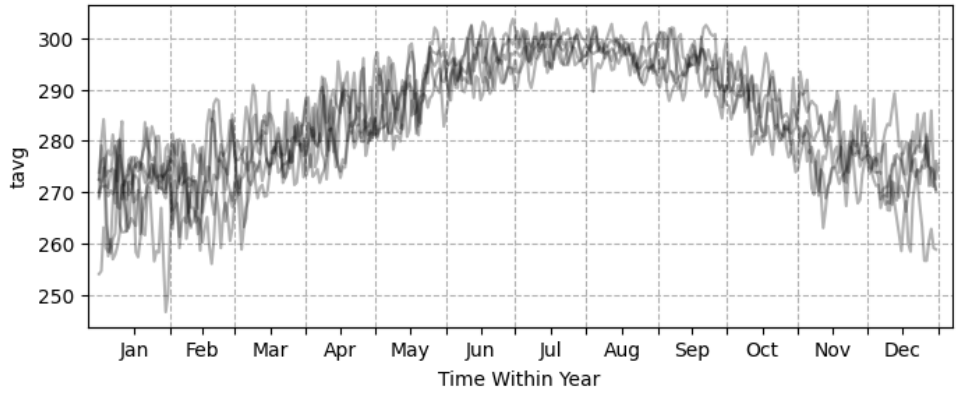
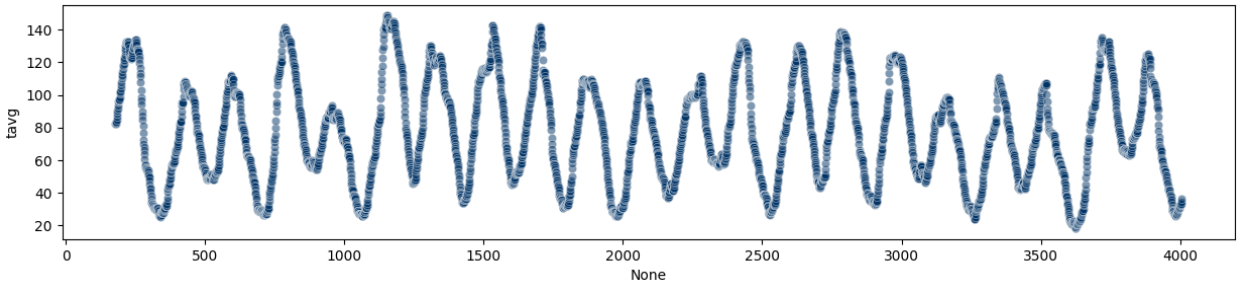


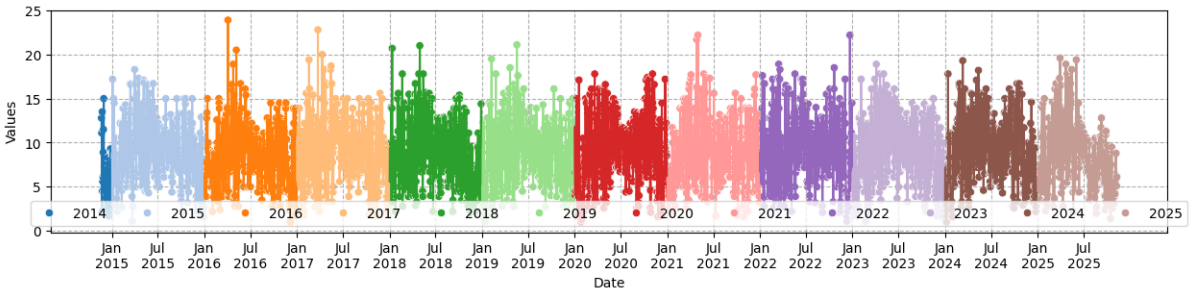
Figure 4: Heteroscedasticidade — *tavg*. Fonte: elaboração própria.



As variáveis *tmin* e *tmax* apresentaram forte associação com a variável alvo, *tavg*, o que é esperado dada a relação estrutural entre temperatura média, mínima e máxima. Entretanto, quando utilizadas em defasagem, condição necessária para garantir sua disponibilidade no momento da previsão, espera-se que essas variáveis não fornecem informação adicional que já não esteja contida na própria *tavg* defasada. Ainda assim, a interação entre *tmin* e *tmax* carrega uma estrutura informacional que não está presente na série temporal de *tavg* de forma isolada.

Com o objetivo de capturar essa informação complementar, foi criada a variável *tamp*, definida como a amplitude térmica diária, calculada pela diferença entre *tmax* e *tmin*. A distribuição dessa nova variável é apresentada na Figura 5, evidenciando como a amplitude de temperatura se comporta ao longo do período analisado.

Figure 5: Distribuição da amplitude térmica — *tamp*. Fonte: elaboração própria.



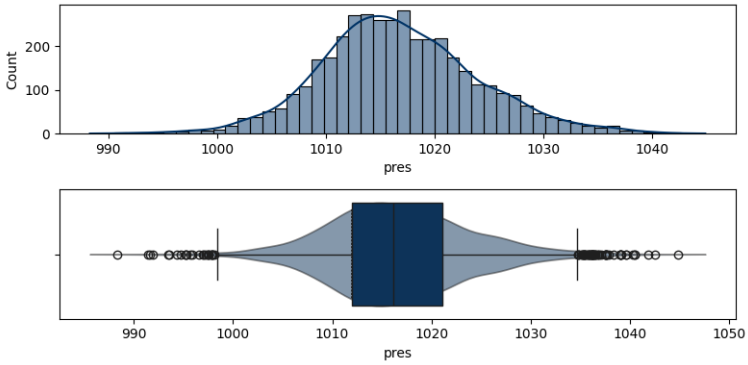
Na variável *prcp* (precipitação), foram observados valores superiores ao esperado, variando entre 0 e 129 mm por dia. Conforme discutido no estudo referenciado [Liu et al. 2013], estabelecer um limiar para classificar eventos de chuva como extremos é um desafio metodológico.

Ainda assim, para fins de identificação de potenciais erros de medição, adotou-se o limite de 50 mm/dia, valor citado pelo artigo tanto no método não paramétrico quanto no não paramétrico de valor crítico absoluto. Com base nesse critério, identificaram-se 17 dias de chuvas extremas. Após verificação manual em relatórios de eventos severos disponibilizados pelo National Weather Service, esses registros foram confirmados como plausíveis para a região de Chicago no período analisado.

A variável *snow* (profundidade de neve) mostrou-se altamente esparsa, com mais de 90% das observações assumindo valor zero. Os picos de atividade coincidem com períodos de maior variabilidade térmica, predominantemente no início e no final do ano, refletindo a forte queda de temperatura típica dos meses de inverno na cidade.

Para a variável *wspd* (velocidade média do vento), observou-se uma faixa de valores entre 1,1 km/h e 11,9 km/h, com média de 4,446 km/h, indicando variação moderada no regime de ventos durante o período estudado. Já a variável *pres* (pressão atmosférica) apresentou distribuição aproximadamente simétrica em formato de sino, conforme ilustrado na Figura 6, sugerindo visualmente uma possível aderência à distribuição normal. No entanto, os testes estatísticos de normalidade de Shapiro–Wilk [Shapiro e Wilk 1965] e Jarque–Bera [Jarque e Bera 1987] afastaram essa hipótese, com p-valores de 2,72e-12 e 2,32e-16, respectivamente, indicando rejeição clara da normalidade.

Figure 6: Distribuição da amplitude atmosférica — *pres*. Fonte: elaboração própria.



3.2 Análise exploratória

Nesta seção, investigou-se as estruturas temporais da variável *tavg* como etapa preparatória para a construção do modelo híbrido de previsão. A análise de tendência, padrões sazonais e autocorrelação tem como objetivo avaliar em que medida a variabilidade da temperatura pode ser capturada por métodos estatísticos clássicos de modelagem de séries temporais.

Para evitar vazamento de informação futura no processo de treinamento, toda a análise exploratória foi realizada exclusivamente sobre o conjunto de dados de treino, compreendendo o período de 21 de novembro de 2014 a 18 de novembro de 2021. Antes de examinar as estruturas temporais propriamente ditas, avaliou-se preliminarmente a possibilidade de reduzir a heteroscedasticidade da série, de modo a atender mais adequadamente aos pressupostos de modelos tradicionais como os da família ARIMA [Box e Jenkins 1970].

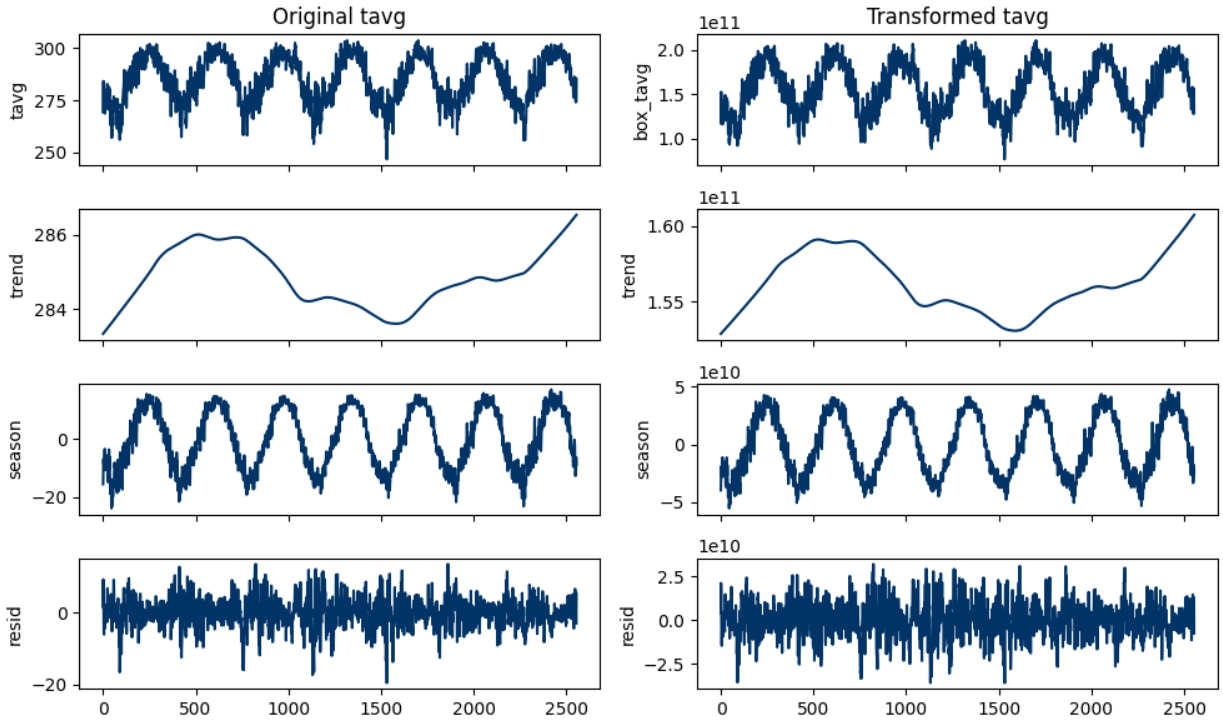
Com esse propósito, aplicou-se a transformação Box–Cox [Box e Cox 1964]. Em seguida, os mesmos procedimentos utilizados para a série original foram reproduzidos na série transformada, possibilitando uma comparação direta entre ambas. A adoção da série transformada seria justificada caso apresentasse uma redução significativa na métrica de avaliação, o erro quadrático médio (RMSE), sem comprometer a interpretabilidade dos componentes temporais. A escolha do RMSE fundamenta-se em sua maior sensibilidade a valores extremos quando comparado ao erro absoluto médio (MAE), sendo ambas métricas amplamente empregadas na avaliação de modelos

preditivos [Hyndman e Athanasopoulos 2018]. Essa característica é particularmente útil para investigar se a transformação contribui para aproximar a série da homoscedasticidade e, simultaneamente, melhorar o desempenho preditivo do modelo.

3.2.1 Decomposição STL

A decomposição clássica via STL [Cleveland et al. 1990] não apresentou diferenças relevantes entre as versões com e sem transformação Box–Cox. Na realidade, o modelo ajustado à série original apresentou um desempenho ligeiramente superior (RMSE = 15,92) em comparação ao modelo ajustado à série transformada (RMSE = 15,93). Além disso, os componentes resultantes da decomposição mostraram pouca divergência visual entre si, conforme ilustrado na Figura 7.

Figure 7: Decomposição via STL. Fonte: elaboração própria.



Para confirmar essa conclusão e garantir que a transformação não introduza melhorias substanciais, procedemos a uma decomposição manual das séries, examinando separadamente tendência, sazonalidade e resíduos.

3.2.2 Tendência

Como a série apresenta ciclos sazonais bem definidos, evitou-se o uso de métodos que pudessem distorcer essa estrutura ao remover a tendência. Assim, optou-se por empregar uma regressão linear [Montgomery, Peck e Vining 2012], estimada por mínimos quadrados ordinários, com o objetivo exclusivo de avaliar a magnitude da tendência sem interferir no formato da série. Embora esse método não seja o mais adequado para modelagem de tendência em séries temporais, sua utilização aqui é apenas diagnóstica, e seu impacto é estatisticamente irrelevante para o objetivo desta análise.

Os resultados indicam que ambas as regressões apresentaram coeficientes angulares muito pequenos em relação aos interceptos (Tabela 2), corroborando a hipótese de que a tendência é praticamente desprezível, consistente com o que já havia sido observado na etapa de sanitização.

