INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Previsão de Séries Temporais Financeiras

MIGUEL GOMES PRETO
Junho de 2024

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Previsão de Séries Temporais Financeiras

Miguel Gomes Preto

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de Software

Orientador: Maria de Fátima Coutinho Rodrigues

Porto, Junho 2024

Declaração de Integridade

Declaro ter conduzido este trabalho académico com integridade.

Não plagiei ou apliquei qualquer forma de uso indevido de informações ou falsificação de resultados ao longo do processo que levou à sua elaboração.

Portanto, o trabalho apresentado neste documento é original e de minha autoria, não tendo sido utilizado anteriormente para nenhum outro fim.

Declaro ainda que tenho pleno conhecimento do Código de Conduta Ética do P.PORTO.

ISEP, Porto, 30 de junho de 2024

Dedicatória

Esta dissertação é dedicada ao Eu do futuro.

Que isto te tenha servido de exemplo.

Resumo

Num mundo tecnológico cada vez mais avançado a inflação é um problema sério que afeta todos. Existem algumas soluções para contornar este problema, sendo uma delas o investimento em mercados de ações. O mercado de ações permite a empresas angariarem capital e fornece aos investidores a possibilidade de gerar lucros. Estes mercados tendem a ser algo complexo de analisar e prever, pelo que, têm sido alvo de vários estudos.

Nesta dissertação foram analisados vários métodos de previsão de séries temporais financeiras, desde métodos estatísticos a métodos de aprendizagem automática. Foram analisados e implementados modelos da família ARMA e GARCH, assim como modelos de aprendizagem automática e modelos híbridos, para diferentes valores temporais de previsão.

Estes modelos foram avaliados com quatro séries temporais distintas, CPIAUCSL, G17MVSFAUTOS, SHEL e LYG.

Para CPIAUCSL, para um horizonte de 3 e 10 ARIMA é o melhor modelo, mas para um horizonte de 20 ML (SVR) é a melhor solução. Para a série temporal G17MVSFAUTOS o modelo com maior desempenho para os horizontes de 3 e 10 foi ML (SVR), enquanto que para o horizonte de 20 foi ML (KNR). Para a SHEL com um horizonte de 3 dias, o modelo ML (SVR) com uma janela de 10 apresenta desempenho superior, mas para os horizontes 10 e 20 o modelo ML (SVR) com uma janela de 5 apresenta melhores resultados. Para a série temporal LYG com um horizonte de 3, o modelo ML (SVR) com uma janela de 5 apresenta bons resultados, para o horizonte de 10, ML (RFR) de janela 5 foi o melhor e para o horizonte de 20, os resultados são mais ambíguos, sendo que ML (KNR) de janela 5, ML (RFR) de janela 5 e ML (SVR) de janela 10 apresentam bons resultados.

Palavras-chave: Aprendizagem Automática, Séries Temporais Financeiras, Previsão

Abstract

In an increasingly advanced technological world, inflation is a serious problem that affects everyone. There are some solutions to overcome this problem, one of which is investing in stock markets. The stock market allows companies to raise capital and provides investors with the possibility of generating profits. These markets tend to be complex to analyze and predict, which is why they have been the subject of several studies.

In this dissertation, various financial time series forecasting methods were analyzed, from statistical methods to machine learning methods. Models from the ARMA and GARCH family were analyzed and implemented, as well as machine learning models and hybrid models, for different temporal forecast values.

These models were evaluated with four different time series, CPIAUCSL, G17MVSFAUTOS, SHEL and LYG.

For CPIAUCSL, for a horizon of 3 and 10 ARIMA is the best model, but for a horizon of 20, ML (SVR) is the best solution. For the G17MVSFAUTOS time series, the model with the highest performance for horizons 3 and 10 was ML (SVR), while for horizon 20 it was ML (KNR). For SHEL with a horizon of 3 days, the ML model (SVR) with a window of 10 presents superior performance, but for horizons 10 and 20 the ML model (SVR) with a window of 5 presents better results. For the LYG time series with a horizon of 3, the ML (SVR) model with a window of 5 shows good results, for the horizon of 10, ML (RFR) with window 5 was the best and for the horizon of 20, the results are more ambiguous, with ML (KNR) with window 5, ML (RFR) with window 5 and ML (SVR) with window 10 showing good results.

Keywords: Machine Learning, Financial Time Series, Forecasting

Agradecimentos

Para começa, um agradecimento à minha orientadora, Professora Fátima Rodrigues, pela ajuda prestada durante este trabalho.

Gostaria de agradecer aos meus pais por estarem sempre presentes e disponíveis quando preciso, e não preciso, de ajuda, suportando-me sempre que necessário. Um agradecimento também à minha irmã, por aleatoriamente me trazer comida.

Gostava também de deixar um agradecimento aos meus amigos, por me ajudarem a manter o resto da pouca sanidade mental que ainda tenho. Um agradecimento especial a Gonçalo Soares e Hugo Esteves que me acompanharam na maior parte desta Odisseia que foi o Mestrado de Engenharia de Software e por viverem comigo os bons e maus dias.

Índice

Lista de Figuras

Lista de Equações

Lista de Excertos de Código

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

AA	Aprendizagem Automática
ACF	*Autocorrelation Function*
ADF	*Augmented Dickey-Fuller*
AI	*Artificial Intelligence*
AIC	*Akaike Information Criteria*
ANN	*Artificial Neural Networks*
AP	Aprendizagem Profunda
AR	*Auto-Regressive*
ARCH	*Auto-Regressive Conditional Heteroskedasticity*
ARIMA	*Auto-Regressive Integrated Moving Average*
ARMA	*Auto-Regressive Moving Average*
AUC	*Area Under Curve*
CNN	*Convolutional Neural Networks*
CRISP-DM	*Cross Industry Standard Process for Data Mining*
DL	*Deep Learning*
DNN	*Deep Neural Networks*
FP	*False Positive*
FN	*False Negative*
GARCH	*Generalized Auto-Regressive Conditional Heteroskedasticity*
GRU	*Gated Recurrent Unit*
IA	Inteligência Artificial
LSTM	*Long Short-Term Memory*
MA	*Moving Average*

MAE	*Mean Absolute Error*
MAPE	*Mean Absolute Percentage Error*
MNPE	*Maximum Negative Prediction Error*
MPPE	*Maximum Positive Prediction Error*
MS	*Mapping Study*
MSE	*Mean Squared Error*
NMSE	*Normalized Mean Squared Error*
NRMSE	*Normalized Root Mean Squared Error*
RF	*Random Forest*
RMSE	*Root Mean Squared Error*
STF	Séries Temporais Financeiras
SVM	*Support Vector Machines*
SVR	*Support Vector Regression*
TN	*True Negative*
TP	*True Positive*

Lista de Símbolos

α	Alfa
β	Beta
δ	Delta
θ	Theta

1 Introdução

O presente capítulo faz uma breve apresentação do projeto desenvolvido, expondo o seu contexto, o problema a resolver e os seus objetivos. Descreve-se a metodologia adotada, são feitas algumas considerações éticas sobre o projeto e finaliza-se com a estrutura do documento.

1.1 Contexto

O Mercado das ações, ou bolsa de valores é o local físico ou digital onde ocorrem trocas e compras de ações e títulos de organizações. A bolsa de valores possibilita às empresas angariarem capital e permite que os investidores tenham acesso a dados em tempo real, como por exemplo o preço de uma ação [1].

Basicamente, o mercado de ações tem dois objetivos, fornecer a possibilidade de organizações angariarem capital e possibilitar que investidores lucrem com a compra e venda das ações.

1.2 Problema

Num mundo tecnológico cada vez mais avançado a inflação é um problema sério que afeta todos.

Uma possível solução disponível para o público contornar ou atenuar o problema é o investimento no mercado de ações, com o propósito de gerar rendimentos para, no mínimo, negar a perda de dinheiro devido à inflação. O mercado de ações é, então, uma parte importante de qualquer economia e, por isso, compreendê-lo é objetivo de vários estudos, pois tal permite aos investidores tomar decisões mais seguras e corretas.

No entanto, as séries temporais financeiras (STF) apresentam um comportamento bastante complexo ao longo do tempo, o que as torna difícil de modelar e prever. Prova disso é a existência de várias abordagens para a análise de séries temporais. A abordagem clássica,

utilizando modelos lineares, como *Auto-Regressive Moving Average* (ARMA), e modelos heterocedásticos, como *Generalized Auto-Regressive Conditional Heteroskedasticity* (GARCH), além de, recentemente, abordagens baseadas em Aprendizagem Automática e mesmo Aprendizagem Profunda.

1.3 Objetivos

O objetivo da atual dissertação é a criação e estudo de estratégias automatizadas de previsão de tendências ou valores de séries temporais financeiras, por meio de algoritmos de estatísticos e de aprendizagem automática (AA).

Para realizar tal tarefa será necessário selecionar séries financeiras a estudar. Será ainda crucial realizar uma comparação entre os modelos criados.

No fim do projeto é espectável ter um conjunto de diferentes modelos funcionais, que consigam, com aceitável taxa de acerto prever o comportamento de uma determinada série temporal. É expectável ainda, ter uma comparação entre estes modelos.

Resumindo, os objetivos são:

- Pesquisa de literatura sobre previsão de séries temporais financeiras e abordagens existentes
- Seleção de séries financeiras
- Estudo dos métodos clássicos e de aprendizagem automática para previsão de séries temporais
- Desenvolvimento de modelos de previsão de séries temporais financeiras
- Avaliação dos modelos com várias séries
- Comparação dos resultados

1.4 Metodologia

A metodologia adotada será *Cross Industry Standard for Data Mining* – CRISP-DM é um processo independente da área da indústria e tecnologia, normalmente usado neste tipo de projetos [2]. A figura 1 é uma representação gráfica do processo:

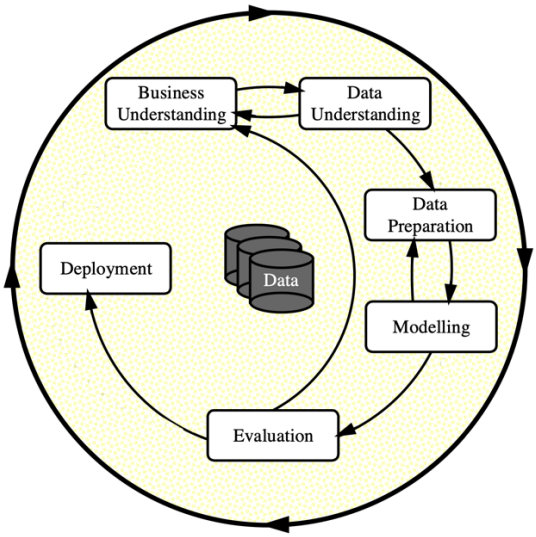

Figura 1 – Modelo de referência genérico CRISP-DM [2]

Bussiness Understanding – Fase inicial que se foca na compreensão de problemas e objetivos do projeto, neste caso, é a fase de pesquisas relacionadas com séries temporais financeiras e algoritmos de previsão de séries temporais.

Data Understanding – Esta fase tem como objetivo estudar os dados recolhidos, identificar problemas de qualidade de dados e encontrar subgrupos de dados mais relevantes. Neste projeto, esta fase passa pela pesquisa do conjunto de dados a ser usado, assim como a sua composição e que variáveis financeiras devem ser consideradas.

Data Preparation – Esta fase cobre todas as atividades relacionadas com a obtenção do conjunto de dados final. Nesta dissertação, após selecionados os dados financeiros a estudar, será necessário a criação de um conjunto de dados, apenas com os dados necessários. Será ainda preciso reajustar este novo conjunto de dados, removendo entradas sem valores.

Modeling – Nesta fase, várias técnicas de modelação são aplicadas. Como diferentes técnicas requisitam que os dados sejam fornecidos em diferentes formatos, existe uma forte ligação desta fase com a anterior. No contexto do atual trabalho, esta é a fase onde os modelos clássicos e de aprendizagem automática serão treinados e ajustados para obter o melhor desempenho.

Evaluation – Nesta fase é espectável que já exista um ou mais modelos criados. É necessário então, analisar e avaliar, não só, os modelos criados, como também os passos seguidos para os criar, com o objetivo de garantir que nada foi esquecido. Após a criação e ajuste dos modelos, estes serão avaliados usando as métricas apropriadas.

Deployment – A criação do modelo, por norma, não é o objetivo final de um projeto, por vezes é a elaboração de um relatório, ou publicação do modelo numa página. Esta fase é dedicada a

isso, realizar as ações necessárias para disponibilizar o modelo. No caso deste estudo, este passo é dedicado à elaboração da dissertação.

De referir que, como a figura 1 demonstra, o processo é cíclico e ágil, uma vez que é possível voltar a passos anteriores para realizar ajustes.

1.5 Considerações éticas

Uma vez que este tema está relacionado com a área financeira, os modelos obtidos correm sempre o risco de serem usados para fins lucrativos. Na eventualidade dos modelos serem disponibilizados publicamente, é imperativo que utilizadores e leitores saibam que o atual projeto é puramente académico e os modelos, independentemente do desempenho que demonstrem, podem ter falhas colocando o capital investido em risco.

De referir que, o trabalho desenvolvido nesta dissertação, não deve ser usado a nível comercial, sendo única e exclusivamente um projeto académico. De acrescentar que, ninguém envolvido nesta dissertação se responsabiliza por perdas geradas devido à aplicação dos modelos referidos.

1.6 Estrutura do Documento

O documento em causa encontra-se dividido em 6 partes, sendo estas a Introdução, Estado da Arte, Dados, Implementação, Resultados e Conclusões.

O capítulo da Introdução foca-se em oferecer uma contextualização global do que é abordado no documento. Aqui é dada uma contextualização inicial do problema, objetivos, a metodologia a seguir e algumas considerações éticas.

O seguinte capítulo, Estado da Arte, tem como objetivo fornecer um enquadramento teórico para que o leitor fique a entender melhor sobre séries temporais financeiras e métodos de previsão das mesmas, assim como métricas para avaliação destes métodos. Além disso, esta secção é ainda responsável por apresentar os principais estudos já realizados nesta área.

O terceiro capítulo, Dados, fornece uma descrição e análise das séries temporais a serem usadas.

No capítulo da Implementação são expostas as partes mais relevantes da implementação assim como discussões sobre as decisões tomadas.

No quinto capítulo, são apresentados os resultados obtidos após a aplicação dos diferentes modelos a cada série temporal.

Por fim, no último capítulo, são tiradas algumas conclusões referentes ao estudo realizado, além de serem sugeridos alguns passos a seguir no futuro.

2 Estado da Arte

Este capítulo é dedicado à exploração de conceitos relacionados com o tema, explorando conteúdos relacionados com séries temporais, modelos estatísticos, modelos de aprendizagem automática, assim como métricas para os avaliar.

2.1 Séries Temporais

2.1.1 Definição de Série Temporal

Séries Temporais, ou *Time Series*, são conjuntos de valores/pontos organizados que ocorrem sequencialmente, num dado período, tal implica que, estes pontos têm uma dependência temporal intrínseca, contrastando com os dados "habituais" (*Cross-Sectional data*), que são analisados num ponto temporal específico, ou onde o tempo não tem importância. Os dados das séries temporais dependem de uma variável de indexação (tempo), ou seja, o Y deixa de ser uma coleção independente de observações e passa a ser uma coleção de pontos sucessivos [3]. A figura 2 mostra o exemplo de uma série temporal.

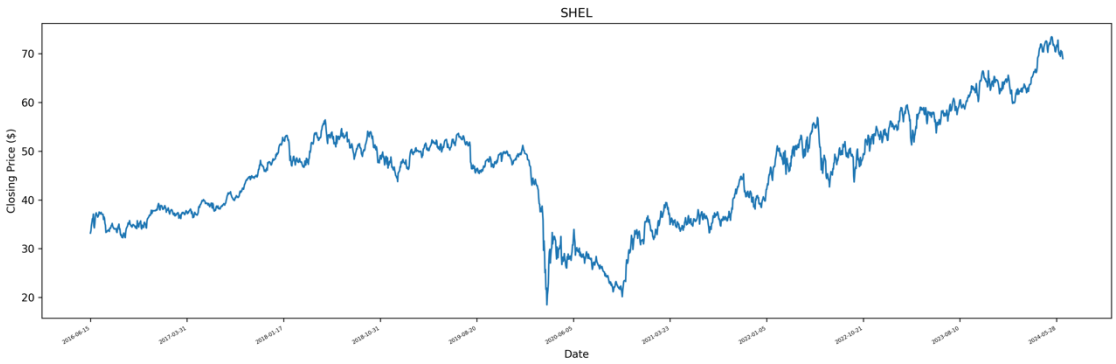

Figura 2 – Série Temporal SHEL

2.1.2 Componentes de uma Série Temporal

Em geral, uma série temporal exibe uma variedade de padrões resultantes da combinação de três principais componentes que caracterizam a série, figura 3. Esses componentes podem ser identificados ao decompor a série através dos dados observados e são descritos por [4] [5]:

- Tendência (T) – É o aumento, diminuição ou estagnação ao longo do tempo
- Sazonalidade (S) – É a variação dentro de um período específico, normalmente causada por fatores externos
- Resíduo (R) – São flutuações de curto prazo que não são sistemáticas nem previsíveis

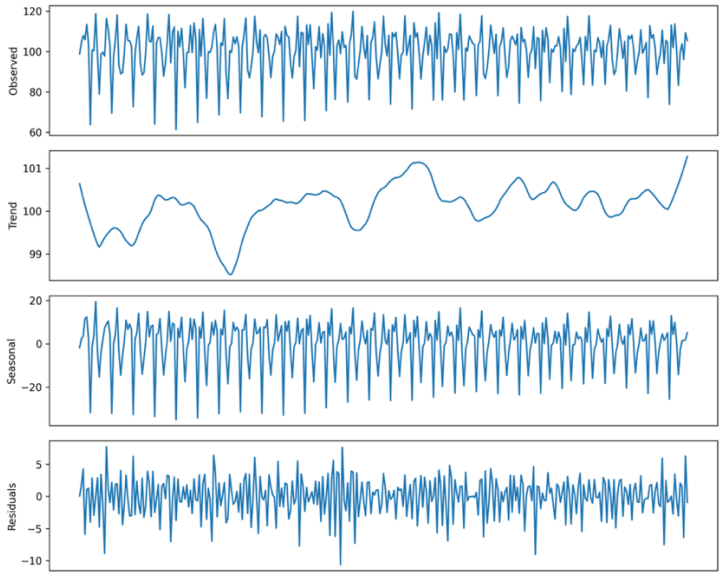

Figura 3 – Exemplo de decomposição de uma série temporal

Considerando x_t uma série temporal, pode-se descrever os dois tipos de modelos mais empregados em séries temporais, que consideram os efeitos desses componentes da seguinte forma:

- Modelo aditivo – Onde se assume a independência entre quaisquer componentes.

$$x_t = T_t + S_t + R_t \quad (1)$$

- Modelo multiplicativo – Onde um componente pode afetar um outro, ou seja, pode existir interação entre os componentes.

$$x_t = T_t * S_t * R_t \quad (2)$$

Dessa forma, a grande diferença entre eles reside na premissa assumida pelos modelos aditivos de independência entre os componentes.

