POLITÉCNICO
DE PORTALEGRE
Escola Superior
de Tecnologia
e Gestão

COMPARAÇÃO DE MODELOS DE PREVISÃO DE SÉRIES TEMPORAIS

Aplicação a Caudais de Água Distribuída em Redes de Abastecimento

Carlos Alberto Gonçalves Pires

TRABALHO DE FINAL DE CURSO

Curso: **Mestrado em Informática**

Docente/Orientador: Mónica Vieira Martins

Ano Letivo: 2022 | 2023

Janeiro | 2024

POLITÉCNICO
DE PORTALEGRE
Escola Superior
de Tecnologia
e Gestão

COMPARAÇÃO DE MODELOS DE PREVISÃO DE SÉRIES TEMPORAIS

Aplicação a Caudais de Água Distribuída em Redes de Abastecimento

Carlos Alberto Gonçalves Pires

Unidade Curricular: **Projeto ou Estágio**

Docente: Mónica Vieira Martins

Curso: **Mestrado em Informática**

Ano Letivo: 2022 | 2023

Janeiro | 2024

"Um dos objetivos da gestão eficiente de um sistema de distribuição da água para consumo público, é garantir o seu fornecimento de forma contínua, com a qualidade e pressão requerida pelos clientes. Neste contexto, a previsão do consumo de água da rede pública em aglomerados urbanos constitui a chave para a gestão de um sistema de distribuição de água."

Manuel Herrera, Luís Torgo, Joaquín Izquierdo*, Predictive models for forecasting hourly urban water demand*.

Todos os modelos estão errados, mas alguns são úteis. – George Box

Agradecimentos

Gostaria de agradecer, em primeiro lugar, à FCC Aqualia, Sucursal em Portugal, mais concretamente ao Country Manager, Jesús Rodríguez, e ao Diretor de Produção, Artur Vidal, por terem autorizado a utilização dos dados de caudais dos quatro setores de abastecimento em causa. Sem esses dados este trabalho não teria sido possível.

De seguida, um agradecimento especial ao professor Valentim Realinho que me motivou a realizar um projeto de mestrado nesta área, após ter visto os dados de que eu dispunha provenientes de um sistema de telegestão.

Como não poderia faltar, agradeço à professora Mónica Martins por ter aceitado ser minha orientadora neste projeto, e por ter tido a paciência e resiliência que um trabalho deste tipo impõe. Soube sempre indicar-me o caminho correto de uma forma cordial e pedagógica, o que se traduziu num incentivo determinante para a realização deste trabalho.

Por último, mas não menos importante, remeto um agradecimento à minha família que, sem perceber muito bem o motivo de ter decidido matricular-me num curso de mestrado, e por isso, necessitar de passar tantas horas alheado da realidade em frente ao computador, soube dar-me a tranquilidade e o conforto que eu necessitava.

Resumo

Este trabalho visa efetuar uma comparação entre modelos de previsão de séries temporais, nomeadamente, *Holt-Winters*, ARIMA, *Long Short-Term Memory* (LSTM) e *Prophet*, quando aplicados a dados reais de caudais de água distribuídos em quatro setores de abastecimento de aglomerados urbanos, situados em zonas rurais, onde existia uma medição de caudal e respetivo registo com uma frequência máxima de uma hora. Os setores em causa foram o de Janeiro de Cima e Aldeia de Joanes, no concelho do Fundão, Degolados, no concelho de Campo Maior e Alcáçova, no concelho de Elvas, todos situados em Portugal. A comparação foi feita de forma determinística utilizando as métricas de avaliação Erro Médio Absoluto, Raiz do Erro Quadrático Médio e Coeficiente de Determinação. As métricas foram calculadas para previsões de curto prazo, 10 dias, e de longo prazo, 3 meses. Os resultados mostraram que o modelo ARIMA era mais eficiente para previsões de curto prazo, sendo o modelo LSTM mais eficiente nas previsões de longo prazo. Verificou-se que em zonas com oscilações de consumo significativas que perturbavam os padrões de uma forma aleatória, nenhum modelo conseguiu efetuar previsões minimamente aceitáveis. Por outro lado, verificou-se a potencialidade dos modelos LSTM e *Prophet* em efetuarem previsões tendo em conta a sazonalidade anual, característica determinante para deteção de anomalias.

Palavras-Chave: séries temporais, modelos de previsão, caudais distribuídos, Holt-Winters, ARIMA, LSTM, Prophet.

Abstract

This work aims to perform a comparison among time series prediction models, namely Holt-Winters, ARIMA, Long Short-Term Memory (LSTM), and Prophet, when applied to real water flow data distributed in four supply sectors of urban clusters located in rural areas. These sectors include Janeiro de Cima and Aldeia de Joanes in the Fundão municipality, Degolados in Campo Maior municipality, and Alcáçova in Elvas municipality, all situated in Portugal. The water flow was measured and recorded with a maximum frequency of one hour.

The comparison was conducted deterministically using evaluation metrics such as Mean Absolute Error, Root Mean Square Error, and Coefficient of Determination. The metrics were calculated for short-term forecasts (10 days) and long-term forecasts (3 months). The results showed that ARIMA model was more efficient for short-term predictions, while the LSTM model was more efficient for long-term predictions.

It was observed that in areas with significant consumption fluctuations that disturbed patterns randomly, no model could make minimally acceptable predictions. On the other hand, the potential of LSTM and Prophet models to make predictions considering annual seasonality, a decisive characteristic for anomaly detection, was confirmed.

Keywords: time series, prediction models, water flow distribution, Holt-Winters, ARIMA, LSTM, Prophet.

Lista de Abreviaturas, Siglas e Símbolos

ADF – Augmented Dickey-Fuller

AIC – Akaike Information Criterion

ANN – Artificial Neural Networks

BPNN – Back-Propagation Neural Network

GRNN – General Regression Neural Networks

IWA – International Water Association

kNN - k-Nearest Neighbors

KPSS – Kwiatkowski-Phillips-Schmidt-Shin

LSTM – Long Short-Term Memory

MAE – Erro Médio Absoluto

MSE – Erro Quadrático Médio

PP – Philips-Perron

R^2 – Coeficiente de Determinação

RMSE – Raiz do Erro Quadrático Médio

RNN – Recurrent Neural Networks

SMAPE – Symmetric Mean Absolute Percentage Error

SVM – Support Vector Machines

VRP – Válvula Redutora de Pressão

ZMC – Zona de Medição e Controlo

ÍNDICE GERAL

ÍNDICE DE ANEXOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

CAPÍTULO I – INTRODUÇÃO

Considerou-se importante realizar um projeto sobre um tema com aplicabilidade real significativa, que se traduzisse numa base substancial para a elaboração de trabalhos futuros mais específicos. Este capítulo efetua assim o enquadramento e justificação, perante situações reais, das necessidades emergentes a que o tema pretende dar resposta, apresentando os objetivos gerais e específicos, a metodologia e meios utilizados e a estrutura geral do trabalho.

Enquadramento e Justificação do Tema

A escassez de água representa um constrangimento muito grande em termos de integração social e desenvolvimento económico. Embora o setor da agricultura seja o mais afetado devido a representar 80% das necessidades, o uso doméstico também tem vindo a demonstrar a mesma tendência ao longo dos anos devido ao crescimento populacional, estilo de vida e aumento da temperatura global. Estas alterações provocadas pela ação humana no ciclo hidrológico, juntamente com o aquecimento global, irão provocar alterações na disponibilidade de água e na sua procura, no sentido em que a sua disponibilidade será cada vez menor e a sua procura será cada vez maior. Este problema obrigará à realização de melhorias de planeamento tanto ao nível da gestão de reservas de água como ao nível da reciclagem de água, utilização eficiente por parte dos utilizadores, e, finalmente, à criação de sistemas de monitorização em tempo real das reservas de água e da procura dos utilizadores [1].

Este trabalho focou-se principalmente na monitorização em tempo real da procura dos utilizadores, ou seja, nos caudais fornecidos ao longo do tempo às redes de distribuição de água. Neste ponto é importante referir que dependendo do tamanho da rede de distribuição e do número de utilizadores, pode ser necessário subdividir essa rede em redes mais pequenas, mais comummente chamadas de Zonas de Medição e Controlo (ZMC).

Um dos objetivos críticos da Operação e Gestão de Redes de Distribuição de Água é o de aumentar a eficiência e eficácia do fornecimento para uma procura de água específica ao menor custo. Dividir a rede original em ZMCs oferece muitas vantagens,

tais como: a) Redução da água não faturada pelo controlo de perdas; b) Simplificação da gestão de pressões utilizando válvulas redutoras de pressão (VRPs); c) Rápida deteção de roturas; d) Isolamento de zonas possibilitando limitar problemas de contaminação; e) Possibilidade de criação de ZMCs com a sua própria origem de água permitindo um melhor controlo da qualidade da água. A IWA (*International Water Association*) indica que a dimensão de uma ZMC deve ser expressa em número de alojamentos (utilizadores com contador) e varia entre 500 e 5000 alojamentos em zonas urbanas, mas a sua dimensão pode variar dependendo da localização e das características físicas de cada sistema [2].

Neste estudo, os caudais em causa, já resultavam da criação de ZMCs e dos registos dos caudalímetros existentes a montante destas. Dependendo da sua dimensão, a ZMC podia ser todo o aglomerado urbano, como no caso de Janeiro de Cima e Degolados ou apenas uma zona, como no caso de Aldeia de Joanes, no Fundão, que representava o consumo de uma Aldeia confinante com a cidade do Fundão, e Alcáçova, em Elvas, que representava o consumo do centro histórico.

Objetivos Gerais e Específicos

O objetivo geral deste trabalho final de mestrado era o de verificar qual o melhor modelo de previsão de séries temporais no que respeita aos caudais de água distribuída numa rede de abastecimento de água para consumo público, de entre os escolhidos para a experiência. Foram escolhidos os modelos, que, de acordo com a pesquisa bibliográfica realizada e com as características evidenciadas pelos mesmos, pareceram ser os mais interessantes de serem testados. Realizou-se uma abordagem que englobou modelos estatísticos clássicos (*Holt-Winters* e ARIMA) passando por um modelo baseado em redes neuronais recorrentes (LSTM) e terminando com um modelo baseado em métodos estatísticos, mas com uma abordagem de modelação de dados mais moderna (*Prophet*).

No que respeita aos objetivos específicos, estes dividiram-se em 4 e constituíram as etapas de desenvolvimento do trabalho:

a) Recolha dos dados

Neste ponto foi efetuada a recolha dos dados dos caudais distribuídos nos aglomerados urbanos de Janeiro de Cima e Degolados bem como dos setores de Aldeia de Joanes, na cidade do Fundão e Alcáçova, na cidade de Elvas. Os dados encontravam-se em Bases de Dados *mySQL* e *SQL Server* que faziam parte de sistemas de telegestão e foram obtidos através de uma aplicação desse sistema que permitia exportar os dados mensalmente para ficheiros CSV e TXT, tendo-se obtido assim um total de 15 ficheiros por local.

b) Pré-processamento dos dados

Os ficheiros CSV foram incluídos num único ficheiro por local contendo apenas como colunas o *Datetime* e o valor do caudal. Os ficheiros TXT foram convertidos para CSV e depois foram tratados da mesma forma que os primeiros.

Ainda que cada modelo pudesse exigir um pré-tratamento específico, nesta fase foram analisadas algumas características intrínsecas deste tipo de séries temporais, como a existência de sazonalidade e a tendência que pudessem evidenciar.

c) Aplicação dos modelos de previsão aos dados

Esta foi a etapa principal deste trabalho e permitiu verificar a aplicabilidade dos modelos aos dados. Interessou conhecer a maior ou menor dificuldade com o pré-processamento específico de cada modelo e as transformações necessárias a realizar antes da aplicação de cada modelo.

Ainda neste ponto interessava saber e o pré-processamento específico e o treino do modelo poderiam ser realizados de forma automática e com o mínimo de intervenção humana possível.

d) Avaliação dos resultados dos modelos

Sendo este o último e o mais importante objetivo específico, o mesmo remete para a comparação dos resultados de previsão de curto prazo e de longo prazo com os respetivos dados reais. Foram utilizadas métricas que permitiram fazer uma avaliação determinística para aferir qual dos modelos poderia fazer uma previsão mais aproximada da realidade nos dois prazos considerados. Nesta avaliação não foi considerado o tempo e os recursos computacionais necessários à utilização de cada

modelo. Da mesma forma, não foram realizados testes estatísticos de hipóteses para aferir a superioridade de um modelo, perante os outros, para determinado fim.

Metodologia e Meios Utilizados

A metodologia utilizada consistiu na realização de 24 ficheiros de desenvolvimento do tipo *Jupyter Notebooks*, sendo os mesmos divididos em conjuntos de 4 ficheiros, um por local. Foi desenvolvido um conjunto para efetuar as rotinas de pré-processamento dos dados, outro para a análise dos mesmos e depois 4 conjuntos de 4 ficheiros para a aplicação de cada modelo e obtenção dos resultados.

Os meios utilizados foram um computador *MacBook Pro* com processador 2.6 GHz *Intel Core i7* de 6 núcleos, 16GB de memória *RAM* e placa gráfica *AMD Radeon Pro 5300M* com 4GB. Sistema operativo *macOS Sonoma*, ambiente de desenvolvimento *Anaconda Navigator* e *Visual Studio Code* e linguagem de programação *python*. Infelizmente não foi possível tirar partido da placa gráfica na compilação do modelo LSTM por uma limitação da biblioteca em causa quando aplicada no *macOS* a correr em processadores *Intel*.

Estrutura Geral do Trabalho

Este trabalho encontra-se dividido em 6 capítulos. O primeiro capítulo trata de assuntos introdutórios, onde se explica o enquadramento e a justificação do tema, bem como onde se apresentam os objetivos gerais e específicos. É ainda apresentada a metodologia, os meios utilizados e a estrutura do trabalho. O segundo capítulo apresenta os modelos considerados, fazendo uma pequena introdução teórica de cada um. Ainda neste capítulo é apresentada a pesquisa bibliográfica realizada substanciando o estado da arte. No terceiro capítulo são apresentados o pré-tratamento e a análise realizada aos dados, apresentando a metodologia e os resultados obtidos para cada local considerado. O quarto capítulo trata da aplicação dos modelos às séries temporais de cada local. Apresenta-se a metodologia comum e específica de cada modelo e depois os resultados obtidos para cada modelo e cada localidade. O quinto capítulo é um capítulo de discussão onde são apresentados os resultados dos melhores modelos em cada localidade e onde se comparam os resultados obtidos com

os observados na revisão bibliográfica. Finalmente, o capítulo sexto apresenta as conclusões do trabalho com indicação dos modelos mais adequados, limitações encontradas e trabalhos futuros.

CAPÍTULO II – REVISÃO DA LITERATURA

Neste capítulo apresenta-se uma revisão da bibliografia existente relacionada ou não com o tema do trabalho. O objetivo foi o de perceber quais os modelos que têm sido mais estudados quando aplicados a este tipo de dados. Apresenta-se ainda uma breve introdução teórica sobre os modelos escolhidos para o trabalho bem como sobre as métricas consideradas para avaliação dos resultados dos mesmos.

2.1. Estado da Arte

Para a realização deste trabalho foi realizada uma pesquisa bibliográfica com o objetivo de conhecer trabalhos realizados na mesma área, mas também noutras áreas. O objetivo desta pesquisa foi o de ajudar a selecionar os modelos de previsão a considerar para o tipo de série temporal em causa.

Considerou-se que a escolha dos modelos de previsão de séries temporais a comparar dependeria das características das séries temporais em causa e da literatura existente sobre este tipo de dados. A quantidade de modelos existentes com possibilidade de aplicação a previsão de séries temporais era muito vasta e a literatura existente sobre a sua comparação era por vezes contraditória no que se referia ao melhor desempenho.

Existe uma necessidade de aumentar a eficiência dos sistemas de distribuição de água estabelecendo a gestão dos sistemas como ativa, baseada na previsão dos consumos e não como reativa, baseada nos consumos que vão ocorrendo. A utilização de metodologias de previsão de consumos de água pode resultar numa redução dos custos de operação em cerca de 18% [3].

De acordo com o estudo referido em [4], as três principais características de uma série temporal são: a) a tendência, que pode ser crescente ou decrescente e pode assumir uma grande variedade de padrões, como linear, exponencial, atenuada e polinomial; b) a sazonalidade, que se refere à ocorrência de padrões cíclicos de variação que se repetem em intervalos de tempo constante; c) os resíduos, que são as flutuações de curto prazo que não são sistemáticas nem previsíveis. Este estudo dividiu os modelos de previsão entre paramétricos e não paramétricos. Dentro dos paramétricos colocou o *Exponencial Smoothing* e o ARIMA e os não paramétricos subdividiu-os em Globais e

Locais. Os modelos paramétricos também podem ser chamados de estatísticos e requerem um conhecimento do tipo de distribuição dos dados para construir os modelos. Os modelos não paramétricos, são também chamados de modelos de *Machine Learning*, e, em oposição aos modelos estatísticos não carecem do conhecimento do tipo de distribuição dos dados. Indica o estudo que estes métodos demostram uma boa performance mesmo nos casos em que as séries temporais apresentam comportamentos não lineares. No caso dos modelos Globais, o estudo coloca aqueles que necessitam de todas as observações disponíveis para construção dos modelos, e dá como exemplo as *Artificial Neural Networks* (ANN) e as *Support Vector Machines* (SVM). No outro caso o estudo coloca os modelos existentes que constituem uma variação do algoritmo kNN. O estudo faz uma comparação dos diferentes tipos de modelos recorrendo a diferentes tipo de séries temporais e apresenta como resultado que os modelos SARIMA (uma variação do modelo ARIMA mas com sazonalidade), SVM e kNN-TSPI (variação do algoritmo kNN) são os que apresentam resultados mais promissores na modelação de dados temporais e na sua previsão.

Um estudo de aplicação dos modelos ARIMA e *Holt-Winters* para previsão da evolução da doença COVID-19, na India, efetuando previsões de novos casos e de mortes, num horizonte temporal de 20 dias, evidenciou uma melhor performance do modelo ARIMA face ao *Holt-Winters*. O modelo ARIMA, ainda que os dados tivessem evidenciado alguma autocorrelação, conseguiu obter precisões acima dos 99%, tanto para casos confirmados como para mortes, enquanto o *Holt-Winters* não foi além dos 87,9% e 95,6%, respetivamente [5].

Com o objetivo de prever o consumo de água dos utilizadores habitantes de um edifício residencial durante a hora seguinte, os autores do estudo referido em [6] utilizaram *Recurrent Neural Networks* (RNN), nomeadamente *Long Short-Term Memory* (LSTM) e *Back-Propagation Neural Network* (BPNN). O estudo evidencia que o modelo LSTM apresenta melhores resultados do que o BPNN mas deve ter-se em conta o horizonte curto da previsão de apenas uma hora, bem como o facto da série temporal ser apenas de um edifício onde residem duas pessoas.

Quando se compara a eficácia do modelo clássico ARIMA com o modelo de *Deep Learning* LSTM, aplicados a séries temporais de valores de índices financeiros, no que

respeita à percentagem de redução da métrica RMSE, entre os valores do modelo ARIMA e os valores do modelo LSTM, verifica-se uma redução muito substancial demonstrando uma performance muito elevada do último face ao primeiro [7]. Contudo, deve ter-se em conta que a previsão testada é de apenas um único passo, ou seja, procurou prever-se apenas o próximo valor, o que significa uma previsão de muito curto prazo.

De uma forma geral, enquanto um modelo autorregressivo assume que um valor de uma série temporal é função de uma combinação linear de valores passados e os processos de média móvel representam os efeitos que os resíduos produzem na série permitindo representar a dinâmica de muitas séries temporais, é também certo que muitas séries apresentam comportamentos não-lineares deixando os métodos clássicos sem capacidade para as prever, o que tem levado ao desenvolvimento de muita investigação dentro dos modelos que têm como base ANN [8].

Os modelos Machine Learning têm sido propostos na literatura académica como alternativa aos modelos estatísticos, mas sem evidência cabal acerca da sua performance relativa. Literalmente existem centenas de artigos que propõem novos algoritmos de Machine Learning, sugerindo metodologias avançadas e aumentos de precisão. Os modelos baseados em Redes Neuronais Profundas, *Deep Learning*, têm sido explorados como uma solução para melhorar as previsões de séries temporais, mas a sua superioridade é caracterizada por ter algumas limitações, por ex.: As suas conclusões baseiam-se em apenas uma ou poucas séries temporais, levantando questões acerca da significância estatística dos seus resultados; Os modelos são avaliados num horizonte de curto prazo, apenas com a previsão do próximo valor, não considerando o médio e longo prazo; A falta de utilização de padrões de referência para comparar a eficiência dos modelos de Machine Learning em relação aos clássicos [9]. Para além disso, os modelos de *Deep Learning* apresentam como exigência a existência de dados em intervalos bem definidos, o que se torna um constrangimento grande cada vez que se possui uma série de dados com observações em falta ou que possuam uma periodicidade aleatória [10].

Dentro dos modelos de *Deep Learning* foram encontradas abordagens diferentes com o objetivo de comparar a sua precisão. Em [11], é apresentado um estudo que pretende

prever o consumo de água mensal de acordo com os efeitos do clima em Yazd, Irão. Neste estudo são apresentados dois modelos LSTM, o primeiro univariável apenas com os valores dos consumos de água mensais, e o segundo, multivariável, contando também com os valores da temperatura do ar mensal. O estudo compara os resultados dos dois modelos pela métrica RMSE e comprova que esta métrica é inferior no modelo multivariável concluindo que este supera a capacidade de previsão do modelo univariável por ter erros de previsão mais baixos. Conclui ainda sobre a importância da adição dos fatores climáticos a este tipo de modelo de previsão.

No estudo referido em [12], é apresentado uma investigação sobre a aplicação de modelos LSTM para a previsão do consumo de água do dia seguinte com o objetivo de otimizar os sistemas de bombagem. O estudo efetua uma comparação da capacidade de previsão dos modelos LSTM em diferentes abordagens, com modelos autorregressivos e até com modelos mais simplistas. Conclui sobre a facilidade com que os modelos LSTM permitem incluir informação adicional, como o dia da semana ou as datas festivas, bem como a possibilidade de se obterem bons resultados com poucos dias de treino. Deixa evidenciada a superioridade deste tipo de modelo perante os outros.

Existem também casos de estudos que utilizam modelos híbridos para demonstrar que os modelos são mais eficazes juntos do que em separado. No estudo referido em [13] é realizado um trabalho empírico para prever o consumo de água diário numa cidade da Arábia Saudita utilizando o modelo ARIMA na sua variação ARMA e o modelo *General Regression Neural Network* (GRNN) que é um modelo de *Deep Learning*. Os autores concluem que a utilização conjunta dos modelos produz resultados melhores do que quando aplicados em separado. No entanto a comparação não é justa uma vez que o modelo GRNN é treinado com mais variáveis, e, portanto, mais informação, logo, é natural que este, em conjunto com o primeiro, produza previsões mais acertadas.

Em [14], são apresentados resultados de previsão de séries temporais de consumo de água no contexto industrial, onde se demonstra que um modelo híbrido composto por uma combinação do modelo *Holt-Winters* com ANN e SARIMA apresentam a melhor performance.

Um modelo mais recente, o *Prophet*, desenvolvido pela equipa do *Facebook*, apresenta-se como uma abordagem diferente baseada em componentes aditivos capaz de lidar com tendências não lineares, vários tipos de sazonalidade (anual, semanal e diária) e com os efeitos dos dias de férias e festivos. Possui uma forte robustez para lidar com dados em falta, mudanças de tendência e com *outliers* [15].

Uma comparação entre os modelos ARIMA, LSTM e *Prophet*, quando aplicados a séries temporais de produção de petróleo mostrou que os modelos ARIMA e LSTM tiveram melhor desempenho do que o *Prophet*. O modelo ARIMA demostrou uma robustez particular e o *Prophet* conseguiu captar as flutuações causadas durante o inverno, o que foi considerado uma característica singular entre os 3 modelos [16].

No setor dos serviços de saúde, uma comparação entre os modelos de previsão de séries temporais, ARIMA e *Prophet*, realizado no Reino Unido, mostrou que, em termos globais o *Prophet* apresentou melhores resultados do que o modelo ARIMA, no entanto foram observadas algumas limitações relacionadas com casos particulares em que não existiam fortes padrões sazonais. Nesses casos o modelo ARIMA apresentava um melhor desempenho. Da mesma forma, nos casos em que os dados continham muitos *outliers* provenientes de erros de medição, o modelo *Prophet* produzia previsões baseadas nesses *outliers* ao contrário do ARIMA [17].

Depois de tudo o exposto anteriormente conclui-se que é notória a complexidade do exercício de seleção dos melhores modelos a aplicar ao trabalho em questão. O melhor modelo será, em princípio, aquele que melhor representar os caudais futuros com base nos caudais registados no passado, sabendo que existe uma sazonalidade intrínseca e um padrão aparente e sabendo que o futuro é desconhecido e o passado não se repete.

2.2. Modelos Considerados

Face ao resultado da revisão bibliográfica realizada ficou bastante claro que deveriam ser utilizados tanto modelos clássicos como modelos modernos. Nos modelos clássicos, pelos resultados obtidos da bibliografia, consideraram-se dois modelos estatísticos, sendo o primeiro mais simples e o segundo mais evoluído. O primeiro escolhido foi então o *Holt-Winters* e o segundo o modelo ARIMA. No caso dos modelos mais modernos, optou-se por um modelo de *Deep Learning*, o LSTM, e por um modelo

híbrido que combina elementos de métodos estatísticos tradicionais com técnicas mais modernas de *Machine Learning*, o *Prophet*. Este último apresenta ainda a faculdade de lidar bem com dados em falta, que, não sendo o caso das séries deste estudo porque as lacunas encontradas foram resolvidas no pré-processamento, é uma característica importante a ter em conta.

2.2.1. *Holt-Winters*

O modelo de *Holt-Winters* foi desenvolvido a partir do modelo de *Holt* com o objetivo de resolver o problema da existência de sazonalidade nos dados. O modelo é composto por uma equação de previsão e três equações de suavização: uma para o nível, uma para a tendência, e uma terceira para a sazonalidade, com os parâmetros de suavização correspondentes [18].

Existem duas variações neste método que diferem na natureza da componente sazonal. O método aditivo é mais indicado quando as variações sazonais são aproximadamente constantes ao longo da série, já o multiplicativo deve ser utilizado quando as variações sazonais vão mudando de forma proporcional ao nível da série. Com o método aditivo, a componente sazonal é expressa em termos absolutos na escala da série observada, e na equação de nível, a série é ajustada sazonalmente através da subtração da componente sazonal. Dentro de cada período sazonal, a componente sazonal somará aproximadamente zero. Já no método multiplicativo, a componente sazonal é expressa em termos relativos (percentagens), e a série é ajustada sazonalmente dividindo-a pela componente sazonal. Dentro de cada período sazonal, a componente sazonal somará aproximadamente m, sendo m o número de conjuntos de observações de cada ciclo sazonal [18].

2.2.2. ARIMA

ARIMA significa Autoregressive Integrated Moving Average, trata-se de um modelo estatístico de previsão baseado no princípio de que a informação sobre valores passados pode, por si só, ser suficiente para prever os valores futuros. O modelo combina componentes de autorregressão (AR), média móvel (MA) e diferenciação (I) para capturar diferentes aspetos da dinâmica da série temporal. Os modelos ARIMA

são caracterizados pela ordem dos seus três parâmetros: ARIMA(p, d, q), em que p corresponde à ordem do componente autorregressivo, q à ordem do componente de média móvel e d representa o número de diferenciações necessárias afim de tornar a série estacionária. O componente autorregresivo (p) estabelece uma relação entre os valores passados da série temporal e o valor atual, assumindo que o valor atual depende de forma linear dos seus valores anteriores. O componente de média móvel estabelece uma relação entre os erros de previsão passados e o valor atual assumindo que o valor atual depende linearmente dos erros de previsão passados. A ordem q representa o número de termos de erros que devem ser incluídos no modelo. Finalmente, o componente de integração efetua diferenciações na série temporal com o objetivo de obter uma série estacionária. A ordem do componente d representa o número de diferenciações necessárias até se obter uma série estacionária [19].

2.2.3. LSTM

LSTM significa *Long Short-Term Memory* e trata-se de um caso particular das redes neuronais recorrentes (RNN) que possuem no seu interior células de estado contextuais que atuam como células de memória de longo prazo ou de curto prazo. O resultado de uma rede LSTM é modulado pelo estado dessas células. Esta propriedade é muito importante nos casos em que se pretende que as previsões dependam de valores anteriores historicamente distantes e não apenas dos valores anteriores mais próximos. Adicionar uma unidade LSTM numa rede neuronal recorrente é como adicionar uma unidade de memória com capacidade de se recordar dos primeiros eventos ocorridos deste o início [20].

2.2.4. *Prophet*

O modelo *Prophet* é um modelo de previsão para séries temporais, criado pelo Facebook, baseado num modelo aditivo onde tendências não lineares são ajustadas com sazonalidades que podem ser anuais, semanais e/ou diárias, para além dos efeitos gerados pelos períodos festivos. Funciona melhor com séries temporais com fortes efeitos sazonais e vários períodos de dados históricos. É robusto a lidar com dados em falta e mudanças de tendências bem como a lidar com *outliers*. A sua equação de

previsão resulta da soma da equação de tendência, que modela as variações não periódicas, com a equação de sazonalidade, que representa as variações periódicas, sejam estas anuais, semanais ou diárias, com a equação que representa as variações ocasionadas pelos dias festivos e finalmente com o erro que representa todas as variações não acolhidas pelo modelo e que se assumem como seguindo uma distribuição normal [21].

2.3. Trabalhos Relacionados

No ponto 2.1. foram referidos diversos estudos encontrados na revisão bibliográfica efetuada. Em alguns casos foram referidos estudos com trabalhos realizados dentro da área deste projeto, pelo que, importa referir de que forma este trabalho se diferencia dos restantes evidenciando a sua relevância.

Verificou-se que, dentro dos casos referidos, encontraram-se estudos que procuravam efetuar previsões de consumos de água de muito curto prazo. Em [6], apresentam-se resultados de previsão apenas para a hora seguinte, em [12], procura-se prever o consumo de água do dia seguinte como um valor global, em [14] efetuam-se previsões horárias, mas apenas se dispõe de 6 semanas com dados de treino e procura-se prever o consumo horário de 1 semana apenas. Neste contexto, o presente trabalho procura efetuar previsões de curto prazo, 10 dias, e de longo prazo, 3 meses, sempre com uma frequência horária o que o torna num trabalho mais abrangente.

Noutros casos, foram evidenciados estudos onde se apresentavam objetivos de previsões de consumos com pouco detalhe, como diários, [13], ou mensais, [11], para além de, nestes dois casos serem utilizados dados de fatores climáticos, como a temperatura do ar, adicionados aos dados de caudal distribuído como informação extra de treino. No presente trabalho a frequência de previsão foi de 1 hora, o que torna as previsões mais detalhadas, mais precisas, mas também mais difíceis. Não foram incluídos dados de fatores climáticos porque não se dispunha dessa informação e, portanto, o objetivo era o de prever apenas com base no histórico de dados de caudal.

2.4. Métricas de Avaliação dos Modelos

No que respeita às métricas de avaliação dos modelos, a raiz do erro quadrático médio (RMSE) tem sido utilizado como uma métrica estatística standard para avaliar a performance de um modelo, tal como o erro médio absoluto (MAE) tem sido bastante utilizado [22]. No estudo referido anteriormente são evidenciadas as características destas duas métricas referindo alguma supremacia do RMSE sobre a MAE, no entanto os autores referem que a utilização conjunta das duas métricas, bem como de outras, é mais apropriado do que apenas uma isolada, pois os seus resultados acabam por se complementar o que permite obter uma avaliação mais justa.

Apesar de métricas como MAPE, MAE, MSE e RMSE serem frequentemente utilizadas em estudos de *Machine Learning*, está demonstrado que não são suficientes para verificar a qualidade dos métodos de regressão. Esta lacuna pode apenas ser colmatada pelos coeficientes de determinação, R^2, e pelo SMAPE. O coeficiente de determinação pode ter valores negativos, se a regressão for pobre, e valores entre 0 e 1 se a regressão for boa. Os valores positivos do R^2 podem ser considerados como a percentagem de precisão obtida pela regressão [23].

Assim, para avaliação da eficácia dos modelos, foram escolhidas as métricas MAE, que fornece uma medida média absoluta do erro obtido de forma direta e não é sensível a *outliers*, a RMSE, que fornece uma medida da magnitude dos erros obtidos, e por último, aquele que foi considerado o mais importante, o coeficiente de determinação, ou R^2, que permite medir a precisão das previsões dos modelos, indicando a percentagem de dados reais que cada modelo consegue explicar.

Para o cálculo das métricas foram utilizados as classes *mean_absolute_error*, *mean_squared_error*, da qual se calculou a raiz quadrada, e *r2_score*, todos da biblioteca *sklearn.metrics*, e foi criada uma função que recebia como parâmetros objetos do tipo *panda series*, calculava as métricas e devolvia uma tabela formatada com os valores pretendidos.

CAPÍTULO III – PRÉ-TRATAMENTO E ANÁLISE DE DADOS

Neste capítulo apresenta-se a metodologia utilizada para efetuar o pré-tratamento dos dados existentes, tal como estes foram obtidos, bem como os vários testes de análise realizados aos dados após o seu tratamento. Apresentam-se ainda os resultados obtidos, tanto no pré-tratamento como na análise, aos dados de cada localidade. As decisões tomadas nesta fase foram fundamentais para a parametrização dos modelos realizada no capítulo seguinte.

3.1. Metodologia

Os dados relativos a cada setor de abastecimento de água encontravam-se guardados em 15 ficheiros que continham os dados de caudal de 15 meses, entre outubro de 2021 e dezembro de 2022. Os ficheiros do setor de Alcáçova, Degolados e Janeiro de Cima encontravam-se no formato *csv* e o de Aldeia de Joanes no formato *txt*.

Entre os meses de outubro de 2021 e setembro de 2022 encontravam-se os dados de um ano completo e representavam os dados de treino, já os meses de outubro a dezembro de 2022 representavam os dados de teste.

Com exceção do setor de Alcáçova onde a frequência de registo de caudal era horária, todos os restantes tinham uma frequência de registo de 5 minutos. Assim, era esperado que em Alcáçova se tivessem (365 dias + 92 dias) x 24 registos = 457 dias x 24 registos = 10.968 dados de caudal, e nos restantes 457 dias x 24 x 12 = 131.616 dados de caudal. Estes números foram importantes para validar a quantidade de dados que se obtiveram da leitura dos ficheiros pois existia a possibilidade de existência de anomalias na frequência dos registos que podia fazer variar estes valores e que teriam de ser resolvidas.

3.1.1. Leitura de Ficheiros

Para cada localidade em estudo foram lidos os ficheiros e concatenados para um *Dataframe* da biblioteca *pandas* com todos os atributos disponíveis incluindo a *Data/Hora.*

Figura 1 – Exemplo de Dataframe do setor de Alcáçova

```
df_total.head()
```

	Data/Hora	Evento	Caudal 0	Caudal 1	SAIDA	Caudal 3	ENT.	Caudal 5	Caudal 6	Caudal 7
0	01/10/21 00:00	Tempo	NaN	NaN	5	NaN	10,278	NaN	NaN	NaN
1	01/10/21 01:00	Tempo	NaN	NaN	5,278	NaN	0	NaN	NaN	NaN
2	01/10/21 02:00	Tempo	NaN	NaN	4,167	NaN	0	NaN	NaN	NaN
3	01/10/21 03:00	Tempo	NaN	NaN	3,333	NaN	0	NaN	NaN	NaN
4	01/10/21 04:00	Tempo	NaN	NaN	2,5	NaN	5,833	NaN	NaN	NaN

Fonte: Elaboração Própria

Da conjugação de todos os ficheiros com todos os dados de treino e de teste obteve-se um *Dataframe*, como se encontra exemplificado na Figura 1, para cada localidade onde se verificou o número total de dados existente por forma a fazer uma verificação prévia da existência de dados em falta.

3.1.2. Eliminação de Dados Irrelevantes para o Estudo

Dependendo da configuração do *Datalogger* que regista os dados de cada localidade, os ficheiros podiam ter mais ou menos atributos, pelo que os *Dataframes* criados podiam também ter mais atributos do que os necessários. Muitas vezes estes *Dataloggers* registam dados de pressão, o estado de entradas digitais, ou até dados de caudal não relacionados com a água distribuída, como no caso de Alcáçova onde existia o registo do caudal de entrada no reservatório. Assim, nesta fase procedeu-se à eliminação de todos os atributos considerados irrelevantes, existentes nos *Dataframes*, deixando ficar apenas a *Data/Hora* e o atributo que representa o caudal de água de entrada fornecida a cada setor.

3.1.3. Formatação de Dados

Depois de ter os *Dataframes* apenas com os atributos necessários, foi necessário efetuar uma conversão dos dados existentes, uma vez que estes eram ambos do tipo *object* e o que era pretendido era que os dados de *Data/Hora* fossem do tipo *datetime* e os dados de caudal fossem do tipo *float*. No caso dos dados de caudal, foi necessária uma atenção especial já que, durante a importação, estes ficaram com virgulas a

separar as casas decimais que tiveram de ser substituídas por pontos antes da conversão.

3.1.4. Verificação da Equidistância das Séries Temporais e Resolução de Anomalias

Como foi dito anteriormente, conhecer a quantidade de dados importados para cada série temporal foi importante para despistar a existência de algum erro grosseiro no número de dados que, no caso da série de Alcáçova terá de ser de 10.968 e nas restantes de 131.616. Acontece que este indicador não é suficiente por si só, pois poderíamos ter dados em falta em determinados momentos e dados em excesso noutros momentos. Entenda-se dados em excesso como dados de caudal repetidos para o mesmo momento. Este balanço entre número de dados em falta e número de dados em excesso poderia estar equilibrado ao ponto de obtermos exatamente a quantidade esperada, no entanto a série não estaria correta, pelo que foi necessário adotar uma verificação adicional.

Esta verificação adicional consistiu em verificar a distância temporal entre cada elemento da série obtendo a frequência de amostragem para o atributo *Data/Hora*. Para tal foi utilizado o método *infer_freq()* da biblioteca *pandas* que nos deveria dar como resultado *H* para a série temporal de Alcáçova e *5T* para as restantes.

Como o resultado da aplicação deste método foi *None* para todas as séries optou-se por adicionar um atributo *equidistance* do tipo booleano a cada *Dataframe* onde, para cada dado era validada a sua distância temporal com o dado anterior e preenchido o valor *False* no caso de não ser a distância esperada e *True* caso contrário, Figura 2.

Figura 2 – Verificação de Distâncias entre Elementos de uma Série

```
# Definição de uma função que verifica distâncias entre os registos e atualiza os valores de uma coluna equidistance
def verify_distance():
    diff = pd.Series(df_total['Data/Hora']).diff()
    equidistance = (diff == pd.Timedelta('5 minute'))

    df_total['equidistance'] = equidistance
    return df_total.loc[df_total['equidistance'] == False]
```

```
verify_distance()
```

	Data/Hora	CDEP	equidistance
0	2021-10-01 00:00:00	0.467	False
43856	2022-03-02 06:41:00	0.533	False
43857	2022-03-02 06:45:00	1.133	False
51000	2022-03-27 03:00:00	0.433	False
62687	2022-05-06 17:58:00	0.233	False
62688	2022-05-06 18:00:00	1.733	False
67579	2022-05-23 17:40:00	1.333	False
113375	2022-10-29 18:01:00	0.167	False
113376	2022-10-29 18:05:00	1.800	False
129628	2022-12-25 04:55:00	0.367	False

Fonte: Elaboração Própria

Com o atributo *equidistance* foi possível bloquear todas as linhas de cada *Dataframe* onde o seu valor era *False* e obter uma listagem das anomalias a resolver.

Para a resolução destas anomalias foram verificados os dados de fronteira de cada dado considerado anómalo por forma a saber se existiam dados em falta ou dados a mais. No caso de se tratar de dados a mais estes foram simplesmente apagados, mas sempre que se encontraram dados em falta os mesmos foram preenchidos com dados iguais aos de períodos semelhantes, por exemplo, os dados em falta de um domingo foram preenchidos pelos dados do mesmo período do domingo mais próximo, e os dados em falta de um dia da semana foram preenchidos pelos dados do mesmo período do dia da semana mais próximo e assim sucessivamente.

Existem situações em que o *Datalogger* é reiniciado quer por falta de energia durante operações de manutenção quer por atualização do valor numérico do volume registado pelo caudalímetro, que emite os impulsos e que depois são convertidos em caudal. Isto pode fazer com que ocorram registos fora do tempo esperado, por exemplo, o registo esperado às 18h00 era omisso, mas existia um registo às 18h01. Este tipo de anomalia foi corrigida alterando a hora para o valor mais próximo da mesma, ou seja, no exemplo anterior o registo das 18h01 foi acertado para as 18h00.

Outro caso que surgiu algumas vezes foi o de faltar apenas um registo entre dois registos conhecidos. Neste caso foi realizada uma média dos dois caudais conhecidos, criado o registo intermédio em falta e colocado esse valor médio de caudal.

Outra anomalia detetada nos dados esteve relacionada com a mudança da hora no mês de março. Se por um lado, no mês de outubro quando os relógios atrasaram uma hora os *Dataloggers* conseguiram reescrever os dados, quando os relógios adiantaram uma hora no mês de março foi deixado um vazio de dados durante essa hora. O comportamento dos equipamentos foi diferente, num caso duplicou registos entre as 2h00 e as 3h00 e nos restantes simplesmente os registos não foram feitos, assim no caso dos registos duplicados foi feita a alteração da hora do registo e no caso dos registos em falta foram utilizados os dados dos registos do mesmo período do dia mais próximo.

Finalmente, após ter resolvido todos os problemas de equidistância, foi realizada uma análise aos zeros. Em alguns setores é normal a existência de um valor nulo de caudal em determinadas horas do dia e noutros isso não acontece. A existência de valores nulos de caudal de forma pontual evidencia uma intervenção na rede de abastecimento com fecho de válvulas, o que desvirtua a realidade, pelo que, sempre que foram encontrados valores nulos de caudal de forma pontual e sem repetição nos dias adjacentes, estes valores foram substituídos pelos valores desses dias.

3.1.5. Reamostragem de Dados

A existência de registos de dados em intervalos de cinco minutos ou em intervalos de uma hora depende única e exclusivamente da programação implementada nos *Dataloggers*. No primeiro caso o dispositivo conta o número de impulsos que ocorrem durante cinco minutos, conhece o volume que representa cada impulso e no final do período divide o volume pelo tempo obtendo o caudal. No segundo caso acontece exatamente o mesmo, mas num período de uma hora.

De forma a garantir uma homogeneidade das séries a estudar e evitar problemas de capacidade de processamento, no caso das séries com mais de 130.000 registos, foi feita uma reamostragem das séries de Aldeia de Joanes, Degolados e Janeiro de Cima para apenas um registo por hora. Esta conversão foi feita da mesma forma que seria

realizada pelo *Datalogger* respetivo, caso estivesse configurado dessa forma, ou seja, foi feito pela média dos caudais registados durante cada hora.

3.1.6. Exportação de Dados

Após todas as etapas de pré-tratamento aplicadas aos dados importados, os mesmos foram exportados para um diretório de dados pré-processados por meio de um ficheiro *csv* contendo todos os dados de cada localidade.

3.1.7. Estacionaridade

A Estacionaridade é um pressuposto importante no estudo de séries temporais porque diversos modelos de séries temporais têm como pressuposto a estacionaridade, ou seja, para que esses modelos consigam fazer previsões sobre determinada série, é necessário que esta se comporte de forma estacionária.

Séries estacionárias são séries que se desenvolvem aleatoriamente no tempo mantendo uma média e variância constantes, já as não estacionárias são aquelas que mudam o seu comportamento evidenciando inclinações ou mudanças de nível.

3.1.7.1. Verificação da Estacionaridade Graficamente

Para a verificação gráfica da Estacionaridade de cada série foi feita uma impressão em écran do gráfico da série. Como se tratava de muitos dados, ao exibir o gráfico acabava por aparecer apenas uma faixa sem possibilidade de perceber ou de visualizar detalhes. Assim, os gráficos foram exibidos utilizando a biblioteca *plotty* que permitiu implementar ferramentas de *zoom* que se tornaram indispensáveis.

Pela visualização dos gráficos foi possível obter uma perceção do comportamento da média e da variância de cada série bem como da eventual existência de comportamentos erráticos que possam trazer dificuldades futuras para os modelos de previsão.

3.1.7.2. Testes Numéricos

Para além da verificação gráfica, que acabou por permitir ter um registo visual da série e que se mostrou bastante importante, foi necessário fazer uma verificação da Estacionaridade recorrendo a testes numéricos porque se traduzem em métodos mais objetivos. Assim, foram utilizados como testes numéricos os testes *KPSS* (*Kwiatkowski-Phillips-Schmidt-Shin*), *ADF* (*Augmented Dickey-Fuller*) e *PP* (*Philips-Perron*). Todos os testes tiveram como critério de rejeição da hipótese nula, os casos em que o valor-p ou o valor da estatística do teste obtidos fosse menor do que o nível de significância escolhido, que neste caso foi de 0.05.

3.1.7.2.1. Teste *KPSS*

O teste *KPSS* é um teste de raíz unitária que verifica a estacionaridade de uma determinada série temporal em torno de uma tendência determinística. A existência de uma raiz unitária indica que a série não é estacionária e a sua ausência indica que a série é estacionária.

Hipóteses do teste *KPSS*:

- Hipótese nula: A série é estacionária:
    - O valor da estatística do teste deve ser menor do que o valor crítico;
- Hipótese alternativa: A série não é estacionária.

Foi utilizado o teste implementado na biblioteca *statsmodels*.

3.1.7.2.2. Teste *Augmented Dickey-Fuller*

O teste de *Augmented Dickey-Fuller* (*ADF*) é um teste estatístico utilizado para verificar se uma série temporal é estacionária ou não. Trata-se de uma extensão do teste de raiz unitária, que verifica se uma série temporal tem uma raiz unitária ou não. O teste *ADF* utiliza um modelo autorregressivo para estimar a presença ou ausência de uma raiz unitária na série.

Hipóteses do teste *ADF*:

- Hipótese nula: A série não é estacionária:
- Hipótese alternativa: A série é estacionária.

Foi utilizado o teste implementado na biblioteca *arch*.

3.1.7.2.3. Teste *Philips-Perron*

O teste de Philips-Perron é um teste de raiz unitária que utiliza um estimador de kernel robusto para reduzir o efeito da autocorrelação.

Hipóteses do teste PP:

- Hipótese nula: A série não é estacionária, contém uma raiz unitária:
- Hipótese alternativa: A série é estacionária.

Foi utilizado o teste implementado na biblioteca *statsmodels*.

3.1.8. Autocorrelação

A verificação da existência de autocorrelação numa série temporal é importante porque verifica a existência de dependência dos dados de uma série entre si, ou seja, verifica se ao longo do tempo, entre determinadas *lags*, os dados são dependentes entre si, o que pode ser um problema para alguns modelos de séries temporais.

Lag é um termo que se refere a um intervalo temporal entre observações e é utilizado para verificar a existência de autocorrelação de valores de variáveis ao longo do tempo [24].

A verificação da existência de autocorrelação foi realizada para dados adjacentes e para dados não adjacentes, sendo que, neste último caso, se chama de autocorrelação parcial.

3.1.8.1. Diagrama ACF

Para verificação da existência de autocorrelação entre os dados adjacentes das séries temporais foi utilizado um diagrama ACF (*Autocorrelation Function*) obtido com recurso à biblioteca *statsmodels* e ao método *plot_acf*.

3.1.8.2. Diagrama PACF

Para verificação da existência de autocorrelação entre dados não adjacentes, portanto, autocorrelação parcial, foi utilizado um diagrama PACF (*Parcial Autocorrelation Function*) obtido com recurso à biblioteca *statsmodels* e ao método *plot_pacf*.

3.1.9. Decomposição

A realização da decomposição das séries temporais foi realizada com o objetivo de poder analisar propriedades das mesmas, como a Tendência, a Sazonalidade e os Resíduos.

Na decomposição de séries temporais existem duas abordagens diferentes para descrever a relação entre os componentes de tendência, sazonalidade e resíduos. A primeira abordagem assume que a série temporal é resultado da soma dos três componentes e por isso se chama de modelo aditivo, já o modelo multiplicativo considera que a série temporal é resultado do seu produto. No primeiro modelo a tendência é considerada como uma adição constante e a sazonalidade representada por variações cíclicas e constantes ao longo do tempo, no segundo a tendência é considerada como um fator multiplicativo e a sazonalidade representa variações cíclicas proporcionais e periódicas ao longo do tempo, mais comuns na área financeira [25].

Para a decomposição das séries foi utilizada a biblioteca *statsmodels* e o método *seasonal_decompose*.

Como parametrização do método foi utilizada uma frequência de 24, uma vez que temos um registo por hora e é espectável a existência de uma sazonalidade a cada 24 horas, e um modelo aditivo. Foram feitas experiências com o modelo multiplicativo, mas este não acrescentou nenhuma informação aos resultados.

3.1.9.1. Análise de Resíduos

Tendo as séries decompostas foi considerado importante realizar uma análise dos resíduos do ponto de vista da autocorreção. Assim foram utilizados os diagramas ACF

e PACF, mencionados anteriormente, apenas para os resíduos de forma a permitir concluir sobre a existência ou não de autocorreção entre os dados de resíduos adjacentes e também para os não adjacentes.

A metodologia utilizada para obtenção dos diagramas ACF e PACF para os resíduos, foi a mesma da utilizada no ponto 3.1.8..

3.1.10. Análise de Normalidade

A Normalidade de uma série temporal é uma propriedade importante pois muitos modelos de previsão, bem como processos de treino, dependem do princípio de que os dados se encontram distribuídos de acordo com uma distribuição Normal [26].

Dada a importância desta característica, foi verificado se a distribuição dos dados das séries temporais em estudo era próxima de uma distribuição Normal.

Para verificação da Normalidade dos dados das séries temporais foi utilizada a biblioteca *scipy.stats*.

3.1.10.1. Verificação de Normalidade Graficamente

Para verificar visualmente através de um gráfico a normalidade da distribuição dos dados das séries temporais foi utilizado o método *probplot* da biblioteca *scipy.stats* que imprime um gráfico de uma reta que representa a localização da normalidade sobreposta com a realidade dos dados de cada série.

3.1.10.2. Testes Numéricos

Para além de uma simples verificação gráfica, considerou-se relevante implementar alguns testes numéricos para verificar a normalidade das séries. Os critérios de rejeição da hipótese nula dos testes bem como o nível de significância escolhido são os indicados no ponto 3.1.7.2. com exceção do teste de *Anderson-Darling* cujo critério de rejeição é indicado na descrição do teste.

3.1.10.2.1. Teste de *Shapiro-Wilk*

O teste de *Shapiro-Wilk* é um teste estatístico utilizado para verificar e um determinado conjunto de dados segue uma distribuição normal. As hipóteses do teste são:

- Hipótese nula: Os dados seguem uma distribuição normal;
- Hipótese alternativa: Os dados não seguem uma distribuição normal.

Este teste foi implementado recorrendo ao método *shapiro* da biblioteca *scipy.stats*.

3.1.10.2.2. Teste de *Anderson-Darling*

Trata-se de outro teste estatístico para fazer a verificação da normalidade de um determinado conjunto de dados. As hipóteses deste teste são:

- Hipótese nula: Os dados seguem uma distribuição normal;
    - Esta hipótese deve ser rejeitada no caso de a estatística do teste ser superior ao valor crítico para o nível de significância escolhido.
- Hipótese alternativa: Os dados não seguem uma distribuição normal.

Este teste foi também implementado recorrendo ao método *anderson* da biblioteca *scipy.stats*

3.1.10.3. Transformações

Infelizmente muitas vezes os dados em estudo não seguem distribuições normais o que impossibilita o correto funcionamento dos modelos tradicionais. Nestes casos é possível aplicar transformações aos dados que melhoram a sua normalidade possibilitando uma análise estatística com maior validez [26].

Nos casos em que a normalidade não se verificou, foram aplicadas transformações aos dados e verificada a sua normalidade novamente. As transformações realizadas foram a logarítmica e por raiz cúbica.

3.1.10.3. Análise Utilizando a Biblioteca *Seaborn*

A biblioteca *Seaborn* permite a criação de gráficos com um potencial de facilitação de análise. Neste caso foi utilizado o método *histplot* para imprimir a distribuição dos dados em forma de histograma e assim visualizar facilmente se essa distribuição se assemelhava a uma distribuição normal.

3.1.11. Diferenciação

A diferenciação dos dados é uma metodologia não muito diferente de uma transformação pois traduz-se na obtenção de um novo conjunto de dados obtidos pela diferença entre cada um dos dados originais. Tem como característica o facto de se perder um dado do conjunto original, ou seja, se tivermos um conjunto de dados com n elementos, ao efetuarmos a diferenciação dos mesmos, iremos obter um conjunto com n-1 elementos.

A diferenciação dos dados tem a faculdade de melhorar a estacionaridade dos dados, pelo que foi realizada para todas as séries permitindo ajudar na parametrização dos modelos mais tarde.

Para obtenção dos dados diferenciados foi utilizada o método *diff* da biblioteca *numpy*.

3.1.11.1. Estacionaridade após Diferenciação

Após a aplicação da diferenciação aos dados das séries em estudo, foi verificada a estacionaridade do novo conjunto de dados com o objetivo de perceber se a diferenciação tinha melhorado a mesma.

A estacionaridade foi verificada utilizando a mesma metodologia já descrita no ponto 3.1.7..

3.2. Resultados do Pré-Tratamento e Análise dos Dados

3.2.1. Setor de Alcáçova

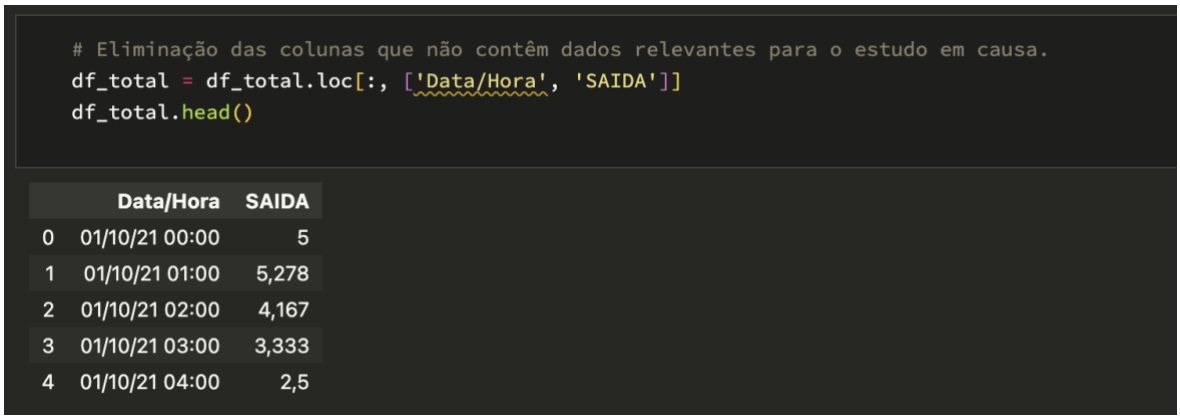
3.2.1.1. Resultados do Pré-Tratamento

Da leitura dos 15 ficheiros relativos ao setor de Alcáçova foi obtido um *Dataframe* com 10970 registos e 10 atributos.

Tendo em conta que o *Datalogger* deste setor efetua um registo de caudal a cada hora, seria de esperar que os dados obtidos fossem de 10968. Esta facto, já indicava que existiam problemas com os registos dos dados, pois existiam, pelo menos, dois registos a mais do que o esperado.

Foram observados todos os atributos existentes e deixados ficar apenas o atributo *Data/Hora* que tinha a informação da data e hora de cada registo e *SAIDA* que tinha o valor do caudal em litros por segundo registado em cada hora, Figura 3. Entre os atributos existentes, existia um atributo que também continha um caudal, mas, neste caso era o caudal de entrada no reservatório em causa e não tinha nenhum interesse para este estudo, pelo que foi descartado.

Figura 3 – Eliminação de Atributos não Relevantes no Setor de Alcáçova



```
# Eliminação das colunas que não contêm dados relevantes para o estudo em causa.
df_total = df_total.loc[:, ['Data/Hora', 'SAIDA']]
df_total.head()
```

	Data/Hora	SAIDA
0	01/10/21 00:00	5
1	01/10/21 01:00	5,278
2	01/10/21 02:00	4,167
3	01/10/21 03:00	3,333
4	01/10/21 04:00	2,5

Fonte: Elaboração Própria

No que respeita à formatação dos dados foi seguida a metodologia referida no ponto 3.1.3., Figura 4.

Figura 4 – Formatação dos dados do Dataframe do Setor de Alcáçova

```
df_total['Data/Hora'] = pd.to_datetime(df_total['Data/Hora'], format='%d/%m/%y %H:%M')
df_total['SAIDA'] = pd.to_numeric(df_total['SAIDA'].str.replace(',', '.'))
df_total.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10970 entries, 0 to 10969
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Data/Hora  10970 non-null  datetime64[ns]
 1   SAIDA      10970 non-null  float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 171.5 KB
```

Fonte: Elaboração Própria

Na verificação dos eventuais problemas de equidistância entre os dados registados foi verificado que a série não era equidistante. Ao adicionar o atributo *equidistance* e ao bloquear a impressão do *Dataframe* sempre que o seu valor fosse *False*, verificou-se a existência de sete valores não equidistantes e com necessidade de serem analisados.

Foi encontrado um valor em falta na noite da mudança da hora de inverno para verão e três valores a mais, fruto de um possível reinício do *Datalogger*. Foi implementada a metodologia descrita no ponto 3.1.4. para ambos os casos e finalmente foi obtido um *Dataframe* com 10968 elementos e com uma frequência horária entre os dados.

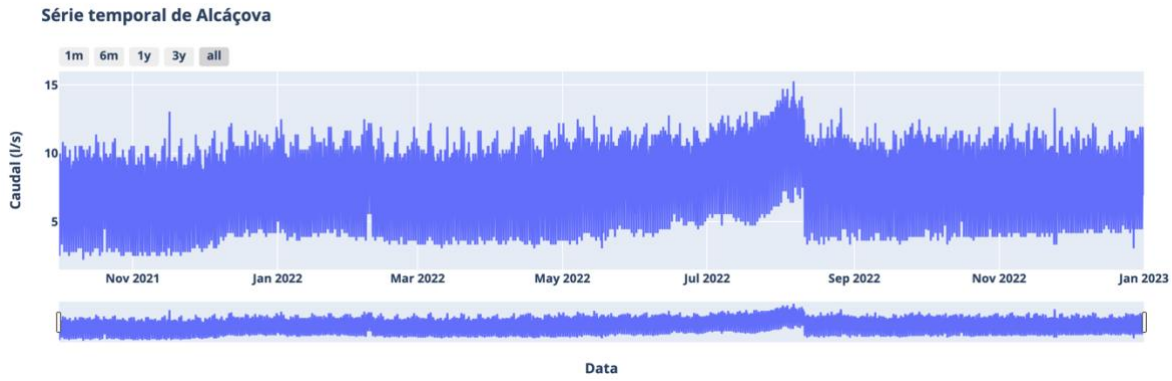
Foi ainda realizada uma verificação da existência de zeros para o valor de caudal em todo o *Dataframe* que poderiam ser resultado de um fecho de válvulas no âmbito de uma operação de manutenção, no entanto não foram encontrados quaisquer registos de zero para este atributo.

Finalmente, os dados foram exportados para o diretório de dados pré-processados para um ficheiro com o nome *Alcacova.csv*.

3.2.1.2. Análise de Dados

No que respeita à análise da estacionaridade da série temporal de Alcáçova, por observação gráfica verificou-se que nem a média nem a variância eram constantes ao longo do tempo, conforme pode ser observado na Figura 5.

Figura 5 – Distribuição da série Temporal de Alcáçova ao longo do tempo

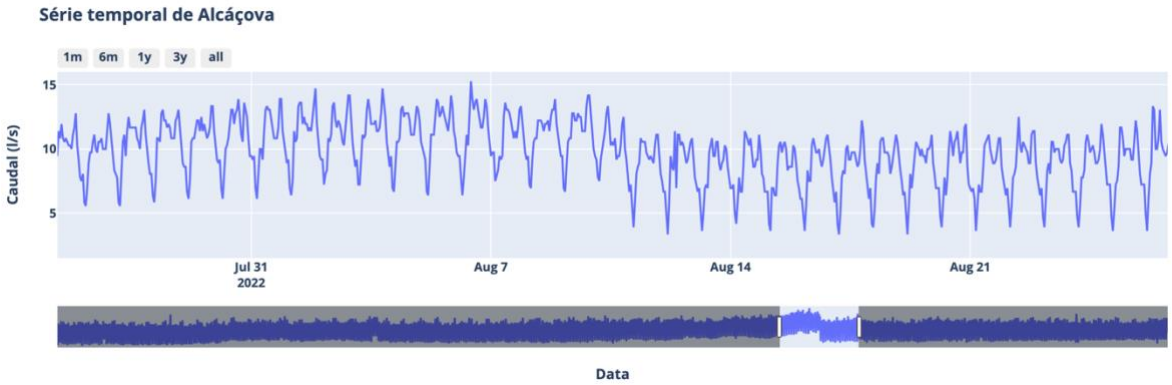


Fonte: Elaboração Própria

Observou-se um aumento da média mais pronunciado e com um pico em agosto, Figura 6, e que depois caía de forma pronunciada iniciando novamente um novo ciclo de subida muito ligeira.

Este tipo de variação poderá ter-se devido à ocorrência de uma rotura na rede de distribuição não detetável à superfície e que por isso não terá sido reparada inicialmente. Ao longo do tempo terá aumentando o caudal de fuga até chegar ao ponto em que passa a ser visível à superfície ou terão sido utilizadas ferramentas auxiliares de deteção, como Geofónes, e terá sido reparada.

Figura 6 – Ampliação na Zona de Pico da Série Temporal de Alcáçova



Fonte: Elaboração Própria

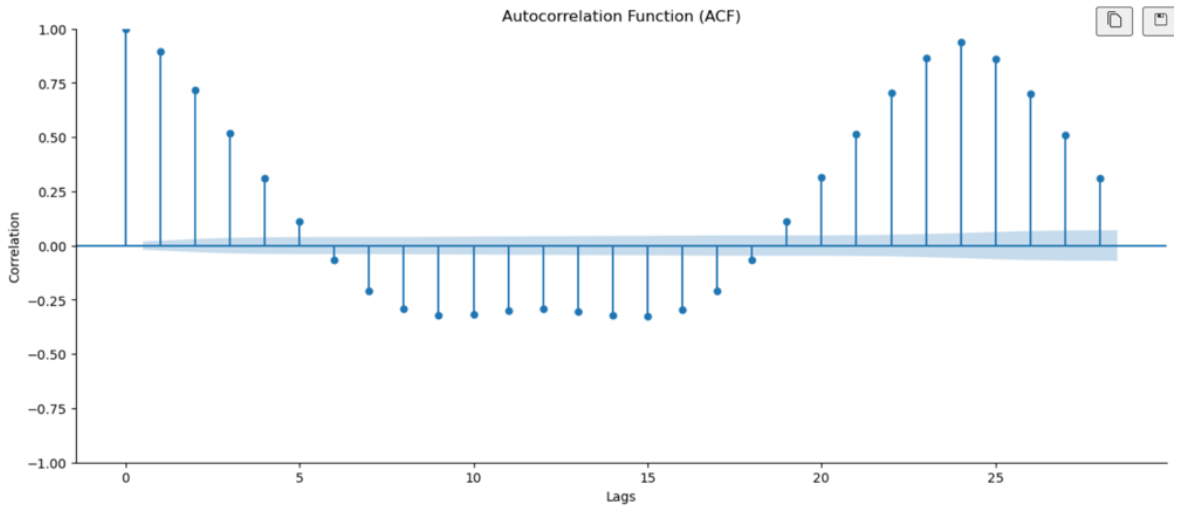
No que respeita aos testes numéricos para verificação da estacionaridade obteve-se como resultados que o teste *KPSS* indicou que a série não era estacionária tendo o teste *Augmented Dickey-Fuller* e *Philips-Perron* indicado que a série era estacionária.

Tendo em conta que o teste *KPSS* apresenta alguns problemas para amostras muito grandes, que é o caso, e que os restantes testes indicam que a série é estacionária, tendo em conta o observado graficamente, conclui-se que a série apresentava uma fraca estacionaridade.

A análise da autocorrelação da série temporal de Alcáçova através do diagrama de autocorrelação evidenciou uma forte dependência entre as *lags* adjacentes,

Figura 7.

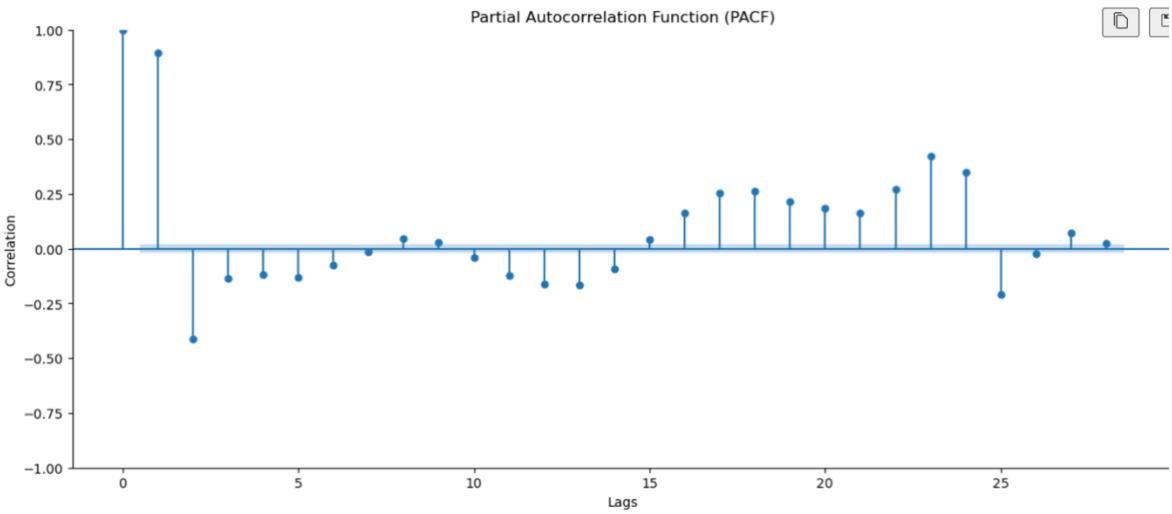
Figura 7 – Diagrama de Autocorrelação da Série de Alcáçova entre Lags Adjacentes



Fonte: Elaboração Própria

Da mesma forma, também o diagrama de autocorrelação parcial mostrou uma grande dependência entre *lags* não adjacentes, Figura 8.

Figura 8 – Diagrama de Autocorrelação da Série de Alcáçova entre Lags não Adjacentes



Fonte: Elaboração Própria

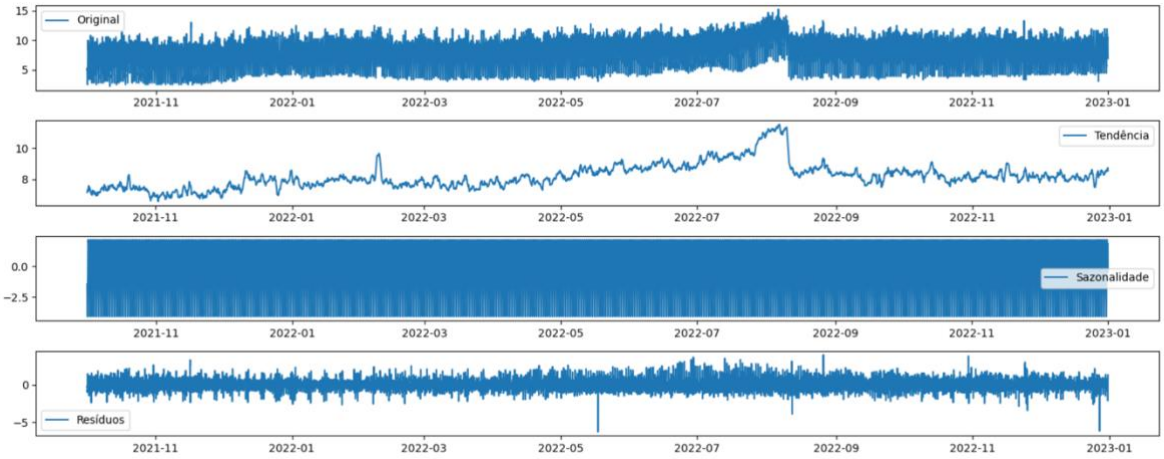
Passando à decomposição da série, foi utilizada uma frequência de 24, uma vez que tínhamos um registo por hora e era expectável a existência de uma sazonalidade a cada 24 horas.

Dos gráficos da série decomposta, evidenciados na Figura 9, notou-se uma tendência crescente até agosto de 2022, data a partir da qual existia uma diminuição da mesma e depois, a longo prazo, aparentava novamente tornar-se mais ou menos constante.

Em relação à sazonalidade apenas se pôde concluir que existe, mas por dificuldade de visualização do gráfico foi difícil concluir se se tratava apenas de uma sazonalidade diária ou se existiam outros períodos possíveis de identificar.

Já o gráfico dos resíduos, por seu lado, mostrou uma certa organização com valores aparentemente bem estruturados, o que levou a pensar que poderiam estar com problemas de autocorrelação.

Figura 9 – Decomposição da Série de Alcáçova

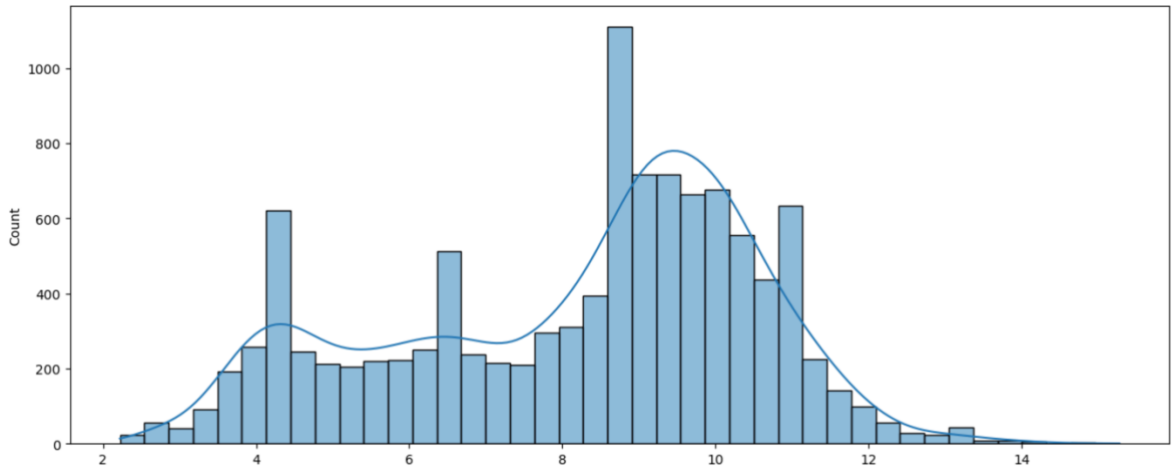


Fonte: Elaboração Própria

Analisando a autocorrelação e a autocorrelação parcial dos resíduos verificou-se a existência de uma forte dependência dos dados, tanto em *lags* adjacentes como em *lags* não adjacentes.

A análise da Normalidade da série, quer graficamente quer através dos testes numéricos, permitiu concluir que a mesma não segue uma distribuição Normal. De resto, a mesma afasta-se bastante daquilo que seria uma distribuição Normal e isso foi possível de comprovar facilmente através do histograma da distribuição da série temporal, Figura 10.