Table 2: Regressão Linear para identificar Tendência

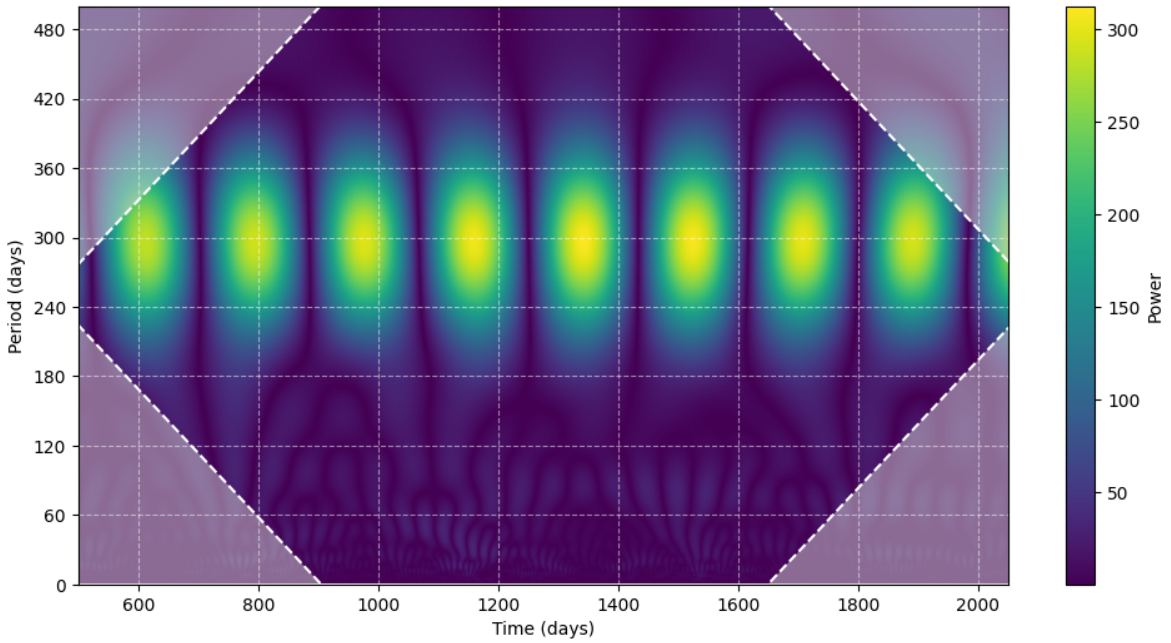
Dados	**Intercepto (α)**	**Coeficiente Angular (β)**
Box-Cox *tavg*	1,521e+11	3,138e+06
tavg Sem transformação	283,2454	0,0012

Com a tendência estimada, avançou-se para a identificação e posterior remoção da sazonalidade presente nas séries de resíduos das regressões aplicadas nesta seção.

3.2.3 Sazonalidade

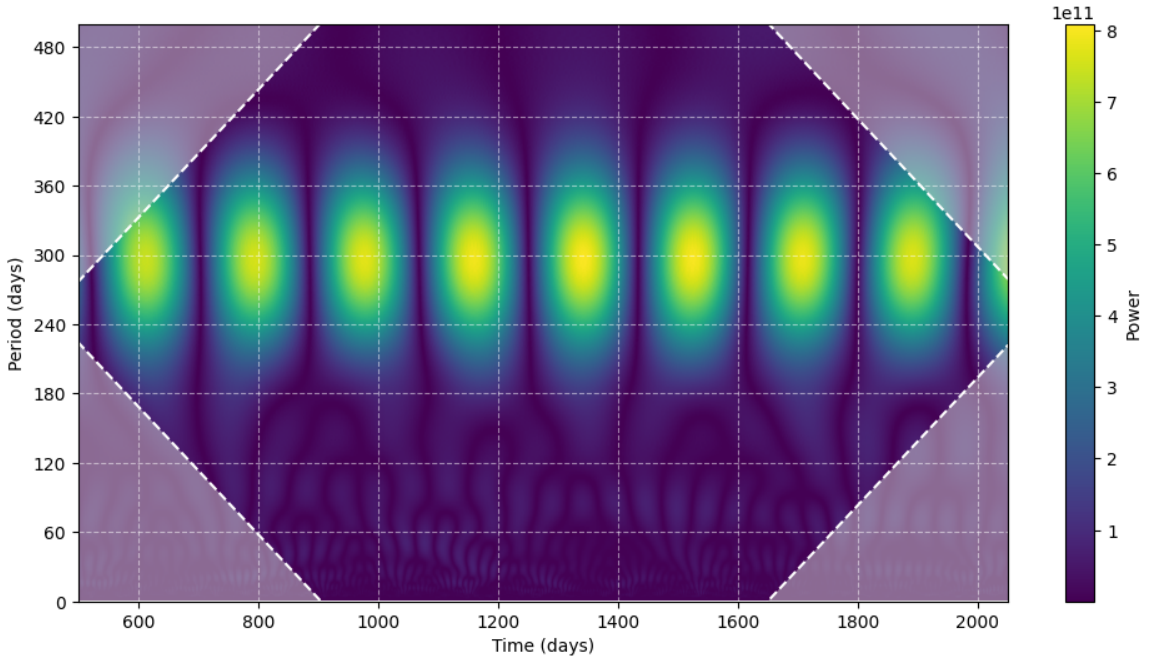
As séries, tanto a transformada quanto a original, apresentam uma sazonalidade anual claramente definida. No entanto, com as informações disponíveis até este ponto da análise, não é possível determinar quantos padrões sazonais distintos estão presentes. Para identificar adequadamente o número de ciclos sazonais, empregou-se o método do *wavelet plot* [Daubechies 1992], que consiste em uma representação visual dos resultados da Transformada de *Wavelets* [Torrence e Compo 1998]. Essa técnica decompõe o sinal em diferentes escalas de tempo e frequência, permitindo a visualização simultânea de padrões locais e globais. Esse método possibilitou identificar com precisão a quantidade de padrões sazonais existentes em cada uma das séries analisadas.

Figure 8: Espectro wavelet com Cone de Influência - *tavg*. Fonte: elaboração própria.



Tanto a Figura 8 quanto a Figura 9 revelam apenas um ciclo sazonal dominante que se repete regularmente ao longo de toda a série temporal. Esse ciclo corresponde à sazonalidade anual, mas sua energia (*power*) se concentra entre 280 e 320 dias, em vez dos 365 dias exatos. Esse deslocamento é característico da transformada *wavelet*, cujas estimativas de período são aproximadas. Fatores como a janela temporal finita, a suavização inerente ao método e a variação temporal da amplitude do sinal podem deslocar o pico de energia para valores ligeiramente inferiores ao período real. Ainda assim, o padrão recorrente confirma de forma inequívoca a existência de uma única sazonalidade.

Figure 9: Espectro wavelet com Cone de Influência - Box-Cox *tavg*. Fonte: elaboração própria.



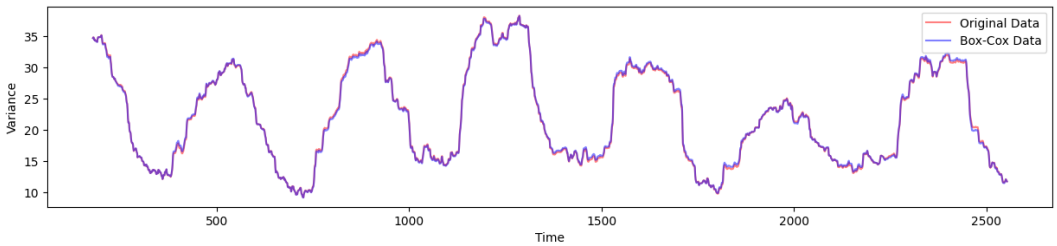
Ao longo do tempo, observa-se que a energia associada à banda sazonal apresenta variações expressivas, reforçando a presença de variância sazonal já mencionada anteriormente. É importante destacar que grande parte dessa banda encontra-se dentro do cone da influência, o que garante que a detecção do ciclo anual é estatisticamente confiável e não decorre de artefatos de borda.

Para remover a sazonalidade das séries, utilizou-se o mesmo método empregado para a extração da tendência: uma regressão estimada por mínimos quadrados ordinários, agora acrescida de dois pares harmônicos capazes de capturar o ciclo sazonal anual. A partir desse modelo, foram obtidos os resíduos que serão utilizados nas etapas subsequentes de diagnóstico.

3.2.4 Diagnóstico

Na Figura 10, observa-se que as séries não apresentam diferenças visíveis na variância móvel com janela de 180 dias, indicando que a transformação Box–Cox provavelmente não foi eficaz em aproximar os dados da homoscedasticidade. Além disso, o RMSE da série sem transformação (23,838184) foi menor do que o RMSE obtido após a aplicação da transformação Box–Cox (23,876681), reforçando que sua utilização não se justifica neste caso.

Figure 10: Variância móvel dos resíduos. Fonte: elaboração própria.



Com a transformação descartada, conclui-se também que os dados de treino não apresentam tendência significativa e possui apenas um ciclo sazonal. Uma estratégia natural de modelagem, considerando essa característica, seria a utilização de um modelo SARIMA, pertencente à família

ARIMA anteriormente mencionada. Contudo, dado que a sazonalidade é anual e os dados possuem frequência diária, o que exigiria um período sazonal de aproximadamente 365, o modelo mais adequado é a Regressão Harmônica Dinâmica, capaz de capturar sazonalidades longas de maneira mais flexível e estatisticamente eficiente.

3.3 Modelagem

Por utilizar um modelo hibrido, a etapa de modelagem foi dividida em duas grandes etapas: Modelagem temporal e modelagem dos resíduos. Cada etapa possui uma estratégia e um procedimento especifico adaptado ao seu objetivo.

3.3.1 Modelagem Temporal

Esta subseção tem como objetivo remover todas as dependências temporais da série, tradicionalmente compostas por tendência, sazonalidade e autocorrelação. No entanto, conforme evidenciado na seção anterior, a série não apresenta tendência significativa. Assim, o foco da modelagem recai exclusivamente sobre a sazonalidade e a autocorrelação, tornando a Regressão Harmônica Dinâmica (RHD) um excelente candidato para essa finalidade.

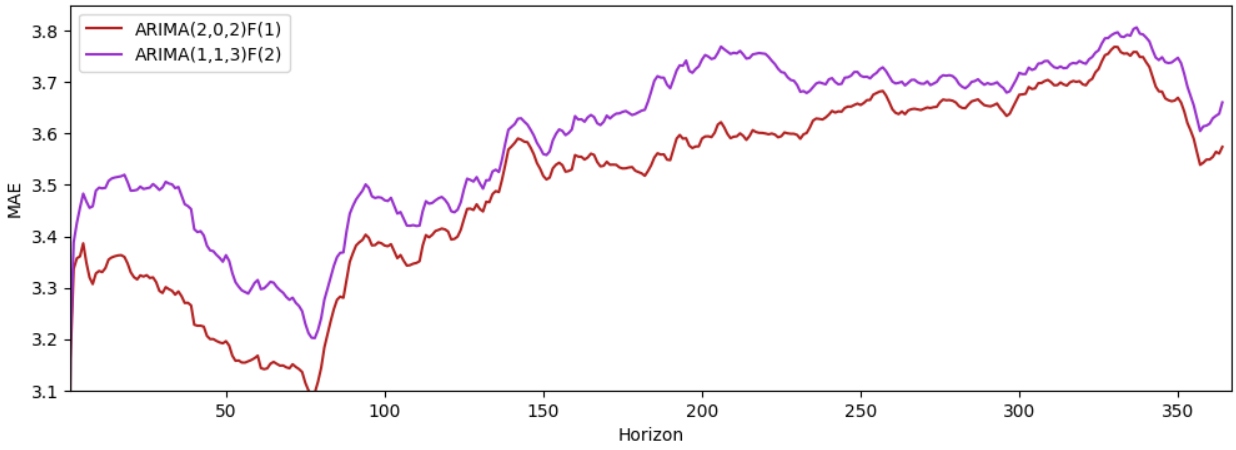
Foram comparados múltiplos modelos RHD com diversas combinações de hiperparâmetros: $p, q \in [0, 4]$, $d \in [0, 1]$ e entre 1 e 4 pares de termos de Fourier.

Para fins de comparação, esses modelos foram agrupados em dois grandes conjuntos: os modelos com $d = 0$ e os modelos com $d = 1$, uma vez que o parâmetro d modifica fundamentalmente a série, inviabilizando sua comparação via AIC [Akaike 1974]. Assim, cada grupo foi avaliado separadamente: todos os modelos foram treinados nos cinco primeiros anos de dados (21/11/2014 a 19/11/2019) e, dentro de cada grupo, selecionou-se o modelo com menor AIC. Em seguida, esses dois modelos foram comparados por meio de validação por janela deslizante [Hyndman e Athanasopoulos 2021] nos dois anos subsequentes, totalizando os sete anos utilizados na análise exploratória.

No grupo com $d = 0$, o modelo de menor AIC foi o ARIMA(2,0,2) com um par de termos de Fourier ($AIC \approx 9,485$). No grupo com $d = 1$, o modelo selecionado foi o ARIMA(1,1,3) com dois pares de termos de Fourier, apresentando AIC praticamente idêntico ao melhor modelo de $d = 0$, diferindo apenas em casas decimais. Esses dois modelos foram então avaliados por janela deslizante, com tamanho mínimo de 365 dias por janela.

A métrica escolhida como critério de comparação foi o erro absoluto médio (MAE), por ser menos sensível a valores extremos, característica especialmente vantajosa no cenário heteroscedástico observado na série.

Figure 11: Validação por janela deslizante. Fonte: elaboração própria.



A Figura 11 não apresenta o MAE de um passo à frente, pois sua inclusão distorcia a escala do gráfico, comprometendo a visualização dos demais horizontes. Para registro, o MAE de um passo à frente foi igual a $2,280935$ para o modelo com $d = 0$ e $2,319515$ para o modelo com $d = 1$. Considerando esse resultado e os horizontes seguintes, o modelo ARIMA(2,0,2) com um par de termos de Fourier apresentou o melhor desempenho tanto no curto quanto no longo prazo.