2.1.3 Estacionariedade

Um conceito importante a ter em consideração quando se fala de séries temporais é estacionariedade. Uma série temporal diz-se estacionária se a média, variância e autocorrelação se mantêm constantes e demonstram independência temporal [6].

O teste Dickey-Fuller [7] serve para verificar a estacionariedade de uma série. Este teste baseia-se na verificação dos pressupostos estatísticos de que a média e a variância se mantêm constantes ao longo do tempo.

Caso uma série temporal não seja estacionária é necessário aplicar a técnica de diferenciação (*differencing*).

2.1.4 Diferenciação

Diferenciação (*differencing*) é uma transformação que calcula a diferença entre um ponto e outro. Esta transformação é útil para estabilizar a média [6].

2.1.5 Random Walk

Uma *Random Walk* é uma série temporal cuja primeira diferenciação (diferenciação de dois valores vizinhos (t_2-t_1)) é estacionária e não correlacionada [6].

2.1.6 Séries Temporais Financeiras

Uma série temporal financeira é uma sequência de pontos que representam o valor de um objeto financeiro ao longo do tempo (por exemplo, ações).

Estas séries estão dependentes de uma variável temporal, pelo que, os valores apresentam uma ordem cronológica específica. Além disso, tendem a ser não estacionárias.

2.2 Modelos para Análise de Séries Temporais

O estudo [8] coleciona e sintetiza os artigos relacionados com a previsão de séries temporais financeiras entre os anos 2011 e 2021. De notar que, este estudo resume sistematicamente a pesquisa inicial no que toca ao tema de previsão de séries temporais financeiras usando métodos de Aprendizagem Automática, referindo ainda algoritmos clássicos. No artigo é apresentada a taxonomia das principais técnicas aplicadas para previsão de séries temporais financeiras, figura 4.

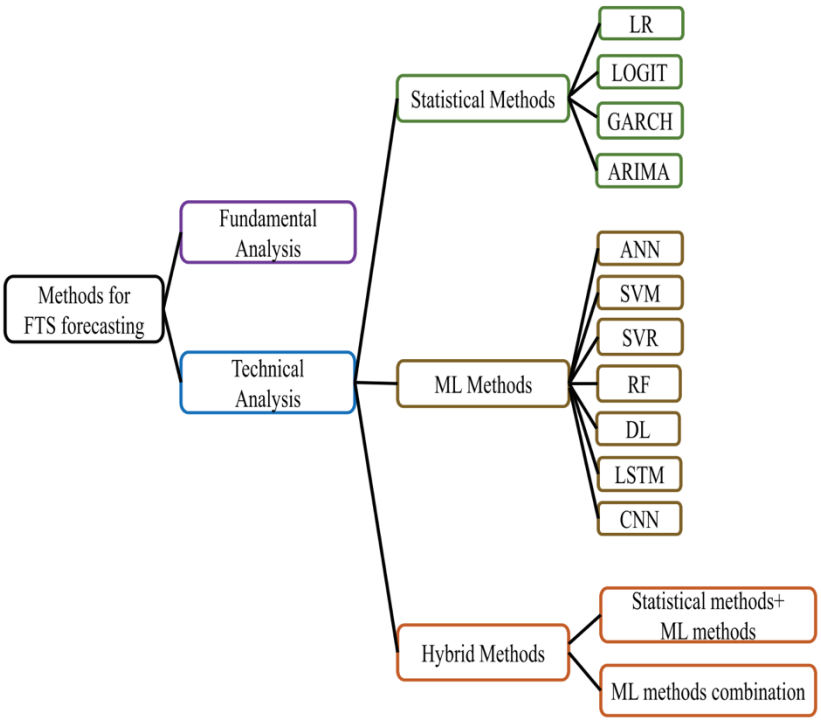

Figura 4 – Taxonomia dos métodos de previsão de séries temporais financeiras [8]

Como é possível observar na figura 4, existem vários tipos de métodos para abordar o problema. Focando nos métodos estatísticos, no âmbito deste projeto, apenas ARMA e GARCH serão abordados. Para métodos de Aprendizagem Automática serão explorados os regressores de *Decission Tree*, *K Neighbors*, *Bagging*, *Random Forest*, *XGBoost* e *SVR*. Por fim, os métodos híbridos, com os quais se visa explorar uma combinação de modelos que trabalhando em conjunto obtêm melhor desempenho do que cada um dos modelos individuais.

Métodos puramente estatísticos têm vários problemas quando tentam lidar com a quantidade massiva de dados complexos e ruidosos presentes nas STFs. De acordo com [8] "Os dados do STF estão a tornar-se cada vez mais difíceis de prever. Embora ARMA, GARCH e outros modelos estatísticos possam considerar as características dependentes da sequência dos dados do STF, a determinação deste fator necessita de ser determinada através de testes econométricos complexos; caso contrário, será difícil realizar a identificação automática com base nos problemas de pesquisa."[1].

Os algoritmos clássicos, ou estatísticos, são, como o nome indica, algoritmos que se baseiam em métodos estatísticos para realizar previsões de valores futuros. Aqui serão explorados principalmente duas famílias de modelos estatísticos, ARMA e GARCH. De referir que, estes modelos já foram intensivamente explorados e várias versões dos mesmos já foram criadas,

[1] Tradução livre do autor, no Original: "FTS data are becoming increasingly difficult to predict. Although ARMA, GARCH and other statistical models can consider the sequence-dependent characteristics of FTS data, the determination of this factor either needs to be determined through complex econometric tests; otherwise, it is difficult to realize automatic identification based on the research problems."

como demonstrado, por exemplo, no estudo [9]. Tendo isto em conta, neste estudo apenas serão consideradas os modelos ARIMA, SARIMA e GARCH.

Como o estudo [10] comprovou, modelos da família ARMA, embora clássicos, podem ser usados para previsão de séries temporais financeiras com confiança (para horizontes não muito grandes), uma vez que produzem previsões não muito longe do valor real.

Aprendizagem automática tem atraído rapidamente vários investigadores, que por sua vez, têm produzido vários resultados académicos.

Uma outra abordagem que será explorada são modelos híbridos. Estes podem ser vantajosos, uma vez que, combinam diferentes modelos para obter melhores previsões. Como o estudo [11] conclui, em algumas situações pode ser vantajoso explorar esta opção, uma vez que, neste caso, mesmo depois de aplicado um modelo da família ARMA, é notável alguma volatilidade nos seus resíduos.

A figura 5 apresenta os 10 estudos mais citados na área [8] e os seus detalhes são apresentados na figura 6.

Sequence	Title	Journal	Authors	Year	Total citations	Citations per year
1	Deep learning with long short-term memory networks for financial market predictions	European Journal of Operational Research	Thomas et al.	2018	368	92
2	A hybrid stock selection model using genetic algorithms and support vector regression	Applied Soft Computing	Huang et al.	2012	113	11.3
3	Deep learning-based feature engineering for stock price movement prediction	Knowledge-Based Systems	Wen et al.	2019	83	27.67
4	ModAugNet: A new forecasting framework for stock market index value with an overfitting prevention LSTM module and a prediction LSTM module	Expert Systems with Applications	Baek et al.	2018	74	18.5
5	A novel hybrid model using teaching–learning-based optimization and a support vector machine for commodity futures index forecasting	International Journal of Machine Learning and Cybernetics	Prasad et al.	2018	63	15.75
6	Fractional Neuro-Sequential ARFIMA-LSTM for Financial Market Forecasting	IEEE Access	Hussain et al.	2020	56	28
7	Stock market one-day ahead movement prediction using disparate data sources	Expert Systems with Applications	Bin et al.	2017	53	10.6
8	Bridging the divide in financial market forecasting: machine learners vs. financial economists	Expert Systems with Applications	Hsu et al.	2016	52	8.67
9	Technical analysis and sentiment embeddings for market trend prediction	Expert Systems with Applications	Andrea et al.	2019	50	16.67
10	Complex Neurofuzzy ARIMA Forecasting A New Approach Using Complex Fuzzy Sets	IEEE Transactions on Fuzzy Systems	Li et al.	2013	49	5.44

Figura 5 – Os 10 artigos mais citados na área [8]

Sequence	Data sets	Time duration	Assets	Predictive variables	Predictions	Main methods	Performance measures
1	S&P500	December,1992 - October,2015	Index	Return indices	Price	LSTM	Accuracy, Standard deviation
2	Taiwan Stock Exchange	1996–2010	Stock	Stock return	Price	hybrid GA-CSVR	Accuracy, Standard deviation
3	CSI 300	December 9, 2013 - December 7, 2016	Stock	Price	Volatility	MFNN	Accuracy
4	KOSPI200 S&P500	January 4, 2000 - July 27, 2017	Stock	Price	Volatility	LSTM	MSE,MAPE MAE
5	MCX COMDEX	January 1,2010 - May 7, 2014	Commodity futures index	Index	Volatility	SVM-CTLBO	RMSE,NMSE MAE,DS
6	Fauji Fertilizer Company (FFC)	January 1, 2009 - May 30, 2018	Stock	Price	Price	ARFIMA-LSTM	MAE,RMSE, MAPE
7	AAPL(Apple NASDAQ)	May 1,2012 - June 1,2015	Stock	Price	Volatility	DT,NN SVM	Accuracy,AUC, Precision, Sensitivity, Specificity
8	TickWrite Data Inc.	2008-C2014	Stock indices	Price	Direction	SVM ANN	ROI Accuracy
9	NASDAQ100	July 3,2017 - June 14,2018	Stock	Price	Direction	RF,SVM NN	Accuracy Recall
10	NASDAQ composite index TAIEX DJI	January 3, 2007 - December 20, 2010 1999 to 2004 1999 to 2004	Stock index	Index	Index value	CNFS-ARIMA	MSE NRMSE NRMSE

Figura 6 – Detalhes sobre os 10 estudos mais citados na área [8]

Desta lista, o estudo número 6 [12] apresenta mais interesse, uma vez que, além de estar a modelar uma ação, os preços anteriores serão usados como previsores para preços futuros.

Embora não seja possível uma comparação direta entre o estudo [12] e os resultados obtidos nesta dissertação, uma vez que são usadas séries diferentes e são previstas variáveis diferentes, este serve como um bom ponto de partida e referência.

2.2.1 Modelos AR(p)

Os modelos AR (*Auto-Regressive*) tentam realizar previsões, tendo por base, única e exclusivamente, os valores passados (também conhecidos como *Lags*). A letra "p" representa o número de *Lags* a considerar. Um modelo que apenas dependa de um valor anterior é referido como AR(1) e a sua expressão seria:

$$Y_t = \beta + \alpha Y_{t-1} + e_t \quad (3)$$

Onde Y_t é o valor a prever, β é o valor de interceção, α é o coeficiente, Y_{t-1}é o *Lag*, e_t é o erro associado com a previsão.

Como é observável na fórmula anterior, o valor a prever Y_t, está dependente do Y anterior, Y_{t-1}.

Se a mesma fórmula fosse aplicada para obter o valor de Y_{t-1} então obtinha-se:

$$Y_{t-1} = \beta + \alpha Y_{t-2} + e_{t-1} \quad (4)$$

Como é possível ver, o valor anterior também depende do valor que o precede. Este encadeamento de dependências continua até ser atingido o primeiro valor da série temporal, que depende de nenhum valor anterior.

Graficamente, a lógica referida anteriormente, pode ser explicada a figura 7.

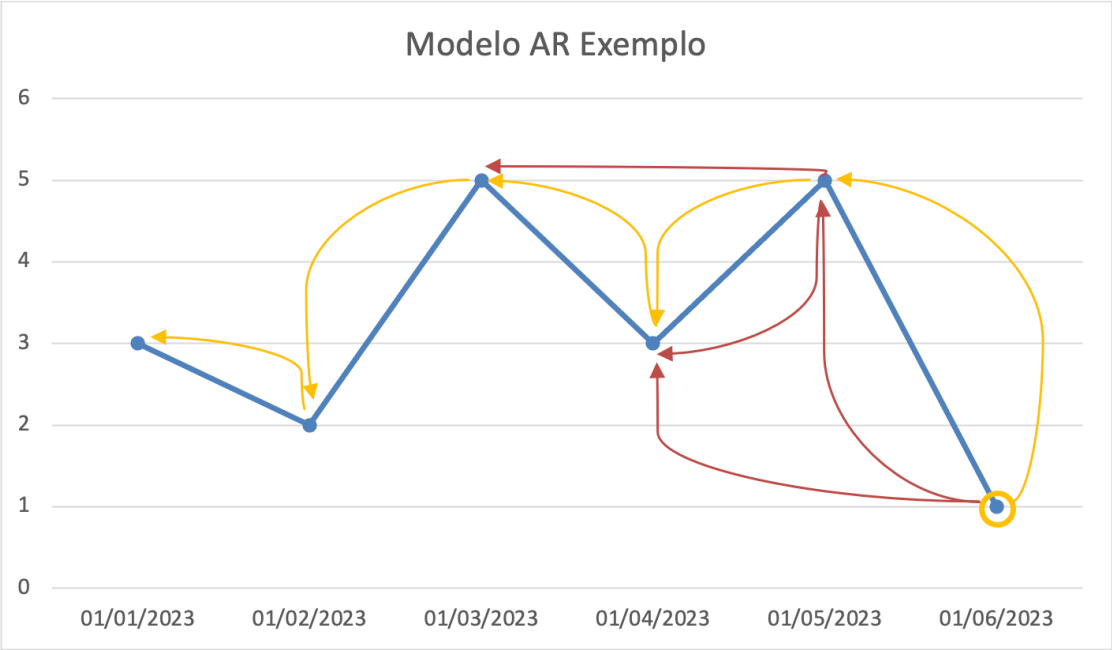

Figura 7 – Exemplo Modelo AR(1) e AR(2) [13]

Na figura 7 está representado um conjunto de valores exemplo para auxiliar na explicação do funcionamento do modelo AR(p). Para prever o valor rodeado a amarelo, o modelo vai usar o valor do Y anterior (setas amarelas), que, por usa vez irá usar o valor do Y anterior a si e assim sucessivamente. Resumindo, todos os valores terão impacto, embora alguns muito pouco, na previsão a realizar, e precisamente por isso é que estes modelos se chamam modelos de longa memória [13].

No caso de se usar mais *Lags*, 2 por exemplo, cada previsão estaria não só dependente do último valor, como também do penúltimo. É possível ver como ficaria a representação gráfica de AR(2) olhando para as setas vermelhas (apenas foram adicionadas setas vermelhas nos dois últimos pontos para manter o gráfico fácil de interpretar).

2.2.2 Modelos MA(q)

Os Modelos MA (*Moving Average*) também tentam prever valores futuros usando valores obtidos anteriormente, mas em vez de usar valores de Y anteriores, usam valores de erro anteriores, ou seja, o valor do erro da previsão do Y anterior, impactará a previsão do Y atual. Estes erros anteriores chamam-se *Error Lags*. A letra "q" representa o número de *Error Lags* a considerar.

Expressão para MA(1):

$$Y_t = \beta + \delta e_{t-1} + e_t \quad (5)$$

Na equação anterior Y_t é o valor a prever, β é o ponto de interseção, δ é o coeficiente, e_{t-1} é o erro da previsão anterior e e_t é o erro da previsão atual.

Como esperado, o erro da previsão anterior vai, de facto, impactar a próxima previsão, mas, ao contrário do que acontece com os modelos AR, neste caso, nem todos os *Error Lags* influenciarão a previsão. Tal é simples de entender seguindo o seguinte exemplo [14]:

$$Y_{ontem} = \beta + \delta e_{ante-ontem} + e_{ontem} \quad (6)$$
$$Y_{hoje} = \beta + \delta e_{ontem} + e_{hoje} \quad (7)$$
$$Y_{amanhã} = \beta + \delta e_{hoje} + e_{amanhã} \quad (8)$$

Acima estão três formulas diferentes, a primeira representa a previsão do valor de ontem, a segunda a previsão do valor de hoje e a terceira a previsão do valor de amanhã.

Interpretando a primeira formula, é possível entender que duas coisas impactam a obtenção do valor de ontem, o erro do dia anterior ($e_{ante-ontem}$) e o erro de ontem (e_{ontem}).

Desviando a atenção para a segunda fórmula, percebe-se que o erro de ontem impacta o resultado de hoje. Contudo, ao analisar a última fórmula é facilmente visível que o erro de ontem não tem qualquer impacto na previsão do valor de amanhã, demonstrando assim, que nem todos os *Error Lags* impactam a previsão. Esta apenas vai depender do valor de "q" (número de erros escolhidos para usar no modelo) MA(q). Em oposição ao modelo anterior (AR), os Modelos MA são caraterizados como modelos de memória curta (*short memory*).

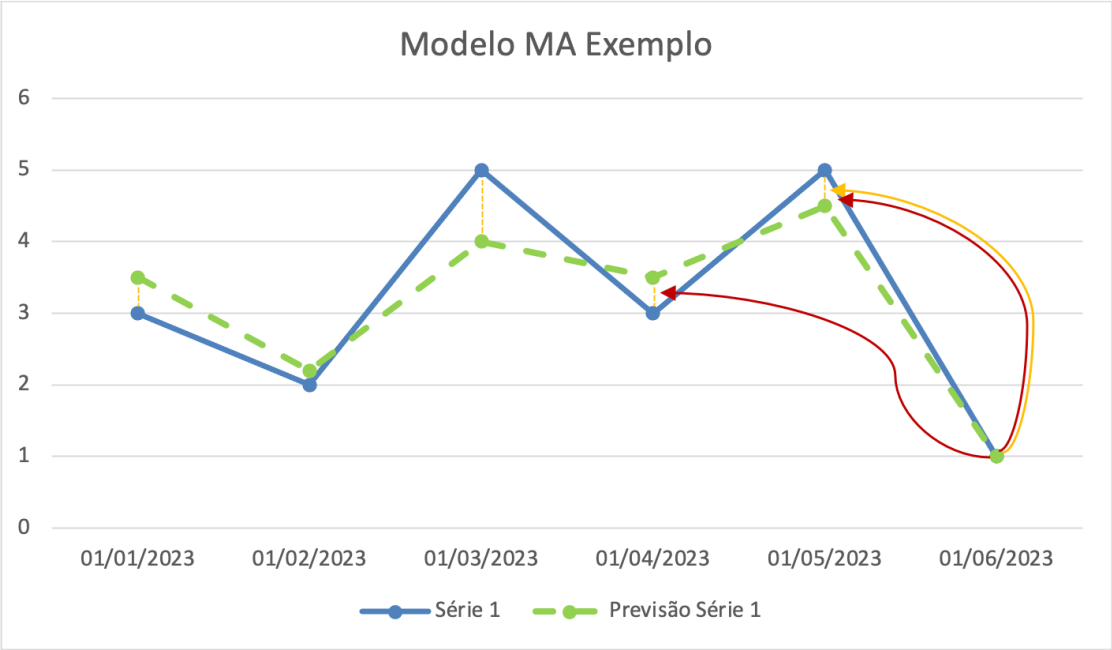

Figura 8 – Exemplo Modelo MA(1) e MA(2) [14]

A figura 8 é uma representação visual de como funciona um modelo MA. Para MA(1) a seta amarela demonstra como o erro da previsão anterior afeta a previsão atual, enquanto que, as setas vermelhas, demonstram como funcionaria para um MA(2).