Figura 10 – Histograma da Distribuição da Série Temporal de Alcáçova

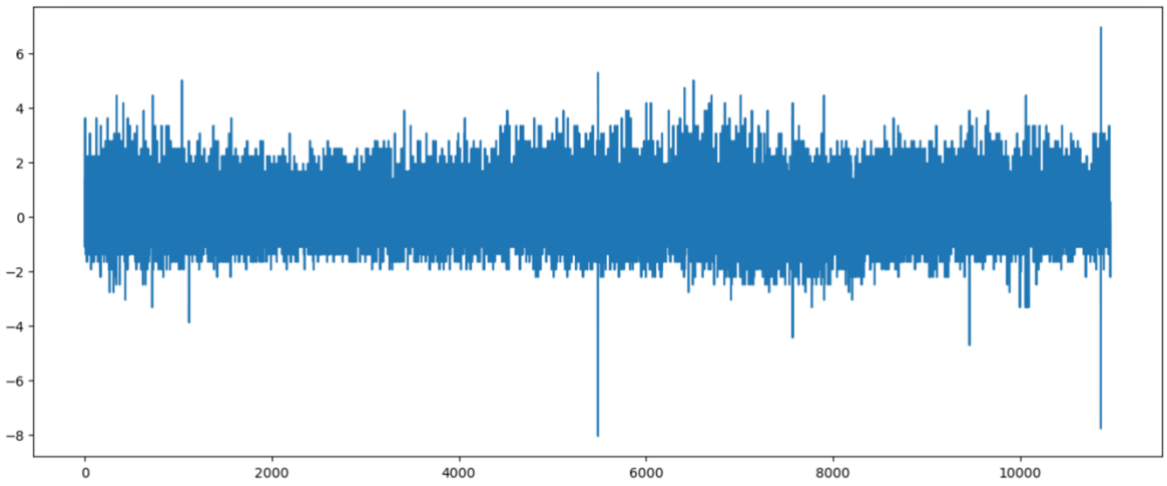


Fonte: Elaboração Própria

Foram feitas tentativas de melhorar a normalidade da série aplicando transformações logarítmicas e por raiz cubica, no entanto, analisando o resultado dos testes após a transformação verificou-se que, não só, nenhuma delas melhorava esta propriedade como inclusive a tornava ainda mas afastada daquilo que era pretendido.

Finalmente, foi aplicada uma diferenciação aos dados da série com o objetivo de melhorar a sua estacionaridade, já que se tinha concluído que a mesma tinha uma fraca estacionaridade. O objetivo era verificar se o teste KPSS passava a classificar a série como estacionária e se, visualmente, era possível obter um gráfico com a média e a variância mais constantes.

Figura 11 – Resultado da Diferenciação dos Dados da Série Temporal de Alcáçova



Fonte: Elaboração Própria

De acordo com a Figura 11, o resultado da diferenciação visualmente apresentou melhorias e o teste KPSS passou a classificar a série como estacionária, bem como os restantes. Por tudo isto concluiu-se que a diferenciação dos dados da série temporal de Alcáçova melhorava a estacionaridade da mesma.

3.2.2. Setor de Aldeia de Joanes

3.2.2.1. Resultados do Pré-Tratamento

Da leitura dos 15 ficheiros relativos ao setor de Aldeia de Joanes foi obtido um *Dataframe* com 131616 registos e 2 atributos.

O *Datalogger* deste setor efetua um registo de caudal a cada 5 minutos, logo, para os 15 meses em causa, o número de dados esperado é precisamente igual ao número de dados obtido, 131616. Em princípio, este facto indicava que não existiria nenhum problema com o número de dados, no entanto, foi necessário e prudente proceder ao teste de frequência a fim de nos certificarmos disso mesmo e também analisar a existência de zeros.

Em relação aos atributos, apenas existiam dois, *Date* e *ALDEIA_JOANES_CNT*. O primeiro continha a informação da data e hora de registo de cada valor de caudal e o segundo continha os dados do próprio caudal em litros por segundo registado a cada

cinco minutos, Figura 12. Assim, neste caso, não foi necessário proceder à eliminação de colunas com dados irrelevantes para o estudo.

Figura 12 – *Dataframe* do setor de Aldeia de Joanes

```
df_total.head()
```

	Date	ALDEIA_JOANES_CNT
0	01/10/2021 00:00:00	0.0
1	01/10/2021 00:05:00	0.0
2	01/10/2021 00:10:00	0.0
3	01/10/2021 00:15:00	0.0
4	01/10/2021 00:20:00	0.0

Fonte: Elaboração Própria

Em relação às necessidades de formatação dos dados, foi feita uma conversão para o tipo *datetime* em todos os dados do atributo *Date* utilizando a biblioteca *pandas* e o método *to_datetime*. Não foi necessária nenhuma conversão para os dados de caudal uma vez que os mesmos já eram do tipo *float*. No caso do atributo *Date*, foi ainda necessário um cuidado especial pois todos os valores tinham um espaço em branco no final que foi necessário retirar utilizando o método *strip* antes da conversão, Figura 13.

Figura 13 – Formatação dos dados do *Dataframe* do setor de Aldeia de Joanes

```
# As datas estão no tipo object e têm um e spaço no fim que deve ser retirado antes da conversão.
df_total['Date'] = df_total['Date'].str.strip()
df_total['Date'] = pd.to_datetime(df_total['Date'], format='%d/%m/%Y %H:%M:%S')
df_total.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 131616 entries, 0 to 131615
Data columns (total 2 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   Date               131616 non-null  datetime64[ns]
 1   ALDEIA_JOANES_CNT  131616 non-null  float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 2.0 MB
```

Fonte: Elaboração Própria

Na verificação dos eventuais problemas de equidistância entre os dados registados foi observado que a série não era equidistante. Ao adicionar o atributo *equidistance* e ao bloquear a impressão do *Dataframe* sempre que o seu valor fosse *False*, verificou-se a existência de 13 valores não equidistantes e com necessidade de serem analisados.

Foram encontrados valores em falta na noite da mudança da hora de inverno para verão, ou seja, faltavam todos os valores que deveriam ter sido registados entre a 1h00 e as 2h00, que neste caso seriam doze. Por outro lado, entre as 2h00 e as 3h00 existiam valores duplicados a cada registo, portanto, doze valores a mais. Foi implementada a metodologia descrita no ponto 3.1.4. para corrigir a anomalia e assim obter um *Dataframe* com 131616 elementos e com uma frequência de registo de cinco minutos entre todos eles.

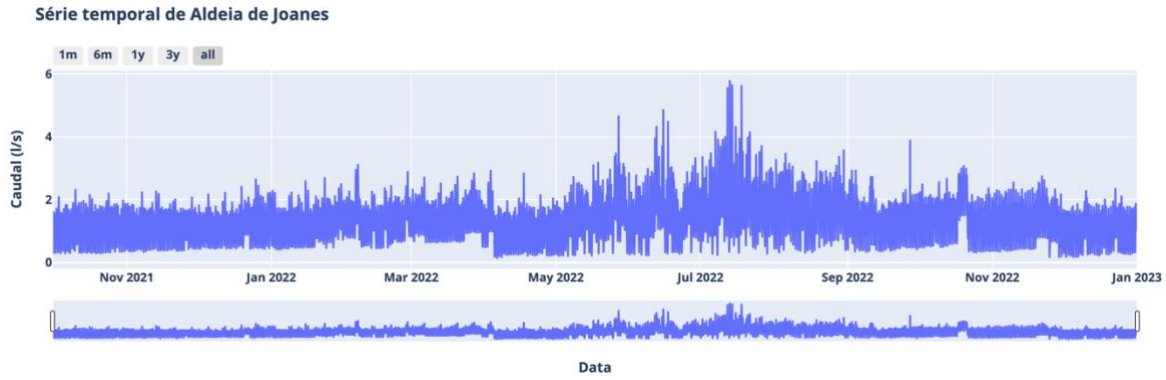
Na verificação da existência de zeros para os valores de caudal foram apenas encontrados cinco registos nessas circunstâncias logo no início do *Dataframe*. Considerou-se que esta ocorrência poderia ter resultado de uma operação de manutenção corretiva na rede de abastecimento onde a água teria estado fechada, já que não era um padrão que se repetisse em outros momentos com o mesmo período horário no restante *Dataframe*. Assim esses valores foram substituídos de acordo com a metodologia referida no ponto 3.1.4. no que respeita a zeros encontrados fora no padrão habitual.

Finalmente foi realizada a reamostragem dos dados, de acordo com a metodologia descrita no ponto 3.1.5. obtendo no final 10968 elementos, ou seja, um elemento por cada hora, e foi realizada a exportação dos dados para um ficheiro com o nome *AldeiaDeJoanes.csv*.

3.2.2.2. Análise de Dados

Analisando graficamente a estacionaridade da série temporal de Aldeia de Joanes, percebeu-se que nem a média nem a variância se mantinham constantes co longo do tempo, Figura 14.

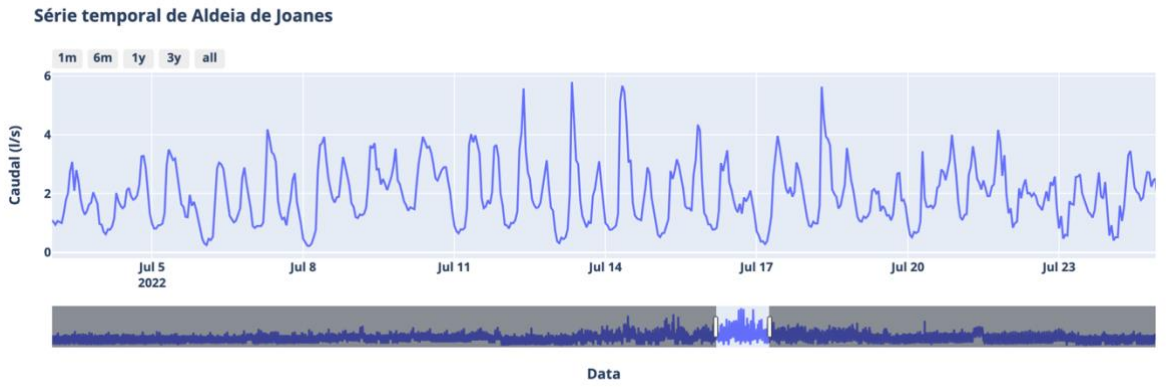
Figura 14 – Distribuição da série Temporal de Aldeia de Joanes ao longo do tempo



Fonte: Elaboração Própria

Nos primeiros meses, a média e a variância, apresentavam um comportamento estável, mas a partir de março começavam a apresentar muitas variações. Notou-se uma alteração substancial nos meses de verão onde o gráfico apresentava muitos picos de consumo de uma forma aparentemente aleatória, Figura 15. A partir de outubro voltava a ter um comportamento mais estável, ainda assim, com mais variações do que nos primeiros meses.

Figura 15 – Ampliação na Zona de Pico da Série Temporal de Aldeia de Joanes



Fonte: Elaboração Própria

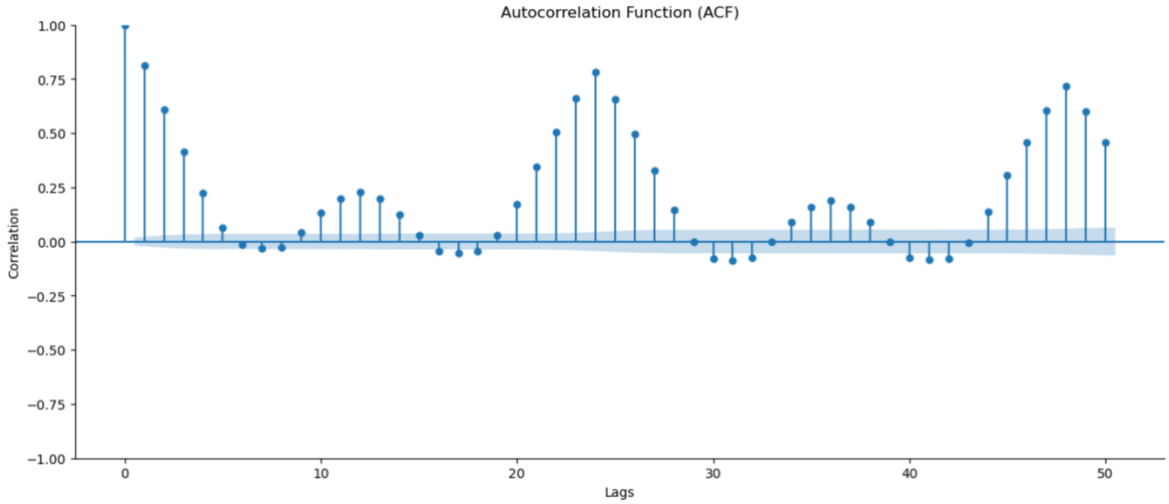
No que respeita aos testes numéricos para verificação da estacionaridade obteve-se como resultados que o teste *KPSS* indicou que a série não era estacionária tendo o teste *Augmented Dickey-Fuller* e *Philips-Perron* indicado que a série era estacionária.

Tendo em conta que o teste KPSS apresenta alguns problemas para amostras muito grandes, que é o caso, e que os restantes testes indicam que a série é estacionária, tendo

em conta o observado graficamente, concluiu-se que a série apresentava uma fraca estacionaridade.

A análise da autocorrelação da série temporal de Aldeia de Joanes, através do diagrama de autocorrelação, evidenciou uma forte dependência entre as *lags* adjacentes, Figura 16.

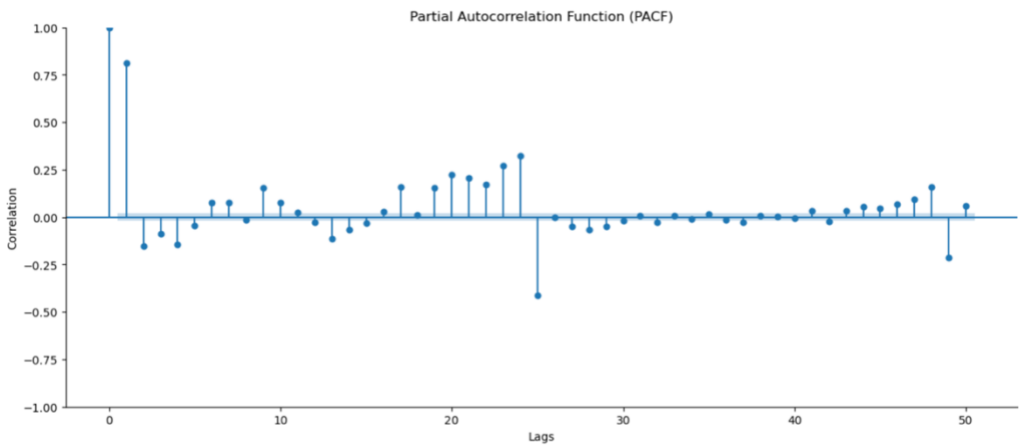
Figura 16 – Diagrama de Autocorrelação entre Lags Adjacentes



Fonte: Elaboração Própria

O gráfico de autocorrelação parcial também indicou autocorrelação entre as *lags*, neste caso não adjacentes, mas menor do que no caso das *lags* adjacentes, Figura 17.

Figura 17 – Diagrama de Autocorrelação entre Lags não Adjacentes



Fonte: Elaboração Própria

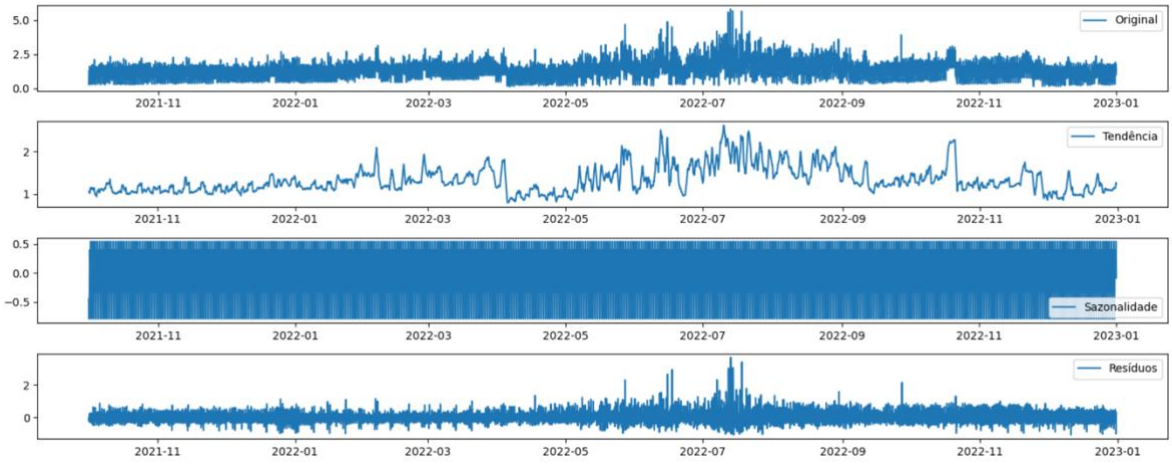
No que respeita à decomposição da série, foi utilizada uma frequência de 24, uma vez que temos um registo por hora e é expectável a existência de uma sazonalidade a cada 24 horas.

Dos gráficos da série decomposta, Figura 18, notou-se a existência de uma tendência estacionária nos primeiros meses que depois passou a decrescente no final do mês de abril, passando depois a crescente até ao final do mês de julho, sendo depois decrescente até outubro. Nos últimos meses a tendência comporta-se de uma forma bastante errática com a existência de alguns picos que não seriam de esperar nesta altura do ano.

Em relação à sazonalidade apenas se pôde concluir que existe, mas por dificuldade de visualização do gráfico foi difícil concluir se se tratava apenas de uma sazonalidade diária ou se existiam outros períodos possíveis de identificar.

Já o gráfico dos resíduos, por seu lado, mostrou uma certa organização com valores aparentemente bem estruturados, o que levou a pensar que poderiam estar com problemas de autocorrelação.

Figura 18 – Decomposição da Série da Aldeia de Joanes

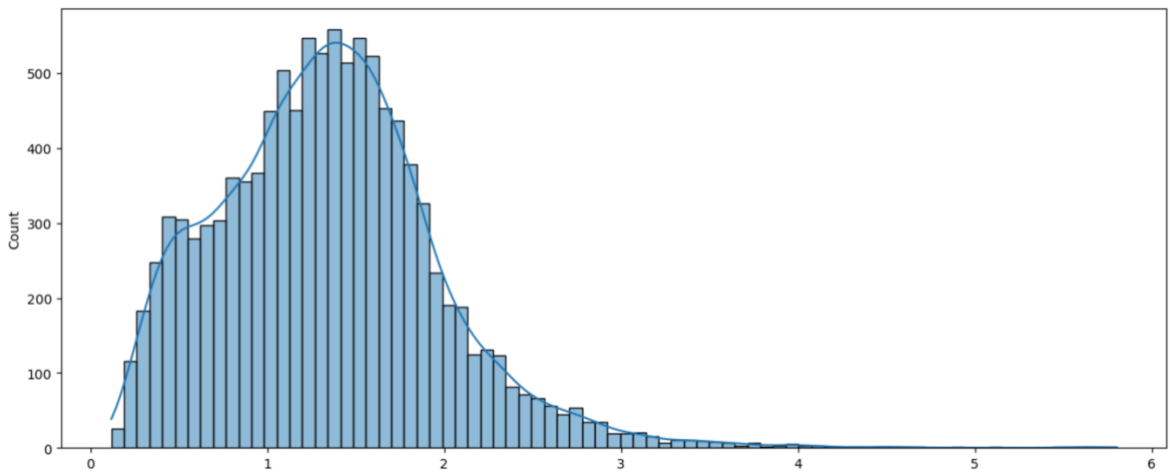


Fonte: Elaboração Própria

Analisando a autocorrelação e a autocorrelação parcial dos resíduos verificou-se a existência de uma forte dependência dos dados em *lags* adjacentes. Nas *lags* não adjacentes também se verificou existência de autocorrelação, mas menos pronunciada do que no caso anterior.

No que respeita à Análise da Normalidade da série, quer graficamente quer através dos testes numéricos, concluiu-se que a mesma não segue uma distribuição Normal. Esta conclusão pôde ser facilmente confirmada pelo histograma de distribuição da série temporal ilustrado na Figura 19.

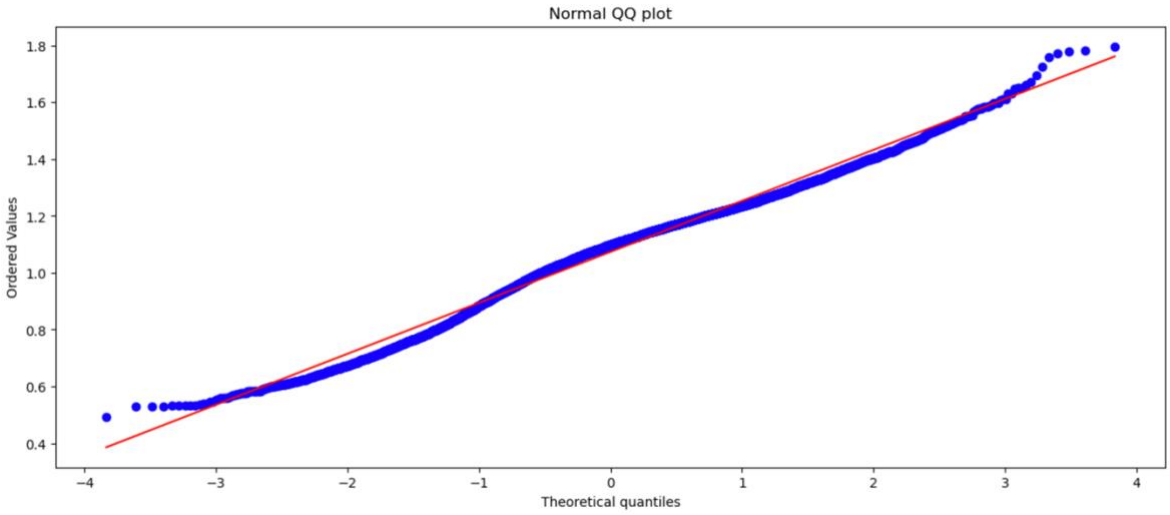
Figura 19 – Histograma de Distribuição da Série Temporal da Aldeia de Joanes



Fonte: Elaboração Própria

Nas transformações efetuadas aos dados da série com o objetivo de melhorar a sua normalidade, tanto a transformação logarítmica como a transformação por raiz cúbica, segundo os testes numéricos, não permitiram que a distribuição da série se transformasse numa distribuição Normal. Não obstante o gráfico *QQ plot* da Figura 20 mostrou uma boa aderência dos dados a uma distribuição Normal após a transformação por raiz cúbica.

Figura 20 – Gráfico QQ Plot da Série Temporal de Aldeia de Joanes com transformação por raiz cúbica

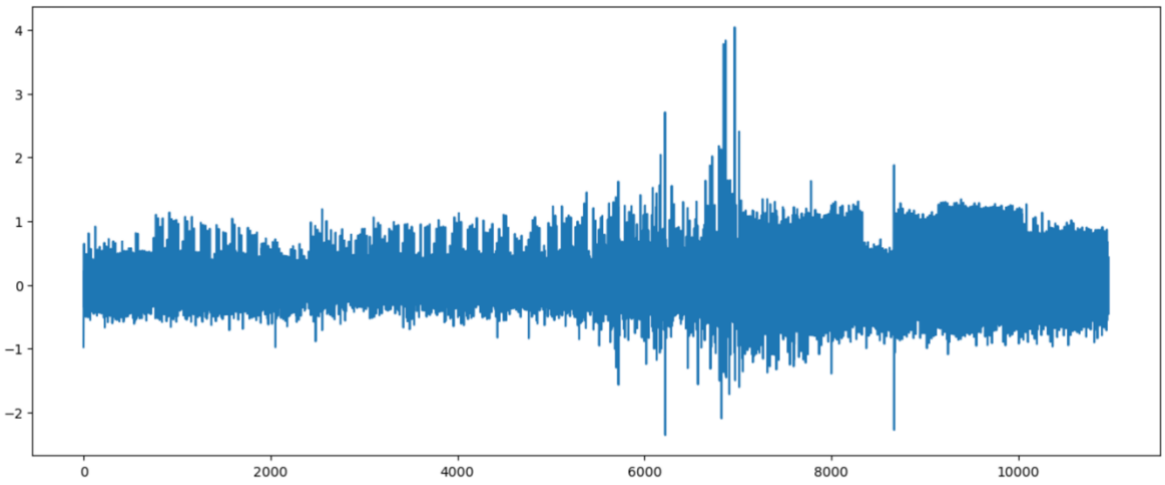


Fonte: Elaboração Própria

Por último, foi aplicada uma diferenciação aos dados da série com o objetivo de melhorar a sua estacionaridade, já que se tinha concluído que a mesma tinha uma fraca estacionaridade. O objetivo era verificar se o teste *KPSS* passava a classificar a série como estacionária e se, visualmente, era possível obter um gráfico com a média e a variância mais constantes.

O resultado da diferenciação pode ser observado na Figura 21 e os resultados numéricos revelaram as mesmas conclusões a que inicialmente, ou seja, o teste KPSS voltou a classificar a série como não estacionária e os outros classificaram-na como estacionária.

Figura 21 – Resultado da Diferenciação dos Dados da Série Temporal de Aldeia de Joanes



Fonte: Elaboração Própria

Assim, concluiu-se que a diferenciação dos dados da série temporal de Aldeia de Joanes não melhorava a estacionaridade da mesma.

3.2.3. Setor de Degolados

3.2.3.1. Resultados do Pré-Tratamento

Foram lidos os 15 ficheiros que continham os dados do setor de Degolados e criado um *Dataframe* onde se observou que tinham sido importados 131585 registos e 10 atributos.

O *Datalogger* deste setor efetuava um registo a cada 5 minutos, pelo que, para os 15 meses em causa, o número de registos esperado era de 131616, este facto indicava que existiam certamente problemas com os registos dos dados, pois o número não coincidia.

Da observação dos atributos existentes concluiu-se que apenas o atributo *Data/Hora* e o atributo *CDEP* continham os dados requeridos, pelo que, todos os restantes foram eliminados, Figura 22. O atributo *Data/Hora* continha os dados com a informação da data e hora de cada registo e o atributo *CDEP* continha os dados com informação do caudal registado, em litros por segundo, a cada 5 minutos.

Figura 22 – Eliminação de Atributos não Relevantes do Setor de Degolados

```
# Eliminação das colunas que não contêm dados relevantes para o estudo em causa.
df_total = df_total.loc[:, ['Data/Hora', 'CDEP']]
df_total.head()
```

	Data/Hora	CDEP
0	01/10/21 00:00	0,467
1	01/10/21 00:05	0,367
2	01/10/21 00:10	0,333
3	01/10/21 00:15	0,267
4	01/10/21 00:20	0,2

Fonte: Elaboração Própria

O formato dos dados dos atributos obtidos foi do tipo *object* para ambos pelo que foi necessário proceder a uma conversão dos mesmos. Os dados do atributo *Data/Hora* foram convertidos para o tipo *datetime* e os dados do atributo *CDEP* foram convertidos para o tipo *float* tendo o cuidado de substituir as vírgulas por pontos antes da conversão, Figura 23.

Figura 23 – Formatação dos Dados do Dataframe do Setor de Degolados

```
df_total['Data/Hora'] = pd.to_datetime(df_total['Data/Hora'], format='%d/%m/%y %H:%M')
df_total['CDEP'] = pd.to_numeric(df_total['CDEP'].str.replace(',', '.'))
df_total.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 131585 entries, 0 to 131584
Data columns (total 2 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   Data/Hora  131585 non-null  datetime64[ns]
 1   CDEP       131585 non-null  float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 2.0 MB
```

Fonte: Elaboração Própria

No teste de equidistância realizado foi confirmado que a série tinha problemas de equidistância, como de resto já era de esperar uma vez que o número de registos era diferente do esperado. Adicionando o atributo *equidistance* e posteriormente bloqueando a impressão do *Dataframe* sempre que o seu valor fosse igual a *False*, observou-se a existência de nove valores não equidistantes e com necessidade de serem analisados.

Das análises dos valores da vizinhança de cada caso observado, verificou-se a existência de 3 registos ligeiramente desfasados, 12 registos em falta relativos à mudança horária de inverno para verão e 19 registos em falta de forma pontual. Foi implementada a metodologia descrita no ponto 3.1.4. para todos os casos e finalmente foi obtido um *Dataframe* com 131616 elementos e com uma frequência de 5 minutos.

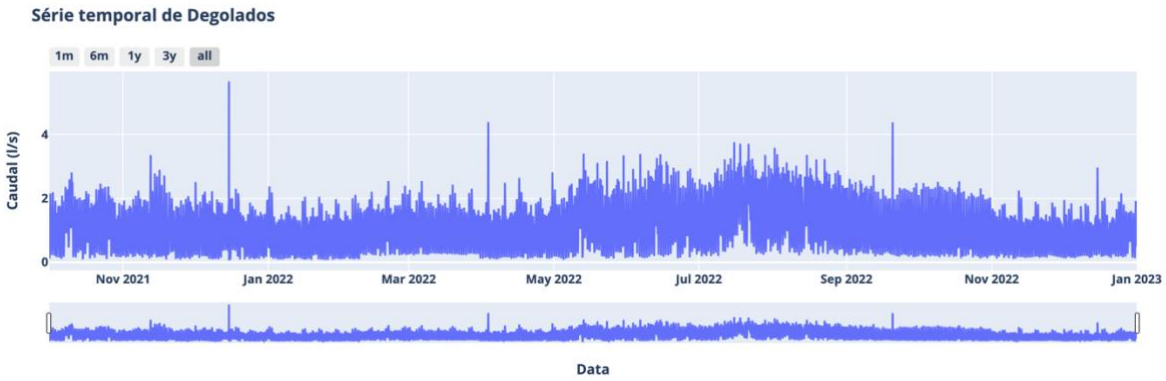
Na verificação da existência de zeros ao longo do *Dataframe*, foram encontrados 21 resultados, no entanto, pelo conhecimento existente do funcionamento deste setor, foi considerado que os mesmos representam a realidade e não se procedeu à sua substituição.

Por último foi realizada a reamostragem dos dados, de acordo com a metodologia descrita no ponto 3.1.5. obtendo no final 10968 elementos, ou seja, um elemento por cada hora, e foi realizada a exportação dos dados para um ficheiro com o nome *Degolados.csv*.

3.2.3.2. Análise de Dados

Da análise da estacionaridade dos dados do setor de Degolados recorrendo ao gráfico da sua distribuição, Figura 24, concluiu-se que a média e a variância variam ao longo do tempo pelo que, pela mera observação do gráfico, concluiu-se pela não estacionaridade.

Figura 24 – Distribuição da Série Temporal de Degolados



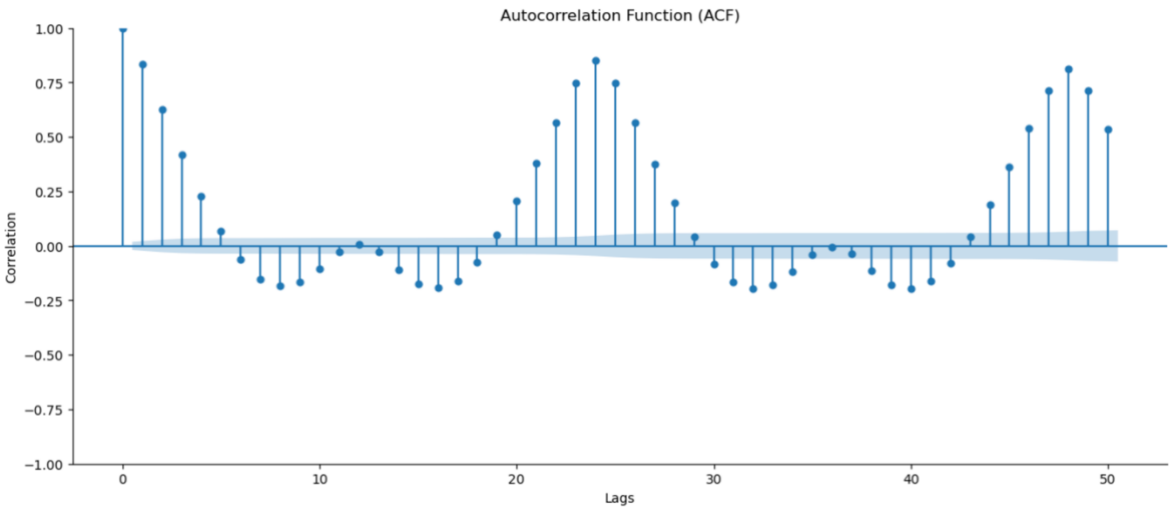
Fonte: Elaboração Própria

A aplicação dos testes numéricos para verificação da estacionaridade revelaram algo já verificado em outras séries, ou seja, o teste *KPSS* revelou que a série era não estacionária e os testes *ADF* e *PP* indicaram que a série era estacionária.

Considerando que o teste *KPSS* apresenta alguns problemas para amostras muito grandes, que é o caso, e que os restantes testes indicam que a série é estacionária, e tendo em conta o observado graficamente, concluiu-se que a série apresentava uma fraca estacionaridade.

No que respeita à existência de autocorrelação, a análise do diagrama de autocorrelação da série temporal de Degolados revelou a existência de uma forte dependência entre as *lags* adjacentes, Figura 25.

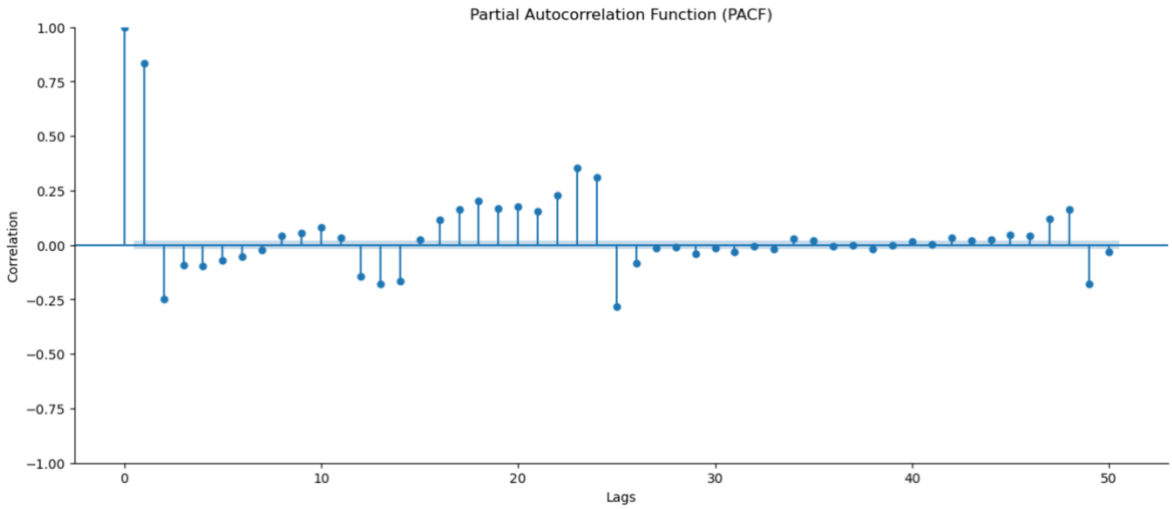
Figura 25 – Diagrama de Autocorrelação de Degolados



Fonte: Elaboração Própria

O gráfico de autocorrelação parcial também indicou autocorrelação entre as *lags*, neste caso não adjacentes, mas menor do que no caso das *lags* adjacentes, Figura 26.

Figura 26 – Diagrama de Autocorrelação Parcial de Degolados



Fonte: Elaboração Própria

Em relação à decomposição da série, foi utilizada uma frequência de 24, uma vez que temos um registo por hora e é expectável a existência de uma sazonalidade a cada 24 horas.

Dos gráficos da série decomposta, Figura 27, notou-se a existência de uma tendência muito irregular durante os primeiros meses, tendencialmente decrescente, passando depois a crescente até aos meses de verão, voltando depois a ser decrescente.

Em relação à sazonalidade apenas se pôde concluir que existe, mas por dificuldade de visualização do gráfico foi difícil concluir se se tratava apenas de uma sazonalidade diária ou se existiam outros períodos possíveis de identificar.

Já o gráfico dos resíduos, por seu lado, mostrou uma certa organização com valores aparentemente bem estruturados, o que levou a concluir que poderiam estar com problemas de autocorrelação.

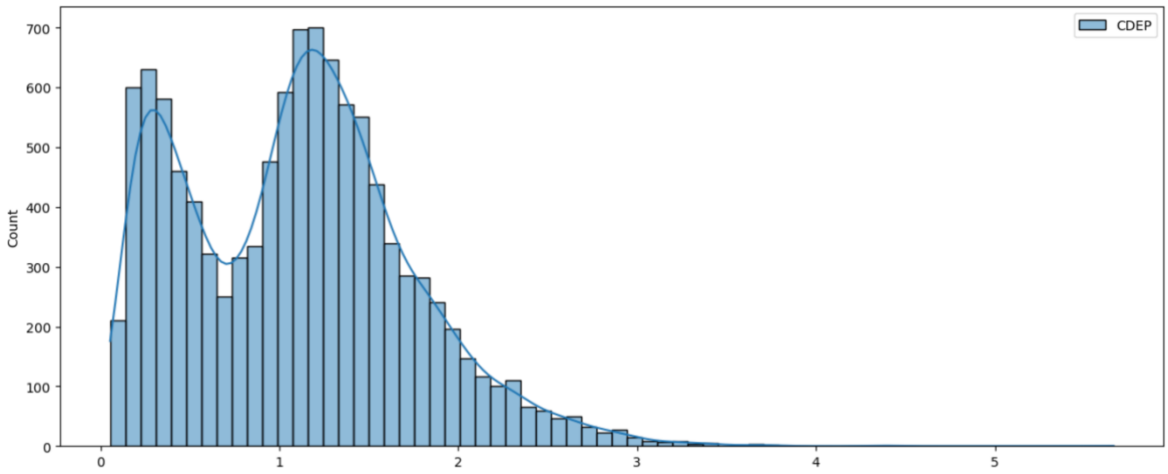
Figura 27 – Decomposição da Série de Degolados

Fonte: Elaboração Própria

Analisando a autocorrelação e a autocorrelação parcial dos resíduos verificou-se a existência de uma forte autocorrelação em *lags* adjacentes. Nas *lags* não adjacentes também se verificou existência de autocorrelação, mas menos pronunciada do que no caso anterior.

Em relação à Análise da Normalidade, tal como ocorreu nas séries temporais anteriores, tanto a análise gráfica da sua distribuição como os testes numéricos utilizados, revelaram que a mesma não seguia uma distribuição Normal, como de resto pode ser observado na Figura 28.

Figura 28 – Histograma de Distribuição da Série Temporal de Degolados



Fonte: Elaboração Própria

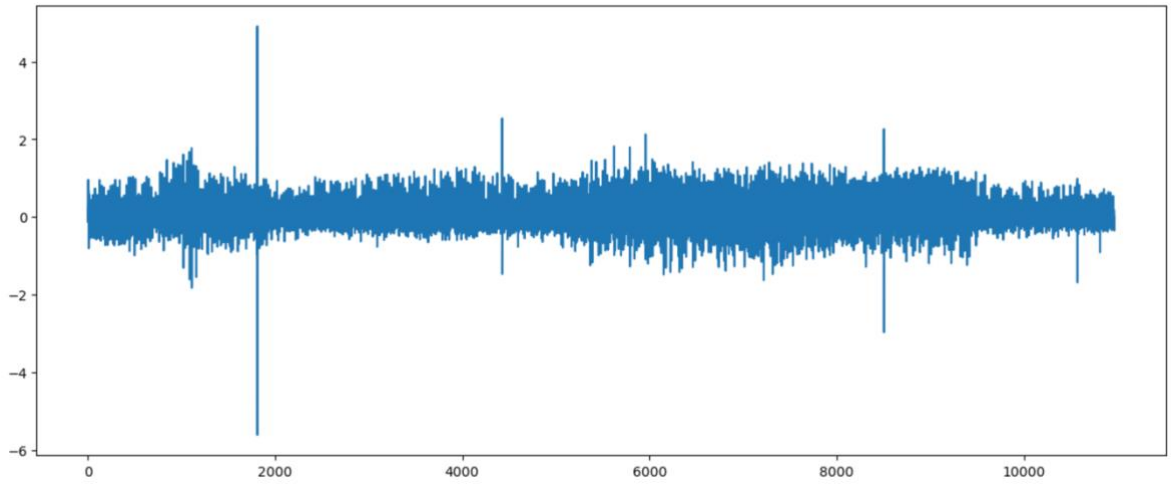
Efetuando as transformações referidas no ponto 3.1.10.3. da metodologia, estas não ajudaram de todo a melhorar a normalidade da série. Tanto os testes estatísticos como a observação gráfica mostraram uma distribuição dos dados muito afastada daquilo que seria uma distribuição normal e sem melhorias no que se refere aos dados originais.

Finalmente, foi aplicada uma diferenciação aos dados da série com o objetivo de melhorar a sua estacionaridade, já que se tinha concluído que a mesma tinha uma fraca estacionaridade. O objetivo era verificar se o teste *KPSS* passava a classificar a série como estacionária e se, visualmente, era possível obter um gráfico com a média e a variância mais constantes.

Os resultados numéricos revelaram as mesmas conclusões a que inicialmente, ou seja, o teste KPSS voltou a classificar a série como não estacionária e os outros classificaram-na como estacionária. Visualmente,

Figura 29, percebeu-se que a média e a variância continuavam com algumas oscilações pelo que a diferenciação não melhorava a estacionaridade.

Figura 29 – Resultado da Diferenciação dos Dados da Série Temporal de Degolados



Fonte: Elaboração Própria

3.2.4. Setor de Janeiro de Cima

3.2.4.1. Resultados do Pré-Tratamento

Da leitura dos 15 ficheiros que continham os dados do setor de Degolados, foi criado um *Dataframe* onde se observou que tinham sido importados 131601 registos e 10 atributos.

O *Datalogger* deste setor efetuava um registo a cada 5 minutos, pelo que, para os 15 meses em causa, o número de registos esperado era de 131616, este facto indicava que existiam certamente problemas com os registos dos dados, pois o número não coincidia.

Da observação dos atributos existentes concluiu-se que apenas o atributo *Data/Hora* e o atributo *CDist* continham os dados requeridos, pelo que, todos os restantes foram eliminados, Figura 30. O atributo *Data/Hora* continha os dados com a informação da data e hora de cada registo e o atributo *CDist* continha os dados com informação do caudal registado, em litros por segundo, a cada 5 minutos.

Figura 30 – Eliminação de Atributos não Relevantes do Setor de Janeiro de Cima

```
# Eliminação das colunas que não contêm dados relevantes para o estudo em causa.
df_total = df_total.loc[:, ['Data/Hora', 'CDist']]
df_total.head()
```

	Data/Hora	CDist
0	01/10/21 00:00	0,467
1	01/10/21 00:05	0,433
2	01/10/21 00:10	0,333
3	01/10/21 00:15	0,3
4	01/10/21 00:20	0,3

Fonte: Elaboração Própria

Observando o tipo de dados dos atributos em causa, verificou-se que estes eram do tipo *object* em ambos os casos, pelo que foi necessário proceder a uma conversão dos mesmos. Os dados do atributo *Data/Hora* foram convertidos para o tipo *datetime* e os dados do atributo *CDist* foram convertidos para o tipo *float* tendo o cuidado de substituir as vírgulas por pontos antes da conversão, Figura 31.

Figura 31 – Formatação dos Dados do Dataframe do Setor de Janeiro de Cima

```
df_total['Data/Hora'] = pd.to_datetime(df_total['Data/Hora'], format='%d/%m/%y %H:%M')
df_total['CDist'] = pd.to_numeric(df_total['CDist'].str.replace(',', '.'))
df_total.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 131601 entries, 0 to 131600
Data columns (total 2 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   Data/Hora  131601 non-null  datetime64[ns]
 1   CDist      131601 non-null  float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 2.0 MB
```

Fonte: Elaboração Própria

Relativamente ao teste de equidistância entre as datas, foi confirmado que a série tinha problemas de equidistância, tal como previsto anteriormente pelo número de elementos obtido no momento da importação dos ficheiros. Adicionando o atributo *equidistance* e posteriormente bloqueando a impressão do *Dataframe* sempre que o seu valor fosse igual a *False*, observou-se a existência de 6 valores não equidistantes e com necessidade de serem analisados.

Das análises dos valores da vizinhança de cada caso observado, verificou-se a existência de 2 registos ligeiramente desfasados, 12 registos em falta relativos à mudança horária de inverno para verão e 3 registos em falta de forma pontual. Foi implementada a metodologia descrita no ponto 3.1.4. para todos os casos e finalmente foi obtido um *Dataframe* com 131616 elementos e com uma frequência de 5 minutos.

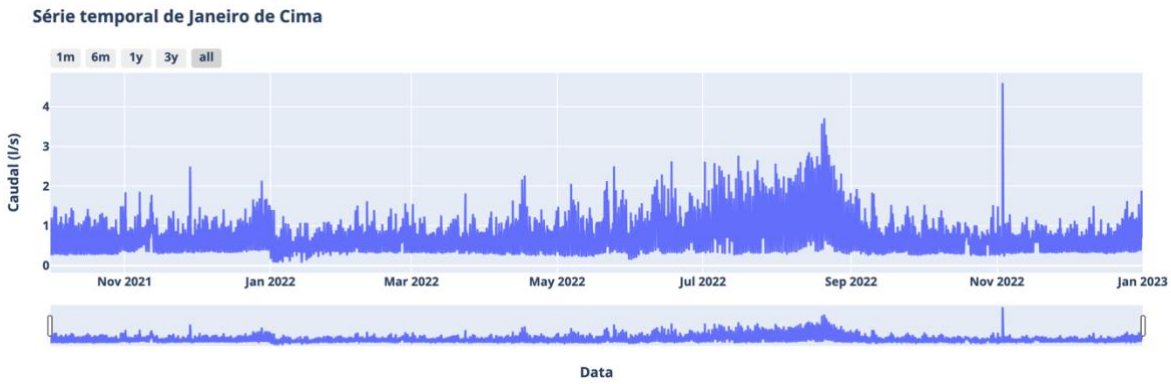
Passando à verificação da existência de zeros ao longo do *Dataframe*, estes, não foram encontrados ao longo de todo o *Dataframe*.

Por último foi realizada a reamostragem dos dados, de acordo com a metodologia descrita no ponto 3.1.5. obtendo no final 10968 elementos, ou seja, um elemento por cada hora, e foi feita a exportação dos dados para um ficheiro com o nome *JaneiroDeCima.csv*.

3.2.4.2. Análise de Dados

Analisando os dados do setor de Janeiro de Cima e no que à estacionaridade respeita, verificou-se que existe bastante inconsistência da média e da variância. A série apresentava um comportamento bastante aleatório ao longo do tempo com uma variabilidade muito grande na sua amplitude, Figura 32.

Figura 32 – Distribuição da Série Temporal de Janeiro de Cima



Fonte: Elaboração Própria

Assim, e pela mera observação gráfica, concluiu-se pela não estacionaridade da série.

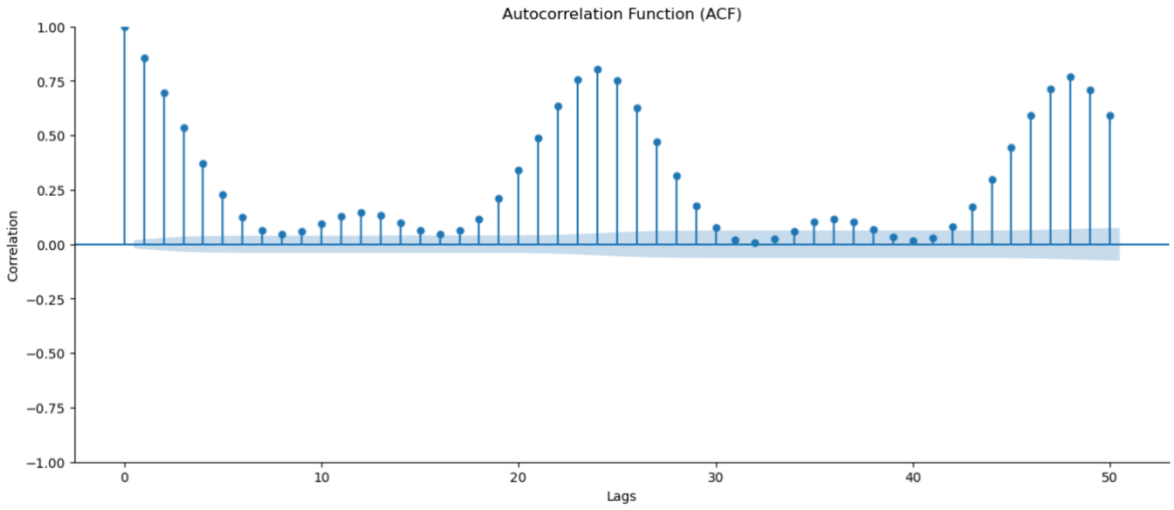
No que respeita à aplicação dos testes numéricos para verificação da estacionaridade, verificou-se que o teste *KPSS* indicava que a série era não estacionária enquanto os

testes *ADF* e *PP* indicaram precisamente o contrário, característica esta já comum a outras séries analisadas.

Considerando que o teste *KPSS* apresenta alguns problemas para amostras muito grandes, que é o caso, e que os restantes testes indicam que a série é estacionária, e apesar do que tinha sido observado graficamente, concluiu-se que a série apresentava uma fraca estacionaridade.

Relativamente à existência de autocorrelação, a análise do diagrama de autocorrelação da série temporal de Janeiro de Cima revelou a existência de uma forte dependência entre as *lags* adjacentes, Figura 33.

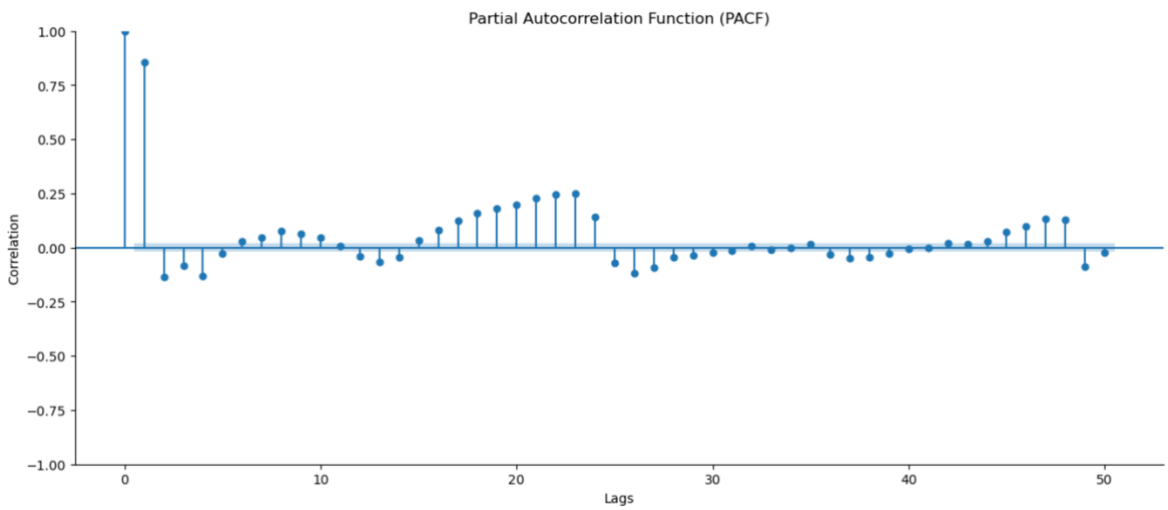
Figura 33 – Diagrama de Autocorrelação de Janeiro de Cima



Fonte: Elaboração Própria

O gráfico de autocorrelação parcial também indicou autocorrelação entre as *lags* não adjacentes, mas menor do que no caso das *lags* adjacentes, Figura 34.

Figura 34 – Diagrama de Autocorrelação Parcial de Janeiro de Cima



Fonte: Elaboração Própria

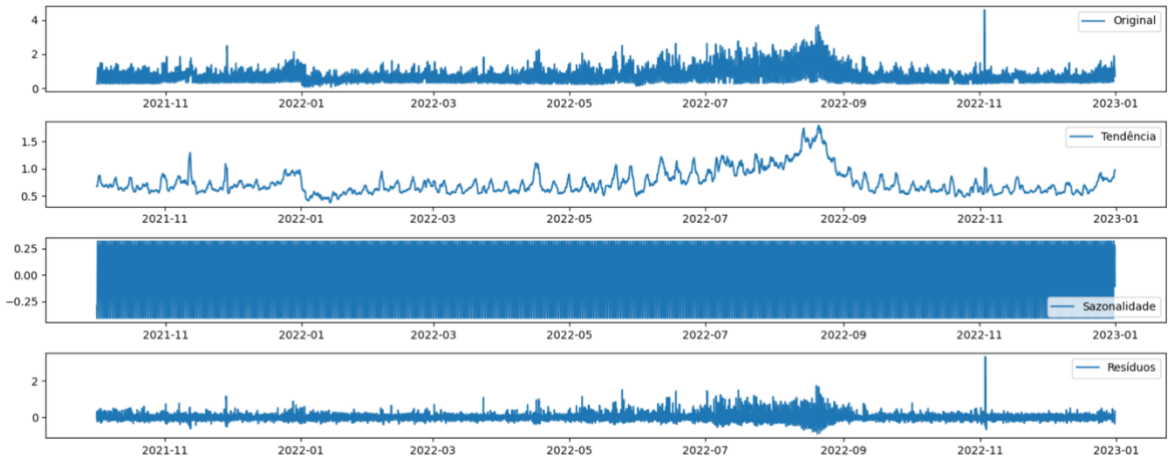
Passando à decomposição da série, foi utilizada uma frequência de 24, uma vez que temos um registo por hora e é expectável a existência de uma sazonalidade a cada 24 horas.

Dos gráficos da série decomposta, Figura 35, notou-se a existência de uma tendência muito irregular durante os primeiros meses, ligeiramente decrescente, passando depois a crescente até aos meses de verão, voltando depois a ser decrescente.

Em relação à sazonalidade apenas se pôde concluir que existe, mas por dificuldade de visualização do gráfico foi difícil concluir se se tratava apenas de uma sazonalidade diária ou se existiam outros períodos possíveis de identificar.

Já o gráfico dos resíduos, por seu lado, mostrou uma certa organização com valores aparentemente bem estruturados, o que levou a concluir que poderiam estar com problemas de autocorrelação.

Figura 35 – Decomposição da Série de Janeiro de Cima

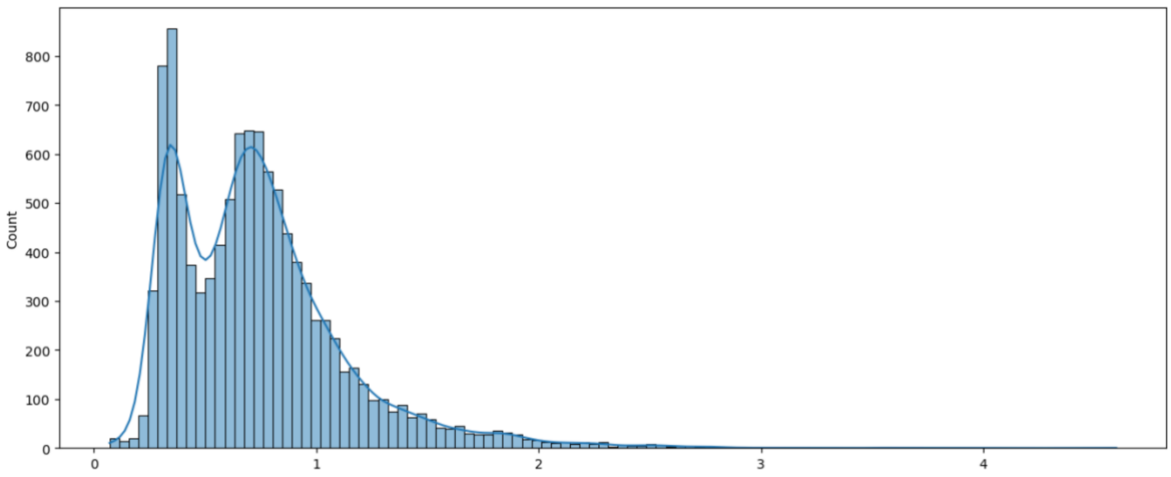


Fonte: Elaboração Própria

Analisando a autocorrelação e a autocorrelação parcial dos resíduos verificou-se a existência de uma forte autocorrelação tanto na *lags* adjacentes como nas *lags* não adjacentes, confirmando as suspeitas geradas pela observação do gráfico dos resíduos.

No que respeita à Análise da Normalidade, mais uma vez, tanto a análise gráfica da sua distribuição, Figura 36, como os testes numéricos utilizados, revelaram que a mesma não seguia uma distribuição Normal.

Figura 36 – Histograma de Distribuição da Série Temporal de Janeiro de Cima



Fonte: Elaboração Própria

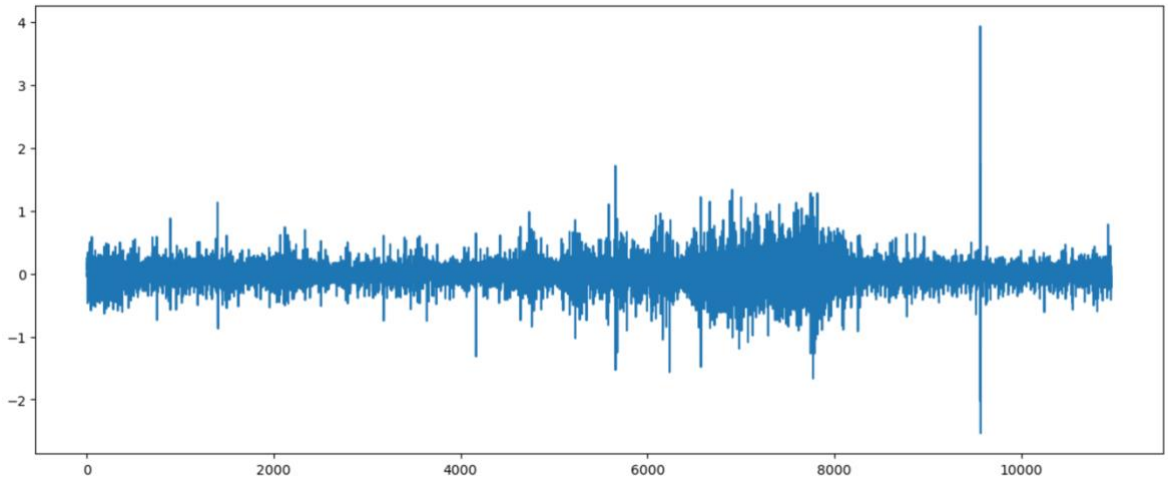
Efetuando as transformações referidas no ponto 3.1.10.3. da metodologia, estas não ajudaram a melhorar a normalidade da série. Tanto os testes estatísticos como a

observação gráfica mostraram uma distribuição dos dados muito afastada daquilo que seria uma distribuição normal e sem melhorias no que se refere aos dados originais.

Por último, foi aplicada uma diferenciação aos dados da série com o objetivo de melhorar a sua estacionaridade, já que se tinha concluído que a mesma tinha uma fraca estacionaridade. O objetivo era verificar se o teste *KPSS* passava a classificar a série como estacionária e se, visualmente, era possível obter um gráfico com a média e a variância mais constantes.

Os resultados numéricos revelaram as mesmas conclusões a que inicialmente, ou seja, o teste *KPSS* voltou a classificar a série como não estacionária e os outros classificaram-na como estacionária. Visualmente, Figura 37, percebeu-se que a média e a variância continuavam algumas oscilações pelo que a diferenciação não melhorava a estacionaridade.

Figura 37 – Resultado da Diferenciação dos Dados da Série Temporal de Janeiro de Cima



Fonte: Elaboração Própria

CAPÍTULO IV – APLICAÇÃO DOS MODELOS ÀS SÉRIES TEMPORAIS

Neste capítulo apresenta-se a metodologia utilizada para a aplicação dos vários modelos às séries temporais, após a preparação e análise realizadas aos dados no capítulo anterior. Ainda neste capítulo, são apresentados os resultados de cada modelo considerado, em alguns casos com as variações consideradas pertinentes para a correta avaliação posterior. São também exibidos os valores das métricas obtidos, tanto para o curto prazo como para o longo prazo.

4.1. Metodologia

4.1.1. Importação de Dados

No capítulo anterior, após o pré-processamento, foi criado um ficheiro *csv* para cada localidade com os dados respetivos. Assim, nesta fase, foi lido cada ficheiro e criado um *Dataframe* também para cada localidade utilizando a biblioteca *pandas*.

Uma vez que se tratavam de muitos dados, a utilização de uma exibição gráfica simples apenas permitiria ver uma mancha, pelo que foi utilizada a biblioteca *plotly* e o objeto *graph_objects* para criar um gráfico interativo que permitisse fazer zoom e visualizar os detalhes da sazonalidade diária que, de outra forma, seria impossível.

Assim, após a importação dos dados foi feita uma impressão, utilizando a biblioteca referida anteriormente, com o objetivo de permitir a visualização de toda a série importada.

4.1.2. Separação de Dados entre Treino e Teste

Qualquer dos modelos a estudar requeria a existência de dados de treino e dados de teste. Assim, a amostra utilizada contemplava 12 meses de dados que foram utilizados como dados de treino e 3 meses que foram utilizados como dados de teste. No caso do modelo LSTM existiu ainda a necessidade de ter dados de validação para a etapa de treino do modelo. Neste caso utilizaram-se os dados dos primeiros 9 meses como dados de treino, os seguintes 3 meses como dados de validação e os restantes 3 como dados de teste, Tabela 1.

Tabela 1 – Separação entre Dados de Treino, Validação e Teste

Modelos	Dados de Treino	Dados de Validação	Dados de Teste
Holt-Winters, ARIMA e *Prophet*	De 01/10/2021 até 30/09/2022 8760 dados	Não aplicável	De 01/10/2022 até 31/12/2022 2208 dados
LSTM	De 01/10/2021 até 30/06/2022 6552 dados	De 01/07/2022 até 30/09/2022 2208 dados	De 01/10/2022 até 31/12/2022 2208 dados

Fonte: Elaboração Própria

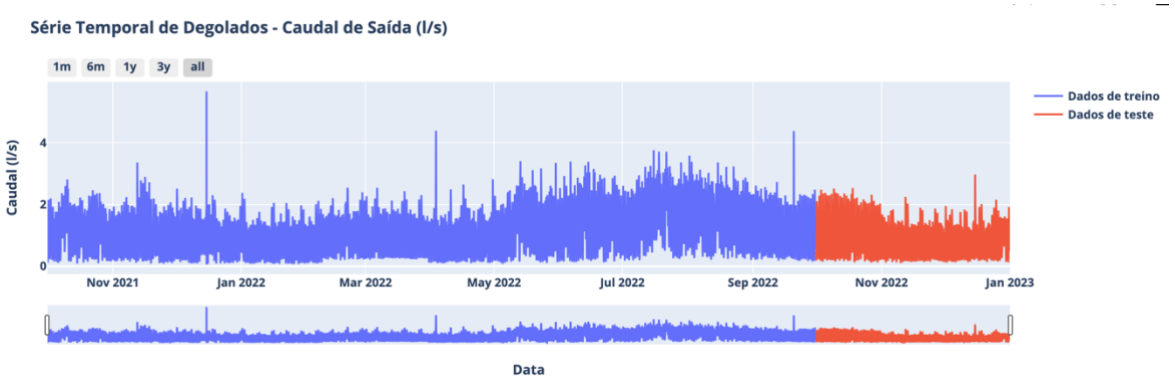
Os dados separados foram obtidos a partir dos *Dataframes* importados para objetos do tipo *panda series*:

a) train_data: objeto com os dados de treino;

b) test_data: objeto com os dados de teste.

Uma vez tendo os dados separados foi exibida uma impressão dos mesmos onde, utilizando cores distintas se podia verificar se a separação tinha sido bem efetuada, Figura 38.

Figura 38 – Exemplo de Gráfico com os Dados Separados



Fonte: Elaboração Própria

4.1.3. Ajustes e Aplicação dos Modelos aos Dados

4.1.3.1. Modelo *Holt-Winters*

Para aplicação do modelo *Holt-Winters* foi utilizada a biblioteca *statsmodels.tsa.holtwinters* e o objeto *ExponentialSmoothing*.

Este modelo admite como parâmetros:

a) ***trend***: refere-se à tendência, mais especificamente ao tipo de equação de tendência que pode ser aditiva ou multiplicativa. Foram feitos testes com ambas, mas no caso da multiplicativa, ao compilar o modelo, este devolvia avisos de *overflow* ou erros de convergência pelo que se optou, em todos os casos, pela equação aditiva;

b) ***seasonal***: tal como na tendência, refere-se ao tipo de equação de sazonalidade que pode ser aditiva ou multiplicativa. Optou-se em todos os casos pela aditiva pois a multiplicativa deu origem a modelos desajustados e sem capacidade de previsão;

c) ***seasonal_periods***: o número de elementos que compõem um ciclo de sazonalidade. Foi escolhido 24 em todos os casos devido à sazonalidade diária observada e uma vez que tínhamos 24 dados em cada dia;

d) ***freq***: a frequência dos dados. Neste caso tínhamos em todos os conjuntos uma frequência horária pelo que foi considerado o valor *H*.

e) ***initialization_method***: o método de inicialização. Após alguns testes foi escolhido o método *heuristic* pois foi o que permitiu obter modelos mais eficientes;

f) ***use_BoxCox***: transformação BoxCox que é uma técnica utilizada para estabilizar a variabilidade dos dados e tornar a distribuição dos mesmos mais próxima de uma distribuição normal. Este parâmetro não foi consensual em todas as séries e refere-se à aplicação ou não de uma transformação aos dados.

Foram testadas várias variações de parâmetros individualmente e em conjunto, mas a configuração que permitiu obter melhores resultados foi a da Figura 39.

Figura 39 – Parâmetros do Modelo *Holt-Winters*

```
# Holt-Winters model
# Parameters
trend = 'add'
seasonal = 'add'
seasonal_periods = 24
freq = 'H'
initialization_method = 'heuristic'
```

Fonte: Elaboração Própria

4.1.3.1.1. Dados Originais e Dados Transformados

Para aplicação do modelo aos dados, após a separação entre dados de treino e de teste, foram utilizados como dados de treino, os dados de treino originais, os dados de treino com uma transformação logarítmica e os dados de treino com uma transformação por raiz cúbica.

No caso dos dados originais o valor do parâmetro *use_BoxCox* utilizado foi o *False* para todas as séries, com exceção da série de Janeiro de Cima onde foi utilizado o valor *True*. Estes valores foram escolhidos por experimentação e selecionados de forma a obter os melhores resultados.

Para os dados transformados, quer logaritmicamente quer por raiz cubica, o parâmetro *use_Box_Cox* utilizado foi um valor fixo de 0.05, para as séries de Alcáçova e Janeiro de Cima, sendo *False* para as restantes. Estes valores foram selecionados da mesma forma explicada anteriormente.

4.1.3.1.2. Obtenção das Previsões do Modelo

Após a parametrização do modelo de cada série temporal foi realizado o treino do mesmo obtendo o modelo treinado utilizando dados originais e dados transformados.

Já com o modelo treinado foram obtidas séries que continham os dados de treino previstos pelo modelo e as previsões do modelo para uma janela temporal de 3 meses após a data do último dado de treino. Esta janela temporal permitiu depois, de acordo com a seleção de realizada, verificar as previsões de curto prazo e as de longo prazo. Tudo isto com o objetivo de comparar os dados de treino reais com os de treino previstos e os dados de teste reais com os dados de teste previstos, ou seja, de conseguir apurar a eficiência do modelo através das suas métricas.

4.1.3.1.3. Visualização Gráfica dos Dados

O modelo treinado permitiu obter os dados de treino ajustados pelo modelo e os dados previstos. Os dados previstos foram obtidos indicando o número de elementos a prever, neste caso, o mesmo número dos elementos contidos na serie de dados de teste.

Seguindo esta metodologia foi possível obter os seguintes objetos do tipo *panda series*:

a) Uma série com os dados de treino ajustados pelo modelo (*train_data_fitted*);

b) Uma série com os dados previstos (*predictions*).

No caso dos modelos que utilizaram dados transformados, foi necessário efetuar uma reversão da transformação após efetuar as previsões e antes de efetuar qualquer tipo de comparação.

A estas séries, juntaram-se as obtidas com os dados originais referidas no ponto 4.1.2. obtendo-se assim um conjunto de 4 séries que permitiram criar um gráfico com as mesmas, sobrepostas, com o objetivo de permitir uma verificação visual da eficiência do modelo. Esta visualização foi realizada para as 4 localidades e utilizando os dados originais e os dados transformados, obtendo-se um total de 12 gráficos.

4.1.3.2. Modelo ARIMA

No caso em apreço, foi verificada a existência de sazonalidade para todas as séries em estudo, pelo que se concluiu que, o modelo ARIMA por si, só não tinha capacidade para lidar com as características das nossas séries temporais.

Assim, para poder lidar com a sazonalidade, foi utilizada uma extensão do modelo ARIMA chamada de SARIMA, que não é mais do que o modelo base, mas com suporte para a sazonalidade. Este modelo representa-se por SARIMA(p, d, q)(P, D, Q)[x], onde os novos parâmetros P, D e Q são os componentes de autorregressão, diferenciação e de média móvel sazonais, respetivamente. O componente x, por seu lado, recebe o valor da frequência da série temporal [19].

4.1.3.2.1. Métodos Utilizados

Para implementar o modelo SARIMA foram utilizados dois métodos: a parametrização manual e a parametrização automática. Para a parametrização manual foi utilizada a classe *SARIMAX* da biblioteca *statsmodels.tsa.statspace.sarimax* e para a parametrização automática foi utilizado a classe *auto_arima* da biblioteca *pmdarima.arima*.

O método manual consistiu em atribuir valores aos componentes não sazonais e sazonais e efetuar a compilação do modelo para cada alteração verificando nos resultados o valor conseguido para o AIC[1]. O objeto criado como uma instância da classe SARIMAX recebeu como parâmetros os dados de treino, os componentes não sazonais, os sazonais e a sazonalidade. Assim procedeu-se à compilação do modelo iniciando todos componentes a zero, com exceção da sazonalidade que foi inicializada e mantida sempre com o valor 24. Este valor deve-se ao facto de se ter verificado a existência de sazonalidade diária e de existirem 24 dados de caudal por dia. Depois foram sendo incrementados os valores dos restantes parâmetros, de forma a obter

[1] Akaike information criterion (AIC) é um método matemático para avaliar a capacidade de um modelo se ajustar aos dados a partir dos quais foi gerado. O seu cálculo tem em consideração o número de variáveis independentes utilizadas e a eficiência do modelo em explicar os dados a partir dos quais foi construído. Quanto menor for o valor do AIC melhor será o ajuste [29].

diferentes combinações entre os mesmos. O incremento foi realizado entre os valores 0 e 3, suspendendo o mesmo sempre que se verificou um aumento do AIC, em relação ao seu valor resultado da combinação anterior, ao invés da sua diminuição.

O método automático consistiu em utilizar uma instância da classe *auto_arima* que recebeu os parâmetros indicados na Tabela 2.

