NOVA IMS Information Management School

MEGI

Master's Degree Program in

Statistics and Information Management

The Predictive Power of Google Trends in the Portuguese Football Club Stock Markets

João Miguel Dias Miranda

Work Project

presented as partial requirement for obtaining the Master's Degree Program in Statistics and Information Management

NOVA Information Management School

Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

NOVA Information Management School

Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

The Predictive Power of Google Trends in the Portuguese Football Club Stock Markets

by

João Miguel Dias Miranda

Master Thesis presented as partial requirement for obtaining the Master's degree in Statistics and Information Management, with a specialization in Information Analysis and Management

Supervised by:

Bruno Damásio, PhD, NOVA Information Management School

July 2024

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledged the Rules of Conduct and Code of Honor from the NOVA Information Management School.

[João Miranda]

[Berlin, July 2024]

ACKNOWLEDGMENTS

I would like to express my gratitude to Professor Bruno Damásio for his invaluable guidance during this research. His knowledge and support were essential to the project's success, and I am grateful to have had this opportunity.

ABSTRACT

Given that the stock market for football clubs is heavily influenced by public sentiment, the aim of this thesis is to investigate the predictability of Google Trends data in the context of the Portuguese football clubs, using it as an indicator of public interest. Specifically, it examines the potential relationship between Google search trends and the stock market price. This research involves the development of three models with distinct features, per football club and machine learning model: one utilizing traditional features related to the stock market, another incorporating both these traditional features and Google Trends data specific to each football club, and lastly, one incorporating these two and Google Trends data associated to financial variables. The stock market prices, volume traded, oil price, gold price and euro to dollar exchange rate were extracted from a python library called 'yfinance', the Google Trends data was imported from 'trends.google.com' and the rest was feature engineered. By comparing the performance of these models, this study seeks to establish whether the inclusion of Google Trends data enhances their predictive accuracy, and which model has the best performance. We will also make the comparison between each football club. The results obtained suggest that the inclusion of Google Trends data regarding football clubs is an improvement to football clubs stock market price predictive models. The inclusion of Google Trends data of economic keywords adds too much complexity to the model, not improving the predictions. The best results, measured with a RMSE metric, were achieved using a Multilayer Perceptron. The findings of this research could provide valuable insights for investors, analysts, and policymakers regarding the usefulness of alternative data sources, such as Google Trends, in predicting stock market movements.

KEYWORDS

Multilayer Perceptron; Investor Sentiment; Google Trends; Stock Price Prediction; Portuguese Football Clubs

Sustainable Development Goals (SGD):

TABLE OF **CONTENTS**

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ANN	Artificial Neural Network
CCI	Commodity Channel Index
FCP	Futebol Clube do Porto
GDP	Gross Domestic Product
GI	Google Index
MDA	Mean Decrease Accuracy
MDI	Mean Decrease Impurity
MLP	Multilayer Perceptron
NN	Neural Network
PPI	Producer Price Index
RMSE	Root Mean Square Error
RSI	Relative Strength Index
SCP	Sporting Clube de Portugal
SLB	Sport Lisboa e Benfica
SMA	Simple Moving Average
SVR	Support Vector Regression

1. INTRODUCTION

Since the beginning of football, several European clubs, in the pursuit of external capital, decided to get listed in the stock market. The earnings on football stocks have been notable over the last years, making it potentially more interesting for professional investors (Hagen & Cunha, 2019).

Understanding the factors influencing stock prices of football clubs is key, with multiple variables having been identified to have significant effects.

The impact of on-pitch performance, game importance, and even betting odds on stock prices have been explored, shedding light on the relationship between football outcomes and market valuations (Renneboog & Vanbrabant, 2000; Castellani et al., 2015). Additionally, off-field factors such as revenue growth and financial performance metrics have been shown to influence stock prices positively, accentuating the multidimensional nature of the stock market in the football industry (Hagen & Cunha, 2019; Prayoga et al., 2022).

With Google having a dominant market share in Portugal's search engine landscape and Google Trends being utilized for economic activity tracking by multiple organizations, the potential of search data as an important indicator has been highlighted (Search Engine Market Share Worldwide, 2023).

Amidst this landscape, the role of Google Trends and searches in financial markets has emerged as a noteworthy area of investigation.

While studies abound on the predictive power of Google Trends in financial markets, there remains a notable gap in understanding its applicability to football club stocks, a field moved by public interest.

This thesis aims to bridge this gap by investigating the role of Google Trends in predicting the stock market price of football clubs, in the Portuguese context, drawing upon methodologies established in financial market analysis.