2.2.3 Modelos ARMA(p,q)

Um modelo ARMA (*Auto-Regressive Moving Average*) é, nada mais, nada menos, que a junção de um modelo AR com um MA. Tipicamente, os modelos ARMA conseguem modelar tão bem como um modelo AR ou MA, mas usando menos termos [15].

Posto isto, a fórmula que representa um modelo ARMA(p,q) é:

$$Y_t = \beta + \alpha_1 Y_{t-1} + \cdots + \alpha_p Y_{t-p} + \delta_1 e_{t-1} + \cdots + \delta_q e_{t-q} + e_t \quad (9)$$

Importante referir que, modelos ARMA são bons a realizar previsões a curto termo, mas o seu desempenho tende a diminuir com previsões a longo termo [16].

Como já foi referido, os modelos ARMA usam valores passados para prever o futuro, através da avaliação de valores de Y (AR) e dos erros (MA) anteriores. Neste tipo de modelos a variância tem de ser incondicional, ou seja, constante. Isto é um problema quando estes modelos são aplicados para a previsão de séries temporais financeiras [17]. Tal é facilmente visível com o exemplo dado pelo professor Zivot E. [17] "Suponha que notamos que os retornos diários recentes têm sido invulgarmente voláteis. Poderemos esperar que os retornos de amanhã também sejam mais voláteis que o costume. No entanto, um modelo ARMA não consegue

capturar este tipo de comportamento pois a sua variância condicional é constante [ou seja, é incondicional].”[2].

2.2.4 Modelos ARIMA(p,d,q)

O modelo ARMA apresenta um problema, apenas pode ser aplicado a dados estacionários (ou seja, a distribuição dos dados apenas depende de diferenças em tempo e não em localizações em tempo).

Para resolver este problema foi acrescentada a componente “I” (*Integrated*) ao modelo, que levou ao surgimento dos modelos ARIMA (*Auto-Regressive Integrated Moving Average*).

De forma simplificada, este modelo vai transformar os dados não estacionários em dados estacionários, tipicamente usando diferenciação.

Nestes modelos, “p” representa o número de termos AR, “d” o número de diferenciações e “q” o número de termos MA.

2.2.5 Modelos SARIMA(p,d,q) (P,D,Q)s

O modelo ARIMA pode encontrar algumas dificuldades quando se depara com séries temporais com presença de sazonalidade. Para contornar esse problema é então acrescentada mais uma camada de complexidade, criando o modelo SARIMA (*Seasonal Auto-Regressive Integrated Moving Average*).

A introdução de sazonalidade implica, também, a introdução de ciclos. O ciclo representa um padrão repetível ao longo do tempo, muitas vezes influenciado por fatores exteriores, tais como, fatores políticos, ambientais e económicos [6]. Este modelo introduz novos parâmetros, nomeadamente “P”, “D” e “Q”, que têm o mesmo significado que os parâmetros “p”, “d” e “q”, mas aplicados à componente sazonal da série. Existe, ainda, outra nova variável, o “s”, que representa frequência, ou seja, o número de observações por ciclo. Assumindo, por exemplo, um ciclo anual e observações mensais, é possível concluir que a frequência será 12 (uma vez que existem 12 meses num ano).

[2] Tradução livre do autor, no Original: ”Suppose we have noticed that recent daily returns have been unusually volatile. We might expect that tomorrow’s return is also more variable than usual. However, an ARMA model cannot capture this type of behavior because its conditional variance is constant.”

2.2.6 Modelos ARCH

O modelo ARCH (*Auto-Regressive Conditional Heteroskedasticity*) foi introduzido em 1982 por Engle R. [18] e, em oposição ao que tendia a acontecer até à altura, este modelo foca-se na modelação da volatilidade dos valores e a influência que esta tem na média [19].

Ao contrário da variância incondicional, tipicamente usada, a variância condicional é afetada por informação externa. Diz-se que este tipo de dados apresenta heterocedasticidade, ou em inglês, *heteroskedasticity* [19].

Como o nome indica, os modelos ARCH seguem uma abordagem Autorregressiva (*Auto-Regressive Approach*), semelhante aos modelos AR, mas em vez de modelar valores absolutos, este modela a volatilidade da variância. Para um modelo ARCH(q) obtém-se a seguinte fórmula:

$$\mathrm{a}_t = \varepsilon_t + \sqrt{\alpha + \alpha_1 {a_{t-1}}^2 + \alpha_2 {a_{t-2}}^2 + \cdots + \alpha_q {a_{t-q-1}}^2} \qquad (10)$$

Onde a_t é o valor a prever, ε_t é ruido ou erro, α são constantes, ${a_{t-1}}^2$ é o valor de anterior elevado a 2.

Estes modelos ARCH(q) tendem a ter picos de volatilidade que acabam por não corresponder à realidade, o que é problemático. Os modelos GARCH tentam resolver este problema.

2.2.7 Modelos GARCH

A fórmula de um modelo GARCH (*Generalized Auto-Regressive Conditional Heteroskedasticity*) é bastante semelhante ao do modelo ARCH, sendo a única diferença a consideração da volatilidade de valores anteriores. A lógica por trás deste acréscimo é: se no passado o valor se afastou bastante da sua média, então, é provável que o valor atual também se afaste. Em suma, isto permite propagar a volatilidade ao longo do tempo, evitando assim picos de volatilidade que não correspondem à realidade.

Portanto, para um modelo GARCH(p,q) tem-se:

$$\mathrm{a}_t = \varepsilon_t + \sqrt{\alpha + \alpha_1 {a_{t-1}}^2 + \cdots + \alpha_p {a_{t-p}}^2 + \beta_1 {\theta_{t-1}}^2 + \cdots + \beta_q {\theta_{t-q}}^2} \qquad (11)$$

Onde todos as variáveis tomam o mesmo valor que no método ARCH, e β_q é uma constante e ${\theta_{t-1}}^2$ é a volatilidade da potência de dois do valor anterior.

2.3 Inteligência Artificial

Inteligência Artificial, ou *Artificial Intelligence* (AI) é, como descrita por Ghosh R. [20], "[...] a ciência e engenharia do desenho de máquinas inteligentes, particularmente programas computacionais inteligentes."[3].

Cada vez mais AI tem avançado e entrado no dia a dia de pessoas comuns, o que muitas vezes faz com que termos como "Inteligência Artificial" (IA) (*Artificial Intelligence*), "Aprendizagem Automática" (AA) (*Machine Learning*) e "Aprendizagem Profunda" (AP) (*Deep Learning*) sejam usados como sinónimos, o que é errado.

Como referido por Esteves H. [21] estes três termos são distintos.

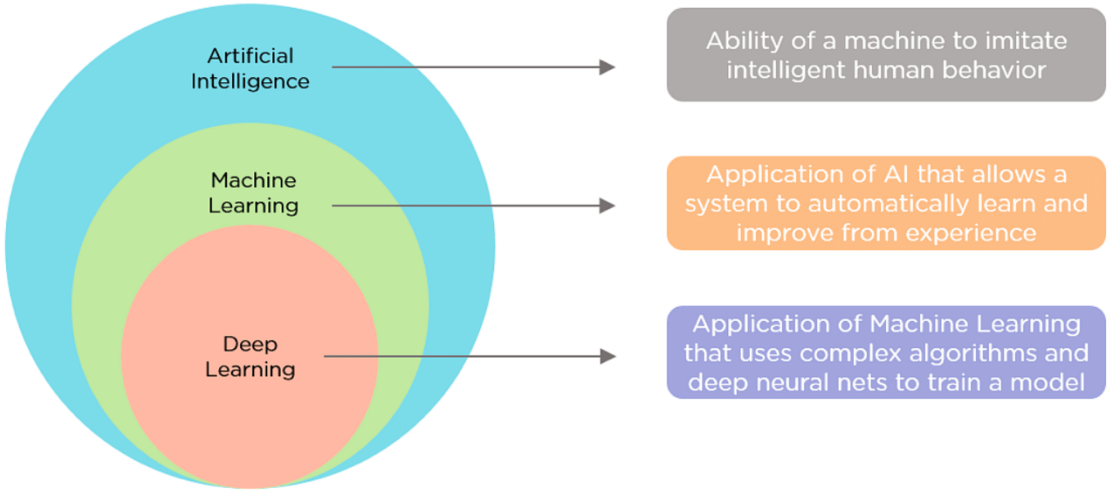

Figura 9 – Conceitos associados a Inteligência Artificial [21]

Como a figura 9 demonstra *Deep Learning* é uma subsecção do campo de *Machine Learning*, que, por sua vez, está incluída no campo de *Artificial Intelligence*.

2.3.1 Aprendizagem Automática

Aprendizagem Automática é a área de ciências computacionais que se foca em replicar a maneira como os seres humanos aprendem, usando, para isso, dados e diversos algoritmos. Os algoritmos são treinados para realizar classificações ou previsões (dependendo do problema em questão) ou até para descobrirem informação importante (em projetos de mineração de dados) [22].

[3] Tradução livre do autor, no Original: "… is the science and engineering of designing intelligent machines particularly intelligent computer programs."

Uma outra maneira de definir o que é aprendizagem automática é a fornecida por Mitchell T. [23] "Um programa computacional é dito ter aprendido com a experiência E, com respeito a uma classe de tarefas T, e desempenho medido D, se o seu desempenho em tarefas T, medido por D, melhora com a experiência E."[4]

Os algoritmos de Aprendizagem Automática podem pertencer a quatro categorias:

- **Aprendizagem Supervisionada –** Neste tipo de aprendizagem os dados para treino já se encontram rotulados. Enquanto os dados de treino são fornecidos ao modelo, este altera os pesos dos mesmos, até serem ajustados apropriadamente [22]. Esta aprendizagem pode dar resposta a dois tipos de problemas, problemas de regressão e problemas de classificação. Para problemas de regressão a variável objetivo é uma variável quantitativa, por exemplo, prever a temperatura do dia seguinte. Por outro lado, num problema de classificação, o valor obtido pertence a um conjunto finito e não organizado de categorias, por exemplo, se a temperatura do dia seguinte é baixa ou elevada (figura 9).

- **Aprendizagem Não Supervisionada –** Por sua vez, este tipo de aprendizagem usa algoritmos já existentes de AA para organizar dados (que não estão rotulados). O objetivo é que estes referidos algoritmos encontrem, autonomamente, padrões nos dados, sem intervenção humana. Precisamente por causa desta capacidade de deteção de padrões nos dados, este tipo de aprendizagem torna-se ideal para análises exploratórias [22].

- **Aprendizagem Semi-supervisionada –** Como o nome indica, esse tipo de aprendizagem é uma mistura das duas já referidas. Durante a fase de treino é fornecido um conjunto mais pequeno de dados rotulados, com o objetivo de guiar a classificação e extração de previsores do maior conjunto de dados não rotulados [22].

- **Aprendizagem por Reforço –** Este tipo de aprendizagem diferencia-se um pouco das supracitadas, uma vez que o modelo não recebe um conjunto de treino. O modelo aprende à base de tentativa erro, onde um conjunto de tentativas bem-sucedidas são recompensadas [22].

A aprendizagem automática tipicamente divide-se em duas partes. Na primeira parte é realizada a seleção dos previsores importantes e os dados são divididos em conjuntos de treino, de teste e de validação. Na segunda fase o modelo é testado para se prever o desempenho do modelo [8].

[4] Tradução livre do autor, no Original: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."

2.3.2 Algoritmos de Aprendizagem Automática

Como as séries temporais são compostas por valores contínuos, e o objetivo deste estudo é a previsão dos mesmos, é necessário recorrer a algoritmos de regressão.

2.3.2.1 Support Vector Regressor (SVR)

SVRs são bastante semelhantes a SVMs a nível tecnológico, uma vez que, são a versão de regressão destas mesmas. Semelhantemente ao que acontece com SVMs, SVRs conseguem transformar informação complexa e não linear em *high-dimensional feature space* e gerar um híper-plano aplicando vários modelos *kernel*, com o objetivo de aumentar a exatidão da previsão do valor de saída. Durante a computação, dependências e relações entre atributos podem ser descobertas automaticamente [8]. Esta última funcionalidade é uma mais-valia para a previsão de séries temporais financeiras.

SVRs são, entre os restantes modelos de AA, os melhores para implementar regressões não lineares com um conjunto de dados pequeno [8].

Por outro lado, SVRs precisam de mais memória para reservar os vetores de suporte. Para além disso, a seleção do *kernel* a usar tende a não ser fácil, tal como, o cálculo do coeficiente dos atributos dos dados, a interpretação do processo e generalização do processo de regressão [8].

2.3.2.2 Decision Tree Regressor (DTR)

Decision Tree, ou árvore de decisão, é um método de aprendizagem automática supervisionado. O seu objetivo é a criação de um modelo que, inferindo regras de decisão baseadas nos dados, consiga prever valores futuros [24]. O modelo gerado representará uma aproximação por partes, como exemplifica a figura 10.

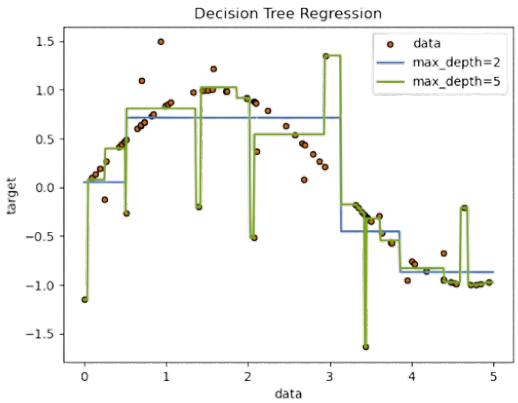

Figura 10 – Exemplo do funcionamento de uma regressão usando uma árvore de decisão [24]

Como a figura 10 mostra, a quantidade de detalhes que a árvore "aprende" é controlada pelo parâmetro `max_depth` e caso este seja demasiado elevado o modelo irá sofrer de *overfitting*.

2.3.2.3 Random Forest Regressor (RFR)

Random Forest é um algoritmo de aprendizagem supervisionada que tem as suas bases no algoritmo de *Decision Tree*. O algoritmo de *Random Forest* é um *ensemble*, o que significa que combina métodos diferentes para tentar obter resultados melhor. Neste caso, cria várias árvores de decisão, cada uma com um conjunto aleatório de previsores (isto é realizado usando a técnica de *bootstrap*), que leva à criação de diferentes *outputs*. Para obter um *output* final, cada árvore "lança o seu voto" e o resultado é decidido com base na maioria. Uma das vantagens do funcionamento do algoritmo prende-se com a evitação automática de *overfitting* [25].

De referir que, a *Random Forest*, semelhantemente ao que acontece nas *Decision Trees*, possui vários híper parâmetros que podem ser usados para melhorar os resultados. Estes parâmetros costumam ser [26]:

- `n_estimators` – É o número de *decision trees* geradas, por norma quanto maior melhor, mas tem custos em termos de desempenho.
- `max_depth` – Usado para definir o número máximo de níveis das árvores, ter muitos níveis pode causar *overfitting*.
- `min_samples_split` – É o número mínimo de elementos necessários para dividir um nó interno.
- `min_samples_leaf` – É o número mínimo de elementos necessários para um nó ser uma folha, ou seja, um nó final apenas se irá dividir em dois se estas novas folhas tiverem este número mínimo de elementos.
- `oob_score` – Opção para usar *out of the bag score.*

Algumas das desvantagens de *Random Forests* são, o facto de alterações pequenas nos dados de treino poderem ter um enorme impacto no modelo estabelecido. *Random Forests* podem dar problemas a nível de consumo de recursos computacionais, aquando da utilização do algoritmo CART e, no que toca ao crescimento da árvore, o melhor previsor vai ser sempre escolhido para se propagar nos nós, evitando a exploração de outros previsores [8].

2.3.2.4 Extreme Gradient Boost Regressor (XGB)

XGB é um algoritmo de aprendizagem supervisionada baseado na implementação do algoritmo *Gradient Boosting*, que cria um *ensemble* sequencial de árvores de decisão, onde, cada árvore, tenta corrigir erros da árvore anterior.

Umas das maiores vantagens do método XGBoost é a sua velocidade e a habilidade de lidar com conjuntos de dados relativamente grandes [27].

Este método tende a obter melhores resultados que *Random Forest* [27] (que também é um *ensemble* que usa *Decission Trees*).

O método tem bastante híper parâmetros para, sendo alguns deles [27]:

- `n_estimators` – É o número de *decision trees* geradas (por norma quanto maior melhor) mas tem custos em termos de desempenho.
- `max_depth` – Usado para definir o número máximo de níveis das árvores (ter muitos níveis pode causar *overfitting*).
- `learning_rate` – Usado para ajustar a contribuição de cada árvore no resultado.
- `subsample` – Representa a fração de dados de treino a serem usados por cada árvore.
- `colsample_bytree` – Usado para ajustar a percentagem de previsores a serem usados por cada árvore.

2.3.2.5 Nearest Neighbors Regressor (KNR)

Na regressão baseada nos vizinhos mais próximos, o valor a prever é obtido tendo em conta a média dos valores vizinhos mais próximos.

Existem diferentes implementações desta regressão, mas a abordada aqui é o regressor de K vizinhos, onde K representa o número de vizinhos usados pelo método [28].

O peso com que cada vizinho influencia o resultado final pode ser modificado, podendo este tomar uma forma uniforme, onde todos os vizinhos têm o mesmo peso, ou com base na distância, onde vizinhos mais distantes impactam menos o resultado final [28], como exemplificado na figura 11.