Tabela 2 – Parâmetros do método autoARIMA

Parâmetro	Descrição	Valor Atribuído
train_data	Dados de treino no formato *panda series.*	***train_data***
trace	Visualizar o progresso do processo automático.	***True***
stepwise	Indicação se o processo deve passar por todas as combinações possíveis (False) ou se deve evoluir de forma optimizada (True).	***True***
seasonal	Indicação se existe sazonalidade ou não.	***True***
stationary	Indicação se se trata de uma série estacionária ou não.	***True e False***
max_p	O número de ordem máximo para o parâmetro autorregressivo (p).	***3***
max_q	O número de ordem máximo para o parâmetro de média móvel (q).	***2***
max_P	O número de ordem máximo para o parâmetro sazonal autorregressivo (P).	***1***
max_Q	O número de ordem máximo para o parâmetro de sazonal de média móvel (Q).	***1***
start_p	O número de ordem inicial para o parâmetro autorregressivo (p).	***0***
start_q	O número de ordem inicial para o parâmetro de média móvel (q).	***0***
start_P	O número de ordem inicial para o parâmetro sazonal autorregressivo (P).	***0***
start_Q	O número de ordem inicial para o parâmetro de sazonal de média móvel (Q).	***0***
m	O valor da sazonalidade.	***24***

Fonte: Elaboração Própria

Note-se que no caso do parâmetro *stationary* o valor atribuído foram os dois possíveis, *True* e *False*, isto porque no ponto 3.2. foi verificado que a estacionaridade das séries temporais era caracterizada por ser fraca devido a resultados contraditórios nos testes numéricos. Assim foram feitos testes com os dois valores e registados os AIC correspondentes, tendo-se elegido o modelo com o menor AIC.

4.1.3.2.2. Análise de Resíduos

Após obter o modelo com o menor AIC, em ambos os casos, parametrização manual e automática, foram obtidos os resíduos do modelo e realizada uma análise aos mesmos.

A análise dos resíduos consistiu na sua visualização gráfica, utilizando o método *plot* do objeto contendo os resíduos, análise de normalidade, de acordo com a metodologia descrita em 3.1.10. com exceção das transformações, autocorrelação, seguindo a metodologia do ponto 3.1.8., e visualização gráfica quando subtraídos e sobrepostos aos dados de treino.

4.1.3.2.3. Previsões

As previsões dos dados de treino do modelo parametrizado manualmente foram obtidas para o objeto *train_data_adjusted* a partir do método *fitted_values* do objeto que continha o modelo compilado. Já no caso do modelo obtido pelo método automático, as previsões dos dados de treino foram obtidas para o objeto *train_data_adjusted_auto* a partir do mesmo método referido anteriormente, mas, neste caso do modelo obtido de forma automática.

No que respeita às previsões dos dados de teste, estas foram obtidas para os objetos *prediction* e *prediction_auto* a partir do método *forecast* no primeiro caso e *predict* no segundo caso, aplicados aos dois modelos obtidos e passando como parâmetro o número de dados a prever, ou seja, 2208.

4.1.3.3. Modelo LSTM

Para a implementação do modelo LSTM aos dados das séries temporais foram utilizados modelos univariáveis e modelos multivariáveis. Os modelos univariáveis

foram aqueles que utilizavam apenas os valores de caudal das séries temporais e os multivariáveis utilizavam, para além dos valores de caudal, algumas variáveis calculadas que transportavam para o modelo informação sobre a data e hora da ocorrência de cada valor de caudal.

4.1.3.3.1. Padronização e Normalização dos Dados

Quando se utilizam algoritmos baseados em redes neuronais, é usual efetuar antes processos de transformação aos dados por forma a facilitar os processos de otimização dos mesmos. A Padronização e a Normalização são dois desses processos de transformação que, embora não sejam obrigatórios para a implementação de modelos baseados em redes neuronais, podem ser facilitadores dos processos de convergência durante a compilação dos mesmos [27].

Assim, para a implementação do modelo LSTM, foram utilizados dados de caudal originais e dados de caudal padronizados. A normalização foi testada, mas, uma vez que não trouxe nenhuma melhoria em termos de resultados foi descartada. Para os modelos univariáveis foi realizado um teste com dados originais seguido de um teste com dados padronizados. Para obter a padronização dos dados de caudal foi utilizado o método *StandardScaler* da biblioteca *sklearn.preprocessing*. No caso dos modelos multivariáveis, foram feitos testes com dados originais e dados normalizados, mas o procedimento considerado correto foi com a transformação para dados de caudal padronizados, pois foi este que permitiu obter os melhores resultados. Nos casos em que se procedeu à transformação dos dados foi necessário, após a obtenção das previsões, efetuar a respetiva transformação inversa.

4.1.3.3.2. Transformação Matricial dos Dados de Treino e de Validação

Para poder criar um modelo de previsão LSTM foi necessário transformar os dados de cada série temporal em 2 tensores X e y. O tensor X, tensor de entrada, continha a informação sobre os valores de caudal, o intervalo de valores de caudal necessários para gerar uma previsão e o número de atributos que, neste caso, era apenas um, ou seja, o caudal. O tensor y, tensor de saída, continha as previsões. Estes tensores foram

construídos através de uma organização matricial univariável dos dados da série conforme indicado na Figura 40.

Figura 40 – Tensores X e y do Modelo LSTM Univariável

$$\underset{\text{X}}{\begin{bmatrix} [[f_1]\ [f_2]\ [f_3] \cdots [f_{22}]\ [f_{23}]] \\ [[f_2]\ [f_3]\ [f_4] \cdots [f_{23}]\ [f_{24}]] \\ [[f_3]\ [f_4]\ [f_5] \cdots [f_{24}]\ [f_{25}]] \\ \vdots \\ [[f_{n-24}]\ [f_{n-23}]\ [f_{n-22}] \cdots [f_{n-2}]\ [f_{n-1}]] \end{bmatrix}} \underset{\text{y}}{\begin{bmatrix} f_{24} \\ f_{25} \\ f_{26} \\ \vdots \\ f_n \end{bmatrix}}$$

Fonte: Elaboração Própria

Na Figura 40, $f_1, f_2, f_3, ..., f_n$, representam os dados de caudal em que, no caso dos tensores dos dados de treino, *n* era igual a 6552, e no caso dos tensores dos dados de validação, *n* era igual a 2208. Por forma a obter os tensores de uma forma expedita foi criada uma função que recebia como parâmetros uma série temporal como objeto do tipo *panda series*, o intervalo desejado como um número inteiro, e devolvia os dois tensores, X e y respetivamente.

Uma vez que o modelo univariável demostrou ter debilidades sérias, considerou-se necessário construir um modelo que incluísse, para além do caudal, a informação temporal.

Assim, procedeu-se à criação de um modelo multivariável que permitisse adicionar as variáveis que iriam representar a data/hora de cada valor de caudal.

Quando se trabalha com dados temporais em contexto de *Machine Learning*, é importante transformar as propriedades do tempo para que os modelos possam utilizá-las de forma adequada. Muitas vezes as propriedades cíclicas do tempo são relevantes para os problemas em estudo, por exemplo, ao construir um modelo para prever tráfego rodoviário, a hora do dia é um fator importante. Uma boa forma de representar o momento do dia é como um ponto num círculo unitário, utilizando as funções *sen* e *cos* [28].

Partindo então da premissa de que é uma boa prática a representação de cada momento do dia como resultado da combinação das funções *sen* e *cos* num círculo

unitário, a mesma prática foi alargada para a representação de cada momento do ano, o que permitiu adicionar ao modelo a sazonalidade anual para além da diária.

Assim, foi criada uma cópia do *Dataframe* original, com todos os dados, e foi criada uma coluna que foi preenchida com o tempo em segundos que tinha passado desde 1 de janeiro de 1979 às 00:00:00 (UTC) e cada data/hora. O momento do dia foi obtido pela conjugação de dois valores, calculados como o *sen* e o *cos* do resultado da operação de multiplicação do número de segundos obtido por 2π, e subsequente divisão pelo número de segundos que tem um dia, 86400, Figura 41, e o momento do ano foi obtido exatamente da mesma forma substituindo o número de segundos que tem um dia, pelo número de segundos que tem um ano, 31556952, Figura 42. Para o cálculo do número de segundos de um ano foi utilizado o tempo absoluto que a Terra demora a completar uma órbita em torno do Sol, ou seja, 365 dias, 48 minutos e 46 segundos, aproximadamente. Este procedimento evitou os constrangimentos de cálculo relacionados com os anos bissextos.

Figura 41 – Equações de Modulação do Momento do Dia

$$
\begin{aligned}
d_{sin} &= sin(wt) \\
d_{cos} &= cos(wt), \text{onde:} \\
w &= \frac{2\pi}{P_{day}}, \\
t &= \text{valor do timestamp} \\
\text{sendo } P_{day} &= \text{número de segundos de um dia}
\end{aligned}
$$

Fonte: Elaboração Própria

O momento de cada dia foi modulado com funções de onda de período diário, tendo-se obtido valores entre 1 e -1, representando todos os momentos do dia entre as 0h00 e as 23h59, respetivamente.

Figura 42 – Equações de Modulação do Momento do Ano

$$
\begin{aligned}
y_{sin} &= sin(wt) \\
y_{cos} &= cos(wt), \text{onde:} \\
w &= \frac{2\pi}{P_{year}}, \\
t &= \text{valor do timestamp} \\
\text{sendo } P_{year} &= \text{número de segundos de um ano}
\end{aligned}
$$

Fonte: Elaboração Própria

De igual forma, foi obtido o momento de cada ano, mas com funções de onde de período anual, ficando representados todos os momentos do ano, entre as 0h00 do dia 1 de janeiro, até às 23h59 do dia 31 de dezembro.

Foram assim adicionadas mais quatro colunas ao *Dataframe* com os valores calculados e foi eliminada a coluna da conversão da data/hora para segundos, Figura 43. A partir daqui foi repetido o processo de separação dos dados de treino, validação e teste conforme o ponto 4.1.2. no que respeita à divisão entre treino validação e teste. Em relação aos objetos criados, pelo motivo de se terem obtido séries multivariáveis, não poderiam ser convertidas em objetos do tipo *panda series,* pelo que se mantiveram como *Dataframes*.

Figura 43 – Dataframe com Multivariáveis que Representam a Data/Hora

```
flow_df = flow_df.drop('Seconds', axis=1)
flow_df.head()
```

Data/Hora	SAIDA	Day sin	Day cos	Year sin	Year cos
2021-10-01 00:00:00	5.000	-1.121614e-11	1.000000	-0.999987	-0.005150
2021-10-01 01:00:00	5.278	2.588190e-01	0.965926	-0.999990	-0.004433
2021-10-01 02:00:00	4.167	5.000000e-01	0.866025	-0.999993	-0.003717
2021-10-01 03:00:00	3.333	7.071068e-01	0.707107	-0.999996	-0.003000
2021-10-01 04:00:00	2.500	8.660254e-01	0.500000	-0.999997	-0.002283

Fonte: Elaboração Própria

A organização matricial multivariável dos dados da série por forma a obter os tensores X e y foi realizada conforme indica a Figura 44.

Figura 44 – Tensores X e y do Modelo LSTM Multivariável

$$\underset{\text{X}}{\begin{bmatrix} \begin{bmatrix} [f_1, d_{sin_1}, d_{cos_1}, y_{sin_1}, y_{cos_1}] & [f_2, d_{sin_2}, d_{cos_2}, y_{sin_2}, y_{cos_2}] & \dots & [f_{23}, d_{sin_{23}}, d_{cos_{23}}, y_{sin_{23}}, y_{cos_{23}}] \\ [f_2, d_{sin_2}, d_{cos_2}, y_{sin_2}, y_{cos_2}] & [f_3, d_{sin_3}, d_{cos_3}, y_{sin_3}, y_{cos_3}] & \dots & [f_{24}, d_{sin_{24}}, d_{cos_{24}}, y_{sin_{24}}, y_{cos_{24}}] \\ [f_3, d_{sin_3}, d_{cos_3}, y_{sin_3}, y_{cos_3}] & [f_4, d_{sin_4}, d_{cos_4}, y_{sin_4}, y_{cos_4}] & \dots & [f_{25}, d_{sin_{25}}, d_{cos_{25}}, y_{sin_{25}}, y_{cos_{25}}] \end{bmatrix} \\ \vdots \\ [[f_{n-24}, d_{sin_{n-24}}, d_{cos_{n-24}}, y_{sin_{n-24}}, y_{cos_{n-24}}] \dots [f_{n-1}, d_{sin_{n-1}}, d_{cos_{n-1}}, y_{sin_{n-1}}, y_{cos_{n-1}}]] \end{bmatrix}} \underset{\text{y}}{\begin{bmatrix} f_{24} \\ f_{25} \\ f_{26} \\ \vdots \\ f_n \end{bmatrix}}$$

Fonte: Elaboração Própria

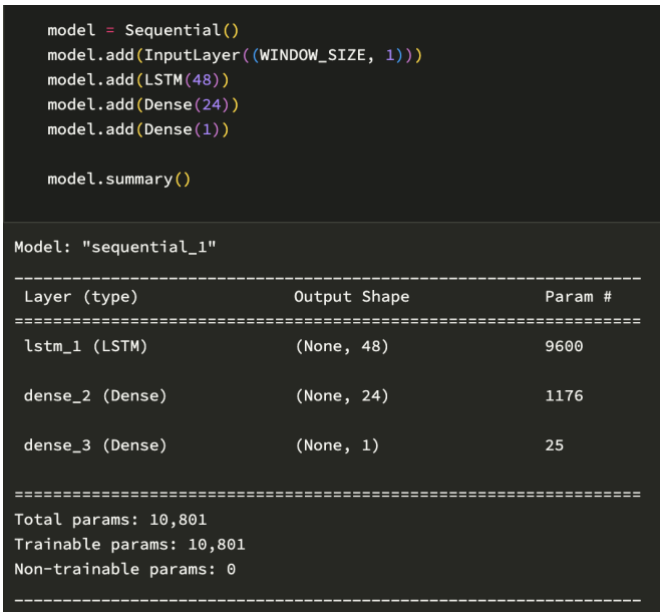
Na Figura 44, f_1, f_2, f_3, ..., f_n, representavam os dados de caudal, d_{sin1}, d_{sin2}, d_{sin3}, ..., d_{sinn-1}, d_{cos1}, d_{cos2}, d_{cos3}, ..., d_{cosn-1}, representavam as coordenadas que identificavam o momento do dia, e y_{sin1}, y_{sin2}, y_{sin3}, ..., y_{sinn-1}, y_{cos1}, y_{cos2}, y_{cos3}, ..., y_{cosn-1}, representavam as coordenadas que identificavam o momento do ano em que aquele valor de caudal tinha ocorrido. Tal como no modelo LSTM Univariável, para os tensores dos dados de treino, *n* era igual a 6552 e para os dados de validação, *n* era igual a 2208. Também neste caso foi criada uma função que permitiu obter os tensores de uma forma simples, recebendo neste caso como parâmetros um *Dataframe* e o intervalo desejado como um número inteiro, devolvendo depois os dois tensores X e y, respetivamente.

4.1.3.3.3. Arquitetura do Modelo

Para a implementação do modelo LSTM foram utilizados os objetos *Sequential, LSTM, Dense, InputLayer, ModelCheckpoint* e *Adam* da biblioteca *tensorflow*.

A arquitetura utilizada para o modelo univariável tinha uma camada de entrada com o número de valores definidos como necessários para gerar uma previsão, uma camada LSTM com 48 nós, uma camada densa com 24 nós e apenas um nó de saída que nos dava o valor previsto. Foram testados outros valores para os números de nós da camada LSTM, mas 48 foi quele que produziu melhores resultados. Foram testadas inclusões de camadas densas com mais e menos nós antes e depois da camada LSTM, mas o modelo que apresentou a melhor convergência foi aquele que incluía uma camada densa com 24 nós antes da camada de saída, Figura 45.

Figura 45 – Arquitetura do Modelo LSTM Univariável



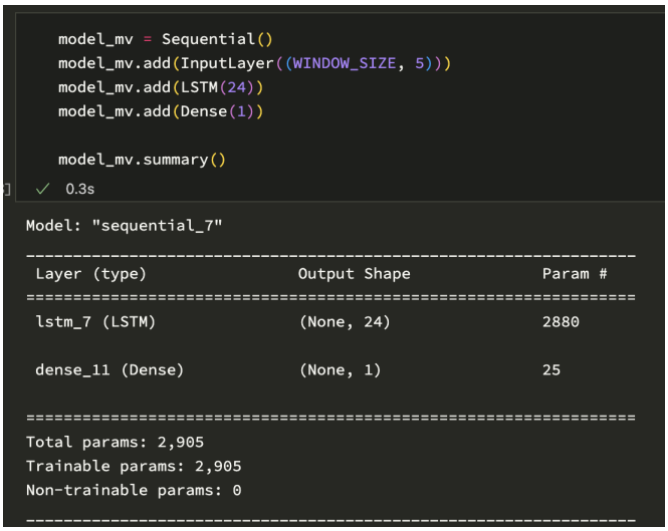
```
    model = Sequential()
    model.add(InputLayer((WINDOW_SIZE, 1)))
    model.add(LSTM(48))
    model.add(Dense(24))
    model.add(Dense(1))

    model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_1 (LSTM)               (None, 48)                9600

 dense_2 (Dense)             (None, 24)                1176

 dense_3 (Dense)             (None, 1)                 25

=================================================================
Total params: 10,801
Trainable params: 10,801
Non-trainable params: 0
_________________________________________________________________
```

Fonte: Elaboração Própria

No caso do modelo multivariável foi testada a mesma arquitetura proposta para o modelo univariável, mas após muitas tentativas concluiu-se que a melhor arquitetura era a que tinha a camada de entrada, agora com 24 x 5 valores definidos como necessários para gerar uma previsão (1 valor de caudal, 2 valores para o momento do dia e 2 valores para o momento do ano), uma camada LSTM com 24 nós e apenas um nó de saída com o valor previsto, Figura 46.

Figura 46 – Arquitetura do Modelo LSTM Multivariável



Fonte: Elaboração Própria

De referir que este modelo, embora tivesse mais variáveis de entrada possuía uma complexidade inferior ao modelo univariável pois tinha menos parâmetros a serem treinados. Existiu, no entanto, uma exceção a esta arquitetura, mais concretamente no número de nós da camada LSTM. Assim, no caso da série temporal de Alcáçova foi verificado que uma arquitetura com uma camada LSTM com 48 nós, permitia obter melhores resultados, pelo que foi a única exceção implementada.

4.1.3.3.4. Compilação e Avaliação do Modelo

Para a compilação do modelo foi criado um objeto *check_point* através do método *ModelCheckpoint* da biblioteca *tensorflow.keras.callbacks* que guardava num ficheiro o modelo que tivesse menos perdas de validação, de acordo com o valor da função de perda obtido, à medida que fossem decorrendo os ciclos de treino.

A compilação do modelo foi feita invocando o método *compile* do modelo criado recebendo como parâmetros:

- *loss=MeanSquaredError();*
- *optimizer=Adam(learning_rate=0.001):*
- *metrics=[RootMeanSquaredError()]*

No caso da série temporal de Janeiro de Cima, foi necessário definir o valor do parâmetro *optimizer* para *Adam(learning_rate=0.0001)*, ou seja, foi necessário diminuir a taxa de aprendizagem do modelo, pois de outra forma não era possível obter valores do coeficiente de determinação positivos.

Finalmente, recorrendo ao método *fit* do modelo e passando como parâmetros os tensores X e y de treino, os tensores X e y de validação, definindo o número de ciclos de treino para 100 e indicando o objeto *check_point* como *callback* para guardar o melhor modelo, foram realizados os ciclos de treino do mesmo.

A avaliação do modelo foi realizada recorrendo ao gráfico de convergência entre os valores da função de perda ocorridas com os dados de treino e sua comparação com os valores da função de perda dos dados de validação ao longo dos ciclos de treino. O objetivo era minimizar ambas as funções de perda, mas, acima de tudo que fossem convergentes à medida que o número de ciclos de treino aumentasse. Quanto melhor fosse a convergência entre as duas curvas melhor seria o modelo.

4.1.3.3.5. Previsões

Para obter as previsões foi inicialmente carregado o melhor modelo que tinha sido guardado através do objeto *ckeck_point*, pois os ciclos de treino podiam a partir de determinada altura começar a divergir e era necessário guardar o melhor modelo logo que ele fosse obtido.

Foram então obtidas as previsões dos dados de treino e de validação ajustadas pelo modelo recorrendo ao método *predict* e passando como parâmetros os dados de treino e de validação. Os primeiros 24 valores de treino ajustados como previsões do modelo não eram previsíveis, de acordo com a Figura 40 e a Figura 44, logo os primeiros 24 valores foram copiados diretamente dos dados de treino reais, sendo esta uma limitação do modelo.

As previsões dos valores novos, ou seja, dos valores que se encontravam no período de teste e que eram desconhecidos para o modelo, foram obtidos seguindo uma metodologia passo a passo. Poderia ter-se simplesmente passado como parâmetro no método *predict*, um vetor com todos os dados de teste, tal como é feito em alguma literatura, e isso produziria um vetor com os dados de teste previstos. No entanto,

pretendeu-se recriar ao máximo um ambiente de produção em que se desconheciam os valores futuros e, portanto, tinha de se prever um a um. Assim foi criado um ciclo de previsão em que, de forma unitária, eram criadas as previsões e colocadas num vetor de previsões até completar o número de previsões dos dados de teste, Figura 47.

Figura 47 – Ciclo de Previsões - Modelo LSTM Univariável

```
multistep_predictions = []

last_x = Xval[-1]
while len(multistep_predictions) < test_data.count():
    p = mymodel.predict(last_x.reshape(1, -1, 1))[0]

    #update the predictions list
    multistep_predictions.append(p)

    #make the new input
    last_x = np.roll(last_x, -1)
    last_x[-1] = p
```

Fonte: Elaboração Própria

Com todos os vetores de previsão, dados de teste ajustados, dados de validação ajustados e dados de teste previstos pelo modelo, foram criadas séries temporais do tipo *panda series* e sobrepostas no gráfico de dados reais a fim de se poder verificar visualmente se as previsões se aproximavam dos valores reais.

4.1.3.4. Modelo *Prophet*

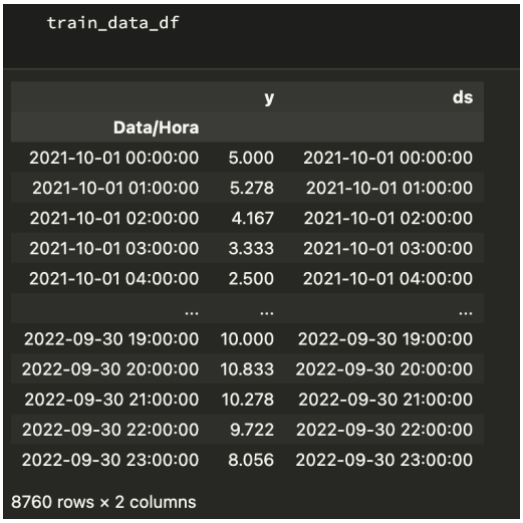
Para implementação do modelo *Prophet* foi utilizado o método *Prophet* da biblioteca *prophet*.

4.1.3.4.1. Estrutura de Dados

Este modelo requere que os dados de teste tenham uma estrutura não de um objeto do tipo *panda series*, mas de um *Dataframe* que contenha uma coluna *y* com os dados da série e uma coluna *ds* com a Data/Hora de cada observação. Assim procedeu-se à conversão do objeto *panda series,* que continha os dados de treino, para um objeto do tipo *Dataframe* com esta configuração. Manteve-se o índice do *Dataframe* com a Data/Hora com o objetivo de não o perder, uma vez que isto não representava nenhum

problema para o procedimento, Figura 48. De notar que não foi necessário efetuar nenhuma transformação aos dados de caudal para obter bons resultados, ou seja, o modelo não se mostrou sensível à utilização dos dados reais.

Figura 48 – Exemplo de Estrutura de Dados de Treino do modelo *Prophet*



train_data_df

Data/Hora	y	ds
2021-10-01 00:00:00	5.000	2021-10-01 00:00:00
2021-10-01 01:00:00	5.278	2021-10-01 01:00:00
2021-10-01 02:00:00	4.167	2021-10-01 02:00:00
2021-10-01 03:00:00	3.333	2021-10-01 03:00:00
2021-10-01 04:00:00	2.500	2021-10-01 04:00:00
...	...	...
2022-09-30 19:00:00	10.000	2022-09-30 19:00:00
2022-09-30 20:00:00	10.833	2022-09-30 20:00:00
2022-09-30 21:00:00	10.278	2022-09-30 21:00:00
2022-09-30 22:00:00	9.722	2022-09-30 22:00:00
2022-09-30 23:00:00	8.056	2022-09-30 23:00:00

8760 rows × 2 columns

Fonte: Elaboração Própria

4.1.3.4.2. Ajuste e Aplicação do Modelo

Para aplicação do modelo às séries foi criada uma instância do objeto *Prophet* com o nome *model*, passando um conjunto de parâmetros selecionados de acordo com os resultados obtidos no ponto 3.2. e também por experimentação, tendo sempre em conta a obtenção dos melhores resultados das métricas. Os parâmetros utilizados dividiram-se entre os que foram comuns para as 4 séries e os que apenas foram definidos em algumas delas. Os que foram comuns são os seguintes:

- ***growth***: refere-se ao crescimento e neste caso foi utilizado o parâmetro *linear* uma vez que o crescimento da variável caudal não tem um limite definido e, portanto, não segue uma função logística;
- ***yearly_seasonality***: refere-se à existência ou não de sazonalidade anual. Foi definido como *True*, ou seja, como existindo sazonalidade anual;
- ***weekly_seasonality***: refere-se à existência ou não de sazonalidade semanal. Foi definido como *False*, ou seja, pelo conhecimento existente dos dados não se

comprovou a existência de sazonalidade semanal. Não obstante, foi experimentado o parâmetro contrário e os resultados das métricas pioravam;

- ***daily_seasonality***: a sazonalidade diária, foi definido como *True* pois ficou bastante comprovado que esta existia.

Depois, nos casos das séries temporais de Alcáçova e de Janeiro de Cima foram ainda adicionados 2 parâmetros aos anteriores:

- ***n_changepoints***: O número de pontos de mudança. Trata-se de um número inteiro que indica o número de pontos de mudança que o modelo deve observar como sendo suscetíveis de conterem alterações na tendência. Este número foi obtido por experimentação com o objetivo de obter as melhores métricas e foi definido como 650;
- ***changepoint_prior_scale***: refere-se à escala de prioridade dos pontos de mudança. Controla a flexibilidade do modelo em encontrar pontos de mudança e o seu valor pode variar entre 0.001 e 0.5. Recorrendo à experimentação foi definido como 0.09.

Uma vez definidos os parâmetros do modelo, foi realizado o ajuste do modelo recorrendo ao método *fit* e passando como parâmetro o *Dataframe* com os dados de treino.

4.1.3.4.3. Obtenção das Previsões

Para obter as previsões foi criado um objeto *future* através do método *make_future_dataframe* do objeto *model* que recebeu como parâmetros o número de períodos a prever, neste caso 2208, a frequência temporal, neste caso *H,* ou seja, uma previsão por hora, e finalmente o parâmetro de inclusão das previsões dos dados de treino, ou seja, os valores dos dados de treino ajustados pelo modelo.

Seguidamente foi realizada a previsão utilizando o método *predict* do objeto *model* e passando como parâmetro o objeto *future* criado anteriormente. O resultado foi um *Dataframe*, ao qual se deu o nome de *forecast*, com todos os dados, incluindo os de treino, e uma série de atributos que o modelo *Prophet* disponibiliza a fim de efetuar algumas análises à previsão efetuada. Recorrendo às ferramentas disponibilizadas pelo

modelo foi obtido um gráfico de incerteza das previsões bem como os gráficos dos componentes da previsão, ou seja, tendência, sazonalidade anual e sazonalidade diária.

Finalmente foram retirados do *Dataframe forecast* os dados de treino e as previsões e criados objetos do tipo *panda series* com os mesmos. Esses objetos foram depois sobrepostos aos valores reais num gráfico por forma a permitir visualizar a qualidade do ajuste do modelo aos dados de treino e a qualidade das previsões.

4.1.4. Métricas dos Modelos

Para poder avaliar a eficiência dos modelos de forma numérica foi seguido o procedimento indicado no ponto 2.4. e obtidas as métricas, para todos os modelos desenvolvidos, tanto para previsões de curto prazo como para longo prazo. Dentro de cada modelo, nos casos em que existiam variações devido a transformações ou utilizações de abordagens diferentes, foram também obtidas as respetivas métricas sempre com o objetivo de poder comparar diferentes abordagens em cada modelo.

4.2. Resultados

4.2.1. Modelo *Holt-Winters*

4.2.1.1. Série Temporal de Alcáçova

A aplicação do modelo *Holt-Winters* à série temporal de Alcáçova demonstrou visualmente uma boa adaptação do modelo aos dados de treino, tanto com dados originais como sujeitos a uma transformação logarítmica e também a uma transformação por raiz cúbica.

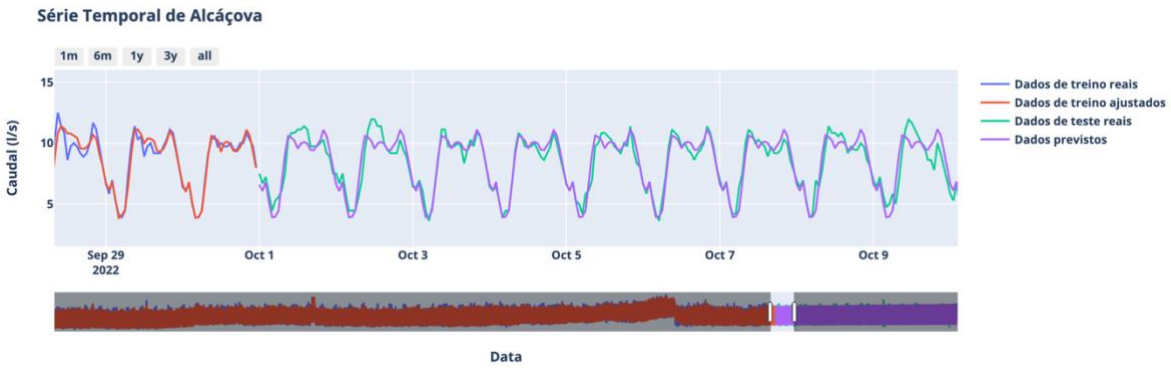
No que respeita às previsões, a tendência nos 3 casos aparentou ser bastante parecida com os dados reais. A amplitude das curvas de previsão mais correta foi a obtida com o modelo com dados transformados logaritmicamente, Figura 49. Com dados originais as curvas apareciam ligeiramente deslocadas na vertical e com os dados transformados por raiz cúbica a amplitude revelava-se um pouco exagerada, Anexo II.

Figura 49 – Previsões do Modelo *Holt-Winters* para Alcáçova com Dados com Transformação Logaritmica

Fonte: Elaboração Própria

Ao ampliarmos o gráfico da Figura 49, na zona da transição entre dados de treino e dados previstos verificámos que as curvas de previsão pareciam relativamente bem ajustadas às curvas reais. Anotou-se, no entanto, que o padrão de repetição se mantinha constante ao longo de todo o período de previsão, Figura 50.

Figura 50 – Ampliação das Previsões do Modelo *Holt-Winters* para Alcáçova com Dados com Transformação Logaritmica



Fonte: Elaboração Própria

No que respeita aos resultados das métricas, para uma previsão de longo prazo, 3 meses, verificou-se que os melhores resultados se obtiveram quando se aplicou o modelo com os dados submetidos a uma transformação logarítmica. Neste caso os resultados evidenciaram um erro médio absoluto de aproximadamente 0.79 l/s, uma raiz do erro quadrático médio aproximado de 1.04 l/s e um coeficiente de determinação aproximado de 0.802, Tabela 3. Isto significa que, o modelo em causa, conseguiu explicar cerca de 80.2% dos dados de teste. Com dados originais e transformados por raiz cúbica os resultados apresentaram-se piores, com maior

significado no último caso onde foi obtido um coeficiente de determinação de apenas 55% aproximadamente.

Para previsões de curto prazo, 10 dias, obtiveram-se resultados substancialmente melhores. As métricas melhoraram nos 3 casos de forma proporcional, mantendo-se o modelo compilado com dados com transformação logarítmica com os melhores resultados, tal como se apresenta na Tabela 3. Neste caso o erro médio absoluto foi de aproximadamente 0.60 l/s, a raiz do erro quadrático médio foi cerca de 1.04 l/s e o coeficiente de determinação foi de 0.867 aproximadamente. Em termos de capacidade de previsão dos dados de teste, este modelo, e para uma previsão de 10 dias, conseguiu explicar quase 87% dos dados.

Tabela 3 – Métricas do Modelo *Holt-Winters* Aplicado à Série de Alcáçova

Tipos de Dados	Dados	MAE (l/s)	RMSE (l/s)	R^2
Originais	Treino	0.47088	0.63232	0.92939
	Teste curto prazo	0.76701	0.97436	0.79662
	Teste longo prazo	1.11398	1.37493	0.65253
Transformação Logarítmica	Treino	0.51715	0.69534	0.91461
	Teste curto prazo	**0.60190**	**0.78912**	**0.86659**
	Teste longo prazo	**0.79151**	**1.03839**	**0.80181**
Transformação por Raiz Cúbica	Treino	0.48845	0.66681	0.92148
	Teste curto prazo	0.90119	1.13563	0.72372
	Teste longo prazo	1.28076	1.56307	0.55093

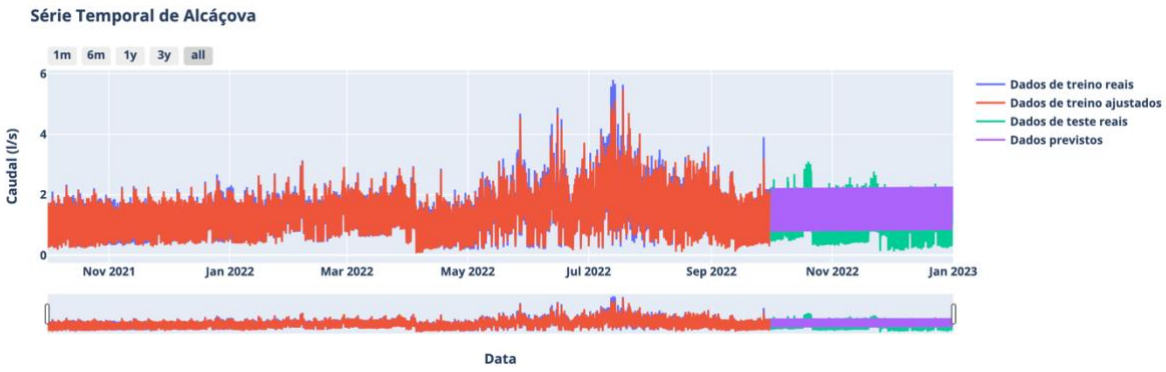
Fonte: Elaboração Própria

4.2.1.2. Série Temporal de Aldeia de Joanes

No caso dos dados da série temporal de Aldeia de Joanes, o modelo em causa apresenta uma boa capacidade de adesão aos dados de treino tanto com os dados originais como com os transformados, tal como ocorreu nos dados de Alcáçova. Não obstante, o

modelo não apresentou quaisquer capacidades de previsão pois os dados previstos afastavam-se bastante dos dados de teste. A utilização de dados com transformação logarítmica ou com transformação por raiz cúbica, Anexo II, não melhorou os resultados em relação à utilização de dados originais, Figura 51.

Figura 51 – Previsões do Modelo *Holt-Winters* para Aldeia de Joanes com Dados Originais



Fonte: Elaboração Própria

A ampliação do gráfico na zona de transição entre o período de treino e o período de teste permitiu-nos perceber que o modelo respeitava a sazonalidade diária de uma forma bastante eficiente, no entanto era incapaz de prever as variações de posicionamento vertical das curvas dos dados de teste, Figura 52. De referir também que se notou um pouco de falta de amplitude nas curvas dos dados previstos pelo modelo.

Figura 52 – Ampliação das Previsões do Modelo *Holt-Winters* para Aldeia de Joanes com Dados Originais

Fonte: Elaboração Própria

Em relação aos resultados das métricas obtidas para uma previsão de longo prazo, todos foram bastante maus e sem variações significativas entre o modelo compilado

com dados originais e com dados transformados. No que respeita aos dados originais foi obtido um erro médio absoluto de cerca de 0.45 l/s, uma raiz do erro quadrático médio de 0.55 l/s aproximadamente e um coeficiente de determinação negativo, Tabela 4. Em todos os casos foi obtido um coeficiente de determinação negativo o que indicou que o modelo em causa não era apropriado para descrever os dados de teste. Também foi considerado que poderia haver problemas nos dados durante o período de teste, nomeadamente que tivessem sido feitas alterações à rede de abastecimento relacionadas com a ocorrência de roturas, o que explicaria as variações verticais nas curvas, no entanto, não foi possível averiguar com certeza essa possibilidade.

Nas métricas dos resultados de curto prazo, os valores obtidos foram algo melhores, ainda assim insuficientes. Os melhores resultados foram obtidos para o modelo com dados com transformação logarítmica com um erro médio absoluto de 0.30 l/s, uma raiz do erro quadrático médio de cerca de 0.38 l/s e um coeficiente de determinação de cerca de 0.40, dados aproximados, Tabela 4. Embora o facto de se ter obtido um coeficiente de determinação positivo fosse encorajador, o seu valor inferior a 0.5 indicou que o modelo não conseguia explicar nem metade dos dados de teste mesmo para um período de curto prazo o que foi considerado claramente insuficiente.

Tabela 4 – Métricas do Modelo *Holt-Winters* Aplicado à Série de Aldeia de Joanes

Tipos de Dados	Dados	MAE (l/s)	RMSE (l/s)	R^2
Originais	Treino	0.17687	0.26747	0.82784
	Teste curto prazo	0.32835	0.38617	0.39334
	Teste longo prazo	0.45353	0.54847	-0.04222
Transformação Logarítmica	Treino	0.18372	0.28551	0.80383
	Teste curto prazo	**0.30528**	**0.38290**	**0.40358**
	Teste longo prazo	0.44172	0.56156	-0.09256
Transformação por Raiz Cúbica	Treino	0.17626	0.27097	0.82330
	Teste curto prazo	0.34197	0.41465	0.30058
	Teste longo prazo	0.49119	0.60625	-0.27335

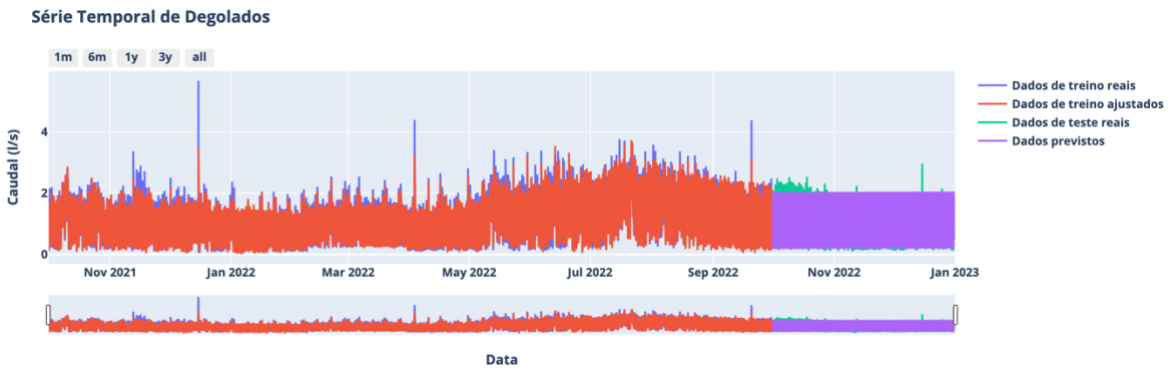
Fonte: Elaboração Própria

4.2.1.3. Série Temporal de Degolados

O modelo de previsão compilado para a séria temporal de Degolados evidenciou uma boa capacidade de ajuste aos dados de treino, tal como visto em séries anteriores. Em relação às previsões o modelo apresentou curvas de amplitude constante e sem tendência crescente ou decrescente. Para os primeiros dias a amplitude ficou um pouco aquém da realidade, mas a partir do final do mês de outubro verificou-se um ajuste bastante bom. Neste caso, as transformações aplicadas aos dados não aumentaram a eficiência do modelo, Anexo II, tendo sido obtidos os melhores resultados com dados originais.

Na Figura 53, pode observar-se a boa adaptação aos dados de treino, mas para as previsões o modelo estabeleceu um valor de amplitude fixo e uma tendência nula e aplicou-o a todo o período de teste.

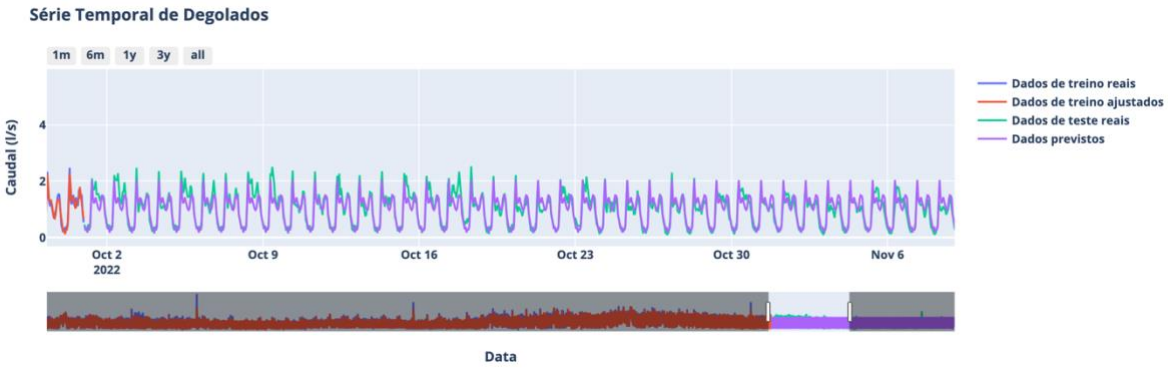
Figura 53 – Previsões do Modelo *Holt-Winters* para Degolados com Dados Originais



Fonte: Elaboração Própria

A ampliação do gráfico na zona de transição entre treino e teste mostrou que o modelo tinha uma boa capacidade de ajustar a sazonalidade prevista, não se tendo verificado nenhum desfasamento entre as curvas de dados de teste e as curvas de dados previstos. Verificou-se sim falta de amplitude das curvas de previsão até ao final de outubro, data a partir da qual se verificou uma melhoria nesse ajustamento, Figura 54.

Figura 54 – Ampliação das Previsões do Modelo *Holt-Winters* para Degolados com Dados Originais



Fonte: Elaboração Própria

Os resultados das métricas obtidas para o modelo compilado com dados originais e para um período de longo prazo foram de 0.21 l/s no que respeita ao erro médio absoluto, 0.29 l/s para a raiz do erro quadrático médio e 0.687 para o coeficiente de determinação, valores aproximados, Tabela 5. Concluiu-se assim que este modelo conseguia explicar cerca de 69% dos dados de teste. Foi verificado ainda que, em termos médios, durante o período dos dados de teste estes tinham uma média de 0.932 l/s e que os dados previstos revelavam uma média de 0.936 l/s.

Para um período de curto prazo os resultados melhoraram significativamente, o erro médio absoluto obtido foi de 0.19 l/s, a raiz do erro quadrático médio de 0.28 l/s e o coeficiente de determinação de 0.79, valores aproximados, Tabela 5. Neste caso, para um período de curso prazo, o modelo conseguiu explicar cerca de 79% dos dados de teste.

Tabela 5 – Métricas do Modelo *Holt-Winters* Aplicado à Série de Degolados

Tipos de Dados	Dados	MAE (l/s)	RMSE (l/s)	R^2
Originais	Treino	0.15895	0.23667	0.86788
	Teste curto prazo	**0.19460**	**0.28224**	**0.79126**
	Teste longo prazo	**0.21077**	**0.28865**	**0.68783**
Transformação Logarítmica	Treino	0.15543	0.23636	0.86823
	Teste curto prazo	0.22681	0.33400	0.70767
	Teste longo prazo	0.21241	0.29512	0.67370
Transformação por Raiz Cúbica	Treino	0.15697	0.23994	0.86421
	Teste curto prazo	0.26849	0.38981	0.60182
	Teste longo prazo	0.22555	0.31646	0.62481

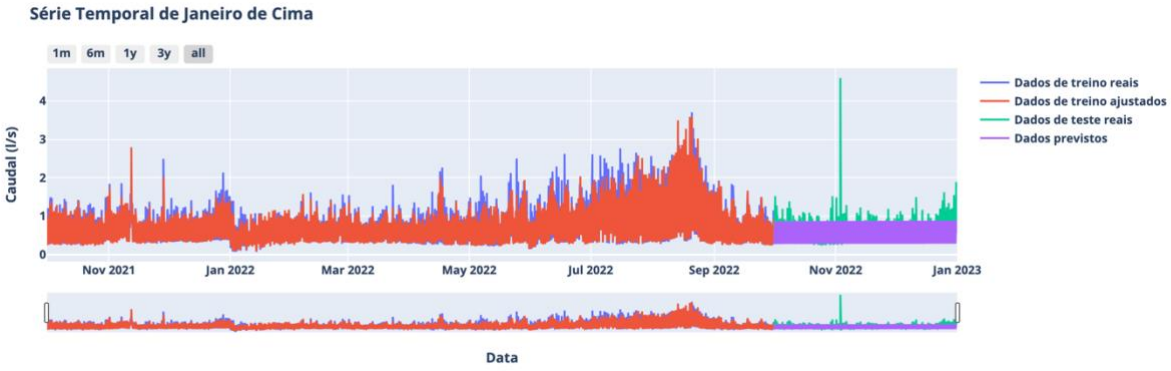
Fonte: Elaboração Própria

4.2.1.4. Série Temporal de Janeiro de Cima

No caso da série temporal de Janeiro de Cima e tal como sucedido nas séries anteriores, o modelo apresentou um bom ajuste no que respeita aos dados de treino. Em relação às previsões, o ajuste das mesmas em relação aos dados de teste apresentou alguma falta de amplitude, principalmente porque existiam alguns períodos com picos no gráfico de dados de teste que as previsões não conseguiram suportar. O modelo apresentou curvas com uma amplitude constante e sem tendência. Essa amplitude constante ao longo de todo o período de teste não conseguiu suprimir os muitos picos

de amplitude dos dados de teste. Verificou-se que o melhor modelo obtido foi o compilado com os dados submetidos a uma transformação logarítmica, Figura 55.

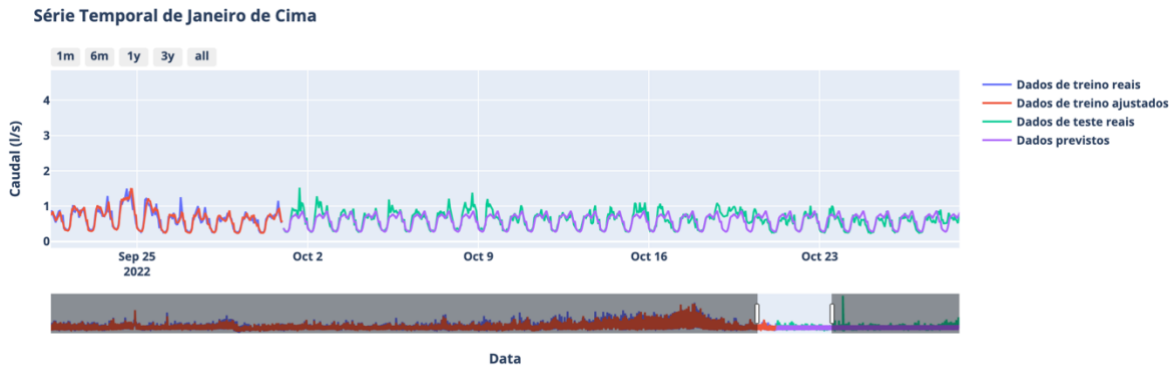
Figura 55 – Previsões do Modelo *Holt-Winters* para Janeiro de Cima com Dados com Transformação Logarítmica



Fonte: Elaboração Própria

A ampliação do gráfico na zona de transição entre o período de treino e o período de teste revelou que o modelo respeitava bastante bem a sazonalidade diária apresentando as curvas de previsão sincronizadas com as curvas de teste, Figura 56. O problema situou-se sim na falta de amplitude apresentada pela curva de previsão e no facto de os dados de teste apresentarem algumas variações aleatórias que o modelo não conseguiu suportar.

Figura 56 – Ampliação das Previsões do Modelo *Holt-Winters* para Janeiro de Cima com Dados com Transformação Logarítmica



Fonte: Elaboração Própria

No que respeita aos resultados numéricos, para um período de longo prazo, os melhores valores das métricas obtidos foram os do modelo com dados sujeitos a uma transformação logarítmica com um erro médio absoluto de 0.12 l/s, uma raiz do erro

quadrático médio de 0.20 l/s e um coeficiente de determinação de 0.46, valores aproximados, Tabela 6. Estes resultados permitiram concluir que, o modelo obtido com dados sujeitos a uma transformação logarítmica, permitia explicar cerca de 46% dos dados de teste, o que é manifestamente insuficiente pois nem metade dos dados de teste consegue explicar. Em termos médios, foi verificado que os dados de teste apresentavam um valor de 0.64 l/s em contraste com os 0.60 l/s dos dados previstos, valores aproximados.

Tabela 6 – Métricas do Modelo *Holt-Winters* Aplicado à Série de Janeiro de Cima

Tipos de Dados	Dados	MAE (l/s)	RMSE (l/s)	R^2
Originais	Treino	0.10862	0.17020	0.83016
	Teste curto prazo	0.11293	0.16342	0.61096
	Teste longo prazo	0.12529	0.21020	0.41555
Transformação Logarítmica	Treino	0.10939	0.17389	0.82271
	Teste curto prazo	**0.10199**	**0.15238**	**0.66175**
	Teste longo prazo	**0.11827**	**0.20268**	**0.45659**
Transformação por Raiz Cúbica	Treino	0.10873	0.17102	0.82852
	Teste curto prazo	0.10882	0.15953	0.62924
	Teste longo prazo	0.12259	0.20768	0.42944

Fonte: Elaboração Própria

Em relação aos resultados das métricas para um período de curto prazo, estes apresentaram melhorias significativas, mantendo-se o modelo com dados com transformação logarítmica com os melhores resultados. O erro médio absoluto foi de 0.10 l/s, a raiz do erro quadrático médio foi de 0.15 l/s e o coeficiente de determinação foi de 0.66, valores aproximados, Tabela 6. Neste caso o modelo consegue explicar cerca de 66% dos dados de teste, o que representa um valor muito mais aceitável do que o conseguido no longo prazo.

4.2.2. Modelo ARIMA

4.2.2.1. Série Temporal de Alcáçova

Na implementação do modelo ARIMA à série temporal de Alcáçova verificou-se, na generalidade, uma boa adaptação do modelo aos dados tanto com a utilização do método de parametrização manual como com a parametrização automática.

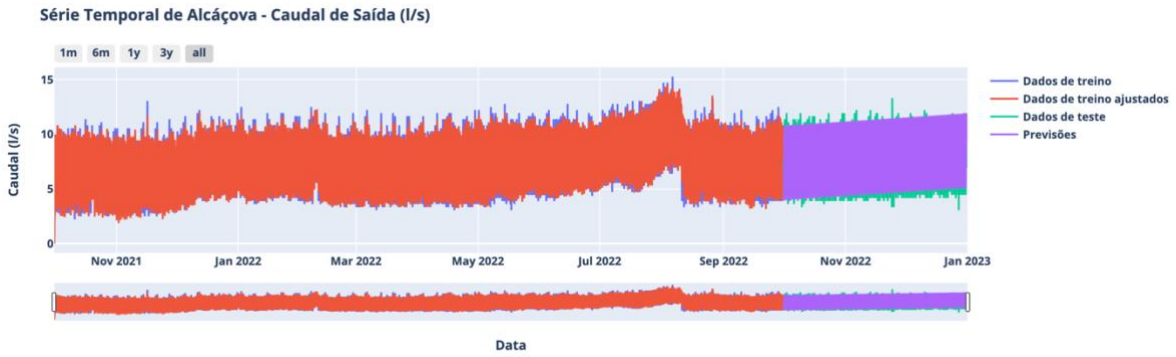
Para a parametrização manual, e após várias tentativas aumentando a ordem dos componentes não sazonais e sazonais, obteve-se o modelo com o AIC mais baixo representado por ARIMA(3, 2, 2)(1, 1, 1)[24]. O AIC obtido para este modelo foi de 14770.95. A análise dos resíduos no que respeita à normalidade, e segundo os testes numéricos, revelou que estes não seguiam uma distribuição normal. Ainda que o histograma de distribuição, Anexo II, visualmente, tivesse uma aparência tipicamente semelhante a uma distribuição normal, o gráfico da *Normal QQ plot*, Anexo II, evidenciou a existência de problemas nas extremidades. Através dos diagramas das funções de autocorrelação e autocorrelação parcial, Anexo II, foi ainda possível verificar que os resíduos não tinham problemas de autocorrelação significativos.

No método de parametrização automático, com a utilização do método *auto_arima*, observou-se que com o parâmetro *stationary=False* se obteve um modelo com um AIC superior àquele que se obteve com o parâmetro contrário. O modelo obtido considerando não estacionaridade foi ARIMA(1, 1, 1)(1, 0, 0)[24] com um AIC de 18654 e considerando estacionaridade foi obtido o modelo ARIMA(0, 0, 2)(1, 0, 1)[24] com um AIC de 15641. Este último modelo é diferente do modelo obtido com a parametrização manual, é menos complexo, mas possui um AIC superior. Concluiu-se assim que o melhor modelo obtido pelo método automático resultou da consideração de que a série era estacionária e, portanto, não tinha necessidade de diferenciação, o que era um resultado contraditório em relação ao modelo parametrizado manualmente com uma diferenciação de segunda ordem. Em relação aos resíduos, os testes numéricos de normalidade revelaram que estes não seguiam uma distribuição normal. Graficamente o histograma da distribuição, Anexo II, assemelhava-se bastante a uma distribuição normal, mas o gráfico da *Normal QQ plot*, Anexo II, evidenciou os mesmos problemas nas extremidades, tal como no caso dos resíduos obtidos pela

parametrização manual. No que respeita à autocorrelação, segundo diagrama de autocorrelação, verificou-se que estes resíduos estavam autocorrelacionados, ainda que o diagrama de autocorrelação parcial revelasse pouca autocorrelação para as lags não adjacentes, Anexo II. Concluiu-se então que o modelo obtido pelo método automático gerava resíduos mais autocorrelacionados do que o modelo obtido pelo método manual.

Passando aos resultados das previsões do modelo obtido pelo método manual, a representação gráfica dos dados de treino e dos dados de teste previstos sobrepostos à série original mostrou uma boa adaptação dos dados de treino ajustados e uma previsão dos dados de teste com uma boa amplitude, embora com uma tendência crescente algo exagerada, Figura 57.

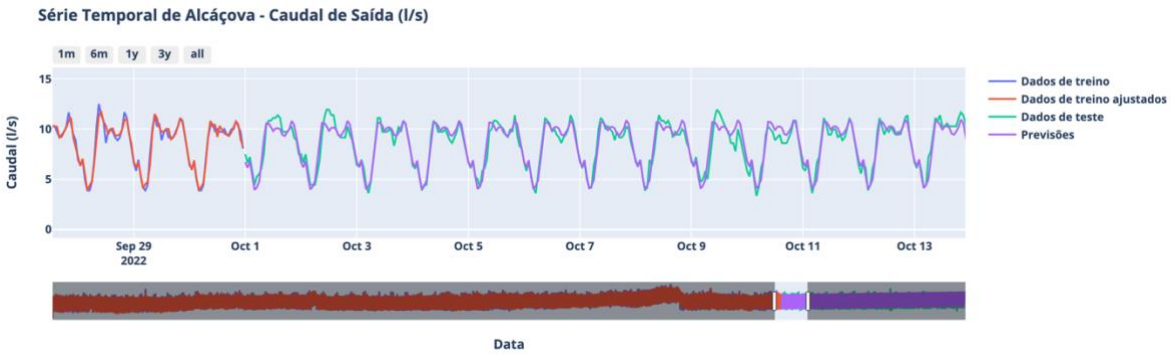
Figura 57 – Previsões do Modelo ARIMA para Alcáçova – Método Manual



Fonte: Elaboração Própria

Ampliando a zona de transição entre os períodos de treino e de teste, verificou-se um ajuste muito bom das curvas de dados de teste em comparação com as curvas das previsões fazendo referência ao correto cumprimento da sazonalidade diária sem evidenciar qualquer desfasamento entre dados de teste e dados previstos, Figura 58. Quando foi verificado o ajuste ao longo do tempo, verificou-se que a partir do início do mês de novembro começava a existir um ligeiro desfasamento entre as curvas e um ligeiro deslocamento vertical, corrompendo assim o período da sazonalidade e os valores máximos e mínimos das curvas de previsão, afetando os resultados das previsões nos últimos dois meses.

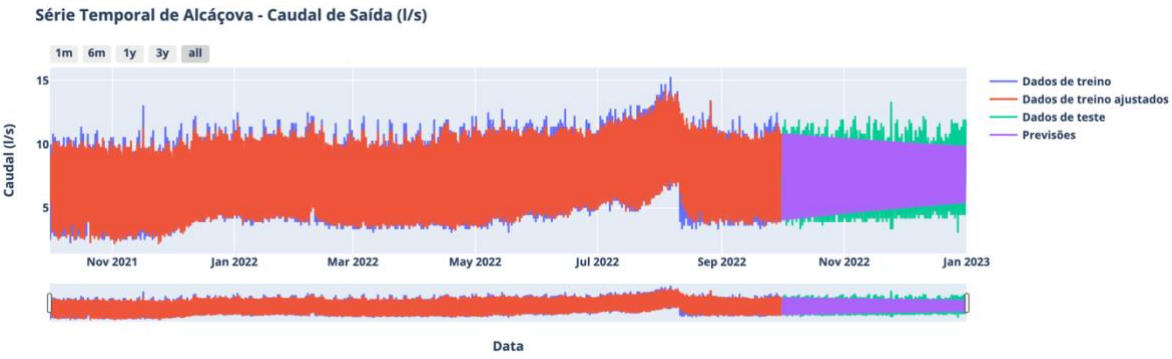
Figura 58 – Ampliação das Previsões do Modelo ARIMA para Alcáçova – Método Manual



Fonte: Elaboração Própria

No caso do modelo obtido pelo método automático, os dados de treino ajustados pelo modelo evidenciaram igualmente uma boa adaptação quando sobrepostos aos dados de treino originais. No caso das previsões, por outro lado, o modelo pareceu não conseguir manter a amplitude ao longo do tempo tendo um início correto, mas um final com muito pouca amplitude quando comparado com o gráfico de dados de teste originais, Figura 59.

Figura 59 – Previsões do Modelo ARIMA para Alcáçova – Método Automático

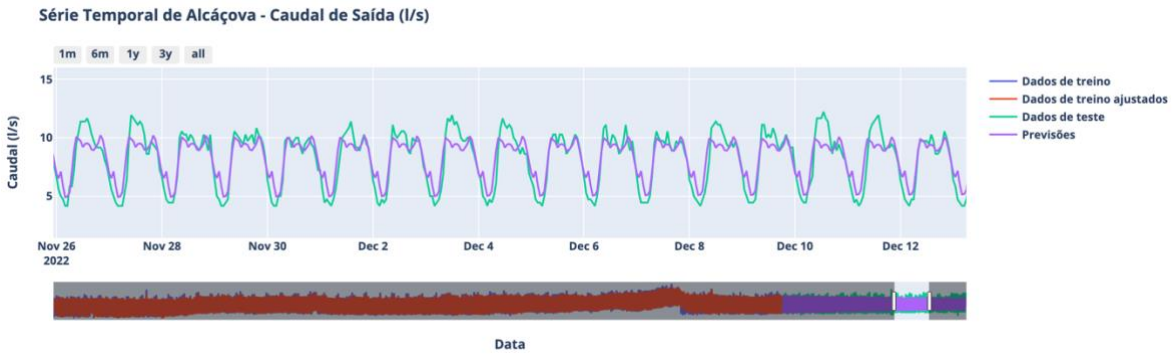


Fonte: Elaboração Própria

Efetuando a verificação ampliada no gráfico de previsões do modelo do método automático verificou-se um ajuste ainda mais rigoroso na zona de transição. Na análise ao longo do tempo não foi verificado qualquer desfasamento entre as duas curvas como existia no método manual, o que foi verificado foi uma redução progressiva da amplitude das curvas de previsão em comparação com as curvas de dados de teste a partir do final de outubro, embora que, com a ampliação, não ficava evidente um erro tão grosseiro como no gráfico não ampliado. Ainda assim, o modelo não conseguiu

prever os máximos e os mínimos das curvas originais sendo este um fator que, claramente piorava a sua performance, Figura 60.

Figura 60 – Ampliação das Previsões do Modelo ARIMA para Alcáçova – Método Automático



Fonte: Elaboração Própria

Reunindo os resultados numéricos através da obtenção das métricas de ambos os modelos, verificou-se que o melhor modelo dependia do intervalo da previsão, ou seja, uma previsão de curto prazo era melhor explicada pelo modelo obtido pelo método manual, com um erro médio absoluto de 0.55 l/s, a raiz do erro quadrático médio de 0.72 l/s e um coeficiente de determinação de 0.89, valores aproximados, que lhe conferia uma capacidade de explicar os dados de teste de cerca de 89%. Não obstante, uma previsão mais alargada, de período igual ao dos dados de teste, seria melhor representada pelo modelo obtido pelo método automático, com um erro médio absoluto de 0.77 l/s, uma raiz do erro quadrático médio de 1.01 l/s e um coeficiente de determinação de 0.81, valores aproximados. Isto significava que este modelo conseguia explicar cerca de 81% dos dados de teste, Tabela 7.

Tabela 7 – Métricas do Modelo ARIMA Aplicado à Série de Alcáçova

Parametrização do Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Manual	Treino	0.42373	0.56906	0.94281
	Teste curto prazo	**0.55562**	**0.72449**	**0.88755**
	Teste longo prazo	0.95888	1.20641	0.73248
Automática	Treino	0.44654	0.59351	0.93779
	Teste curto prazo	0.55293	0.74435	0.88131
	Teste longo prazo	**0.76914**	**1.00876**	**0.81296**

Fonte: Elaboração Própria

Anotou-se o aspeto curioso no qual, segundo a representação gráfica, as previsões do modelo obtido pelo método manual parecia ter menor performance para o período de longo prazo quando na verdade os resultados numéricos das métricas obtidas, acabaram por revelar precisamente o oposto. Mesmo no caso das previsões de curto prazo, o outro modelo parametrizado manualmente tem melhores métricas, mas não de forma significativa, para além de ser muito mais complexo.

4.2.2.2. Série Temporal de Aldeia de Joanes

A aplicação do modelo ARIMA à série temporal de Aldeia de Joanes mostrou-se difícil, tal como já havia ocorrido no modelo *Holt-Winters*. O desenvolvimento da série com períodos com comportamentos aleatórios dificultou a tarefa de parametrização do modelo e acabou por se traduzir em resultados bastante redutores.

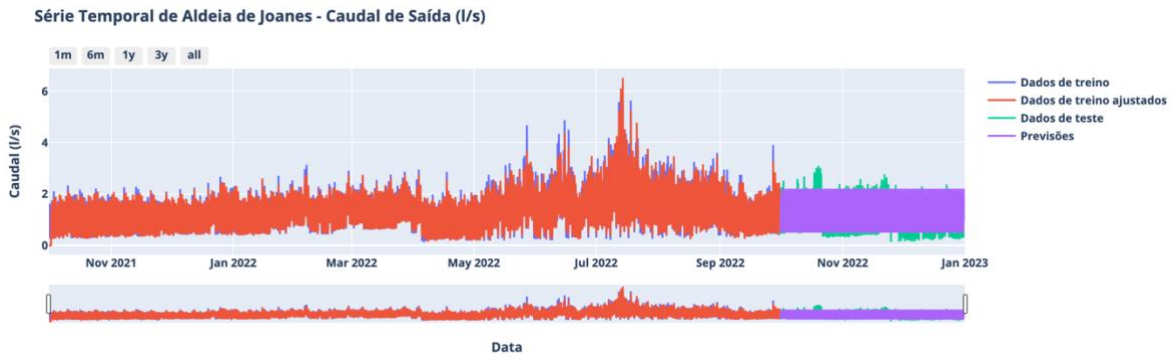
Não obstante, seguindo a metodologia explicada em 4.1.3.2., após várias tentativas para a parametrização manual com o objetivo de minimizar o AIC, obteve-se o modelo com um AIC de 1508 representado por: ARIMA(1, 0, 1)(2, 1, 0)[24]. Sobre a normalidade dos resíduos gerados por este modelo, foi verificado que os mesmos não seguiam uma distribuição normal, quer pelos testes numéricos quer pelo resultado do gráfico *Normal QQ plot*, Anexo II. Este último evidenciou problemas com os dados de resíduos das extremidades. Em relação à existência ou não de autocorrelação nos resíduos,

verificou-se, pelos diagramas de autocorrelação e autocorrelação parcial, que os dados dos resíduos estavam pouco autocorrelacionados, Anexo II.

Passando para o método automático e com o parâmetro *stationary=False* foi possível obter um modelo com um AIC de 2429 e com o parâmetro contrário obteve-se um modelo com um AIC de 2379. Assim o modelo conseguido considerando que os dados eram estacionários era teoricamente melhor. Este modelo era representado por ARIMA(1, 0, 1)(1, 0, 0)[24] e era menos complexo do que aquele obtido pelo método manual. Na análise da normalidade dos resíduos deste modelo verificou-se que os testes numéricos classificaram a sua distribuição como não seguindo uma distribuição normal bem como as verificações gráficas realizadas com o gráfico da *Normal QQ plot* e com o histograma da distribuição dos dados dos resíduos, Anexo II. Finalmente, em relação à autocorrelação e autocorrelação parcial foi verificado que existia alguma dependência entre *lags* adjacentes e não adjacentes, Anexo II. Pôde assim concluir-se que o modelo obtido pelo método automático gerava resíduos com mais problemas de autocorrelação do que o modelo obtido pelo método manual.

Em relação aos resultados das previsões do modelo do método manual, a representação gráfica dos dados de treino ajustados e dos dados de teste previstos sobrepostos à série original mostrou boa adaptação dos dados de treino ajustados, mas uma previsão dos dados de teste incapaz de lidar com as variações que existiam na realidade. A amplitude das curvas de previsão foi considerada correta apenas deslocada, muitas vezes, verticalmente em relação à realidade. A tendência apresentada era ligeiramente decrescente, mas dada a aleatoriedade dos dados de teste foi difícil concluir se esta era correta ou não, Figura 61.

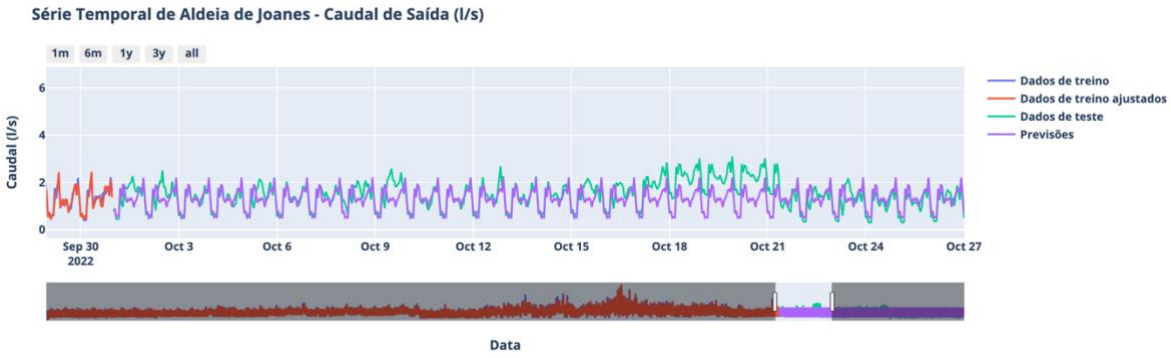
Figura 61 – Previsões do Modelo ARIMA para Aldeia de Joanes – Método Manual



Fonte: Elaboração Própria

Na ampliação do gráfico de previsão na zona de transição do período de treino para o período de teste foi possível verificar que o modelo consegue cumprir a sazonalidade diária e que o problema se prende mesmo com a dificuldade em lidar com a aleatoriedade das deslocações verticais das curvas de dados reais, Figura 62. Foi anotado que estas deslocações repentinas podiam resultar de roturas nas redes de abastecimento que podiam efetivamente produzir aquelas deslocações, não se tratando, portanto, de erros na recolha dos dados.

Figura 62 – Ampliação das Previsões do Modelo ARIMA para Aldeia de Joanes – Método Manual

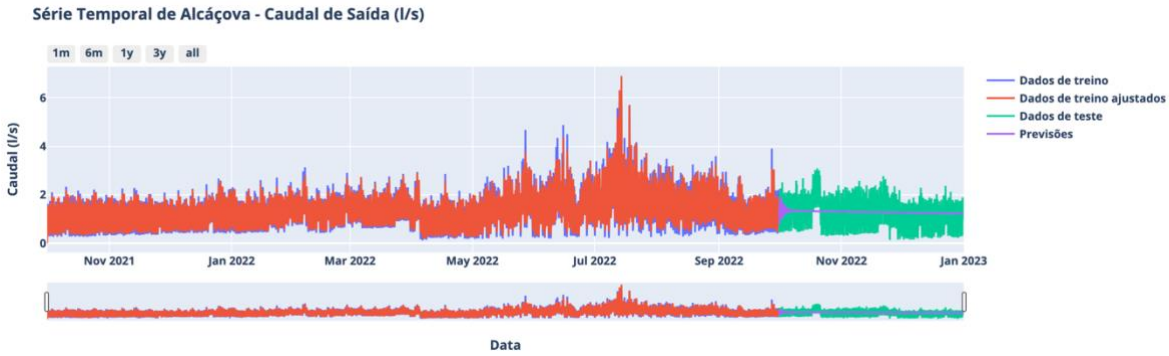


Fonte: Elaboração Própria

No caso do modelo obtido através do método automático, a representação gráfica que sobrepôs as previsões aos dados reais, mostrou que para os dados de treino ajustados a adaptação aos dados reais foi bastante boa, o mesmo não de pôde dizer em relação à sobreposição das previsões do período de teste aos dados de teste reais, ou seja, às previsões propriamente ditas. As previsões deste modelo obtido pelo método automático traduziram-se basicamente numa linha contínua ao longo do período de

teste sem capacidade de respeitar a sazonalidade, ou seja, verificou-se a incapacidade do modelo para efetuar previsões fora do período de treino, Figura 63.

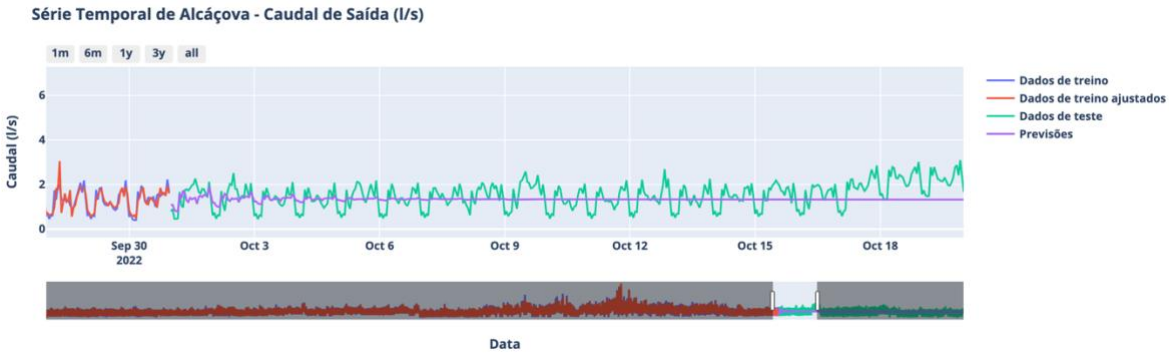
Figura 63 – Previsões do Modelo ARIMA para Aldeia de Joanes – Método Automático



Fonte: Elaboração Própria

Ao ampliar a representação gráfica na zona de transição entre os períodos de treino e teste verificou-se que durante os primeiros três dias de previsão o modelo ainda aparentava tentar respeitar a sazonalidade, mas depois as curvas convergiam para uma linha continua e assim se mantinham até ao final do período de previsão, Figura 64.