Table 3: Resultados do modelo RHD

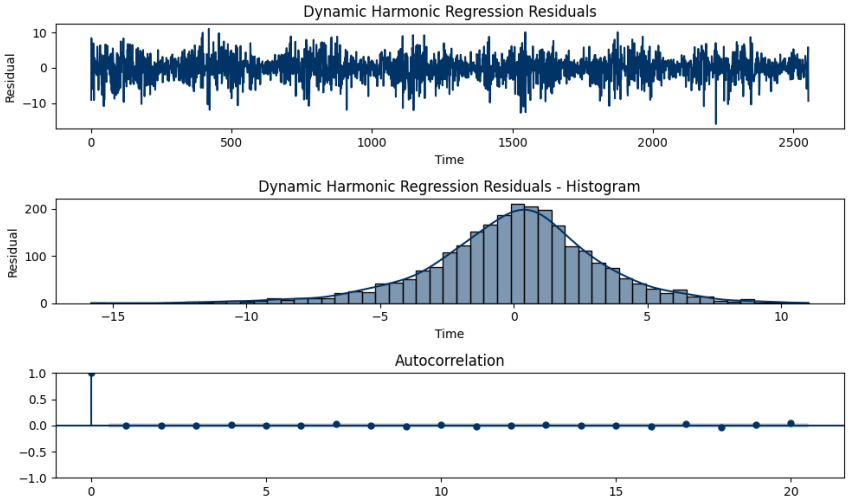
Parâmetro	**Coef.**	**Std. Err.**	**z**	**P$>\|z\|$**	**[0.025**	**0.975]**
const	284,8000	0,282	1009,899	0,000	284,247	285,353
sin_1	-12,3950	0,402	-30,844	0,000	-13,183	-11,607
cos_1	-6,4772	0,340	-19,045	0,000	-7,144	-5,811
ar.L1	1,0953	0,117	9,354	0,000	0,866	1,325
ar.L2	-0,2263	0,077	-2,933	0,003	-0,378	-0,075
ma.L1	-0,1920	0,116	-1,655	0,098	-0,419	0,035
ma.L2	-0,2933	0,036	-8,191	0,000	-0,363	-0,223
sigma2	10,0837	0,230	43,876	0,000	9,633	10,534

No. Observations	2555
Log Likelihood	-6578,051
AIC	13172,102
BIC	13218,868
HQIC	13189,062
Ljung-Box (L1)	0,00 (p = 0,95)
Jarque-Bera	238,16 (p = 0,00)
Heteroskedasticity (H)	0,79 (p = 0,00)
Skew	-0,38
Kurtosis	4,29

O sumário do modelo selecionado, Tabela 3, indica um nível médio de 284,8 Kelvin. Os coeficientes dos harmônicos sazonais, ambos altamente significativos, confirmam a presença de um ciclo anual bem definido. A estrutura autorregressiva revela forte persistência temporal, evidenciada pelo coeficiente AR(1) superior a 1 compensado por um AR(2) negativo, configurando uma dinâmica dependente dos valores anteriores, porém ainda estacionária. Os termos de médias móveis, embora um deles apresente significância marginal, contribuem para capturar variações de curto prazo não explicadas pela componente autorregressiva. A variância residual estimada implica um desvio-padrão de aproximadamente 3,2 unidades, refletindo a variabilidade diária remanescente após a remoção dos padrões sazonais e autocorrelacionados.

Os testes de diagnóstico (Tabela 3) confirmam que o modelo se ajusta adequadamente à estrutura da série. O teste de Ljung–Box [Ljung e Box 1978] indica ausência de autocorrelação significativa nos resíduos, o que também pode ser visualizado na Figura 12. O teste de Jarque–Bera rejeita a normalidade dos erros, resultado comum em séries ambientais e coerente com a heteroscedasticidade previamente observada. O teste de heterocedasticidade aponta leve variação não constante na variância residual, mas esse comportamento é esperado e não compromete o uso do modelo para previsão da média.

Figure 12: Análise Gráfica dos Residuos. Fonte: elaboração própria.



Dessa forma, conclui-se que o modelo selecionado removeu adequadamente todas as dependências temporais, permitindo que seus resíduos sejam utilizados em um modelo de aprendizado de máquina sem violar o pressuposto de independência da variável alvo.

3.3.2 Modelos de Aprendizado de Máquina

Nesta subseção, realizamos a engenharia de variáveis, a seleção de modelo, a seleção de variáveis e, por fim, o ajuste fino do modelo. Dado o contexto do problema, os modelos escolhidos como possíveis candidatos foram o *LightGBM* [Ke et al. 2017] e o *XGBoost* [Chen e Guestrin 2016], por serem algoritmos não lineares de gradiente boosting com desempenho consistentemente elevado em tarefas de previsão.

Na etapa de engenharia de variáveis, o objetivo não foi avaliar previamente a qualidade explicativa das variáveis criadas, mas sim construir um conjunto exaustivo de transformações que pudesse expor qualquer comportamento potencialmente útil para os modelos. Como o objetivo é prever 365 passos à frente, todas as variáveis foram defasadas entre 365 e 730 dias. Dessa forma, o conjunto original de seis variáveis indisponíveis no momento da previsão, *tavg*, *prcp*, *snow*, *wspd*, *pres* e *tamp*, foi expandido para 2.190 variáveis.

Além da defasagem, incorporaram-se diferenças entre dias consecutivos, prática comum para capturar mudanças relativas (momentum), ampliando o conjunto para 4.380 variáveis. Apesar de os resíduos do modelo temporal não apresentarem dependência direta no tempo, incluiu-se também variáveis de calendário (dia do ano, dia do mês, mês e ano), resultando em um total de 4.384 variáveis disponíveis antes da seleção de variáveis.

Na etapa de seleção do modelo, foram treinados um *XGBoost* e um *LightGBM*, ambos utilizando o conjunto completo de variáveis geradas. Os dois modelos foram então comparados por meio de validação cruzada com cinco dobras. Assim como na modelagem RHD, a métrica de comparação adotada foi o erro absoluto médio (MAE). O *LightGBM* apresentou o melhor desempenho, obtendo o menor MAE entre os modelos avaliados, conforme apresentado na Tabela 4.

Table 4: Desempenho dos modelos

Modelo	**MAE**
XGBoost	2.4
LightGBM	2.3

Devido ao elevado número de variáveis, a seleção foi dividida em três etapas. Na primeira etapa, foram removidas todas as variáveis cujo *gain* estimado pelo *LightGBM* foi igual a zero, reduzindo o conjunto inicial de 4.384 para 1.806 variáveis.

Na segunda etapa, como 1.806 variáveis ainda representavam um espaço excessivamente amplo, foram eliminadas todas as variáveis cujo *gain* fosse menor ou igual a 0,05% do *gain* total. Esse limiar foi definido de forma arbitrária, com o objetivo de reduzir o volume de preditores a um nível que tornasse viável a etapa subsequente. Após essa filtragem, restaram 627 variáveis.

O método *Greedy Forward Selection*[Hastie, Tibshirani e Friedman 2009] foi utilizado na terceira etapa, este método inicia com um modelo vazio e, de maneira sequencial, adiciona a cada iteração a variável que mais reduz a métrica de interesse, no caso, o MAE. Esse método é particularmente eficaz em cenários com grande número de variáveis e possíveis interações, permitindo identificar um subconjunto reduzido e altamente informativo.

Para acelerar o processo, o modelo utilizado foi configurado com as seguintes especificações:

- Número de iterações de boosting = 75;
- Profundidade máxima das árvores = 4;
- Taxa de *subsample* = 0,6;
- Execução paralelizada por passos.

O procedimento avançou por 34 iterações, até que fosse atingido o critério de parada: deterioração superior a 0,01 no MAE entre passos consecutivos. O menor MAE obtido ocorreu no passo 33, com MAE igual a 2,2495.

As variáveis selecionadas foram $pres\_703$, $tamp\_404$, $prcp\_702$, $tamp\_709\_diff$, $wspd\_718$, $prcp\_411$, $wspd\_646$, $tamp\_582$, $tamp\_523\_diff$, $tamp\_372$, $prcp\_382$, $tamp\_469$, $prcp\_703$, $tavg\_378\_diff$, $wspd\_657$, $tavg\_412$, $wspd\_416$, $tavg\_486\_diff$, $wspd\_524$, $tamp\_397\_diff$, $snow\_370\_diff$, $pres\_389\_diff$, $wspd\_597$, $pres\_440\_diff$, $tavg\_682\_diff$, $snow\_694$, $tavg\_695\_diff$, $tamp\_532\_diff$, $tamp\_382$, $wspd\_401\_diff$, $tavg\_728$, $tamp\_691\_diff$ e $prcp\_376$, onde o primeiro termo antes do subtraço indica a série de origem, o segundo representa a defasagem em dias, e a presença de um terceiro termo indica se a variável corresponde a uma diferença temporal.

Com o modelo selecionado e após a criação e seleção das variáveis, a etapa final consiste no ajuste fino do modelo, isto é, na escolha dos melhores hiperparâmetros. Inicialmente, foram identificadas as regiões promissoras do espaço de hiperparâmetros utilizando o método *Optuna* [Akiba et al. 2019], que realiza buscas inteligentes capazes de aprender, a cada tentativa, quais partes do espaço são mais favoráveis, testando combinações de forma eficiente, rápida e adaptativa.

Após essa etapa exploratória, aplicou-se um *grid search* [Hastie, Tibshirani e Friedman 2009] para o refinamento final dos hiperparâmetros. Assim como nas etapas anteriores, a métrica de seleção foi o MAE, calculado por meio de validação cruzada com cinco dobras.

Na Tabela 5, apresentam-se os intervalos dos hiperparâmetros explorados pelo *Optuna* e os valores selecionados ao final da otimização.

Table 5: Espaço de busca e resultados do optuna

Hyperparametro	**Intervalo**	**Resultado**
num_ leaves	[5; 50]	6
max_ depth	[3; 12]	11
learning_ rate	[0,01; 0,2]	0,0795599077344849
n_ estimators	[100; 800]	280
min_ child_ samples	[5; 50]	16
min_ child_ weight	[1e-5; 1,0]	0,5675840268741394
lambda_ l1	[0,0; 2,0]	1.0533880401272688
lambda_ l2	[0,0; 2,0]	1.5361414878826434
feature_ fraction	[0,6; 1,0]	0,712067260102739
bagging_ fraction	[0,6; 1,0]	0,8041159972132971
min_ gain_ to_ split	[0,0]	0,0
bagging_ freq	[1,0]	1,0

Um detalhe relevante é que foi definido *early stopping* igual a 200, de modo que o treinamento fosse interrompido sempre que o desempenho no conjunto de validação deixasse de melhorar. A partir dos resultados fornecidos pelo *Optuna*, foram definidos os valores candidatos utilizados no *grid search*, apresentados de forma arredondada para evitar o excesso de casas decimais na Tabela 6, juntamente com os hiperparâmetros correspondentes à combinação que apresentou o melhor desempenho.

Table 6: Hyperparamentros e resultados para *grid search*

Hyperparametro	**Valores**	**Resultado**
num_ leaves	2; 6; 10	
learning_ rate	0,0556919; 0,0795599; 0,1034278	
n_ estimators	224, 280, 336	
min_ child_ samples	11; 16; 21	
lambda_ l1	0.7373716; 1.0533880; 1.3694044	
lambda_ l2	1.0752990, 1.5361414, 1.9969839	
feature_ fraction	0,6408605; 0,7120672; 0,7832739	
bagging_ fraction	0.7237043; 0.8041159; 0.8845275	

As combinações desses hiperparâmetros, associadas à validação cruzada, resultaram em um total de 32.805 modelos treinados nesta última etapa. Após a seleção final dos hiperparâmetros, conduziu-se uma nova validação cruzada, desta vez com 10 dobras, para estimar o desempenho definitivo do modelo. O MAE estimado foi igual a 2.3439, valor pouco promissor, considerando que os resíduos do RHD, variável objetivo prevista pelo *LightGBM*, apresentavam MAE total de aproximadamente ≈ 2.3779.

O código utilizado para a implementação dos modelos e experimentos descritos nesta seção encontra-se disponível em: https://github.com/pedrogabrielmoura/ChicagoWeatherForecast

4 Resultados

O modelo híbrido é definido pela diferença entre a predição do *LightGBM* e a predição da Regressão Harmônica Dinâmica (RHD). Essa formulação decorre do fato de que o *LightGBM* foi treinado para prever o erro de predição da RHD, portanto, ao subtrair a predição do *LightGBM* da predição da RHD, obtém-se uma estimativa corrigida do valor futuro.

Para fins de comparação, os dados foram divididos em três grandes grupos: dados de treino, dados de calibração e dados de teste.

- Dados de treino: 2014-11-21 a 2021-11-18;
- Dados de calibração: 2021-11-19 a 2023-11-18;
- Dados de teste: 2023-11-19 a 2025-11-09.

A comparação entre o modelo RHD e o modelo híbrido foi realizada por meio de validação em janela deslizante. Em cada iteração, o treinamento utilizou pelo menos todo o conjunto de treino. Todas as janelas produziram previsões com horizonte de 365 dias.

Figure 13: Comparação de modelo via janela deslizante. Fonte: elaboração própria.