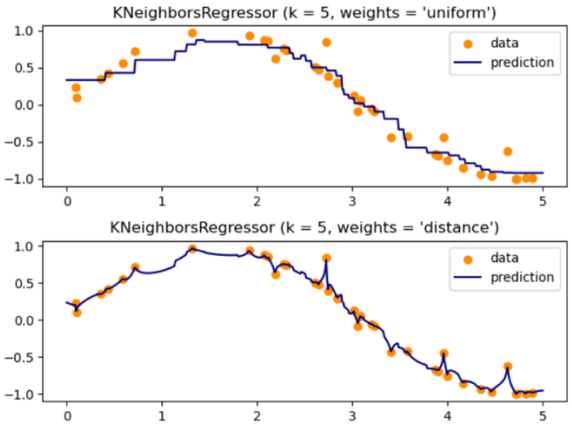

Figura 11 – Diferença entre peso uniforme e peso com base em distância para KNeighborsRegressor [28]

2.3.2.6 Bagging Regressor (BGR)

BGR é um *ensemble* que cria várias instâncias de um regressor base para tentar obter uma previsão melhor. Por defeito os regressores base usados são *Decision Trees* [29].
Este método utiliza *bootstrap* com reposição para criar diferentes subconjuntos de dados que posteriormente são usados para treinar os regressores base.

Por fim, é calculada a média dos resultados de cada modelo para formar a previsão final. Tal permite que cada modelo base capture diferentes padrões nos dados, aumentando o desempenho do modelo final [29].

2.4 Métodos de Avaliação

A avaliação está dependente do tipo de previsões a serem realizadas. Por norma, os estudos relacionados com previsão de STFs tentam prever a tendência ou a volatilidade do preço [8].

Por norma, a previsão de STFs seria um problema de regressão, mas é possível ser um problema de classificação. Caso haja um foco maior, não na previsão do valor, mas sim na previsão da tendência do movimento, este problema pode passar rapidamente a um problema de classificação binária, onde o objetivo é prever a tendência do preço, ou seja, se vai aumentar ou diminuir.

No contexto desta dissertação, o valor do preço será a variável a prever, pelo que, é um problema de regressão.

Para problemas de regressão, no contexto de STFs, costumam ser usadas as seguintes métricas [8]:

- **Mean Squared Error (MSE)** – Representa a distância dos pontos à regressão aplicada.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \quad (12)$$

Onde Y_i representa os valores observados e $\hat{Y}_i$ representa os valores previstos.

- ***Mean Absolute Error (MAE)*** – Representa a diferença absoluta média entre o valor real e o valor previsto.

$$MAE = \frac{\sum_{i=1}^{n}|e_i|}{n} \quad (13)$$

Onde $|e_i| = |Y_i - \hat{Y}_i|$ e por sua vez Y_i representa os valores observados e $\hat{Y}_i$ representa os valores previstos.

- ***Mean Absolute Percentage Error (MAPE)*** – É semelhante a MAE, mas representa a diferença percentual média entre o valor real e o valor previsto.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{Y_i - \hat{Y}_i}{Y_i}\right| \quad (14)$$

Onde Y_i representa os valores observados e $\hat{Y}_i$ representa os valores previstos.

- ***Root Mean Squared Error (RMSE)*** – Representa o desvio padrão do erro da previsão.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{n}} \quad (15)$$

Onde Y_i representa os valores observados, $\hat{Y}_i$ representa os valores previstos e n é o número de dados.

Nota: Existem também as versões normalizadas de MSE (NMSE), e RMSE (NRSME).

3 Dados

Neste capítulo serão apresentadas e analisadas as séries temporais usadas. Começar-se-á por referir de onde foram obtidas as séries, será feita uma descrição de cada uma e por fim, falar-se-á das formatações aplicadas.

3.1 Origem

Os quatro conjuntos de dados usados para treinar e testar os modelos têm duas origens diferentes. Os dois conjuntos de dados em formato de CSV foram obtidos do Banco da Reserva Federal de St. Louis [30]. Os outros conjuntos de dados foram obtidos recorrendo à biblioteca `yfinance`[5].

Ambas as implementações podem ser distinguidas nos ficheiros criados. Ficheiros com o sufixo "_csv" usam os dados do Banco da Reserva Federal de St. Louis, importados de um CSV. Em contrapartida, ficheiros com o sufixo "_yfinance" usam dados obtidos recorrendo à biblioteca `yfinance`.

3.2 Análise Exploratória de Séries Temporais

3.2.1 Série CPIAUCSL

Este conjunto de dados representa o índice de preços ao consumidor para todos os consumidores urbanos, por outras palavras, é um índice de preços de um conjunto de bens e serviços pagos por consumidores urbanos. A variação do índice pode representar várias coisas,

[5] Biblioteca open-source de Python que usa as APIs públicas do Yahoo para obter informação e dados históricos de séries temporais financeiras.

como a inflação, deflação, mudanças nos hábitos de compra, etc. O conjunto de dados tem uma frequência mensal e tem 927 entradas com valores desde 01/01/1947 até 01/03/2024 [31] e a sua representação gráfica encontra-se na figura 12.

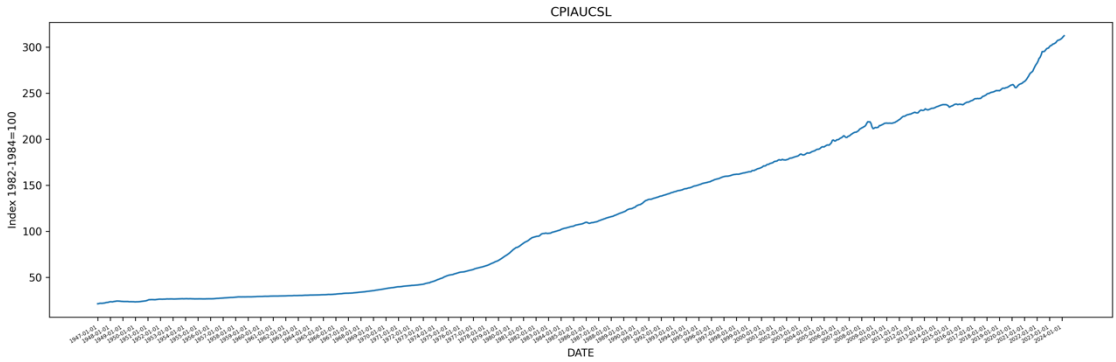

Figura 12 – Série CPIAUCSL

Na tabela 1 são apresentadas as principais estatísticas descritivas da série.

Tabela 1 – Descrição da série CPIAUCSL

count	mean	std	min	25%	50%	75%	max
927	119,65	85,40	21,48	32,32	107,90	191,65	312,23

Do histograma da figura 13, depreende-se a não normalidade da série.

Da decomposição da figura 13 conclui-se que não existe uma componente sazonal forte.

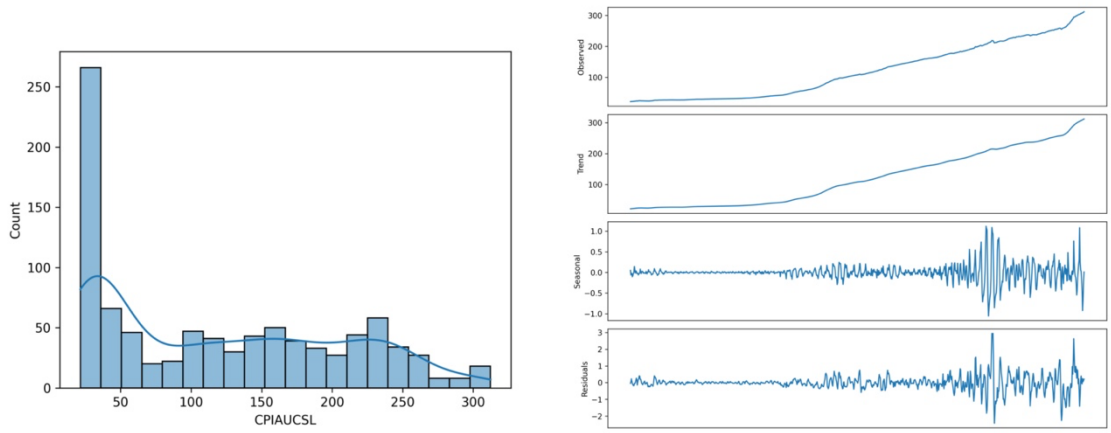

Figura 13 – Histograma(esquerda) e decomposição (direita) da série CPIAUCSL

Este conjunto de dados foi selecionado, uma vez que, já está ajustado para a sazonalidade (como demonstrado na figura 13), permitindo assim, realizar uma comparação entre os diferentes algoritmos, ignorando o impacto da sazonalidade.

A figura 13 demostra a decomposição da série temporal CPIAUCSL, onde é possível observar as três componentes diferentes: tendência, sazonalidade e resíduos. Estes gráficos permitem ao utilizador analisar e decidir se existe sazonalidade significativa, o que pode impactar os modelos a serem usados.

Tendo em conta o histograma verifica-se a não normalidade da distribuição dos valores. É notória ainda, que a série tem uma tendência crescente (figura 12), sem comportamento sazonal (figura 13), sendo não estacionária, como demonstrado na tabela 5 mais à frente.

3.2.2 Série G17MVSFAUTOS

Este conjunto de dados representa, percentualmente e em média, as oscilações entre a produção de veículos esperada e a obtida. Um valor de 110 significa que foram produzidos mais 10% de veículos do que o esperado.

O conjunto de dados tem uma frequência mensal e tem 341 entradas com valores desde 01/01/1996 até 01/05/2024 [32]. A sua representação gráfica encontra-se na figura 14.

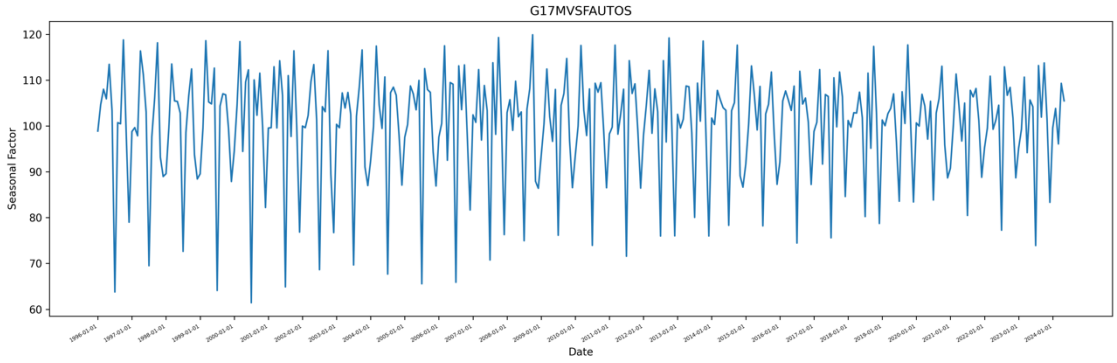

Figura 14 – Série G17MVSFAUTOS

Na tabela 2 são apresentadas as principais estatísticas descritivas da série.

Tabela 2 – Descrição da série G17MVSFAUTOS

count	mean	std	min	25%	50%	75%	max
341	100,15	11,79	61,42	96,77	102,30	107,77	119,92

Do histograma da figura 15, depreende-se a não normalidade da série.

Da decomposição da figura 15 conclui-se que existe uma componente sazonal forte.

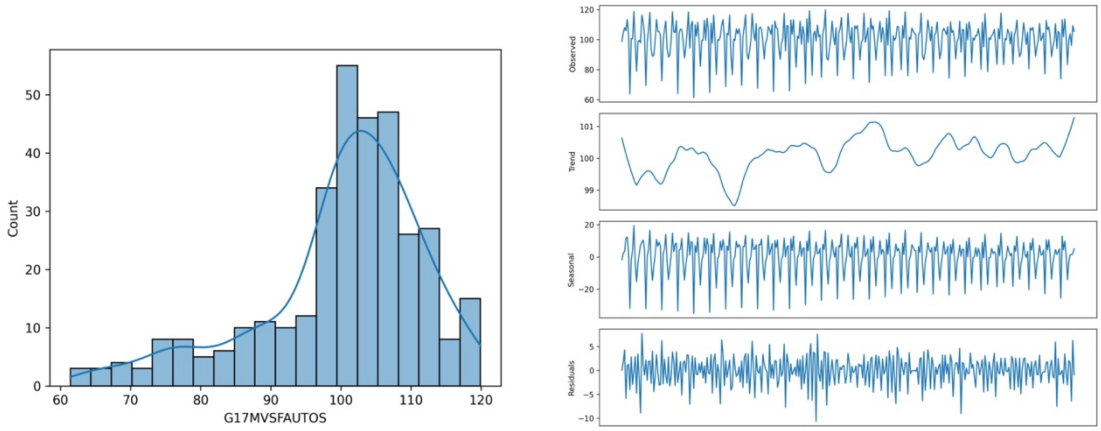

Figura 15 – Histograma(esquerda) e decomposição (direita) da série G17MVSFAUTOS

Este conjunto de dados foi selecionado com o objetivo de estudar o impacto que a sazonalidade tem nos algoritmos aplicados.

De notar, a diferença entre as componentes sazonais das decomposições das séries CPIAUCSL e G17MVSFAUTOS presentes nas figuras 13 e 15, respetivamente. É esperado que a série CPIAUCSL tenha uma componente sazonal franca, algo que é confirmado. Por outro lado, é esperado que a série G17MVSFAUTOS tenha uma forte componente sazonal, algo que também comprovado.

Tendo em conta o histograma, verifica-se a não normalidade da distribuição dos valores. É ainda notório que a série não apresenta uma forte tendência (figura 14), com forte comportamento sazonal (figura 15), sendo estacionária, como demonstrado na tabela 5 mais à frente.

3.2.3 Série Shell

Usando a biblioteca `yfinance` é obtida a série temporal financeira para a empresa Shell. De referir que, esta série tem frequência diária. Efetivamente, a série tem valores a começar 12/12/1980 até ao dia atual. Devido a restrições de recursos e *hardware*, apenas serão considerados valores entre as datas 15/06/2016 e 15/06/2024 e a sua representação gráfica encontra-se na figura 16.

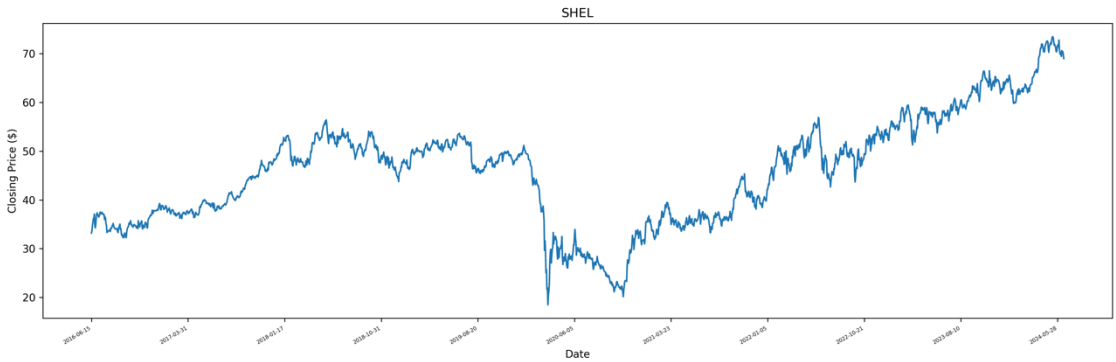

Figura 16 – Série SHEL

Na tabela 3 são apresentadas as principais estatísticas descritivas da série.

Tabela 3 – Descrição da série SHEL

count	mean	std	min	25%	50%	75%	max
2014	45,90	10,90	18,48	37,11	47,55	52,49	73,47

Do histograma da figura 17, depreende-se a não normalidade da série.

Da decomposição da figura 17 conclui-se que existe uma componente sazonal significativa.

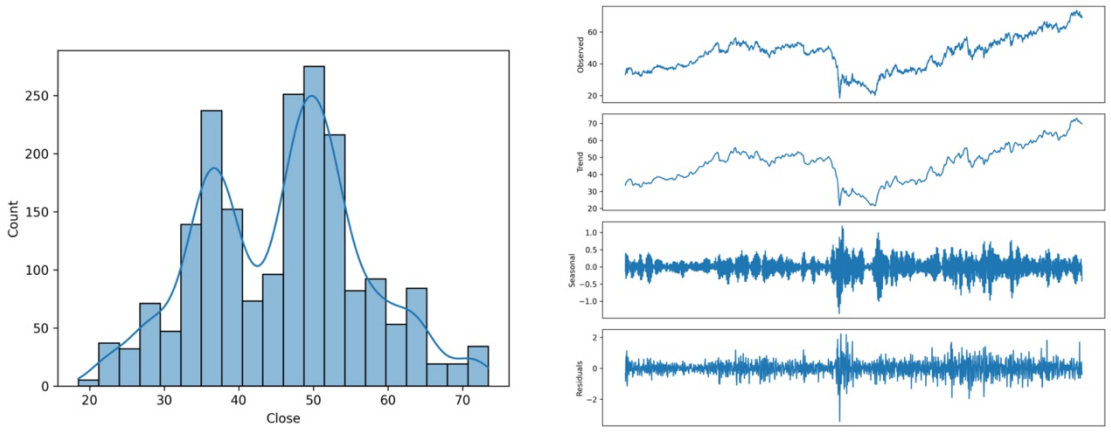

Figura 17 – Histograma(esquerda) e decomposição (direita) da série SHEL

De notar que, os pedidos de histórico da série realizados à biblioteca retornar um *DataFrame* composto pelo índice (a data do dia) e por mais 7 colunas, *Open, Close, High, Low, Volume, Dividends* e *Stock Splits*. Para esta série e para a seguinte, a coluna de interesse é a *Close*, que representa o preço da ação no fim do dia.

Tendo em conta o histograma verifica-se a não normalidade da distribuição dos valores. Para além disso, é visível uma tendência crescente (figura 16), com comportamento sazonal (figura 17), sendo não estacionária, como demonstrado na tabela 5 mais à frente.

3.2.4 Série Lloyds Banking Group

Usando a biblioteca `yfinance` é obtida a série temporal financeira para a empresa Lloyds Banking Group. De referir que, esta série tem frequência diária. Para este estudo apenas serão considerados valores entre as datas 15/06/2016 e 15/06/2024 e a sua representação gráfica encontra-se na figura 18.