Figura 64 – Ampliação das Previsões do Modelo ARIMA para Aldeia de Joanes – Método Automático



Fonte: Elaboração Própria

Neste caso, ao obter resultados tão maus, principalmente no modelo do método automático, foram feitas tentativas de melhorar o mesmo indo um pouco além da metodologia indicada alterando os parâmetros da função *auto_arima* e não foram obtidos resultados diferentes.

Em relação aos resultados das métricas obtidas para os dois modelos verificou-se, como seria de esperar, tendo em conta a análise das representações gráficas, que os

melhores resultados foram os do modelo obtido pelo método manual qualquer que fosse o prazo da previsão. Assim, para uma previsão de curto prazo foi obtido um erro médio absoluto de 0.24 l/s, uma raiz do erro quadrático médio de 0.37 l/s e um coeficiente de determinação de 0.45, valores aproximados. Para uma previsão de longo prazo, o erro médio absoluto obtido foi de 0.33 l/s, a raiz do erro quadrático médio de 0.45 l/s e o coeficiente de determinação foi de 0.29, valores aproximados, Tabela 8.

Tabela 8 – Métricas do Modelo ARIMA Aplicado à Série de Aldeia de Joanes

Parametrização do Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Manual	Treino	0.17647	0.26998	0.82460
	Teste curto prazo	**0.24250**	**0.36735**	**0.45106**
	Teste longo prazo	**0.32898**	**0.45211**	**0.29183**
Automática	Treino	0.18817	0.27802	0.81400
	Teste curto prazo	0.37675	0.45951	0.14106
	Teste longo prazo	0.42293	0.52632	0.04026

Fonte: Elaboração Própria

Os resultados dos coeficientes de determinação obtidos significavam que o modelo com parametrização manual conseguia explicar cerca de 45% dos dados de teste num período de 10 dias, mas apenas cerca de 29% num período de 3 meses. Qualquer um destes valores foi considerado muito baixo e, portanto, considerou-se que o modelo não tinha capacidade de efetuar previsões.

4.2.2.3. Série Temporal de Degolados

No que respeita à implementação do modelo ARIMA à série temporal de Degolados obteve-se uma adaptação razoável com o modelo obtido pelo método manual e uma adaptação medíocre no modelo obtido pelo método automático.

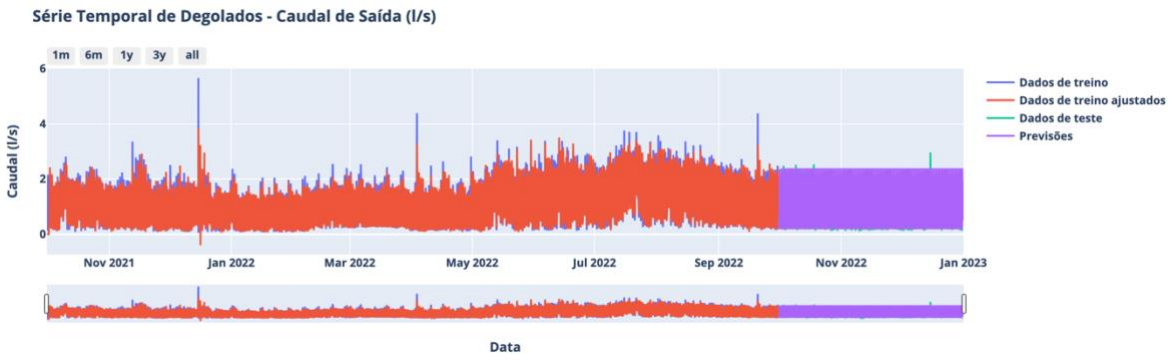
Na parametrização manual, depois de várias tentativas com o objetivo de obter o menor AIC, foi obtido um modelo representado por ARIMA(1, 0, 1)(2, 1, 0)[24]. O AIC

obtido para este modelo foi de 236. A análise da normalidade dos resíduos deste modelo resultou em que a sua distribuição não seguia uma distribuição normal, segundo os testes numéricos e a análise gráfica. Mais uma vez o gráfico da *Normal QQ plot*, Anexo II, evidenciou problemas de normalidade nos dados das extremidades. Em relação à autocorrelação, os diagramas de autocorrelação e autocorrelação parcial revelaram a existência de uma dependência muito baixa entre as *lags* adjacentes e não adjacentes, Anexo II.

No método de parametrização automático, com a utilização do método *auto_arima*, observou-se que com o parâmetro *stationary=False* se obteve um modelo com um AIC superior àquele que se obteve com o parâmetro contrário. O modelo obtido considerando não estacionaridade foi ARIMA(3,1, 0)(1, 0, 0)[24] com um AIC de 2844 e considerando estacionaridade foi obtido o modelo ARIMA(2, 0, 0)(1, 0, 0)[24] com um AIC de 1806. Este último modelo é diferente do modelo obtido com a parametrização manual, é menos complexo, mas possui um AIC superior. Em relação aos resíduos, os testes numéricos de normalidade revelaram que estes não seguiam uma distribuição normal. Graficamente o histograma da distribuição assemelhava-se bastante a uma distribuição normal, mas o gráfico da *Normal QQ plot* evidenciou os mesmos problemas nas extremidades, tal como no caso dos resíduos obtidos pela parametrização manual, Anexo II. No que respeita à autocorrelação, segundo o diagramas de autocorrelação e autocorrelação parcial, verificou-se que os resíduos estavam autocorrelacionados, tanto para as lags adjacentes como para as não adjacentes, Anexo II.

Passando aos resultados das previsões do modelo obtido pelo método manual, a representação gráfica dos dados de treino e dos dados de teste previstos, sobrepostos à série original mostrou uma boa adaptação dos dados de treino ajustados, mas uma previsão dos dados de teste com uma amplitude algo exagerada. A tendência ligeiramente decrescente afigurou-se como correta e de acordo com os dados de teste reais, Figura 65.

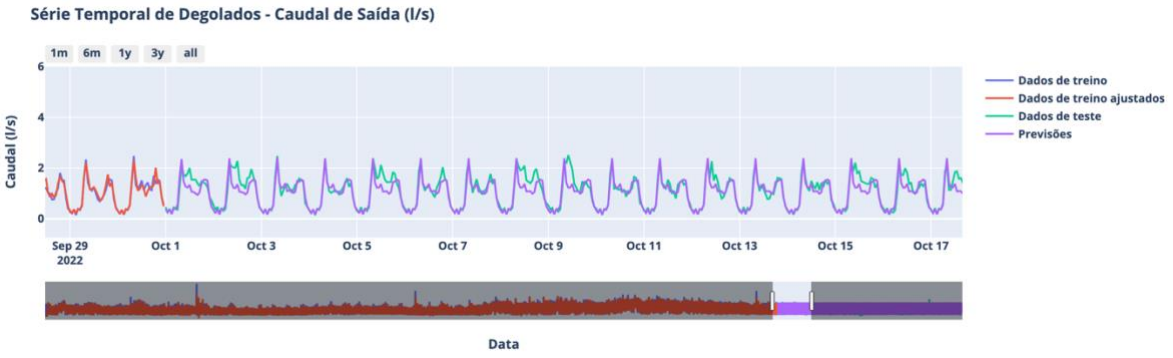
Figura 65 – Previsões do Modelo ARIMA para Degolados – Método Manual



Fonte: Elaboração Própria

A ampliação da zona de transição entre os períodos de treino e de teste, permitiu confirmar que a amplitude das curvas de previsão era, por vezes, um pouco exagerada. Esta ampliação permitiu, no entanto, verificar a forma eficiente como o modelo implementava a previsão de sazonalidade sem qualquer desfasamento entre as curvas de teste e as curvas de previsão, Figura 66.

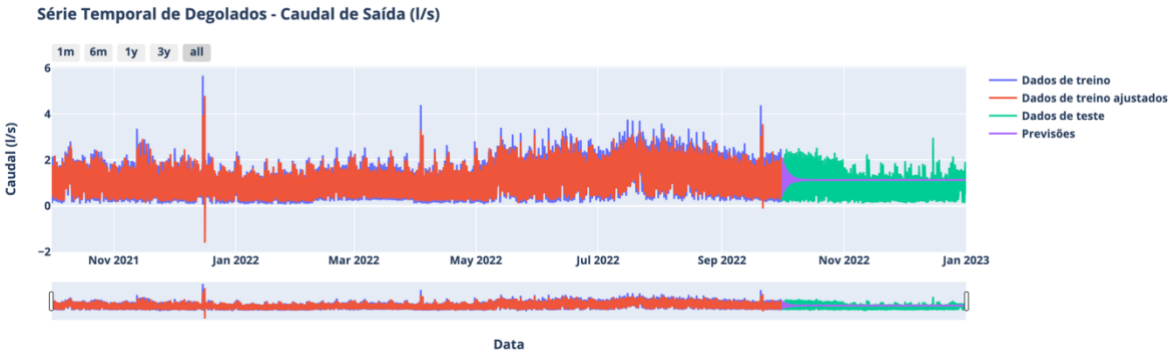
Figura 66 – Ampliação das Previsões do Modelo ARIMA para Degolados – Método Manual



Fonte: Elaboração Própria

Em relação aos resultados das previsões do modelo obtido pelo método automático, os dados de treino ajustados pelo modelo evidenciaram igualmente uma boa adaptação quando sobrepostos aos dados de treino originais. No caso das previsões, por outro lado, o modelo não se mostrou minimamente capaz de efetuar qualquer tipo de previsão. A representação gráfica das curvas de previsão convergia rapidamente, logo nos primeiros dias, para uma reta e assim se mantinha durante o período restante, Figura 67.

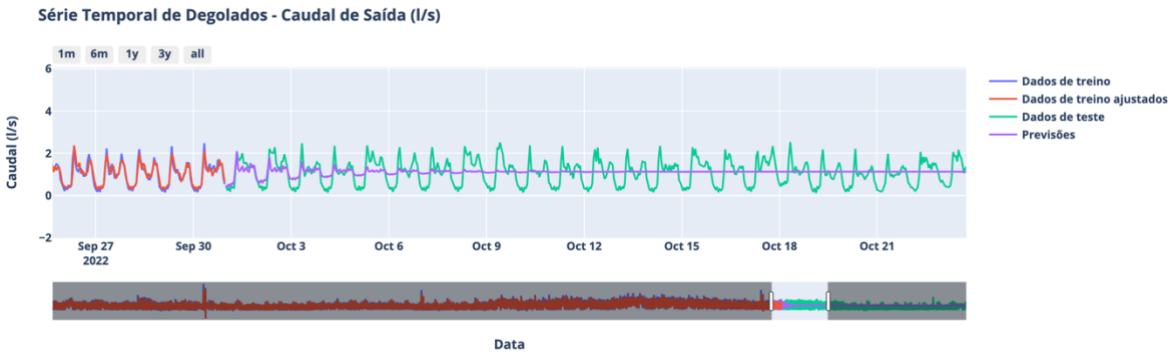
Figura 67 – Previsões do Modelo ARIMA para Degolados – Método Automático



Fonte: Elaboração Própria

Efetuando uma ampliação da representação gráfica na zona de transição entre os períodos de treino e de teste, observou-se que durante os primeiros 4 dias o modelo ainda aparentava seguir a sazonalidade diária, mas a partir do quinto dia as curvas convergiam para uma reta que se manteve até ao final do período de previsão, Figura 68.

Figura 68 – Ampliação das Previsões do Modelo ARIMA para Degolados – Método Automático



Fonte: Elaboração Própria

Relativamente aos resultados das métricas obtidos, os melhores resultados foram os correspondentes ao modelo obtido pelo método manual, tendo sido, as métricas do modelo obtido pelo método automático, consideradas medíocres. Assim, e para uma previsão de curto prazo, os melhores resultados traduziram-se num erro médio absoluto de 0.19 l/s, a raiz do erro quadrático médio de 0.29 l/s e um coeficiente de determinação de 0.78, valores aproximados. No caso de uma previsão de longo prazo, o erro médio absoluto foi de 0.24 l/s, a raiz do erro quadrático médio foi de 0.34 l/s e o coeficiente de determinação foi 0.55, valores aproximados, Tabela 9.

Tabela 9 – Métricas do Modelo ARIMA Aplicado à Série de Degolados

Parametrização do Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Manual	Treino	0.16433	0.25226	0.84991
	Teste curto prazo	**0.19191**	**0.29071**	**0.77854**
	Teste longo prazo	**0.23672**	**0.34499**	**0.55410**
Automática	Treino	0.18367	0.26824	0.83028
	Teste curto prazo	0.41849	0.51206	0.31293
	Teste longo prazo	0.42996	0.54049	-0.09445

Fonte: Elaboração Própria

Pelos resultados das métricas obtidos para o modelo com a parametrização manual e no que respeita especificamente aos resultados dos coeficientes de determinação, concluiu-se que, numa previsão de curto prazo, o modelo conseguia explicar cerca de 78% dos dados de teste, e numa previsão de longo prazo, o modelo perdia bastante performance, mas ainda assim, conseguia explicar cerca de 55% dos dados de teste.

4.2.2.4. Série Temporal de Janeiro de Cima

A aplicação do modelo ARIMA à série temporal de Janeiro de Cima revelou-se algo complicada e, embora tenha sido realizado um esforço de realização de testes no método manual com a alteração sistemática de parâmetros, os resultados que foram alcançados acabaram por ficar longe do esperado.

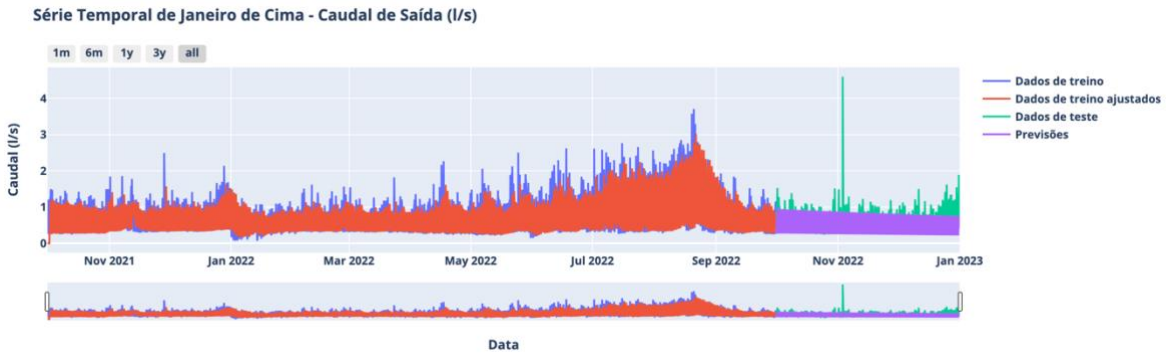
Assim, após várias tentativas de minimizar o AIC, mas também de obter o melhor resultado do coeficiente de determinação, chegou-se ao modelo representado por ARIMA(0, 0, 0)(3, 0, 1)[24] com um AIC de -2435. A análise da normalidade dos resíduos, verificada graficamente e com testes numéricos, deste modelo revelou que a distribuição dos mesmos não segue uma distribuição normal. Neste caso, o gráfico da *Normal QQ plot*, Anexo II, revelou problemas de afastamento em relação à normal, não só nas extremidades da distribuição, mas sim na sua totalidade. Os diagramas de

autocorrelação evidenciaram a existência de uma forte autocorrelação entre as *lags* adjacentes, mas uma fraca autocorrelação para as *lags* não adjacentes, Anexo II.

No método automático verificou-se que o modelo obtido com o parâmetro *stationary=False*, considerando que a série era não estacionária, era ligeiramente melhor do que o seu oposto, ambos com valores de AIC muito parecidos e traduzindo-se em coeficientes de determinação extremamente baixos, a rondar o valor nulo. O valor do AIC conseguido foi de -4602 e o modelo ficou caracterizado por ARIMA(3, 1, 2)(1, 0, 0)[24]. A análise dos resíduos deste modelo, em relação à sua normalidade, revelou que a distribuição dos mesmos não seguia uma distribuição normal, de acordo com os testes numéricos e com as representações gráficas, Anexo II. Os diagramas de autocorrelação dos resíduos mostraram a existência de autocorrelação tanto nas *lags* adjacentes como nas não adjacentes, embora no caso das adjacentes se tenha verificado que era inferior àquela obtida com a análise de autocorrelação realizada aos resíduos gerados no modelo do método manual, Anexo II.

Em relação aos resultados das previsões do modelo do método manual, a representação gráfica dos dados de treino ajustados e dos dados de teste previstos sobrepostos à série original mostrou uma adaptação razoável dos dados de treino ajustados, mas uma previsão dos dados de teste incapaz de lidar com as variações que existiam na realidade. A amplitude das curvas de previsão foi considerada insuficiente em relação às curvas de dados reais no que respeita aos seus máximos, sendo este facto mais evidente no último mês de previsão. A tendência apresentada era ligeiramente decrescente enquanto os dados reais apresentavam alguma estabilidade desta propriedade, Figura 69.

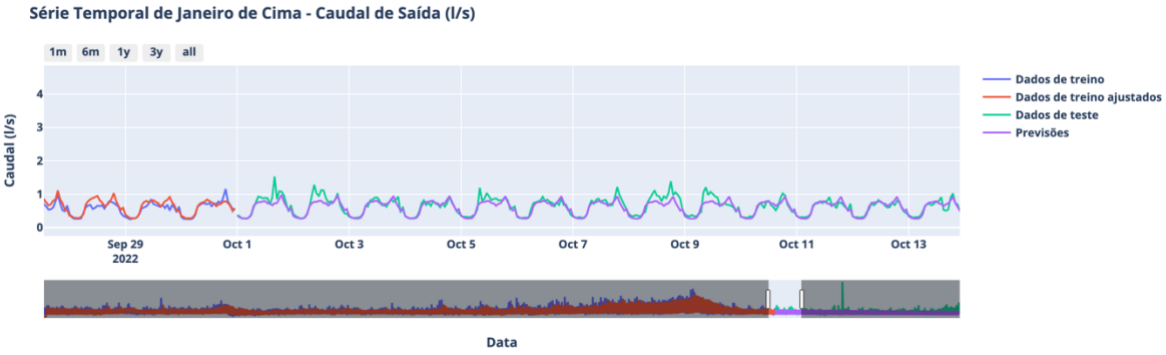
Figura 69 – Previsões do Modelo ARIMA para Janeiro de Cima – Método Manual



Fonte: Elaboração Própria

Na ampliação do gráfico de previsão na zona de transição do período de treino para o período de teste foi possível verificar que o modelo conseguia cumprir a sazonalidade diária e que o problema se prendia mesmo com a dificuldade em lidar com a aleatoriedade dos valores máximos das curvas de dados reais que surgiam de uma forma que, inicialmente parecia aleatória, mas ao verificar mais detalhadamente, pode-se concluir que se tratava dos fins de semana, Figura 70.

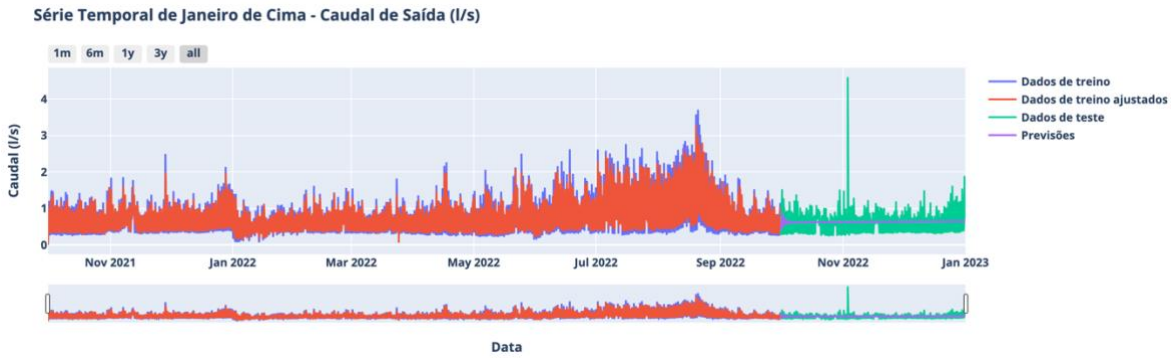
Figura 70 – Ampliação das Previsões do Modelo ARIMA para Janeiro de Cima – Método Manual



Fonte: Elaboração Própria

No caso do modelo obtido através do método automático, a representação gráfica que sobrepôs as previsões aos dados reais, mostrou que para os dados de treino ajustados a adaptação aos dados reais era razoável, o mesmo não se pôde dizer em relação à sobreposição das previsões aos dados de teste reais. As previsões deste modelo obtido pelo método automático traduziram-se basicamente numa linha contínua ao longo do período de teste sem capacidade de respeitar a sazonalidade, ou seja, verificou-se a incapacidade do modelo para efetuar previsões fora do período de treino, Figura 71.

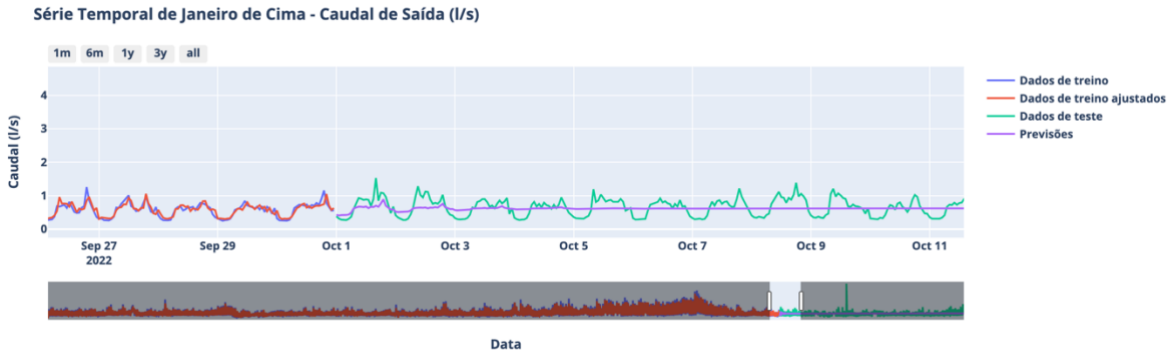
Figura 71 – Previsões do Modelo ARIMA para Janeiro de Cima – Método Automático



Fonte: Elaboração Própria

Ao ampliar a representação gráfica na zona de transição entre os períodos de treino e teste verificou-se que durante o primeiro dia de previsão o modelo ainda aparentava tentar respeitar a sazonalidade, mas depois as curvas convergiam para uma linha continua e assim se mantinham até ao final do período de previsão, Figura 72.

Figura 72 – Ampliação das Previsões do Modelo ARIMA para Janeiro de Cima – Método Automático



Fonte: Elaboração Própria

Em relação aos resultados das métricas obtidas para os dois modelos verificou-se, como seria de esperar, tendo em conta a análise das representações gráficas, que os melhores resultados foram os do modelo obtido pelo método manual qualquer que fosse o prazo da previsão. Assim, para uma previsão de curto prazo foi obtido um erro médio absoluto de 0.09 l/s, a raiz do erro quadrático médio de 0.14 l/s e um coeficiente de determinação de 0.71, valores aproximados. Para uma previsão de longo prazo, o erro médio absoluto obtido foi de 0.14 l/s, a raiz do erro quadrático médio de 0.23 l/s e o coeficiente de determinação foi de 0.32, valores aproximados, Tabela 10.

Tabela 10 – Métricas do Modelo ARIMA Aplicado à Série de Janeiro de Cima

Parametrização do Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Manual	Treino	0.14088	0.21285	0.73437
	Teste curto prazo	**0.09376**	**0.14182**	**0.70701**
	Teste longo prazo	**0.13692**	**0.22685**	**0.31927**
Automática	Treino	0.12727	0.18590	0.79739
	Teste curto prazo	0.20575	0.24511	0.12480
	Teste longo prazo	0.20085	0.27225	0.01951

Fonte: Elaboração Própria

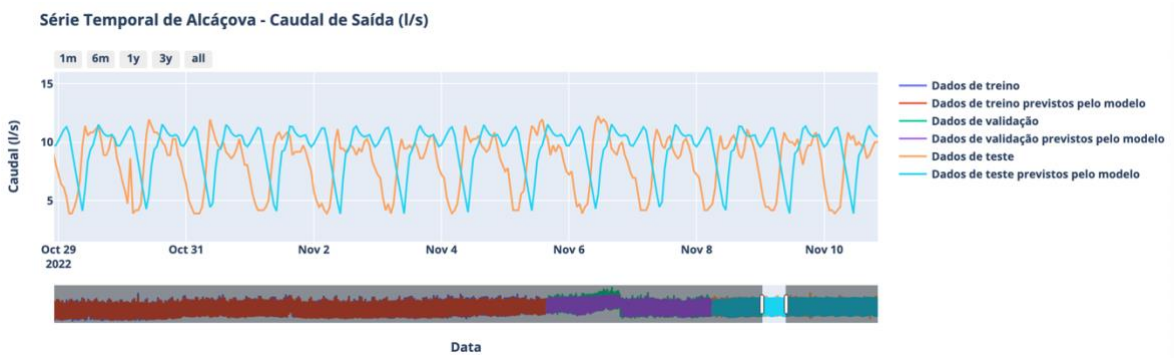
Os resultados dos coeficientes de determinação obtidos significavam que o modelo com parametrização manual conseguia explicar cerca de 71% dos dados de teste num período de 10 dias, mas apenas cerca de 32% num período de 3 meses. Para o período de curto prazo o valor foi considerado bastante razoável, mas para o de longo prazo foi considerado muito baixo e, portanto, considerou-se que o modelo não tinha capacidade de efetuar previsões de longo prazo.

4.2.3. Modelo LSTM

4.2.3.1. Série Temporal de Alcáçova

Ao aplicar o modelo LSTM à série temporal de Alcáçova utilizando um modelo univariável com dados originais ou com dados padronizados, verificou-se que os modelos revelavam ter problemas a efetuar previsões. Embora as curvas de dados de treino ajustados e dados de validação ajustados sobrepostas aos dados reais, mostrarem uma boa aderência, verificou-se que as curvas de previsão perdiam o ajuste aos dados de teste logo ao final de uns dias iniciando um processo de desfasagem entre as curvas de previsão e as curvas de dados de teste reais. Isto revelou a incapacidade dos modelos univariáveis em respeitarem a sazonalidade diária dos dados, Figura 73.

Figura 73 – Ampliação das Previsões do Modelo LSTM Univariável com Dados Padronizados para Alcáçova



Fonte: Elaboração Própria

De referir ainda que o resultado do gráfico de previsões sem estar ampliado, quando se utilizaram dados originais, era semelhante ao obtido com dados pradonizados no que respeita à boa amplitude das curvas de previsão, Figura 74. Com dados originais apresentava uma tendência ligeiramente decrescente em relação aos dados reais e com dados padronizados tinha uma tendência mais ajustada com os dados reais. Não obstante a boa aparência dos gráficos sem ampliação o problema de ambos foi mesmo no desfasamento iniciado ao final de alguns dias de previsão.

Figura 74 – Previsões do Modelo LSTM Univariável com Dados Originais para Alcáçova

Fonte: Elaboração Própria

Estes resultados, embora desanimadores, permitiram perceber a importância da criação de um modelo que tivesse incluída informação sobre a data e a hora em que ocorreu cada valor de caudal. Ficou bastante claro que os modelos, em ambos os casos, se perdiam nas suas previsões e isto apenas poderia ser causado pela falta da informação dos dados temporais, afinal, os modelos apenas tinham sequencias de valores.

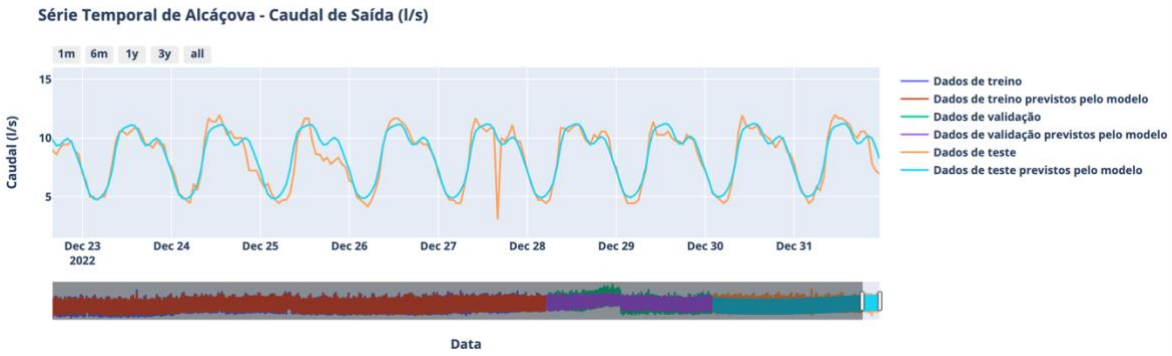
A utilização do modelo multivariável com dados padronizados permitiu obter um modelo com capacidade de efetuar previsões, respeitando a sazonalidade diária e com uma característica intrínseca destes modelos que, neste caso, fica marcada pela capacidade de efetuar previsões seguindo os padrões ocorridos na mesma altura do ano anterior. Esta característica provocou uma ligeira deslocação vertical em relação aos dados reais, mas foi considerada muito interessante no sentido em que nos poderia alertar que algo tinha provocado uma alteração nos valores de caudal de um ano para o outro, na mesma altura do ano, Figura 75.

Figura 75 – Previsões do Modelo LSTM Multivariável com Dados Padronizados para Alcáçova

Fonte: Elaboração Própria

Quando se procedeu à ampliação das curvas de previsão, verificou-se que efetivamente existia sempre um ligeiro desajuste vertical, motivado pelos valores de treino na mesma altura do ano, mas do ano anterior, no entanto verificou-se que o ajuste sazonal diário era muito bom, mesmo nos últimos dias do período de previsão, Figura 76.

Figura 76 – Ampliação das Previsões do Modelo LSTM Multivariável com Dados Padronizados para Alcáçova



Fonte: Elaboração Própria

Relativamente aos resultados numéricos obtidos através das métricas dos 3 modelos considerados, consoante a sua tipologia, univariável ou multivariável, e tipo de dados utilizados, originais ou padronizados, concluiu-se que o melhor modelo considerado foi o que utilizou multivariáveis com dados padronizados. Para uma previsão de curto prazo este modelo apresentou um erro médio absoluto de 0.78 l/s, uma raiz do erro quadrático médio de 0.98 l/s e um coeficiente de determinação de 0.80, valores aproximados. Já para uma previsão de longo prazo o modelo apresentou um erro médio absoluto de 0.74 l/s, uma raiz do erro quadrático médio de 1.00 l/s e um coeficiente de determinação de 0.82 l/s, valores aproximados, Tabela 11. Estes resultados permitiram concluir que, o melhor modelo, para uma previsão de curto prazo, conseguia explicar cerca de 80% dos dados de teste, enquanto numa previsão de longo prazo conseguia explicar cerca de 82% dos mesmos dados.

Tabela 11 – Métricas do Modelo LSTM Aplicado à Série de Alcáçova

Tipo de Dados e de Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Modelo Univariável com Dados Originais	Treino	0.45134	0.59877	0.93399
	Teste curto prazo	1.03144	1.28290	0.64742
	Teste longo prazo	2.10210	2.79378	-0.43464
Modelo Univariável com Dados Padronizados	Treino	0.39139	0.52353	0.94954
	Teste curto prazo	1.17459	1.49636	0.52032
	Teste longo prazo	2.80172	3.47484	-1.21936
Modelo Multivariável com Dados Padronizados	Treino	0.41627	0.54457	0.94540
	Teste curto prazo	**0.77981**	**0.97659**	**0.79568**
	Teste longo prazo	**0.74354**	**0.99628**	**0.81756**

Fonte: Elaboração Própria

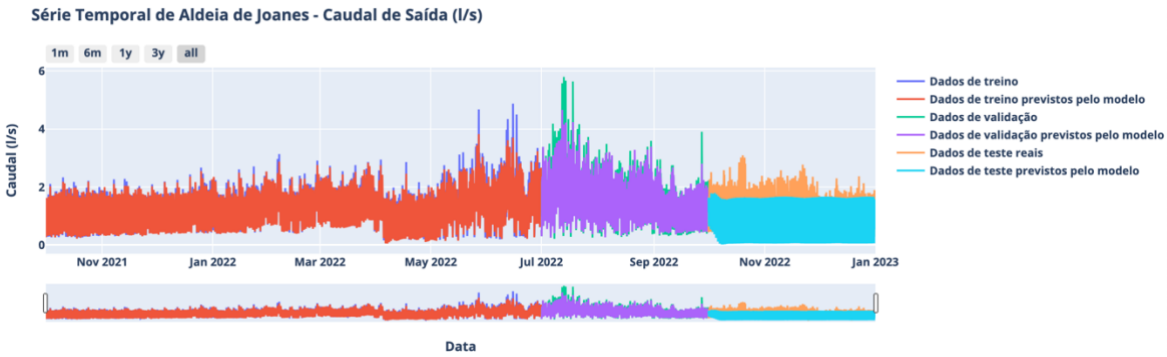
Em relação aos restantes modelos considerados, verificou-se que, para uma previsão de curto prazo ambos apresentavam resultados menos bons, mas ainda assim, resultados válidos. No caso das previsões de longo prazo, ambos os modelos apresentaram valores de coeficientes de determinação negativos o que revelou a sua incapacidade para efetuar este tipo de previsões. Estes resultados foram considerados coerentes com a análise gráfica efetuada em que se verificou que estes modelos ao final de algum tempo de previsão iniciavam um processo de desfasamento, em relação à sazonalidade diária, sem retorno.

4.2.3.2. Série Temporal de Aldeia de Joanes

A aplicação do modelo LSTM à série temporal de Aldeia de Joanes mostrou-se bastante difícil confirmando o que já tinha sido verificado com o modelo *Holt-Winters* e ARIMA relativamente a esta série.

A aplicação do modelo univariável com dados originais e com dados padronizados teve como resultado curvas de previsão que apresentavam um desfasamento, a partir de certa altura, da sazonalidade diária, mas também um deslocamento vertical considerado até inesperado, pela imagem gráfica que apresentava, Figura 77.

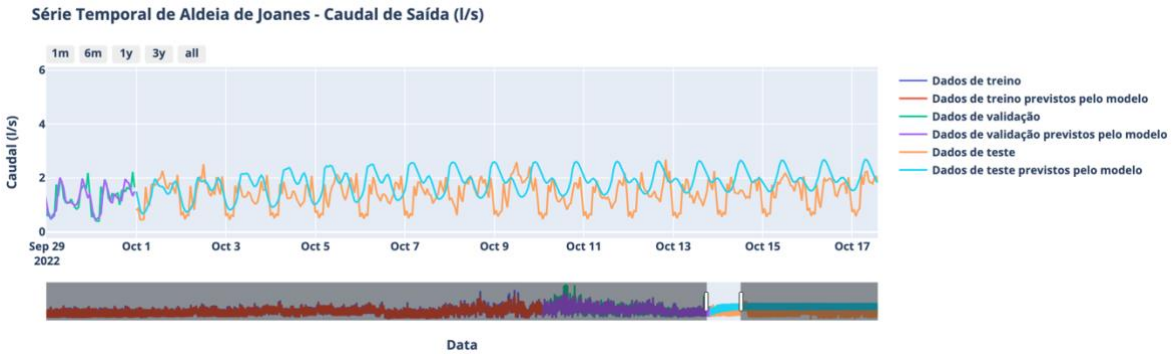
Figura 77 – Previsões do Modelo LSTM Univariável com Dados Originais para Aldeia de Joanes



Fonte: Elaboração Própria

Este deslocamento vertical era verificado nos dois casos, no entanto com dados padronizados o deslocamento era ascendente. Ao se ampliar as curvas de previsão, verificou-se que pouco ou nada tinham a ver com as curvas de caudal relativas aos dados reais, quer pelo deslocamento vertical quer pelo desfasamento que apresentavam, Figura 78.

Figura 78 – Ampliação das Previsões do Modelo LSTM Univariável com Dados Padronizados para Aldeia de Joanes

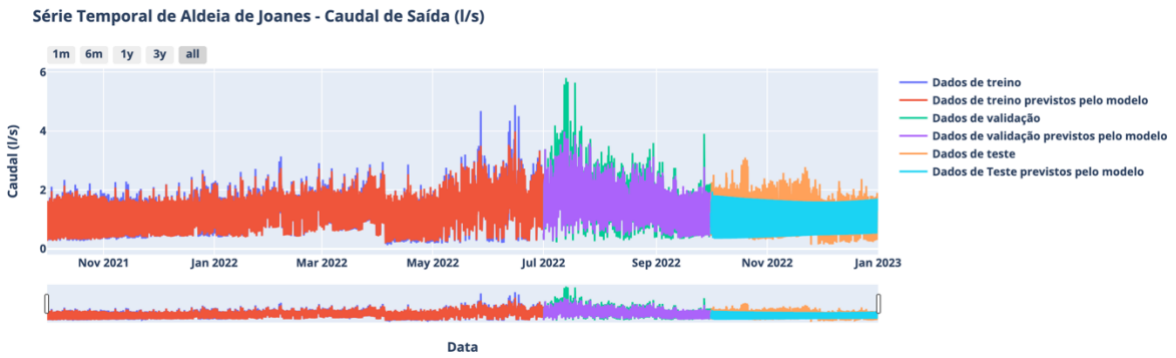


Fonte: Elaboração Própria

Ao aplicar o modelo multivariável com dados padronizados, as previsões melhoraram, mas não o suficiente. O gráfico de previsões mostrou alguma capacidade de o modelo apresentar previsões de acordo com a tendência e forma registadas no ano anterior na

mesma altura, mas, o facto de existir uma grande aleatoriedade de dados reais no período de teste, muito diferente do que tinha ocorrido no ano anterior, impediu o modelo de obter resultados minimamente aceitáveis, Figura 79.

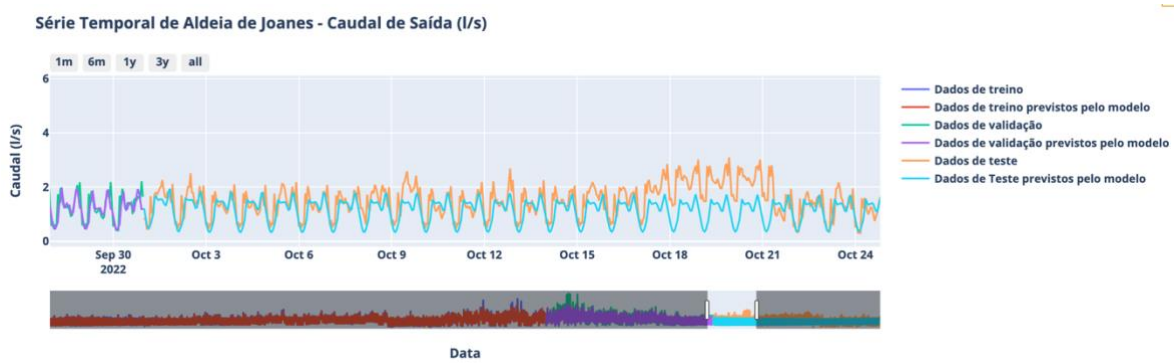
Figura 79 – Previsões do Modelo LSTM Multivariável com Dados Padronizados para Aldeia de Joanes



Fonte: Elaboração Própria

Ampliando o gráfico de previsões na zona de transição entre validação e teste, verificou-se que as curvas de previsão se encontravam muito bem ajustadas em relação à sazonalidade, mas a variabilidade das curvas reais não era seguida com a mesma eficiência, Figura 80.

Figura 80 – Ampliação das Previsões do Modelo LSTM Multivariável com Dados Padronizados para Aldeia de Joanes



Fonte: Elaboração Própria

Em relação aos resultados numéricos, todos os valores das métricas obtidos foram maus, pois em nenhum caso existe um valor do coeficiente de determinação que permita afirmar que existe, de alguma forma, em algum dos modelos considerados com capacidade de explicar pelo menos cinquenta por cento dos dados de teste reais. Ainda

assim, os valores menos maus obtidos, foram obtidos com o modelo multivariável com dados padronizados, com um erro médio absoluto de 0.34 l/s, uma raiz do erro quadrático médio de 0.45 l/s e um coeficiente de determinação de 0.21, valores aproximados, para uma previsão de curto prazo. No caso da previsão de longo prazo foi obtido um erro médio absoluto de 0.33 l/s, uma raiz do erro quadrático médio de 0.45 l/s e um coeficiente de determinação de 0.29, valores aproximados, Tabela 12.

Tabela 12 – Métricas do Modelo LSTM Aplicado à Série de Aldeia de Joanes

Tipo de Dados e de Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Modelo Univariável com Dados Originais	Treino	0.14000	0.19937	0.87844
	Teste curto prazo	0.51941	0.66129	-0.77892
	Teste longo prazo	0.49514	0.65871	-0.50330
Modelo Univariável com Dados Padronizados	Treino	0.14164	0.19500	0.88276
	Teste curto prazo	0.57341	0.72178	-1.11929
	Teste longo prazo	0.80785	0.97134	-2.26884
Modelo Multivariável com Dados Padronizados	Treino	0.13681	0.19440	0.88347
	Teste curto prazo	**0.33508**	**0.44157**	**0.20682**
	Teste longo prazo	**0.33421**	**0.45292**	**0.28926**

Fonte: Elaboração Própria

Estes resultados permitiram concluir que, num período de curto prazo, o modelo menos mau obtido, conseguiu explicar cerca de 21% dos dados de teste enquanto num período de longo prazo essa percentagem subiu para cerca de 29%.

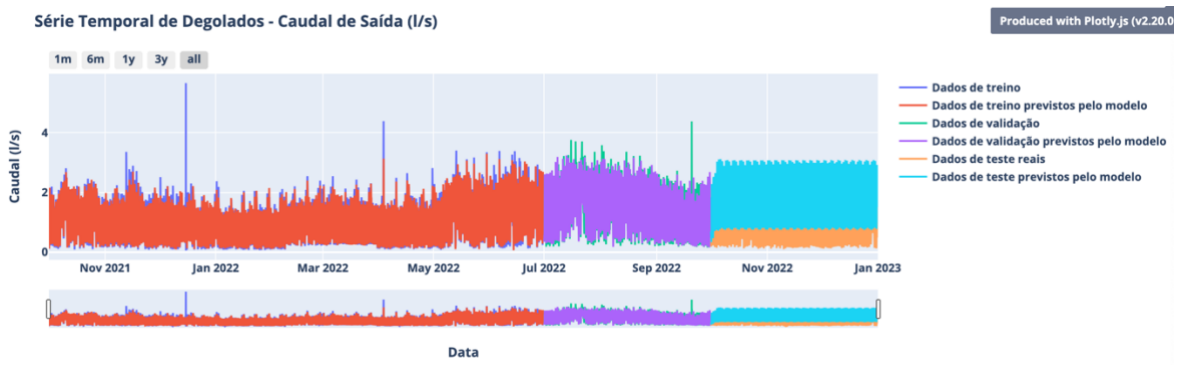
Apesar dos resultados das métricas obtidos que se traduziram na incapacidade do modelo em explicar os dados de teste reais, considerou-se relevante a boa capacidade de previsão do modelo multivariável no que respeitava à sazonalidade diária, mas principalmente na característica de efetuar previsões com base no ocorrido em período igual do ano anterior.

4.2.3.3. Série Temporal de Degolados

No caso da serie temporal de Degolados, a aplicação do modelo univariável com dados originais e com dados padronizados, à semelhança do que já tinha ocorrido com outras séries, traduziu-se em maus resultados quer graficamente quer de forma numérica.

A sobreposição do gráfico de dados previstos com o gráfico de dados reais, mostrou que a amplitude das curvas de previsão era relativamente próxima à dos dados reais, mas que se encontrava deslocada verticalmente. Esta deslocação ficou apresentada no gráfico com um aspeto que denunciava um comportamento inesperado do modelo, Figura 81.

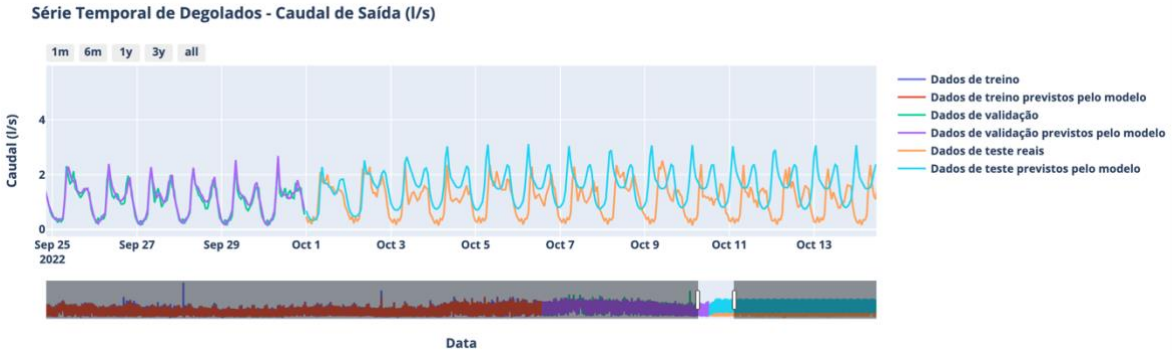
Figura 81 – Previsões do Modelo LSTM Univariável com Dados Originais para Degolados



Fonte: Elaboração Própria

A ampliação do gráfico de previsões sobreposto na zona de transição entre validação e teste, revelou, para além da deslocação vertical, a ocorrência muito rápida do início do desfasamento entre o período das curvas de teste reais e das curvas previstas, evidenciando assim a falta de capacidade do modelo em respeitar a sazonalidade diária, Figura 82. Resultados idênticos foram verificados com a utilização de dados padronizados com o mesmo tipo de modelo.

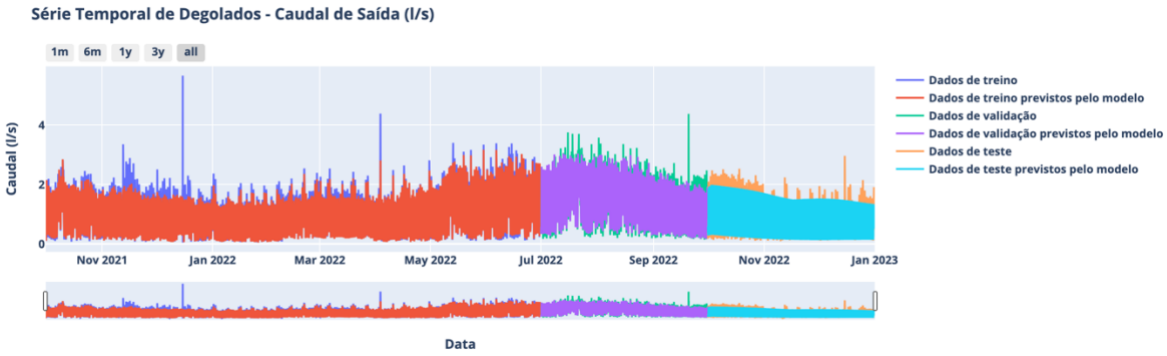
Figura 82 – Ampliação das Previsões do Modelo LSTM Univariável com Dados Originais para Degolados



Fonte: Elaboração Própria

Com a utilização do modelo multivariável e dados padronizados foi possível obter uma representação gráfica de dados previstos mais coerente com a real, embora com um pouco de falta de amplitude, principalmente durante o primeiro mês de previsões. Neste caso já não foi revelado nenhum deslocamento vertical inexplicável e notou-se uma tendência em respeitar o ocorrido no mesmo período do ano anterior, Figura 83.

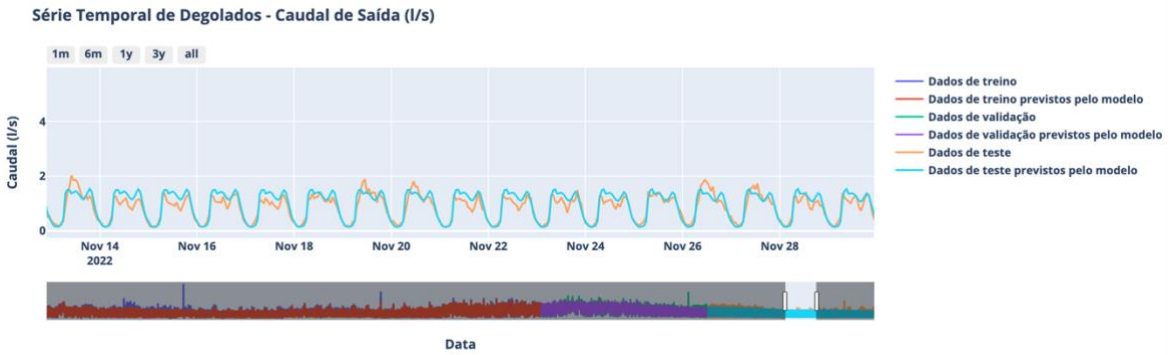
Figura 83 – Previsões do Modelo LSTM Multivariável com Dados Padronizados para Degolados



Fonte: Elaboração Própria

Ampliando as curvas de dados reais e de previsão, aproximadamente a meio do período de previsão, foi observada a capacidade inequívoca do modelo em respeitar a sazonalidade e uma amplitude muito boa em relação às curvas de dados reais. Notou-se que havia alguma variabilidade diária nas curvas reais no que respeita aos seus máximos, não sendo esta prevista pelo modelo, sendo mesmo, a maior limitação detetada, Figura 84.

Figura 84 – Ampliação das Previsões do Modelo LSTM Multivariável com Dados Padronizados para Degolados



Fonte: Elaboração Própria

Os resultados das métricas obtidos mostraram que o modelo multivariável com dados padronizados foi o melhor entre os considerados. Para uma previsão de curto prazo obteve-se um erro médio absoluto de 0.23 l/s, uma raiz do erro quadrático médio de 0.30 l/s e um coeficiente de determinação de 0.76, valores aproximados. No caso da previsão de longo prazo, obteve-se um erro médio absoluto de 0.19 l/s, uma raiz do erro quadrático médio de 0.25 l/s e um coeficiente de determinação de 0.76, valores aproximados, Tabela 13. Em termos de significado dos coeficientes de determinação, concluiu-se que, o melhor modelo, conseguiu explicar cerca de 76% dos dados de teste, em ambos os períodos de previsão, ainda que ligeiramente superior no longo prazo.

Tabela 13 – Métricas do Modelo LSTM Aplicado à Série de Degolados

Tipo de Dados e de Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Modelo Univariável com Dados Originais	Treino	0.13846	0.20416	0.88481
	Teste curto prazo	0.61349	0.82462	-0.78184
	Teste longo prazo	0.85886	1.05783	-3.19231
Modelo Univariável com Dados Padronizados	Treino	0.13951	0.20481	0.88407
	Teste curto prazo	0.54981	0.72136	-0.36355
	Teste longo prazo	0.85976	1.02287	-2.91536
Modelo Multivariável com Dados Padronizados	Treino	0.13324	0.19991	0.88955
	Teste curto prazo	**0.23426**	**0.30331**	**0.75893**
	Teste longo prazo	**0.18761**	**0.25236**	**0.76141**

Fonte: Elaboração Própria

Anotou-se ainda a particularidade deste modelo, em conseguir melhores resultados de métricas em previsões de longo prazo do que no curto prazo, ou seja, métricas de erros mais baixas e coeficientes de determinação mais altos, mesmo que ligeiramente.

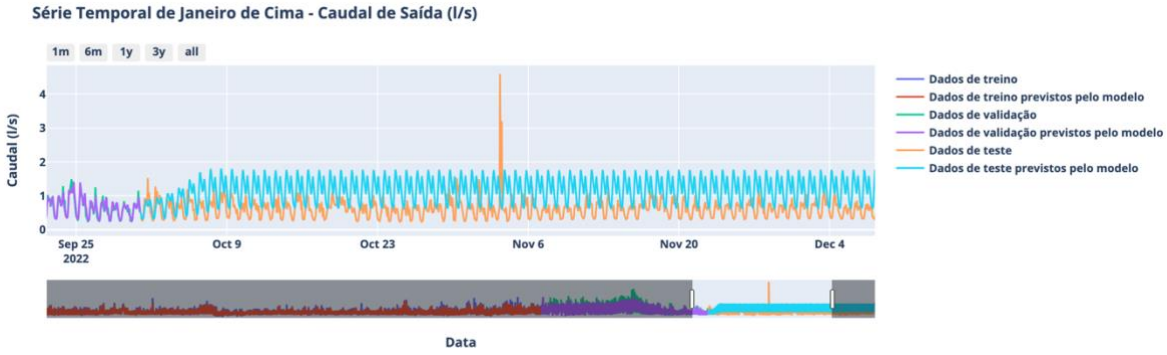
4.2.3.4. Série Temporal de Janeiro de Cima

A aplicação do modelo LSTM à série temporal de Janeiro de Cima mostrou-se bastante difícil, e, neste caso, não só existiram problemas com o modelo univariável, mas também o modelo multivariável ficou longe de conseguir resultados minimamente razoáveis.

A representação gráfica das previsões do modelo univariável, com dados originais, Anexo II, tiveram como resultado curvas que inicialmente apresentavam uma deslocação vertical e depois convergiam para uma linha reta. Já no caso dos dados padronizados, essa mesma deslocação ocorreu, mas as curvas mantiveram-se em todo

o período de previsão, ainda que deslocadas verticalmente e desfasadas das curvas reais em relação à sazonalidade, Figura 85.

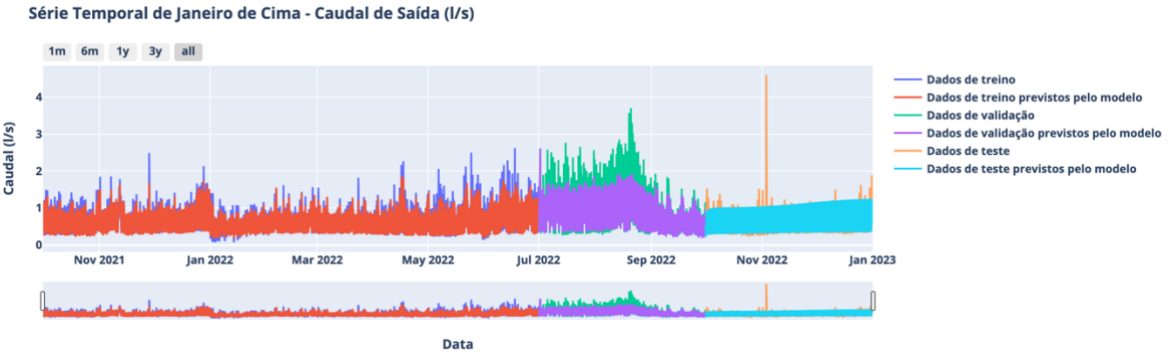
Figura 85 – Ampliação das Previsões do Modelo LSTM Univariável com Dados Padronizados para Janeiro de Cima



Fonte: Elaboração Própria

No caso do modelo multivariável foi possível obter uma representação gráfica que visualmente pareceu estar adequada aos valores reais e, aparentemente, em consonância com o ocorrido no mesmo período do ano anterior, Figura 86.

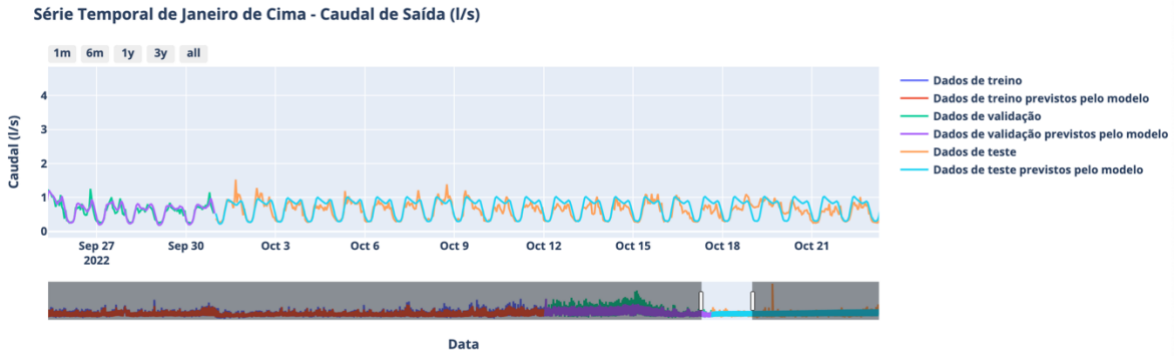
Figura 86 – Previsões do Modelo LSTM Multivariável com Dados Padronizados para Janeiro de Cima



Fonte: Elaboração Própria

A ampliação da representação gráfica na zona de transição mostrou que o modelo se ajustava muito bem à sazonalidade diária, no entanto, os dados reais tinham muitas oscilações nos valores máximos das curvas, o que evidenciava a existência de uma limitação do modelo que não conseguia seguir essas oscilações, Figura 87.

Figura 87 – Ampliação das Previsões do Modelo LSTM Multivariável com Dados Padronizados para Janeiro de Cima



As métricas obtidas pela aplicação dos modelos considerados permitiram concluir que o melhor modelo foi o multivariável com dados padronizados onde, numa previsão de curto prazo, se obteve um erro médio absoluto de 0.11 l/s, uma raiz do erro quadrático médio de 0.16 l/s e um coeficiente de determinação de 0.64, valores aproximados. Para uma previsão de longo prazo, o erro médio absoluto foi de 0.16 l/s, a raiz do erro quadrático médio foi de 0.23 l/s e o coeficiente de determinação foi de 0.33, valores aproximados, Tabela 14.

Tabela 14 – Métricas do Modelo LSTM Aplicado à Série de Janeiro de Cima

Tipo de Dados e de Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Modelo Univariável com Dados Originais	Treino	0.09428	0.13951	0.80167
	Teste curto prazo	0.51010	0.58618	-4.00562
	Teste longo prazo	0.73480	0.78051	-7.05833
Modelo Univariável com Dados Padronizados	Treino	0.08924	0.13480	0.81484
	Teste curto prazo	0.39545	0.48837	-2.47456
	Teste longo prazo	0.56414	0.68097	-5.13397
Modelo Multivariável com Dados Padronizados	Treino	0.08787	0.13350	0.81839
	Teste curto prazo	**0.11215**	**0.15742**	**0.63900**
	Teste longo prazo	**0.15543**	**0.22518**	**0.32928**

Fonte: Elaboração Própria

Estes resultados tinham como significado que para uma previsão de curto prazo o modelo conseguia explicar cerca de 64% dos dados de teste, enquanto numa previsão de longo prazo esse valor descia para cerca de 33%.

4.2.4. Modelo *Prophet*

4.2.4.1. Série Temporal de Alcáçova

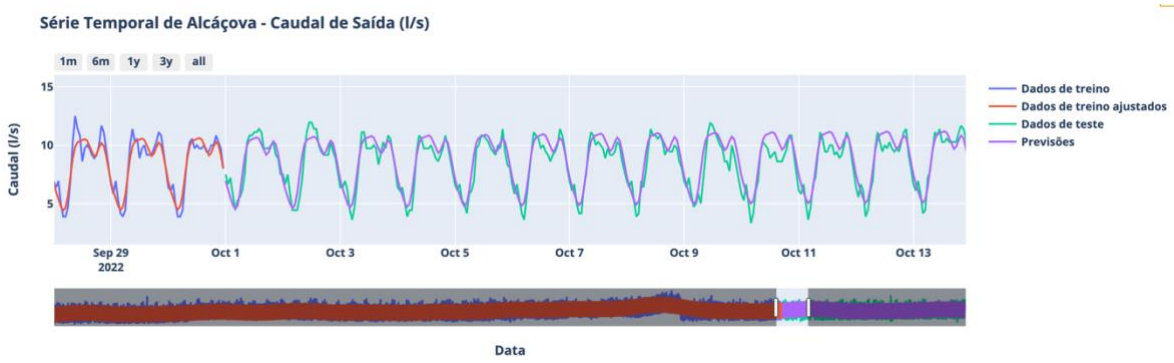
A aplicação do modelo *Prophet* à série temporal de Alcáçova permitiu obter um modelo capaz de gerar previsões dos dados de treino e dos dados de teste bem ajustadas na sua generalidade. No entanto observou-se alguma falta de amplitude nas curvas de previsão prejudicando os máximos e os mínimos existentes tanto nos dados de treino como nos dados de teste. O modelo demostrou ter uma capacidade excelente no que respeita ao cumprimento da sazonalidade anual dos dados, ou seja, observou-se que as previsões seguiam as mesmas oscilações de tendência que os dados de treino ajustados pelo modelo no ano anterior nas mesmas datas, Figura 88.

Figura 88 – Previsões do Modelo *Prophet* para Alcáçova

Fonte: Elaboração Própria

A ampliação do gráfico de previsões na zona de transição permitiu observar o bom ajuste no modelo no que respeita à sazonalidade diária, no entanto, também se comprovou a existência de falta de amplitude, principalmente nos valores mínimos das curvas. De qualquer forma, verificou-se nessa ampliação que os problemas de falta de amplitude já vinham do período de treino, pelo que se considerou uma limitação do modelo, Figura 89.

Figura 89 – Ampliação das Previsões do Modelo *Prophet* para Alcáçova



Fonte: Elaboração Própria

Relativamente aos resultados numéricos obtidos através das métricas do modelo, concluiu-se que, para uma previsão de curto prazo este modelo apresentou um erro médio absoluto de 0.70 l/s, uma raiz do erro quadrático médio de 0.87 l/s e um coeficiente de determinação de 0.84, valores aproximados. Já para uma previsão de longo prazo o modelo apresentou um erro médio absoluto de 0.72 l/s, uma raiz do erro quadrático médio de 0.92 l/s e um coeficiente de determinação de 0.84 l/s, valores aproximados, Tabela 15.

Tabela 15 – Métricas do Modelo *Prophet* Aplicado à Série de Alcáçova

Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Prophet	Treino	0.64318	0.84176	0.87457
	Teste curto prazo	**0.69998**	**0.87344**	**0.83657**
	Teste longo prazo	**0.71676**	**0.91971**	**0.84452**

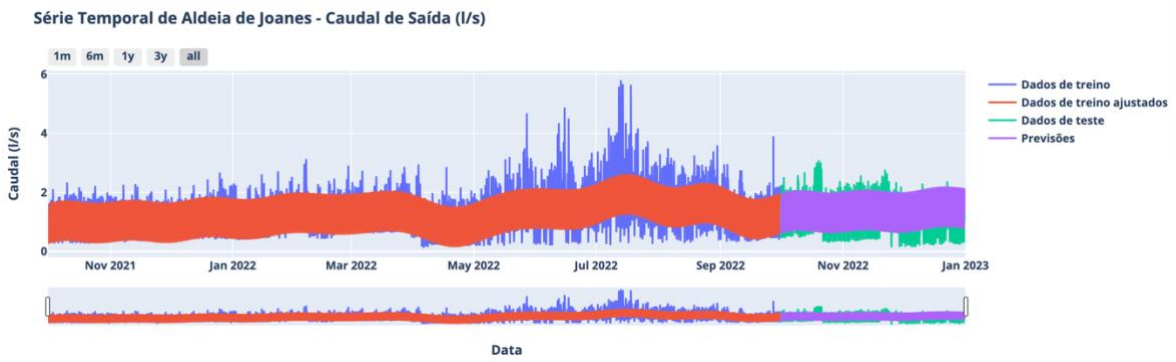
Fonte: Elaboração Própria

Estes resultados permitiram concluir que, o modelo obtido, tanto para uma previsão de curto prazo como de longo prazo, conseguiu explicar cerca de 84% dos dados de teste, ou seja, a sua performance não se viu afetada por aumentar o período de previsão.

4.2.4.2. Série Temporal de Aldeia de Joanes

No caso da aplicação deste modelo à série temporal de Aldeia de Joanes, verificaram-se bastantes problemas na qualidade das previsões, como de resto já tinha acontecido com outros modelos. O modelo apresentou uma falta de capacidade em ajustar os dados de treino previstos pelo modelo durante todo o período de verão, entre maio e setembro. Neste período os dados de treino apresentavam máximos e mínimos consideráveis, mas o modelo manteve a sua amplitude ao longo de todo ano o que se traduziu numa incapacidade de efetuar boas previsões dos dados de treino nesta altura. Claro está que as previsões a efetuar, ou seja, os dados de teste, não estão definidos nesta altura do ano, mas pelo ajustamento que o modelo tem seguramente obter-se-iam valores previstos para esta altura muito aquém da realidade. A sobreposição dos dados reais aos dados previstos permitiu ainda observar a boa característica de o modelo respeitar a sazonalidade anual, repetindo as variações de tendência observadas no ano anterior, no entanto existe uma falta de amplitude em relação às curvas de dados de teste, principalmente nos seus valores mínimos, Figura 90.

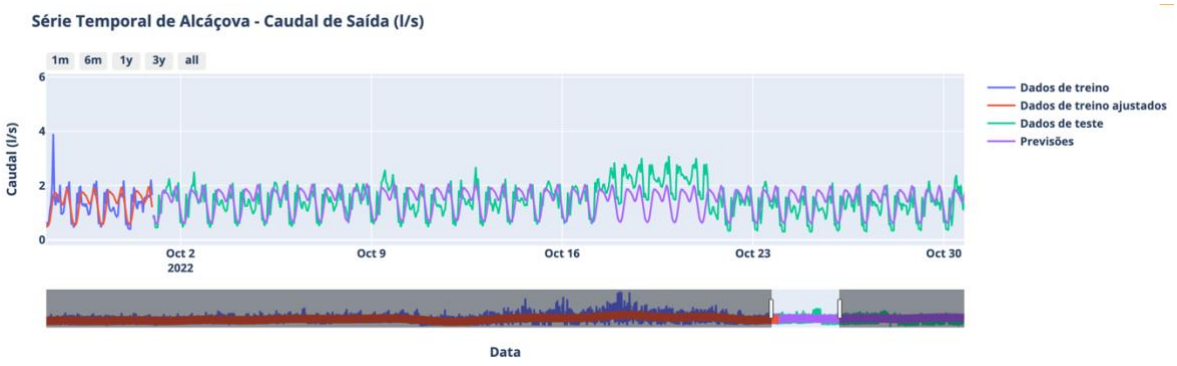
Figura 90 – Previsões do Modelo *Prophet* para Aldeia de Joanes



Fonte: Elaboração Própria

A ampliação do gráfico de previsões mostrou que o modelo respeitava muito bem a sazonalidade diária, mas era incapaz de seguir as variações que os dados de teste reais desta série apresentavam no que respeita às deslocações verticais, tanto ascendentes como descendentes. Como exemplo, a observação ampliada do mês de outubro de 2022, permite verificar uma subida das curvas reais nos dias 18, 19, 20 e depois uma descida abrupta a partir do dia 21. Este comportamento é típico da possível ocorrência de uma rotura no dia 18 e a sua reparação no dia 21, e o modelo mostrou-se incapaz de lidar com esta situação, Figura 91.

Figura 91 – Ampliação das Previsões do Modelo *Prophet* para Aldeia de Joanes



Fonte: Elaboração Própria

A partir do dia 21 o gráfico permitiu observar que a falta de amplitude das curvas de previsão impediam o modelo de atingir os mínimos verificados nos dados de teste reais.

No que respeita aos resultados numéricos, todos os valores das métricas obtidos foram medíocres, os coeficientes de determinação obtidos, tanto para a previsão de curto

prazo como para a de longo prazo não permitiram afirmar que o modelo conseguiria explicar pelo menos cinquenta por cento dos dados de teste reais. Assim, para uma previsão de curto prazo foi obtido um erro médio absoluto de 0.31 l/s, uma raiz do erro quadrático médio de 0.38 l/s e um coeficiente de determinação de 0.42, valores aproximados. No caso de uma previsão de longo prazo foi obtido um erro médio absoluto de 0.42 l/s, uma raiz do erro quadrático médio de 0.51 l/s e um coeficiente de determinação de 0.09, valores aproximados, Tabela 16.

Tabela 16 – Métricas do Modelo *Prophet* Aplicado à Série de Aldeia de Joanes

Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Prophet	Treino	0.29353	0.40139	0.61228
	Teste curto prazo	**0.30820**	**0.37871**	**0.41657**
	Teste longo prazo	**0.42050**	**0.51273**	**0.08920**

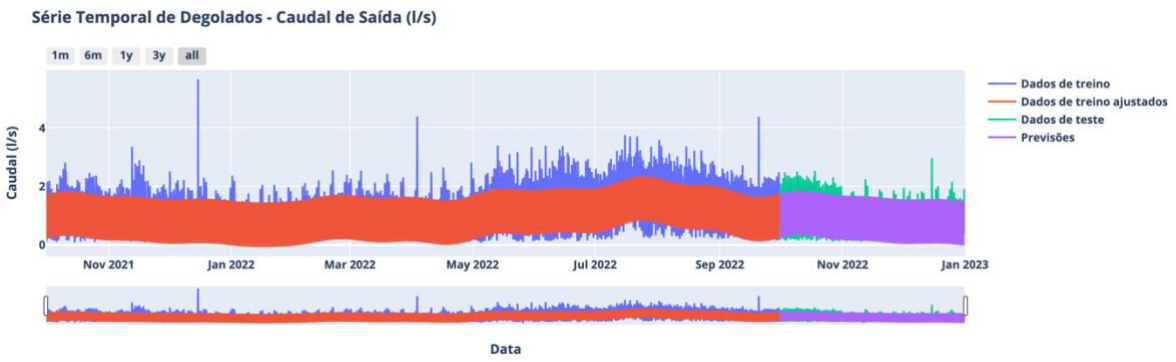
Fonte: Elaboração Própria

Estes resultados permitiram concluir que o modelo, para uma previsão de curto prazo, conseguiu explicar cerca de 42% dos dados de teste, enquanto para uma previsão de longo prazo conseguiu explicar cerca de 9% dos mesmos dados.

4.2.4.3. Série Temporal de Degolados

A aplicação deste modelo aos dados da série temporal de Degolados permitiu obter previsões razoáveis em relação aos dados de teste. A análise da sobreposição gráfica revelou problemas de falta de amplitude dos primeiros dois meses de previsão, principalmente no que se refere aos valores máximos atingidos. Mais uma vez, ficou claro que este modelo consegue respeitar a sazonalidade anual mantendo as variações de tendência do ano anterior nas mesmas datas, Figura 92.

Figura 92 – Previsões do Modelo *Prophet* para Degolados

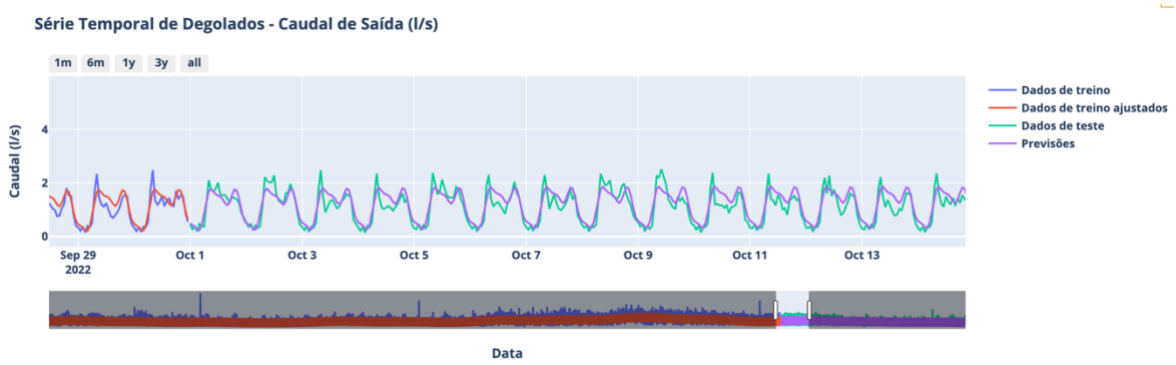


Fonte: Elaboração Própria

O maior problema identificado pela observação gráfica foi mesmo a falta de amplitude, ou seja, aparentemente o modelo estabeleceu uma amplitude média das curvas de previsão, mesmo para os dados de treino, e mantêm essa amplitude ao longo de toda a série. Isto significa que nos períodos de verão, onde existiu maior consumo as previsões do modelo acabaram por ficar aquém do previsto.

Para verificar o correto cumprimento da sazonalidade diária, foi efetuada uma ampliação do gráfico de previsões, sobreposto ao real, e foi verificado que o primeiro apresentou um ajuste muito bom aos dados reais. Curiosamente, com o gráfico ampliado, não ficou tão visível o desajuste em relação à amplitude, Figura 93.