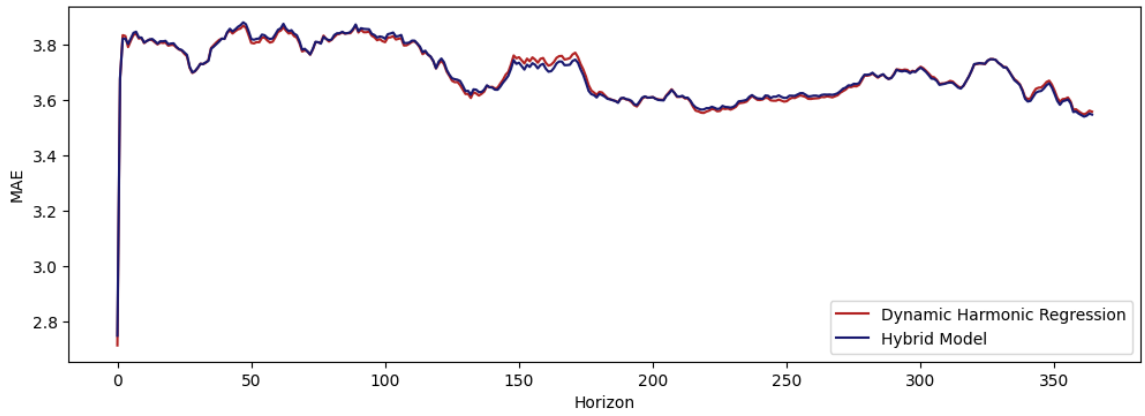
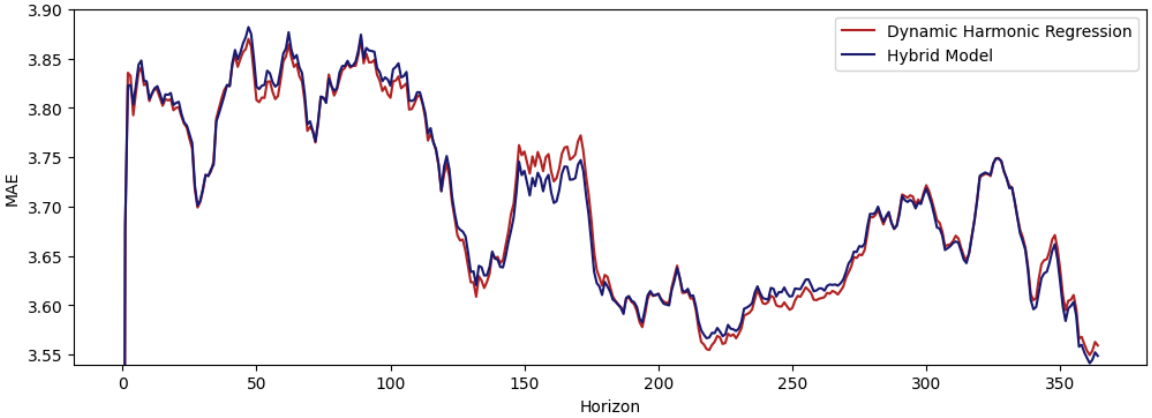


Figure 14: Comparação de modelo via janela deslizante ampliado. Fonte: elaboração própria.

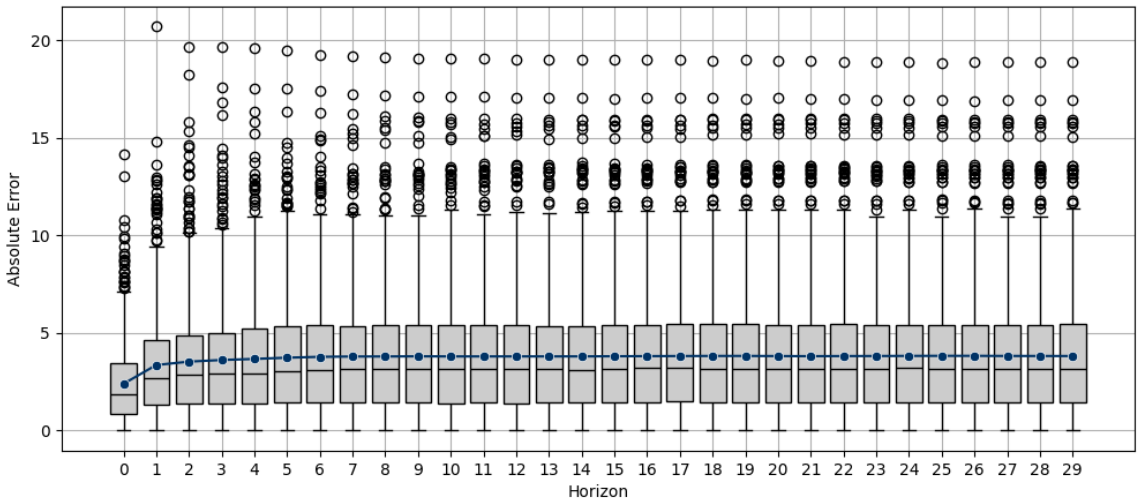


Pelas Figuras 13 e 14, não é possível identificar visualmente qual modelo apresenta melhor desempenho. Aparentemente, o modelo RHD possui vantagem em horizontes curtos, enquanto o modelo híbrido parece apresentar desempenho superior em horizontes mais longos. Para confirmar essa impressão, foi calculada a área sob a curva de ambos os modelos. Considerando todo o período, os resultados são praticamente idênticos: RHD = 1347,15 e híbrido = 1347,43. No entanto, ao restringir o cálculo às últimas 255 observações, isto é, as previsões mais distantes, obtêm-se RHD = 928,29 e híbrido = 927,96. Dessa forma, o modelo híbrido foi considerado o melhor no horizonte de interesse.

Na Figura 15, apresenta-se o comportamento do erro absoluto por horizonte de previsão. A linha representa o MAE, enquanto os boxplots ilustram a variabilidade do erro absoluto dentro de

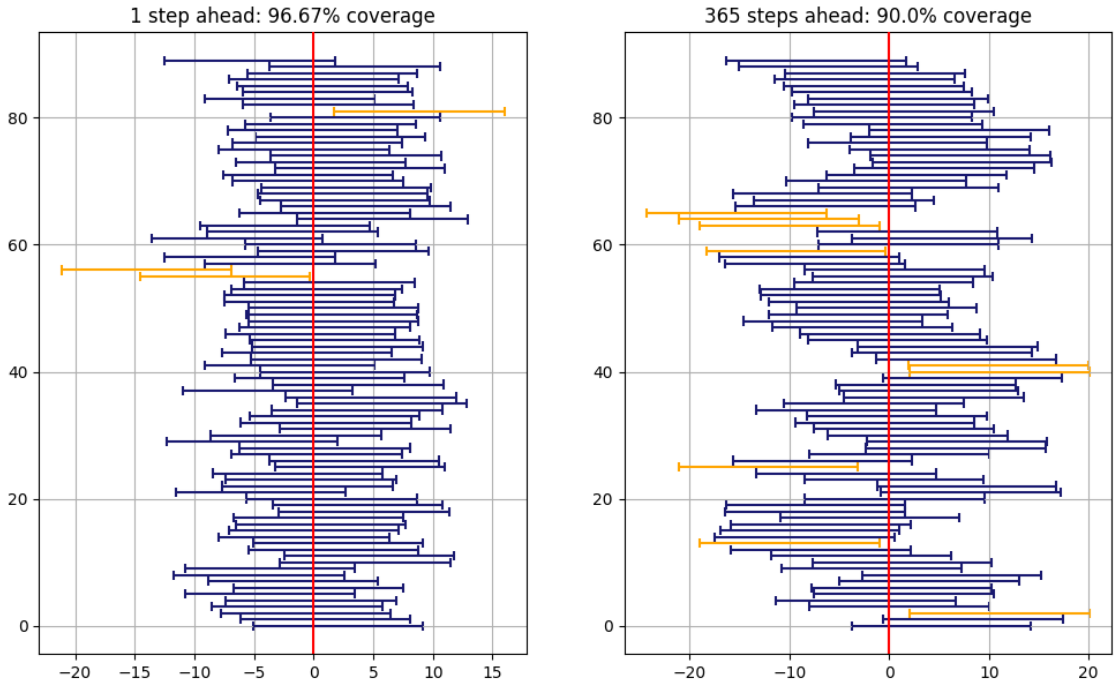
cada horizonte. A visualização foi limitada aos primeiros 30 passos à frente para evitar sobrecarga visual.

Figure 15: Comportamento do erro absoluto por horizonte. Fonte: elaboração própria.



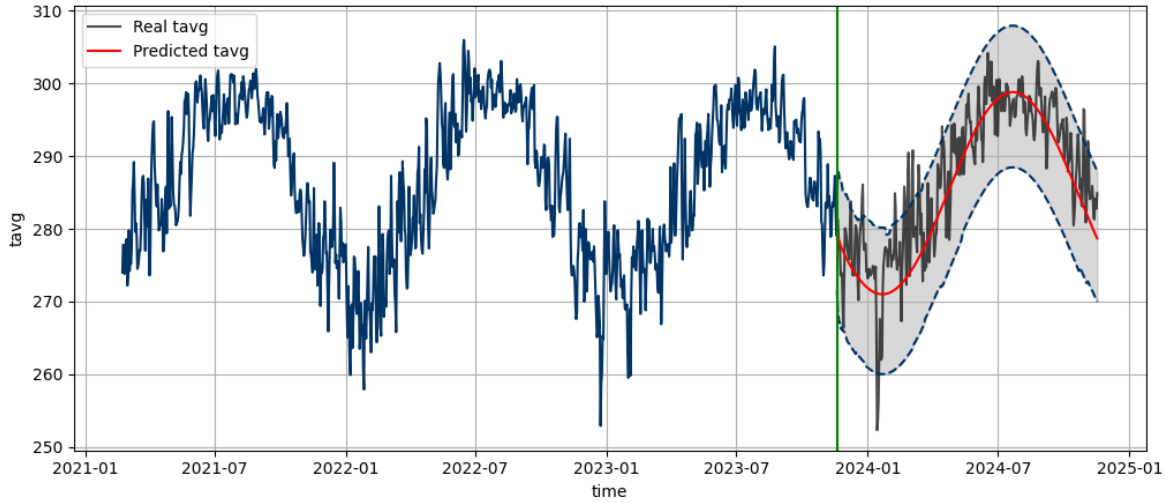
Quanto aos intervalos de confiança, mesmo que o modelo final fosse o RHD, não seria possível utilizar os intervalos teóricos, pois os resíduos não apresentam normalidade e são heteroscedásticos. Assim, adotou-se o método conhecido como *intervalo de predição empírico* [Lee e Scholtes 2014]. Na Figura 16, apresenta-se a cobertura empírica de previsões para 95% de confiança, com exemplos para um passo à frente (à esquerda) e para 365 passos à frente (à direita).

Figure 16: Cobertura dos intervalos de confiança. Fonte: elaboração própria.



Por fim, a Figura 17 apresenta o resultado final da modelagem híbrida ao longo de todo o período de teste.

Figure 17: Cobertura dos intervalos de confiança. Fonte: elaboração própria.



5 Discussão e interpretação

Embora o modelo híbrido tenha sido selecionado como a melhor alternativa em termos de desempenho preditivo, a comparação entre o custo computacional de treinamento, o esforço envolvido na construção do modelo e o ganho marginal obtido sugere que sua adoção não é necessariamente justificável. Ainda assim, é importante considerar que o cenário ao qual o modelo foi submetido foi particularmente desfavorável: uma base de dados relativamente pequena para um algoritmo como o *LightGBM*, aliada à necessidade de utilizar variáveis explicativas com defasagens extremamente longas. Em contextos menos extremos, com bases maiores, covariáveis disponíveis mais próximas do horizonte de previsão e maior riqueza informacional, é plausível que o modelo híbrido apresente desempenho significativamente superior ao da RHD isolada.

Outro ponto relevante diz respeito à perda de algumas propriedades estatísticas associadas aos modelos clássicos. O modelo híbrido, por operar sobre resíduos e não sobre a série original, inviabiliza o uso direto de ferramentas analíticas tradicionais, como intervalos de confiança derivados da variância dos erros. No presente estudo, isso não representou um problema, devido à heteroscedasticidade clara dos resíduos, que já tornaria os intervalos teóricos da RHD inadequados. Entretanto, em séries com variância aproximadamente constante, abrir mão de intervalos analíticos poderia resultar em perda expressiva de interpretabilidade e utilidade prática.

Quanto ao resultado final do modelo, prever a temperatura diária com um intervalo de confiança relativamente estreito pode ser extremamente útil para diversas aplicações operacionais e estratégicas, como planejamento energético, logística urbana, preparo para eventos climáticos específicos e otimização de recursos em setores sensíveis à temperatura. A capacidade de fornecer não apenas a previsão pontual, mas também a incerteza associada, amplia substancialmente o valor informacional do modelo, permitindo decisões mais robustas e fundamentadas.

6 Conclusão

Este estudo investigou a aplicação de um modelo híbrido composto por uma Regressão Harmônica Dinâmica (RHD) e um modelo *LightGBM* treinado para corrigir seus resíduos, no contexto particularmente desafiador de prever a temperatura média diária com 365 passos à frente. Mesmo diante de condições adversas, forte sazonalidade anual, variância sazonal significativa, covariáveis com defasagem elevada e uma base relativamente pequena para métodos de aprendizagem de máquina, o modelo híbrido demonstrou desempenho competitivo, superando a RHD em horizontes mais longos e oferecendo melhorias quando analisada a área sob a curva dos erros.

Apesar do ganho absoluto ter sido moderado, o resultado é promissor. O modelo híbrido operou em um cenário nitidamente desfavorável, no qual as técnicas de aprendizado de máquina não puderam explorar plenamente seu potencial. Ainda assim, conseguiu produzir previsões mais precisas em parte do horizonte, demonstrando sua capacidade de capturar padrões residuais que o modelo estatístico não absorveu. Esses achados indicam que, em contextos menos extremos, como em bases mais extensas, horizontes de previsão menores ou quando há variáveis exógenas contemporâneas relevantes, a vantagem do modelo híbrido tende a ser ainda mais expressiva.

Além disso, o estudo mostrou que, devido à heteroscedasticidade dos resíduos, os intervalos de confiança teóricos da RHD não seriam adequados. Nesse contexto, o uso de intervalos de predição empírica apresentou-se como uma alternativa viável, fornecendo uma medida de incerteza coerente com o comportamento real dos erros. Isso evidencia que o modelo híbrido, mesmo sem acesso a intervalos analíticos tradicionais, consegue fornecer estimativas de incerteza confiáveis, mantendo a flexibilidade da abordagem de *machine learning* ao mesmo tempo em que preserva ferramentas úteis para interpretação prática das previsões.

Os resultados também evidenciaram a importância da combinação entre métodos estatísticos interpretáveis e algoritmos de aprendizagem de máquina capazes de capturar relações não lineares, sobretudo em séries ambientais complexas. A integração dessas abordagens oferece uma estrutura que preserva a interpretabilidade da decomposição temporal e, ao mesmo tempo, adiciona poder preditivo por meio da modelagem de resíduos.