Portugal serves as an intriguing case study for several reasons. The country has a fervent football culture, with a passionate fan base, and the notable performance of Portuguese clubs on the European stage highlights the relevance and impact of football in Portugal socio-economic environment. Also, the technological scenery in Portugal supports this study, since Google has been the leading search engine in the country, holding over 95% of the market share since 2014. This dominance ensures a robust dataset from Google Trends, as a large portion of online searches related to football clubs is captured accurately, providing reliable insights into public interest and sentiment.

This project selects the three most prominent clubs listed on the Portuguese stock exchange (Euronext Lisbon) for comprehensive analysis: 'Sport Lisboa e Benfica', 'Futebol Clube do Porto' and 'Sporting Clube de Portugal'.

Due to the influence of the economic environment in the stock market, an additional layer of complexity was incorporated, which could provide deeper insights into stock market dynamics.

With this information, the primary research question driving this work is:

- What is the impact of Google Searches on the stock prices of the Portuguese clubs traded in the stock market?

Supplementary to this primary question, the research also evolves into secondary questions including:

- What is the impact of Google Searches of economic topics on the stock prices of the Portuguese clubs traded in the stock market?
- What machine learning model has the best performance?
- What combination of features achieves the best results?

In conclusion, by examining the relation between Google Trends data, technical indicators, and target variable, this study seeks to elucidate the potential of search data as a predictive tool in the context of football club investments.

To achieve this, this project begins with a comprehensive literature review, analysing important concepts and identifying research gaps. Following this, we present the data and features used in the models, along with relevant descriptive statistics. The methodology section outlines the strategy employed to achieve our research objectives. Subsequently, we provide a detailed explanation of the two models used in this study: Support Vector Regression and Artificial Neural Networks. The results achieved are then presented and discussed, leading to our conclusions. Finally, we offer suggestions for future work based on our findings.

2. LITERATURE REVIEW

2.1. METHODS

A meticulous search was conducted to find relevant studies for the literature review and to better understand the gaps and the connections within these topics.

The search terms used were “stock AND (google trend OR google search OR search engine)”, “stock AND predicting AND (football OR soccer)”, “stock AND (soccer OR football)” and “predicting AND stock AND (google trend OR google search OR search engine)” which created multiple combinations of topics.

The academic databases used to extract these published articles were Scopus and Google Schoolar.

The Diagrams can be seen in figure 2.1.

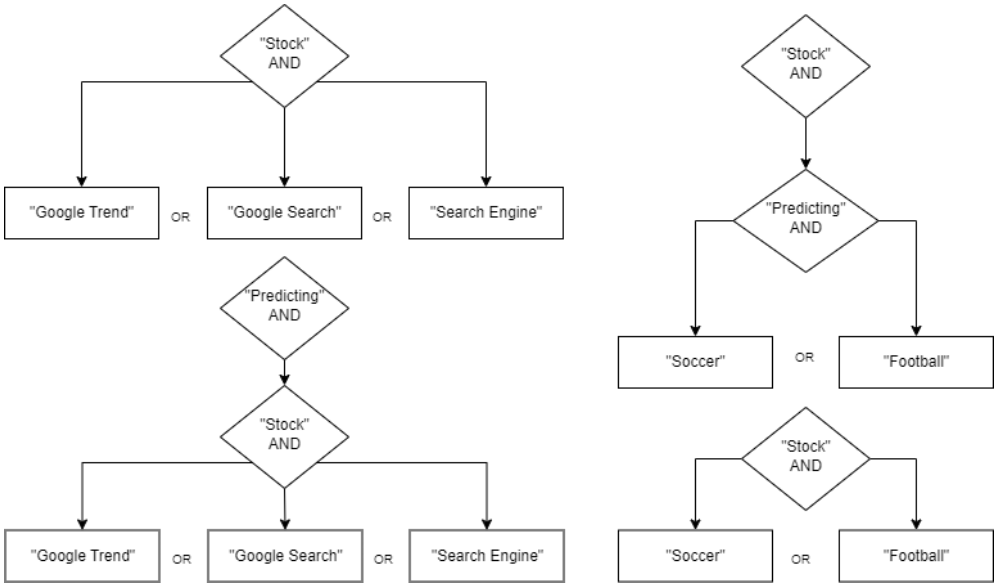

Figure 2.1 Search Terms Diagrams

Initially, 286 publications were obtained just in Scopus, before applying any restrictions.

Following the screening process, some cleaning steps were undertaken to retain only the relevant works for analysis.

The first step was to remove duplicates, as some publications had been extracted using different keyword combinations. From the initial 286 publications, 10 duplicates were removed.

Subsequently, only the most relevant and significant publications were retained. Out of the remaining 276, 115 key publications from Scopus were kept.