Figura 93 – Ampliação das Previsões do Modelo *Prophet* para Degolados



Fonte: Elaboração Própria

No que respeita aos resultados das métricas obtidos, para uma previsão de curto prazo obteve-se um erro médio absoluto de 0.25 l/s, uma raiz do erro quadrático médio de 0.31 l/s e um coeficiente de determinação de 0.76, valores aproximados. Já para a previsão de longo prazo, foi obtido um erro médio absoluto de 0.23 l/s, uma raiz do

erro quadrático médio de 0.28 l/s e um coeficiente de determinação de 0.70, valores aproximados, Tabela 17.

Tabela 17 – Métricas do Modelo *Prophet* Aplicado à Série de Degolados

Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Prophet	Treino	0.24291	0.32664	0.74835
	Teste curto prazo	**0.24848**	**0.30568**	**0.75515**
	Teste longo prazo	**0.22622**	**0.28351**	**0.6989**

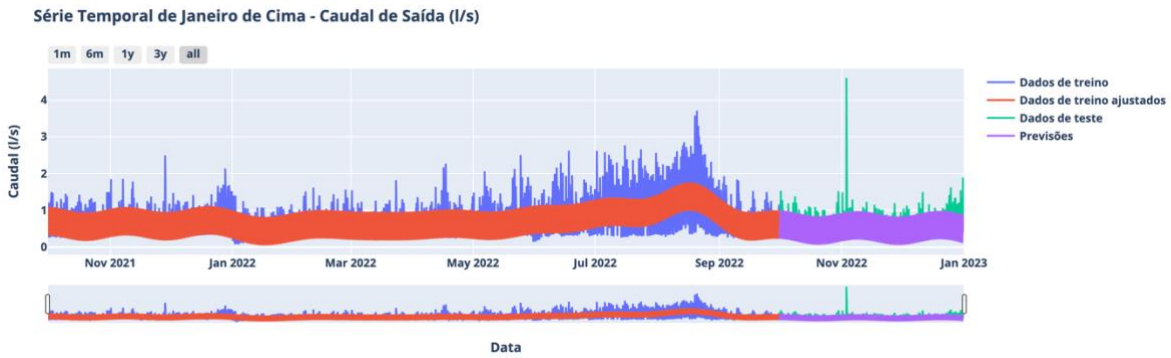
Fonte: Elaboração Própria

Os resultados obtidos permitiram concluir que o modelo, numa previsão de curto prazo, conseguia explicar cerca de 76% dos dados de teste, enquanto numa previsão de longo prazo apenas conseguia explicar cerca de 70% dos mesmos dados. Não obstante esta observação, verificou-se que os valores das métricas de erro obtidas foram mais baixos para a previsão de longo prazo do que para a de curto prazo.

4.2.4.4. Série Temporal de Janeiro de Cima

As previsões do modelo para a série de Janeiro de Cima mostraram, mais uma vez, através da sobreposição de gráficos, muita dificuldade no ajuste às curvas de dados de treino reais durante o período de verão, entre os meses de junho a setembro. Neste período os dados reais possuíam uma amplitude que o modelo não conseguiu ajustar, principalmente no que respeita aos máximos. É certo que o período de previsão não se encontra neste período, ainda assim, considerou-se muito pertinente anotar este facto o qual se considerou como sendo uma limitação do modelo. Foi também verificado que, o modelo apresenta para o período previsto as mesmas variações de tendência observadas no ano anterior no mesmo período, o que significa um cumprimento escrupuloso da sazonalidade anual, Figura 94.

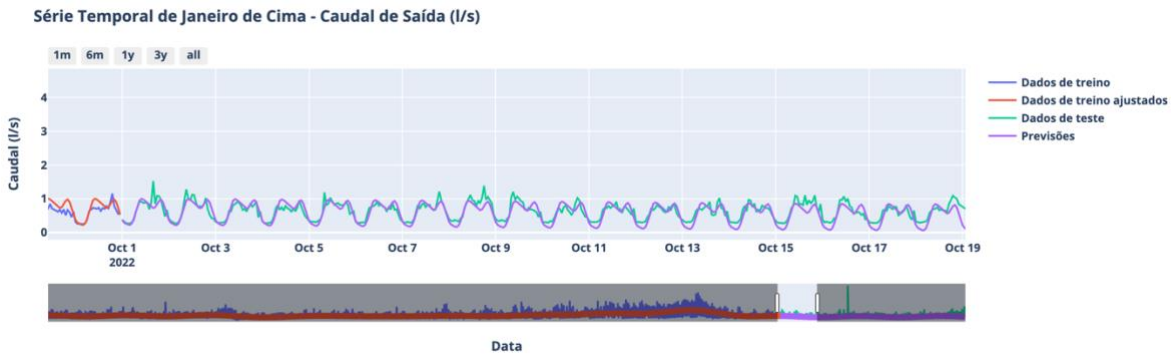
Figura 94 – Previsões do Modelo *Prophet* para Janeiro de Cima



Fonte: Elaboração Própria

A ampliação do gráfico de previsões na zona de transição, permitiu observar o correto cumprimento da sazonalidade diária por parte do modelo, e apenas uma pequena dificuldade em efetuar previsões nos fins de semana onde os máximos não ficavam previstos corretamente. Verificou-se ainda algum exagero nos mínimos previstos em relação aos dados reais, Figura 95.

Figura 95 – Ampliação das Previsões do Modelo *Prophet* para Janeiro de Cima



Fonte: Elaboração Própria

A aplicação deste modelo, numa previsão de curto prazo, permitiu obter um erro médio absoluto de 0.11 l/s, uma raiz do erro quadrático médio de 0.15 l/s e um coeficiente de determinação de 0.67, valores aproximados. Na previsão de longo prazo, foi obtido um erro médio absoluto de 0.15 l/s, uma raiz do erro quadrático médio de 0.22 l/s e um coeficiente de determinação de 0.37, valores aproximados, Tabela 18.

Tabela 18 – Métricas do Modelo *Prophet* Aplicado à Série de Janeiro de Cima

Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Prophet	Treino	0.16999	0.23828	0.66710
	Teste curto prazo	**0.11169**	**0.15022**	**0.67128**
	Teste longo prazo	**0.15186**	**0.21766**	**0.37332**

Fonte: Elaboração Própria

Pelos resultados obtidos, pôde-se concluir que o modelo, numa previsão de curto prazo, conseguia explicar cerca de 67% dos dados de teste, enquanto numa previsão de longo prazo, essa percentagem ficava reduzida para apenas cerca de 37%.

Foi ainda anotado o facto de as métricas de erro obtidas terem sido mais baixas nos períodos de teste do que no período de treino. Foi considerado que este facto se poderia ter ficado a dever à dificuldade do modelo em ajustar o modelo no período de treino durante os meses de verão, constituindo assim uma limitação do modelo, conforme referido anteriormente.

CAPÍTULO V – DISCUSSÃO

Este capítulo tem como objetivo inicial o agrupamento dos resultados obtidos anteriormente para o curto prazo e longo prazo. Foram considerados em cada localidade, apenas os melhores resultados de cada modelo e construídas tabelas com os mesmos. Seguidamente apresentam-se tabelas dos valores médios das métricas de cada modelo, considerando apenas as localidades onde os modelos apresentaram alguma capacidade de efetuar previsões. Finalmente, apresenta-se uma comparação dos resultados obtidos com as indicações que tinham sido deixadas pela revisão bibliográfica no ponto 2.1..

5.1. Métricas Obtidas por Modelo para Previsões de Curto Prazo

A aplicação dos modelos às séries temporais das várias localidades obrigou a utilizar procedimentos com algumas variações dependendo do modelo em causa, o que levou à obtenção de vários resultados para cada localidade e modelo, com exceção do modelo *Prophet*. Nesta fase procedeu-se então à recolha dos resultados das métricas para previsões de curto prazo, apenas com os melhores resultados, obtidos no ponto 4.2., com o objetivo de conseguir identificar o modelo com as melhores métricas para cada localidade.

5.1.1. Alcáçova

No caso de Alcáçova todos os modelos apresentaram resultados bastante bons tendo o *Holt-Winters* e o ARIMA sido os melhores. Os modelos LSTM e *Prophet* apresentaram resultados menos bons, mas foi considerado que o facto de serem modelos com uma capacidade de implementar e seguir uma sazonalidade anual, e tendo em conta que o comportamento do caudal distribuído no período de teste era superior ao do ano anterior, estes dois modelos acabaram por ficar prejudicados. Não obstante, esta característica destes dois modelos foi anotada como interessante em alguns contextos.

Assim, o melhor modelo identificado foi o modelo ARIMA com parametrização manual, onde foi conseguido um erro médio absoluto de 0.56 l/s, uma raiz do erro quadrático

médio de 0.72 l/s e um coeficiente de determinação de 0.89, valores aproximados, Tabela 19.

Tabela 19 – Comparação de Métricas dos Modelos – Alcáçova – Curto Prazo

Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Holt-Winters (Transformação Logarítmica)	Treino	0.51715	0.69534	0.91461
	Teste	0.60190	0.78912	0.86659
ARIMA (Parametrização Manual)	Treino	0.42373	0.56906	0.94281
	Teste	**0.55562**	**0.72449**	**0.88755**
LSTM (Multivariável Dados Padronizados)	Treino	0.41627	0.54457	0.94540
	Teste	0.77981	0.97659	0.79568
Prophet	Treino	0.64318	0.84176	0.87457
	Teste	0.69998	0.87344	0.83657

Fonte: Elaboração Própria

Foi concluído que, estes resultados, permitiam afirmar que o modelo ARIMA com a parametrização manual conseguiu explicar cerca de 89% dos dados de teste numa previsão de curto prazo, no caso da série temporal de Alcáçova.

5.1.2. Aldeia de Joanes

A série temporal de Aldeia de Joanes apresentou muitos problemas para todos os modelos devido ao seu desenvolvimento irregular, principalmente depois do período dos meses de verão. Considerou-se que, possivelmente todas aquelas deslocações verticais observadas se poderiam ter ficado a dever a roturas na rede de abastecimento que apareciam durante alguns dias e depois de serem reparadas os valores deslocavam-se novamente para baixo. Por outro lado, o comportamento da série nos meses de verão deixou a ideia de que poderia existir muita população flutuante, como emigrantes ou segundas habitações, onde durante os restantes períodos do ano as casas se poderiam encontrar fechadas e durante o verão a sua maioria encontrar-se

habitada. Isso poderia explicar a variação abrupta entre os consumos dos meses de verão e os consumos nos restantes períodos. Ficou assim registada a impossibilidade em conseguir obter um modelo que permitisse efetuar previsões minimamente aceitáveis num setor de abastecimento com estas características relacionadas com uma aleatoriedade muito significativa nos padrões de distribuição de água.

Nenhum dos modelos conseguidos apresentou uma capacidade mínima de efetuar previsões pois qualquer um deles apresentou um coeficiente de determinação abaixo de 0.50 o que significa que nenhum conseguiu explicar nem 50% dos dados de teste.

Tabela 20 – Comparação de Métricas dos Modelos – Aldeia de Joanes – Curto Prazo

Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Holt-Winters (Transformação Logarítmica)	Treino	0.18372	0.28551	0.80383
	Teste	0.30528	0.38290	0.40358
ARIMA (Parametrização Manual)	Treino	0.17647	0.26998	0.82460
	Teste	**0.24250**	**0.36735**	**0.45106**
LSTM (Multivariável Dados Padronizados)	Treino	0.13681	0.19440	0.88347
	Teste	0.33508	0.44157	0.20682
Prophet	Treino	0.29353	0.40139	0.61228
	Teste	0.30820	0.37871	0.41657

Fonte: Elaboração Própria

Apesar de tudo, o modelo que apresentou os resultados menos maus foi o ARIMA com parametrização manual onde foi obtido um erro médio absoluto de 0.24 l/s, uma raiz do erro quadrático médio de 0.37 l/s e um coeficiente de determinação de 0.45, valores aproximados, Tabela 20.

5.1.3. Degolados

No caso da série temporal de Degolados foram obtidos resultados de métricas dos vários modelos muito equivalentes. Todos os modelos apresentaram boas capacidades

de ajuste aos dados de treino e também uma boa capacidade de previsão. Os valores das métricas para uma previsão de curto prazo mostraram que o melhor modelo foi o *Holt-Winters* onde foi obtido um erro médio absoluto de 0.19 l/s, uma raiz do erro quadrático médio de 0.28 l/s e um coeficiente de determinação de 0.79, valores aproximados, Tabela 21.

Tabela 21 – Comparação de Métricas dos Modelos – Degolados – Curto Prazo

Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Holt-Winters (Dados Originais)	Treino	0.15895	0.23667	0.86788
	Teste	**0.19460**	**0.28224**	**0.79126**
ARIMA (Parametrização Manual)	Treino	0.16433	0.25226	0.84991
	Teste	0.19191	0.29071	0.77854
LSTM (Multivariável Dados Padronizados)	Treino	0.13324	0.19991	0.88955
	Teste	0.23426	0.30331	0.75893
Prophet	Treino	0.24291	0.32664	0.74835
	Teste	0.24848	0.30568	0.75515

Fonte: Elaboração Própria

Os modelos que apresentaram resultados mais baixos dos coeficientes de determinação e erros maiores foram o LSTM e o *Prophet*. À semelhança do que sucedeu na série de Alcáçova, considerou-se que a característica que estes modelos têm de implementarem uma sazonalidade anual, para além da diária, acabou por prejudicar os resultados das métricas nas previsões de curto prazo.

Assim, concluiu-se que para uma previsão de curto prazo, o modelo *Holt-Winters*, quando aplicado à série temporal de Degolados, conseguiu explicar cerca de 79% dos dados de teste.

5.1.4. Janeiro de Cima

Em relação à aplicação dos modelos à série temporal de Janeiro de Cima foram experimentadas dificuldades semelhantes às da série de Aldeia de Joanes embora numa escala menor. A série apresentava uma aleatoriedade grande do seu comportamento e um aumento dos máximos muito significativo nos meses de verão. Apesar de tudo, para previsões de curto prazo, foi possível obter resultados das métricas muito aceitáveis, tendo sido considerado como melhor o modelo ARIMA com parametrização manual. A aplicação deste modelo permitiu obter previsões com um erro médio absoluto de 0.09 l/s, uma raiz do erro quadrático médio de 0.14 l/s e um coeficiente de determinação de 0.71, valores aproximados, Tabela 22.

Tabela 22 – Comparação de Métricas dos Modelos – Janeiro de Cima – Curto Prazo

Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Holt-Winters (Transformação Logarítmica)	Treino	0.10939	0.17389	0.82271
	Teste	0.10199	0.15238	0.66175
ARIMA (Parametrização Manual)	Treino	0.14088	0.21285	0.73437
	Teste	**0.09376**	**0.14182**	**0.70701**
LSTM (Multivariável Dados Padronizados)	Treino	0.08787	0.13350	0.81839
	Teste	0.11215	0.15742	0.63900
Prophet	Treino	0.16999	0.23828	0.66710
	Teste	0.11169	0.15022	0.67128

Fonte: Elaboração Própria

Neste caso, o modelo *Prophet* apresentou resultados ligeiramente melhores do que o modelo *Holt-Winters*, tendo sido o LSTM o modelo com resultados piores das métricas.

Como conclusão, para uma previsão de curto prazo da série temporal de Janeiro de Cima, verificou-se que o melhor modelo foi o ARIMA com parametrização manual, com uma capacidade de conseguir explicar cerca de 71% dos dados de teste.

5.2. Métricas Obtidas por Modelo para Previsões de Longo Prazo

Neste ponto foi seguida a mesma metodologia indicada no ponto 5.1. apenas com a alteração de se tratar agora de previsões de longo prazo.

5.2.1. Alcáçova

Para uma previsão de longo prazo, todos os modelos apresentaram resultados muito bons no que às métricas respeita. O modelo que melhor se adaptou aos dados de treino e, por isso, apresentou os valores mais baixos de erros e do coeficiente de determinação em relação aos mesmos foi o modelo LSTM, utilizando um modelo multivariável com dados padronizados. No entanto, em relação à capacidade de efetuar previsões, foi o modelo *Prophet* que apresentou melhores resultados. Para as previsões efetuadas por este modelo foi conseguido um erro médio absoluto de 0.72 l/s, uma raiz do erro quadrático médio de 0.92 l/s e um coeficiente de determinação de 0.84, valores aproximados, Tabela 23.

Tabela 23 – Comparação de Métricas dos Modelos – Alcáçova – Longo Prazo

Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Holt-Winters (Transformação Logarítmica)	Treino	0.51715	0.69534	0.91461
	Teste	0.79151	1.03839	0.80181
ARIMA (Parametrização Automática)	Treino	0.44654	0.59351	0.93779
	Teste	0.76914	1.00876	0.81296
LSTM (Multivariável Dados Padronizados)	Treino	0.41627	0.54457	0.94540
	Teste	0.74354	0.99628	0.81756
Prophet	Treino	0.64318	0.84176	0.87457
	Teste	**0.71676**	**0.91971**	**0.84452**

Fonte: Elaboração Própria

Os resultados obtidos permitiram concluir que para a série temporal de Alcáçova, o modelo *Prophet*, para uma previsão de longo prazo conseguiu explicar cerca de 84% dos dados de teste.

5.2.2. Aldeia de Joanes

No caso da Aldeia de Joanes, manteve-se tudo o já referido no ponto 5.1.2. mesmo para uma previsão de longo prazo. Nenhum modelo apresentou previsões que evidenciassem explicar pelo menos 50% dos dados de teste, Tabela 24.

Tabela 24 – Comparação de Métricas dos Modelos – Aldeia de Joanes – Longo Prazo

Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Holt-Winters (Dados Originais)	Treino	0.17687	0.26747	0.82784
	Teste	0.45353	0.54847	-0.04222
ARIMA (Parametrização Manual)	Treino	0.17647	0.26998	0.82460
	Teste	**0.32898**	**0.45211**	**0.29183**
LSTM (Multivariável Dados Padronizados)	Treino	0.13681	0.19440	0.88347
	Teste	0.33421	0.45292	0.28926
Prophet	Treino	0.29353	0.40139	0.61228
	Teste	0.42050	0.51273	0.08920

Fonte: Elaboração Própria

Por este facto considerou-se que, no caso da Aldeia de Joanes, nenhum dos modelos estudados permitiu obter previsões válidas.

5.2.3. Degolados

Os resultados das previsões de longo prazo para a série temporal de Degolados, mostraram que os modelos *Holt-Winters* e ARIMA apresentavam um ajuste muito bom aos dados de treino, mas depois nos dados de teste perdiam bastante eficiência, o que indiciou a existência de algum *overfitting*. Apenas os modelos LSTM e *Prophet* permitiram obter resultados mais próximos entre dados de treino e de teste.

O melhor modelo obtido foi o LSTM segundo um modelo multivariável com dados padronizados, que gerou previsões com um erro médio absoluto de 0.19 l/s, uma raiz do erro quadrático médio de 0.25 l/s e um coeficiente de determinação de 0.76, valores aproximados, Tabela 25.

Tabela 25 – Comparação de Métricas dos Modelos – Degolados – Longo Prazo

Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Holt-Winters (Dados Originais)	Treino	0.15895	0.23667	0.86788
	Teste	0.21077	0.28865	0.68783
ARIMA (Parametrização Manual)	Treino	0.16433	0.25226	0.84991
	Teste	0.23672	0.34499	0.55410
LSTM (Multivariável Dados Padronizados)	Treino	0.13324	0.19991	0.88955
	Teste	**0.18761**	**0.25236**	**0.76141**
Prophet	Treino	0.24291	0.32664	0.74835
	Teste	0.22622	0.28351	0.69890

Fonte: Elaboração Própria

Estes resultados permitiram concluir que, no caso na série temporal de Degolados, o modelo LSTM, multivariável com dados padronizados, conseguiu explicar cerca de 76% dos dados de teste numa previsão de longo prazo.

5.2.4. Janeiro de Cima

Os modelos obtidos para a série temporal de Janeiro de Cima, ao contrário do que tinha sucedido para uma previsão de curto prazo, numa previsão de longo prazo, não foram capazes de gerar previsões que permitissem explicar pelo menos 50% dos dados de teste, Tabela 26.

Os modelos *Holt-Winters*, ARIMA e LSTM, destacaram-se na capacidade de se ajustar aos dados de treino, mas o mesmo não aconteceu para as previsões.

Assim sendo, foi concluído que para este prazo de previsão, não foi possível obter nenhum modelo capaz de gerar previsões para a série temporal de Janeiro de Cima.

Tabela 26 – Comparação de Métricas dos Modelos – Janeiro de Cima – Longo Prazo

Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Holt-Winters (Transformação Logarítmica)	Treino	0.10939	0.17389	0.82271
	Teste	**0.11827**	**0.20268**	**0.45659**
ARIMA (Parametrização Manual)	Treino	0.14088	0.21285	0.73437
	Teste	0.13692	0.22685	0.31927
LSTM (Multivariável Dados Padronizados)	Treino	0.08787	0.13350	0.81839
	Teste	0.15543	0.22518	0.32928
Prophet	Treino	0.16999	0.23828	0.66710
	Teste	0.15186	0.21766	0.37332

Fonte: Elaboração Própria

5.3. Média das Métricas

Após se terem considerado os melhores resultados das métricas para cada localidade, divididos entre previsões de curto prazo e de longo prazo, verificou-se uma incoerência nos melhores resultados consoante a localidade. Por outro lado, existiam localidades que evidenciavam resultados de modelos com coeficientes de determinação abaixo de 0.50.

Assim, decidiu-se desconsiderar todos os resultados dos modelos em localidades com coeficiente de determinação inferir a 0.50 e efetuar a média das restantes métricas, separadas por período de previsão, por forma a obter resultados mais conclusivos. No curto prazo, foram desconsiderados os resultados das métricas obtidos para Aldeia de Joanes e, no longo prazo, foram desconsiderados os de Aldeia de Joanes e Janeiro de Cima.

Obtendo os valores médios das métricas, para previsões de curto prazo, foi construída a Tabela 27, onde constam as médias das métricas obtidas em cada localidade, com exceção de Aldeia de Joanes, por modelo, e onde se pôde verificar que os melhores resultados médios obtidos foram os do modelo ARIMA com um erro médio absoluto de

0.28 l/s, uma raiz do erro quadrático médio de 0.39 l/s e um coeficiente de determinação de 0.79, valores aproximados.

Tabela 27 – Média das Métricas para Previsões de Curto Prazo

Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Holt-Winters	Teste	0.29950	0.40791	0.77320
ARIMA	Teste	**0.28043**	**0.38567**	**0.79103**
LSTM	Teste	0.37541	0.47911	0.73120
Prophet	Teste	0.35338	0.44311	0.75433

Fonte: Elaboração Própria

No caso das previsões de longo prazo, desconsiderando os valores obtidos em Aldeia de Joanes e Janeiro de Cima, foi construída a Tabela 28, onde se pôde verificar que os melhores resultados médios obtidos foram os do modelo LSTM com um erro médio absoluto de 0.19 l/s, uma raiz do erro quadrático médio de 0.25 l/s e um coeficiente de determinação de 0.76, valores aproximados.

Tabela 28 – Média das Métricas para Previsões de Longo Prazo

Modelo	Dados	MAE (l/s)	RMSE (l/s)	R^2
Holt-Winters	Teste	0.21077	0.28865	0.68783
ARIMA	Teste	0.23672	0.34499	0.55410
LSTM	Teste	**0.18761**	**0.25236**	**0.76141**
Prophet	Teste	0.22622	0.28351	0.69890

Fonte: Elaboração Própria

5.4. Resultados dos Modelos Clássicos vs. Modernos

Considerando os modelos *Holt-Winters* e ARIMA como modelos clássicos e o LSTM e o *Prophet* como modernos, verificou-se que os resultados das métricas obtidos indicam uma performance superior para os modelos clássicos quando se abordam previsões de

curto prazo. Da mesma forma, os resultados indicam que os modelos modernos conseguem melhores resultados para as previsões de longo prazo.

De acordo com o estudo referido em [5], que compara o modelo *Holt-WInters* com o modelo ARIMA, era de esperar que o último tivesse uma melhor performance em relação ao primeiro de forma significativa. Os resultados médios mostraram que, no curto prazo, isso sucedeu, ainda que de forma pouco significativa, mas no longo prazo verificou-se o contrário. Considerou-se que os resultados do modelo ARIMA ficaram prejudicados pelas características evidenciadas pelos dados das séries. A fraca estacionaridade, forte existência de autocorrelação e o facto de a distribuição dos dados estar muito afastada de uma distribuição normal, poderá ter tido sido determinante para o seu fraco desempenho.

Ao contrário do referido em [9], quando os autores indicam que as previsões dos modelos baseados em redes neuronais têm boas performances apenas em horizontes de curto prazo, onde se prevê apenas o próximo valor, verificou-se que, neste caso, não é assim. O modelo LSTM testado apresentou melhores resultados para previsões de longo prazo do que para as de curto prazo. Por outro lado, este modelo foi testado de uma forma que tentou representar aquilo que seria o seu funcionamento num ambiente de produção, prevendo valor a valor e apresentando o conjunto de resultados. Confrontando os resultados obtidos com os apresentados em alguma literatura, onde se exemplifica o seu funcionamento e se apresentam resultados extraordinários seguindo metodologias só possíveis em ambientes laboratoriais, como é o caso referido em [20], concluiu-se que os resultados obtidos são mais realistas.

No que respeita à capacidade de captação da sazonalidade, os 4 modelos conseguiram ajustar-se muito bem à sazonalidade diária, o mesmo não se pode dizer em relação à sazonalidade anual. Os modelos *Holt-Winters* e ARIMA não conseguiram de todo demostrar que as suas previsões tinham em conta o sucedido no mesmo período do ano anterior. Verificou-se que as suas previsões eram mais influenciadas pelos resultados mais próximos do início do período de previsão. Quer isto dizer que estes modelos perdem precisão sempre que as séries evidenciam fortes padrões sazonais. No caso do modelo ARIMA, esta característica agora verificada já tinha sido enunciada em [17] numa comparação com o modelo *Prophet*. Em contrapartida, os modelos LSTM

e *Prophet*, mostraram previsões com uma forte tendência em cumprir o padrão sazonal anual. Isto contraria o referido em [16], que deixa subentendido que o modelo LSTM não tem capacidade de cumprir um padrão sazonal anual.

No caso do modelo *Prophet*, o mesmo demostrou boas capacidades de ajuste das suas previsões seguindo corretamente as sazonalidades diária e anual, no entanto notou-se uma limitação significativa na amplitude das curvas de previsão. Essa limitação ficou mais visível nas previsões ajustadas do período de treino das séries com muita variação dos valores máximos de caudal distribuído no período de verão. Este facto deixou muitas reservas na avaliação deste modelo porque se o período de previsão fosse o período de verão poderíamos ter previsões muito inferiores aos seus valores reais.

CAPÍTULO VI – CONCLUSÃO

Neste capítulo apresenta-se a análise global dos resultados obtidos, o modelo considerado mais adequado, referem-se as limitações do trabalho e apresentam-se propostas de trabalhos futuros.

6.1. Análise dos Resultados

Efetuando uma análise global aos resultados obtidos, concluiu-se que prazo de previsão teve um papel determinante nos resultados obtidos. Se por um lado, numa previsão de curto prazo, os modelos *Holt-Winters* e ARIMA, sendo modelos considerados clássicos, apresentaram melhores resultados, numa previsão de longo prazo estes modelos perdiam significativamente a sua precisão. Os modelos LSTM e *Prophet*, tiveram resultados mais coerentes entre os dois tipos de prazo, sendo ligeiramente melhores no longo prazo do que no curto prazo.

Os resultados das métricas obtidos apresentaram-se coerentes entre si, ou seja, o menor erro médio absoluto, correspondeu ao menor valor da raiz do erro quadrático médio e ao maior valor do coeficiente de determinação, pelo que se concluiu que os resultados obtidos apresentavam robustez.

O modelo ARIMA poderá ter sido prejudicado pelas características dos dados das séries em causa, razão pela qual terá ficado tão próximo do modelo *Holt-Winters* no curto prazo, tendo sido mesmo pior no longo prazo.

O modelo LSTM apresentou-se como aquele que melhor conseguiu respeitar a sazonalidade anual efetuando previsões com base no que tinha sucedido nos mesmos períodos do ano anterior. Apenas demostrou problemas do período de transição entre treino e teste não tendo sido possível obter bons resultados para uma previsão de curto prazo.

O modelo *Prophet* apresentou uma performance equivalente ao LSTM no que respeita ao cumprimento da sazonalidade anual, no entanto a limitação que apresentou na amplitude das previsões não lhe terá permitido ter melhores resultados.

Os modelos LSTM e *Prophet* apresentaram características muito importantes para a previsão de caudais distribuídos em redes de abastecimento. O facto de conseguirem respeitar não só a sazonalidade diária, mas também a anual, faz com que estes modelos sejam a chave para alguns problemas relacionados com monitorização de redes de abastecimento. Interessa não só prever o valor do caudal, mas mais importante ainda será, indicar, com base em iguais períodos de anos anteriores, qual o valor do caudal que deveria ter sido distribuído.

Para o modelo ARIMA e *Holt-Winters*, o facto de não terem apresentado capacidade de implementar uma sazonalidade anual, torna-os modelos de previsão precisos no curto prazo, mas imprecisos no longo prazo. Se, numa rede de distribuição de água, o número de roturas for sempre aumentando, eles não terão a capacidade de nos alertar por desconhecerem a necessidade de efetuar previsões ponderando o sucedido em períodos homólogos de anos anteriores.

6.2. Escolha do Modelo Mais Adequado

De acordo com os resultados obtidos o modelo considerado mais adequado para efetuar previsões de curto prazo foi o modelo ARIMA. Apesar dos problemas de autocorrelação evidenciados por este tipo de séries temporais, bem como do afastamento da sua distribuição em relação a uma distribuição normal, o modelo ARIMA apresentou, para o curto prazo, os melhores valores das métricas consideradas. Já para o caso das previsões de longo prazo, foi o modelo LSTM o que apresentou os melhores valores de métricas bem como a sua excelente capacidade em efetuar previsões respeitando a sazonalidade diária e anual. O modelo LSTM, comparado com o modelo *Prophet*, é também aquele que permite um maior controlo de afinação dos seus parâmetros. O modelo *Prophet* apresenta-se como um modelo que permite poucos ajustes, o que pode levar a resultados inesperados.

6.3. Limitações do Trabalho

Em primeiro lugar, importa fazer referência à limitação imposta pelo hardware utilizado que impediu a utilização das séries de Aldeia de Joanes, Degolados e Janeiro

de Cima, com resultados a cada 5 minutos. Esta limitação obrigou a uma conversão inicial da frequência destas séries para uma frequência horária.

Outra limitação importante diz respeito aos resultados, devido à aleatoriedade dos valores de caudal das séries temporais de Aldeia de Joanes e de Janeiro de Cima. O facto de serem localidades com uma população flutuante significativa, corrompe os padrões das séries tornando-as aleatórias e impede o bom funcionamento dos modelos. No caso da Aldeia de Joanes, a situação foi ainda pior devido à ocorrência de muitas roturas na rede durante o período de previsão que incluíram uma aleatoriedade muito significativa na série impedindo o funcionamento de qualquer modelo, mesmo em previsões de curto prazo. Estas roturas foram identificadas pela interpretação das curvas dos dados de treino e confirmadas nos registos de manutenção da rede.

6.4. Trabalhos Futuros

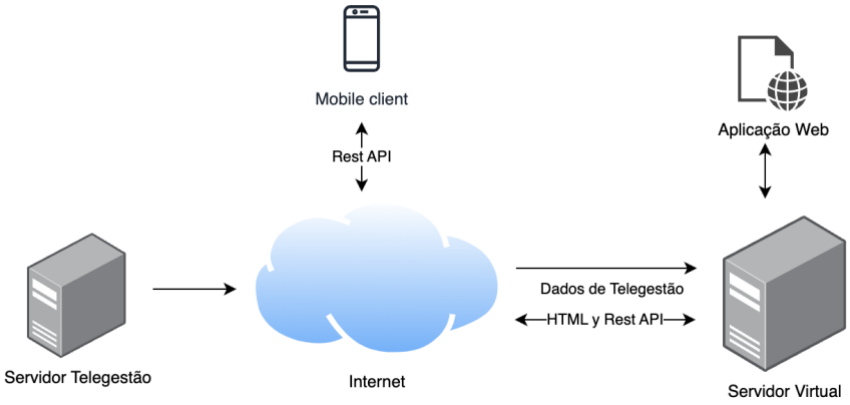
Como trabalhos futuros seria importante testar estes modelos com outras séries de setores de abastecimento para verificar a persistência dos resultados.

Uma vez testada e comprovada a sua persistência, seria interessante implementar um sistema de deteção de anomalias em redes de abastecimento de água baseado em previsões de caudais distribuídos e sua posterior comparação com os valores reais. Os caudais distribuídos em setores de redes de abastecimento tendem, com o passar do tempo, a ser sempre menores ou iguais aos do ano anterior. Este facto é indicador de uma boa gestão. Se os valores forem superiores, ou tiverem uma tendência de serem superiores, fora de uma margem de aceitação, deve ser considerada a possível ocorrência de uma anomalia. Acontece que nem sempre é possível ter pessoas a fazer comparações diretas entre valores atuais e valores passados, pelo que uma ferramenta deste tipo seria verdadeiramente útil.

Assim, como trabalho futuro, integrado na empresa FCC Aqualia, S.A., Sucursal em Portugal, propõe-se o desenvolvimento de um projeto piloto, que se assumiria como um protótipo, para deteção de anomalias que ocorram em redes de distribuição de água, baseado em alterações nos padrões dos caudais distribuídos. Este protótipo seria uma ferramenta informática composta por uma infraestrutura de base ao projeto piloto, que permitisse calcular e armazenar as previsões em tempo real ao longo dos

dias. Esta infraestrutura deveria contar com um servidor virtual, numa solução em *Cloud*, que tivesse uma cópia da base de dados do sistema de telegestão, pelo menos com os dados selecionados para o projeto, e um conjunto de *scripts* que, diariamente, copiassem os novos dados e calculassem uma previsão de até 3 meses, Figura 96. A frequência de previsão deveria ser de 1 hora mesmo que os dados disponíveis sejam mais detalhados. Esta frequência é aquela que, a experiência prática do terreno, nos tem mostrado ser mais facilmente assimilável e percetível, do ponto de vista da experiência humana, para o objetivo em causa.

Figura 96 – Infraestrutura de um Projeto Piloto de Deteção de Anomalias – Trabalho Futuro



Fonte: Elaboração Própria

Os dados reais e as previsões deveriam ser visualizáveis através de uma aplicação *Web* a desenvolver. Cada utilizador deveria ter uma conta que lhe permitiria entrar na aplicação e visualizar os dados, assim como configurar as suas preferências de notificações para a aplicação móvel.

A aplicação móvel seria desenvolvida para o efeito e disporia de um serviço de notificações de acordo com o configurado na aplicação *Web*. As notificações seriam possíveis anomalias detetadas pelo sistema de cada vez que o padrão de caudal distribuído ficasse fora de um intervalo de segurança definido pelo utilizador. O utilizador poderia interatuar com o sistema através da aplicação móvel, informando se se tratava de uma anomalia ou não, ou seja, se a variação era desconhecida ou, pelo contrário, teria sido causada por alguma intervenção na rede. Isto indicaria ao sistema se devia ou não considerar o valor anómalo nas previsões seguintes.

Depois de desenvolvido, o projeto piloto entraria numa fase de testes. Nesta fase seriam escolhidos vários utilizadores, responsáveis por vários setores de abastecimento a testar. O objetivo seria testar o sistema em condições reais de funcionamento, onde este se alimentaria diariamente de novos dados de caudal e atualizaria as suas previsões. Os utilizadores deveriam então consultar e configurar o seu grupo de setores e interatuar com a aplicação móvel testando a qualidade das notificações recebidas.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] V. A. Tzanakakis, N. V. Paranychianakis, e A. N. Angelakis, «Water supply and water scarcity», *Water (Switzerland)*, vol. 12, n. 9. MDPI AG, 2020. doi: 10.3390/w12092347.

[2] X. K. Bui, M. S. Marlim, e D. Kang, «Water network partitioning into district metered areas: A state-of-the-art review», *Water (Switzerland)*, vol. 12, n. 4. MDPI AG, 1 de Abril de 2020. doi: 10.3390/W12041002.

[3] A. Antunes, A. Andrade-Campos, A. Sardinha-Lourenço, e M. S. Oliveira, «Short-term water demand forecasting using machine learning techniques», *Journal of Hydroinformatics*, vol. 20, n. 6, pp. 1343–1366, 2018, doi: 10.2166/hydro.2018.163.

[4] A. R. S. Parmezan, V. M. A. Souza, e G. E. A. P. A. Batista, «Evaluation of statistical and machine learning models for time series prediction: Identifying the state-of-the-art and the best conditions for the use of each model», *Inf Sci (N Y)*, vol. 484, pp. 302–337, Mai. 2019, doi: 10.1016/j.ins.2019.01.076.

[5] M. Panda, «Application of ARIMA and Holt-Winters forecasting model to predict the spreading of COVID-19 for India and its states», doi: 10.1101/2020.07.14.20153908.

[6] A. Boudhaouia e P. Wira, «A Real-Time Data Analysis Platform for Short-Term Water Consumption Forecasting with Machine Learning», *Forecasting*, vol. 3, n. 4, pp. 682–694, Set. 2021, doi: 10.3390/forecast3040042.

[7] S. Siami-Namini, N. Tavakoli, e A. Siami Namin, «A Comparison of ARIMA and LSTM in Forecasting Time Series», em *Proceedings - 17th IEEE International Conference on Machine Learning and Applications, ICMLA 2018*, Institute of Electrical and Electronics Engineers Inc., Jan. 2019, pp. 1394–1401. doi: 10.1109/ICMLA.2018.00227.

[8] A. Tealab, «Time series forecasting using artificial neural networks methodologies: A systematic review», *Future Computing and Informatics Journal*, vol. 3, n. 2, pp. 334–340, Dez. 2018, doi: 10.1016/j.fcij.2018.10.003.

[9] S. Makridakis, E. Spiliotis, e V. Assimakopoulos, «Statistical and Machine Learning forecasting methods: Concerns and ways forward», *PLoS One*, vol. 13, n. 3, Mar. 2018, doi: 10.1371/journal.pone.0194889.

[10] B. Lim e S. Zohren, «Time-series forecasting with deep learning: A survey», *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 379, n. 2194. Royal Society Publishing, 5 de Abril de 2021. doi: 10.1098/rsta.2020.0209.

[11] A. Niknam, H. K. Zare, H. Hosseininasab, e A. Mostafaeipour, «Developing an LSTM model to forecast the monthly water consumption according to the effects of the climatic factors in Yazd, Iran», *Journal of Engineering Research*, vol. 11, n. 1, p. 100028, 2023, doi: 10.1016/j.jer.2023.100028.

[12] C. Kühnert, N. M. Gonuguntla, H. Krieg, D. Nowak, e J. A. Thomas, «Application of LSTM networks for water demand prediction in optimal pump control», *Water (Switzerland)*, vol. 13, n. 5, pp. 1–19, 2021, doi: 10.3390/w13050644.

[13] M. A. Al-Zahrani e A. Abo-Monasar, «Urban residential water demand prediction based on artificial neural networks and time series models», *Water Resources Management*, vol. 29, n. 10, pp. 3651–3662, Ago. 2015, doi: 10.1007/s11269-015-1021-z.

[14] A. Ticherahine, A. Boudhaouia, P. Wira, e A. Makhlouf, «Time series forecasting of hourly water consumption with combinations of deterministic and learning models in the context of a tertiary building», em *2020 International Conference on Decision Aid Sciences and Application, DASA 2020*, Institute of Electrical and Electronics Engineers Inc., Nov. 2020, pp. 116–121. doi: 10.1109/DASA51403.2020.9317176.

[15] S. Chaturvedi, E. Rajasekar, S. Natarajan, e N. McCullen, «A comparative assessment of SARIMA, LSTM RNN and Fb Prophet models to forecast total and peak monthly energy demand for India», *Energy Policy*, vol. 168, Set. 2022, doi: 10.1016/j.enpol.2022.113097.

[16] Y. Ning, H. Kazemi, e P. Tahmasebi, «A comparative machine learning study for time series oil production forecasting: ARIMA, LSTM, and Prophet», *Comput Geosci*, vol. 164, Jul. 2022, doi: 10.1016/j.cageo.2022.105126.

[17] D. Duarte e J. Faerman, «Comparison of Time Series Prediction of Healthcare Emergency Department Indicators with ARIMA and Prophet», Academy and Industry Research Collaboration Center (AIRCC), Dez. 2019, pp. 123–133. doi: 10.5121/csit.2019.91810.

[18] R. J. Hyndman e G. Athanasopoulos, «Forecasting: Principles and Practice - 7.3 Holt-Winters' seasonal method», https://otexts.com/fpp2. Acedido: 8 de Dezembro de 2023. [Em linha]. Disponível em: https://otexts.com/fpp2/holt-winters.html

[19] P. Banerjee, «ARIMA Model for Time Series Forecasting», https://www.kaggle.com/. Acedido: 8 de Dezembro de 2023. [Em linha]. Disponível em: https://www.kaggle.com/code/prashant111/arima-model-for-time-series-forecasting

[20] Francesca. Lazzeri, *Machine learning for time series forecasting with Python*. Wiley, 2021, ISBN10-1119682363.

[21] S. J. Taylor e B. Letham, «Forecasting at Scale», doi: 10.7287/peerj.preprints.3190v2.

[22] T. Chai e R. R. Draxler, «Root mean square error (RMSE) or mean absolute error (MAE)? - Arguments against avoiding RMSE in the literature», *Geosci Model Dev*, vol. 7, n. 3, pp. 1247–1250, Jun. 2014, doi: 10.5194/gmd-7-1247-2014.

[23] D. Chicco, M. J. Warrens, e G. Jurman, «The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation», *PeerJ Comput Sci*, vol. 7, pp. 1–24, 2021, doi: 10.7717/PEERJ-CS.623.

[24] Department of Statistics Online Programs, «The Pennsylvania State University», https://online.stat.psu.edu/statprogram/. Acedido: 8 de Novembro de 2023. [Em linha]. Disponível em: https://online.stat.psu.edu/stat462/node/188/

[25] Rob. J. Hyndman e G. Athanasopoulos, «Forecasting: Principles and Practice - time series components», https://otexts.com/fpp2. Acedido: 8 de Dezembro de 2023. [Em linha]. Disponível em: https://otexts.com/fpp2/components.html

[26] Ben. Auffarth, *Machine Learning for Time-Series with Python Forecast, Predict, and Detect Anomalies with State-Of-the-art Machine Learning Methods.* Packt Publishing, Limited, 2021, ISBN10-1801819629.

[27] Aniruddha Bhandari, «Feature Engineering: Scaling, Normalization, and Standardization (Updated 2023)», https://www.analyticsvidhya.com/blog. Acedido: 8 de Novembro de 2023. [Em linha]. Disponível em: https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/

[28] Dan Allison, «How to Encode the Cyclic Properties of Time with Python», https://medium.com. Acedido: 9 de Dezembro de 2023. [Em linha]. Disponível em: https://medium.com/@dan.allison/how-to-encode-the-cyclic-properties-of-time-with-python-6f4971d245c0

[29] R. Bevans, «Akaike Information Criterion | When & How to Use It (Example)», https://www.scribbr.com/statistics. Acedido: 2 de Setembro de 2023. [Em linha]. Disponível em: https://www.scribbr.com/statistics/akaike-information-criterion/

ANEXOS

ANEXO 1 – *Jupyter Notebooks* – Exemplo de Alcáçova

In [ ]:
```
# NoteBook:     pre-processingAlcacova.ipynb
# Autor:        Carlos Pires
# Data:         30-12-2023
# Descrição:    Este notebook efetua as operações necessárias ao pré-tratamento dos dados
#               relativos à série temporal de Alcáçova.
```

1. Pré-tratamento dos dados

In [ ]:
```
# Importação de Bibliotecas

import pandas as pd
from datetime import datetime
```

1.1. Pré-tratamento dos dados relativos ao sector de Alcáçova

Os dados relativos ao sector de Alcáçova encontram-se em 15 ficheiros csv que representam 15 meses, entre outubro de 2021 e dezembro de 2022. Entre os meses de outubro de 2021 e setembro de 2022 encontra-se um ano completo e representam os dados de treino. Já os meses de outubro a dezembro de 2022 representam os dados de teste. Os registos deste sector têm uma frequência horária pelo que é esperado que tenhamos (365 dias + 92 dias) x 24 registos = 457 dias x 24 registos = 10.968 dados. Este número é importante para validar a quantidade de dados que obtemos da leitura dos ficheiros pois existe a possibilidade de existência de anomalias na frequência dos registos que pode fazer variar este valor e que terão de ser resolvidas.

1.1.1. Leitura dos ficheiros

In [ ]:
```
# Definição de variáveis

files_path = '../dados/Alcacova/'
files_names = ('01-Alcacova-Outubro2021.csv',
               '02-Alcacova-Novembro2021.csv',
               '03-Alcacova-Dezembro2021.csv',
               '04-Alcacova-Janeiro2022.csv',
               '05-Alcacova-Fevereiro2022.csv',
               '06-Alcacova-Marco2022.csv',
               '07-Alcacova-Abril2022.csv',
               '08-Alcacova-Maio2022.csv',
               '09-Alcacova-Junho2022.csv',
               '10-Alcacova-Julho2022.csv',
               '11-Alcacova-Agosto2022.csv',
               '12-Alcacova-Setembro2022.csv',
               '13-Alcacova-Outubro2022.csv',
               '14-Alcacova-Novembro2022.csv',
               '15-Alcacova-Dezembro2022.csv'
               )
df_total = pd.DataFrame()
```

In [ ]:
```
# Ler todos os ficheiros para um único dataframe

for files in files_names:
    print(files_path + files)
    df = pd.read_csv(files_path + files, sep=';')
    df_total = pd.concat([df_total, df], axis=0, ignore_index=True)
```

In [ ]:
```
df_total.head()
```

In [ ]:
```
df_total.tail()
```

In [ ]:
```
df_total.info()
```

Da conjugação de todos os ficheiros com todos os dados de treino e teste obtém-se um dataframe com 10.970 registos. Tendo em conta que o número de registos esperado, 10.968 dados, isto já indica que existe

algum problema com a frequência de registo dos dados. Vamos começar por deixar ficar apenas as colunas Data/Hora e SAIDA, pois são estas que têm a informação que necessitamos. Quanto ao índice, nesta fase do pré-processamento será do tipo numérico e a Data/hora será tratada como uma coluna de dados.

1.1.2. Eliminação das colunas que não têm dados relevantes para o estudo

In [ ]:
```
# Eliminação das colunas que não contêm dados relevantes para o estudo em causa.

df_total = df_total.loc[:, ['Data/Hora', 'SAIDA']]
df_total.head()
```

In [ ]:
```
df_total.info()
```

Os dados Data/Hora e os dados de caudal (SAIDA) são do tipo Object. É necessário transformar os dados Data/hora para datetime e os dados de caudal para dados do tipo float. É necessária uma atenção especial porque os dados de caudal têm virgulas em vez de pontos pelo que será necessário trocar para pontos antes da conversão.

1.1.3. Formatação dos dados

In [ ]:
```
# Colocar os dados "Data/Hora" do tipo datetime

df_total['Data/Hora'] = pd.to_datetime(df_total['Data/Hora'], format='%d/%m/%y %H:%M')

# Transformar os dados de caudal em dados do tipo float

df_total['SAIDA'] = pd.to_numeric(df_total['SAIDA'].str.replace(',', '.'))

df_total.info()
```

In [ ]:
```
df_total.head()
```

Já temos os valores de Data/Hora do tipo datetime e os valores de caudal do tipo float, tal como era pretendido.

1.1.4. Teste da equidistância da série temporal e resolução de anomalias

Antes já tínhamos verificado que existia algum problema em relação ao número de registo de dados pois esperávamos ter 10.968 dados e temos 10.970. Para analisar e tentar encontrar o ou os problemas que temos na série temporal, iremos fazer um teste de equidistância entre as datas, ou seja, verificamos se todos os valores distam o mesmo ou se existem casos onde isso não se verifica.

In [ ]:
```
# Teste de equidistância

freq = pd.infer_freq(df_total['Data/Hora'])
print(freq)
```

A resposta ao teste é "None", o que significa que a nossa série temporal não é equidistante. para encontrarmos os problemas vamos adicionar uma coluna no dataframe com valores booleanos, onde True significa equidistancia de 1 hora e False significa problemas de equidistância.

In [ ]:
```
diff = pd.Series(df_total['Data/Hora']).diff()
equidistance = (diff == pd.Timedelta('1 hour'))

df_total['equidistance'] = equidistance
df_total.head()
```

Verificamos os casos em que temos equidistancia False:

In [ ]:
```
df_total.loc[df_total['equidistance'] == False]
```

O primeiro caso vamos desprezar uma vez que será sempre False pois não tem valor anterior para comparar. Temos então 7 valores a analisar. vamos verificar os valores da vizinhança de cada um para poder resolver

os problemas.

In [ ]:
```
df_total.loc[4245:4255]
```

Podemos verificar que falta um valor relativo à data 2022-03-27 02:00:00, passa da hora 01:00:00 para 03:00:00. Para resolver vamos calcular a média entre estes dois valores da vizinhança e adicionar o resultado no valor das 02:00:00. (5.833 + 4.444) / 2 = 5,1385.

In [ ]:
```
nova_data = datetime.strptime('2022-03-27 02:00:00', '%Y-%m-%d %H:%M:%S')

nova_linha = pd.DataFrame({'Data/Hora': [nova_data],
                           'SAIDA': [5.1385],
                           'equidistance': [False]})
df_total = pd.concat([df_total.loc[:4249],
                      nova_linha,
                      df_total.loc[4250:]]).reset_index(drop=True)
```

In [ ]:
```
df_total.loc[4245:4255]
```

RESOLVIDO. Vamos para o próximo Falso.

In [ ]:
```
df_total.loc[df_total['equidistance'] == False]
```

O próximo valor falso é o do índice 5488. Vamos verificar as suas vizinhanças:

In [ ]:
```
df_total.loc[5480:5495]
```

Neste caso trata-se de um valor a mais fruto de um reinicio do datalogger possivelmente. Basta apagá-lo.

In [ ]:
```
df_total.drop(index=5488, inplace=True)
df_total.reset_index(drop=True, inplace=True)
```

In [ ]:
```
df_total.loc[5480:5495]
```

RESOLVIDO. Vamos para o próximo.

In [ ]:
```
df_total.loc[df_total['equidistance'] == False]
```

In [ ]:
```
df_total[7568:7575]
```

Neste caso trata-se de um valor a mais fruto de um reinicio do datalogger possivelmente. Basta apagá-lo.

In [ ]:
```
df_total.drop(index=7570, inplace=True)
df_total.reset_index(drop=True, inplace=True)
```

In [ ]:
```
df_total.loc[df_total['equidistance'] == False]
```

Verificamos o próximo:

In [ ]:
```
df_total[10863:10868]
```

Novamente um valor a mais com caudal 0. Iremos remove-lo.

In [ ]:
```
df_total.drop(index=10864, inplace=True)
df_total.reset_index(drop=True, inplace=True)
```

In [ ]:
```
df_total.loc[df_total['equidistance'] == False]
```

Recalculamos a coluna equidistance e verificamos novamente a frequência:

In [ ]:
```
diff = pd.Series(df_total['Data/Hora']).diff()
equidistance = (diff == pd.Timedelta('1 hour'))

df_total['equidistance'] = equidistance
df_total.head()
```


In [ ]:
```
freq = pd.infer_freq(df_total['Data/Hora'])
print(freq)
```

Agora sim temos frequência horária.

Verificamos os registos com equidistância False:

In [ ]:
```
df_total.loc[df_total['equidistance'] == False]
```

Apenas indica o primeiro registo porque não tem um registo anterior para comparar.

In [ ]:
```
df_total.info()
```

Confirma-se que temos 10.968 registos, tal como tinha sido previsto. Vamos eliminar a coluna equidistance e ficamos apenas com a Data/Hora e SAIDA.

In [ ]:
```
# Eliminação das colunas que não contêm dados relevantes para o estudo em causa.
df_total = df_total.loc[:, ['Data/Hora', 'SAIDA']]
df_total.head()
```

Normalmente nestas séries os valores com caudal 0 são valores que devem ser verificados porque podem resultar de um reinício do datalogger e não devem ser considerados. Assim fazemos uma verificação final procurando zeros:

In [ ]:
```
df_total.loc[df_total['SAIDA'] == 0]
```

1.1.5. Exportação dos dados

No final exportamos os dados para um único ficheiro:

In [ ]:
```
df_total.to_csv("../DadosPreProcessados/Alcacova.csv", index=False)
```

In [ ]:
```
# NoteBook:     Analise-Alcacova.ipynb
# Autor:        Carlos Pires
# Data:         30-12-2023
# Descrição:    Este notebook efetua as operações necessárias para realizar a análise
#               dos dados existentes relativos à série temporal de Alcáçova.
```

2. Análise dos Dados

In [ ]:
```
# Importação de bibliotecas

import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 15, 6
import plotly.graph_objects as go
import statsmodels.tsa.stattools
from statsmodels.tsa.stattools import kpss
from statsmodels.tsa.stattools import adfuller
from arch.unitroot import PhillipsPerron
```

Definição de funções de apoio

In [ ]:
```
# Criar o gráfico interativo com a biblioteca Plotly
def graph(serie, data, nome_serie):

    fig = go.Figure()
    fig.add_trace(go.Scatter(x=serie.index, y=getattr(serie, data), mode='lines'))

    # Configurar os eixos e o layout do gráfico
    fig.update_layout(
        xaxis=dict(title='Data'),
        yaxis=dict(title='Caudal (l/s)'),
        title=nome_serie
    )

    # Habilitar o zoom interativo
    fig.update_layout(
        xaxis=dict(
            rangeselector=dict(
                buttons=list([
                    dict(count=1, label='1m', step='month', stepmode='backward'),
                    dict(count=6, label='6m', step='month', stepmode='backward'),
                    dict(count=1, label='1y', step='year', stepmode='backward'),
                    dict(count=3, label='3y', step='year', stepmode='backward'),
                    dict(step='all')
                ])
            ),
            rangeslider=dict(visible=True),
            type='date'
        )
    )

    # Mostrar o gráfico
    fig.show()
```

In [ ]:
```
# Função de teste KPSS
def kpss_test(timeseries):
    print ('Results of KPSS Test:')
    kpsstest = kpss(timeseries, regression='ct', nlags="auto")
    kpss_output = pd.Series(kpsstest[0:3], index=['Test Statistic','p-value','#Lags Used'])
    for key,value in kpsstest[3].items():
        kpss_output['Critical Value (%s)'%key] = value
    print (kpss_output)
```

In [ ]:
```
# Função de teste Augmented Dickey-Fuller

def adf_test(timeseries):
    print ('Results of Augmented Dickey-Fuller Test::')
```



```
    dftest = adfuller(timeseries, regression='ct', autolag='AIC')
    df_output = pd.Series(dftest[0:3], index=['Test Statistic','p-value','#Lags Used'])
    for key, value in dftest[4].items():
        df_output['Critical Value (%s)'%key] = value
    print (df_output)
```

In [ ]:
```
# Função de teste Phillips-Perron

def pp_test(timeseries):
    result = PhillipsPerron(timeseries, trend='ct')
    print(result)
```

Importação de Dados

In [ ]:
```
serie_alcacova = pd.read_csv('../DadosPreProcessados/Alcacova.csv',
                             sep=',',
                             index_col='Data/Hora',
                             parse_dates=True
                )
serie_alcacova
```

2.1. Análise da série temporal de Alcáçova

2.1.1. Estacionaridade

A Estacionaridade é um pressuposto importante no estudo de séries temporais porque diversos modelos de séries temporais têm como pressuposto a estacionaridade, ou seja, para que esses modelos consigam fazer previsões sobre determinada série, é necessário que esta se comporte de forma estacionária.

As séries que se desenvolvem aleatoriamente no tempo mantêndo uma média e variância constantes são chamadas de estacionárias. As não estacionárias são aquelas que mudam o seu comportamento evidenciando inclinações ou mudanças de nível.

Em alguns casos é possível transformar uma série não estacionária em estacionária utilizando transformações logaritmicas ou exponenciais (método Box-Cox).

2.1.1.1. Verificação da Estacionaridade Graficamente

In [ ]:
```
graph(serie_alcacova, "SAIDA", 'Série temporal de Alcáçova')
```

Por observação do gráfico da série temporal percebe-se que nem a média nem a variância são constantes ao longo do tempo. Existe um aumento da média mais pronunciado com um pico em agosto e que depois cai de forma abrupta iniciando novamente um novo ciclo de subida muito ligeira.

Possivelmente isto deve-se à ocorrência de uma rotura da rede que não é detetável e por isso não é reparada inicialmente. Ao longo do tempo vai aumentando o caudal até chegar ao ponto em que é visível à superfície e é reparada ou é detetada utilizando ferramentas auxiliares como Geofónes.

De qualquer forma são propriedades típicas deste tipo de séries e teremos de lidar com elas.

2.1.1.2. Teste KPSS

Teste KPSS (Kwiatkowski-Phillips-Schmidt-Shin)

O teste KPSS é um teste de raíz unitária que verifica a estacionaridade de uma determinada série temporal em torno de uma tendência determinística.

Hípóteses do teste KPSS:

- H0 = a série é estacionária: estatística do teste < valor crítico (rejeitar se p-value < nível de significância)
- HA = a série não estacionária

Nível de significância: 0.05

In [ ]:
```
kpss_test(serie_alcacova);
```

Análise do resultado do teste KPSS:

Neste caso o p-value é menor que o nível de significância escolhido, pelo que só por si nos leva a rejeitar a hipótese nula e a aceitar a hipótese alternativa de que a série é não estacionária. Por outro lado a estatística do teste também tem um valor superior ao do valor crítico de 5%.

2.1.1.3. Teste Augmented Dickey-Fuller

O teste de Augmented Dickey-Fuller (ADF) é um teste estatístico utilizado para verificar se uma série temporal é estacionária ou não. Trata-se de uma extensão do teste de raiz unitária, que verifica se uma série temporal tem uma raiz unitária ou não. Uma raiz unitária indica que a série não é estacionária e a sua ausência indica que a série é estacionária. O teste Augmented Dickey-Fuller utiliza um modelo autoregressivo para estimar a presença ou ausência de uma raiz unitária na série.

Hipóteses do Teste Augmented Dickey-Fuller:

- H0 = não é estacionária: rejeitar se (estatística do teste < valor crítico);
- HA = é estacionária.

Nível de significância: 5%

In [ ]:
```
adf_test(serie_alcacova)
```

Análise do resultado do teste Augmented Dickey-Fuller:

- O valor da estatística do teste é -5.087758 o que é menor do que o valor crítico de 5% (-3.410892);

Pela razão acima exposta deve ser rejeitada a hipótese nula, pelo que se conclui que a série é estacionária.

2.1.1.4. Teste Philips-Perron (PP)

O teste Phillips-Perron é um teste de raíz unitária que utiliza um estimador de kernel robusto para reduzir o efeito da autocorrelação. As hipóteses do teste PP são:

- Hipótese nula (H0): A série é não estacionária: contém uma raíz unitária (rejeitar se p-value < nível de significância);
- Hipótese alternativa (H1): A série é estacionária.

Nível de significância: 0.05

In [ ]:
```
pp_test(serie_alcacova)
```

Análise do resultado do teste Phillips-Perron:

- o p-value é menor do que o nível de significância pelo que se deve rejeitar a hipótese nula e aceitar a alternativa, pelo que a serie é estacionária.

A maioria dos resultados dos testes numéricos de estacionaridade indicam que a série é estacionária, no entanto, tendo em conta que um dos testes, o KPSS, indica que a série é não estacionária e os outros dois indicam que é estacionária, deve ser considerada uma série com uma fraca estacionaridade.

2.1.2. Autocorrelação

In [ ]:
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

A verificação da existencia de autocorrelação numa série temporal é importante porque verifica a existência de dependência dos dados de uma série entre si, ou seja, verifica se ao longo do tempo, entre determinadas

lags, os dados estão dependentes entre si, o que pode ser um problema para alguns modelos de séries temporais.

2.1.2.1. Diagrama ACF

Trata-se do diagrama de autocorrelação que verifica a existencua de dependência entre lags adjacentes.

In [ ]:
```
fig, ax = plt.subplots(figsize=(15, 6))

plot_acf(serie_alcacova, ax=ax, lags=28)

ax.set_xlabel('Lags')
ax.set_ylabel('Correlation')
ax.set_title('Autocorrelation Function (ACF)')
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

plt.show()
```

Da análise do gráfico pode concluir-se que existe uma forte autocorrelação nos dados.

2.1.2.2. Diagrama PACF

Parecido com o anterior mas agora testa a dependência dos lags que não são adjacentes.

In [ ]:
```
fig, ax = plt.subplots(figsize=(15, 6))

plot_pacf(serie_alcacova, ax=ax, lags=28, method='ywm')

ax.set_xlabel('Lags')
ax.set_ylabel('Correlation')
ax.set_title('Partial Autocorrelation Function (PACF)')
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

plt.show()
```

O gráfico de autocorrelação parcial também indica autocorrelação entre os dados.

2.1.3. Decomposição

A decomposição da série temporal pretende analisar propriedades como a Tendência, a Sazonalidade e os Resíduos.

In [ ]:
```
# Importação da biblioteca que permite efetuar a decomposição

from statsmodels.tsa.seasonal import seasonal_decompose
```

In [ ]:
```
decompose = seasonal_decompose(serie_alcacova, period=24, model='additive')
plt.subplot(411)
plt.plot(serie_alcacova, label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(decompose.trend, label='Tendência')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(decompose.seasonal,label='Sazonalidade')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(decompose.resid, label='Resíduos')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

Na decomposição da série foi utilizada uma frequencia de 24 uma vez que temos 1 registo por hora e é espectável a existência de uma sazonalidade a cada 24 horas.

Nota-se a existência de uma tendência crescente até agosto de 2022, data a partir da qual existe uma diminuição da mesma e depois, a longo prazo, aparenta tornar-se mais ou menos constante.

No que respeita a sazonalidade apenas se pode concluir que existe, mas por dificuldade de visualização do gráfico é difícil conluir se se trata apenas de uma sazonalidade diária ou se existem outros periodos possíveis de identificar.

O último gráfico é o grafico dos resíduos.

2.1.3.1. Análise dos Resíduos

Por observação dos gráficos do ponto anterior verifica-se que o gráfico dos resíduos aparenta seguir algum tipo de padrão. Assim, iremos verificar a autocorrelação dos mesmos.

In [ ]:
```
decompose = seasonal_decompose(serie_alcacova, period=24, model='additive')
residuals = decompose.resid
for value in residuals:
    print(value)
```

In [ ]:
```
# Testar autocorrelação dos resíduos

fig, ax = plt.subplots(figsize=(16, 8))
plot_acf(residuals.dropna(), ax=ax, lags=28)
ax.set_xlabel('Lags')
ax.set_ylabel('Autocorrelation')
ax.set_title('Autocorrelation Function of Residuals')
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
plt.show()
```

In [ ]:
```
# Testar autocorrelação parcial dos resíduos

fig, ax = plt.subplots(figsize=(16, 8))
plot_pacf(residuals.dropna(), ax=ax, lags=28, method='ywm')
ax.set_xlabel('Lags')
ax.set_ylabel('Autocorrelation')
ax.set_title('Autocorrelation Function of Residuals')
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
plt.show()
```

Confirma-se assim que os resíduos têm uma forte autocorrelação.

2.1.4. Análise da Normalidade

In [ ]:
```
serie = pd.Series(serie_alcacova['SAIDA'].values, index=serie_alcacova.index)
```

In [ ]:
```
# Importação de bibliotecas em falta

import scipy.stats as stats
import numpy as np
```

2.1.4.1. Verificação da Normalidade Graficamente

In [ ]:
```
stats.probplot(serie, dist='norm', plot=plt)
plt.title("Normal QQ plot")
plt.show()
```

A série demonstra problemas de normalidade no início e no final.

2.1.4.2. Teste de Shapiro-Wilk

O teste de Shapiro-Wilk é um teste estatítico utilizado para verificar se um determinado conjunto de dados segue uma distribuição normal. As hipóteses do teste são:

- Hipótese nula (H0): Os dados seguem uma distribuição normal. Rejeitar se p-value < nível de significância;
- Hipótese alternativa (H1): Os dados não seguem uma distribuição normal.

Nível de significância: 0.05

In [ ]:
```
e, p = stats.shapiro(serie)
print('Estatística do teste: {}'.format(e))
print('p-valor: {}'.format(p))
```

Análise do resultado do teste:

- p-value é menor do que 0.05, pelo que se rejeita a H0 => Os dados não seguem uma distribuição normal.

2.1.4.3. Teste Anderson-Darling

O teste de Anderson-Darling é um teste estatístico para verificar se um determinado conjunto de dados segue uma distribuição normal. As hipóteses do teste são:

- Hipótese nula (H0): Os dados seguem uma distribuição normal. Rejeitar se estatítica do teste > valor crítico;
- Hipótese alternativa (H1): Os dados não seguem uma distribuição normal.

Nível de significância: 0.05

In [ ]:
```
# Importação da biblioteca para efetuar o teste de Anderson-Darling

from scipy.stats import anderson
```

In [ ]:
```
result = anderson(serie)
```

In [ ]:
```
print('Test statistic: %.3f' % result.statistic)
print('Critical values:')
for i in range(len(result.critical_values)):
    sl, cv = result.significance_level[i], result.critical_values[i]
    print('%.1f%%: %.3f' % (sl, cv))
```

Análise do resultado do teste:

- O valor da estatística do teste é maior do que o valor crítico de 5%, pelo que se conclui que os dados da série não seguem uma distribuição normal.

2.1.4.4 Transformação Logarítmica

A transformação por logarítmica vai permitir diminuir a variância e melhorar a normalidade

In [ ]:
```
serielog = np.log(serie)
serielog
```

In [ ]:
```
# Verificação do gráfico da Normal QQ plot após transformação logarítmica

stats.probplot(serielog, dist='norm', plot=plt)
plt.title("Normal QQ plot")
plt.show()
```

Teste de normalidade após transformação logarítmica

In [ ]:
```
result = anderson(serielog)
print('Test statistic: %.3f' % result.statistic)
print('Critical values:')
for i in range(len(result.critical_values)):
    sl, cv = result.significance_level[i], result.critical_values[i]
    print('%.1f%%: %.3f' % (sl, cv))
```

Análise do resultado da transformação logarítmica:

Pode concluir-se que a transformação logarítmica não só não melhora a aderência da distribuição dos dados da série a uma distribuição normal como ainda piora a situação inicial.

2.1.4.5. Transformação por raiz cúbica

In [ ]:
```
serie_raizc = (serie)**(1/3)
serie_raizc
```

In [ ]:
```
# Verificação do gráfico da Normal QQ plot após transformação por raiz cúbica

stats.probplot(serie_raizc, dist='norm', plot=plt)
plt.title("Normal QQ plot")
plt.show()
```

Teste de normalidade após a transfomação por raiz cúbica:

In [ ]:
```
result = anderson(serie_raizc)
print('Test statistic: %.3f' % result.statistic)
print('Critical values:')
for i in range(len(result.critical_values)):
    sl, cv = result.significance_level[i], result.critical_values[i]
    print('%.1f%%: %.3f' % (sl, cv))
```

Análise do resultado da transformação por raiz cúbica:

Pode concluir-se que a transformação por raiz cúbica não só não melhora a aderência da distribuição dos dados da série a uma distribuição normal como ainda piora a situação inicial.

2.1.4.6. Análise utilizando a biblioteca seaborn

In [ ]:
```
import seaborn as sns

# Histograma de distribuição da série original

sns.histplot(serie, kde=True);
```

In [ ]:
```
# Histograma de distribuição da série após transformação logarítmica

sns.histplot(serielog, kde=True);
```

In [ ]:
```
# Histograma de distribuição da série após transformação por raiz cúbica

sns.histplot(serie_raizc, kde=True);
```

Os gráficos evidenciam claramente que as transformações efetuadas apenas pioram a condição dos dados iniciais.

2.1.5. Diferenciação

In [ ]:
```
serie_diff = np.diff(serie)
```

In [ ]:
```
# Gráfico da série original após diferenciação.

plt.plot(serie_diff)
plt.show()
```

2.1.5.1. Testes de estacionaridade após diferenciação

Teste KPSS (Kwiatkowski-Phillips-Schmidt-Shin)

O teste KPSS é um teste de raíz unitária que verifica a estacionaridade de uma determinada série temporal em torno de uma tendência determinística.

Hípóteses do teste KPSS:

- H0 = a série é estacionária: estatística do teste < valor crítico (rejeitar se p-value < nível de significância)
- HA = a série não estacionária

Nível de significância: 0.05

In [ ]:
```
kpss_test(serie_diff)
```

Hipóteses do Teste Augmented Dickey-Fuller:

- H0 = não é estacionária: rejeitar se (estatística do teste < valor crítico);
- HA = é estacionária.

Nível de significância: 5%

In [ ]:
```
adf_test(serie_diff)
```

O teste Phillips-Perron é um teste de raíz unitária que utiliza um estimador de kernel robusto para reduzir o efeito da autocorrelação. As hipóteses do teste PP são:

- Hipótese nula (H0): A série é não estacionária: contém uma raíz unitária (rejeitar se p-value < nível de significância);
- Hipótese alternativa (H1): A série é estacionária.

Nível de significância: 0.05

In [ ]:
```
pp_test(serie_diff)
```

Graficamente verifica-se que a estacionaridade melhora, os resultados dos testes estatíticos alteram-se no que respeita às suas conclusões, ou seja: todos indicam agora que a série é estacionária.

In [ ]:
```
# NoteBook:     Holt-Winters-Alcacova.ipynb
# Autor:        Carlos Pires
# Data:         30-12-2023
# Descrição:    Este notebook efetua as operações necessárias para aplicar o Modelo
#               Holt-Winters à série temporal de Alcáçova.
```

3. Modelo Holt-Winters

In [ ]:
```
# importação de bibliotecas

import pandas as pd
import numpy as np
# graphics object
import plotly.graph_objects as go
# Holt-Winters model
from statsmodels.tsa.holtwinters import ExponentialSmoothing
# Statistic metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
# grid view formatting
from tabulate import tabulate
```

Definição de funções de apoio

In [ ]:
```
# Criar o gráfico interativo com a biblioteca Plotly para qualquer numero de dataframes

def graph(title_name="NO NAME", *args):

    fig = go.Figure()

    for i, arg in enumerate(args):
        dataframe = arg[0]
        attribute = arg[1]
        serie_name = arg[2]
        fig.add_trace(go.Scatter(x=dataframe.index,
                                 y=getattr(dataframe, attribute),
                                 name=serie_name,
                                 mode='lines',
                                 showlegend=True
                                 )
                      )

    # Configurar os eixos e o layout do gráfico
    fig.update_layout(
        xaxis=dict(title='Data'),
        yaxis=dict(title='Caudal (l/s)'),
        title=title_name
    )

    # Habilitar o zoom interativo
    fig.update_layout(
        xaxis=dict(
            rangeselector=dict(
                buttons=list([
                    dict(count=1, label='1m', step='month', stepmode='backward'),
                    dict(count=6, label='6m', step='month', stepmode='backward'),
                    dict(count=1, label='1y', step='year', stepmode='backward'),
                    dict(count=3, label='3y', step='year', stepmode='backward'),
                    dict(step='all')
                ])
            ),
            rangeslider=dict(visible=True),
            type='date'
        )
    )

    # Mostrar o gráfico
    fig.show()
```


In [ ]:
```
# Função de cálculo e apresentação de métricas

def evaluate_metrics(train_data, train_data_fitted, test_data, predictions):

    mae_train = mean_absolute_error(train_data, train_data_fitted)
    mae_predict = mean_absolute_error(test_data, predictions)

    rmse_train = np.sqrt(mean_squared_error(train_data, train_data_fitted))
    rmse_predict = np.sqrt(mean_squared_error(test_data, predictions))

    r2_train = r2_score(train_data, train_data_fitted)
    r2_predict = r2_score(test_data, predictions)

    data = {
        'Dados': ['Treino', 'Teste'],
        'MAE': [mae_train, mae_predict],
        'RMSE': [rmse_train, rmse_predict],
        'R2': [r2_train, r2_predict]
    }

    df_evaluate = pd.DataFrame(data)

    print('MÉTRICAS DE AVALIAÇÃO DO MODELO')
    print(tabulate(df_evaluate, headers='keys', tablefmt='pretty', showindex=False))
```

3.1. Holt-Winters - Série Temporal de Alcáçova

O modelo de Holt-Winters foi desenvolvido a partir do modelo de Holt com o objetivo de resolver o problema da existencia de sazonalidade nos dados. O modelo é composto pela equação de previsão e três equações de suavização: uma para o nível l_t, uma para a tendência b_t e uma terceira para a sazonalidade s_t com os parâmetros de suavizaçaão correspondentes α, β e γ. É utilizado m para caracterizar a frequência da sazonalidade.

Existem duas variações neste método que diferem na natureza da componente sazonal. O método aditivo é mais indicado quando as variações sazonais são aproximadamente constantes ao longo da série, já o multiplicativo deve ser utilizado quando as variações sazonais vão mudando de forma proporcional ao nível da série.

Com o método aditivo, a componente sazonal é expressa em termos absolutos na escala da série observada, e na equação de nível, a série é ajustada sazonalmente através da subtração da componente sazonal. Dentro de cada ano, a componente sazonal somará aproximadamente zero. Com o método multiplicativo, a componente sazonal é expressa em termos relativos (percentagens), e a série é ajustada sazonalmente dividindo-a pela componente sazonal. Dentro de cada ano, a componente sazonal somará aproximadamente m.

3.1.1. Holt-Winters - método aditivo

Equação de previsão:

- $\hat{y}_{t+h|t} = l_t + hb_t + s_{t+h-m(k+1)}$

Equação de nível:

- $l_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1})$

Equação de tendência:

- $b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1}$

Equação de sazonalidade:

- $s_t = \gamma(y_t - l_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m},$

onde k é a parte inteira de $(h-1)/m$, o que garante que as estimativas dos indices sazonais utilizados

para a previsão vêm do final do ano da amostra.

A equação de nível mostra uma média ponderada entre a sazonalidade ajustada à observação $(y_t - s_{t-m})$ e a previsão não sazonal $(l_{t-1} + b_{t-1})$ para o periodo t.

A equação da tendência mostra que b_t é uma média ponderada da tendência estimada para o periodo t baseada em $l_t - l_{t-1}$ e b_{t-1}, a estimativa anterior da tendência.

A equação de sazonalidade mostra uma média ponderada entre o índice de sazonalidade atual, $(y_t - l_{t-1} - b_{t-1})$, e o índice de sazonalidade da mesma estação no ano anterior, ou seja, m periodos de tempo atrás.

3.1.2. Holt-Winters - método multiplicativo

Equação de previsão:

- $\hat{y}_{t+h|t} = (l_t + hb_t)s_{t+h-m(k+1)}$

Equação de nível:

- $l_t = \alpha\frac{y_t}{s_{t-m}} + (1-\alpha)(l_{t-1} + b_{t-1})$

Equação de tendência:

- $b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1}$

Equação de sazonalidade:

- $s_t = \gamma\frac{y_t}{(l_{t-1}+b_{t-1})} + (1-\gamma)s_{t-m}$

3.1.3. Importação dos dados

In [ ]:
```
df = pd.read_csv('../DadosPreProcessados/Alcacova.csv',
                 sep=',',
                 index_col='Data/Hora',
                 parse_dates=True
                 )
```

In [ ]:
```
df
```

In [ ]:
```
# Apresentação do gráfico com os dados importados

graph("Série Temporal de Alcáçova - Caudal de Saída (l/s)",
      (df, "SAIDA", "Dados reais de caudal")
    )
```

3.1.4. Separação dos dados entre treino e teste

Os dados de treino e de teste serão separados da seguinte forma:

- De 01/10/2021 até 30/09/2022 serão dados de treino;
- De 01/10/2022 até 31/12/2022 serão os dados de teste.

Os dados são obtidos a partir do dataframe inicial para dois objetos do tipo "panda series".

In [ ]:
```
# Dividir os dados em conjuntos de treino e de teste

train_data = df[df.index <= '2022-09-30 23:00:00']['SAIDA']
test_data = df[df.index > '2022-09-30 23:00:00']['SAIDA']
```

In [ ]:
```
# Apresentação do gráfico com a divisão realizada

graph("Série Temporal de Alcáçova - Caudal de Saída (l/s)",
      (pd.DataFrame(train_data, index=train_data.index), "SAIDA", "Dados de treino"),
```



```
      (pd.DataFrame(test_data, index=test_data.index), "SAIDA", "Dados de teste"))
```

3.1.5. Ajuste e aplicação do modelo Holt-Winters aos dados

In [ ]:
```
# Holt-Winters model

# Parameters

trend = 'add'
seasonal = 'add'
seasonal_periods = 24
freq = 'H'
initialization_method = 'heuristic'
use_BoxCox = False

model = ExponentialSmoothing(
    train_data,
    trend=trend,
    seasonal=seasonal,
    seasonal_periods=seasonal_periods,
    freq=freq,
    initialization_method=initialization_method,
    use_boxcox=use_BoxCox
)

model_fit = model.fit()
```

3.1.5.1. Obtenção dos dados previstos pelo modelo

In [ ]:
```
# Obtenção dos dados de treino ajustados pelo modelo

train_data_fitted = model_fit.fittedvalues
train_data_fitted.name = "SAIDA"

# previsões para o conjunto de teste

predictions = model_fit.forecast(len(test_data))
predictions.name = "SAIDA"
```

3.1.5.2. Visualização gráfica dos dados

In [ ]:
```
graph("Série Temporal de Alcáçova",
      (pd.DataFrame(train_data,
                    index=train_data.index),
                    "SAIDA",
                    "Dados de treino reais"),
      (pd.DataFrame(train_data_fitted,
                    index=train_data.index),
                    "SAIDA",
                    "Dados de treino ajustados"),
      (pd.DataFrame(test_data,
                    index=test_data.index),
                    "SAIDA",
                    "Dados de teste reais"),
      (pd.DataFrame(predictions,
                    index=test_data.index),
                    "SAIDA",
                    "Dados previstos")
    )
```

3.1.5.3. Avaliação do Modelo

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_data_fitted,
                 test_data=test_data,
                 predictions=predictions)
```

3.1.6. Ajuste e aplicação do modelo Holt-Winters utilizando uma

transformação logarítmica

In [ ]:
```
train_data_log = np.log1p(train_data)
```

In [ ]:
```
# Ajuste  do modelo Holt-Winters

# Parameters

trend = 'add'
seasonal = 'add'
seasonal_periods = 24
freq = 'H'
initialization_method = 'heuristic'
use_BoxCox = 0.05
damped_trend = False

model_log = ExponentialSmoothing(
    train_data_log,
    trend=trend,
    seasonal=seasonal,
    seasonal_periods=seasonal_periods,
    freq=freq,
    initialization_method=initialization_method,
    use_boxcox=use_BoxCox,
    damped_trend=damped_trend
)

model_log_fit = model_log.fit()
```

Utilizando a transformação por log é necessário utilizar o parâmetro use_boxcox o qual se experimentaram vários valores até se ter chegado ao 0.05.