Por fim, este trabalho abre diversas oportunidades para pesquisas futuras. Entre elas, destacam-se a aplicação do modelo híbrido em bases de dados maiores, a investigação de modelos mais sofisticados, como redes neurais recorrentes, arquiteturas baseadas em Transformers ou métodos probabilísticos, a incorporação das predições de modelos de aprendizado de máquina nos termos de média móvel de modelos RHD e o uso de covariáveis exógenas contemporâneas, que poderiam ampliar significativamente o poder explicativo do componente de *machine learning*. Além disso, estudos que explorem heteroscedasticidade sazonal explícita, *quantile regression* ou modelagem bayesiana de incertezas também representam caminhos promissores.

Em síntese, mesmo diante de um cenário altamente restritivo, o modelo híbrido apresentou resultados encorajadores e demonstrou seu potencial como uma alternativa poderosa para aprimorar previsões de longo prazo em séries temporais complexas. A evidência empírica obtida sugere que, com condições mais favoráveis, essa abordagem pode superar de forma clara os métodos puramente estatísticos, consolidando-se como uma ferramenta útil e versátil em aplicações meteorológicas e ambientais.

References

[Akaike 1974]AKAIKE, H. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, IEEE, v. 19, n. 6, p. 716–723, 1974.

[Akiba et al. 2019]AKIBA, T. et al. Optuna: A next-generation hyperparameter optimization framework. In: ACM. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. [S.l.], 2019. p. 2623–2631.

[Box e Cox 1964]BOX, G. E. P.; COX, D. R. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, Wiley, v. 26, n. 2, p. 211–252, 1964.

[Box e Jenkins 1970]BOX, G. E. P.; JENKINS, G. M. *Time Series Analysis: Forecasting and Control*. [S.l.]: Holden-Day, 1970.

[Camelo et al. 2017]CAMELO, H. et al. Métodos de previsão de séries temporais e modelagem híbrida ambos aplicados em médias mensais de velocidade do vento para regiões do nordeste do brasil. *Revista Brasileira de Meteorologia*, v. 32, n. 4, 2017. Disponível em: <https://www.scielo.br/j/rbmet/a/XCmF6ssBPJvfDVJgWZVDF6t/?lang=pt>.

[Chen e Guestrin 2016]CHEN, T.; GUESTRIN, C. Xgboost: A scalable tree boosting system. In: ACM. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. [S.l.], 2016. p. 785–794.

[Cleveland et al. 1990]CLEVELAND, R. B. et al. Stl: A seasonal-trend decomposition procedure based on loess. *Journal of Official Statistics*, v. 6, n. 1, p. 3–73, 1990.

[Daubechies 1992]DAUBECHIES, I. *Ten Lectures on Wavelets*. [S.l.]: SIAM, 1992.

[Hastie, Tibshirani e Friedman 2009]HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2. ed. [S.l.]: Springer, 2009.

[Hyndman e Athanasopoulos 2018]HYNDMAN, R. J.; ATHANASOPOULOS, G. *Forecasting: Principles and Practice*. 2. ed. [S.l.]: OTexts, 2018.

[Hyndman e Athanasopoulos 2021]HYNDMAN, R. J.; ATHANASOPOULOS, G. *Forecasting: Principles and Practice*. 3. ed. OTexts, 2021. Disponível em: <https://otexts.com/fpp3/>.

[Jarque e Bera 1987]JARQUE, C. M.; BERA, A. K. A test for normality of observations and regression residuals. *International Statistical Review*, v. 55, n. 2, p. 163–172, 1987.

[Ke et al. 2017]KE, G. et al. Lightgbm: A highly efficient gradient boosting decision tree. In: *Advances in Neural Information Processing Systems*. [S.l.: s.n.], 2017. v. 30.

[Lee e Scholtes 2014]LEE, Y. S.; SCHOLTES, S. Empirical prediction intervals revisited. *International Journal of Forecasting*, v. 30, n. 2, p. 217–234, 2014.

[Liu et al. 2013]LIU, B. et al. Uncertainty in determining extreme precipitation thresholds. *Journal of Hydrology*, v. 503, p. 233–245, 2013.

[Ljung e Box 1978]LJUNG, G. M.; BOX, G. E. P. On a measure of lack of fit in time series models. *Biometrika*, Oxford University Press, v. 65, n. 2, p. 297–303, 1978.

[Montgomery, Peck e Vining 2012]MONTGOMERY, D. C.; PECK, E. A.; VINING, G. G. *Introduction to Linear Regression Analysis*. 5. ed. [S.l.]: Wiley, 2012.

[Shapiro e Wilk 1965]SHAPIRO, S. S.; WILK, M. B. An analysis of variance test for normality (complete samples). *Biometrika*, Oxford University Press, v. 52, n. 3-4, p. 591–611, 1965.

[Torrence e Compo 1998]TORRENCE, C.; COMPO, G. P. A practical guide to wavelet analysis. *Bulletin of the American Meteorological Society*, v. 79, n. 1, p. 61–78, 1998.