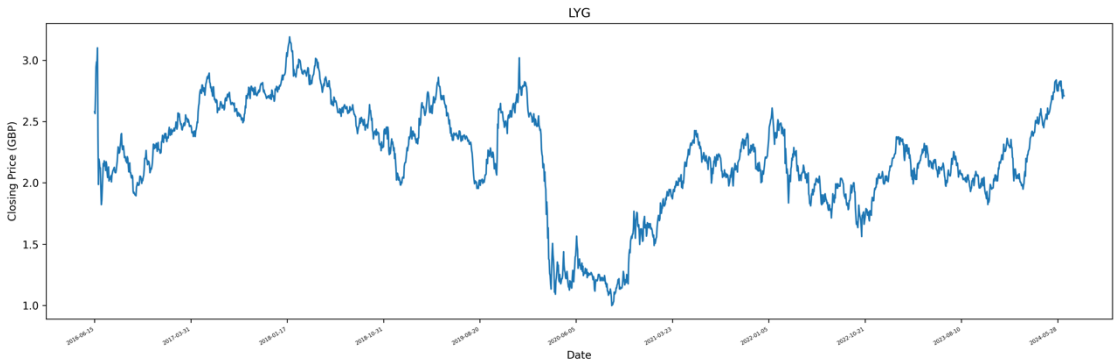

Figura 18 – Série LYG

Na tabela 4 são apresentadas as principais estatísticas descritivas da série.

Tabela 4 – Descrição da série LYG

count	mean	std	min	25%	50%	75%	max
2014	2,21	0,44	0,99	2,01	2,23	2,53	3,19

Do histograma da figura 19, depreende-se a não normalidade da série.

Da decomposição da figura 19 conclui-se que existe uma componente sazonal significativa.

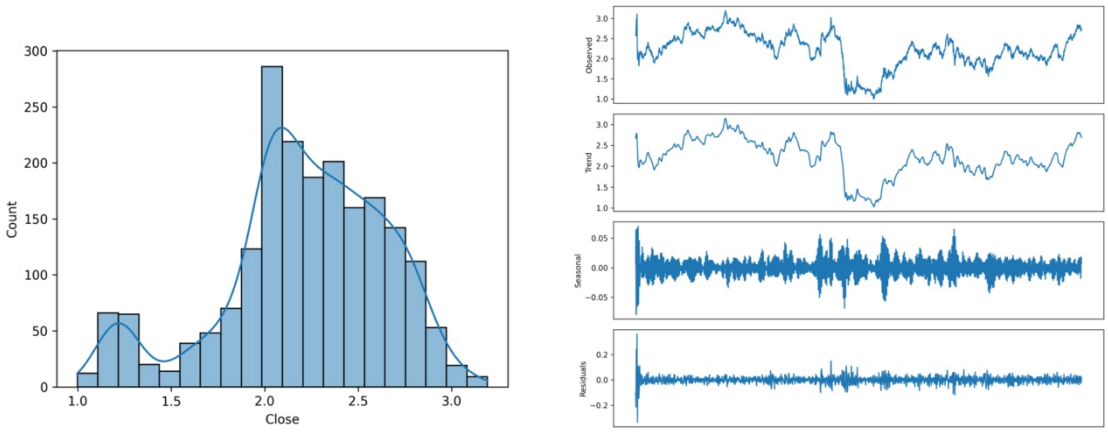

Figura 19 – Histograma(esquerda) e decomposição (direita) da série LYG

Tendo em conta o histograma verifica-se a não normalidade da distribuição dos valores. Observa-se ainda que, a série não apresenta uma forte tendência (figura 18), com comportamento sazonal (figura 19), sendo não estacionária, como demonstrado na tabela 5 mais à frente.

3.3 Preparação dos Dados

Para a normalização será usada a técnica de MinMax, onde todos os valores serão convertidos para o seu equivalente numa escala de 0 (zero) a 1 (um), com 1 a corresponder ao maior valor presente no conjunto de treino.

A aplicação de normalização ajuda a evitar problemas no futuro (e muitas vezes até se torna essencial para o bom funcionamento dos algoritmos), no sentido em que assegura estabilidade e evita que um previsor tenha desproporcionalmente mais influencia devido à sua escala.

Importante referir que, para os modelos de GARCH, os dados não serão normalizados, uma vez que, estes tipos de modelos funcionam de uma maneira um pouco diferente dos outros, pois não tentam prever valores absolutos da variável objetivo, mas sim, a volatilidade da mesma. Aplicar uma normalização faz com que o modelo tenha dificuldade a captar componentes e caraterísticas importantes da série.

4 Implementação

Nesta fase, serão discutidas as decisões de implementação. Serão implementados métodos estatísticos, métodos de aprendizagem automática e métodos híbridos.

De referir que o código da implementação está disponível aqui.

4.1 Modelos Estatísticos

4.1.1 Metodologia de Implementação Computacional dos Modelos ARIMA/SARIMA

Do ponto de vista deste estudo, é vantajosa a implementação de ambos os modelos, pois permite observar e analisar a diferença que a componente da sazonalidade tem nos resultados.

Importante referir que, o método de SARIMA segue os mesmos passos que o método ARIMA segue, mas acrescenta, a estes, detalhes relacionados com a sazonalidade.

Para aplicar os algoritmos de ARIMA e SARIMA é necessário realizar alguns testes aos dados a serem usados. Para o algoritmo ARIMA e SARIMA o processo base deve ser o seguinte:

1. ADF (*Augmented Dickey-Fuller*) – O teste ADF é usado para comprovar se os dados são estacionários ou não. Caso os dados não sejam estacionários será necessário diferenciar os dados. Como já referido, uma das vantagens de ARIMA comparado com ARMA é que consegue lidar com dados não estacionários, desde que receba o parâmetro "d" correto. Este parâmetro "d" representa o número de diferenciações aplicadas até os dados se tornarem estacionários. Para o modelo SARIMA, o mesmo é

aplicado, mas é ainda necessário descobrir o valor de "D" (o número de diferenciações à componente sazonal). Este teste permite, então, obter o valor correto de "d" e "D".

2. ACF (*Autocorrelation Function*) – O teste ACF permite saber se a série temporal em análise é uma *Random Walk* ou não, revelando assim, como os seus valores se correlacionam.

3. p, q, P e Q – Para aplicar o algoritmo ARIMA é necessário facultar como parâmetros, entre outros, o valor de "p" e "q". Como já referido, o valor de "p" representa o número de parâmetros AR a usar, enquanto que o valor "q" representa o número de parâmetros MA a usar. Estes valores tendem a variar dependendo dos dados. Para descobrir quais os melhores valores de "p" e "q" será criado uma função que irá testar várias combinações de valores e decidir qual a melhor. Para o modelo de SARIMA, o mesmo acontece, mas com a adição dos parâmetros "P" e "Q".

4. Testar resíduos QQ plot – Após descobertos os valores dos parâmetros a facultar ao algoritmo, e após a criação do modelo, é necessário testar se este é vaiável. Para tal realiza-se testes aos resíduos obtidos usando o teste QQ plot. QQ plot é uma ferramenta gráfica que permite concluir se os resíduos são aproximadamente normais ou não. É esperado que os resíduos tenham uma distribuição aproximadamente normal, caso tal não aconteça, significa que existe alguma relação nos dados que não foi captada corretamente pelo modelo.

5. Teste Ljung Box – É necessário realizar ainda mais um teste, o teste de Ljung Box, para confirmar se os resíduos não estão correlacionados. Correlação entre os resíduos pode significar que existe alguma relação nos dados que não foi captada corretamente pelo modelo.

6. Previsões – Após estes testes é possível então realizar previsões com o modelo criado.

7. Avaliação de Previsões – Por fim é necessário recolher métricas relativas às previsões para ter dados para realizar comparações entre diferentes métodos preditivos.

Com o objetivo de manter a organização ao longo da implementação, foi criada a classe `TimeSeries` onde são guardados, não só a série temporal em si, mas também vários outros dados sobre a mesma, como por exemplo, o nível de diferenciação a aplicar e os conjuntos de teste e treino.

```
1.  class TimeSeries:
2.      def __init__(self, name, history):
3.          self.name = name
4.          self.history = history
5.          self.differenceLevel = 0
6.          self.sazDifferenceLevel = 0
7.          self.p = 0
8.          self.q = 0
9.          self.sazP = 0
10.         self.sazQ = 0
```

```
11.         self.differenceData = history
12.         self.sazDifferenceData = history
13.         self.residuals = None
14.         self.train = None
15.         self.test = None
16.
17.     def incrementDifferenceLevel(self):
18.         self.differenceLevel += 1
19.
20.     def decrementDifferenceLevel(self):
21.         self.differenceLevel -= 1
22.
23.     def resetDifferenceLevel(self):
24.         self.differenceLevel = 0
25.
26.     def incrementSazDifferenceLevel(self):
27.         self.sazDifferenceLevel += 1
28.
29.     def decrementSazDifferenceLevel(self):
30.         self.sazDifferenceLevel -= 1
31.
32.     def resetSazDifferenceLevel(self):
33.         self.sazDifferenceLevel = 0
34.
35.     def setP(self, p):
36.         self.p = p
37.
38.     def setQ(self, q):
39.         self.q = q
40.
41.     def setSazP(self, p):
42.         self.sazP = p
43.
44.     def setSazQ(self, q):
45.         self.sazQ = q
46.
47.     def setResiduals(self, residuals):
48.         self.residuals = residuals
49.
50.     def setDifferenceData(self, diffData):
51.         self.differenceData = diffData
52.
53.     def setSazDifferenceData(self, diffData):
54.         self.sazDifferenceData = diffData
55.
56.     def setTrainSet(self, train):
57.         self.train = train
58.
59.     def setTestSet(self, test):
60.         self.test = test
```

Excerto de Código 1 – Classe TimeSeries para modelos ARMA

O uso desta classe facilita, como já referido, a organização, uma vez que, a informação estará disponível em apenas uma instância desta mesma classe. Além disso, não se corre o risco de reescrever não propositadamente uma variável que contém informação sensível.

De referir que, a versão da implementação presente no excerto de código 1 (excluindo as linhas a vermelho) foi desenhada para o algoritmo ARIMA. Para esta ser adaptada para o algoritmo SARIMA, basta adicionar as linhas a vermelho.

Para aplicar o ADF, será usada a função `adfuller()`. Usando esta função é possível concluir se a série é estacionária ou se precisa de ser diferenciada.

```
1.  stayInLoop = True
2.  diff = data.history
3.  while stayInLoop :
4.      adfResult = adfuller(diff)
5.      print(f'ADF Statistic: {adfResult[0]}')
6.      print(f'p-value: {adfResult[1]}')
7.      print('----------------------')
8.      if adfResult[0] <= ADF_MAX_ALLOWED_VALUE and adfResult[1] < ADF_P_VALUE :
9.          stayInLoop = False
10.     else :
11.         diff = data.differenceData.diff().dropna()
12.         data.setDifferenceData(diff)
13.         data.incrementDifferenceLevel()
```

Excerto de Código 2 – Cálculo do ADF e diferenciação

O excerto de código 2 demonstra como foi implementado o teste ADF em combinação com a aplicação de diferenciação, com o objetivo de determinar quantas vezes é necessário diferenciar para a série se tornar estacionária.

Os resultados obtidos para cada série temporal estão representados na tabela 5.

Tabela 5 – Resultados ADF por série

	ADF	**P-value**	**Número de Diferenciações**
CPIAUCSL	-3,5866	0,0060	1
G17MVSFAUTOS	-3,3997	0,0110	0
SHEL	-44,9288	0,0000	1
LYG	-16,3115	3,1926e-29	1

Da tabela 5 é possível concluir que após uma diferenciação todas as séries passam a ser estacionárias, exceto a série G17MVSFAUTOS, que já era estacionária.

Para aplicar o teste ACF basta apenas chamar a função `plot_acf()` da biblioteca `statsmodels`. Como referido anteriormente, aplicar o teste ACF permite saber se a série em questão é uma *Random Walk* ou não. As figuras 20 e 21 apresentam os resultados da aplicação do `plot_acf()` nas séries.

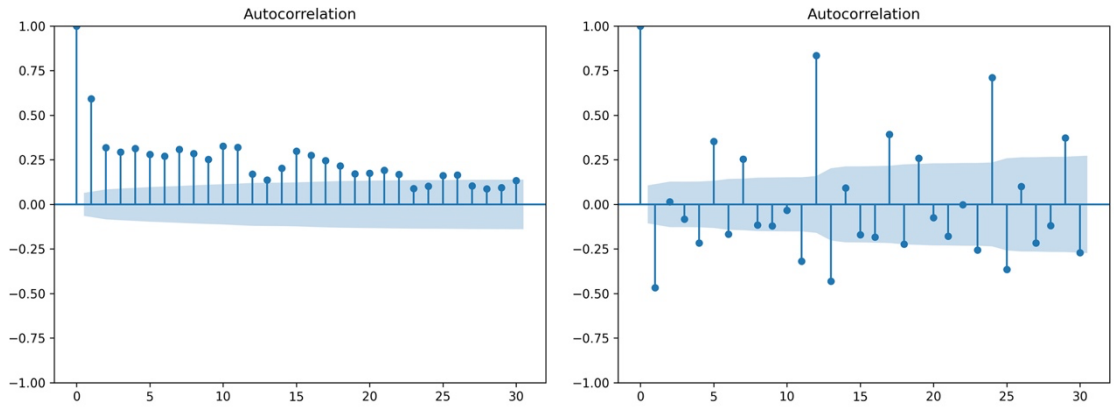

Figura 20 – ACF da série CPIAUCSL (esquerda) e da série G17MVSFAUTOS (direita)

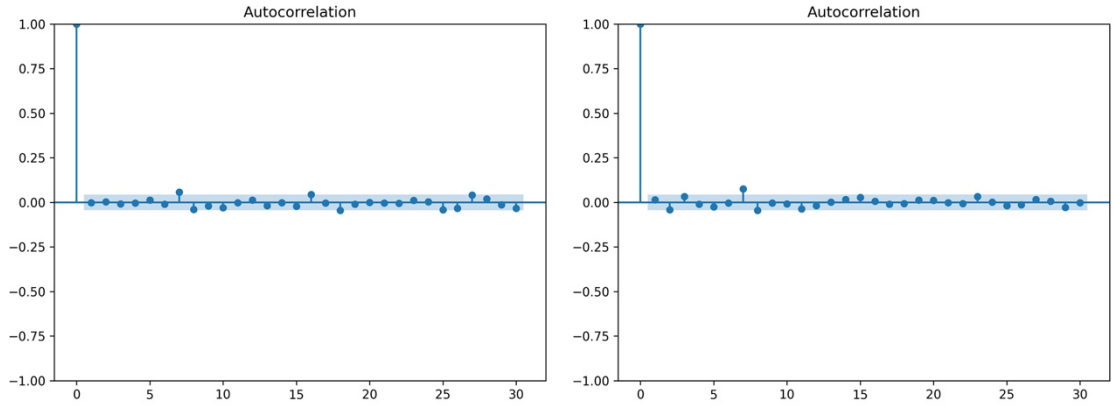

Figura 21 – ACF da série SHEL (esquerda) e da série LYG (direita)

Como é possível observar, na figura 20 existe autocorrelação entre as variáveis, significando que as séries não são *random walks*.

Por outro lado, a figura 21 mostra exatamente o contrário, as séries em causa (SHEL e LYG) são *random walks*, o que significa que não faz sentido aplicar métodos estatísticos a estas séries.

Em seguida será necessário dividir os dados no segmento de treino e no segmento de teste, e ao mesmo tempo normalizá-los. Para este estudo, foram usados 90% dos dados para treino e 10% para teste. De referir que, para a normalização será usado o `MinMaxScaler` da biblioteca `sklearn`. Um detalhe importante é que os dados de treino devem ser normalizados usando o método `fit_transform()` enquanto que os dados de teste devem usar o `transform()`.

É agora necessário descobrir o melhor "p" e "q" para o determinado conjunto de dados. Para resolver este problema, será usada a função `optimizeARIMA` [6]. Esta função, ao receber um conjunto de valores possíveis para "p" e "q", irá criar um modelo para cada combinação de valores e calculará o valor de AIC (*Akaike Information Criteria*) para cada modelo. Por fim, a função irá retornar uma lista das combinações de valores e os devidos valores de AIC, permitindo escolher o "p" e "q" que se ajustam melhor aos dados. De referir que, quanto menor o AIC melhor.

Após o algoritmo parametrizado todos cria-se então o modelo final:

```
1.  model = SARIMAX(data.train, order=(data.p,data.differenceLevel,data.q),
    simple_differencing=False)
```

Excerto de Código 3 – Criação do modelo ARIMA

O modelo é criado usando o valor de "p" (guardado em `data.p`) e "q" (guardado em `data.q`) obtidos anteriormente, assim como o nível de diferenciação também já calculado. De referir que, o modelo é treinado apenas com os dados de treino, como se pode confirmar no excerto de código 3 (`data.train`).

Depois do modelo ter sido criado e treinado com os dados, é necessário realizar testes aos valores residuais, para tal é possível obter o QQ Plot usando o método `plot_diagnostics` (excerto de código 4).

```
1.  model_fit.plot_diagnostics(figsize=(10,8))
2.  data.setResiduals(model_fit.resid)
```

Excerto de Código 4 – Gerar Diagnósticos

A implementação do excerto de código 4 apresenta os seguintes resultados:

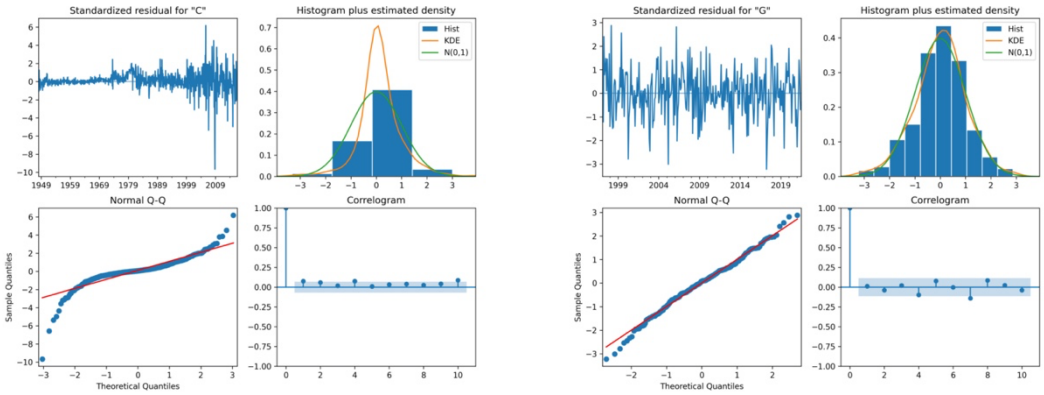

Figura 22 – Resultados do QQ_plot da série CPIAUCSL (esquerda) e da série G17MVSFAUTOS (direita)

É possível notar na figura 22, olhando para o terceiro gráfico (contando da esquerda para a direita, de cima para baixo) de ambas as séries, que os resíduos se aproximam a uma distribuição normal, uma vez que os pontos se encontrar relativamente alinhados com a reta vermelha.