3.1.6.1. Obtenção dos dados previstos pelo modelo

In [ ]:
```
# Obtenção dos dados de treino ajustados pelo modelo

train_data_log_fitted = model_log_fit.fittedvalues
train_data_log_fitted.name = "SAIDA"

# previsões para o conjunto de teste

predictions_log = model_log_fit.forecast(len(test_data))
predictions_log.name = "SAIDA"
```

In [ ]:
```
# Como foram utilizadas transformações deve ser verificado se algum valor ficou como NaN

find_null = train_data_log_fitted.isna().idxmax()
print(find_null)
```

3.1.6.2. Reversão da transformação

In [ ]:
```
train_data_logreverted_fitted = np.expm1(train_data_log_fitted)
predictions_logreverted = np.expm1(predictions_log)
```

3.1.6.3. Visualização dos gráfica dos dados

In [ ]:
```
graph("Série Temporal de Alcáçova",
        (pd.DataFrame(train_data,
                      index=train_data.index),
                      "SAIDA",
                      "Dados de treino reais"),
        (pd.DataFrame(train_data_logreverted_fitted,
                       index=train_data_logreverted_fitted.index),
                       "SAIDA",
                       "Dados de treino ajustados"),
        (pd.DataFrame(test_data,
                       index=test_data.index),
                       "SAIDA",
                       "Dados de teste reais"),
```



```
        (pd.DataFrame(predictions_logreverted,
                      index=test_data.index),
                      "SAIDA",
                      "Dados previstos")
    )
```

3.1.6.4. Avaliação do modelo

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_data_logreverted_fitted,
                 test_data=test_data,
                 predictions=predictions_logreverted
                 )
```

O modelo melhora significativamente quando se aplica uma transformação por log. Ainda assim é necessário igualar o parâmetro use_BoxCox a 0.05, de outra forma os resultados são maus. Neste caso o modelo consegue explicar cerca de 80% dos dados de teste reais.

3.1.7. Ajuste e aplicação do modelo Holt-Winters utilizando uma transformação por raiz cúbica.

In [ ]:
```
train_data_raizc = (train_data)**(1/3)
train_data_raizc
```

In [ ]:
```
# Ajuste  do modelo Holt-Winters

# Parameters

trend = 'add'
seasonal = 'add'
seasonal_periods = 24
freq = 'H'
initialization_method = 'heuristic'
use_BoxCox = 0.05
damped_trend = False

model_raizc = ExponentialSmoothing(
    train_data_raizc,
    trend=trend,
    seasonal=seasonal,
    seasonal_periods=seasonal_periods,
    freq=freq,
    initialization_method=initialization_method,
    use_boxcox=use_BoxCox,
    damped_trend=damped_trend
)

model_raizc_fit = model_raizc.fit()
```

3.1.7.1. Obtenção dos dados previstos pelo modelo

In [ ]:
```
# Obtenção dos dados de treino ajustados pelo modelo

train_data_raizc_fitted = model_raizc_fit.fittedvalues
train_data_raizc_fitted.name = "SAIDA"

# previsões para o conjunto de teste

predictions_raizc = model_raizc_fit.forecast(len(test_data))
predictions_raizc.name = "SAIDA"
```

3.1.7.2. Reversão da transformação

In [ ]:
```
train_data_raizc_reverted_fitted = train_data_raizc_fitted ** 3
predictions_raizc_reverted = predictions_raizc ** 3
```

3.1.7.3. Visualização gráfica dos dados

In [ ]:
```
graph("Série Temporal de Alcáçova",
      (pd.DataFrame(train_data,
                    index=train_data.index),
                    "SAIDA",
                    "Dados de treino reais"),
       (pd.DataFrame(train_data_raizc_reverted_fitted,
                     index=train_data_raizc_reverted_fitted.index),
                     "SAIDA",
                     "Dados de treino ajustados"),
       (pd.DataFrame(test_data,
                     index=test_data.index),
                     "SAIDA",
                     "Dados de teste reais"),
       (pd.DataFrame(predictions_raizc_reverted,
                     index=test_data.index),
                     "SAIDA",
                     "Dados previstos")
    )
```

3.1.7.4. Avaliação do modelo

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_data_raizc_reverted_fitted,
                 test_data=test_data,
                 predictions=predictions_raizc_reverted
                 )
```

A transformação por raiz cubica não conduz a melhores resultados do que a transformação logaritmica.

3.1.8. Análise do resultado para previsões de curto prazo (10 dias)

Considerou-se importante verificar a performance dos modelos para um período mais curto uma vez que períodos muito longos podem potenciar a ocorrência de erros de previsão com a consequente desistência do modelo em causa quando o mesmo pode fazer boas previsões de curto prazo e, em algumas situações, serão essas as mais importantes. Considerou-se que o período mais curto seria de 10 dias pois do ponto de vista de análise e de desenvolvimento futuro deste trabalho será esse o período mínimo.

Assim sendo, uma vez que temos um registo por hora, estamos interessados em comparar os primeiros 24x10 = 240 valores.

3.1.8.1. Avaliação do modelo para uma previsão de curto prazo

Dados originais:

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_data_fitted,
                 test_data=test_data.iloc[:240],
                 predictions=predictions.iloc[:240]
                 )
```

Dados com transformação logarítmica

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_data_logreverted_fitted,
                 test_data=test_data.iloc[:240],
                 predictions=predictions_logreverted.iloc[:240]
                 )
```

Dados transformados por raiz cubica

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_data_raizc_reverted_fitted,
                 test_data=test_data.iloc[:240],
                 predictions=predictions_raizc_reverted.iloc[:240]
                 )
```


Observa-se que reduzindo o periodo de análise para 10 dias se obtêm melhores resultados em qualquer dos modelos mas principalmente quando efetuada uma transformação por log onde o modelo consegue explicar quase 87% dos dados de teste.

In [ ]:
```
# NoteBook:     ARIMA-Alcacova.ipynb
# Autor:        Carlos Pires
# Data:         30-12-2023
# Descrição:    Este notebook efetua as operações necessárias para aplicar o Modelo
#               ARIMA à série temporal de Alcáçova.
```

4. Modelo ARIMA

In [ ]:
```
# importação de bibliotecas

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 15, 6

# SARIMA model (ARIMA com sazonalidade)
from statsmodels.tsa.statespace.sarimax import SARIMAX

# AutoARIMA
from pmdarima.arima import auto_arima

# Statistic metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# grid view formatting
from tabulate import tabulate

# graphics
import plotly.graph_objects as go
import scipy.stats as stats
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import seaborn as sns
```

Definição de funções de apoio

In [ ]:
```
# Criar o gráfico interativo com a biblioteca Plotly para qualquer numero de dataframes

def graph(title_name="NO NAME", *args):

    fig = go.Figure()

    for i, arg in enumerate(args):
        dataframe = arg[0]
        attribute = arg[1]
        serie_name = arg[2]
        fig.add_trace(go.Scatter(x=dataframe.index,
                                 y=getattr(dataframe, attribute),
                                 name=serie_name,
                                 mode='lines',
                                 showlegend=True
                                )
                     )

    # Configurar os eixos e o layout do gráfico
    fig.update_layout(
        xaxis=dict(title='Data'),
        yaxis=dict(title='Caudal (l/s)'),
        title=title_name
    )

    # Habilitar o zoom interativo
    fig.update_layout(
        xaxis=dict(
            rangeselector=dict(
                buttons=list([
                    dict(count=1, label='1m', step='month', stepmode='backward'),
                    dict(count=6, label='6m', step='month', stepmode='backward'),
                    dict(count=1, label='1y', step='year', stepmode='backward'),
```



```
                    dict(count=3, label='3y', step='year', stepmode='backward'),
                    dict(step='all')
                ])
            ),
            rangeslider=dict(visible=True),
            type='date'
        )
    )

    # Mostrar o gráfico
    fig.show()
```

In [ ]:
```
# Função de cálculo e apresentação de métricas

def evaluate_metrics(train_data, train_data_fitted, test_data, predictions):

    mae_train = mean_absolute_error(train_data, train_data_fitted)
    mae_predict = mean_absolute_error(test_data, predictions)

    rmse_train = np.sqrt(mean_squared_error(train_data, train_data_fitted))
    rmse_predict = np.sqrt(mean_squared_error(test_data, predictions))

    r2_train = r2_score(train_data, train_data_fitted)
    r2_predict = r2_score(test_data, predictions)

    data = {
        'Dados': ['Treino', 'Teste'],
        'MAE': [mae_train, mae_predict],
        'RMSE': [rmse_train, rmse_predict],
        'R2': [r2_train, r2_predict]
    }

    df_evaluate = pd.DataFrame(data)

    print('MÉTRICAS DE AVALIAÇÃO DO MODELO')
    print(tabulate(df_evaluate, headers='keys', tablefmt='pretty', showindex=False))
```

4.1. Modelo ARIMA - Série Temporal de Alcáçova

ARIMA significa Auto Regressive Integrated Moving Average, trata-se de um modelo estatístico utilizado para análise e orevisão de séries temporais. O modelo combina componentes de autorregressão (AR), média móvel (MA) e diferenciação (I) para capturar diferentes aspetos da dinâmica da série temporal.

ARIMA(p, d, q)

Os três componentes principais do ARIMA são:

- Componente de AutoRegressão (AR): trata-se da relação entre os valores passados da série temporal e o valor atual. Assume que o valor atual depende linearmente dos seus valores anteriores, com um atraso definido como "ordem" (p). A ordem p do componente AR indica quantos lags anteriores devem ser considerados para a previsão;
- Componente de Média Móvel (MA): trata-se da relação entre os erros de previsão passados e o valor atual. Assume que o valor atual depende linearmente dos erros de previsão passados, com um atraso definido como "ordem" (q). A ordem q do componente MA indica quantos erros passados são considerados para a previsão.
- Componente de Diferenciação: Trabalha com a estacionaridade da série temporal. A diferenciação é realizada subtraindo o valor atual do valor anterior com o objetivo de remover tendências ou padrões de de longo prazo. A ordem de diferenciação (d) indica quantas vezes a série temporal é diferenciada até se tornar estacionária.

4.1.1. Considerações

Quando foi ralizada a análise à série temporal em causa, foi verificado que a série tinha as seguintes

características:

- Fraca estacionaridade;
- Forte Autocorrelação;
- Forte Autocorrelação Parcial;
- Existência de tendências;
- Existência de sazonalidade;
- Resíduos com forte Autocorrelação e Autocorrelação Parcial;
- Não segue uma distribuição Normal;
- As transformação por log e raiz cúbica não melhoraram a normalidade;
- A diferenciação de primeira ordem não melhorou a estacionaridade.

Todas estas caracaterísticas são contrárias ao hipotético sucesso da aplicação do modelo ARIMA a estes dados. É previsivel que os resultados não sejam bons, ainda assim, e devido à existência de sazonalidade, irá ser aplicado o modelo ARIMA com sazonalidade ao qual se dá o nome de SARIMA.

4.1.2. SARIMA (ARIMA com sazonalidade)

O modelo SARIMA é uma extensão do modelo ARIMA onde se incluem componentes para lidar com a sazonalidade:

SARIMA(p, d, q)x(P, D, Q, s)

Neste caso são adicionados 4 parâmetros:

- P - Componente de autoregressão sazonal: relação entre os valores sazonais passados e o valor atual;
- Q - Componente de de média móvel sazonal: relação entre os erros sazonais de previsão passados e o valor atual;
- D - Compoenete de diferenciação sazonal: lida com a estacionaridade sazonal da série temporal;
- s - Periodicidade sazonal.

Este será o modelo que se irá implementar aos dados que temos uma vez que temos de lidar com uma sazonalidade de 24h, já verificada anteriormente.

4.1.3. Importação dos dados

In [ ]:
```
df = pd.read_csv('../DadosPreProcessados/Alcacova.csv',
                 sep=',',
                 index_col='Data/Hora',
                 parse_dates=True
                 )
```

In [ ]:
```
# Apresentação do gráfico com os dados importados

graph("Série Temporal de Alcáçova - Caudal de Saída (l/s)",
      (df, "SAIDA", "Dados reais de caudal")
      )
```

4.1.4. Separação dos dados entre treino e teste

Os dados de treino e de teste serão separados da seguinte forma:

- De 01/10/2021 até 30/09/2022 serão dados de treino;
- De 01/10/2022 até 31/12/2022 serão os dados de teste.

Os dados são obtidos a partir do dataframe inicial para dois objetos do tipo "panda series".

In [ ]:
```
# Dividir os dados em conjuntos de treino e de teste

train_data = df[df.index <= '2022-09-30 23:00:00']['SAIDA']
test_data = df[df.index > '2022-09-30 23:00:00']['SAIDA']
```


In [ ]:
```
train_data.name = "SAIDA"
test_data.name = "SAIDA"

train_data.index.freq = "H"
test_data.index.freq = "H"
```

In [ ]:
```
graph("Série Temporal de Alcáçova - Caudal de Saída (l/s)",
      (pd.DataFrame(train_data, index=train_data.index), "SAIDA", "Dados de treino"),
      (pd.DataFrame(test_data, index=test_data.index), "SAIDA", "Dados de teste"))
```

4.1.5. Ajuste e aplicação do modelo ARIMA aos dados

In [ ]:
```
# Ajuste do modelo SARIMA

order = (3, 1, 2) # (p, d, q) - Componentes não sazonais
seasonal_order = (1, 1, 1, 24) # (P, D, Q, s) - Componentes sazonais

sarima_model = SARIMAX(train_data, order=order, seasonal_order=seasonal_order)
sarima_result = sarima_model.fit()
```

In [ ]:
```
print(sarima_result.summary())
```

AIC: (0, 0, 0)(0, 0, 0, 24) = 62398

AIC: (1, 1, 1)(1, 1, 1, 24) = 18293

AIC: (1, 1, 1)(1, 1, 1, 24) = 14848

AIC: (2, 1, 1)(1, 1, 1, 24) = 14831

AIC: (2, 1, 2)(1, 1, 1, 24) = 14782

AIC: (3, 1, 2)(1, 1, 1, 24) = 14770

4.1.5.1. Análise dos resíduos

In [ ]:
```
sarima_resid = sarima_result.resid
```

In [ ]:
```
sarima_resid.plot()
plt.show();
```

Testes de Normalidade

In [ ]:
```
stats.probplot(sarima_resid, dist="norm", plot=plt)
plt.title("Normal QQ plot")
plt.show()
```

O teste de Shapiro-Wilk é um teste estatítico utilizado para verificar se um determinado conjunto de dados segue uma distribuição normal. As hipóteses do teste são:

- Hipótese nula (H0): Os dados seguem uma distribuição normal. Rejeitar se p-value < nível de significância;
- Hipótese alternativa (H1): Os dados não seguem uma distribuição normal.

Nível de significância: 0.05

In [ ]:
```
e, p = stats.shapiro(sarima_resid)
print('Estatística de teste: {}'.format(e))
print('p-valor: {}'.format(p))
```

O teste de Anderson-Darling é um teste estatístico para verificar se um determinado conjunto de dados segue uma distribuição normal. As hipóteses do teste são:

- Hipótese nula (H0): Os dados seguem uma distribuição normal. Rejeitar se estatítica do teste > valor crítico;
- Hipótese alternativa (H1): Os dados não seguem uma distribuição normal.

Nível de significância: 0.05

In [ ]:
```
from scipy.stats import anderson
```

In [ ]:
```
result = anderson(sarima_resid)
```

In [ ]:
```
print('Test statistic: %.3f' % result.statistic)
print('Critical values:')
for i in range(len(result.critical_values)):
    sl, cv = result.significance_level[i], result.critical_values[i]
    print('%.1f%%: %.3f' % (sl, cv))
```

In [ ]:
```
sns.histplot(sarima_resid, kde=True);
```

Conclui-se que os resíduos não seguem uma distribuição normal.

Autocorrelação

In [ ]:
```
# Diagrama ACF

plot_acf(sarima_resid, lags=50)
plt.show();
```

In [ ]:
```
# Diagrama PACF

plot_pacf(sarima_resid, lags=50)
plt.show();
```

Os resíduos apresentam alguma autocorrelação.

Análise Gráfica

In [ ]:
```
res = train_data-sarima_resid
res.name = "SAIDA"

graph("Série Temporal de Alcáçova - Caudal de Saída (l/s)",
      (pd.DataFrame(train_data, index=train_data.index), "SAIDA", "Dados de treino"),
      (pd.DataFrame(res, index=train_data.index), "SAIDA", "Resíduos")
)
```

Os resíduos apresentam-se concordantes com a série original.

4.1.5.2. Previsão

In [ ]:
```
# Dados de treino ajustados

train_data_adjusted = sarima_result.fittedvalues
train_data_adjusted.name = "SAIDA"
train_data_adjusted
```

In [ ]:
```
#Previsões

prediction = sarima_result.forecast(2208)
prediction.name = "SAIDA"
prediction
```

In [ ]:
```
graph("Série Temporal de Alcáçova - Caudal de Saída (l/s)",
        (pd.DataFrame(train_data,
                      index=train_data.index),
                      "SAIDA", "Dados de treino"),
        (pd.DataFrame(train_data_adjusted,
                      index=train_data.index),
                      "SAIDA",
                      "Dados de treino ajustados"),
        (pd.DataFrame(test_data,
                      index=test_data.index),
                      "SAIDA",
                      "Dados de teste"),
```



```
        (pd.DataFrame(prediction,
                      index=test_data.index),
                      "SAIDA",
                      "Previsões")
)
```

Gráficamente observa-se que as previsões apresentam uma tendência de subida mais acentuada do que os dados reais. Esta tendência gerará erros mais acentuados no final no periodo da previsão.

4.1.5.3. Avaliação do modelo

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_data_adjusted,
                 test_data=test_data,
                 predictions=prediction
                 )
```

O modelo consegue explicar cerca de 94% dos dados de treino mas apenas 73% dos dados de teste. Assume-se que o mesmo possa estar em overfitting, o que não é de estranhar dadas as características dos dados vistas anteriormente.

4.1.6. Utilização do método AutoARIMA

O Auto ARIMA é um método computacional que se baseia na utilização de uma biblioteca que procura testar diferentes parâmetros obtendo diferentes modelos, com o objetivo de convergir para aquele que tenha o menor AIC.

4.1.6.1. Ajuste e aplicação do modelo ARIMA utilizando a biblioteca AutoARIMA

In [ ]:
```
### AUTOARIMA

model_auto = auto_arima(train_data,
                        trace = True,
                        stepwise = True,
                        seasonal=True,
                        stationary=True,
                        max_p=3,
                        max_q=2,
                        max_P=1,
                        max_Q=1,
                        start_p=0,
                        start_q=0,
                        start_P=0,
                        start_Q=0,
                        m=24
            )
```

Com estacionaridade = False:

Best model: ARIMA(1,1,1)(1,0,0)[24] intercept Total fit time: 244.444 seconds 18654

Com estacionaridade = True

Best model: ARIMA(0,0,2)(1,0,1)[24] intercept Total fit time: 207.217 seconds 15641

Observa-se que passar estacionaridade true conduz a um melhor AIC

In [ ]:
```
print(model_auto.aic())
```

A utilização do método Auto ARIMA leva-nos a um modelo diferente daquele que foi parametrizado anteriormente, menos complexo mas com um AIC um pouco superior.

4.1.6.2. Análise dos Resíduos

In [ ]:
```
model_auto_resid = model_auto.resid()
```

In [ ]:
```
model_auto_resid.plot()
plt.show();
```

Testes de Normalidade dos Resíduos

In [ ]:
```
stats.probplot(model_auto_resid, dist="norm", plot=plt)
plt.title("Normal QQ plot")
plt.show()
```

O teste de Shapiro-Wilk é um teste estatítico utilizado para verificar se um determinado conjunto de dados segue uma distribuição normal. As hipóteses do teste são:

- Hipótese nula (H0): Os dados seguem uma distribuição normal. Rejeitar se p-value < nível de significância;
- Hipótese alternativa (H1): Os dados não seguem uma distribuição normal.

Nível de significância: 0.05

In [ ]:
```
e, p = stats.shapiro(model_auto_resid)
print('Estatística de teste: {}'.format(e))
print('p-valor: {}'.format(p))
```

O teste de ANderson-Darling é um teste estatístico para verificar se um determinado conjunto de dados segue uma distribuição normal. As hipóteses do teste são:

- Hipótese nula (H0): Os dados seguem uma distribuição normal. Rejeitar se estatítica do teste > valor crítico;
- Hipótese alternativa (H1): Os dados não seguem uma distribuição normal.

Nível de significância: 0.05

In [ ]:
```
from scipy.stats import anderson
```

In [ ]:
```
result = anderson(model_auto_resid)
```

In [ ]:
```
print('Test statistic: %.3f' % result.statistic)
print('Critical values:')
for i in range(len(result.critical_values)):
    sl, cv = result.significance_level[i], result.critical_values[i]
    print('%.1f%%: %.3f' % (sl, cv))
```

In [ ]:
```
sns.histplot(model_auto_resid, kde=True);
```

Conclui-se que os resíduos não seguem uma distribuição normal.

Autocorrelação

In [ ]:
```
# Diagrama ACF

plot_acf(model_auto_resid, lags=50)
plt.show();
```

In [ ]:
```
# Diagrama PACF

plot_pacf(sarima_resid, lags=50)
plt.show();
```

Os resíduos apresentam uma autocorrelação mais forte do que no modelo anterior.

Análise Gráfica

In [ ]:
```
res = train_data-model_auto_resid
res.name = "SAIDA"

graph("Série Temporal de Alcáçova - Caudal de Saída (l/s)",
      (pd.DataFrame(train_data, index=train_data.index), "SAIDA", "Dados de treino"),
```



```
      (pd.DataFrame(res, index=train_data.index), "SAIDA", "Resíduos"))
```

Os resíduos apresentam-se concordantes com a série original.

4.1.6.3. Previsão

In [ ]:
```
# Dados de treino ajustados

train_data_adjusted_auto = model_auto.fittedvalues()
train_data_adjusted_auto.name = "SAIDA"
train_data_adjusted_auto
```

In [ ]:
```
# Previsões

prediction_auto = model_auto.predict(2208)
prediction_auto.name = "SAIDA"
prediction_auto
```

In [ ]:
```
graph("Série Temporal de Alcáçova - Caudal de Saída (l/s)",
       (pd.DataFrame(train_data,
                     index=train_data.index),
                     "SAIDA",
                     "Dados de treino"),
       (pd.DataFrame(train_data_adjusted_auto,
                     index=train_data.index),
                     "SAIDA",
                     "Dados de treino ajustados"),
       (pd.DataFrame(test_data,
                     index=test_data.index),
                     "SAIDA",
                     "Dados de teste"),
       (pd.DataFrame(prediction_auto,
                     index=test_data.index),
                     "SAIDA",
                     "Previsões")
)
```

Graficamente observa-se que as previsões vão perdendo amplitude sendo os erros maiores no final do periodo da previsão.

4.1.6.4. Avaliação do modelo

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_data_adjusted,
                 test_data=test_data,
                 predictions=prediction_auto)
```

O modelo consegue explicar cerca de 94% dos dados de treino e 81% dos dados de teste.

4.1.7. Comparação das métricas dos dois modelos

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_data_adjusted,
                 test_data=test_data,
                 predictions=prediction)
```

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_data_adjusted_auto,
                 test_data=test_data,
                 predictions=prediction_auto)
```

Comparando as métricas dos dois modelos verifica-se que o modelo eleito pelo Auto ARIMA é melhor do que o parametrizado manualmente. Possui erros menores e tem um valor de R2 de cerca de 81%, para além de ser um modelo menos complexo. Evidentemente qualquer um dos modelos tem problemas de previsão a longo prazo, mas no que respeita ao periodo de previsão em causa, deve ser escolhido o segundo modelo como o melhor do modelo ARIMA para os dados em questão.

4.1.8. Análise dos resultados para previsões de curto prazo (10 dias)

Considerou-se importante verificar a performance dos modelos para um período mais curto uma vez que períodos muito longos podem potenciar a ocorrência de erros de previsão com a consequente desistência do modelo em causa quando o mesmo pode fazer boas previsões de curto prazo e, em algumas situações, serão essas as mais importantes. Considerou-se que o período mais curto seria de 10 dias pois do ponto de vista de análise e de desenvolvimento futuro deste trabalho será esse o período mínimo.

Assim sendo, uma vez que temos um registo por hora, estamos interessados em comparar os primeiros 24x10 = 240 valores.

4.1.8.1. Avaliação do modelo para uma previsão de curto prazo

Modelo parametrizado manualmente

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_data_adjusted,
                 test_data=test_data.iloc[:240],
                 predictions=prediction.iloc[:240])
```

Modelo resultado do Autoarima

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_data_adjusted_auto,
                 test_data=test_data.iloc[:240],
                 predictions=prediction_auto.iloc[:240])
```

Para uma previsão de curto prazo (10 dias) o modelo parametrizado manualmente apresenta resultados ligeiramente superiores. Este, consegue explicar quase 89% dos dados de teste numa previsão para os próximos 10 dias.

In [ ]:
```
# NoteBook:     LSTM-Alcacova.ipynb
# Autor:        Carlos Pires
# Data:         30-12-2023
# Descrição:    Este notebook efetua as operações necessárias para aplicar o Modelo
#               LSTM à série temporal de Alcáçova.
```

5. Modelo LSTM

In [ ]:
```
# importação de bibliotecas

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 15, 6

# Tensorflow
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, InputLayer
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.metrics import RootMeanSquaredError

np.random.seed(42)
tf.random.set_seed(seed=42)

# Statistic metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
# grid view formatting
from tabulate import tabulate

# graphics
import plotly.graph_objects as go
```

Definição de funções de apoio

In [ ]:
```
# Criar o gráfico interativo com a biblioteca Plotly para qualquer numero de dataframes

def graph(title_name="NO NAME", *args):

    fig = go.Figure()

    for i, arg in enumerate(args):
        dataframe = arg[0]
        attribute = arg[1]
        serie_name = arg[2]
        fig.add_trace(go.Scatter(x=dataframe.index,
                                 y=getattr(dataframe, attribute),
                                 name=serie_name,
                                 mode='lines',
                                 showlegend=True))

    # Configurar os eixos e o layout do gráfico
    fig.update_layout(
        xaxis=dict(title='Data'),
        yaxis=dict(title='Caudal (l/s)'),
        title=title_name
    )

    # Habilitar o zoom interativo
    fig.update_layout(
        xaxis=dict(
            rangeselector=dict(
                buttons=list([
                    dict(count=1, label='1m', step='month', stepmode='backward'),
                    dict(count=6, label='6m', step='month', stepmode='backward'),
                    dict(count=1, label='1y', step='year', stepmode='backward'),
```

```
                    dict(count=3, label='3y', step='year', stepmode='backward'),
                    dict(step='all')
                ])
            ),
            rangeslider=dict(visible=True),
            type='date'
        )
    )

    # Mostrar o gráfico
    fig.show()
```

In [ ]:
```
# Função de cálculo e apresentação de métricas

def evaluate_metrics(train_data, train_data_fitted, test_data, predictions):

    mae_train = mean_absolute_error(train_data, train_data_fitted)
    mae_predict = mean_absolute_error(test_data, predictions)

    rmse_train = np.sqrt(mean_squared_error(train_data, train_data_fitted))
    rmse_predict = np.sqrt(mean_squared_error(test_data, predictions))

    r2_train = r2_score(train_data, train_data_fitted)
    r2_predict = r2_score(test_data, predictions)

    data = {
        'Dados': ['Treino', 'Teste'],
        'MAE': [mae_train, mae_predict],
        'RMSE': [rmse_train, rmse_predict],
        'R2': [r2_train, r2_predict]
    }

    df_evaluate = pd.DataFrame(data)

    print('MÉTRICAS DE AVALIAÇÃO DO MODELO')
    print(tabulate(df_evaluate, headers='keys', tablefmt='pretty', showindex=False))
```

In [ ]:
```
# Transformação matricial da série para n = 8

# [[[1], [2], [3], [4], [5]]] [6]
# [[[2], [3], [4], [5], [6]]] [7]
# [[[3], [4], [5], [6], [7]]] [8]

def df_to_X_y(df, window_size):

    df_as_np = df.to_numpy()
    X = []
    y = []

    for i in range(len(df_as_np) - window_size):
        row = [[a] for a in df_as_np[i:i+window_size]]
        X.append(row)

        label = df_as_np[i+window_size]
        y.append(label)

    return np.array(X), np.array(y)
```

In [ ]:
```
# Transformação matricial da série multivariável para n=8

# [[[f1, ds1, dc1, ys1, yc1], [f2, ds2, dc2, ys2, yc2], [f3, ds3, dc3, ys3, yc3],
# [f4, ds4, dc4, ys4, yc4], [f5, ds5, dc5, ys5, yc5]]] [f6]
# [[[f2, ds2, dc2, ys2, yc2], [f3, ds3, dc3, ys3, yc3], [f4, ds4, dc4, ys4, yc4],
# [f5, ds5, dc5, ys5, yc5], [f6, ds6, dc6, ys6, yc6]]] [f7]
# [[[f3, ds3, dc3, ys3, yc3], [f4, ds4, dc4, ys4, yc4], [f5, ds5, dc5, ys5, yc5],
# [f6, ds6, dc6, ys6, yc6], [f7, ds7, dc7, ys7, yc7]]] [f8]

# fx valor do caudal no momento x
# dsx e dcx valores de seno e coseno que representam o momento do dia
# ysx e ycx valores do seno e coseno que representam o momento do ano

def df_to_X_y_multivariate(df, window_size=6):
```



```
    df_as_np = df.to_numpy()
    X = []
    y = []

    for i in range(len(df_as_np) - window_size):
        row = [r for r in df_as_np[i:i+window_size]]
        X.append(row)

        label = df_as_np[i+window_size][0]
        y.append(label)

    return np.array(X), np.array(y)
```

5.1. LSTM - Série Temporal de Alcáçova

5.1.2. Importação dos dados

In [ ]:
```
df = pd.read_csv('../DadosPreProcessados/Alcacova.csv',
                 sep=',',
                 index_col='Data/Hora',
                 parse_dates=True)
```

In [ ]:
```
graph("Série Temporal de Alcáçova - Caudal de Saída (l/s)",
      (df, "SAIDA", "Dados reais de caudal"))
```

5.1.3. Separação dos dados entre treino, validação e teste

Normalmente os dados seriam separados entre treino e teste, acontece que o modelo LSTM necessida de dados de teste para o treino, o que significa que o modelo ajustado fica a conhecer os dados que utilizaríamos para teste. Assim, e por forma a manter as premissas da proposta enunciada inicialmente, utilizar-se-ão 9 meses dos dados de treino e os restantes 3 serão dados de validação mantendo assim os outros 3 meses como dados de teste a verificar no final.

In [ ]:
```
# Dividir os dados em conjuntos de treino, validação e teste.
# O modelo utilizará os dados de treino e validação.
# Os dados de teste serão depois previstos

train_data = df[df.index <= '2022-06-30 23:00:00']['SAIDA']
val_data = df[(df.index > '2022-06-30 23:00:00') &
              (df.index <= '2022-09-30 23:00:00')]['SAIDA']
test_data = df[(df.index > '2022-09-30 23:00:00') &
               (df.index <= '2022-12-31 23:00:00')]['SAIDA']

train_data.name = "SAIDA"
val_data.name = "SAIDA"
test_data.name = "SAIDA"

# número de dados total e pertencentes ao conjunto de treino, validação e teste
df.count(), train_data.count(), val_data.count(), test_data.count()
```

In [ ]:
```
graph("Série Temporal de Alcáçova - Caudal de Saída (l/s)",
        (pd.DataFrame(train_data,
                      index=train_data.index),
                      "SAIDA",
                      "Dados de treino"),
        (pd.DataFrame(val_data,
                      index=val_data.index),
                      "SAIDA",
                      "Dados de validação"),
        (pd.DataFrame(test_data,
                      index=test_data.index),
                      "SAIDA",
                      "Dados de teste"),
)
```

5.1.4. Aplicação do modelo LSTM

5.1.4.1. Transformação matricial dos dados de treino e de validação

In [ ]:
```
# time step
# foram testadas outras janelas temporais, no entanto o valor de 24,
# igual ao número de elementos que completam um ciclo de sazonalidade diária,
# foi aquele com o qual se obtiveram os melhores resultados.

WINDOW_SIZE = 24

Xtrain, ytrain = df_to_X_y(train_data, WINDOW_SIZE)
Xval, yval = df_to_X_y(val_data, WINDOW_SIZE)

Xtrain.shape, ytrain.shape, Xval.shape, yval.shape
```

5.1.4.2. Arquitetura do modelo

Propõe-se uma arquitetura com uma layer de entrada com os valores do WINDOW_SIZE, uma layer LSTM com 48 nós, uma layer Dense com 24 nós e apenas um nó de saída que nos dá o valor presvisto. Foram testados outros números de nós na layer LSTM. 48 foi o número que produzia melhores resultados. Foram testadas a inclusões de layers Dense com 24 nós antes e depois da layer LSTM. O modelo que apresentou a melhor convergência foi aquele que inclui uma layer Dense com 24 nós antes da layer de saída e é este que se apresenta de seguida.

In [ ]:
```
model = Sequential()
model.add(InputLayer((WINDOW_SIZE, 1)))
model.add(LSTM(48))
model.add(Dense(24))
model.add(Dense(1))

model.summary()
```

5.1.4.3. Compilação

In [ ]:
```
check_point = ModelCheckpoint(
    'best_model.h5',
    monitor='val_loss',
    save_best_only=True
)

model.compile(
    loss=MeanSquaredError(),
    optimizer=Adam(learning_rate=0.001),
    metrics=[RootMeanSquaredError()]
)
```

In [ ]:
```
r = model.fit(
    Xtrain,
    ytrain,
    validation_data=(Xval, yval),
    epochs=100,
    callbacks=[check_point],
)
```

5.1.4.4. Avaliação do modelo

In [ ]:
```
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend();
```

In [ ]:
```
# Evaluate the model
train_loss = model.evaluate(Xtrain, ytrain, verbose=0)
print('Train loss:', train_loss)
val_loss = model.evaluate(Xval, yval, verbose=0)
print('Val loss:', val_loss)
```


O modelo apresenta uma boa convergência

In [ ]:
```
#load best model

mymodel = tf.keras.models.load_model('best_model.h5')
```

5.1.4.5. Previsão dos dados de treino e de validação

Interessa agora utilizar os dados de treino e validação e verificar se o modelo consegue fazer boas previsões. Como o modelo foi treinado com estes valores é previsível que os resultados sejam bons.

In [ ]:
```
Ptrain = mymodel.predict(Xtrain).flatten()
Pval = mymodel.predict(Xval).flatten()
```

Nestas previsões existem valores que não são previsíveis, ou seja, os primeiros valores com o tamanho da janela temporal WINDOW_SIZE não são previsíveis. Assim, e por forma a ter séries previstas completas, os valores que estão em falta serão preenchidos pelos valores reais.

In [ ]:
```
# In train predictions first WINDOW_SIZE data are not predictable,
# so it will be completed with real values.
#
# In val predictions first WINDOW_SIZE data are not predictable,
# so it will be completed with real values.

train_vector = train_data[:WINDOW_SIZE]
Ptrain = np.concatenate((train_vector, Ptrain))

val_vector = val_data[:WINDOW_SIZE]
Pval = np.concatenate((val_vector, Pval))

Ptrain.shape, Pval.shape
```

In [ ]:
```
# Create Panda Series

train_predictions = pd.Series(data=Ptrain, index=train_data.index)
train_predictions.name = "SAIDA"

val_predictions = pd.Series(data=Pval, index=val_data.index)
val_predictions.name = "SAIDA"
```

In [ ]:
```
graph("Série Temporal de Alcáçova - Caudal de Saída (l/s)",
        (pd.DataFrame(train_data,
                      index=train_data.index),
                      "SAIDA",
                      "Dados de treino"),
        (pd.DataFrame(train_predictions,
                      index=train_predictions.index),
                      "SAIDA",
                      "Dados de treino previstos pelo modelo"),
        (pd.DataFrame(val_data,
                      index=val_data.index),
                      "SAIDA", "Dados de validação"),
        (pd.DataFrame(val_predictions,
                      index=val_predictions.index),
                      "SAIDA",
                      "Dados de validação previstos pelo modelo"),
)
```

Aumentando o zoom no gráfico pode observar-se que as curvas estão bem ajustadas.

5.1.4.6. Avaliação das previsões de treino e de validação

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_predictions,
                 test_data=val_data,
                 predictions=val_predictions)
```

Tal como previsto os resultados das métricas são bons mas as métricas apresentadas anteriormente apenas

nos dão boas indicações de que o modelo consegue prever eficazmente os dados de treino e de validação, mas não nos dão nenhuma indicação sobre a sua capacidade de prever valores novos, ou seja, valores que o modelo não conheceu durante a sua "aprendizagem".

5.1.4.7. Previsões de valores novos - Multistep forecast

Tal como referido inicialmente, existem 3 meses de dados de teste entre Outubro de 2022 e Dezembro de 2022 que não foram dados a conhecer ao modelo. São estes os dados que serão previstos pelo modelo e depois comparados com os valores reais.

O modelo irá efetuar uma previsão do valor seguinte pegando sempre nos 24 valores anteriores conhecidos. Inicialmente começará a utilizar os valores reais do final do Xval que irão sendo substituídos por novos valores previstos. A partir de determienada altura estaremos a fazer previsões com base em previsões utilizando apenas o modelo treinado anteriormente.

In [ ]:
```
multistep_predictions = []

last_x = Xval[-1]
while len(multistep_predictions) < test_data.count():
    p = mymodel.predict(last_x.reshape(1, -1, 1))[0]

    #update the predictions list
    multistep_predictions.append(p)

    #make the new input
    last_x = np.roll(last_x, -1)
    last_x[-1] = p
```

In [ ]:
```
predictions = [array[0] for array in multistep_predictions]
predictions
```

In [ ]:
```
# Create Panda Serie

test_predictions = pd.Series(data=predictions, index=test_data.index)
test_predictions.name = "SAIDA"
```

In [ ]:
```
test_predictions
```

In [ ]:
```
graph("Série Temporal de Alcáçova - Caudal de Saída (l/s)",
        (pd.DataFrame(train_data,
                      index=train_data.index),
                      "SAIDA",
                      "Dados de treino"),
        (pd.DataFrame(train_predictions,
                      index=train_predictions.index),
                      "SAIDA",
                      "Dados de treino previstos pelo modelo"),
        (pd.DataFrame(val_data,
                      index=val_data.index),
                      "SAIDA",
                      "Dados de validação"),
        (pd.DataFrame(val_predictions,
                      index=val_predictions.index),
                      "SAIDA",
                      "Dados de validação previstos pelo modelo"),
        (pd.DataFrame(test_data,
                      index=test_data.index),
                      "SAIDA",
                      "Dados de teste reais"),
        (pd.DataFrame(test_predictions,
                      index=test_data.index),
                      "SAIDA",
                      "Dados de teste previstos pelo modelo"),
)
```

A amplitude das curvas que compoem os valores previstos aparenta ser bastante boa, no entanto fazendo zoom rapidamente se percebe que existe um desfasamento entre a curva de dados de teste reais e a de

dados de teste previstos.

5.1.4.8. Avaliação das previsões efetuadas pelo modelo

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_predictions,
                 test_data=test_data, p
                 redictions=test_predictions)
```

Tal como referido no ponto anterior, enbora a amplitude aparente ser boa, o desfazamento existente entre os dados de teste e as previsões é muito significativo resultando na obtenção de métricas inaceitáveis e concluindo na incapacidade do modelo em fazer previsões.

5.1.5. Aplicação do modelo LSTM utilizando dados padronizados

Com a tentativa de melhorar os resultados obtidos, vão ser padronizados os dados de caudal.

Foi utilizada a biblioteca StandardScaler que para cada dado de caudal efetua o seguinte:

$$z = \frac{x - \mu}{\sigma}$$

Foi também testada a normalização com MinMaxScaler (normalização entre 0 e 1) mas os resultados não melhoraram em relação aos da padronização.

A normalização utiliza a seguinte fórmula:

$$X_c = \frac{X - X_{min}}{X_{max} - X_{min}}$$

5.1.5.1. Padronização dos dados

In [ ]:
```
# Importação de biblioteca necessária

from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
scaler = StandardScaler()
```

In [ ]:
```
train_data_array = train_data.values.reshape(-1,1)
val_data_array = val_data.values.reshape(-1,1)

train_data_scaler = scaler.fit(np.array(train_data_array))
val_data_scaler = scaler.fit(np.array(val_data_array))

train_data_standardized = scaler.transform(train_data_array)
val_data_standardized = scaler.transform(val_data_array)
```

In [ ]:
```
# Create Panda Series

train_data_standardized_serie = pd.Series(
    data=train_data_standardized.flatten(),
    index=train_data.index)
val_data_standardized_serie = pd.Series(
    data=val_data_standardized.flatten(),
    index=val_data.index)
```

5.1.5.2. Transformação matricial dos dados de treino e de validação

In [ ]:
```
WINDOW_SIZE = 24

Xtrain_std, ytrain_std = df_to_X_y(train_data_standardized_serie, WINDOW_SIZE)
Xval_std, yval_std = df_to_X_y(val_data_standardized_serie, WINDOW_SIZE)

Xtrain_std.shape, ytrain_std.shape, Xval_std.shape, yval_std.shape
```

5.1.5.3. Arquitetura do modelo

Propõe-se uma arquitetura com uma layer de entrada com os valores do WINDOW_SIZE, uma layer LSTM com 48 nós, uma layer Dense com 24 nós e apenas um nó de saída que nos dá o valor presvisto. Foram testados outros números de nós na layer LSTM. 48 foi o número que produzia melhores resultados. Foram testadas a inclusões de layers Dense com 24 nós antes e depois da layer LSTM. O modelo que apresentou a melhor convergência foi aquele que inclui uma layer Dense com 24 nós antes da layer de saída e é este que se apresenta de seguida.

In [ ]:
```
model_std = Sequential()
model_std.add(InputLayer((WINDOW_SIZE, 1)))
model_std.add(LSTM(48))
model_std.add(Dense(24))
model_std.add(Dense(1))

model_std.summary()
```

5.1.5.4. Compilação

In [ ]:
```
check_point_std = ModelCheckpoint(
    'best_model_std.h5',
    monitor='val_loss',
    save_best_only=True
)

model_std.compile(
    loss=MeanSquaredError(),
    optimizer=Adam(learning_rate=0.001),
    metrics=[RootMeanSquaredError()]
)
```

In [ ]:
```
r_std = model_std.fit(
    Xtrain_std,
    ytrain_std,
    validation_data=(Xval_std, yval_std),
    epochs=100,
    callbacks=[check_point_std])
```

5.1.5.5. Avaliação do modelo

In [ ]:
```
plt.plot(r_std.history['loss'], label='train loss')
plt.plot(r_std.history['val_loss'], label='val loss')
plt.legend();
```

In [ ]:
```
# Evaluate the model
train_loss_std = model_std.evaluate(Xtrain_std, ytrain_std, verbose=0)
print('Train loss:', train_loss_std)
test_loss_std = model_std.evaluate(Xval_std, yval_std, verbose=0)
print('Test loss:', test_loss_std)
```

In [ ]:
```
#load best model
mymodel_std = tf.keras.models.load_model('best_model_std.h5')
```

5.1.5.6. Previsão dos dados de treino e de validação

Interessa agora utilizar os dados de treino e validação e verificar se o modelo consegue fazer boas previsões. Como o modelo foi treinado com estes valores é previsível que os resultados sejam bons.

In [ ]:
```
Ptrain_std = mymodel_std.predict(Xtrain_std).flatten()
Pval_std = mymodel_std.predict(Xval_std).flatten()
```

Nestas previsões existem valores que não são previsíveis, ou seja, os primeiros valores com o tamanho da janela temporal WINDOW_SIZE não são previsíveis. Assim, e por forma a ter séries previstas completas, os valores que estão em falta serão preenchidos pelos valores reais.

In [ ]:
```
# In train predictions first WINDOW_SIZE data are not predictable,
# so it will be completed with real values.
#
# In val predictions first WINDOW_SIZE data are not predictable,
# so it will be completed with real values.

train_vector_std = train_data_standardized_serie[:WINDOW_SIZE]
Ptrain_std = np.concatenate((train_vector_std, Ptrain_std))

val_vector_std = val_data_standardized_serie[:WINDOW_SIZE]
Pval_std = np.concatenate((val_vector_std, Pval_std))

Ptrain_std.shape, Pval_std.shape
```

In [ ]:
```
# Inverse transformation and create Panda Series

Ptrain_undo_std = scaler.inverse_transform(Ptrain_std.reshape(-1,1))
Pval_undo_std = scaler.inverse_transform(Pval_std.reshape(-1,1))

train_predictions_std = pd.Series(
    data=Ptrain_undo_std.flatten(),
    index=train_data.index)
train_predictions_std.name = "SAIDA"

val_predictions_std = pd.Series(
    data=Pval_undo_std.flatten(),
    index=val_data.index)
val_predictions_std.name = "SAIDA"
```

In [ ]:
```
graph("Série Temporal de Alcáçova - Caudal de Saída (l/s)",
        (pd.DataFrame(train_data,
                      index=train_data.index),
                      "SAIDA",
                      "Dados de treino"),
        (pd.DataFrame(train_predictions_std,
                      index=train_predictions_std.index),
                      "SAIDA",
                      "Dados de treino previstos pelo modelo"),
        (pd.DataFrame(val_data,
                      index=val_data.index),
                      "SAIDA",
                      "Dados de validação"),
        (pd.DataFrame(val_predictions_std,
                      index=val_predictions_std.index),
                      "SAIDA",
                      "Dados de validação previstos pelo modelo"),
)
```

Aumentando o zoom no gráfico pode observar-se que as curvas estão bem ajustadas.

5.1.5.7. Avaliação das previsões de treino e de validação

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_predictions_std,
                 test_data=val_data,
                 predictions=val_predictions_std)
```

Tal como previsto os resultados das métricas são bons mas as métricas apresentadas anteriormente apenas nos dão boas indicações de que o modelo consegue prever eficazmente os dados de treino e de validação, mas não nos dão nenhuma indicação sobre a sua capacidade de prever valores novos, ou seja, valores que o modelo não conheceu durante a sua "aprendizagem".

5.1.5.8. Previsões de valores novos - Multistep forecast

Tal como referido inicialmente, existem 3 meses de dados de teste entre Outubro de 2022 e Dezembro de 2022 que não foram dados a conhecer ao modelo. São estes os dados que serão previstos pelo modelo e depois comparados com os valores reais.

O modelo irá efetuar uma previsão do valor seguinte pegando sempre nos 24 valores anteriores conhecidos. Inicialmente começará a utilizar os valores reais do final do Xval_std que irão sendo substituídos por novos valores previstos. A partir de determienada altura estaremos a fazer previsões com base em previsões utilizando apenas o modelo treinado anteriormente.

In [ ]:
```
multistep_predictions_std = []

last_x_std = Xval_std[-1]
while len(multistep_predictions_std) < test_data.count():
    p_std = mymodel_std.predict(last_x_std.reshape(1, -1, 1))[0]

    #update the predictions list
    multistep_predictions_std.append(p_std)

    #make the new input
    last_x_std = np.roll(last_x_std, -1)
    last_x_std[-1] = p_std
```

In [ ]:
```
predictions_std = [array[0] for array in multistep_predictions_std]
predictions_std
```

In [ ]:
```
predictions_undo_std = scaler.inverse_transform(
    np.array(multistep_predictions_std).reshape(-1,1))
```

In [ ]:
```
# Create Panda Serie

test_predictions_std = pd.Series(
    data=predictions_undo_std.flatten(),
    index=test_data.index)
test_predictions_std.name = "SAIDA"
```

In [ ]:
```
graph("Série Temporal de Alcáçova - Caudal de Saída (l/s)",
        (pd.DataFrame(train_data,
                      index=train_data.index),
                      "SAIDA",
                      "Dados de treino"),
        (pd.DataFrame(train_predictions_std,
                      index=train_predictions_std.index),
                      "SAIDA",
                      "Dados de treino previstos pelo modelo"),
        (pd.DataFrame(val_data,
                      index=val_data.index),
                      "SAIDA",
                      "Dados de validação"),
        (pd.DataFrame(val_predictions_std,
                      index=val_predictions_std.index),
                      "SAIDA",
                      "Dados de validação previstos pelo modelo"),
        (pd.DataFrame(test_data,
                      index=test_data.index),
                      "SAIDA",
                      "Dados de teste"),
        (pd.DataFrame(test_predictions_std,
                      index=test_predictions_std.index),
                      "SAIDA",
                      "Dados de teste previstos pelo modelo"),
)
```

A amplitude das curvas que compoem os valores previstos aparenta ser bastante boa, no entanto fazendo zoom rapidamente se percebe que existe um desfasamento entre a curva de dados de teste reais e a de dados de teste previstos.

5.1.5.9. Avaliação das previsões efetuadas pelo modelo

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_predictions_std,
                 test_data=test_data,
                 predictions=test_predictions_std)
```


Tal como referido no ponto anterior, embora a amplitude aparente ser boa, o desfazamento existente entre os dados de teste e as previsões é muito significativo resultando na obtenção de métricas inaceitáveis e concluindo na incapacidade do modelo em fazer previsões.

5.1.6. Aplicação do modelo LSTM utilizando multivariáveis

Pelos resultados dos pontos anteriores e chegados a este ponto, pode-se concluir que o modelo consegue uma boa amplitude para as previsões de teste mas por alguma razão não consegue ser rigoroso na sazonalidade e começa a prever dados desfazados dos reais. Claro está que, como a partir de certa altura as previsões são feitas com dados que anteriormente foram previstos este desfasamento vai sendo cada vez mais acentuado.

Interessa então tentar dar alguma ajuda ao modelo para que ele consiga ser mais rigoroso com a sazonalidade. Se inicialmente a escolha de uma janela temporal de 24 valores aparentava ser suficiante, os resultados mostram-nos o contrário.

Interessa assim adicionar mais variáveis com informação útil ao modelo que o ajude a ser fiel à sazonalidade. Essas variáveis estão disponíveis no index do dataframe e tratam-se do tempo, ou seja, Data/hora de cada observação.

O tempo é o elemento que os outros modelos dispõem mas que até agora não tínhamos disponibilizado ao modelo LSTM. Não esávamos a ser justos e por isso os resultados eram tão maus.

5.1.6.1. Criação do dataframe com as variáveis necessárias ao tempo

In [ ]:
```
df
```

In [ ]:
```
flow_df = df.copy()
flow_df
```

In [ ]:
```
# Obtenção do Timestamp do index que tem a data/hora

flow_df['Seconds'] = flow_df.index.map(pd.Timestamp.timestamp)
flow_df
```

As variáveis necessárias para o tempo devem dar-nos o momento do dia e o momento do ano. Assim conseguimoes implementar sazonalidade diária e anual. A metodologia utilizada traduz-se no seguinte:

Momento do dia:

- O momento do dia será dado por duas variáveis calculadas como o seno e o conseno do Timestamp x 2xPI/numero de segundos de um dia.

São necessárias duas variáveis se apenas usassemos o seno por exemplo, não saberiamos se as 12h eram meio dia ou meia noite e com a introdução do coseno isto já não acontece

Momento do ano:

- O momento do ano será dado por duas variáveis calculadas como o seno e o conseno do Timestamp x 2xPI/numero de segundos de um ano.

In [ ]:
```
day = 60 * 60 * 24
year = 365.2422 * day

flow_df['Day sin'] = np.sin(flow_df['Seconds'] * (2 * np.pi / day))
flow_df['Day cos'] = np.cos(flow_df['Seconds'] * (2 * np.pi / day))
flow_df['Year sin'] = np.sin(flow_df['Seconds'] * (2 * np.pi / year))
flow_df['Year cos'] = np.cos(flow_df['Seconds'] * (2 * np.pi / year))
```

In [ ]:
```
flow_df.head()
```

In [ ]:
```
flow_df = flow_df.drop('Seconds', axis=1)
flow_df.head()
```

5.1.6.2. Padronização dos dados de caudal

As variáveis correspondentes ao tempo são valores entre -1 e 1, já os dados de caudal são valores positivos com uma escala muito diferente. Esta diferença de escalas pode dificultar a operecionalidade do modelo pelo que procederemos à padronização dos dados de caudal.

Neste ponto foi testada também a normalização do caudal (0 a 1) mas não melhorou os resultados.

In [ ]:
```
scaler_mv = StandardScaler()
```

In [ ]:
```
flow_df['Saida Std'] = scaler_mv.fit_transform(df[['SAIDA']])
flow_df
```

5.1.6.3. Divisão dos dados entre treino, validação e teste

In [ ]:
```
# Dividir os dados em conjuntos de treino, validação e teste.
# O modelo utilizará os dados de treino e validação.
# Os dados de teste serão depois previstos.

train_data_mv = flow_df[
    flow_df.index <= '2022-06-30 23:00:00'][
        ['Saida Std', 'Day sin', 'Day cos', 'Year sin', 'Year cos']]
val_data_mv = flow_df[
    (flow_df.index > '2022-06-30 23:00:00') &
    (flow_df.index <= '2022-09-30 23:00:00')][
        ['Saida Std', 'Day sin', 'Day cos', 'Year sin', 'Year cos']]
test_data_mv = flow_df[
    (flow_df.index > '2022-09-30 23:00:00') &
    (flow_df.index <= '2022-12-31 23:00:00')][
        ['SAIDA', 'Day sin', 'Day cos', 'Year sin', 'Year cos']]

train_data_mv.name = "SAIDA"
val_data_mv.name = "SAIDA"
test_data_mv.name = "SAIDA"

# número de dados total e pertencentes ao conjunto de treino, validação e teste
flow_df.count(), train_data_mv.count(), val_data_mv.count(), test_data_mv.count()
```

In [ ]:
```
train_data_mv
```

In [ ]:
```
val_data_mv
```

5.1.6.4. Transformação matricial dos dados de treino e de validação respeitando o conceito de multivariável

In [ ]:
```
# time step
WINDOW_SIZE = 24

Xtrain_mv, ytrain_mv = df_to_X_y_multivariate(train_data_mv, WINDOW_SIZE)
Xval_mv, yval_mv = df_to_X_y_multivariate(val_data_mv, WINDOW_SIZE)

Xtrain_mv.shape, ytrain_mv.shape, Xval_mv.shape, yval_mv.shape
```

5.1.6.5. Arquitetura do modelo

Propõe-se uma arquitetura simples com uma layer de entrada com os valores do WINDOW_SIZE, uma layer LSTM com 48 nós e apenas um nó de saída que nos dá o valor presvisto. Foram testados outros números de nós na layer LSTM. 48 foi o número que produzia melhores resultados. Foram testadas a inclusões de layers Dense com 24 nós antes e depois da layer LSTM mas aumentava a complexidade do modelo e não melhorava os resultados.

In [ ]:
```
model_mv = Sequential()
model_mv.add(InputLayer((WINDOW_SIZE, 5)))
model_mv.add(LSTM(48))
model_mv.add(Dense(1))
```



```
model_mv.summary()
```

5.1.6.6. Compilação

In [ ]:
```
check_point_mv = ModelCheckpoint(
    'best_model_mv.h5',
    save_best_only=True
)

model_mv.compile(
    loss=MeanSquaredError(),
    optimizer=Adam(learning_rate=0.001),
    metrics=[RootMeanSquaredError()]
)
```

In [ ]:
```
r_mv = model_mv.fit(Xtrain_mv, ytrain_mv,
                    validation_data=(Xval_mv, yval_mv),
                    epochs=100,
                    callbacks=[check_point_mv])
```

5.1.6.7. Avaliação do modelo

In [ ]:
```
plt.plot(r_mv.history['loss'], label='train loss')
plt.plot(r_mv.history['val_loss'], label='val loss')
plt.legend();
```

In [ ]:
```
#load best model
mymodel_mv = tf.keras.models.load_model('best_model_mv.h5')
```

5.1.6.8. Previsão dos dados de treino e de validação

Interessa agora utilizar os dados de treino e validação e verificar se o modelo consegue fazer boas previsões. Como o modelo foi treinado com estes valores é previsível que os resultados sejam bons.

In [ ]:
```
# Previsão para os dados de treino e de validação usando o modelo

Ptrain_mv = mymodel_mv.predict(Xtrain_mv).flatten()
Pval_mv = mymodel_mv.predict(Xval_mv).flatten()
```

Nestas previsões existem valores que não são previsíveis, ou seja, os primeiros valores com o tamanho da janela temporal WINDOW_SIZE não são previsíveis. Assim, e por forma a ter séries previstas completas, os valores que estão em falta serão preenchidos pelos valores reais.

In [ ]:
```
# In train predictions first WINDOW_SIZE data are not predictable, so it will be completed wi
# In val predictions first WINDOW_SIZE data are not predictable, so it will be completed with

train_data_mv_serie = train_data_mv['Saida Std'].to_numpy()
val_data_mv_serie = val_data_mv['Saida Std'].to_numpy()

train_vector_mv = train_data_mv_serie[:WINDOW_SIZE]
Ptrain_mv = np.concatenate((train_vector_mv, Ptrain_mv))

val_vector_mv = val_data_mv_serie[:WINDOW_SIZE]
Pval_mv = np.concatenate((val_vector_mv, Pval_mv))

Ptrain_mv.shape, Pval_mv.shape
```

In [ ]:
```
# Create Panda Series

train_predictions_mv_undo_std = pd.Series(
    data=scaler_mv.inverse_transform(Ptrain_mv.reshape(-1,1)).flatten(),
    index=train_data.index)
train_predictions_mv_undo_std.name = "SAIDA"

val_predictions_mv_undo_std = pd.Series(
    data=scaler_mv.inverse_transform(Pval_mv.reshape(-1,1)).flatten(),
    index=val_data.index)
val_predictions_mv_undo_std.name = "SAIDA"
```

In [ ]:
```
graph("Série Temporal de Alcáçova - Caudal de Saída (l/s)",
        (pd.DataFrame(train_data,
                      index=train_data.index),
                      "SAIDA",
                      "Dados de treino"),
        (pd.DataFrame(train_predictions_mv_undo_std,
                      index=train_predictions_mv_undo_std.index),
                      "SAIDA",
                      "Dados de treino previstos pelo modelo"),
        (pd.DataFrame(val_data,
                      index=val_data.index),
                      "SAIDA",
                      "Dados de validação"),
        (pd.DataFrame(val_predictions_mv_undo_std,
                      index=val_predictions_mv_undo_std.index),
                      "SAIDA",
                      "Dados de validação previstos pelo modelo"),
)
```

5.1.6.9. Avaliação das previsões de treino e de validação

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_predictions_mv_undo_std,
                 test_data=val_data,
                 predictions=val_predictions_mv_undo_std)
```

Tal como previsto os resultados das métricas são bons mas as métricas apresentadas anteriormente apenas nos dão boas indicações de que o modelo consegue prever eficazmente os dados de treino e de validação, mas não nos dão nenhuma indicação sobre a sua capacidade de prever valores novos, ou seja, valores que o modelo não conheceu durante a sua "aprendizagem".

5.1.6.10. Previsões de valores novos - Multistep forecast

Tal como referido inicialmente, existem 3 meses de dados de teste entre Outubro de 2022 e Dezembro de 2022 que não foram dados a conhecer ao modelo. São estes os dados que serão previstos pelo modelo e depois comparados com os valores reais.

O modelo irá efetuar uma previsão do valor seguinte pegando sempre nos 24 valores anteriores conhecidos. Inicialmente começará a utilizar os valores reais do final do Xval_mv que irão sendo substituídos por novos valores previstos. A partir de determienada altura estaremos a fazer previsões com base em previsões utilizando apenas o modelo treinado anteriormente.

In [ ]:
```
multistep_predictions_mv = []

last_x_mv = Xval_mv[-1]
last_x_mv = last_x_mv[np.newaxis,:]
i=0
while len(multistep_predictions_mv) < test_data.count():
    p_mv = mymodel_mv.predict(last_x_mv)

    # update the predictions list
    multistep_predictions_mv.append(p_mv)

    # neste caso é necessário calcular o timestamp do valor previsto
    # para o incluir e ser usado nas próximas previsões
```



```
    time_mv = pd.Timestamp.timestamp(test_data_mv.index[i])
    day_sin = np.sin(time_mv * (2 * np.pi / day))
    day_cos = np.cos(time_mv * (2 * np.pi / day))
    year_sin = np.sin(time_mv * (2 * np.pi / year))
    year_cos = np.cos(time_mv * (2 * np.pi / year))

    p_mv = np.append(p_mv, [day_sin, day_cos, year_sin, year_cos])
    i = i + 1

    # make the new input
    last_x_mv = np.roll(last_x_mv, -5)
    last_x_mv[0][-1] = p_mv
```

In [ ]:
```
predictions_mv = [array[0][0] for array in multistep_predictions_mv]
predictions_mv
```

In [ ]:
```
predictions_mv_undo_std = scaler_mv.inverse_transform(
    np.array(predictions_mv).reshape(-1,1))
predictions_mv_undo_std
```

In [ ]:
```
# Create Panda Serie

test_predictions_mv_undo_std = pd.Series(
    data=(predictions_mv_undo_std.flatten()),
    index=test_data.index)
test_predictions_mv_undo_std.name = "SAIDA"
```

In [ ]:
```
graph("Série Temporal de Alcáçova - Caudal de Saída (l/s)",
        (pd.DataFrame(train_data,
                      index=train_data.index),
                      "SAIDA",
                      "Dados de treino"),
        (pd.DataFrame(train_predictions_mv_undo_std,
                      index=train_predictions_mv_undo_std.index),
                      "SAIDA",
                      "Dados de treino previstos pelo modelo"),
        (pd.DataFrame(val_data, i
                      ndex=val_data.index),
                      "SAIDA",
                      "Dados de validação"),
        (pd.DataFrame(val_predictions_mv_undo_std,
                      index=val_predictions_mv_undo_std.index),
                      "SAIDA",
                      "Dados de validação previstos pelo modelo"),
        (pd.DataFrame(test_data,
                      index=test_data.index),
                      "SAIDA",
                      "Dados de teste"),
        (pd.DataFrame(test_predictions_mv_undo_std,
                      index=test_predictions_mv_undo_std.index),
                      "SAIDA",
                      "Dados de teste previstos pelo modelo"),
)
```

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_predictions_mv_undo_std,
                 test_data=test_data,
                 predictions=test_predictions_mv_undo_std)
```

Conclui-se que o modelo apresenta bons resultados de previsão e que consegue explicar cerca de 80% dos valores reais. Percebe-se que a curva de tendencia segue a sazonalidade anual do ano anterior e, fazendo zoom, observa-se uma sazonalidade diária quase perfeita e sem desfazamentos.

Comparação dos 3 Modelos

MODELO UNIVARIÁVEL DADOS ORIGINAIS

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_predictions,
                 test_data=test_data,
```

```
                predictions=test_predictions)
```

MODELO UNIVARIÁVEL DADOS PADRONIZADOS

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_predictions_std,
                 test_data=test_data,
                 predictions=test_predictions_std)
```

MODELO MULTIVARIÁVEL DADOS PADRONIZADOS

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_predictions_mv_undo_std,
                 test_data=test_data,
                 predictions=test_predictions_mv_undo_std)
```

5.1.7. Análise dos resultados para uma previsão de curto prazo (10 dias)

Considerou-se importante verificar a performance dos modelos para um período mais curto uma vez que períodos muito longos podem potenciar a ocorrência de erros de previsão com a consequente desistência do modelo em causa quando o mesmo pode fazer boas previsões de curto prazo e, em algumas situações, serão essas as mais importantes. Considerou-se que o período mais curto seria de 10 dias pois do ponto de vista de análise e de desenvolvimento futuro deste trabalho será esse o período mínimo.

Assim sendo, uma vez que temos um registo por hora, estamos interessados em comparar os primeiros 24x10 = 240 valores.

5.1.7.1. Avaliação do modelo para uma previsão de curto prazo

Com dados originais

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_predictions,
                 test_data=test_data.iloc[:240],
                 predictions=test_predictions.iloc[:240])
```

Com dados padronizados

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_predictions_std,
                 test_data=test_data.iloc[:240],
                 predictions=test_predictions_std.iloc[:240])
```

Com multivariáveis e dados padronizados

In [ ]:
```
evaluate_metrics(train_data=train_data, t
                 rain_data_fitted=train_predictions_mv_undo_std,
                 test_data=test_data.iloc[:240],
                 predictions=test_predictions_mv_undo_std.iloc[:240])
```

No caso de uma previsão de curto prazo, os modelos que antes não permitiam fazer qualquer previsão devido ao desfasamento entre os dados de teste e os dados previstos melhoram, pois para os primeiros dias o desfasamento não é muito evidente. Ainda assim o modelo que utiliza multivariáveis continua a ser o único que produz previsões aceitáveis.