7 Anexo

```
"""
This module contains utility functions for time series analysis, including:
- Stationarity tests (ADF and KPSS).
- Time series plotting (basic time series and seasonal patterns).
- Autocorrelation and Partial Autocorrelation plotting.
- Time series model evaluation using ARIMA.

Each function is designed to facilitate exploration and modeling of time series data.
"""

import pandas as pd
import numpy as np
from tqdm import tqdm
import calendar
import matplotlib.pyplot as plt
import seaborn as sns
import pywt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
from scipy.stats import linregress
from statsmodels.tsa.stattools import adfuller, kpss
from datetime import timedelta

def stationary_tests(serie: pd.Series):
    """
    Perform Augmented Dickey-Fuller (ADF) and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) tests
    to check if a time series is stationary.

    Args:
        serie (pd.Series): Time series data to test.

    Example:
        stationary_tests(time_series_data)
    """
    # ADF Test
    print(f"Augmented Dickey-Fuller (ADF) - p-value: {adfuller(serie)[1]:.2%}\n"
          "\tHo: Non-Stationarity \n\t"
          "Ha: Is Stationary\n\t")

    # KPSS Test
    print(f"Kwiatkowski-Phillips-Schmidt-Shin (KPSS) - p-value: {kpss(serie)[1]:.2%}\n"
          "\tHo: Is Stationary\n\t"
          "Ha: Non-Stationarity\n\t")
    return

def ts_plot(serie: pd.Series, time_index: pd.Series, figsize=(15, 3)):
    """
    Plot a time series with different colors for each year.

    Args:
```

```
        serie (pd.Series): The time series data to plot.
        time_index (pd.Series): The time index for the data.
        figsize (tuple): The size of the plot. Default is (15, 3).

    Example:
        ts_plot(time_series_data, time_index)
    """
    # Creating a color palette for each year
    color_palette = {year: sns.color_palette("tab20", time_index.dt.year.nunique())[i] for i,

    # Plot
    plt.figure(figsize=figsize)
    sns.scatterplot(x=time_index, y=serie, hue=time_index.dt.year, palette=color_palette)
    sns.lineplot(x=time_index, y=serie, hue=time_index.dt.year, palette=color_palette, legend

    # Adjust x-ticks for better date visibility
    date_list = pd.date_range(start=time_index.min(), end=time_index.max(), freq='2QS')
    plt.xticks(date_list, date_list.strftime('%b\n%Y'))

    # Additional layout settings
    plt.legend(loc='lower center', ncols=time_index.dt.year.nunique())
    plt.grid(linestyle='--', )
    plt.xlabel("Date")
    plt.ylabel("Values")
    plt.show()

def ts_quick_insights(serie: pd.Series, time_index: pd.Series, figsize=(8, 3)):
    """
    Generate quick insights about a time series, including trend and seasonal analysis.

    Args:
        serie (pd.Series): The time series data.
        time_index (pd.Series): The time index.
        figsize (tuple): The size of the plot. Default is (15, 5).

    Example:
        ts_quick_insights(time_series_data, time_index)
    """
    # Calculate moving average
    moving_average = serie.rolling(window=30, min_periods=1).mean()

    # Linear regression for trend
    x = range(len(serie))
    slope, intercept, r_value, p_value, std_err = linregress(x, serie)

    # Trend plot
    plt.figure(figsize=figsize)

    ax1 = sns.scatterplot(x=x, y=serie)
    sns.lineplot(x=x, y=moving_average, color='orange', label='Moving Average')
    sns.lineplot(x=[0, len(serie)], y=[intercept, slope * len(serie) + intercept],
```

```
                color='red', label='Linear Regression')

    plt.xlim([-50, len(serie) + 50])
    plt.xticks([i for i, d in enumerate(time_index) if d in pd.date_range(start=time_index.mi
    ax1.set_xticklabels(pd.date_range(start=time_index.min(), end=time_index.max(), freq='2QS
    plt.grid(linestyle='--')

    plt.show()

    # Seasonal plot
    plt.figure(figsize=figsize)

    for year in time_index.dt.year.unique():
        serie_per_year = serie.loc[time_index.dt.year == year].reset_index(drop=True)
        ax2 = sns.lineplot(serie_per_year, color='black', alpha=0.3)

    # Layout for seasonal plot
    month_ticks = np.cumsum([calendar.monthrange(1997, month)[1] for month in range(1, 13)])
    plt.xticks(month_ticks, [])
    ax2.secondary_xaxis(location=0).set_xticks(month_ticks - np.diff(month_ticks, prepend=0)
                                               labels=pd.date_range(start='1997-01-01', end
    plt.grid(linestyle='--')
    plt.xlabel("\nTime Within Year")
    plt.xlim([-5, 371])

    plt.show()

def plot_acf_pacf(serie: pd.Series, focus: str = 'normal'):
    """
    Plot both the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF)

    Args:
        serie (pd.Series): The time series data.
        focus (str): Type of ACF/PACF plot. Can be 'normal', 'seasonal', or 'both'. Default i

    Example:
        plot_acf_pacf(time_series_data)
    """
    # Initialize the figure

    if focus == 'normal':
        fig = plt.figure(figsize=(15, 6))
        ax1 = plt.subplot2grid((2, 1), (0, 0))
        ax2 = plt.subplot2grid((2, 1), (1, 0))
        plot_acf(serie, lags=32, ax=ax1)
        ax1.set_title('Autocorrelation Month')
        plot_pacf(serie, lags=32, ax=ax2)
        ax2.set_title('Partial Autocorrelation Month')

    elif focus == 'seasonal':
        fig = plt.figure(figsize=(15, 6))
```

```
        ax3 = plt.subplot2grid((2, 1), (0, 0))
        ax4 = plt.subplot2grid((2, 1), (1, 0))
        plot_acf(serie, lags=400, ax=ax3)
        ax3.set_title('Autocorrelation Yearly')
        ax3.set_xlim(300, 400)
        plot_pacf(serie, lags=400, ax=ax4)
        ax4.set_title('Partial Autocorrelation Yearly')
        ax4.set_xlim(300, 400)

    else:
        fig = plt.figure(figsize=(20, 10))
        ax1 = plt.subplot2grid((3, 2), (0, 0))
        ax2 = plt.subplot2grid((3, 2), (0, 1))
        ax3 = plt.subplot2grid((3, 2), (1, 0), colspan=2)
        ax4 = plt.subplot2grid((3, 2), (2, 0), colspan=2)
        plot_acf(serie, lags=32, ax=ax1)
        ax1.set_title('Autocorrelation Month')
        plot_pacf(serie, lags=32, ax=ax2)
        ax2.set_title('Partial Autocorrelation Month')
        plot_acf(serie, lags=400, ax=ax3)
        ax3.set_title('Autocorrelation Yearly')
        ax3.set_xlim(300, 400)
        plot_pacf(serie, lags=400, ax=ax4)
        ax4.set_title('Partial Autocorrelation Yearly')
        ax4.set_xlim(300, 400)

    plt.tight_layout()
    plt.show()
    return

def ts_model_eval(data: pd.Series, len_test:int,
                  p:int, d:int, q:int, m:int = 0,
                  P:int = 0,D:int = 0,Q: int = 0,
                  steps: int = 1):
    """
    Evaluate an ARIMA model by splitting the data into training and testing sets, fitting the
    and evaluating its forecasting performance using RMSE and MAPE.

    Args:
        data (pd.Series): Time series data to model.
        len_test (int): Number of observations for testing.
        p (int), d (int), q (int): ARIMA order parameters.
        m (int): Period for seasonal ARIMA. Default is 0.
        P (int), D (int), Q (int): Seasonal ARIMA order parameters. Default is 0.
        steps (int): Number of steps for forecasting. Default is 1.

    Example:
        ts_model_eval(time_series_data, len_test=30, p=1, d=1, q=1)
    """
    # Split data
    N = len(data)
```

```
    train = data.head(N-len_test)
    test = data.tail(len_test)

    # ARIMA model fitting
    model_base = ARIMA(endog=train, order=(p, d, q),
                       seasonal_order=(P, D, Q, m))
    model_base_fit = model_base.fit()
    display(model_base_fit.summary())

    # Forecasting
    list_forecast_steps = []
    list_intv_forecast = []
    for i in tqdm(range(len_test, 0, -steps), desc=f"Forecast {len_test} days"):
        train_step = data.head(N-i)
        model = ARIMA(endog=train_step, order=(1, 1, 2))
        model_fit = model.fit()
        list_forecast_steps.append(model_fit.forecast(steps=steps))
        list_intv_forecast.append(model_fit.get_forecast(steps=1).conf_int(alpha=0.05))

    # Calculate error
    forecast_steps = pd.concat(list_forecast_steps)
    forecast_interval_steps = pd.concat(list_intv_forecast)

    error = test - forecast_steps
    rmse = np.sqrt(np.mean(error**2))
    mape = np.mean(abs(error/test))

    # Plot results
    plt.figure(figsize=(15, 4))
    sns.lineplot(train, label='Train')
    sns.lineplot(test, label='Test', color='green')
    plt.fill_between(forecast_interval_steps.index,
                     forecast_interval_steps[f'lower {data.name}'],
                     forecast_interval_steps[f'upper {data.name}'],
                     alpha=0.2)
    sns.lineplot(forecast_steps, label='Forecast', color='orange', linestyle='--')

    plt.xlim(data.index.max() - timedelta(days=int(1.5*len_test)), data.index.max())
    plt.title(f'RMSE: {rmse:.2f} MAPE: {mape:.2%}')
    plt.show()
    return model_base_fit, forecast_steps

def wavelet(signal, min_period, max_period, dt=1):
    scales = np.arange(min_period, max_period + 1)

    # Wavelet
    coeffs, freqs = pywt.cwt(signal, scales=scales, wavelet='morl', sampling_period=dt)
    periods = scales
    time = np.arange(len(signal))

    # Cone of influence approximation
```

```
    coi = np.sqrt(2) * max_period * (1 - np.abs(time - len(signal)/2) / (len(signal)/2))

    plt.figure(figsize=(12,6))
    plt.imshow(np.abs(coeffs),
            extent=[0, len(signal), periods[-1], periods[0]],
            aspect='auto',
            cmap='viridis')

    # COI ‘‘belly’’
    plt.plot(time, coi, 'w--', linewidth=1.5)
    plt.plot(time, periods[-1] - coi + periods[0], 'w--', linewidth=1.5)

    # Gray area
    plt.fill_between(range(0, len(signal)), periods[-1] - coi + periods[0], where=coi > 0, fa
    plt.fill_between(range(0, len(signal)), coi, [len(signal)]*len(signal), facecolor='lightg

    plt.xlabel("Time (days)")
    plt.xlim(500, 2050)
    plt.ylabel("Period (days)")
    plt.ylim(1, 500)
    plt.yticks(range(0,500,60))
    plt.colorbar(label='Power')
    plt.grid(linestyle='--', color='w', alpha=0.5)
    plt.show()

"""
Module for loading and saving historical weather data.

This module defines a function to download daily historical weather data from
Meteostat for a given location and timeframe. The retrieved data can be saved
to a file in a specified format and location.
"""

from pathlib import Path
from datetime import datetime
from dateutil.relativedelta import relativedelta
from meteostat import Point, Daily, units
from config import EXTRACTION_TIMEFRAME, LATITUDE, LONGITUDE, DATA_PATH_RAW


def data_loader(n_years: int = EXTRACTION_TIMEFRAME, lat: float = LATITUDE,
                long: float = LONGITUDE, save_path: str = Path(DATA_PATH_RAW)) -> None:
    """
    Downloads and saves daily historical weather data for a specified location and timeframe.

    Args:
        n_years (int, optional): Number of years to retrieve historical weather data, ending
                                 Must be a positive integer. Defaults to EXTRACTION_TIMEFRAME
        lat (float, optional): Latitude of the location. Defaults to LATITUDE from config.
        long (float, optional): Longitude of the location. Defaults to LONGITUDE from config.
        save_path (str, optional): File path where the data should be saved. Defaults to DEFA
```

```
    Returns:
        None

    Raises:
        ValueError: If n_years is not a positive integer.
    """

    # Validate timeframe parameter
    if n_years <= 0:
        raise ValueError("Number of years must be a positive integer.")

    # Initialize location based on latitude and longitude
    location = Point(lat, long)

    # Define the date range based on n_years
    end_date = datetime.now()
    start_date = end_date - relativedelta(years=n_years)

    # Fetch daily weather data for the defined location and date range
    data = Daily(location, start_date, end_date)
    data = data.convert(units.scientific)
    data = data.fetch()

    # Save the data to CSV at the specified path
    data.reset_index(inplace=True)
    data.to_parquet(save_path / 'weather_raw_data.parquet')

    return

import numpy as np
import pandas as pd
import missingno as msno
import os
import calendar

## Visualization
import matplotlib.pyplot as plt
import seaborn as sns
import pylab as py

# Statistics
from scipy import stats
import statsmodels.api as sm

# Functions
import sys
from pathlib import Path
sys.path.insert(1, Path.cwd().parents[1].as_posix())

from src.ts_utils import *
```

```
from src.data_loader import data_loader

from config import *

plt.rcParams['axes.prop_cycle'] = plt.cycler(color=['#003366'])

def visualize_quantity(serie: pd.Series):

    # Visualization
    fig, ax = plt.subplots(2, 1, figsize=(8, 4))

    sns.histplot(x=serie, kde=True, ax=ax[0])

    ax[1] = sns.violinplot(x=serie, inner="quart")
    plt.setp(ax[1].collections, alpha=.5)
    sns.boxplot(x=serie, ax=ax[1])

    plt.tight_layout()
    plt.show()

    # Description
    descr_df = serie.describe().reset_index()
    descr_df.columns = ['values', serie.name]

    return descr_df

weather_df = pd.read_parquet('../../data/raw/weather_raw_data.parquet')
display(weather_df.tail())
weather_df.info()

quantity_cols = ['tavg', 'tmin', 'tmax', 'prcp', 'snow', 'wdir', 'wspd', 'wpgt', 'pres', 'tsu
weather_df[quantity_cols] = weather_df[quantity_cols].astype(float)
weather_df[quantity_cols].head()

quality_col = 'time'
weather_df[quality_col] = pd.to_datetime(weather_df[quality_col], format="%Y-%m-%d")
weather_df[[quality_col]].head()

any(weather_df.duplicated())

any(weather_df.duplicated('time'))

for col in weather_df.columns:
    if any(weather_df[col].isna()):
        n_miss = weather_df[col].isna().sum()
        print('{column_name} - {n_missing} ({missin_percent:.2%})'.format(column_name=col,
                                                                          n_missing=n_miss,
                                                                          missin_percent= n_miss/


weather_df = weather_df.drop(columns=['wdir', 'wpgt', 'tsun'])
```

```
weather_df['time'].min(), weather_df['time'].max()


weather_df.sort_values('time', inplace=True)
weather_df.loc[weather_df['snow'].isna(), ['time', 'snow']]

weather_df = weather_df[:-11].copy()
any(weather_df['snow'].isna())

weather_df['time'].describe()


date_range = pd.date_range(start=weather_df['time'].min(), end=weather_df['time'].max(), freq
any(weather_df['time'] != date_range)

visualize_quantity(weather_df['tavg'])

sample_weather_df = weather_df.head(2000).copy()
ts_plot(sample_weather_df['tavg'], sample_weather_df['time'])

ts_quick_insights(sample_weather_df['tavg'], sample_weather_df['time'])