É necessário ainda confirmar se os resíduos estão correlacionados ou não, para tal é usado o teste de Ljung Box, recorrendo à função `acorr_ljungbox`, como demostra o excerto de código 5 passando como parâmetro os resíduos e o número de *lags* a ter em consideração.

```
1.  statAndPValue = acorr_ljungbox(data.residuals, np.arange(1, 11, 1))
```

Excerto de Código 5 – Teste LJungBox

Tabela 6 – Resultados do teste LJungBox

P-values	CPIAUCSL	G17MVSFAUTOS
1º *Lag*	0,621087	1,702372e-36
2º *Lag*	0,613918	1,261505e-59
3º *Lag*	0,794422	2,615385e-74
4º *Lag*	0,590611	1,877861e-84
5º *Lag*	0,718097	8,205531e-94
6º *Lag*	0,795929	3,623712e-101
7º *Lag*	0,841781	3,299026e-107
8º *Lag*	0,858021	5,021209e-112
9º *Lag*	0,889143	3,224036e-113
10º *Lag*	0,872960	1,242865e-112

Como a tabela 6 demonstra, os resíduos da série CPIAUCSL são não correlacionados, pois o P-value é maior que 0,05, ou seja, não se pode rejeitar a hipótese nula que diz que não há autocorrelação significativa nos resíduos da série, enquanto que os da série G17MVSFAUTOS são correlacionados, o que implica, voltar ao passo da descoberta do valor de “p”, “q”, “P” e “Q” e encontrar valores melhores.

Os valores dos parâmetros obtidos para CPIAUCSL e G17MVSFAUTOS estão representados na tabela 7.

Tabela 7 – Híper Parâmetros usados

Parâmetro	CPIAUCSL	G17MVSFAUTOS
p	10	2
d	1	0
q	7	1
P	Não aplicável	2
D	Não aplicável	1
Q	Não aplicável	2

Após todos estes testes as previsões podem ser feitas. Com o objetivo de obter um maior número de previsões e para dar uso a todo o conjunto de dados de teste, será implementada a função `rollingForecast` [6].

```
1.  def rollingForecast(df: pd.DataFrame, trainLen: int, testLen: int, horizon:
    int, order: Tuple) -> list:
2.
3.      totalLen = trainLen + testLen
4.
5.      pred = df[-testLen:].copy()
6.      pred[:] = None
```

```
7.      helper = []
8.
9.      for i in range(trainLen-horizon, totalLen-horizon, 1):
10.         model = SARIMAX(df[:i+1], order=order, simple_differencing=False)
11.         res = model.fit(disp=False)
12.         predictions = res.forecast(steps=horizon)
13.         predictionValue = predictions.iloc[-1]
14.
15.         pred.loc[df.iloc[i+horizon].name, df.columns[0]]=predictionValue
16.
17.     return pred
```

Excerto de Código 6 – *Rolling Forecast* ARIMA

Na implementação presente no excerto de código 6 existem os parâmetros `df`, referente ao *DataFrame* que contém todos os dados, `trainLen`, que contém o tamanho do conjunto de treino, `testLen`, que contém o tamanho do conjunto de teste, `horizon`, que de forma semelhante ao que acontece na janela deslizante implementada para modelos de aprendizagem automática, representa o número de *steps* a prever no futuro e `order` que é um tuplo (com três entradas, a primeira é o valor de "p", a segunda é o valor de "d" e por fim o valor de "q").

A figura 23 ajuda a explicar a lógica por de trás do `rollingForecast()`.

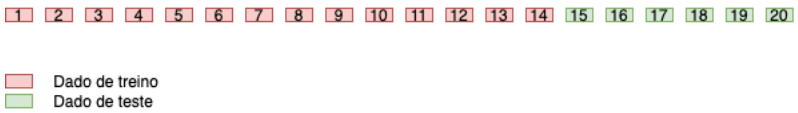

Figura 23 – Representação Visual de dados de treino e teste

A lógica desta solução assemelhas à lógica da janela deslizante usada normalmente em aprendizagem automática:

- Iteração 1 – Valores de 1 a 13 seriam usados para treino e seria realizada uma previsão para o valor 15.

- Iteração 2 – Valores de 1 a 14 seriam usados para treino e seria realizada uma previsão para o valor 16.

Isto possibilita obter previsões para todos os valores de teste, além de, posteriormente, tornar a comparação de resultados entre métodos estatísticos e métodos de aprendizagem automática mais justa, uma vez que a aquisição de previsões é mais semelhante.

Por fim é necessário ainda obter as métricas de classificação de desempenho do modelo.

As métricas a ser usadas serão MAE, MSE e MAPE. Para recolher tais métricas foi criada a função `modelEvaluation` que ao receber o valor dos *DataFrames*, um com os valores de teste e outro com as previsões, calculará as métricas, como demostrado no excerto de código 7.

```
1.  def modelEvaluation (nameModel: str, yTest: pd.DataFrame, yPred: pd.DataFrame)
    -> pd.Series:
2.
3.      mae = mean_absolute_error(yTest, yPred)
4.      mse = mean_squared_error(yTest, yPred)
5.      mape = mean_absolute_percentage_error(yTest, yPred)
6.
7.      res = pd.Series({'Model': nameModel, 'mae' : mae, 'mse': mse, 'mape' :
    mape})
8.
9.      return res
```

Excerto de Código 7 – Função de obtenção de métricas de avaliação

Importante referir que, os valores fornecidos à função presente no excerto de código 7 serão apenas as previsões do valor do horizonte e não dos valores intermédios, por outras palavras, e recorrendo novamente à Figura 23 (assumindo um horizonte de 2), ao prever o valor do dado 16 será necessário prever antes o dado 15 também. A previsão deste valor 15 não será tida em conta para as métricas finais, uma vez que o erro presente na sua previsão refletir-se-á no erro da previsão do ponto 16.

Para as séries sazonais deve ser usado o SARIMA, para tal será necessário ainda realizar o teste de ADF e diferenciação para componente sazonal da série, para obter o valor de "D" (SARIMA(p,d,q)(P,D,Q)s). A implementação seria em tudo igual ao do excerto de código 2. A única diferença seria na linha 11 que passaria a ser a do excerto de código 8.

```
1.  diff = data.sazDifferenceData.diff(12).dropna()
```

Excerto de Código 8 – Diferenciação Sazonal

A diferença da linha 11 do excerto de código 8 para a do excerto de código 2 é o parâmetro do método `diff()`. Este parâmetro representa o período sazonal (por exemplo, se a sazonalidade é anual e os dados da série temporal são obtidos mensalmente, então o valor será 12, pois existem 12 meses num ano).

É necessário ainda, como já referido, encontrar o melhor valor para as componentes "P" e "Q" do algoritmo de SARIMA, o que é realizado aplicando a mesma lógica usada para encontrar os melhores valores de "p" e "q" [6].

Por fim, após obtidos os melhores valores de "P" e "Q" é preciso facultá-los à classe SARIMAX. O excerto de código 9 substituiria o excerto de código 3 e linha 10 do excerto de código 6.

```
1.  model = SARIMAX(data.train, order=(data.p,data.differenceLevel,data.q),
    seasonalOrder=(data.sazP,data.sazDifferenceLevel,data.sazQ,s),
    simple_differencing=False)
```

Excerto de Código 9 – Criação do modelo SARIMA

4.1.2 GARCH

GARCH funciona de maneira diferente quando comparado com modelos da classe ARMA.

GARCH não é usado para prever os valores absolutos do preço da ação, mas prevê sim a volatilidade da série.

Inicialmente, obtém-se os dados da mesma maneira que para os modelos já explicados, mas é necessário converter os dados em retornos. Retornos, na área financeira, representam a mudança percentual no valor de determinado bem ao longo de um período e são dados pela fórmula:

$$Returns = \frac{P_{t1} - P_{t2}}{P_{t2}} \quad (16)$$

Onde P_{t1} é o preço do bem a determinado tempo $t1$ e P_{t2} é o preço do bem a determinado tempo $t2$.

Para obter estes retornos diários foi usada a função `pct_change()`, como demonstra o excerto de código 10. Os valores são multiplicados por 100 para o modelo GARCH trabalhar com valores de uma escala maior, uma vez que este tem dificuldade a lidar com valores perto do zero.

```
1.  hist = hist.pct_change().dropna()*100
```

Excerto de Código 10 – Cálculo dos Retornos

Após obtidos os retornos, os dados são então divididos em conjunto de treino e teste recorrendo à função presente no excerto de código 11.

```
1.  def splitData(df: pd.DataFrame, valSplit: float = 0.15, testSplit: float =
    0.15) -> Tuple[pd.DataFrame, pd.DataFrame, pd.DataFrame]:
2.      dfCopy = df.copy()
3.      valInd = int(len(dfCopy) * valSplit)
4.      testInd = int(len(dfCopy) * testSplit)
5.      train = dfCopy[:-(valInd+testInd)]
6.      val = dfCopy[-(valInd+testInd) : -testInd]
7.      test = dfCopy[-testInd:]
8.
9.      return train, val, test
```

Excerto de Código 11 – Função de Divisão de Dados

De forma semelhante ao que acontece com os modelos da classe ARMA, é necessário também encontrar os melhores valores para “p” e “q”, para tal foi contruída a função presente no excerto de código 12 [6].

```
1.  def optimizeGARCH(train: pd.DataFrame, orderList: list) -> pd.DataFrame:
2.
3.      results = []
4.
5.      for order in orderList:
6.          model = arch_model(train, p=order[0], q=order[1]).fit(disp=False)
```

```
7.
8.          aic = model.aic
9.          results.append([order, aic])
10.
11.     resultDf = pd.DataFrame(results)
12.     resultDf.columns = ['(p,q)', 'AIC']
13.     resultDf = resultDf.sort_values(by='AIC',
    ascending=True).reset_index(drop=True)
14.
15.     return resultDf
```

Excerto de Código 12 – Função de Otimização GARCH

São obtidos então os valores ideais de "p" e "q" (com base na métrica AIC). Estes valores serão agora usados para criar um modelo GARCH e realizar as previsões para os valores de teste, recorrendo à implementação do `rollingForecast` presente no excerto de código 13.

```
1.  def rollingForecast(df: pd.DataFrame, trainLen: int, testLen: int, horizon:
    int, order: Tuple[int, int, int]) -> pd.DataFrame:
2.
3.      totalLen = trainLen + testLen
4.
5.      pred = pd.DataFrame()
6.
7.      for i in range(trainLen-horizon, totalLen-horizon, 1):
8.          model = arch_model(
9.                          df[:i+1],
10.                         p=order[0],
11.                         q=order[1]).fit(disp=False)
12.
13.         predictions = model.forecast(horizon=horizon)
14.         predictionValue = np.sqrt(predictions.variance.values[-1,:][-1])
15.
16.         pred.loc[df.iloc[i+horizon].name, df.columns[0]] = predictionValue
17.
18.     return pred
```

Excerto de Código 13 – *Rolling Forecast* GARCH

Ao realizar uma previsão com um horizonte igual a 3, o modelo GARCH não prevê apenas o valor da variância para o *step* 3, mas prevê também para o *step* 2 e 1. Por causa disto, a linha 16 vai buscar apenas o último valor, ou seja, a previsão do *step* pretendido. Em seguida é ainda necessário calcular a raiz quadrada dessa variância, para obter a volatilidade, uma vez que:

$$Volatility = \sqrt{Variance} \quad (17)$$

Uma vez que GARCH vai prever a volatilidade, é necessário calcular a volatilidade real para posteriormente ser possível comparar valores de teste com os previstos. Para tal, foi acrescentada a seguinte linha presente no excerto de código 14.

```
1. testRollingVolatility=np.sqrt(data.history.rolling(window=HORIZON+1).std()).sh
   ift(-HORIZON).dropna()
```

Excerto de Código 14 – Cálculo da Volatilidade dos Dados

Como é possível observar que é usada a função `rolling()` em parceria com `std()`, com o objetivo de obter o desvio padrão. Após obtido o desvio padrão, é calculado a raiz quadrada deste para obter a volatilidade. Posteriormente é aplicada a função `shift()` para realinhar os valores com os respetivos índices.

Por fim, para obter as métricas de avaliação é usada a função `modelEvaluation()` presente no excerto de código 7.

4.2 Modelos de Aprendizagem Automática

Existem diferentes algoritmos de aprendizagem automática que podem ser aplicados, pelo que, a implementação foi pensada para ser genérica o suficiente para que a eventual troca de algoritmo cause o mínimo de impacto possível.

Mais uma vez, e com o objetivo de manter as implementações de diferentes métodos o mais semelhante possível, foi criada uma classe `TimeSeries`, desta vez com menos parâmetros, pois não há a necessidade de realizar testes e aplicar diferenciação aos dados.

```
1.  class TimeSeries:
2.      def __init__(self, name, history):
3.          self.name = name
4.          self.history = history
5.          self.train = None
6.          self.test = None
7.
8.      def setTrainSet(self, train):
9.          self.train = train
10.
11.     def setTestSet(self, test):
12.         self.test = test
```

Excerto de Código 15 – Classe TimeSeries para AA

O excerto de código 15 demostra as alterações realizadas à classe `TimeSeries` para suportar implementações de aprendizagem automática.

Após a obtenção dos dados será necessário dividi-los no conjunto de treino e teste e posteriormente normalizá-los. A normalização, será feita recorrendo à classe `MinMaxScaler` da biblioteca `sklearn`, como usado na implementação de métodos estatísticos.

No contexto de aprendizagem automática, os dados são "ingeridos" pelos modelos de maneira diferente, o que força o uso de *data windowing*. Este processo consiste na seleção de subconjuntos de dados nos quais são definidos os previsores e os valores a prever [6].

O nome da função responsável por este *data windowing* é `slideWindow` e a sua implementação pode ser observada no excerto de código 16.

```
1.  def slideWindow(df: pd.DataFrame, window: int, horizon: int) -> pd.DataFrame:
2.
3.      d = df.values
4.      X, y = [], []
5.      idx = df.index[window:]
6.      for end in range(window, len(df), 1):
7.          start = end - window
8.          out = end + horizon
9.          X.append(d[start:end].reshape(-1))
10.         y.append(d[end:out].ravel())
11.
12.     cols_x = [f'x{i}' for i in range(1, window+1)]
13.     cols_y = [f'y{i}' for i in range(1, horizon+1)]
14.
15.     df_xs = pd.DataFrame(X, index=idx, columns=cols_x)
16.     df_y = pd.DataFrame(y, index=idx, columns=cols_y)
17.
18.     return pd.concat([df_xs, df_y], axis=1).dropna()
```

Excerto de Código 16 – Implementação SlideWindow

A função recebe um *DataFrame*, o número de previsores a usar e o número de *steps* a prever. Olhando para a linha 6, é possível reparar na declaração da variável `end`. Esta variável representa o índice do previsor final, enquanto que nas linhas subsequentes estão presentes a variável `start` e `out`, que representam, o índice do primeiro previsor e o índice do último *step* a prever, respetivamente. Seguidamente, nas linhas 9 e 10, usando uma combinação destes valores são obtidos os valores dos previsores e dos valores a prever. Após repetido o processo para todos os valores, são criados os nomes das colunas a usar (linhas 12 e 13). As linhas 15 e 16 são responsáveis pela criação de *DataFrames* com os valores e devidos índices. Por fim, na linha 18 são concatenados os dois *DataFrames* previamente criados, obtendo assim, apenas um que possui os previsores e valores a prever. Importante notar, o uso do método `.dropna()`. Este método é importante, uma vez que, é responsável por remover todas as entradas para as quais não foi possível obter um conjunto inteiro de previsores e valores a prever. Tal acontece, pois quando o ciclo da linha 6 se encontra na sua última iteração não conseguirá ir buscar os valores a prever (horizonte), uma vez que eles não existem.

Após aplicada a função `slideWindow()` para o conjunto de treino e teste, como demostra o excerto de código 17, os valores obtidos são divididos em previsores e valores a prever linhas 11, 12, 14 e 15.

```
1.  WINDOW = 12
2.  HORIZON = 3
3.
4.  trainShift = slideWindow(data.train, WINDOW, HORIZON)
5.  testShift = slideWindow(data.test, WINDOW, HORIZON)
6.
7.  featureStartString = ['x']
8.  features = [col for col in trainShift.columns if any(col.startswith(s) for s
    in featureStartString)]
```

```
9.  label = trainShift.columns.difference(features)
10.
11. xTrain = trainShift[features]
12. yTrain = trainShift[label]
13.
14. xTest =  testShift[features]
15. yTest =  testShift[label]
16.
17. print('xTrain:',xTrain.shape,'yTrain', yTrain.shape, 'xTest:',xTest.shape,
    'yTest:', yTest.shape)
```

Excerto de Código 17 – Separação de previsores e valores a prever de treino e teste (AA) (Normal)

De notar que, as linhas 7 a 9 foram acrescentadas para tornar a distinção das colunas de previsores e valores a prever mais fácil. A implementação mais simples e direta seria assumir que todas as colunas com x<número> no seu nome seriam previsores e as com y<número> seriam valores a prever, mas ao implementar como o excerto de código 17, recorrendo à lista da linha 7 para declarar o sufixo das colunas dos previsores e à linha 8 para os filtrar, facilita bastante a adição de previsores no futuro.