In [ ]:
```
# NoteBook:     Prophet-Alcacova.ipynb
# Autor:        Carlos Pires
# Data:         30-12-2023
# Descrição:    Este notebook efetua as operações necessárias para aplicar o Modelo
#               Prophet à série temporal de Alcáçova.
```

6. Modelo Prophet

In [ ]:
```
# importação de bibliotecas

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 15, 6

# Prophet
from prophet import Prophet

# Statistic metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
# grid view formatting
from tabulate import tabulate

# graphics
import plotly.graph_objects as go
```

Definição de funções de apoio

In [ ]:
```
# Criar o gráfico interativo com a biblioteca Plotly para qualquer numero de dataframes

def graph(title_name="NO NAME", *args):

    fig = go.Figure()

    for i, arg in enumerate(args):
        dataframe = arg[0]
        attribute = arg[1]
        serie_name = arg[2]
        fig.add_trace(go.Scatter(x=dataframe.index,
                                 y=getattr(dataframe, attribute),
                                 name=serie_name,
                                 mode='lines',
                                 showlegend=True))

    # Configurar os eixos e o layout do gráfico
    fig.update_layout(
        xaxis=dict(title='Data'),
        yaxis=dict(title='Caudal (l/s)'),
        title=title_name
    )

    # Habilitar o zoom interativo
    fig.update_layout(
        xaxis=dict(
            rangeselector=dict(
                buttons=list([
                    dict(count=1, label='1m', step='month', stepmode='backward'),
                    dict(count=6, label='6m', step='month', stepmode='backward'),
                    dict(count=1, label='1y', step='year', stepmode='backward'),
                    dict(count=3, label='3y', step='year', stepmode='backward'),
                    dict(step='all')
                ])
            ),
            rangeslider=dict(visible=True),
            type='date'
        )
    )
```

```
    # Mostrar o gráfico
    fig.show()
```

In [ ]:
```
# Função de cálculo e apresentação de métricas

def evaluate_metrics(train_data, train_data_fitted, test_data, predictions):

    mae_train = mean_absolute_error(train_data, train_data_fitted)
    mae_predict = mean_absolute_error(test_data, predictions)

    rmse_train = np.sqrt(mean_squared_error(train_data, train_data_fitted))
    rmse_predict = np.sqrt(mean_squared_error(test_data, predictions))

    r2_train = r2_score(train_data, train_data_fitted)
    r2_predict = r2_score(test_data, predictions)

    data = {
        'Dados': ['Treino', 'Teste'],
        'MAE': [mae_train, mae_predict],
        'RMSE': [rmse_train, rmse_predict],
        'R2': [r2_train, r2_predict]
    }

    df_evaluate = pd.DataFrame(data)

    print('MÉTRICAS DE AVALIAÇÃO DO MODELO')
    print(tabulate(df_evaluate, headers='keys', tablefmt='pretty', showindex=False))
```

6.1. Facebook Prophet - Série Temporal de Alcáçova

O Facebook Prophet é um modelo de previsão para séries temporais baseado num modelo aditivo onde tendências não lineares são ajustadas com sazonalidades que podem ser anuais, semanais e/ou diárias, para além dos efeitos gerados pelos periodos festivos. Funciona melhor com séries temporais com fortes efeitos sazonais e vários periodos de dados históricos. É robusto a lidar com dados em falta e mudanças de tendências bem como a lidar com outliers.

6.1.1. Componentes do modelo Prophet

- Equação de previsão:

$$y(t) = g(t) + s(t) + h(t) + \epsilon(t)$$

onde:

- $g(t)$ - Tendência (representa as mudanças não periódicas);
- $s(t)$ - Sazonalidade (representa as mudanças periódicas que podem acontecer a diferentes escalas: diárias, semanais e/ou anuais);
- $h(t)$ - Dias festivos;
- $\epsilon(t)$ - Erro (É assumido como tendo uma distribuição normal).

6.1.1.1. Equação de tendência

- Linear (declive x tempo + ordenada na origem)

$$g(t) = (k + a(t)^T\delta)t + (m + a(t)^T\gamma)$$

- Logística (C(t) x sigmoid(linear(t)))

$$g(t) = \frac{C(t)}{1 + exp(-(k + a(t)^T\delta)(t - (m + a(t)^T\gamma)))}$$

6.1.1.2. Equação de Sazonalidade

A equação de sazonalidade é obtida tendo como base uma série de Fourier.

$$s(t) = \sum_{n=1}^{N}\left(a_n cos\left(\frac{2\pi nt}{P}\right) + b_n sin\left(\frac{2\pi nt}{P}\right)\right)$$

O parâmetro P é o periodo. Por exemplo para dados diários a sazonalidade anual seria obtida com P = 365.25 e a sazonalidade semanal com P = 7.

6.1.2. Importação dos dados

In [ ]:
```
df = pd.read_csv('../DadosPreProcessados/Alcacova.csv',
                 sep=',',
                 index_col='Data/Hora',
                 parse_dates=True)
```

In [ ]:
```
graph("Série Temporal de Alcáçova - Caudal de Saída (l/s)",
      (df, "SAIDA", "Dados reais de caudal"))
```

6.1.3. Separação dos dados entre treino e teste

Para além da separação dos dados é necessário criar um dataset que tenha como colunas "ds", com as datas/horas e "y" com os dados respetivos. Este aspeto é um requisito do modelo.

In [ ]:
```
# Dividir os dados em conjuntos de treino e de teste

train_data = df[df.index <= '2022-09-30 23:00:00']['SAIDA']
test_data = df[df.index > '2022-09-30 23:00:00']['SAIDA']

train_data.name = "SAIDA"
test_data.name = "SAIDA"

train_data.index.freq = "H"
test_data.index.freq = "H"

train_data_df = pd.DataFrame(train_data, index=train_data.index)
train_data_df['ds'] = train_data_df.index
train_data_df.columns = ['y', 'ds']

test_data_df = pd.DataFrame(test_data, index=test_data.index)
test_data_df['ds'] = test_data_df.index
test_data_df.columns = ['y', 'ds']
```

In [ ]:
```
train_data_df
```

In [ ]:
```
test_data_df
```

6.1.4. Ajuste e aplicação do modelo aos dados

In [ ]:
```
# define-se um modelo linear com sazonalidade anual e diária.
# O número de changepoints foi escolhido por forma a melhorar os resultados
# o changepoint_prior_scale foi aumentado para aumentar a flexibilidade do
# modelo em relação à tendência.

model = Prophet(growth='linear',
            yearly_seasonality=True,
            weekly_seasonality=False,
            daily_seasonality=True,
            n_changepoints=650,
            changepoint_prior_scale=0.09
            )
```

In [ ]:
```
model.fit(train_data_df)
```

6.1.4.1. Obtenção dos dados previstos pelo modelo

In [ ]:
```
future = model.make_future_dataframe(periods=2208, freq='H', include_history=True)
```

In [ ]:
```
forecast = model.predict(future)
```

In [ ]:
```
forecast
```

In [ ]:
```
model.plot(forecast, uncertainty=True);
```

Componentes do modelo

In [ ]:
```
model.plot_components(forecast);
```

6.1.4.2. Visualização gráfica dos dados

In [ ]:
```
train_data = df[df.index <= '2022-09-30 23:00:00']['SAIDA']
test_data = df[df.index > '2022-09-30 23:00:00']['SAIDA']
```

In [ ]:
```
train_data_adjusted = forecast[
    forecast.ds <= '2022-09-30 23:00:00'][['yhat','ds']]
train_data_adjusted.index = train_data_adjusted.ds
train_data_adjusted.drop('ds', axis=1, inplace=True)


predictions = forecast[
    forecast.ds > '2022-09-30 23:00:00'][['yhat', 'ds']]
predictions.index = predictions.ds
predictions.drop('ds', axis=1, inplace=True)
```

In [ ]:
```
graph("Série Temporal de Alcáçova - Caudal de Saída (l/s)",
        (pd.DataFrame(train_data,
                      index=train_data.index),
                      "SAIDA",
                      "Dados de treino"),
        (pd.DataFrame(train_data_adjusted,
                      index=train_data.index),
                      "yhat",
                      "Dados de treino ajustados"),
        (pd.DataFrame(test_data,
                      index=test_data.index),
                      "SAIDA",
                      "Dados de teste"),
        (pd.DataFrame(predictions,
                      index=test_data.index),
                      "yhat",
                      "Previsões")
)
```