variance_temp = weather_df['tavg'].rolling(180).var() # Variation by semester
variance_temp = variance_temp.dropna()

plt.figure(figsize=(15, 3))
sns.scatterplot(x=variance_temp.index, y=variance_temp, alpha=0.5)
plt.show()

plt.figure(figsize=(15, 3))
sns.lineplot(weather_df['tavg'], alpha=0.5, color='blue', label='Average Temperature')
sns.lineplot(weather_df['tmin'], alpha=0.5, color='orange', label='Minimum Temperature')
sns.lineplot(weather_df['tmax'], alpha=0.5, color='red', label='Maximum Temperature')
plt.show()

weather_df['tamp'] = weather_df['tmax'] - weather_df['tmin']
sample_weather_df['tamp'] = sample_weather_df['tmax'] - sample_weather_df['tmin']

weather_df.drop(columns=['tmax', 'tmin'], inplace=True)
weather_df['tamp'].describe()


ts_plot(weather_df['tamp'], weather_df['time'])

stationary_tests(weather_df['tamp'])

weather_df['prcp'].describe()

ts_plot(sample_weather_df['prcp'], sample_weather_df['time'])
ts_quick_insights(sample_weather_df['prcp'], sample_weather_df['time'])
```

```
weather_df.loc[weather_df['prcp'] > 50].sort_values('prcp')


weather_df['snow'].describe()

weather_df['snow'].value_counts(normalize=True).head(4)

ts_quick_insights(sample_weather_df['snow'], sample_weather_df['time'])

visualize_quantity(weather_df['wspd'])


visualize_quantity(weather_df['pres'])

print('H0: The sample comes from a normal distribution\nHa: The sample does not come from a n

stats.shapiro(weather_df['pres'])

print('H0: The sample comes from a normal distribution\nHa: The sample does not come from a n
stats.jarque_bera(weather_df['pres'])

ts_quick_insights(weather_df['pres'], weather_df['time'])

fig, ax = plt.subplots(6, 2, figsize=(7, 12), gridspec_kw={'width_ratios': [3, 1]}, sharey='r

df_time = weather_df.loc[weather_df['time'].dt.year.isin([2016, 2017, 2018, 2019, 2020, 2021]
date_list = pd.date_range(start=df_time['time'].min(), end=df_time['time'].max(), freq='2QS')

color_palette = {year: sns.color_palette("tab20", df_time['time'].dt.year.nunique())[i]
                 for i, year in enumerate(df_time['time'].dt.year.unique())}

for r, col in enumerate(['tavg', 'prcp', 'snow', 'wspd', 'pres', 'tamp']):
    sns.lineplot(x='time', y=col, hue=df_time['time'].dt.year, palette=color_palette,
                 data=df_time, ax=ax[r,0], legend=False)
    ax[r,0].set_title(f'{col.title()}')
    ax[r,0].set_xticks(date_list, date_list.strftime('%b\n%Y'))
    ax[r,0].set_xlim(pd.to_datetime('2016-01-01'), pd.to_datetime('2021-01-01'))
    ax[r,0].grid()

for r, col in enumerate(['tavg', 'prcp', 'snow', 'wspd', 'pres', 'tamp']):
    sns.histplot(y=df_time[col], kde=True, ax=ax[r,1])
    ax[r,1].set_title(f'Histogram')
    ax[r,1].set_xlabel('')
    ax[r,1].set_ylabel('')


plt.tight_layout()
plt.show()

display(weather_df.head())
```

```
weather_df.info()

weather_df.to_parquet(os.path.join(DATA_PATH_WRANGLE, 'weather_sanitized.parquet'))


## Base
import os
import numpy as np
import pandas as pd

## Visualizations
import matplotlib.pyplot as plt
import seaborn as sns

# Statistic
import scipy
import statsmodels.api as sm
from statsmodels.tsa.seasonal import STL
from statsmodels.stats.diagnostic import het_breuschpagan

# Funções criadas
import sys
from pathlib import Path
sys.path.insert(1, Path.cwd().parents[1].as_posix())

from src.ts_utils import *

from config import *

plt.rcParams['axes.prop_cycle'] = plt.cycler(color=['#003366'])

df = pd.read_parquet(os.path.join(DATA_PATH_WRANGLE, 'weather_sanitized.parquet'))

df_train_valid = df.head(7 * 365).copy()

data_desc = lambda x, y: print(f"{y.title()}:\n\t- Min: {x['time'].min().strftime('%Y-%m-%d')
                                            f"\n\t- Max: {x['time'].max().strftime('%Y-%m-%d')
                                            f"\n\t- Years: {round(x.shape[0]/365, 2)}")
data_desc(df_train_valid, "train & validation data")

ts_quick_insights(df_train_valid['tavg'], df_train_valid['time'])

df_train_valid['box_tavg'], lmbda = scipy.stats.boxcox(df_train_valid['tavg'])
lmbda

ts_quick_insights(df_train_valid['box_tavg'], df_train_valid['time'])

res = STL(df_train_valid['tavg'], period=365, seasonal=31).fit()
box_res = STL(df_train_valid['box_tavg'], period=365, seasonal=31).fit()

fig, ax = plt.subplots(4, 2, figsize=(10, 6), sharex='col')
```

```
# raw
for c, decomp in enumerate([res, box_res]):
    sns.lineplot(data=decomp.observed, ax = ax[0,c])
    sns.lineplot(data=decomp.trend, ax = ax[1,c])
    sns.lineplot(data=decomp.seasonal, ax = ax[2,c])
    sns.lineplot(data=decomp.resid, ax = ax[3,c])

ax[0,0].set_title("Original tavg")
ax[0,1].set_title("Transformed tavg")

plt.tight_layout()

plt.show()

box_yhat = box_res.trend + box_res.seasonal
box_resid = df_train_valid['tavg'] - scipy.special.inv_boxcox(box_yhat, lmbda)
(res.resid ** 2).mean(), (box_resid ** 2).mean()

X = pd.DataFrame({'n_days': range(len(df_train_valid))})
X = sm.add_constant(X)


Y = df_train_valid['tavg']
model_original = sm.OLS(Y, X)
results_original = model_original.fit()

print(results_original.summary())

df_train_valid['tendency'] = results_original.predict()

Y = df_train_valid['box_tavg']
model_box = sm.OLS(Y, X)
results_box = model_box.fit()

print(results_box.summary())

df_train_valid['box_tendency'] = results_box.predict()

n = len(df_train_valid)
t = np.arange(n)

omega1 = (2 * np.pi / 365.25)
omega2 = 2 * (2 * np.pi / 365.25)
omega3 = 3 * (2 * np.pi / 365.25)

wavelet(df_train_valid['tavg'] - df_train_valid['tendency'], 1, 500)

y = df_train_valid['tavg'] - df_train_valid['tendency']
X = np.column_stack([
    np.ones(n), # Intercept
```

```
    np.sin(omega1*t),
    np.cos(omega1*t),
    np.sin(omega2*t),
    np.cos(omega2*t),
    np.sin(omega3*t),
    np.cos(omega3*t)
])

model = sm.OLS(y, X)
res = model.fit()
res.summary()

X = np.column_stack([
    np.ones(n),
    np.sin(omega1*t),
    np.cos(omega1*t),
    np.sin(omega2*t),
    np.cos(omega2*t)
])

model = sm.OLS(y, X)
res = model.fit()
res.summary()

df_train_valid['seasonal'] = res.predict()
df_train_valid['resid'] = df_train_valid['tavg'] - df_train_valid['tendency'] - df_train_vali

plt.figure(figsize=(15, 3))
sns.scatterplot(x=df_train_valid['time'], y=df_train_valid['tavg'], alpha=0.3, label="Origina
sns.lineplot(x=df_train_valid['time'], y=df_train_valid['tendency'] + df_train_valid['seasona
plt.xlabel("Time")
plt.ylabel("tavg")
plt.show()

wavelet(df_train_valid['box_tavg'] - df_train_valid['box_tendency'], 1, 500)

box_y = df_train_valid['box_tavg'] - df_train_valid['box_tendency']
box_X = np.column_stack([
    np.ones(n),
    np.sin(omega1*t),
    np.cos(omega1*t),
    np.sin(omega2*t),
    np.cos(omega2*t),
    np.sin(omega3*t),
    np.cos(omega3*t)
])

box_model = sm.OLS(box_y, box_X)
box_res = box_model.fit()
box_res.summary()
```

```
box_y = df_train_valid['box_tavg'] - df_train_valid['box_tendency']
box_X = np.column_stack([
    np.ones(n),
    np.sin(omega1*t),
    np.cos(omega1*t),
    np.cos(omega2*t)
])

box_model = sm.OLS(box_y, box_X)
box_res = box_model.fit()
box_res.summary()

df_train_valid['box_seasonal'] = box_res.predict()
df_train_valid['box_resid'] = df_train_valid['tavg'] - scipy.special.inv_boxcox(df_train_vali


plt.figure(figsize=(15, 3))
sns.scatterplot(x=df_train_valid['time'], y=df_train_valid['tavg'], alpha=0.3, label="Origina
sns.lineplot(x=df_train_valid['time'], y=scipy.special.inv_boxcox(df_train_valid['box_tendenc
             linestyle='--', color="red", label="Prediction")
plt.xlabel("Time")
plt.ylabel("tavg")
plt.show()


stationary_tests(df_train_valid['resid'])


stationary_tests(df_train_valid['box_resid'])


df_train_valid['roll_resid_var'] = df_train_valid['resid'].rolling(180).var()
df_train_valid['roll_box_resid_var'] = df_train_valid['box_resid'].rolling(180).var()
variance_temp = df_train_valid[['roll_resid_var', 'roll_box_resid_var']].dropna()

plt.figure(figsize=(15, 3))
sns.lineplot(x=variance_temp.index, y=variance_temp['roll_resid_var'], alpha=0.5, color="red"
sns.lineplot(x=variance_temp.index, y=variance_temp['roll_box_resid_var'], alpha=0.5, color="
plt.xlabel("Time")
plt.ylabel("Variance")
plt.show()

(df_train_valid[['box_resid', 'resid']]**2).mean()

## Base
import os
import pickle
import numpy as np
import pandas as pd
from itertools import product
```

```
## Visualizations
import matplotlib.pyplot as plt
import seaborn as sns

# Model
from scipy.stats import iqr, boxcox
from scipy.special import inv_boxcox
from statsmodels.tsa.arima_model import ARIMA
import statsmodels.api as sm
from scipy.stats import iqr

# Ignore all warnings
import warnings
warnings.filterwarnings("ignore")

# Funções criadas
import sys
from pathlib import Path
sys.path.insert(1, Path.cwd().parents[1].as_posix())

from src.ts_utils import *

from config import *

plt.rcParams['axes.prop_cycle'] = plt.cycler(color=['#003366'])

def fourier(order, period, t):
    comp = pd.DataFrame(index=range(0, t))
    for k in range(1, order+1):
        comp[f'sin_{k}'] = np.sin(2*np.pi*k*comp.index/period)
        comp[f'cos_{k}'] = np.cos(2*np.pi*k*comp.index/period)

    return comp

df = pd.read_parquet(os.path.join(DATA_PATH_WRANGLE, 'weather_sanitized.parquet'))

df_train = df.head(5 * 365).copy()
df_train_valid = df.head(7 * 365).copy()

df_train['time'].min(), df_train['time'].max()

df_train_valid['time'].min(), df_train_valid['time'].max()


candidates = pd.DataFrame(product(range(0, 5), repeat=3), columns=['p', 'q', 'f'])
candidates = candidates.loc[(candidates['f'] >= 1) & ~((candidates['p'] == 0) & (candidates['
                            (candidates['p'] <= 3) & (candidates['q'] <= 3)].reset_index(drop
candidates.head(3)

model_aic = []
for r, parameters in tqdm(candidates.iterrows(), total=candidates.shape[0], desc='Candidates'
```

```
    # Data
    p, q, f = parameters
    y = df_train['tavg']
    X = fourier(f, 365, len(y))

    # Model
    model = ARIMA(y, X, order=(p,0,q)).fit()
    model_aic.append([f'ARIMA({p},0,{q})F({f})', model.aic])


cand_d0 = pd.DataFrame(model_aic, columns = ['model', 'AIC']).sort_values('AIC')
cand_d0.head(5)

model_aic = []
for r, parameters in tqdm(candidates.iterrows(), total=candidates.shape[0], desc='Candidates'
    # Data
    p, q, f = parameters
    y = df_train['tavg']
    X = fourier(f, 365, len(y))

    # Model
    model = ARIMA(y, X, order=(p,1,q)).fit()
    model_aic.append([f'ARIMA({p},1,{q})F({f})', model.aic])

cand_d1 = pd.DataFrame(model_aic, columns = ['model', 'AIC']).sort_values('AIC')
cand_d1.head(5)

new_candidate = pd.DataFrame({'p': [2, 1],
                              'd': [0, 1],
                              'q': [2, 3],
                              'f': [1, 2]
                             })
new_candidate

mae_models = pd.DataFrame()
for r, parameters in new_candidate.iterrows():
    p, d, q, f = parameters
    y = df_train_valid['tavg']
    X = fourier(f, 365, df_train_valid.shape[0])

    error_list = []
    for i in tqdm(range(-2*365, -365), desc=f"Candidate {r+1}", position=0, leave=True):
        # Sets
        X_train = X[:i]
        X_valid = X[i:i+365]
        y_valid = y[i:i+365]
        y_train = y[:i]

        # Model
        model = ARIMA(y_train, X_train, order=(p,d,q)).fit()
        forecast = model.forecast(365, exog=X_valid)
```

```
        # Error
        error = abs(forecast - y_valid).reset_index(drop=True)
        error_list.append(error)

    resid_df = pd.concat(error_list, axis=1, ignore_index=True)
    mae_models[f'ARIMA({p},{d},{q})F({f})'] = resid_df.mean(axis=1)

plt.figure(figsize=(12, 4))

sns.lineplot(x=mae_models.index, y=mae_models['ARIMA(2,0,2)F(1)'], c='firebrick', label='ARIM
sns.lineplot(x=mae_models.index, y=mae_models['ARIMA(1,1,3)F(2)'], c='darkorchid', label='ARI

plt.xlabel('Horizon')
plt.ylabel('MAE')

plt.xlim(1, 367)
plt.ylim(3.1, 3.85)

plt.show()

mae_models.head(5)

df_ml = pd.concat([df_train_valid, fourier(1, 365, df_train_valid.shape[0])], axis=1)

model = ARIMA(df_ml['tavg'], df_ml[["sin_1", "cos_1"]], order=(2,0,2)).fit()
model.summary()

fig, ax = plt.subplots(3, 1, figsize=(10, 6))

sns.lineplot(model.resid, ax=ax[0])
ax[0].set_title('Dynamic Harmonic Regression Residuals')
ax[0].set_xlabel('Time')
ax[0].set_ylabel('Residual')

sns.histplot(model.resid, kde=True, ax=ax[1])
ax[1].set_title('Dynamic Harmonic Regression Residuals - Histogram')
ax[1].set_xlabel('Time')
ax[1].set_ylabel('Residual')

sm.graphics.tsa.plot_acf(model.resid, lags=20, ax=ax[2])

plt.tight_layout()
plt.show()

df_ml['ml_resid'] = model.resid
df_ml.to_parquet(os.path.join(DATA_PATH_WRANGLE, 'weather_linear_resid.parquet'))


## Base
import os
```

```
import pickle
import numpy as np
import pandas as pd
from joblib import Parallel, delayed
from tqdm_joblib import tqdm_joblib

## Visualizations
import matplotlib.pyplot as plt
import seaborn as sns

# Model
import optuna
from optuna.samplers import TPESampler #
from sklearn.model_selection import KFold, ShuffleSplit, cross_val_score, GridSearchCV
from sklearn.metrics import mean_absolute_error, make_scorer

from lightgbm import LGBMRegressor, early_stopping, log_evaluation
from xgboost import XGBRegressor

# Ignore all warnings
import warnings
warnings.filterwarnings("ignore")

import logging
logging.getLogger("lightgbm").setLevel(logging.CRITICAL)
logging.getLogger("lightgbm.engine").setLevel(logging.CRITICAL)
logging.getLogger("lightgbm.basic").setLevel(logging.CRITICAL)


# Funções criadas
import sys
from pathlib import Path
sys.path.insert(1, Path.cwd().parents[1].as_posix())

from src.ts_utils import *

from config import *


plt.rcParams['axes.prop_cycle'] = plt.cycler(color=['#003366'])



df = pd.read_parquet(os.path.join(DATA_PATH_WRANGLE, 'weather_linear_resid.parquet'))
df.info()


df['ml_resid'].describe(), abs(df['ml_resid']).mean()


list_columns = ['tavg', 'prcp', 'snow', 'wspd', 'pres', 'tamp']
```

```
for c in list_columns:
    for lag in range(365, 2*365):
        df[f'{c}_{lag}'] = df[c].shift(lag)


df.dropna().shape

not_include = {'time', 'tavg', 'prcp', 'snow', 'wspd', 'pres', 'tamp', 'sin_1', 'cos_1', 'sea
set_all_columns = set(df.columns)
set_columns = set_all_columns - not_include

len(set_columns)


for c in set_columns:
    df[f'{c}_diff'] = df[c].diff()

df.dropna().shape


df['day_month'] = df['time'].dt.day
df['day_year'] = df['time'].dt.day_of_year
df['month'] = df['time'].dt.month.astype('category')
df['year'] =  df['time'].dt.year

df.dropna().shape


set_all_columns = set(df.columns)
set_columns = set_all_columns - not_include
len(set_columns)

df.dropna(inplace=True)
X = df[list(set_columns)]
y = df['ml_resid']