Este uso da função `slideWindow` está correto para realizar previsões, mas para manter a comparação de resultados o mais justa possível, foi também implementado um `rollingForecast` para os métodos de aprendizagem automática. Embora estes consigam, por norma, prever melhor mais *steps* maiores. Isto implica que o excerto de código 17 tenha de ser ajustado para acomodar o *rolling forecast*, resultando no excerto de código 18.

```
1.  lastTrainIndex = data.train.iloc[-1].name
2.  organizedData = slideWindow(pd.concat([data.train, data.test]), WINDOW,
    HORIZON)
3.
4.  featureStartString= ['x']
5.  features = [col for col in organizedData.columns if any(col.startswith(s) for
    s in featureStartString)]
6.  label = organizedData.columns.difference(features)
7.
8.  organizedDataTrain = organizedData.loc[:lastTrainIndex]
9.  organizedDataTest = organizedData[len(organizedDataTrain):]
10.
11. xTrain = organizedDataTrain[features]
12. yTrain = organizedDataTrain[label]
13.
14. xTest =  organizedDataTest[features]
15. yTest =  organizedDataTest[label]
16.
17. print('xTrain:',xTrain.shape,'yTrain', yTrain.shape, 'xTest:', xTest.shape,
    'yTest:', yTest.shape)
```

Excerto de Código 18 – Separação de previsores e valores a prever de treino e teste (AA) (RollingForecast)

Como é possível observar, o conjunto de dados de treino e teste foram novamente juntos num só conjunto. Tal permite a criação das janelas deslizantes. Posteriormente, os conjuntos são novamente separados, mas esta implementação permite que o *rolling forecast* seja aplicado

sobre os dados, uma vez que, ele poderá ter controlo de quando e onde separa os conjuntos de treino e teste, sem se perder valores. A implementação do *rolling forecast* está presente no excerto de código 19.

```
1.  def   rollingForecast(xTrain:   pd.DataFrame,   yTrain:   pd.DataFrame,   xTest:
    pd.DataFrame, horizon: int, model: Any ) -> list:
2.      totalLen = len(xTrain) + len(xTest)
3.      preds = []
4.
5.      for i in range(len(xTrain)-horizon, totalLen-horizon, 1):
6.
7.          xTrainRolling = pd.concat([xTrain, xTest], axis=0)[:i]
8.          yTrainRolling = pd.concat([yTrain, yTest], axis=0)[:i]
9.          xTestRolling          =          pd.concat([xTrain,          xTest],
    axis=0)[i+horizon:i+horizon+1]
10.
11.         model.fit(xTrainRolling, yTrainRolling)
12.         yPred = model.predict(xTestRolling)
13.         preds.append(yPred[-1])
14.
15.     return preds
```

Excerto de Código 19 – *Rolling Forecast* AA

De referir que, como é possível observar na linha 13, apenas o último valor da previsão será tido em conta. Quando é realizada uma previsão para um horizonte superior a 1, os modelos têm também de prever os steps intermédios até chegarem ao *step* final. Essas previsões intermédias serão descartadas, uma vez que, qualquer erro ao realiza-las, estará também refletido no resultado final.

```
1.  def getBestModel(model, paramGrid, xTrain, yTrain, splits=5):
2.
3.      tscv = TimeSeriesSplit(n_splits=splits)
4.      gridSearch = GridSearchCV(estimator=model, param_grid=paramGrid, cv=tscv,
    n_jobs=-1, scoring='neg_mean_squared_error')
5.      gridSearch.fit(xTrain, yTrain)
6.
7.      bestModel = gridSearch.best_estimator_
8.
9.      return bestModel
```

Excerto de Código 20 – Função cujo objetivo é descobrir os melhores parâmetros para cada modelo

O excerto de código 20 mostra a implementação da função `getBestModel()`, que ao receber um algoritmo de aprendizagem automática (`model`), um conjunto de parâmetros para testar (`paramGrid`), o conjunto de dados de treino (`xTrain` e `yTrain`) e o número de partições a usar (`splits`), recorre a uma `GridSearchCV` para encontrar a combinação de parâmetros que dará melhores resultados. De referir que, o parâmetro `scoring` da `GridSearchCV` irá ter o valor de "neg_mean_squared_error". Isto define o método de avaliação dos conjuntos de parâmetros. Neste caso, os parâmetros que fizerem com que o modelo tenha o mínimo de MSE serão os melhores.

```
dtrParams = { 'max_depth': [10, 50, 100] }
knrParams = { 'n_neighbors': [1, 2, 3, 5, 7], 'weights': ['uniform',
'weights']}
bgrParams = { 'n_estimators': [10, 50, 100, 200, 300, 500] }
rfrParams = {
    'n_estimators': [50, 100, 200, 500],
    'max_depth': [3, 5, 7, 10, 20]
    }
xgbParams = {
    'n_estimators': [100, 200, 500, 700],
    'max_depth': [3, 5, 7, 10, 20],
    'learning_rate': [0.1, 0.2, 0.5],
    'subsample': [0.7, 0.8, 1.0],
    'colsample_bytree': [0.7, 0.8, 1.0]
    }
svrParams = {
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
    'C': [0.1, 1, 10, 50, 100],
    'epsilon': [0.01, 0.1, 1.0]
    }
```

Excerto de Código 21 – Híper parâmetros a serem usado na função getBestModel

O excerto de código 21 demostra os parâmetros a serem usados pela função `getBestModel` para cada modelo.

Como referido anteriormente, houve uma tentativa de tornar esta implementação o mais genérica possível, com o objetivo de ser relativamente simples e rápido a aplicação de diferentes algoritmos de aprendizagem automática. Tendo isso em mente, o excerto de código 22 foi implementado.

De acrescentar que, no excerto de código 22 é possível observar como são usados os parâmetros previamente definidos.

```
algs = []
algs.append(('dtr', DecisionTreeRegressor(), dtrParams))
algs.append(('knr', KNeighborsRegressor(), knrParams))
algs.append(('bgr', BaggingRegressor(n_jobs=-1), bgrParams))
algs.append(('rfr', RandomForestRegressor(n_jobs=-1), rfrParams))
algs.append(('xgb', XGBRegressor(n_jobs=-1), xgbParams))
algs.append(('svr', SVR(), svrParams))

listPreds = []
for name, alg, params in algs:

    print(f'Forecasting Using: {name}')
    predColumn = max(yTest.columns, key=lambda col: int(col[1:]))
    bestModel = getBestModel(alg, params, xTrain, yTrain[predColumn])
    print(bestModel)
    predictions = rollingForecast(xTrain, yTrain[predColumn], xTest,
yTest[predColumn], HORIZON, bestModel)

    results = yTest[[predColumn]].copy()
    results.columns = [y]
    results[name] = predictions
    results[[y, name]] = scaler.inverse_transform(results[[y, name]])

```

```
23.     listPreds.append((name, results))
```

Excerto de Código 22 – Realização de revisões usando vários métodos de AA

Como é possível ver da linha 1 a 7, foi criada uma lista para conter todos os modelos de aprendizagem automática a aplicar. Estes modelos serão aplicados um a um dentro do ciclo `for` presente na linha 10. Dentro deste ciclo, para cada modelo, será descoberto o melhor conjunto de híper parâmetros (linha 14). Seguidamente, os modelos serão treinados, e as previsões serão realizadas (linha 16).

O `rollingForecast()` vai retornar uma lista com os valores de previsão devidamente ordenados de acordo com os índices presentes na variável `yTest`. Uma vez que é apenas devolvida uma lista com os valores previstos, as linhas 18 a 20 encarregam-se de criar um *DataFrame* com os índices e respetivos valores de teste e previsões. Para tal, é obtido o nome da coluna com os valores de teste de interesse (linha 13). É fornecida uma função *lambda* à função `max()`, responsável por ir buscar os números presentes nos nomes das colunas. Será então retornado o nome que possua o maior número no seu nome (por exemplo, entre y1, y2 e y3 seria retornado y3). Assim sendo, é garantido que valores de teste intermédios não são considerados para as métricas de avaliação.

A linha 21 encarrega-se de inverter a normalização.

Por fim, os valores são guardados numa lista, em formato de tuplo (acompanhado pelo nome do modelo), que contém as previsões de todos os modelos (linha 23) para futuramente comparações serem realizadas.

Em termos de medição de métricas de avaliação, a função a usar é a mesma que está presente no excerto de código 7.

4.3 Modelos Híbridos

4.3.1 Aprendizagem Automática e GARCH

O Modelo Híbrido criado usa GARCH e Aprendizagem Automática para obter as suas previsões. Após obtidos os dados, o modelo GARCH tem a responsabilidade de prever a volatilidade do preço da ação. Posteriormente, os valores previstos de volatilidade, são fornecidos ao modelo de aprendizagem automática, que, em junção com valores anteriores do preço irá realizar previsões.

Após obtidos os dados (valores do preço e retornos) e criados o conjunto de treino e teste, estes são facultados ao modelo GARCH. O modelo GARCH vai agir da mesma maneira como foi explicado no subcapítulo “4.1.2 - GARCH”. No fim, o método irá retornar um tuplo onde o

primeiro valor é um *DataFrame* com as previsões para o conjunto de teste e o segundo é também um *DataFrame* que contém os valores das métricas de avaliação.

Depois do método GARCH concluir e retornar os valores das previsões da volatilidade, é calculada a volatilidade real para os dados de treino a serem usados pelo método de aprendizagem automática. Este é um passo importante, uma vez que, o modelo de aprendizagem automática vai ser treinado em valores reais de volatilidade. Por sua vez as previsões (realizadas por GARCH) serão usadas como um novo previsor no conjunto de teste.

Todos os seguintes passos são iguais aos previamente vistos para os modelos de aprendizagem automática, existindo apenas uma diferença no que toca à criação da janela deslizante. Esta fica um pouco mais complexa, pois há a necessidade de introduzir um novo previsor, como o excerto de código 23 demonstra.

```
1.  lastTrainIndex = data.train.iloc[-1].name
2.  organizedData = slideWindow(pd.concat([data.train, data.test]), WINDOW,
    HORIZON)
3.
4.  organizedData[newColumnName] = volatility
5.
6.  featureStartString= ['x','garchPrediction']
7.  features = [col for col in organizedData.columns if any(col.startswith(s) for
    s in featureStartString)]
8.  label = organizedData.columns.difference(features)
9.
10. organizedDataTrain = organizedData.loc[:lastTrainIndex]
11. organizedDataTest = organizedData[len(organizedDataTrain):]
12.
13. xTrain = organizedDataTrain[features]
14. yTrain = organizedDataTrain[label]
15.
16. xTest =  organizedDataTest[features]
17. yTest =  organizedDataTest[label]
18.
19. print('xTrain:',xTrain.shape,'yTrain', yTrain.shape, 'xTest:', xTest.shape,
    'yTest:', yTest.shape)
```

Excerto de Código 23 – Separação de previsores e valores a prever de treino e teste (Híbrido)

O excerto de código 23, quando comparado com o excerto de código 18, apresenta diferenças, nomeadamente as linhas 4 e 6. Na linha 4 é adicionada uma coluna no *DataFrame*, contendo os valores reais de volatilidade (para treino) e previsões do GARCH (para teste) calculados previamente. Por fim, é necessário adicionar o nome da coluna na lista da linha 6 para estes novos valores serem considerados como um novo previsor.

Como referido antes, o resto da implementação é semelhante à demonstrada no capítulo “4.2 – Modelos de Aprendizagem Automática”.

4.3.2 SARIMA e GARCH

Foi implementado um outro modelo híbrido recorrendo a modelos SARIMA e GARCH.

Inicialmente, os dados são facultados ao modelo SARIMA para treinar o modelo, em seguida os resíduos do modelo SARIMA são usados para treinar o modelo. No fim, ambos os modelos realizam as suas previsões e os resultados de ambos são combinados para obter uma previsão do preço da ação.

Este modelo, inicialmente segue os mesmos passos que o modelo SARIMA já discutido acima, pelo que não há diferenças que justifiquem novos excertos de código.

Após obtidos os dados, são aplicados o teste ADF e a diferenciação (normal e sazonal) para obter o valor de "d" e "D". Seguidamente, o conjunto de dados é dividido num conjunto de treino (90%) e teste (10%). É definida, então, uma função de otimização (semelhante à representada no excerto de código 12, mas com as devidas alterações para suportar um modelo SARIMA) para cada modelo. Seguidamente, recorrendo a essa função são descobertos os melhores valores de "p", "q", "P" e "Q" para SARIMA e "p" e "q" para GARCH.

Importante referir que, para o modelo SARIMA esta função de otimização usa os dados de treino para obter os valores, mas para o modelo GARCH, em vez do conjunto de treino, usam-se os valores dos resíduos da função SARIMA.

Após obtidos os melhores valores a usar em ambos os modelos, é realizado um *rolling forecast* como demonstrado no excerto de código 24.

```
1.  def rollingForecast(df: pd.DataFrame, trainLen: int, testLen: int, horizon:
    int, sarimaOrder: Tuple[int, int, int], sarimaSazOrder: Tuple[int, int, int,
    int], garchOrder: Tuple[int, int]) -> pd.DataFrame:
2.
3.      totalLen = trainLen + testLen
4.
5.      pred = pd.DataFrame()
6.
7.      for i in range(trainLen-horizon, totalLen-horizon, 1):
8.          sarimaModel = SARIMAX(df[:i+1], order=sarimaOrder,
    seasonal_order=sarimaSazOrder)
9.          sarimaFit = sarimaModel.fit(disp=False)
10.         sarimaResid = sarimaFit.resid
11.
12.         garchModel = arch_model(sarimaResid, p=garchOrder[0], q=garchOrder[1])
13.         garchFit = garchModel.fit(disp='off')
14.
15.         sarimaResults = sarimaFit.get_forecast(steps=horizon)
16.         sarimaResultMean = sarimaResults.predicted_mean.iloc[-1]
17.
18.         garchResults = garchFit.forecast(horizon=horizon)
19.         garchResultVolatility = np.sqrt(garchResults.variance.values[-1, :][-
    1])
20.
21.         predictionValue = sarimaResultMean + garchResultVolatility
22.
23.         pred.loc[df.iloc[i+horizon].name, df.columns[0]] = predictionValue
```

```
24.
25.     return pred
```

Excerto de Código 24 – *Rolling Forecast* SARIMA-GARCH

O excerto de código 24 mostra o processo para realizar previsões. Na linha 8 é possível ver que o modelo de SARIMA é treinado com os dados facultados à função, enquanto que na linha 12, em contraste, o modelo GARCH é treinado com os valores residuais do modelo SARIMA. Tal é realizado para permitir ao modelo GARCH tentar captar alguma relação que tenha escapado ao modelo SARIMA.

Após o treino, são realizadas as previsões (linha 15 a 19) pelos diferentes modelos. Por fim as previsões juntam-se para formar uma previsão apenas (linha 21). Esta previsão é então guardada numa lista.

No que toca à aquisição de métricas de avaliação, esta é realizada recorrendo à função presente no excerto de código 7.

5 Resultados

Neste capítulo serão apresentados e discutidos os resultados.

Importante referir que, por uma questão de restrições temporais, de hardware e recursos, apenas valores entre 15-06-2016 e 15-06-2024 foram analisados para as séries temporais financeiras “SHEL” e “LYG”.

De acrescentar que, para todos os modelos em questão, 90% dos dados foram usados para treino e 10% para teste, em contraste com os habituais 80% e 20%, respetivamente. Tal deve-se a restrições de recursos, uma vez que, quanto maior o conjunto de dados de teste, mais demorado e mais recursos são necessários para realizar o *rolling forecast*. Além disso, é de notar que todos os modelos foram treinados nos mesmos dados de treino e testados nos mesmos dados de teste para que os resultados possam ser comparáveis.

5.1 Série CPIAUCSL

Para modelos ML e ML-GARCH, foi usada uma janela de 12 meses.

Tabela 8 – Resultados CPIAUCSL para horizontes temporais 3, 10 e 20 meses

Horizon		3				10				20			
Models		**MAE**	**MSE**	**MAPE**	**Time**	**MAE**	**MSE**	**MAPE**	**Time**	**MAE**	**MSE**	**MAPE**	**Time**
ARIMA CPIAUCSL		**1,2620**	**2,9934**	**0,0047**	0:04:29.01	**3,5441**	**27,2270**	**0,0129**	0:04:14.22	7,9183	140,6702	0,0284	0:04:17.72
ML CPIAUCSL	DTR (window: 12)	3,7484	21,2121	0,0137	0:00:02.70	9,0962	124,4744	0,0325	0:00:01.13	17,5982	415,3255	0,0620	0:00:01.12
	KNR (window: 12)	3,4903	18,0271	0,0127	0:00:00.67	9,2165	128,7645	0,0329	0:00:00.47	17,5007	410,3501	0,0617	0:00:00.55
	BGR (window: 12)	4,0767	24,8723	0,0149	0:00:17.91	9,5646	136,6207	0,0342	0:01:22.43	18,0677	435,4899	0,0636	0:00:22.81
	RFR (window: 12)	4,0796	24,4693	0,0149	0:00:59.86	9,5975	137,9060	0,0343	0:00:57.15	18,0863	436,0981	0,0637	0:00:53.22
	XGB (window: 12)	3,8977	23,2093	0,0142	0:03:24.91	9,1050	123,4675	0,0325	0:01:55.15	17,2946	406,5246	0,0609	0:01:44.38
	SVR (window: 12)	1,7289	5,5006	0,0063	0:02:08.05	3,7939	31,7484	0,0136	0:04:10.20	**7,2294**	**122,6054**	**0,0251**	0:08:38.69

Segundo os resultados obtidos, o modelo ARIMA apresenta os melhores resultados para um horizonte de 3 e 10 meses, sendo que ML (SVR) de janela 12 também apresenta bons resultados para um horizonte de 3 meses, mas quando comparados os tempos de execução, é facilmente compreensível que ARIMA é o método superior. Para um horizonte mais extenso, como por exemplo de 20 meses, o modelo ML (SVR) com janela de 12 já demonstra resultados superiores que ARIMA.

No que toca a métodos de aprendizagem automática, é possível notar que, entre estes métodos, há um claro vencedor, o modelo com base em SVRs, que apresenta melhores resultados quando comparado com os outros modelos de aprendizagem automática.

5.2 Série G17MVSFAUTOS

Para este conjunto de dados, para os modelos SARIMA e SARIMA-GARCH foi usada uma sazonalidade de 12, uma vez que os dados são mensais e um ano tem 12 meses.

De acrescentar que, para modelos ML e ML-GARCH, foi usada uma janela de 12 meses.

Tabela 9 – Resultados G17MVSFAUTOS para horizontes temporais 3, 10 e 20 meses

Horizon		3				10				20			
Models		**MAE**	**MSE**	**MAPE**	**Time**	**MAE**	**MSE**	**MAPE**	**Time**	**MAE**	**MSE**	**MAPE**	**Time**
SARIMA G17MVSFAUTOS (s: 12)		2,6395	13,8861	0,0267	0:14:35.86	2,5627	14,4496	0,0263	0:14:32.61	2,8705	14,2914	0,0295	0:14:17.33
ML G17MVSFAUTOS	DTR (window: 12)	4,1591	28,5997	0,0412	0:00:01.39	3,9490	24,1572	0,0410	0:00:00.22	4,1750	28,3006	0,0409	0:00:00.22
	KNR (window: 12)	2,4477	10,0296	0,0248	0:00:00.22	2,1831	9,5173	0,0222	0:00:00.17	**2,1600**	**8,3082**	**0,0220**	0:00:00.22
	BGR (window: 12)	2,5179	13,2873	0,0256	0:00:29.02	2,6437	15,1946	0,0269	0:00:13.24	2,9037	14,6471	0,0293	0:00:14.15
	RFR (window: 12)	2,5342	13,9430	0,0258	0:00:22.48	2,5917	15,0088	0,0263	0:00:35.44	2,9532	15,5270	0,0301	0:00:24.66
	XGB (window: 12)	2,5216	13,1138	0,0254	0:01:36.42	2,5142	13,2836	0,0258	0:00:56.05	2,9446	16,3691	0,0299	0:00:56.49
	SVR (window: 12)	**1,8680**	**8,8149**	**0,0192**	0:00:17.51	**2,0000**	**8,5841**	**0,0205**	0:00:12.24	3,2842	19,8239	0,0347	0:00:19.95
SARIMA GARCH G17MVSFAUTOS (s: 12)		3,8563	23,7013	0,0399	0:06:30.66	3,9890	25,3856	0,0413	0:06:23.34	4,2220	28,3846	0,0438	0:06:26.11

De notar que, o modelo SARIMA-GARCH aparenta ter tido problemas a modelar as relações deste conjunto de dados, uma vez que, apresenta dos piores desempenhos de todos os modelos, apenas superado pelo algoritmo DTR de aprendizagem automática.