6.1.4.3. Avaliação do modelo

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_data_adjusted,
                 test_data=test_data,
                 predictions=predictions)
```

6.1.5. Análise dos resultados para uma previsão de curto prazo (10 dias)

Considerou-se importante verificar a performance dos modelos para um período mais curto uma vez que períodos muito longos podem potenciar a ocorrência de erros de previsão com a consequente desistência do modelo em causa quando o mesmo pode fazer boas previsões de curto prazo e, em algumas situações, serão essas as mais importantes. Considerou-se que o período mais curto seria de 10 dias pois do ponto de vista de análise e de desenvolvimento futuro deste trabalho será esse o período mínimo.

Assim sendo, uma vez que temos um registo por hora, estamos interessados em comparar os primeiros 24x10 = 240 valores.

6.1.5.1. Avaliação do modelo para uma previsão de curto prazo

In [ ]:
```
evaluate_metrics(train_data=train_data,
                 train_data_fitted=train_data_adjusted,
                 test_data=test_data.iloc[:240],
                 predictions=predictions.iloc[:240])
```

ANEXO 2 – Figuras dos Gráficos Obtidos Durante o Processamento

2. FIGURAS DOS GRÁFICOS OBTIDOS DURANTE O PROCESSAMENTO

2.1. Figuras dos Gráficos Obtidos por Aplicação do Modelo *Holt-Winters*

De seguida apresentam-se todas as figuras com os gráficos obtidos durante a aplicação do modelo *Holt-Winters*, não representadas no texto principal.

2.1.1. Série Temporal de Alcáçova

Figura 1 – Previsões do Modelo Holt-Winters para Alcáçova com Dados Originais

Fonte: Elaboração Própria

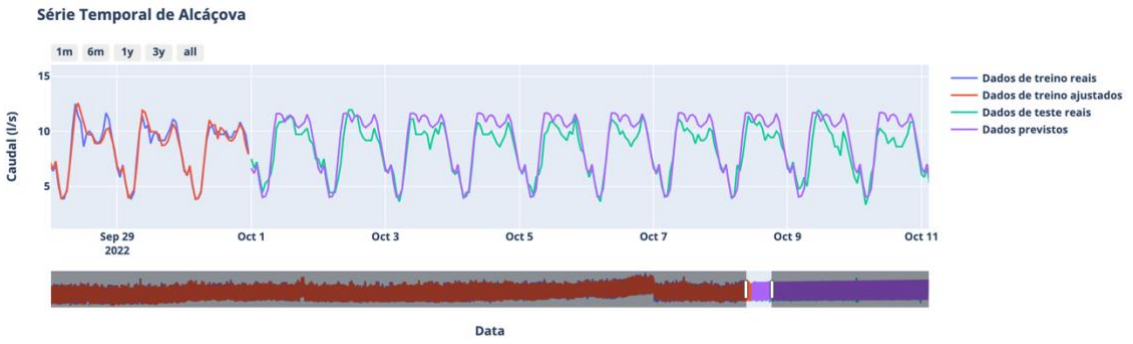
Figura 2 – Ampliação das Previsões do Modelo Holt-Winters para Alcáçova com Dados Originais

Fonte: Elaboração Própria

Figura 3 – Previsões do Modelo Holt-Winters para Alcáçova com Dados Transformados por Raiz Cúbica

Fonte: Elaboração Própria

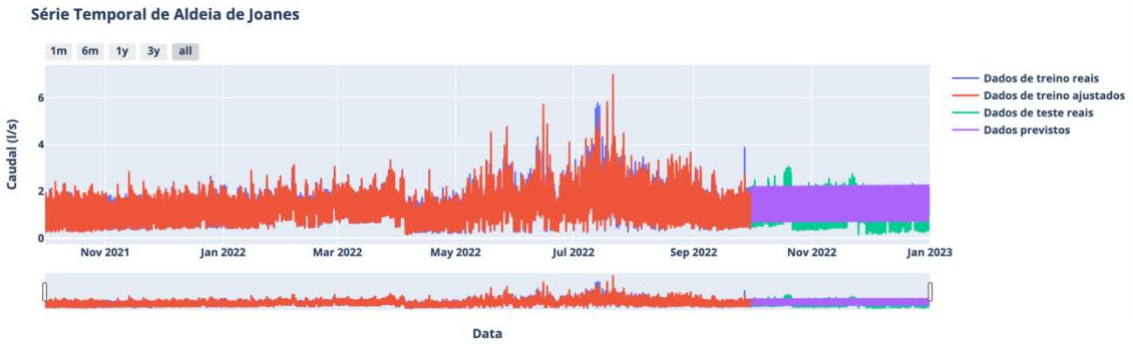
Figura 4 – Ampliação das Previsões do Modelo Holt-Winters para Alcáçova com Dados Transformados por Raiz Cúbica



Fonte: Elaboração Própria

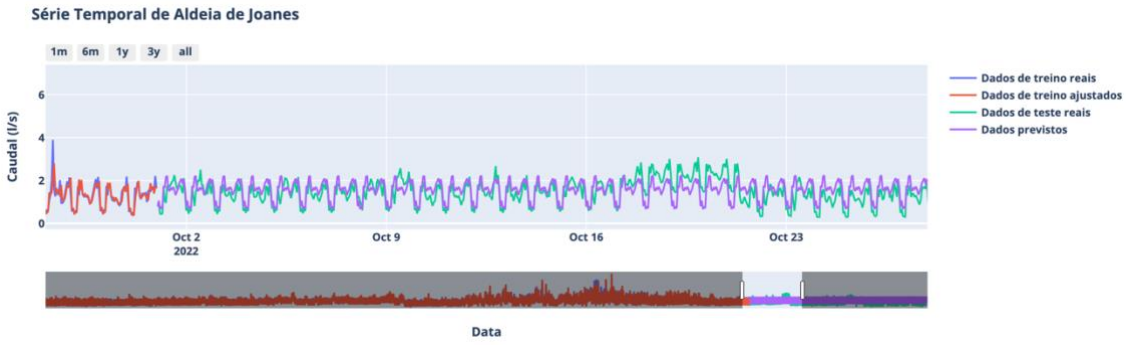
2.1.2. Série Temporal de Aldeia de Joanes

Figura 5 – Previsões do Modelo Holt-Winters para Aldeia de Joanes com Dados com Transformação Logarítmica



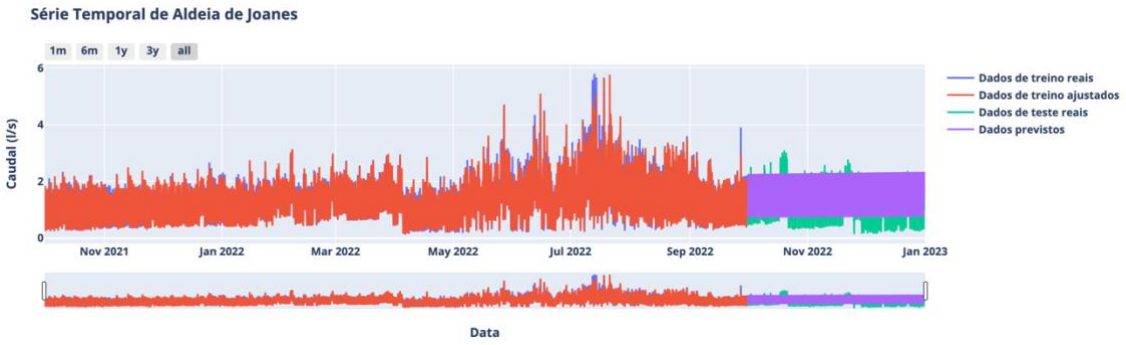
Fonte: Elaboração Própria

Figura 6 – Ampliação das Previsões do Modelo Holt-Winters para Aldeia de Joanes com Dados com Transformação Logarítmica



Fonte: Elaboração Própria

Figura 7 – Previsões do Modelo Holt-Winters para Aldeia de Joanes com Dados com Transformação por Raiz Cúbica



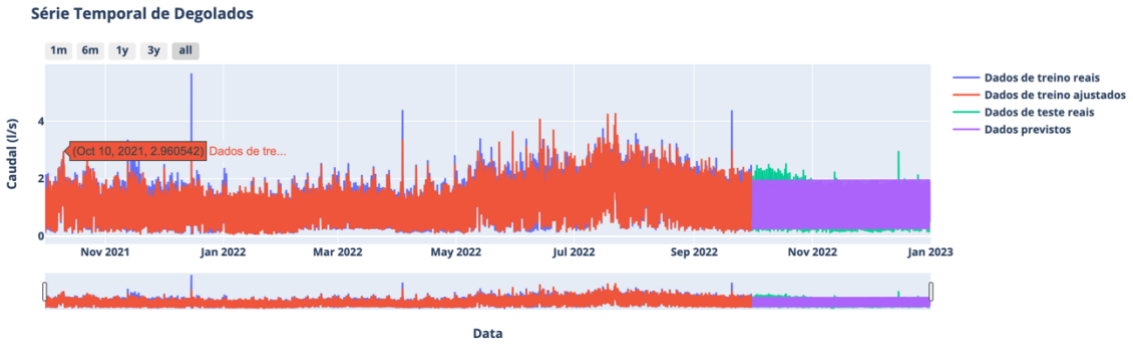
Fonte: Elaboração Própria

Figura 8 – Ampliação das Previsões do Modelo Holt-Winters para Aldeia de Joanes com Dados com Transformação por Raiz Cúbica

Fonte: Elaboração Própria

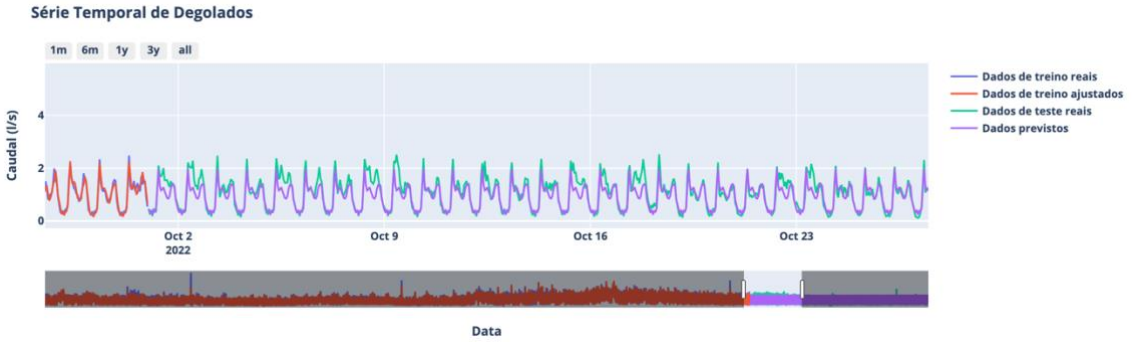
2.1.3. Série Temporal de Degolados

Figura 9 – Previsões do Modelo Holt-Winters para Degolados com Dados com Transformação Logarítmica



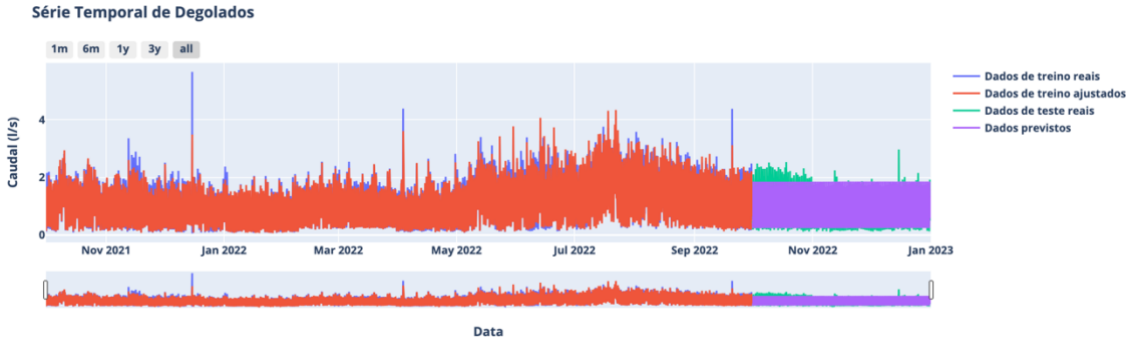
Fonte: Elaboração Própria

Figura 10 – Ampliação das Previsões do Modelo Holt-Winters para Degolados com Dados com Transformação Logarítmica



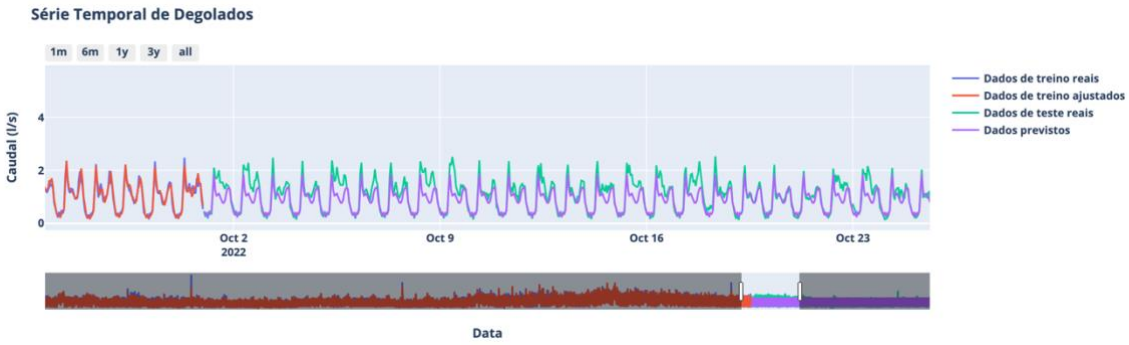
Fonte: Elaboração Própria

Figura 11 – Previsões do Modelo Holt-Winters para Degolados com Dados com Transformação por Raiz Cúbica



Fonte: Elaboração Própria

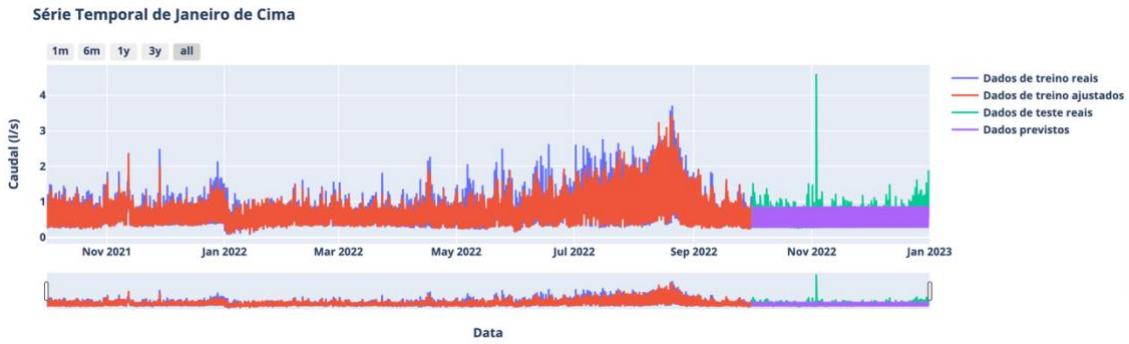
Figura 12 – Ampliação das Previsões do Modelo Holt-Winters para Degolados com Dados com Transformação por Raiz Cúbica



Fonte: Elaboração Própria

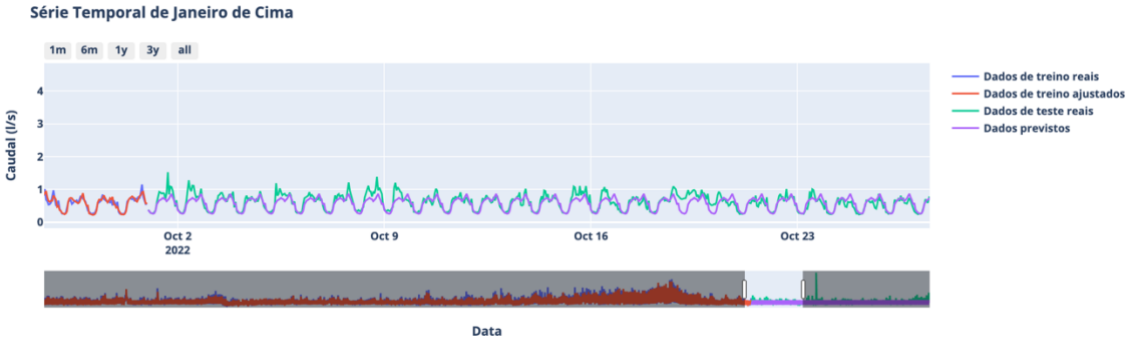
2.1.4. Série Temporal de Janeiro de Cima

Figura 13 – Previsões do Modelo Holt-Winters para Janeiro de Cima com Dados Originais



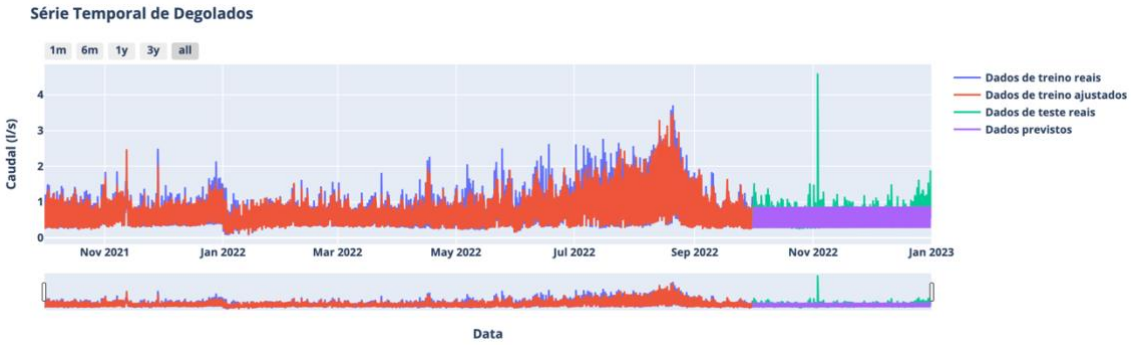
Fonte: Elaboração Própria

Figura 14 – Ampliação das Previsões do Modelo Holt-Winters para Janiro de Cima com Dados Originais



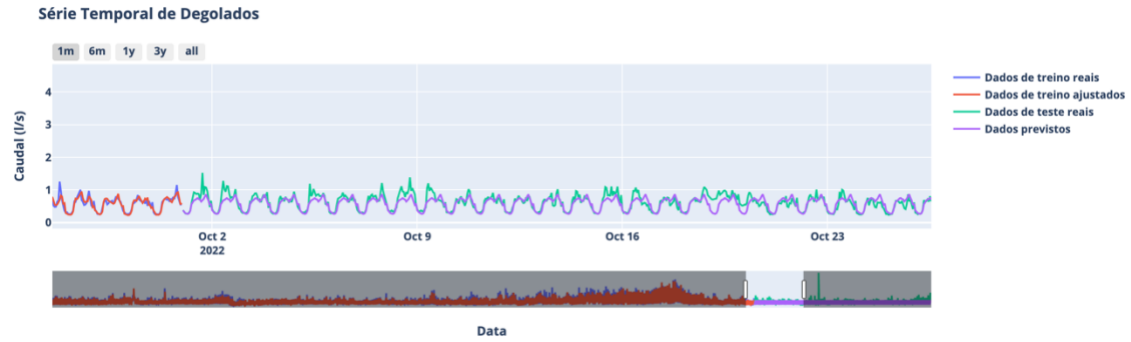
Fonte: Elaboração Própria

Figura 15 – Previsões do Modelo Holt-Winters para Janeiro de Cima com Dados com Transformação por Raiz Cúbica



Fonte: Elaboração Própria

Figura 16 – Ampliação das Previsões do Modelo Holt-Winters para Janeiro de Cima com Dados com Transformação por Raiz Cúbica



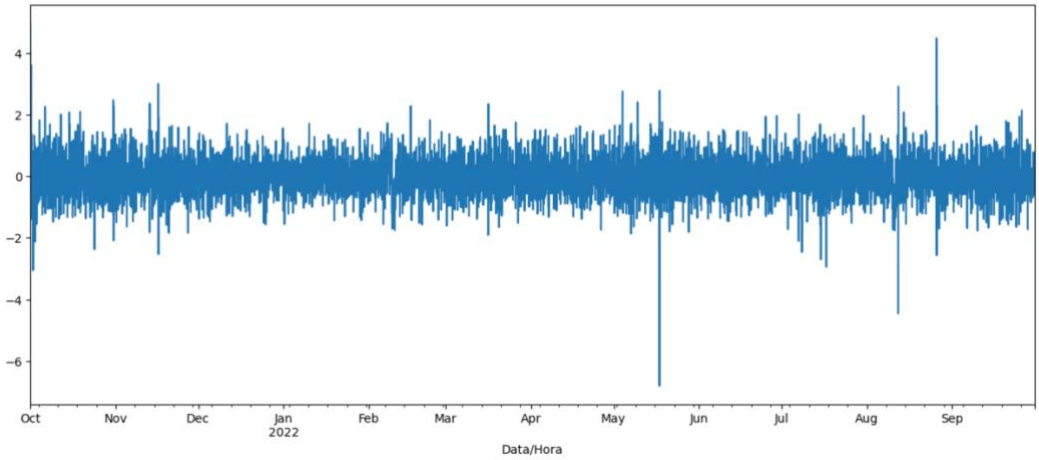
Fonte: Elaboração Própria

2.2. Figuras dos Gráficos Obtidos por Aplicação do Modelo ARIMA

De seguida apresentam-se todas as figuras com os gráficos obtidos durante a aplicação do modelo ARIMA, não representadas no texto principal.

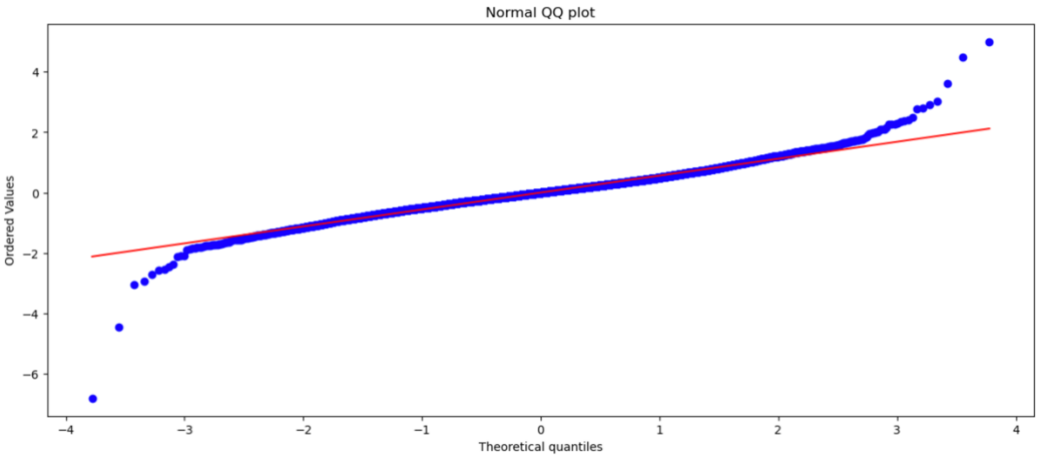
2.2.1. Série Temporal de Alcáçova

Figura 17 – Resíduos do Modelo ARIMA – Método Manual – Alcáçova



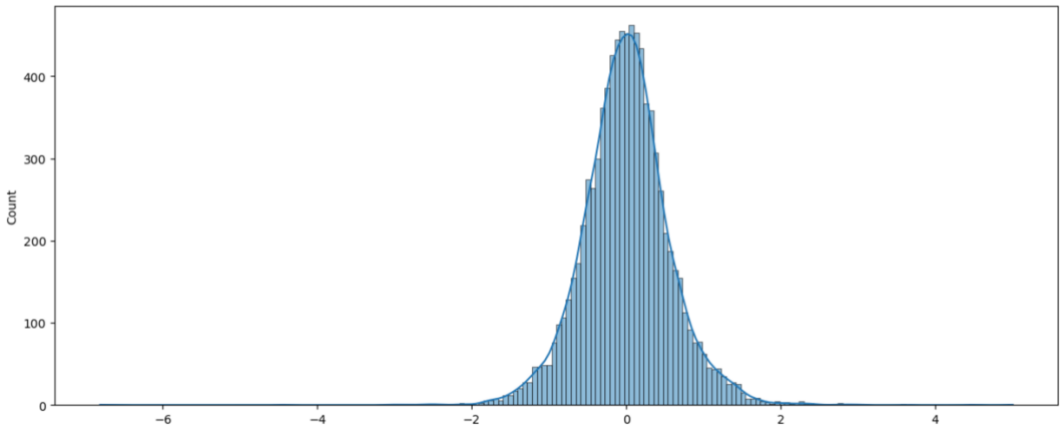
Fonte: Elaboração Própria

Figura 18 – Normal QQ Plot dos Resíduos do Modelo ARIMA – Método Manual – Alcáçova



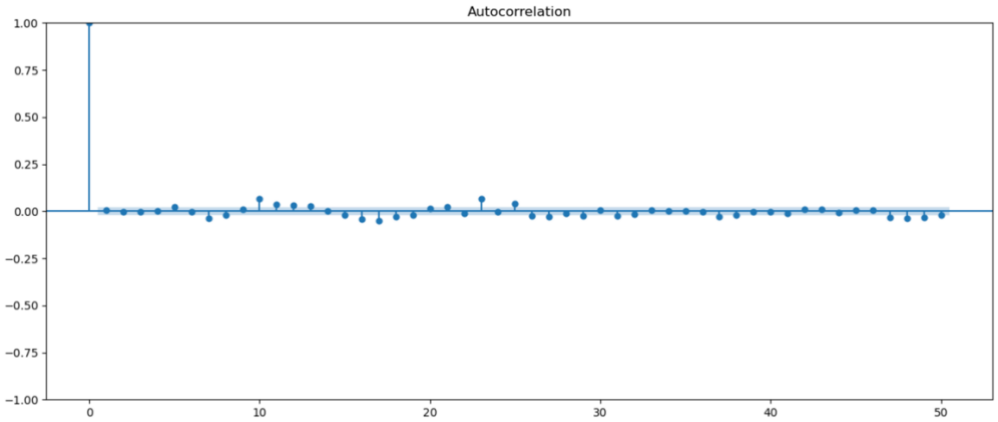
Fonte: Elaboração Própria

Figura 19 – Distribuição dos Resíduos do Modelo ARIMA – Método Manual - Alcáçova



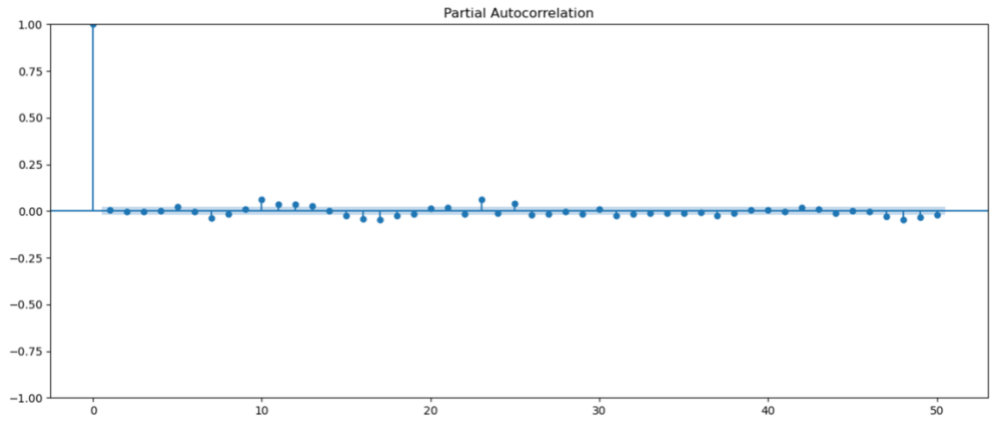
Fonte: Elaboração Própria

Figura 20 – Diagrama de Autocorrelação dos Resíduos do Modelo ARIMA – Método Manual - Alcáçova



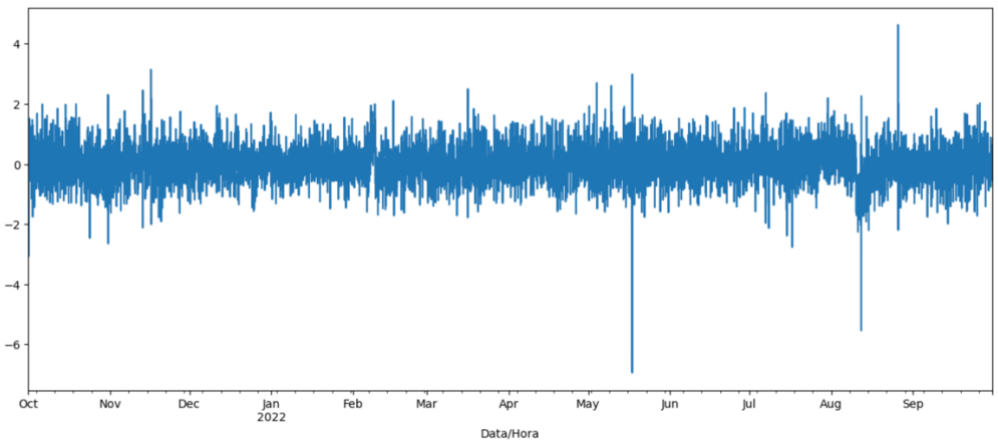
Fonte: Elaboração Própria

Figura 21 – Diagrama de Autocorrelação Parcial dos Resíduos do Modelo ARIMA – Método Manual - Alcáçova



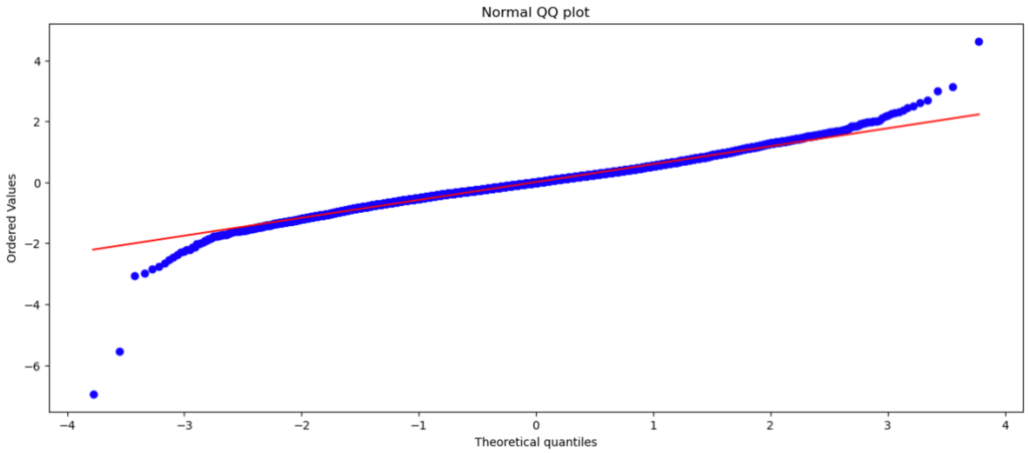
Fonte: Elaboração Própria

Figura 22 – Resíduos do Modelo ARIMA – Método Automático – Alcáçova



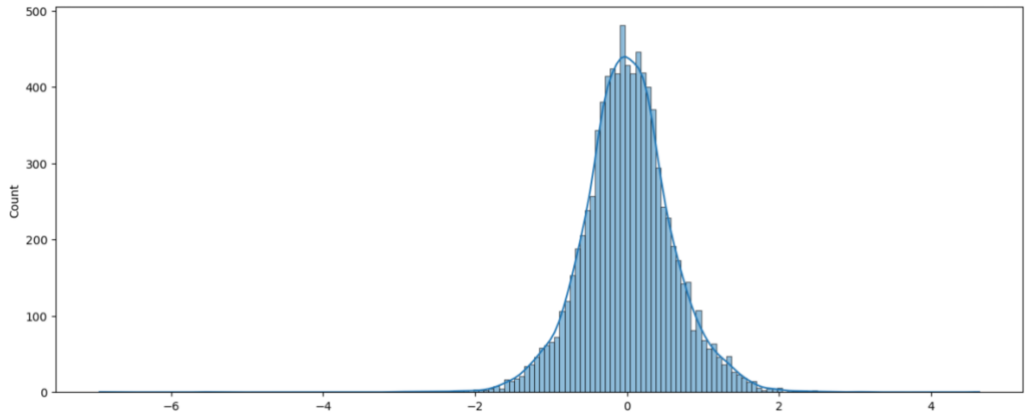
Fonte: Elaboração Própria

Figura 23 – Normal QQ Plot dos Resíduos do Modelo ARIMA – Método Automático – Alcáçova



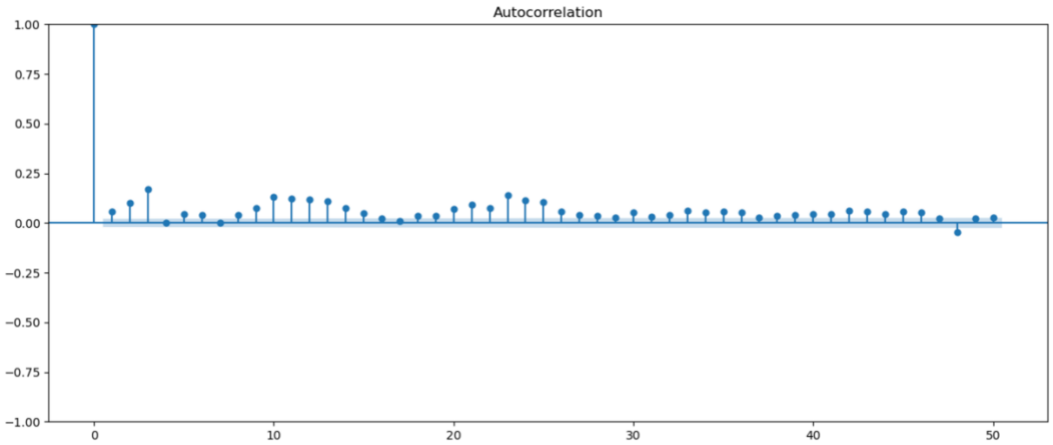
Fonte: Elaboração Própria

Figura 24 - Distribuição dos Resíduos do Modelo ARIMA – Método Automático - Alcáçova



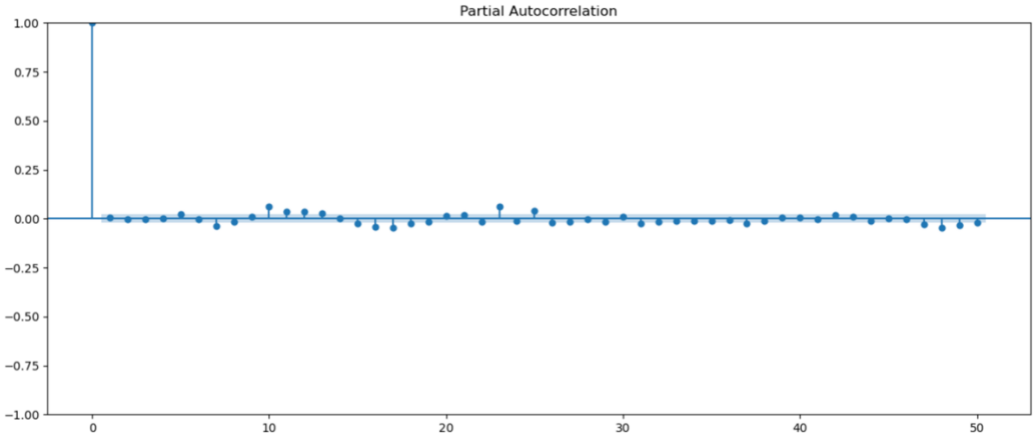
Fonte: Elaboração Própria

Figura 25 - Diagrama de Autocorrelação dos Resíduos do Modelo ARIMA – Método Automático – Alcáçova



Fonte: Elaboração Própria

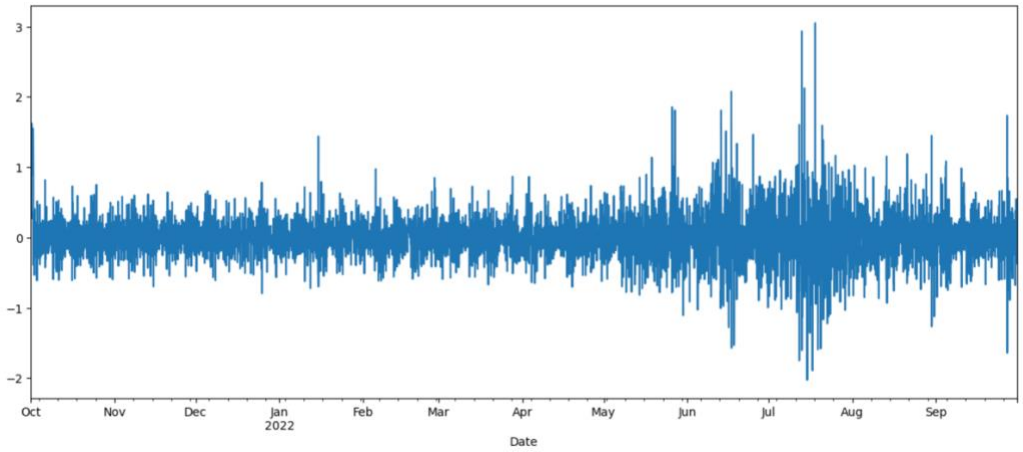
Figura 26 – Diagrama de Autocorrelação Parcial dos Resíduos do Modelo ARIMA – Método Automático – Alcáçova



Fonte: Elaboração Própria

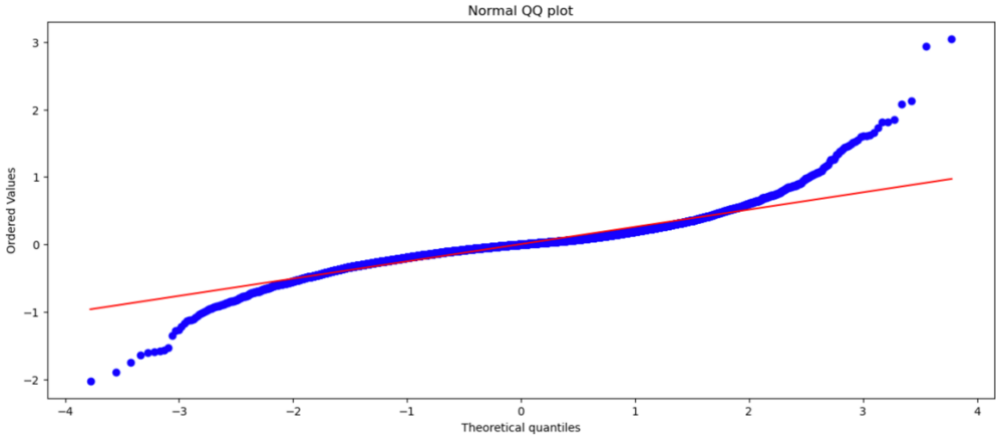
2.2.2. Série Temporal de Aldeia de Joanes

Figura 27 - Resíduos do Modelo ARIMA – Método Manual – Aldeia de Joanes



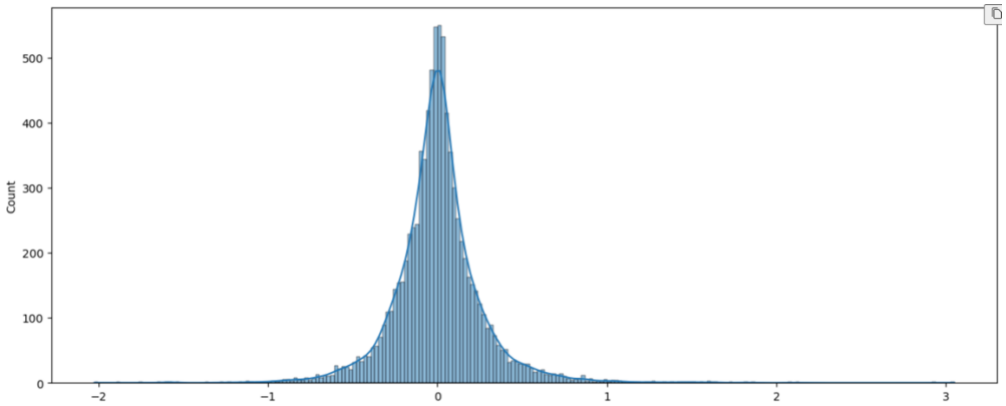
Fonte: Elaboração Própria

Figura 28 - Normal QQ Plot dos Resíduos do Modelo ARIMA – Método Manual – Aldeia de Joanes



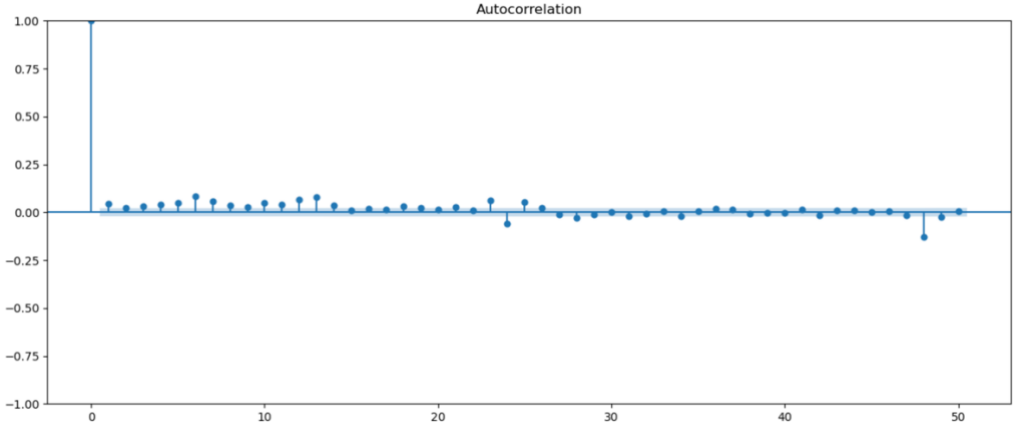
Fonte: Elaboração Própria

Figura 29 - Distribuição dos Resíduos do Modelo ARIMA – Método Manual – Aldeia de Joanes



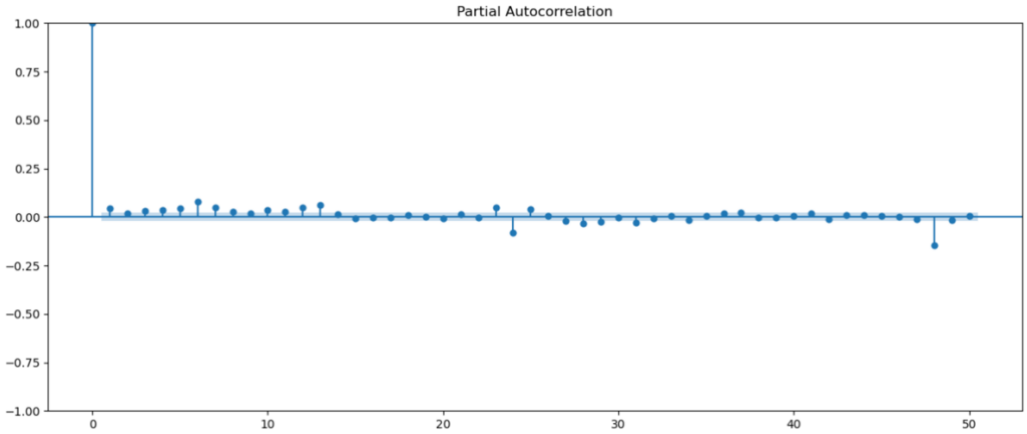
Fonte: Elaboração Própria

Figura 30 – Diagrama de Autocorrelação dos Resíduos do Modelo ARIMA – Método Manual – Aldeia de Joanes



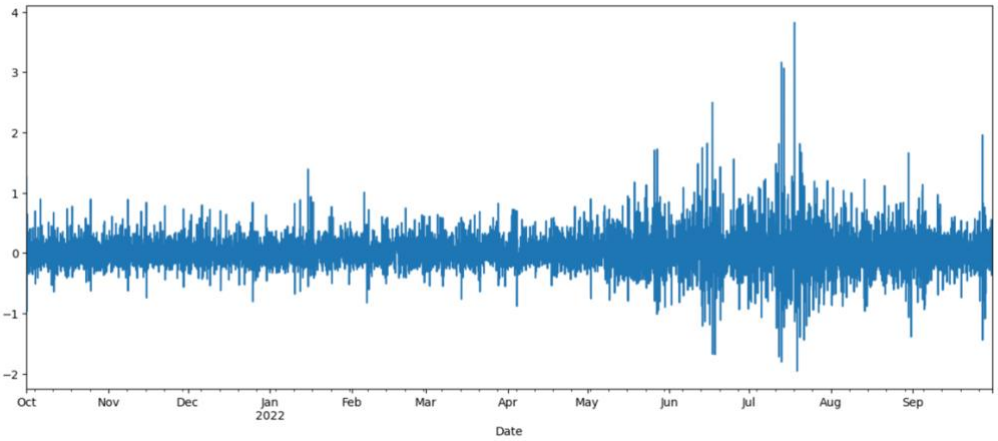
Fonte: Elaboração Própria

Figura 31 – Diagrama de Autocorrelação Parcial dos Resíduos do Modelo ARIMA – Método Manual – Aldeia de Joanes



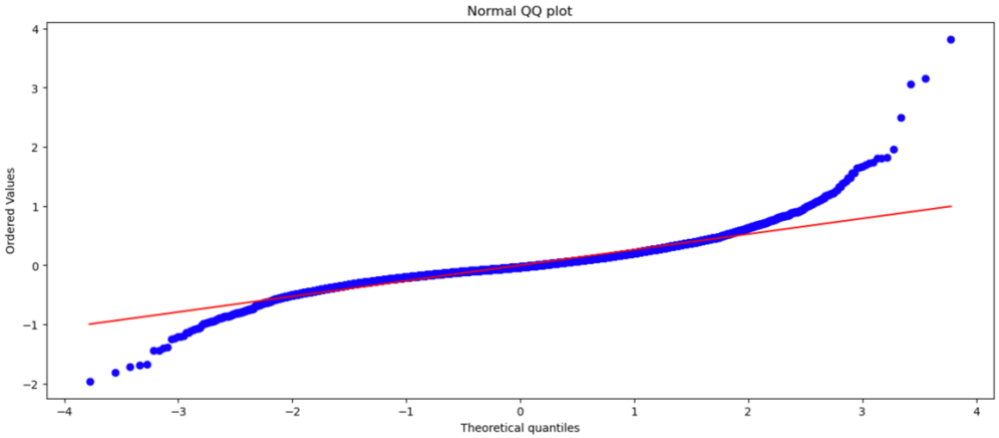
Fonte: Elaboração Própria

Figura 32 – Resíduos do Modelo ARIMA – Método Automático – Aldeia de Joanes



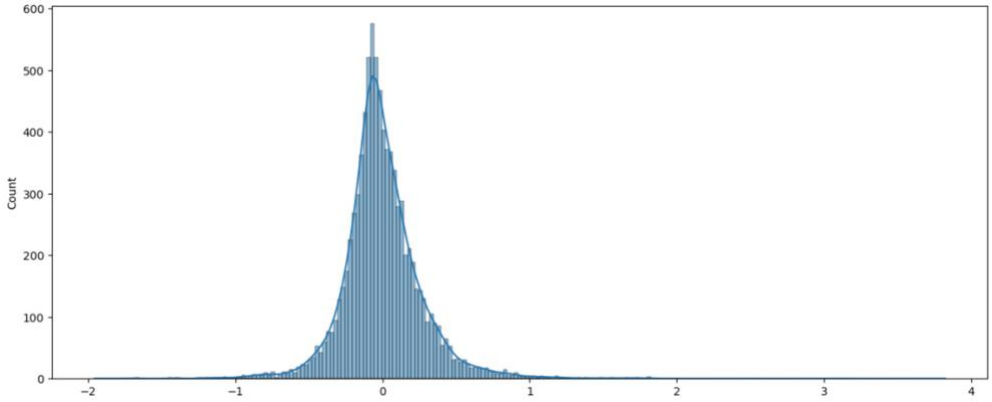
Fonte: Elaboração Própria

Figura 33 – Normal QQ Plot dos Resíduos do Modelo ARIMA – Método Automático – Aldeia de Joanes



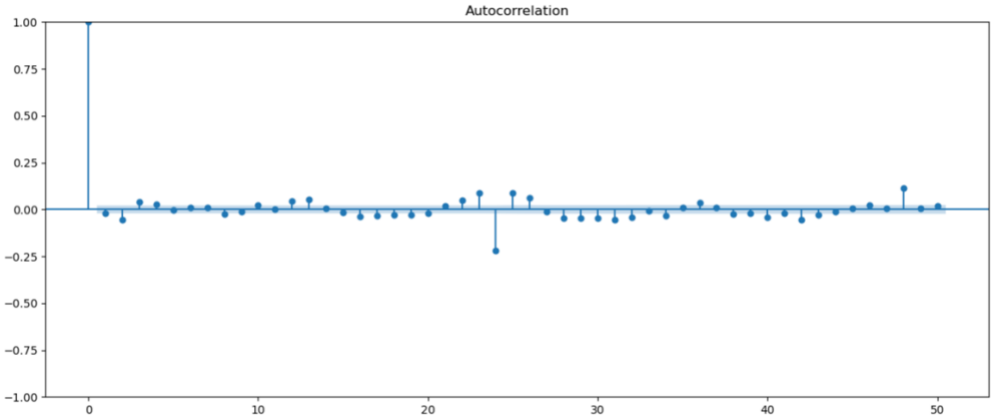
Fonte: Elaboração Própria

Figura 34 - Distribuição dos Resíduos do Modelo ARIMA – Método Automático – Aldeia de Joanes



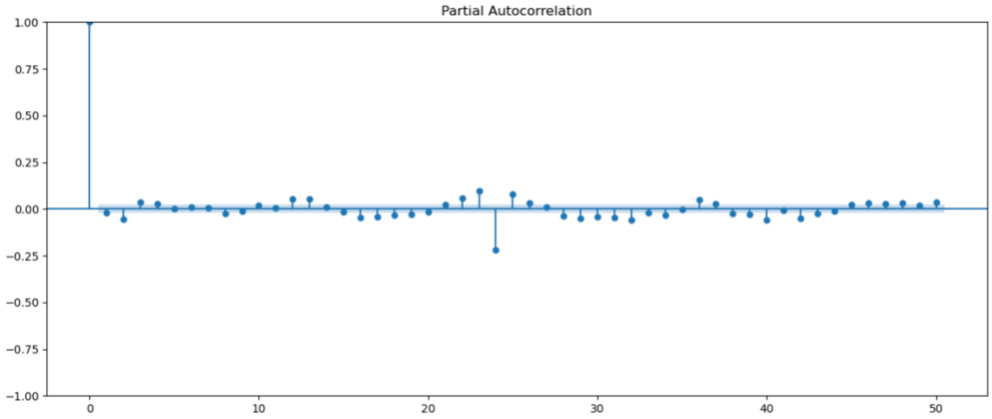
Fonte: Elaboração Própria

Figura 35 – Diagrama de Autocorrelação dos Resíduos do Modelo ARIMA – Método Automático – Aldeia de Joanes



Fonte: Elaboração Própria

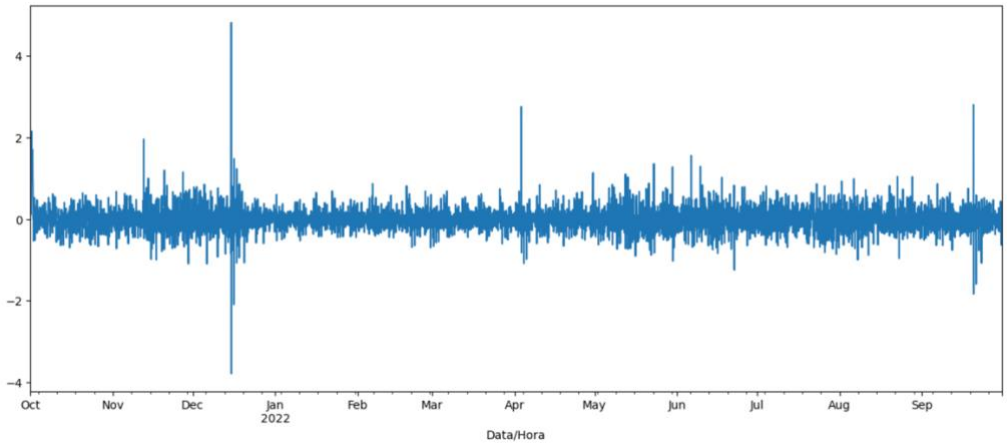
Figura 36 – Diagrama de Autocorrelação Parcial dos Resíduos do Modelo ARIMA – Método Automático – Aldeia de Joanes



Fonte: Elaboração Própria

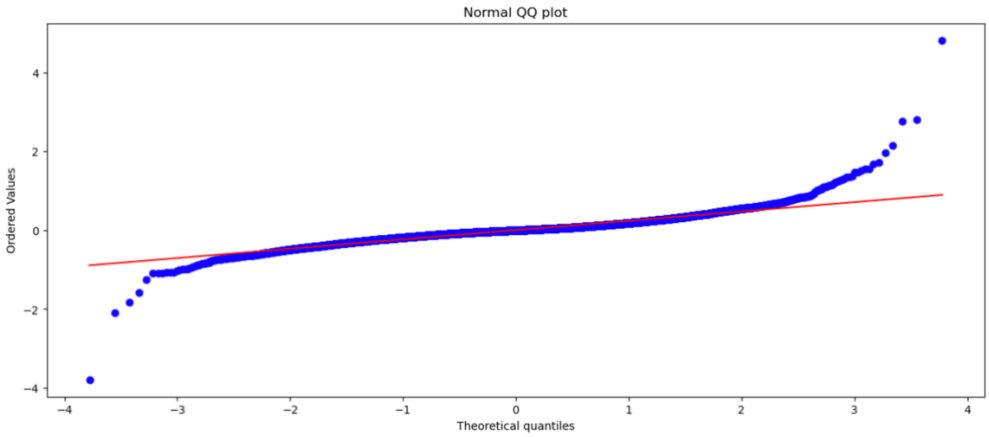
2.2.3. Série Temporal de Degolados

Figura 37 – Resíduos do Modelo ARIMA – Método Manual – Degolados



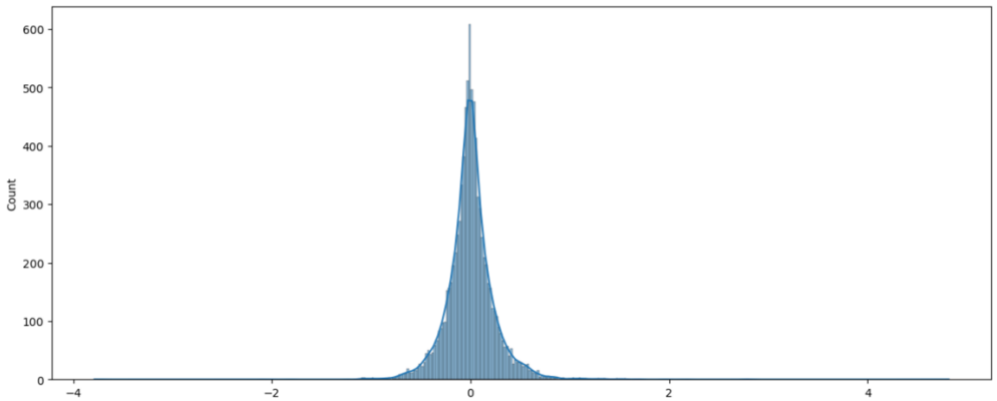
Fonte: Elaboração Própria

Figura 38 - Normal QQ Plot dos Resíduos do Modelo ARIMA – Método Manual – Degolados



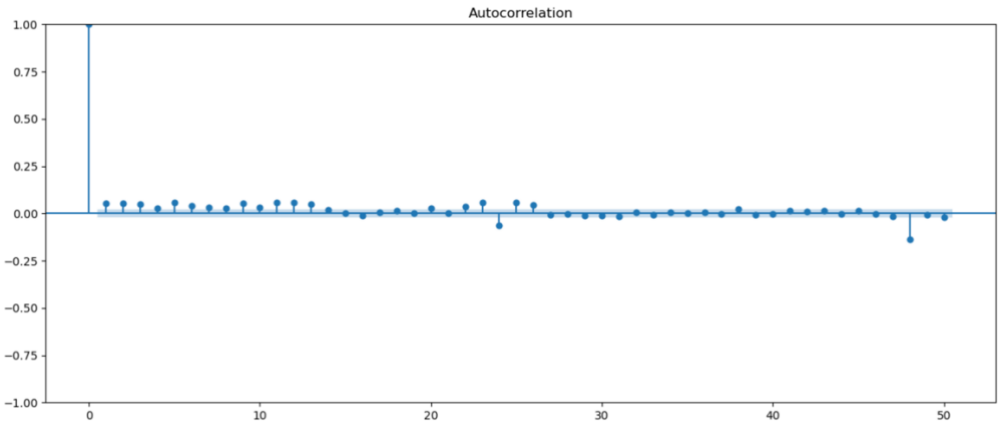
Fonte: Elaboração Própria

Figura 39 - Distribuição dos Resíduos do Modelo ARIMA – Método Manual – Degolados



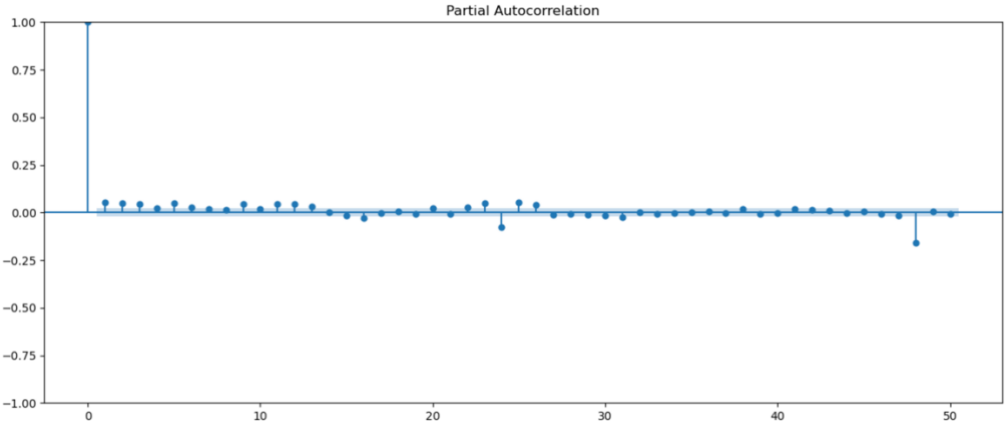
Fonte: Elaboração Própria

Figura 40 – Diagrama de Autocorrelação dos Resíduos do Modelo ARIMA – Método Manual – Degolados



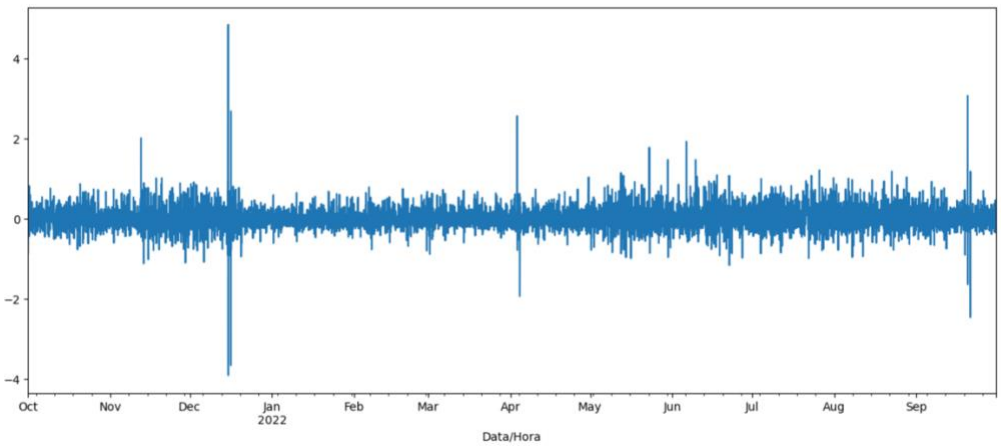
Fonte: Elaboração Própria

Figura 41 – Diagrama de Autocorrelação Parcial dos Resíduos do Modelo ARIMA – Método Manual – Degolados



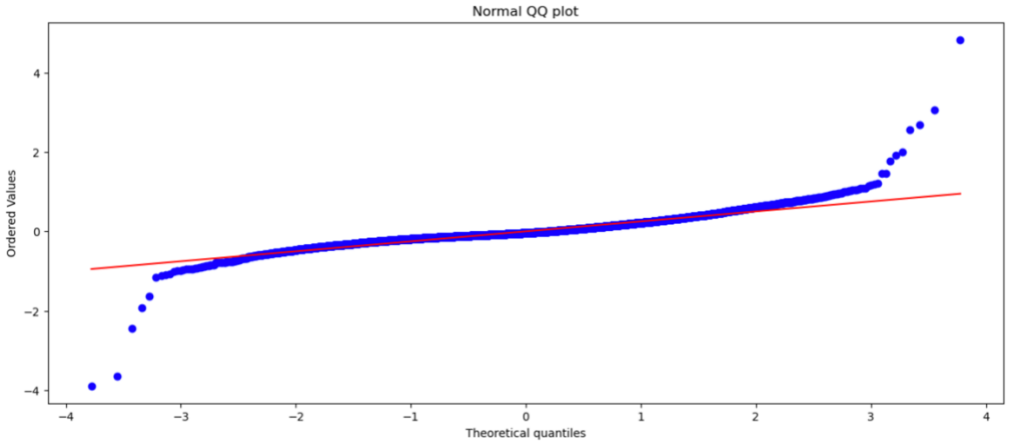
Fonte: Elaboração Própria

Figura 42 - Resíduos do Modelo ARIMA – Método Automático – Degolados



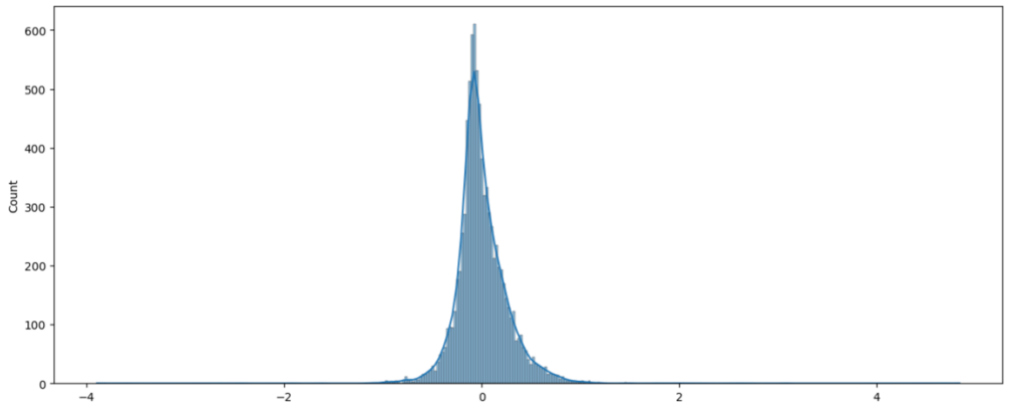
Fonte: Elaboração Própria

Figura 43 – Normal QQ Plot dos Resíduos do Modelo ARIMA – Método Automático – Degolados



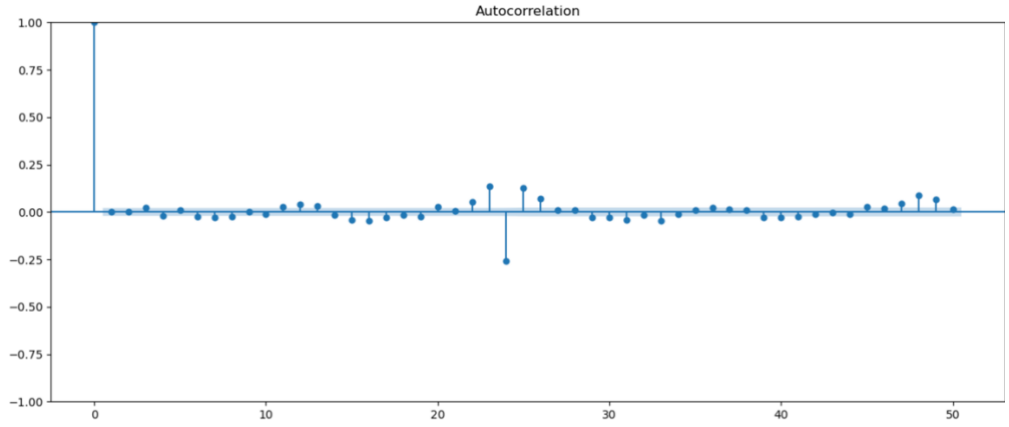
Fonte: Elaboração Própria

Figura 44 – Distribuição dos Resíduos do Modelo ARIMA – Método Automático – Degolados



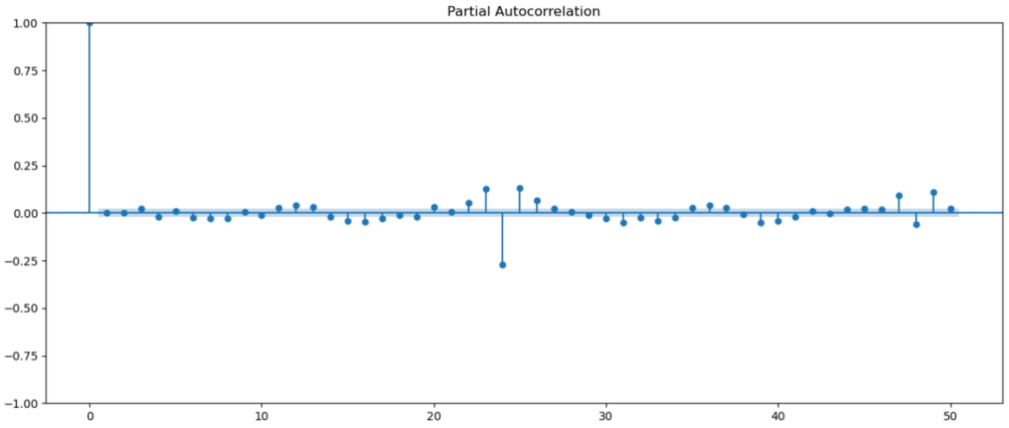
Fonte: Elaboração Própria

Figura 45 - Diagrama de Autocorrelação dos Resíduos do Modelo ARIMA – Método Automático – Degolados



Fonte: Elaboração Própria

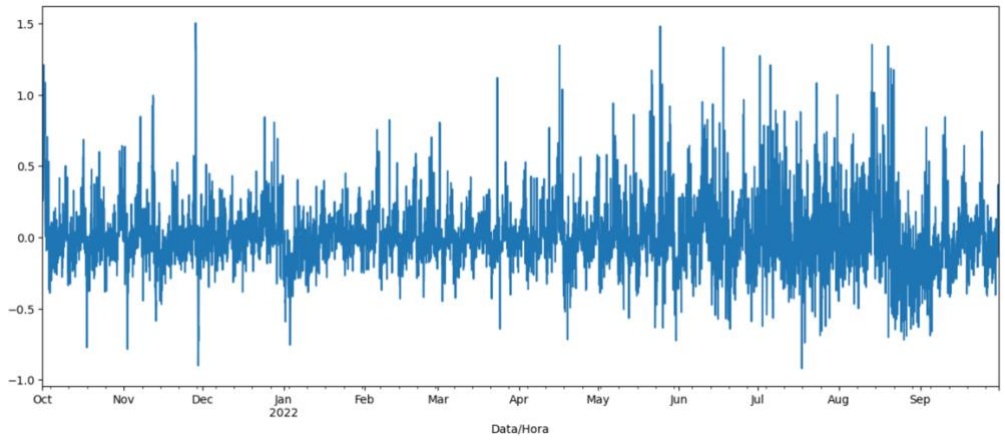
Figura 46 - Diagrama de Autocorrelação Parcial dos Resíduos do Modelo ARIMA – Método Automático – Degolados



Fonte: Elaboração Própria

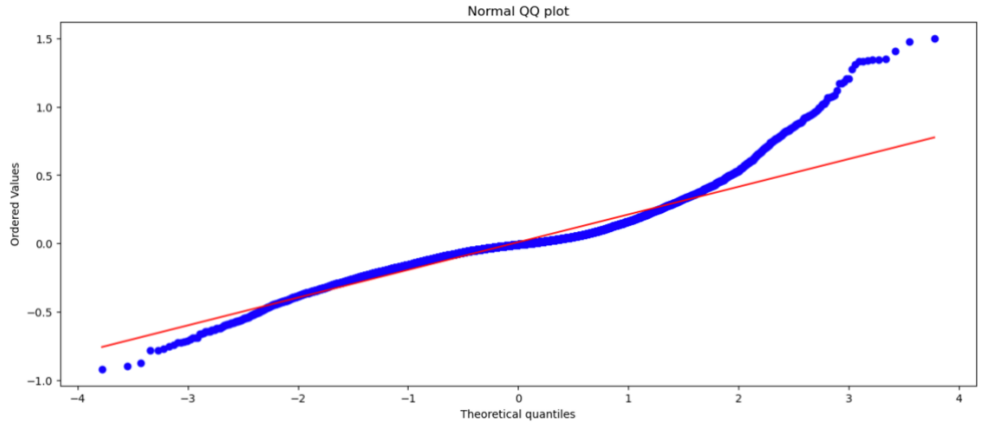
2.2.4. Série Temporal de Janeiro de Cima

Figura 47 – Resíduos do Modelo ARIMA – Método Manual – Janeiro de Cima



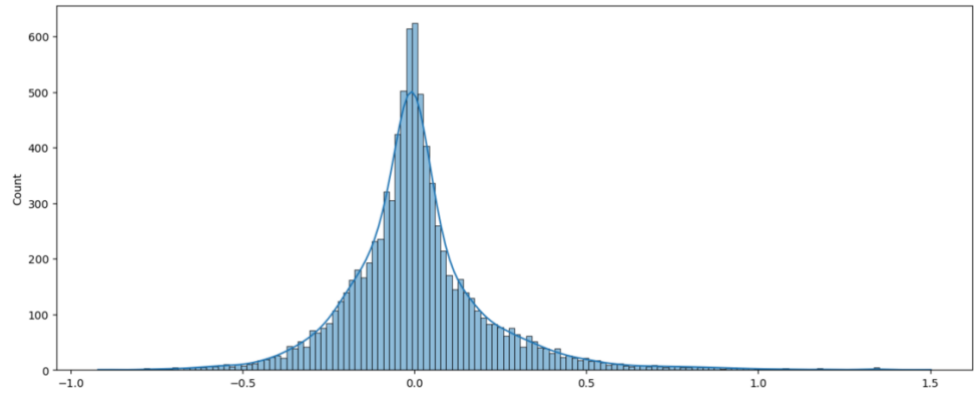
Fonte: Elaboração Própria

Figura 48 – Normal QQ Plot dos Resíduos do Modelo ARIMA – Método Manual – Janeiro de Cima



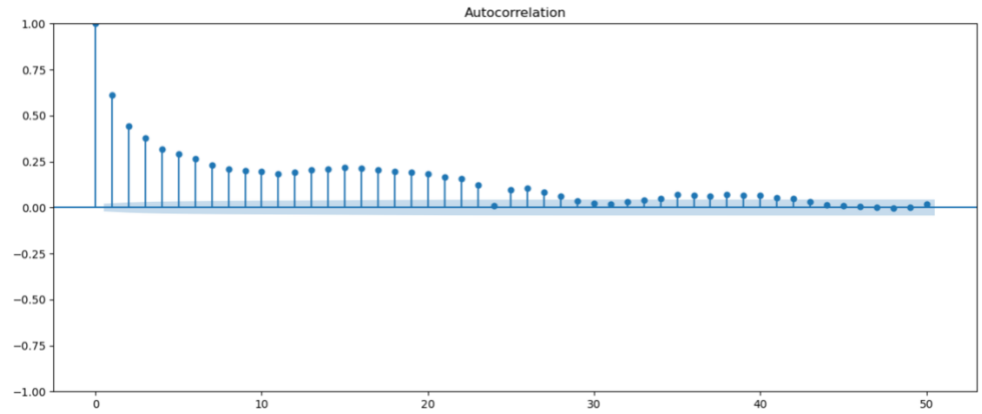
Fonte: Elaboração Própria

Figura 49 – Distribuição dos Resíduos do Modelo ARIMA – Método Manual – Janeiro de Cima



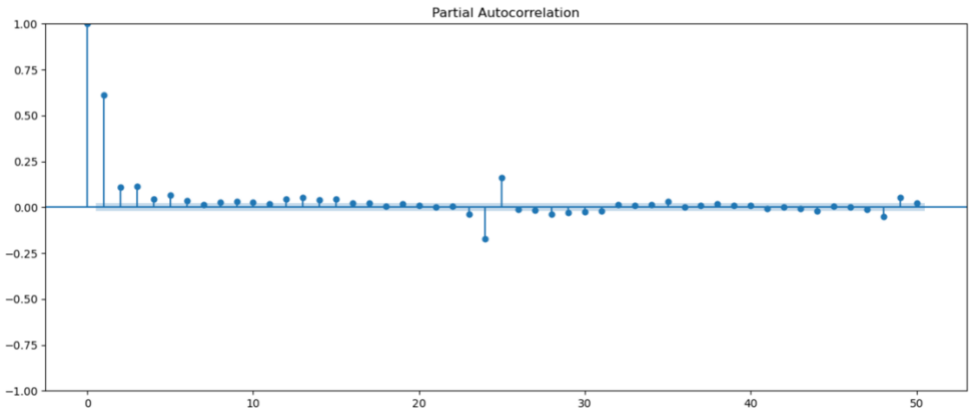
Fonte: Elaboração Própria

Figura 50 – Diagrama de Autocorrelação dos Resíduos do Modelo ARIMA – Método Manual – Janeiro de Cima



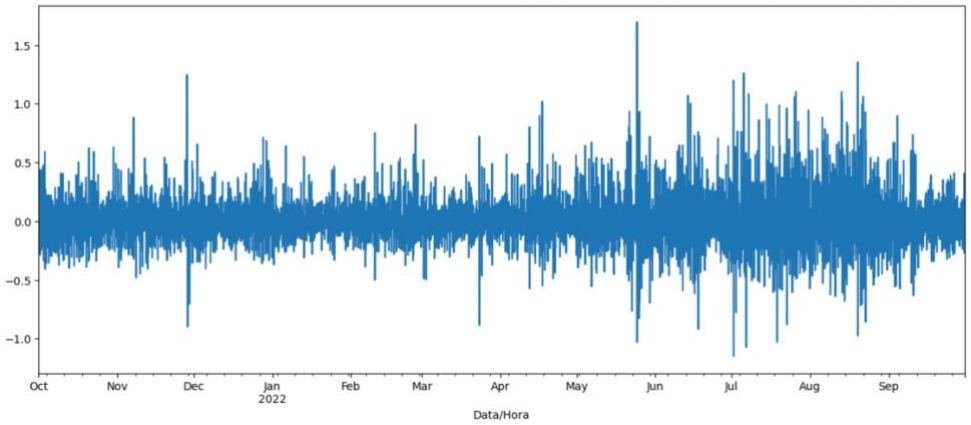
Fonte: Elaboração Própria

Figura 51 - Diagrama de Autocorrelação Parcial dos Resíduos do Modelo ARIMA – Método Manual – Janeiro de Cima



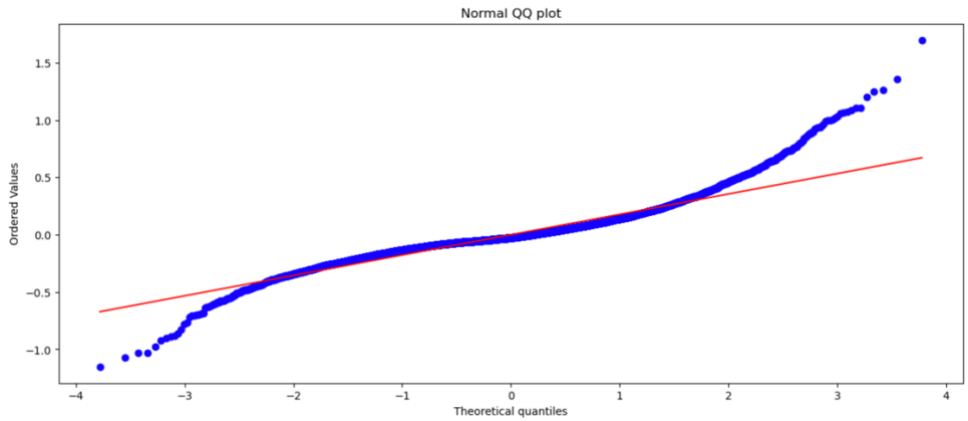
Fonte: Elaboração Própria

Figura 52 - Resíduos do Modelo ARIMA – Método Automático – Janeiro de Cima



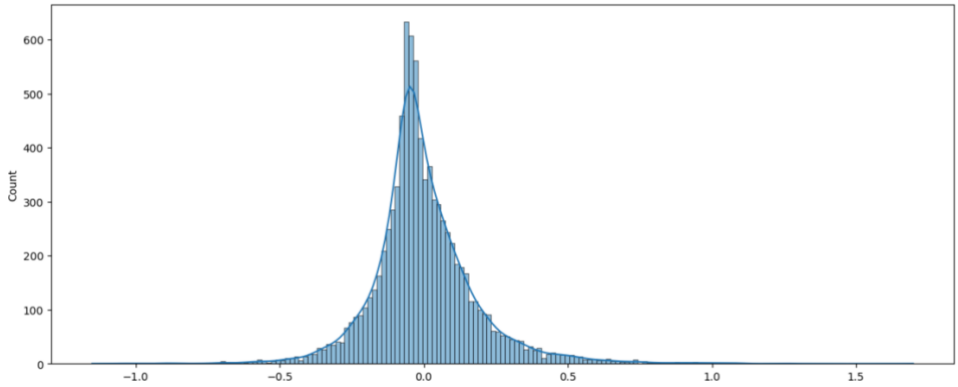
Fonte: Elaboração Própria

Figura 53 - Normal QQ Plot dos Resíduos do Modelo ARIMA – Método Automático – Janeiro de Cima



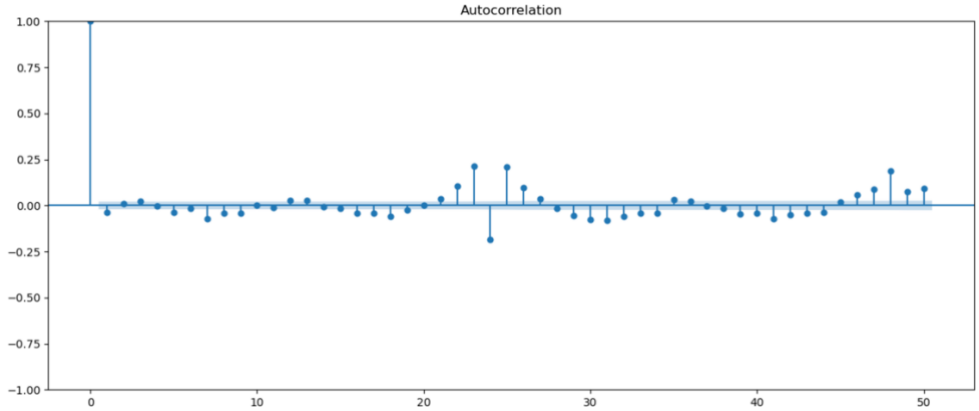
Fonte: Elaboração Própria

Figura 54 - Distribuição dos Resíduos do Modelo ARIMA – Método Automático – Janeiro de Cima



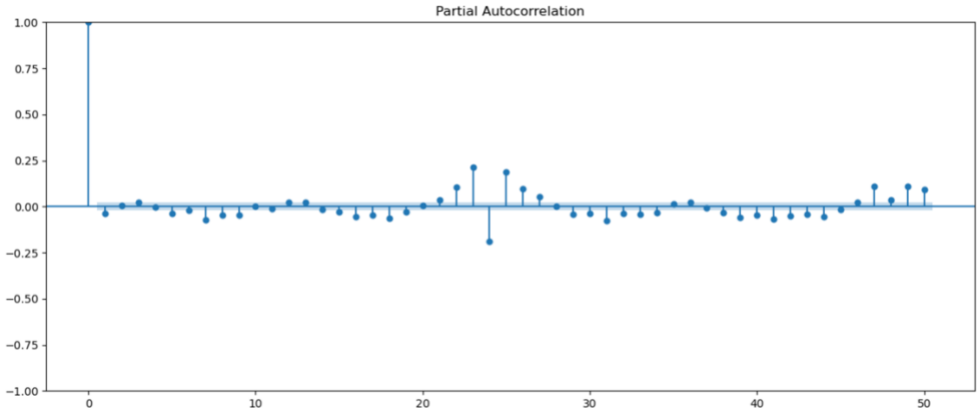
Fonte: Elaboração Própria

Figura 55 – Diagrama de Autocorrelação dos Resíduos do Modelo ARIMA – Método Automático – Janeiro de Cima



Fonte: Elaboração Própria

Figura 56 - Diagrama de Autocorrelação Parcial dos Resíduos do Modelo ARIMA – Método Automático – Janeiro de Cima



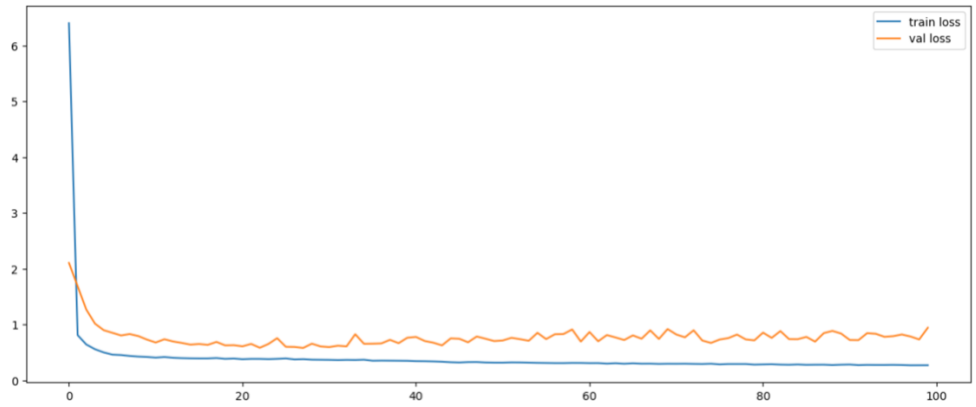
Fonte: Elaboração Própria

2.3. Figuras dos Gráficos Obtidos por Aplicação do Modelo LSTM

De seguida apresentam-se todas as figuras com os gráficos obtidos, durante a aplicação do modelo LSTM, não apresentadas no texto principal.

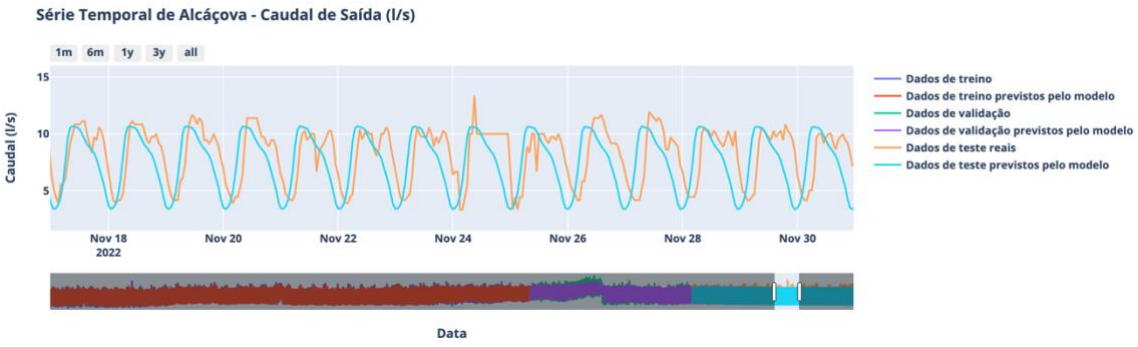
2.3.1. Série Temporal de Alcáçova

Figura 57 – Função de Custo Treino/Validação – Modelo LSTM Univariável com Dados Originais - Alcáçova



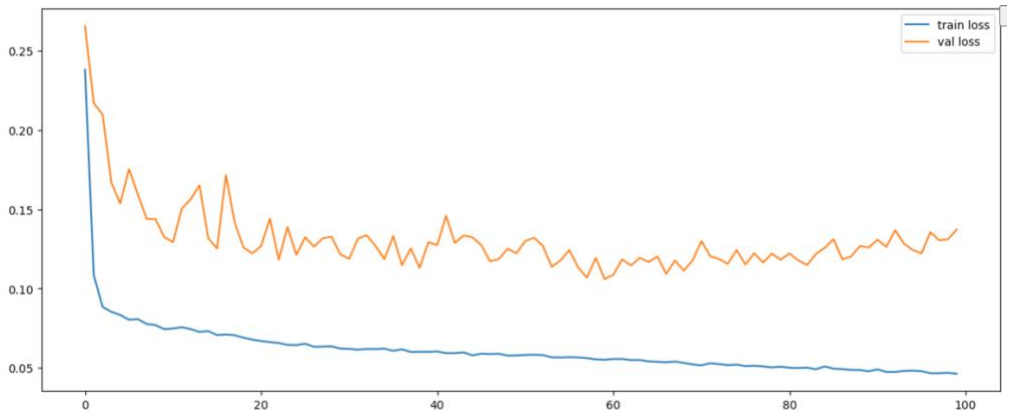
Fonte: Elaboração Própria

Figura 58 – Ampliação das Previsões do Modelo LSTM Univariável com Dados Originais para Alcáçova



Fonte: Elaboração Própria

Figura 59 – Função de Custo Treino/Validação – Modelo LSTM Univariável com Dados Padronizados - Alcáçova

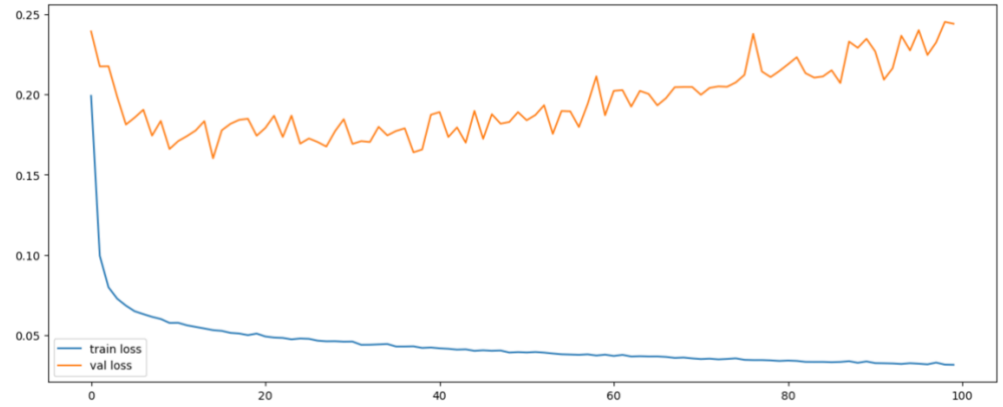


Fonte: Elaboração Própria

Figura 60 – Previsões do Modelo LSTM Univariável com Dados Padronizados para Alcáçova

Fonte: Elaboração Própria

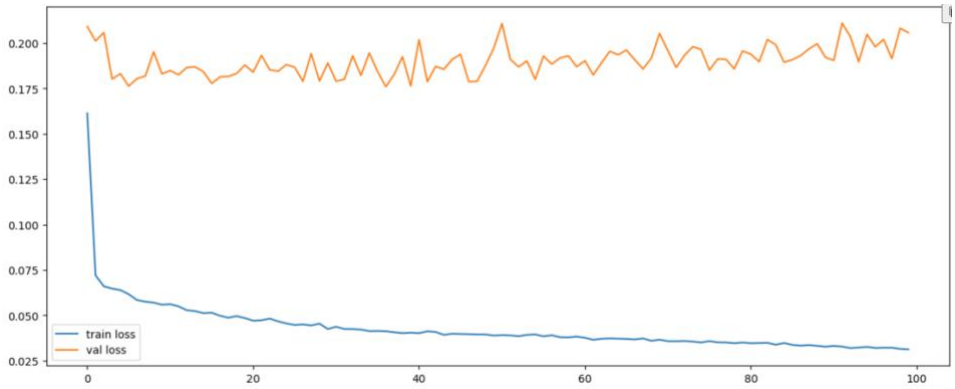
Figura 61 – Função de Custo Treino/Validação – Modelo LSTM Multivariável com Dados Padronizados - Alcáçova



Fonte: Elaboração Própria

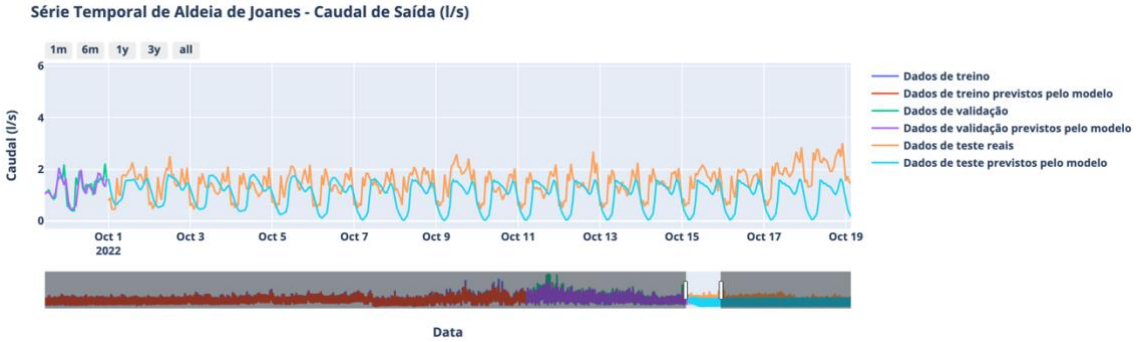
2.3.2. Série Temporal de Aldeia de Joanes

Figura 62 – Função de Custo Treino/Validação – Modelo LSTM Univariável com Dados Originais – Aldeia de Joanes



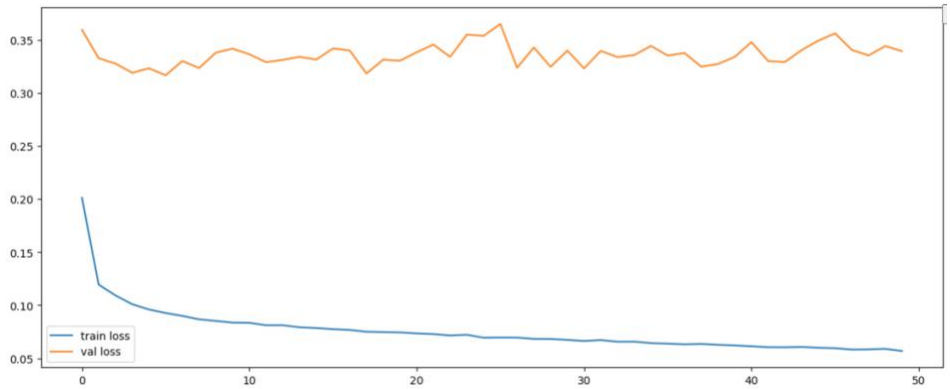
Fonte: Elaboração Própria

Figura 63 – Ampliação das Previsões do Modelo LSTM Univariável com Dados Originais para Aldeia de Joanes



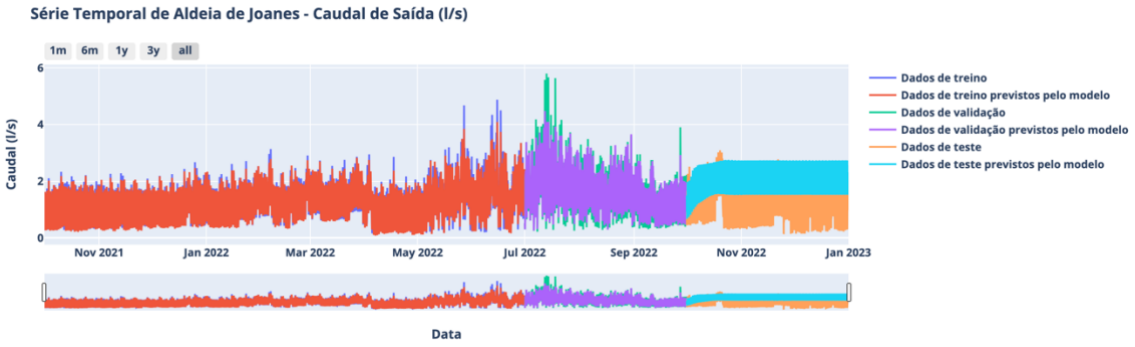
Fonte: Elaboração Própria

Figura 64 – Função de Custo Treino/Validação – Modelo LSTM Univariável com Dados Padronizados – Aldeia de Joanes



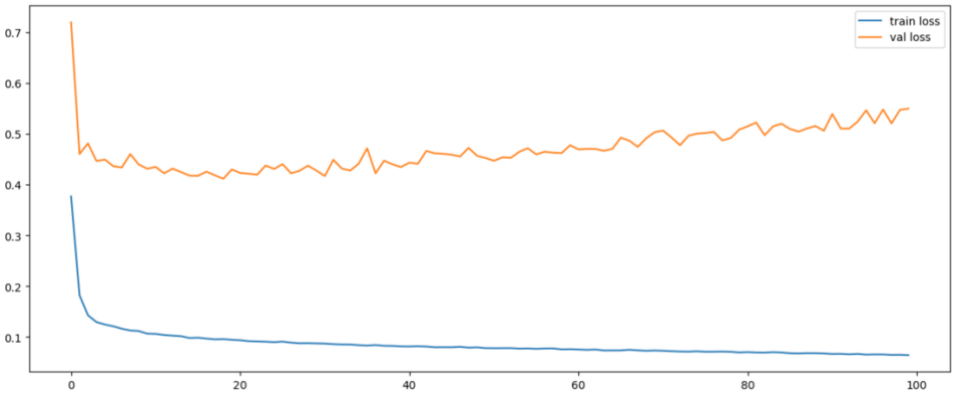
Fonte: Elaboração Própria

Figura 65 – Previsões do Modelo LSTM Univariável com Dados Padronizados para Aldeia de Joanes



Fonte: Elaboração Própria

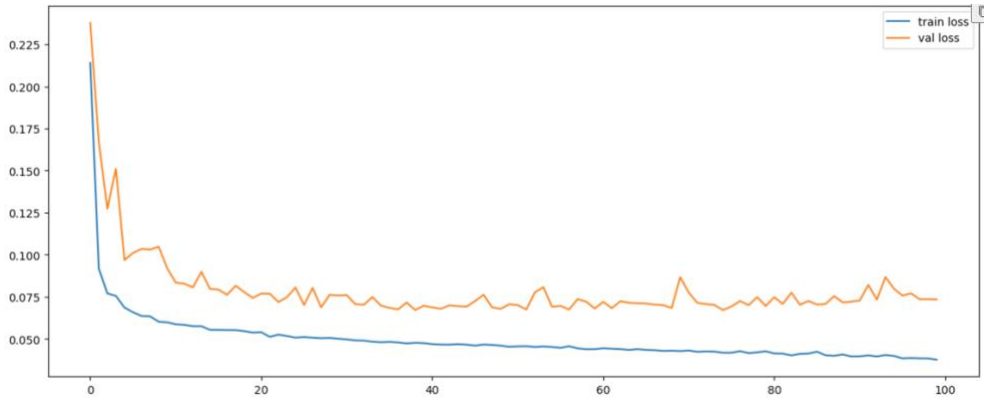
Figura 66 – Função de Custo Treino/Validação – Modelo LSTM Multivariável com Dados Padronizados – Aldeia de Joanes



Fonte: Elaboração Própria

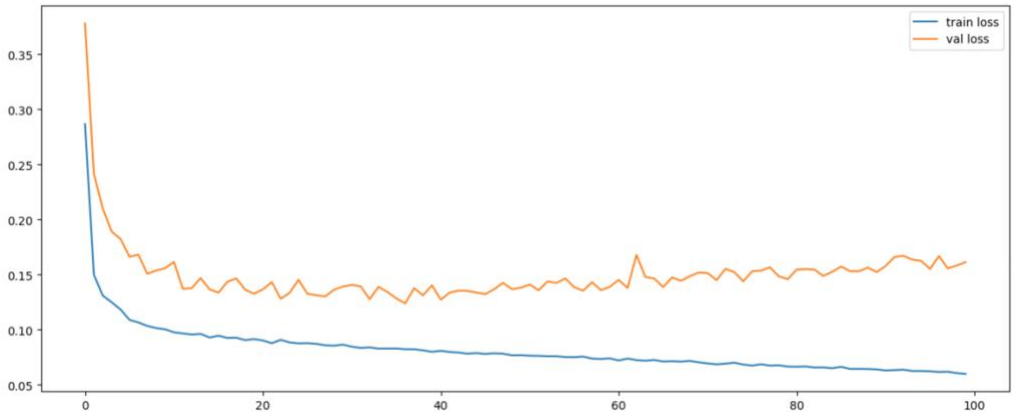
2.3.3. Série Temporal de Degolados

Figura 67 – Função de Custo Treino/Validação – Modelo LSTM Univariável com Dados Originais – Degolados



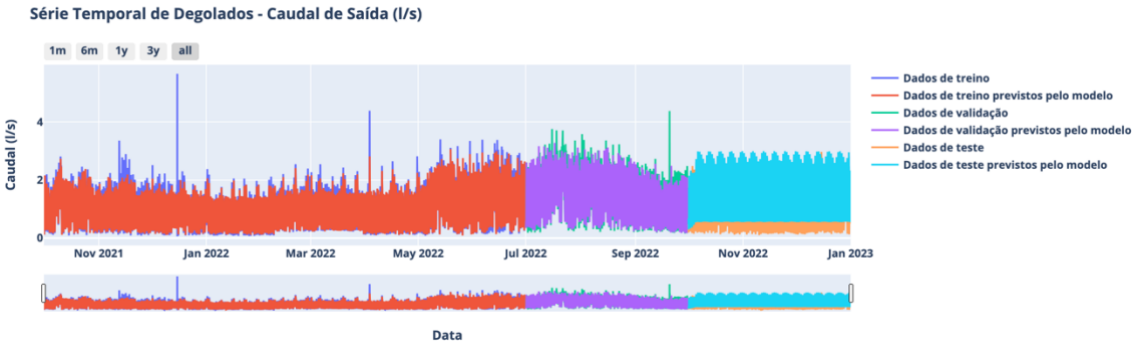
Fonte: Elaboração Própria

Figura 68 - Função de Custo Treino/Validação – Modelo LSTM Univariável com Dados Padronizados – Degolados



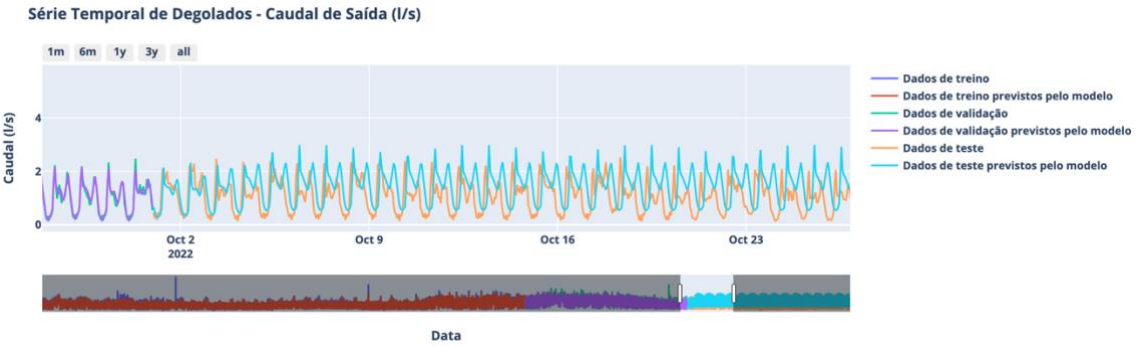
Fonte: Elaboração Própria

Figura 69 – Previsões do Modelo LSTM Univariável com Dados Padronizados para Degolados



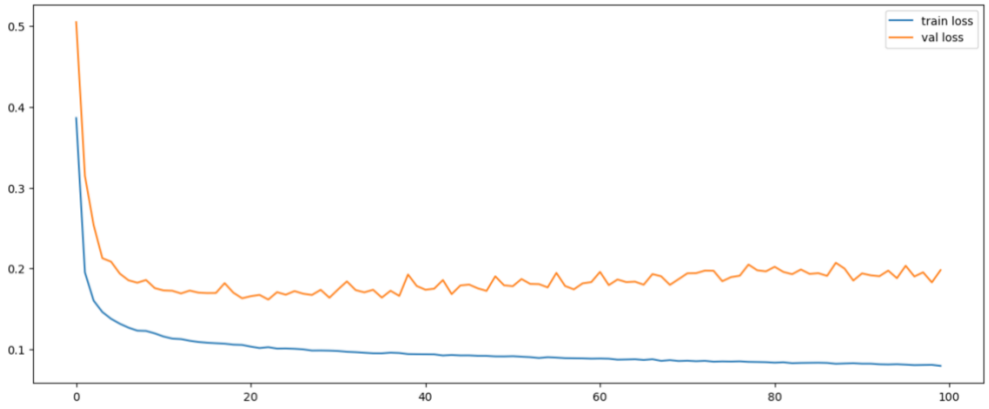
Fonte: Elaboração Própria

Figura 70 – Ampliação das Previsões do Modelo LSTM Univariável com Dados Padronizados para Degolados



Fonte: Elaboração Própria

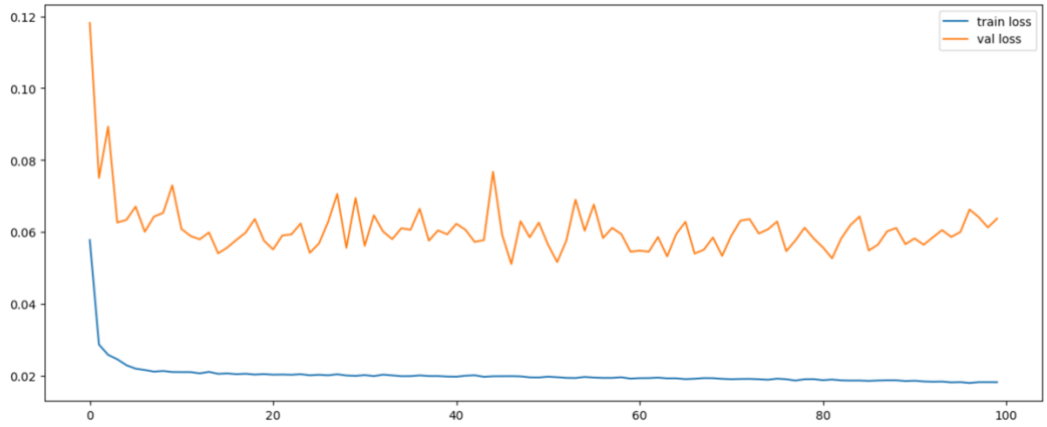
Figura 71 - Função de Custo Treino/Validação – Modelo LSTM Multivariável com Dados Padronizados – Degolados



Fonte: Elaboração Própria

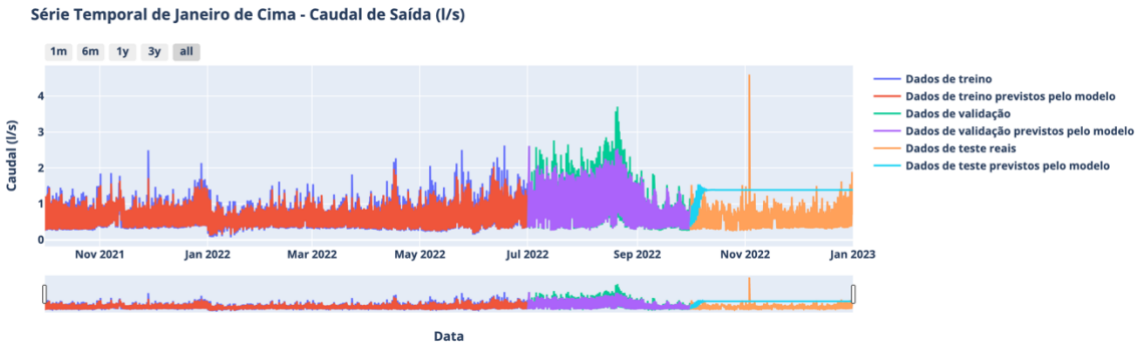
2.3.4. Série Temporal de Janeiro de Cima

Figura 72 – Função de Custo Treino/Validação – Modelo LSTM Univariável com Dados Originais – Janeiro de Cima



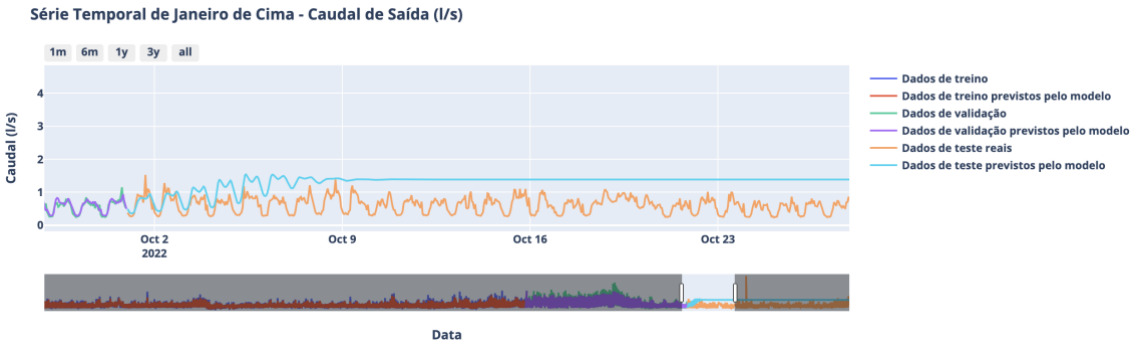
Fonte: Elaboração Própria

Figura 73 – Previsões do Modelo LSTM Univariável com Dados Originais – Janeiro de Cima



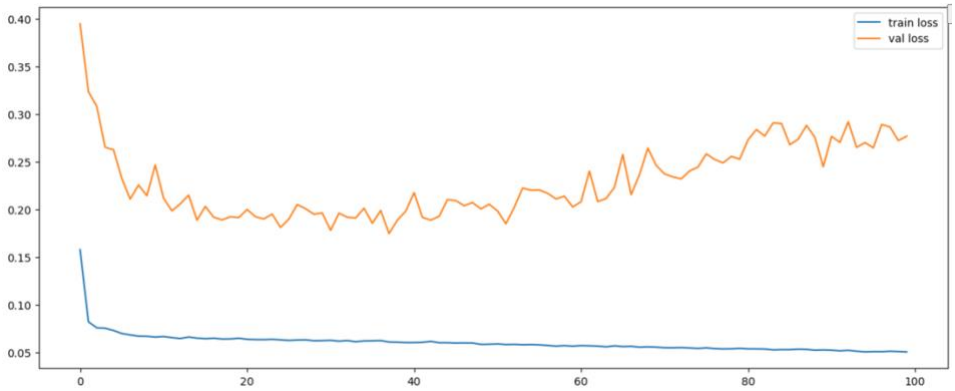
Fonte: Elaboração Própria

Figura 74 – Ampliação das Previsões do Modelo LSTM Univariável com Dados Originais – Janeiro de Cima



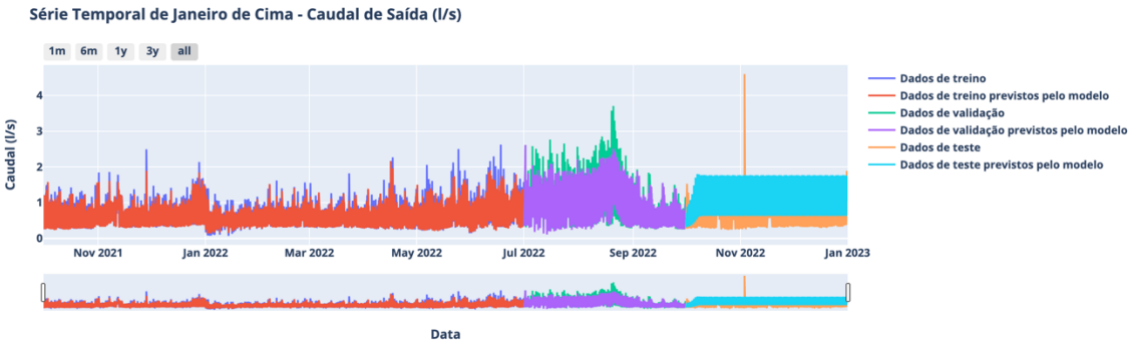
Fonte: Elaboração Própria

Figura 75 - Função de Custo Treino/Validação – Modelo LSTM Univariável com Dados Padronizados – Janeiro de Cima



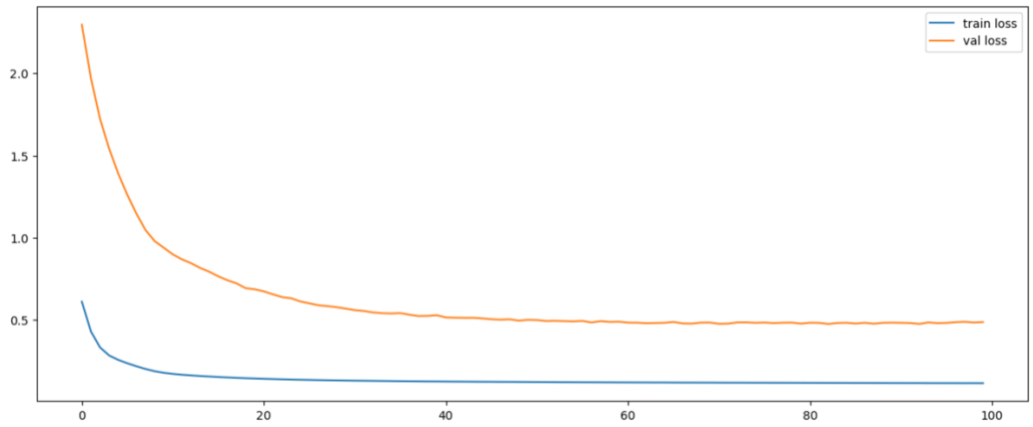
Fonte: Elaboração Própria

Figura 76 - Previsões do Modelo LSTM Univariável com Dados Padronizados – Janeiro de Cima



Fonte: Elaboração Própria

Figura 77 - Função de Custo Treino/Validação – Modelo LSTM Multivariável com Dados Padronizados – Janeiro de Cima



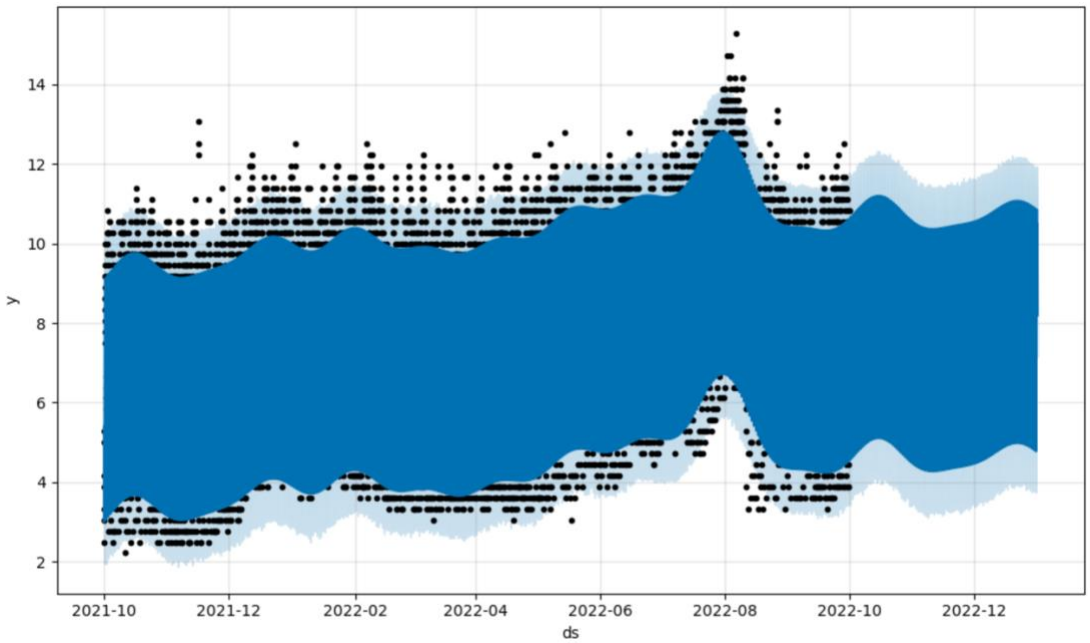
Fonte: Elaboração Própria

2.4. Figuras dos Gráficos Obtidos por Aplicação do Modelo *Prophet*

De seguida apresentam-se todas as figuras com os gráficos obtidos, durante a aplicação do modelo *Prophet*, não apresentadas no texto principal.

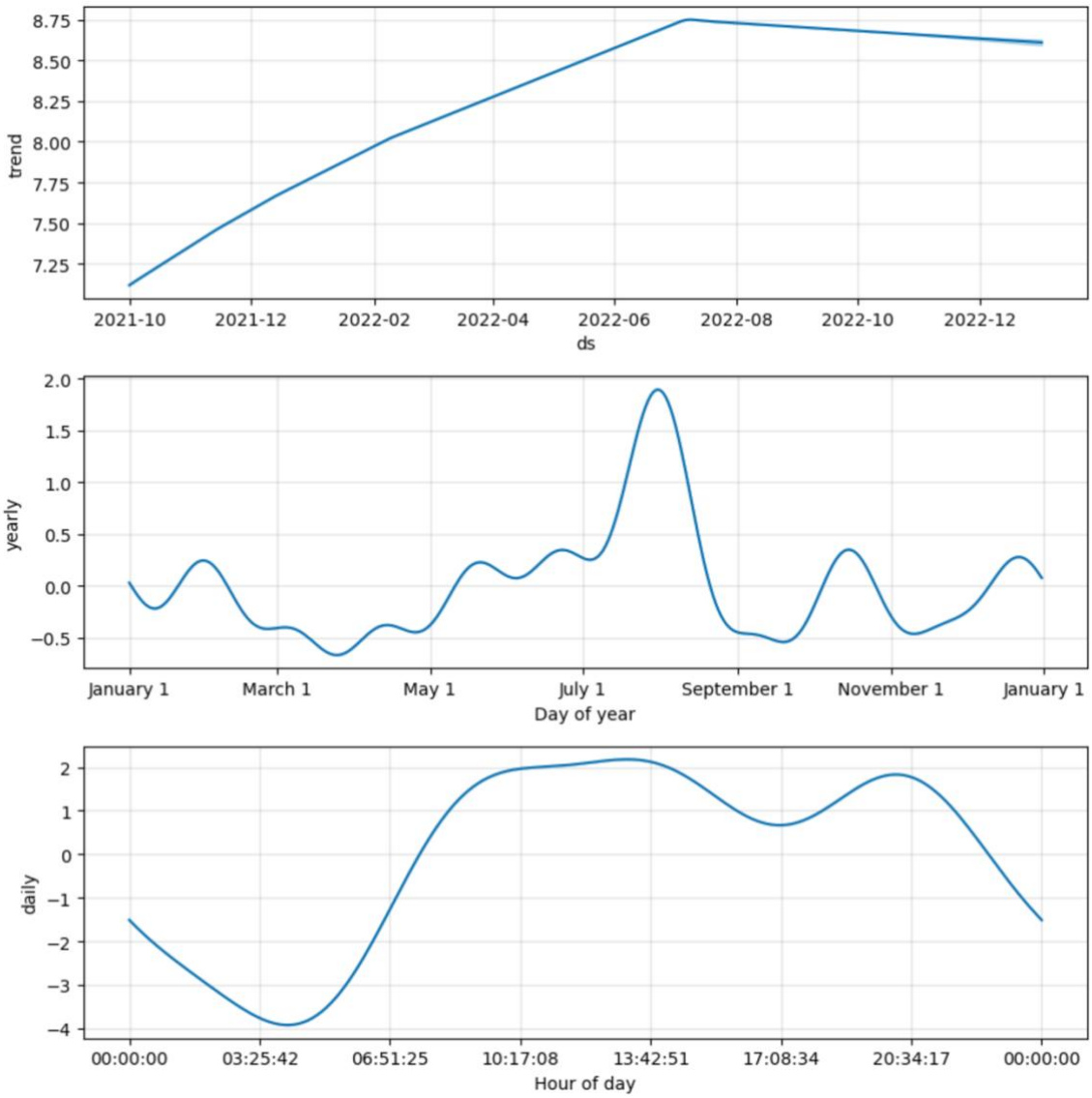
2.4.1. Série Temporal de Alcáçova

Figura 78 – Previsões do Modelo Prophet com Intervalos de Incerteza – Alcáçova



Fonte: Elaboração Própria

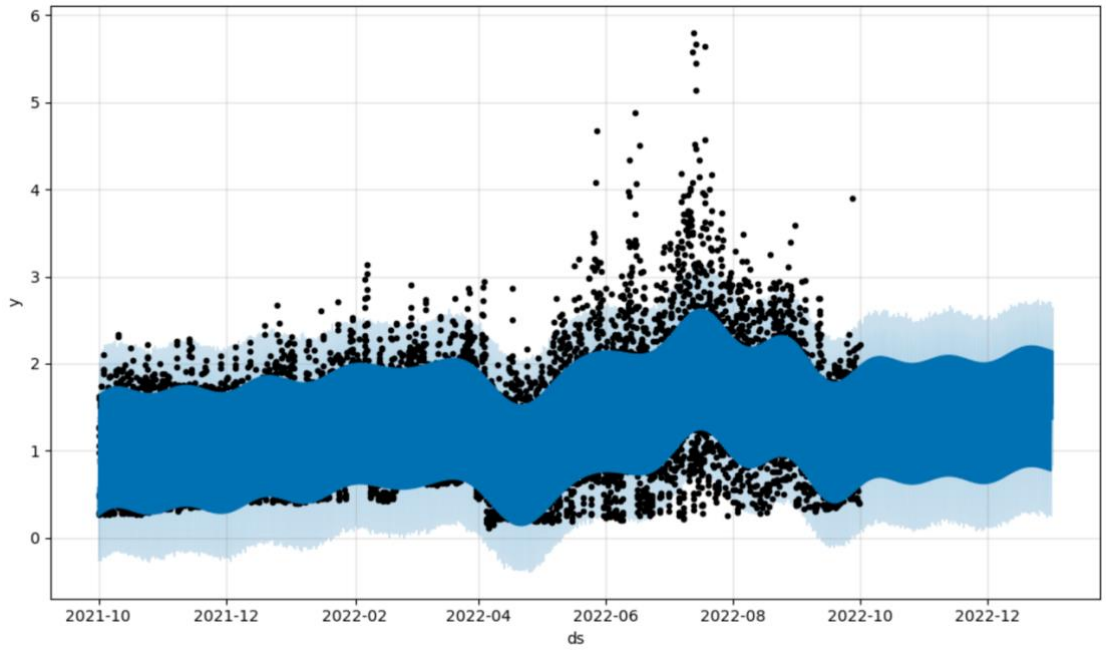
Figura 79 – Componentes do Modelo Prophet – Alcáçova



Fonte: Elaboração Própria

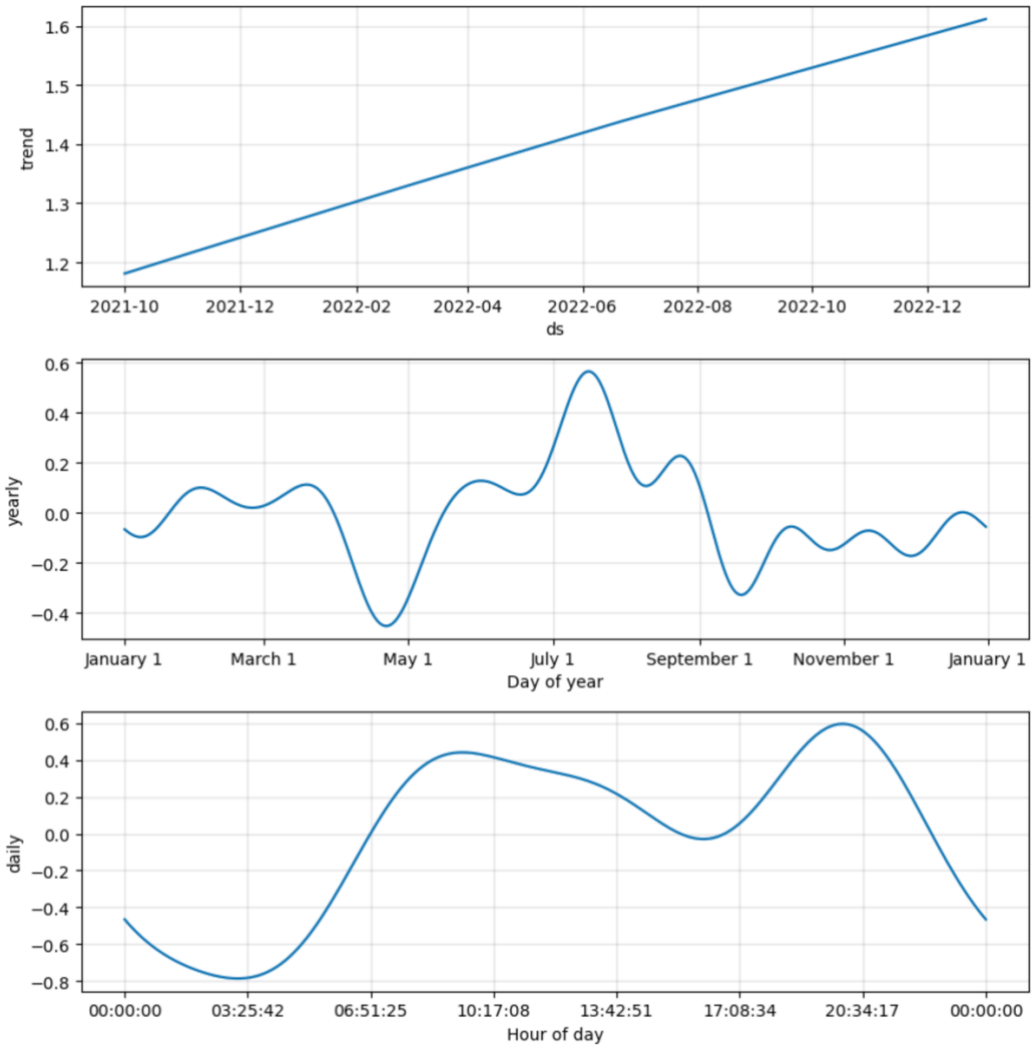
2.4.2. Série Temporal de Aldeia de Joanes

Figura 80 – Previsões do Modelo Prophet com Intervalos de Incerteza – Aldeia de Joanes



Fonte: Elaboração Própria

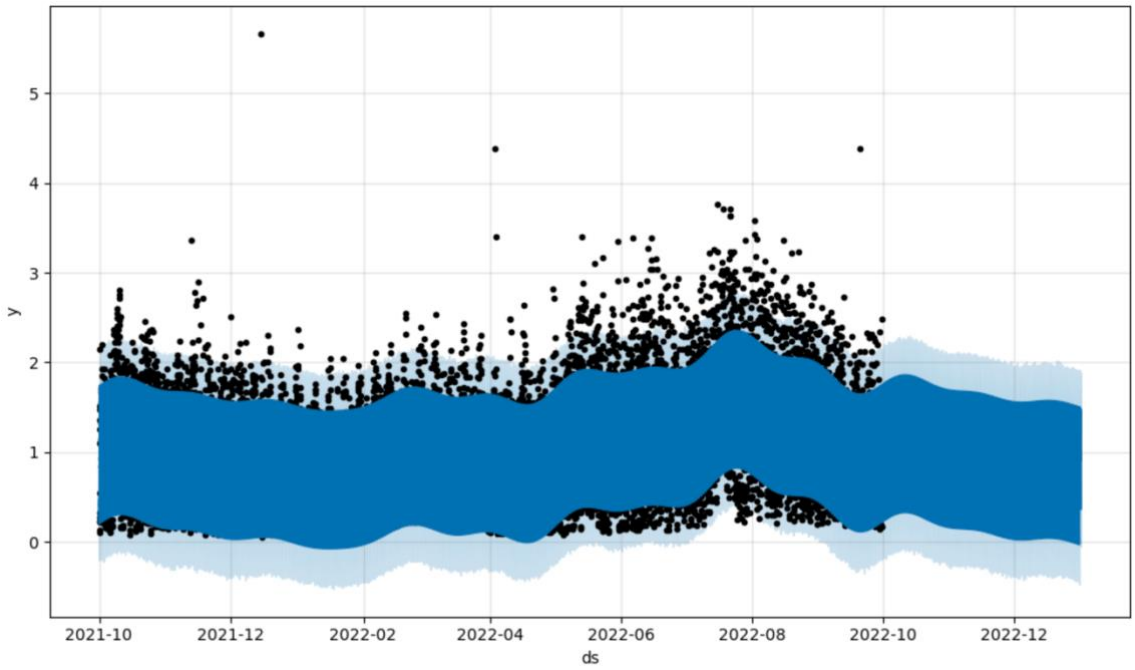
Figura 81 - Componentes do Modelo Prophet – Aldeia de Joanes



Fonte: Elaboração Própria

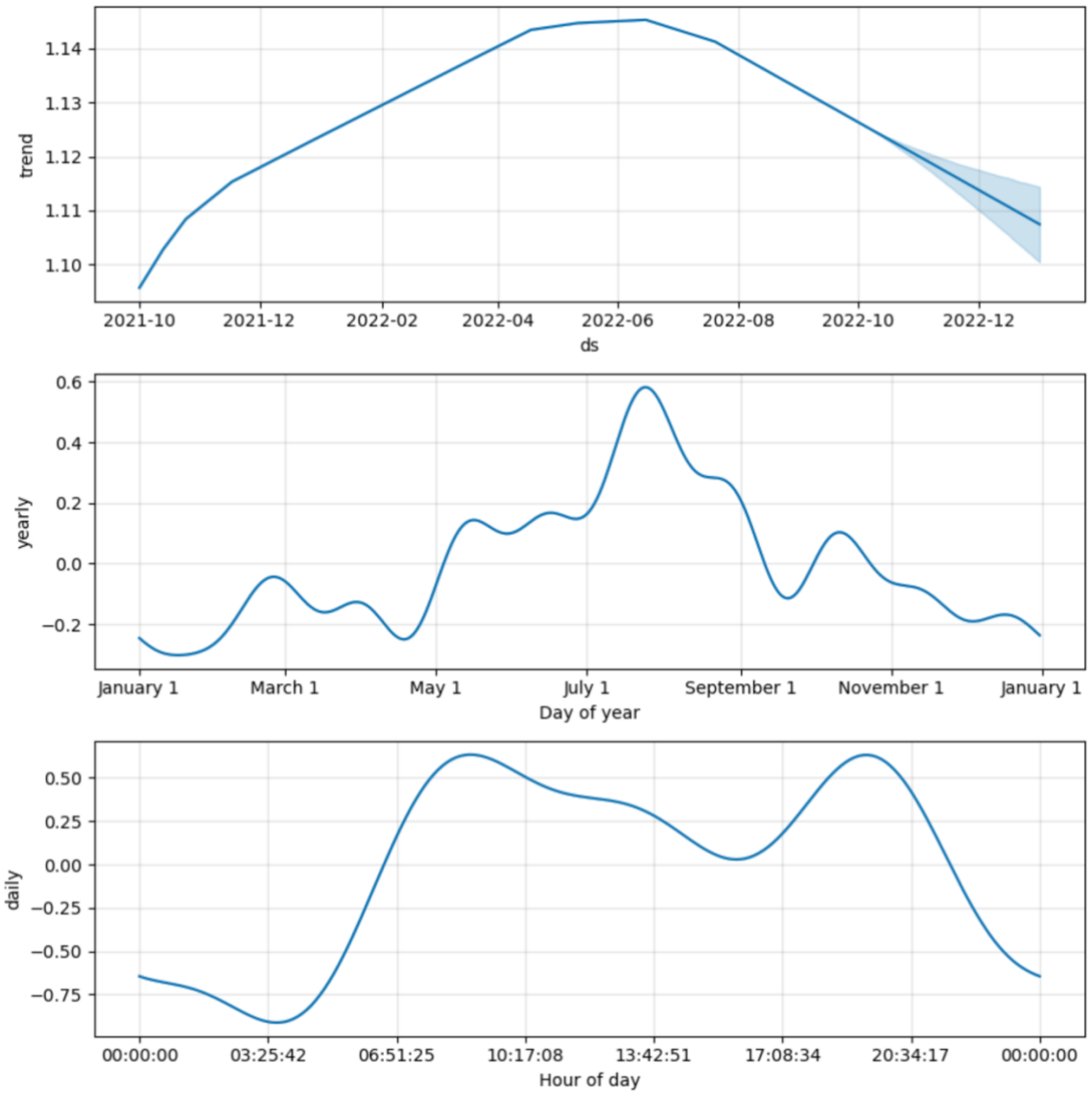
2.4.3. Série Temporal de Degolados

Figura 82 – Previsões do Modelo Prophet com Intervalos de Incerteza – Degolados



Fonte: Elaboração Própria

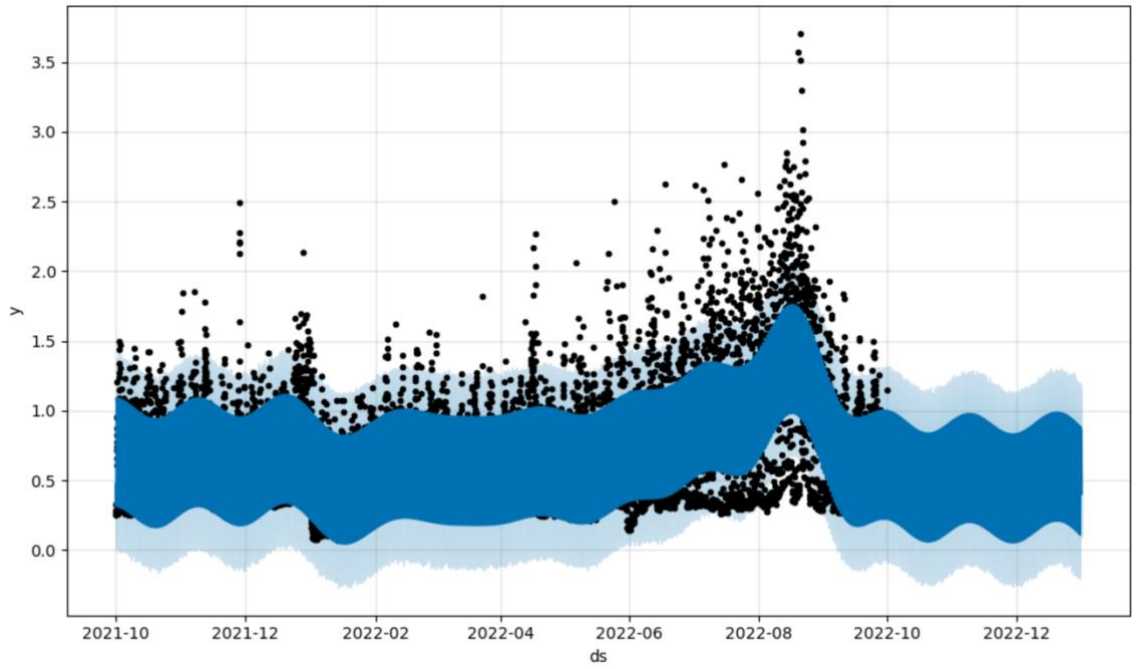
Figura 83 - Componentes do Modelo Prophet – Degolados



Fonte: Elaboração Própria

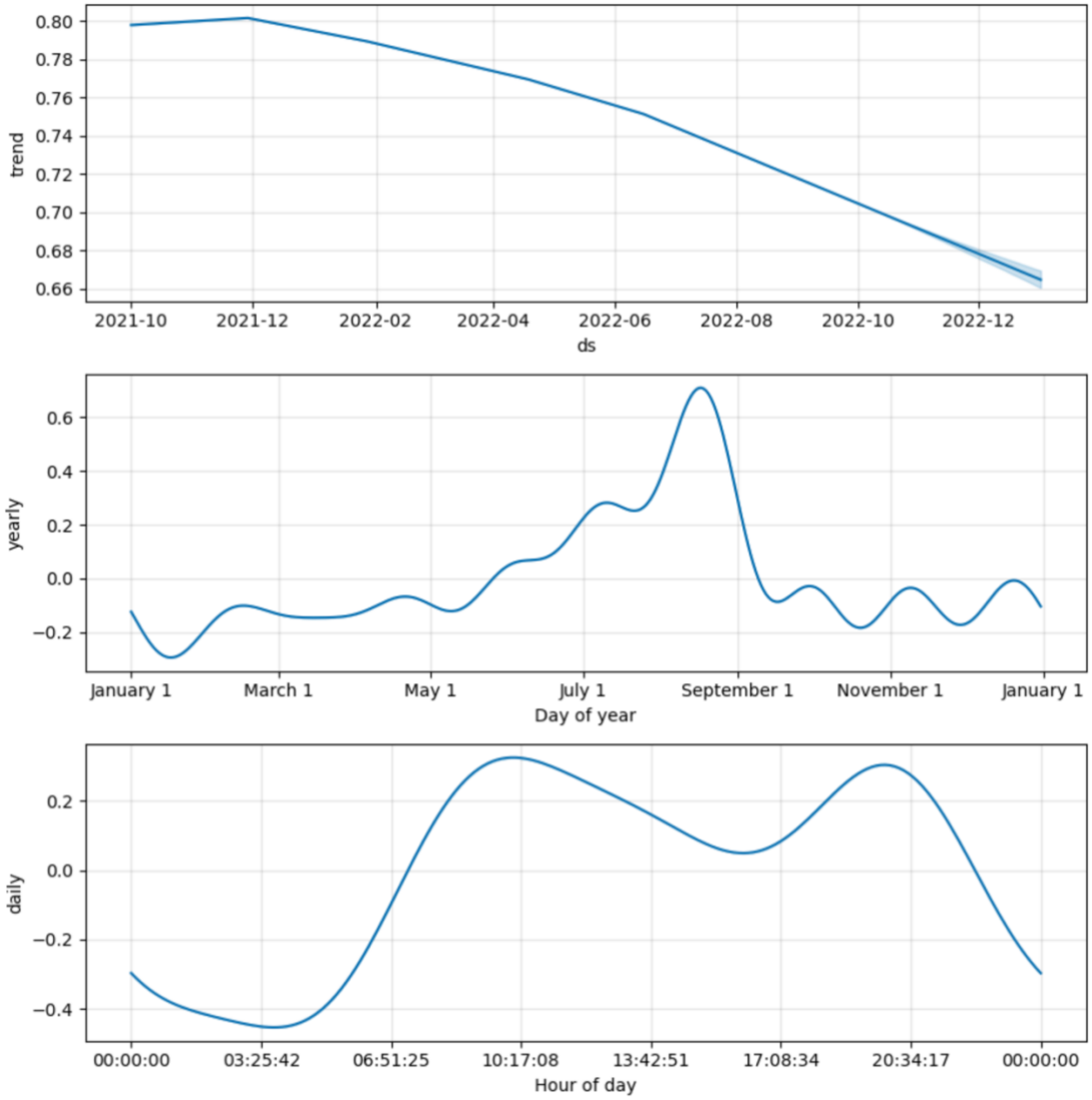
2.4.4. Série Temporal de Janeiro de Cima

Figura 84 – Previsões do Modelo Prophet com Intervalos de Incerteza – Janeiro de Cima



Fonte: Elaboração Própria

Figura 85 – Componentes do Modelo Prophet – Janeiro de Cima



Fonte: Elaboração Própria