model_xgb  = XGBRegressor(random_state=25, enable_categorical=True)
model_lgbm = LGBMRegressor(random_state=25)

cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=42)

mae_xgb  = -cross_val_score(model_xgb,  X, y, cv=cv, n_jobs=-2, scoring="neg_mean_absolute_er
mae_lgbm = -cross_val_score(model_lgbm, X, y, cv=cv, n_jobs=-2, scoring="neg_mean_absolute_er

print(f"XGBoost Mean MAE:  {mae_xgb.mean():.4f}")
print(f"LightGBM Mean MAE: {mae_lgbm.mean():.4f}")


model = LGBMRegressor(random_state=25).fit(X, y)

importance_gain = model.booster_.feature_importance(importance_type='gain')
```

```
feature_names = model.booster_.feature_name()

feat_imp = pd.DataFrame({
    'feature': feature_names,
    'gain': importance_gain
}).sort_values(by='gain', ascending=False)

feat_imp

feat_imp.loc[feat_imp['gain'] > 0, 'gain'].describe()


feature_pool = feat_imp.loc[feat_imp['gain'] > feat_imp['gain'].sum()* 0.0005, 'feature'].tol
len(feature_pool)

def evaluate_feature(feature, current_features, X, y):
    features_to_use = current_features + [feature]
    cv = ShuffleSplit(n_splits=3, test_size=0.25, random_state=42)
    maes = []
    for train_idx, val_idx in cv.split(X):
        X_train, X_val = X.iloc[train_idx][features_to_use], X.iloc[val_idx][features_to_use]
        y_train, y_val = y.iloc[train_idx], y.iloc[val_idx]
        model = LGBMRegressor(num_boost_round = 75, max_depth = 4, subsample = 0.6, verbose=-
        model.fit(X_train, y_train)
        y_pred = model.predict(X_val)
        maes.append(mean_absolute_error(y_val, y_pred))

    return np.mean(maes), feature

selected_features = []
remaining_features = feature_pool
performance_history = []

for step in range(1, 50):
    description = f'step_{step} (MAE: {performance_history[-1]:.4f})' if step > 1 else f'step

    # Parallel evaluation using joblib
    results = Parallel(n_jobs=-2)(
        delayed(evaluate_feature)(f, selected_features, X, y) for f in tqdm(remaining_feature
    )
    best_mae, best_feature = min(results, key=lambda x: x[0])

    # Check improvement
    if step > 10 and best_mae - performance_history[-1] > 0.01:
        print("No improvement, stopping selection.")
        break

    # Save results
    selected_features.append(best_feature)
    remaining_features.remove(best_feature)
    performance_history.append(best_mae)
```

```
selected_features, performance_history

X = X[selected_features[:performance_history.index(min(performance_history))]]
X.shape


X_values = X.values
y_values = y.values

kf = KFold(n_splits=5, shuffle=True, random_state=25)

def objective(trial):
    params = {
        "objective": "regression",
        "metric": "mae",
        "boosting_type": "gbdt",
        "verbosity": -1,

        "num_leaves": trial.suggest_int("num_leaves", 5, 50),
        "max_depth": trial.suggest_int("max_depth", 3, 12),

        "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.2),
        "n_estimators": trial.suggest_int("n_estimators", 100, 800),

        "min_child_samples": trial.suggest_int("min_child_samples", 5, 50),
        "min_child_weight": trial.suggest_float("min_child_weight", 1e-5, 1.0),

        "min_gain_to_split": 0.0,

        "lambda_l1": trial.suggest_float("lambda_l1", 0.0, 2.0),
        "lambda_l2": trial.suggest_float("lambda_l2", 0.0, 2.0),

        "feature_fraction": trial.suggest_float("feature_fraction", 0.6, 1.0),
        "bagging_fraction": trial.suggest_float("bagging_fraction", 0.6, 1.0),
        "bagging_freq": 1,
    }

    maes = []
    for train_idx, valid_idx in kf.split(X_values):
        X_train, X_valid = X_values[train_idx], X_values[valid_idx]
        y_train, y_valid = y_values[train_idx], y_values[valid_idx]

        model = LGBMRegressor(**params)

        model.fit(
            X_train, y_train,
            eval_set=[(X_valid, y_valid)],
            callbacks=[
                early_stopping(200, verbose=False),
                log_evaluation(period=0)
```

```
            ]
        )
        pred = model.predict(X_valid)
        maes.append(mean_absolute_error(y_valid, pred))

    return np.mean(maes)

n_trials = 100
optuna.logging.set_verbosity(optuna.logging.ERROR)
sampler = TPESampler(seed=25)
study = optuna.create_study(sampler=sampler, direction="minimize")

with tqdm(total=n_trials, desc="Hyperparameter Search", position=0) as pbar:
    for _ in range(n_trials):
        study.optimize(objective, n_trials=1, catch=(Exception,))
        pbar.update(1)

print("Best Optuna params:")
print(study.best_params)

best = study.best_params

grid = {
    "learning_rate": [max(1e-5, best["learning_rate"] * f) for f in [0.7, 1.0, 1.3]],
    "num_leaves": sorted(list({max(2, best["num_leaves"] + d) for d in [-4, 0, 4]})),
    "min_child_samples": sorted(list({max(2, best["min_child_samples"] + d) for d in [-5, 0,
    "feature_fraction": sorted(list({min(1.0, best["feature_fraction"] * f) for f in [0.9, 1.
    "bagging_fraction": sorted(list({min(1.0, best["bagging_fraction"] * f) for f in [0.9, 1.
    "lambda_l1": [max(0, best["lambda_l1"] * f) for f in [0.7, 1.0, 1.3]],
    "lambda_l2": [max(0, best["lambda_l2"] * f) for f in [0.7, 1.0, 1.3]],
    "n_estimators": [int(best["n_estimators"] * n) for n in [0.8, 1.0, 1.2]]
}

grid

base_model = LGBMRegressor(
    objective="regression",
    random_state=25,
    verbosity=-1,
    n_jobs=1
)

grid_search = GridSearchCV(
    estimator=base_model,
    param_grid=grid,
    scoring="neg_mean_absolute_error",
    cv=kf,
    n_jobs=-2,
    verbose=0,
    refit=True
)
```

```
total = 6561 * 5
with tqdm_joblib(total=total, leave=False, position=0) as progress_bar:
    grid_search.fit(X, y)

print("Grid best params:", grid_search.best_params_)
print("Grid best MAE:", -grid_search.best_score_)

model = LGBMRegressor(
    **grid_search.best_params_,
    objective="regression",
    random_state=25,
    n_jobs=-2,
    verbosity=-1,
)

kf = KFold(n_splits=10, shuffle=True, random_state=25)
final_lgbm = -cross_val_score(model, X, y, cv=kf, n_jobs=-2, scoring="neg_mean_absolute_error
print(f"LightGBM All MAEs: {final_lgbm}")
print(f"LightGBM Mean MAE: {final_lgbm.mean():.4f}")

light = model.fit(X, y)
pickle.dump(light, open('lightgbm_model.pkl', 'wb'))

## Base
import os
import pickle
import numpy as np
import pandas as pd
from joblib import Parallel, delayed

## Visualizations
import matplotlib.pyplot as plt
import seaborn as sns

# Model
from statsmodels.tsa.arima.model import ARIMA

# Funções criadas
import sys
from pathlib import Path
sys.path.insert(1, Path.cwd().parents[1].as_posix())

from src.ts_utils import *

from config import *

plt.rcParams['axes.prop_cycle'] = plt.cycler(color=['#003366'])

def fourier(order, period, t):
    comp = pd.DataFrame(index=range(0, t))
```

```
    for k in range(1, order+1):
        comp[f'sin_{k}'] = np.sin(2*np.pi*k*comp.index/period)
        comp[f'cos_{k}'] = np.cos(2*np.pi*k*comp.index/period)

    return comp

df = pd.read_parquet(os.path.join(DATA_PATH_WRANGLE, 'weather_sanitized.parquet'))
df.info()

light_feature = ['pres_703', 'tamp_404', 'prcp_702', 'tamp_709_diff', 'wspd_718',
       'prcp_411', 'wspd_646', 'tamp_582', 'tamp_523_diff', 'tamp_372',
       'prcp_382', 'tamp_469', 'prcp_703', 'tavg_378_diff', 'wspd_657',
       'tavg_412', 'wspd_416', 'tavg_486_diff', 'wspd_524', 'tamp_397_diff',
       'snow_370_diff', 'pres_389_diff', 'wspd_597', 'pres_440_diff',
       'tavg_682_diff', 'snow_694', 'tavg_695_diff', 'tamp_532_diff',
       'tamp_382', 'wspd_401_diff', 'tavg_728', 'tamp_691_diff']

for feat in light_feature:
    if '_diff' in feat:
        origin, lag, _ = feat.split('_')
        df[feat] = df[origin].shift(int(lag)).diff()
    else:
        origin, lag = feat.split('_')
        df[feat] = df[origin].shift(int(lag))

df = pd.concat([df, fourier(1, 365, df.shape[0])], axis=1)

df.info()

df_train = df.head(7 * 365).copy()
df_calib = df.head(9 * 365).tail(2 * 365).copy()
df_test = df.tail(df.shape[0] - (9 * 365)).copy()

df_train.shape, df_calib.shape, df_test.shape

df_train['time'].min(), df_train['time'].max()

df_calib['time'].min(), df_calib['time'].max()

df_test['time'].min(), df_test['time'].max()

with open('lightgbm_model.pkl', 'rb') as f:
    tree_model = pickle.load(f)
tree_model

df_train['light_pred'] = 0
df_calib['light_pred'] = tree_model.predict(df_calib[light_feature])
df_test['light_pred'] = tree_model.predict(df_test[light_feature])
df['light_pred'] = tree_model.predict(df[light_feature])
```

```
df_ecv = pd.concat([df_train, df_calib], ignore_index=True).sort_values('time')
X = df_ecv[["sin_1", "cos_1", "light_pred"]].copy()
y = df_ecv['tavg'].copy()

error = {'DHR': [], 'Boost': [], 'DHRXL': []}

for i in tqdm(range(-2*365, -365), position=0, leave=True):
    # Sets
    X_train = X[:i]
    X_valid = X[i:i+365]
    y_train = y[:i]
    y_valid = y[i:i+365]

    # DHR model
    ## Train Model
    DHR = ARIMA(y_train, X_train[["sin_1", "cos_1"]], order=(2,0,2)).fit()

    ## Forecast
    DHR_forecast = DHR.forecast(365, exog=X_valid[["sin_1", "cos_1"]])
    error['DHR'].append(abs(DHR_forecast - y_valid).reset_index(drop=True))

    # Boosting model
    Boost_forecast = DHR_forecast - X_valid['light_pred']
    error['Boost'].append(abs(Boost_forecast - y_valid).reset_index(drop=True))

dhr_mae = pd.concat(error['DHR'], axis=1, ignore_index=True).mean(axis=1)
boost_mae = pd.concat(error['Boost'], axis=1, ignore_index=True).mean(axis=1)


plt.figure(figsize=(12, 4))

sns.lineplot(x=dhr_mae.index, y=dhr_mae, c='firebrick', label='Dynamic Harmonic Regression')
sns.lineplot(x=boost_mae.index, y=boost_mae, c='midnightblue', label='Hybrid Model')

plt.xlabel('Horizon')
plt.ylabel('MAE')


plt.show()

plt.figure(figsize=(12, 4))

sns.lineplot(x=dhr_mae.index, y=dhr_mae, c='firebrick', label='Dynamic Harmonic Regression')
sns.lineplot(x=boost_mae.index, y=boost_mae, c='midnightblue', label='Hybrid Model')

plt.xlabel('Horizon')
plt.ylabel('MAE')

plt.ylim(3.54, 3.9)
```

```
plt.show()

np.trapezoid(dhr_mae), np.trapezoid(boost_mae)

np.trapezoid(dhr_mae.tail(255)), np.trapezoid(boost_mae.tail(255))

dic_resid = {}

for step, i in tqdm(enumerate(range(-2*365, 0)), position=0, total=2*365, leave=True):

    # Sets
    X_train = X[:i]
    X_valid = X[i:i+365] if i < -365 else X[i:]
    y_train = y[:i]
    y_valid = y[i:i+365] if i < -365 else y[i:]

    # DHR model
    ## Train Model
    DHR = ARIMA(y_train, X_train[["sin_1", "cos_1"]], order=(2,0,2)).fit()

    ## Forecast
    steps_ahead = 365 if i < -365 else -i
    DHR_forecast = DHR.forecast(steps_ahead, exog=X_valid[["sin_1", "cos_1"]])
    dic_resid[step] = (DHR_forecast  - X_valid['light_pred']- y_valid).reset_index(drop=True)

df_resid = pd.DataFrame(dic_resid).T
df_resid.head()

interval_range = df_resid.head(365).apply(lambda x: np.quantile(x, [0.025, 0.975])).T
interval_range.columns = ['lower', 'upper']

df_ecv_time_index = df_ecv.set_index('time')
df_ecv_time_index.index.freq='D'

DHR = ARIMA(df_ecv_time_index['tavg'], df_ecv_time_index[["sin_1", "cos_1"]], order=(2,0,2)).
forecast = DHR.forecast(365, exog=df_test[["sin_1", "cos_1"]].head(365)).reset_index()
forecast['lower_y'] = forecast['predicted_mean'] + interval_range['lower']
forecast['upper_y'] = forecast['predicted_mean'] + interval_range['upper']

plt.figure(figsize=(12, 5))

sns.lineplot(x='time', y='tavg', data=df_ecv.tail(1000))
sns.lineplot(x='time', y='tavg', data=df_test.head(365), color='black', alpha=0.7, label='Rea

sns.lineplot(x='index', y='predicted_mean', data=forecast, color='r', label='Predicted tavg')
sns.lineplot(x='index', y='lower_y', data=forecast, linestyle='--')
sns.lineplot(x='index', y='upper_y', data=forecast, linestyle='--')
plt.fill_between(forecast['index'], y1=forecast['lower_y'], y2=forecast['upper_y'], color='gr

plt.axvline(x=forecast['index'][0], color='g')
```

```
plt.grid()
plt.show()

all_resid = {}

for step, i in tqdm(enumerate(range(-df_test.shape[0], 0)), position=0, total=df_test.shape[0]

    # Sets
    X_train = df[:i][["sin_1", "cos_1", "light_pred"]]
    X_valid = df[i:][["sin_1", "cos_1", "light_pred"]]
    y_train = df[:i]['tavg']
    y_valid = df[i:]['tavg']

    # DHR model
    ## Train Model
    DHR = ARIMA(y_train, X_train[["sin_1", "cos_1"]], order=(2,0,2)).fit()

    ## Forecast
    DHR_forecast = DHR.forecast(-i, exog=X_valid[["sin_1", "cos_1"]])
    all_resid[step] = (DHR_forecast - y_valid - X_valid['light_pred']).reset_index(drop=True)

df_all_resid = pd.DataFrame(all_resid)
n=90
df_cover = pd.DataFrame({'min_0': [interval_range['lower'].iloc[0] - res for res in df_all_res
                                'max_0': [interval_range['upper'].iloc[0] - res for res in df_al
                                'min_365': [interval_range['lower'].iloc[364] - res for res in d
                                'max_365': [interval_range['upper'].iloc[364] - res for res in d

df_cover['color_365'] = df_cover['color_0'] = 'midnightblue'
df_cover.loc[(df_cover['min_0'] > 0) | (df_cover['max_0'] < 0), 'color_0'] = 'orange'
df_cover.loc[(df_cover['min_365'] > 0) | (df_cover['max_365'] < 0), 'color_365'] = 'orange'

cover_0 = 100 * (df_cover['color_0'] == 'midnightblue').sum() / df_cover.shape[0]
cover_365 = 100 * (df_cover['color_365'] == 'midnightblue').sum() / df_cover.shape[0]

fig, ax = plt.subplots(1, 2, figsize=(12, 7))

ax[0].axvline(x=0, color='r', zorder=3)
ax[1].axvline(x=0, color='r', zorder=3)

for i, row in df_cover.iterrows():
    ax[0].hlines(y=i, xmin=row.iloc[0], xmax=row.iloc[1], color=row.iloc[5])
    ax[1].hlines(y=i, xmin=row.iloc[2], xmax=row.iloc[3], color=row.iloc[4])


ax[0].scatter(x=df_cover['min_0'], y=df_cover.index, marker='|', color=df_cover['color_0'])
ax[0].scatter(x=df_cover['max_0'], y=df_cover.index, marker='|', color=df_cover['color_0'])
ax[1].scatter(x=df_cover['min_365'], y=df_cover.index, marker='|', color=df_cover['color_365']
ax[1].scatter(x=df_cover['max_365'], y=df_cover.index, marker='|', color=df_cover['color_365']
```

```
ax[0].set_title(f"1 step ahead: {round(cover_0, 2)}% coverage")
ax[1].set_title(f"365 steps ahead: {round(cover_365, 2)}% coverage")

ax[0].grid()
ax[1].grid()

plt.show()

df_resid_box = df_all_resid[range(0, n)].head(30).T.melt()
df_resid_box['AE'] = abs(df_resid_box['value'])

plt.figure(figsize=(12, 5))

sns.boxplot(x='variable', y='AE', data=df_resid_box, color='0.8', linewidth=1, linecolor='bla
sns.lineplot(x='variable', y='AE', data=df_resid_box.groupby('variable').mean().reset_index()
sns.scatterplot(x='variable', y='AE', data=df_resid_box.groupby('variable').mean().reset_inde

plt.ylabel("Absolute Error")
plt.xlabel("Horizon")
plt.grid()
plt.show()
```