ML (SVR) apresenta novamente um desempenho relativamente bom, principalmente para o horizonte 3 e 10.

Para o horizonte 20 o modelo ML (KNR) de janela 12 acaba por ser o melhor modelo. É interessante de notar que para um horizonte de 20 dias, o modelo previamente com melhores resultados (ML(SVR) de janela 12) acaba por ter um desempenho relativamente fraco.

5.3 Série Shell

Para modelos ML foi usada uma janela de 5 e 10 dias.

De notar que, poderia ter sido usada uma janela maior, mas seriam necessários mais recursos computacionais.

Tabela 10 – Resultados SHEL para horizontes temporais 3, 10 e 20 dias

Horizon		3				10				20			
Models		MAE	MSE	MAPE	Time	MAE	MSE	MAPE	Time	MAE	MSE	MAPE	Time
ML SHEL	DTR (window: 5)	1,6987	4,4474	0,0260	0:00:07.98	2,3762	9,3249	0,0360	0:00:04.54	3,6343	21,5685	0,0539	0:00:04.36
	KNR (window: 5)	1,4287	3,2168	0,0218	0:00:02.52	2,1773	7,8515	0,0328	0:00:02.53	3,5692	20,5053	0,0531	0:00:02.72
	BGR (window: 5)	1,4519	3,2574	0,0222	0:02:21.66	2,0693	7,3807	0,0312	0:02:18.09	3,3466	17,7953	0,0497	0:06:00.00
	RFR (window: 5)	1,3926	3,1798	0,0213	0:01:09.52	3,6229	20,9316	0,0538	0:00:59.62	4,0114	24,9356	0,0591	0:01:22.76
	XGB (window: 5)	1,6056	4,1730	0,0245	0:05:13.45	2,0674	7,5382	0,0312	0:03:31.93	3,3208	18,9399	0,0491	0:02:56.86
	SVR (window: 5)	0,9774	1,4871	**0,0149**	0:24:51.49	**1,6785**	**4,6231**	**0,0257**	0:25:11.83	**2,2683**	**8,2740**	**0,0341**	0:24:51.22
	DTR (window: 10)	1,5953	3,9893	0,0245	0:00:07.16	2,1356	7,5172	0,0324	0:00:04.54	3,4955	21,2987	0,0519	0:00:06.36
	KNR (window: 10)	1,5133	3,5678	0,0231	0:00:01.93	2,0640	8,2151	0,0310	0:00:01.46	3,2135	17,9431	0,0476	0:00:00.82
	BGR (window: 10)	1,3892	3,0305	0,0212	0:02:50.30	2,0648	7,6119	0,0312	0:00:20.91	3,1396	16,5667	0,0466	0:02:43.10
	RFR (window: 10)	3,0953	15,9265	0,0462	0:00:46.22	3,6203	21,0039	0,0537	0:01:44.04	3,9897	24,6532	0,0588	0:00:39.51
	XGB (window: 10)	1,4406	3,2257	0,0220	0:03:36.02	1,9983	6,7703	0,0303	0:02:58.90	2,9226	14,8103	0,0433	0:05:44.71
	SVR (window: 10)	**0,9765**	**1,4836**	**0,0149**	0:36:59.76	1,6878	4,6532	0,0258	0:39:43.34	2,2882	8,2916	0,0343	0:34:46.71
ML GARCH SHEL	DTR (window: 5)	1,9526	6,0452	0,0300	0:00:25.57	2,1786	8,8686	0,0331	0:00:26.20	4,4126	32,5969	0,0660	0:00:22.41
	KNR (window: 5)	1,5159	3,5745	0,0232	0:00:21.33	2,1834	7,7075	0,0329	0:00:21.86	3,5880	20,8658	0,0534	0:00:20.83
	BGR (window: 5)	1,5102	3,7508	0,0231	0:00:40.84	2,0678	7,5606	0,0313	0:04:29.62	3,9922	26,2625	0,0596	0:01:41.21
	RFR (window: 5)	1,5035	3,9548	0,0230	0:02:21.30	2,2177	8,6644	0,0336	0:02:18.62	4,0086	26,5760	0,0598	0:02:46.73
	XGB (window: 5)	1,6508	4,7133	0,0252	0:04:48.02	2,3659	9,8004	0,0359	0:12:26.25	3,8574	25,3592	0,0574	0:03:39.48
	SVR (window: 5)	1,1614	2,0622	0,0177	0:08:42.77	1,7712	4,9807	0,0270	0:01:27.74	2,4529	9,9427	0,0366	0:01:19.52

Como a figura 21 mostra, esta série é uma *Random Walk*, o que invalida o uso de métodos estatísticos para realizar previsões, sobrando assim, os métodos de aprendizagem automática.

Para um horizonte de 3 dias, o modelo ML (SVR) com uma janela de 10 apresenta desempenho superior, embora, ainda para este horizonte, ML (SVR) com uma janela de 5 demonstre também resultados favoráveis, mas para os horizontes 10 e 20 o modelo ML (SVR) com uma janela de 5 apresenta melhores resultados.

De realçar que, para um horizonte de 3 dias, não existe uma diferença muito grande de valores entre os modelos ML (SVR) de janela 5 e de janela 10, tendo ambos até, o mesmo valor de MAPE.

Comparando resultados de ML com ML-GARCH é notável que o modelo de GARCH teve dificuldades a capturar corretamente a volatilidade da série temporal.

5.4 Série Lloyds Banking Group

Para modelos ML foi usada uma janela de 5 e 10 dias.

De notar que, poderia ter sido usada uma janela maior, mas seriam necessários mais recursos computacionais.

Tabela 11 – Resultados LYG para horizontes temporais 3, 10 e 20 dias

Horizon		3				10				20			
Models		MAE	MSE	MAPE	Time	MAE	MSE	MAPE	Time	MAE	MSE	MAPE	Time
ML LYG	DTR (window: 5)	0,0619	0,0065	0,0277	0:00:04.86	0,1143	0,0195	0,0508	0:00:04.34	0,1804	0,0441	0,0800	0:00:03.88
	KNR (window: 5)	0,0521	0,0040	0,0233	0:00:02.65	0,1016	0,0146	0,0450	0:00:02.48	**0,1704**	0,0407	0,0744	0:00:02.78
	BGR (window: 5)	0,0532	0,0044	0,0239	0:02:26.00	0,1043	0,0155	0,0464	0:01:04.64	0,1770	0,0415	0,0776	0:01:12.87
	RFR (window: 5)	0,0590	0,0054	0,0269	0:00:53.64	**0,1001**	**0,0140**	**0,0446**	0:01:35.46	0,1709	**0,0375**	0,0749	0:02:30.99
	XGB (window: 5)	0,0568	0,0048	0,0256	0:02:14.63	0,1072	0,0172	0,0480	0:03:07.58	0,1886	0,0479	0,0828	0:02:47.61
	SVR (window: 5)	**0,0506**	**0,0039**	**0,0227**	0:26:41.96	0,1033	0,0153	0,0458	0:23:31.38	0,1724	0,0404	0,0747	0:17:01.02
	DTR (window: 10)	0,0580	0,0053	0,0263	0:00:05.26	0,1203	0,0215	0,0537	0:00:05.20	0,1810	0,0458	0,0807	0:00:04.12
	KNR (window: 10)	0,0630	0,0056	0,0281	0:00:01.40	0,1058	0,0161	0,0473	0:00:01.21	0,1757	0,0407	0,0785	0:00:01.79
	BGR (window: 10)	0,0528	0,0042	0,0239	0:07:08.00	0,1040	0,0148	0,0464	0:01:39.96	0,1794	0,0418	0,0797	0:00:52.08
	RFR (window: 10)	0,0608	0,0057	0,0276	0:00:48.64	0,1044	0,0148	0,0463	0:00:51.92	0,1768	0,0397	0,0782	0:01:15.97
	XGB (window: 10)	0,0536	0,0045	0,0243	0:03:02.06	0,1049	0,0160	0,0468	0:02:20.18	0,2015	0,0566	0,0888	0:08:07.80
	SVR (window: 10)	0,0509	**0,0039**	0,0228	0:42:54.27	0,1015	0,0149	0,0451	0:39:22.30	0,1710	0,0403	**0,0742**	0:22:01.52
ML GARCH LYG	DTR (window: 5)	0,0974	0,0153	0,0441	0:00:50.57	0,2265	0,0775	0,0997	0:00:51.61	0,3490	0,2236	0,1489	0:00:50.46
	KNR (window: 5)	0,0720	0,0079	0,0321	0:00:48.28	0,1803	0,0532	0,0782	0:00:47.94	0,3736	0,2357	0,1602	0:00:49.34
	BGR (window: 5)	0,0680	0,0069	0,0305	0:01:08.45	0,1627	0,0407	0,0709	0:01:30.20	0,3324	0,1781	0,1422	0:01:07.09
	RFR (window: 5)	0,0653	0,0065	0,0295	0:01:26.41	0,1591	0,0384	0,0694	0:01:35.01	0,3285	0,1717	0,1408	0:01:36.94
	XGB (window: 5)	0,0629	0,0064	0,0282	0:05:27.81	0,1591	0,0437	0,0692	0:04:09.06	0,3067	0,1544	0,1304	0:04:35.65
	SVR (window: 5)	0,0602	0,0057	0,0268	0:38:24.75	0,1330	0,0246	0,0584	0:17:31.41	0,2513	0,0873	0,1059	0:18:49.69

Como a figura 21 mostra, esta série é uma *Random Walk*, o que invalida o uso de métodos estatísticos para realizar previsões, sobrando assim, os métodos puramente de aprendizagem automática.

Mais uma vez, para um horizonte de 3, o modelo ML (SVR) de janela 5 obteve o melhor resultado, embora o modelo ML (SVR) com janela 10 tenha obtido resultados muito semelhantes. De acrescentar que, para este caso ML (SVR) de janela 5 é a melhor opção tendo em conta o tempo que leva a realizar as previsões, quando comparado com ML (SVR) de janela 10.

Para o horizonte de 10, o modelo ML (RFR) de janela 5 obteve os melhores resultados em todas as métricas (MAE, MSE e MAPE).

Por fim, para o horizonte de 20, os resultados são mais ambíguos uma vez que o melhor valor de MAE é obtido pelo modelo ML (KNR) de janela 5, o melhor valor de MSE é obtido pelo modelo ML (RFR) de janela 5 e o melhor valor de MAPE é obtido pelo modelo ML (SVR) de janela 10.

6 Conclusões

Em suma, os objetivos aos quais esta dissertação se propunha foram concluídos com sucesso, desde a pesquisa da literatura, à comparação de resultados, passando pela implementação de diferentes modelos estatísticos e de aprendizagem automática.

Foram implementados diferentes modelos estatísticos. ARIMA atua de forma satisfatória para séries pequenas e sem presença de sazonalidade, enquanto que SARIMA já obtém um melhor desempenho ao lidar com esta componente de uma série temporal. O modelo GARCH, por sua vez, tenta prever a volatilidade, com base nos retornos diários. Como não faria sentido realizar uma comparação direta entre previsões de valor e previsões de volatilidade desse valor, o modelo GARCH não foi usado individualmente, mas sim em combinação com outros modelos, criando assim modelos híbridos. Estes modelos acabaram por não ter um desempenho muito elevado.

Foram também implementados modelos de aprendizagem automática, usando diferentes algoritmos. Estes tendem a apresentar resultados satisfatórios, principalmente os modelos que usam o algoritmo SVR.

Os métodos estatísticos não puderam ser aplicados às séries SHEL e LYG, uma vez que estas falharam no teste ACF, confirmando que são uma *Random Walk*. Assim, uma das vantagens dos métodos de aprendizagem automática é a não requisição de amplos testes os que os métodos estatísticos requerem.

Por fim, o modelo que tende a ter melhor desempenho para a maioria das situações analisadas é o ML (SVR), embora, ARIMA também apresente bons resultados para horizontes mais curtos, desde que a série temporal cumpra todos os requisitos todos para aplicar este modelo.

6.1 Trabalho Futuro

Existem principalmente três abordagens diferentes a seguir para trabalho futuro:

- Aprendizagem Profunda – A exploração deste tipo de modelos é justificada, uma vez que estes tendem a captar melhor as relações dos dados de séries temporais financeiras e obter previsões melhores.

- Máquinas melhores – Adquirir mais recurso e máquinas melhores para correr os modelos permitirá obter melhores resultados e em tempo mais útil. Além disso, permitirá, também, que os modelos sejam treinados com um conjunto de dados maior.

- Híper Parâmetros – Realizar uma exploração mais a fundo dos híper parâmetros dos métodos de aprendizagem automática

Referências

[1] CFI Team, “Stock Exchange - Definition, Use, Examples, Types.” Accessed: Dec. 28, 2023. [Online]. Available: https://corporatefinanceinstitute.com/resources/equities/stock-exchange/

[2] R. Wirth and J. Hipp, “CRISP-DM: Towards a Standard Process Model for Data Mining”.

[3] Aric LaBarr, *What is Time Series Data*, (Apr. 24, 201904).

[4] T. Zaw, S. S. Kyaw, and A. N. Oo, “ARMA Model for Revenue Prediction,” in *Proceedings of the 11th International Conference on Advances in Information Technology*, New York, NY, USA: ACM, Jul. 2020, pp. 1–6. doi: 10.1145/3406601.3406617.

[5] Aric LaBarr, *What is Time Series Decomposition*, (2019).

[6] Marco Peixeiro, *Time Series Forecasting in Python*. Manning Publications Co., 2022.

[7] D. A. Dickey and W. A. Fuller, “Distribution of the estimators for autoregressive time series with a unit root.,” *J Am Stat Assoc*, pp. 427–431, 1979.

[8] Y. Tang *et al.*, “A survey on machine learning models for financial time series forecasting,” *Neurocomputing*, vol. 512, pp. 363–380, Nov. 2022, doi: 10.1016/j.neucom.2022.09.003.

[9] M. T. Ismail, B. Audu, and M. M. Tumala, “Comparison of forecasting performance between MODWT-GARCH(1,1) and MODWT-EGARCH(1,1) models: Evidence from African stock markets,” *Journal of Finance and Data Science*, vol. 2, no. 4, pp. 254–264, Dec. 2016, doi: 10.1016/J.JFDS.2017.03.001.

[10] H. Tang, “Stock Prices Prediction Based on ARMA Model,” in *2021 International Conference on Computer, Blockchain and Financial Development (CBFD)*, IEEE, Apr. 2021, pp. i–iv. doi: 10.1109/CBFD52659.2021.00046.

[11] S. Ling, K. Zhu, and C. C. Yee, “Diagnostic checking for non-stationary ARMA models with an application to financial data,” *The North American Journal of Economics and Finance*, vol. 26, pp. 624–639, Dec. 2013, doi: 10.1016/j.najef.2013.02.025.

[12] A. H. Bukhari, M. A. Z. Raja, M. Sulaiman, S. Islam, M. Shoaib, and P. Kumam, “Fractional neuro-sequential ARFIMA-LSTM for financial market forecasting,” *IEEE Access*, vol. 8, pp. 71326–71338, 2020, doi: 10.1109/ACCESS.2020.2985763.

[13] Aric LaBarr, *What are Autoregressive (AR) Models*, (2019).

[14] Aric LaBarr, *What are Moving Average (MA) Models*, (2019).

[15] Aric LaBarr, *What are ARIMA Models*, (2019).

[16] C. Narendra Babu and B. Eswara Reddy, “Prediction of selected Indian stock using a partitioning–interpolation based ARIMA–GARCH model,” *Applied Computing and Informatics*, vol. 11, no. 2, pp. 130–143, Jul. 2015, doi: 10.1016/j.aci.2014.09.002.

[17] E. Zivot, “GARCH Models,” 2011, pp. 477–504. doi: 10.1007/978-1-4419-7787-8_18.

[18] R. F. Engle, “Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation,” *Econometrica*, vol. 50, no. 4, p. 987, Jul. 1982, doi: 10.2307/1912773.

[19] Aric LaBarr, *What are ARCH & GARCH Models*, (2019).

[20] R. Ghosh and T. U. Wien, “Learning Outcomes of Classroom Research”, Accessed: Dec. 27, 2023. [Online]. Available: https://www.researchgate.net/publication/358050961

[21] H. Esteves, “Deteção de patologia cardíaca utilizando aprendizagem profunda.”

[22] IBM, “What is Machine Learning? | IBM.” Accessed: Dec. 27, 2023. [Online]. Available: https://www.ibm.com/topics/machine-learning

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

[24] “1.10. Decision Trees — scikit-learn 1.5.0 documentation.” Accessed: Jun. 22, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/tree.html#tree

[25] E. Ferreira Gomes, “Tree-based Methods,” 2023, Accessed: Dec. 28, 2023. [Online]. Available: https://hastie.su.domains/ISLP/ISLP_website.pdf

[26] “RandomForestRegressor — scikit-learn 1.5.0 documentation.” Accessed: Jun. 22, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

[27] “XGBoost Documentation — xgboost 2.1.0 documentation.” Accessed: Jun. 22, 2024. [Online]. Available: https://xgboost.readthedocs.io/en/stable/index.html

[28] “1.6. Nearest Neighbors — scikit-learn 1.5.0 documentation.” Accessed: Jun. 22, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/neighbors.html#regression

[29] “BaggingRegressor — scikit-learn 1.5.0 documentation.” Accessed: Jun. 23, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingRegressor.html

[30] “Federal Reserve Economic Data.” Accessed: Jun. 27, 2024. [Online]. Available: https://fred.stlouisfed.org/

[31] “Consumer Price Index for All Urban Consumers: All Items in U.S. City Average (CPIAUCSL) | FRED | St. Louis Fed.” Accessed: May 26, 2024. [Online]. Available: https://fred.stlouisfed.org/series/CPIAUCSL

[32] “Regular Seasonal Factors: Auto Production (G17MVSFAUTOS) | FRED | St. Louis Fed.” Accessed: May 26, 2024. [Online]. Available: https://fred.stlouisfed.org/series/G17MVSFAUTOS