After merging the results from both identification sources, no duplicates were found, resulting in a total of 132 relevant publications.

The identification flow can be seen in figure 2.2.

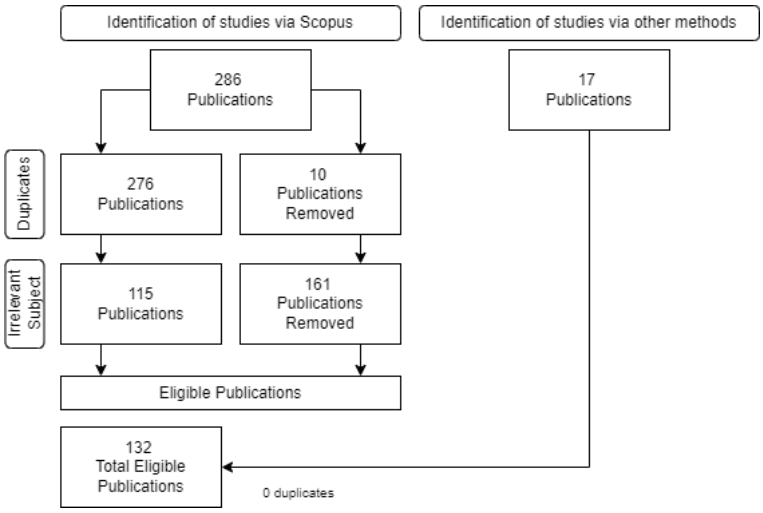

Figure 2.2 Identification Flow

2.2. RESULTS

Building on the identified publications, network analysis using VOS viewer allows for a more detailed visualization of keyword occurrence within the literature.

This method reveals the structural dynamics and relationships of researched fields. By mapping these keywords, we can identify key links and emerging trends.

In this literature review, we utilize VOS viewer to analyse the 132 publications from Scopus and Google Scholar, providing in figure 2.3 an overview of the academic landscape.

Only keywords with at least one connection, were kept on the analysis.

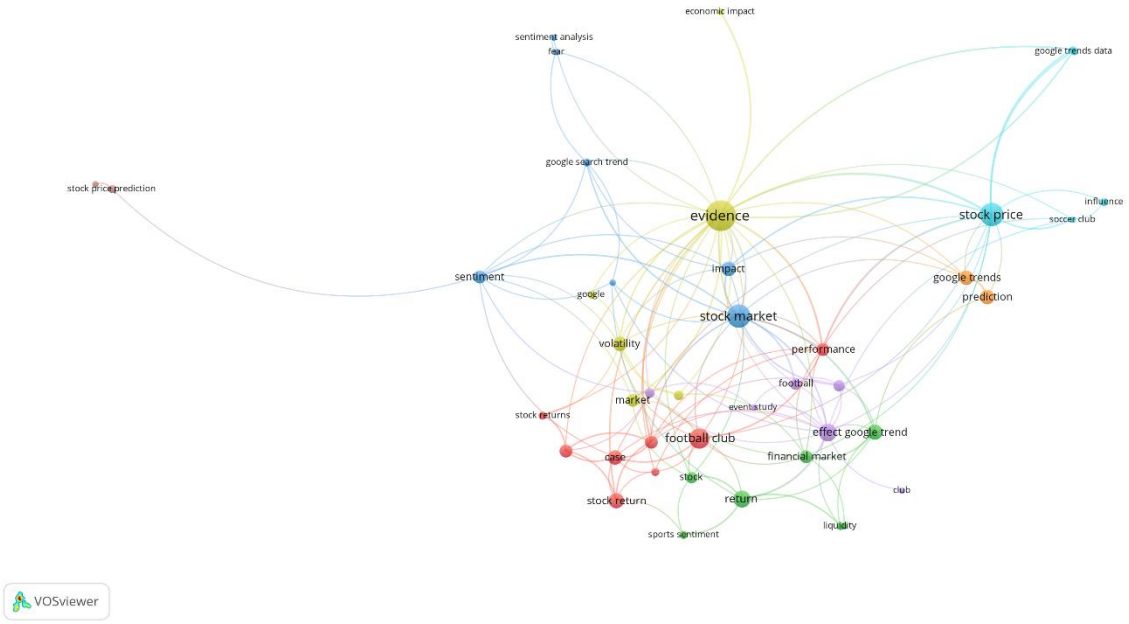

Figure 2.3 Network of Keywords

Doing an analysis on the publication year is also crucial as it highlights the evolution and development of research trends over time. This temporal perspective helps us to identify emerging topics, the progression of knowledge, and periods of significant scientific advancements or declines within the field.

In figure 2.4 we can see by publication date, the relevance of the topic.

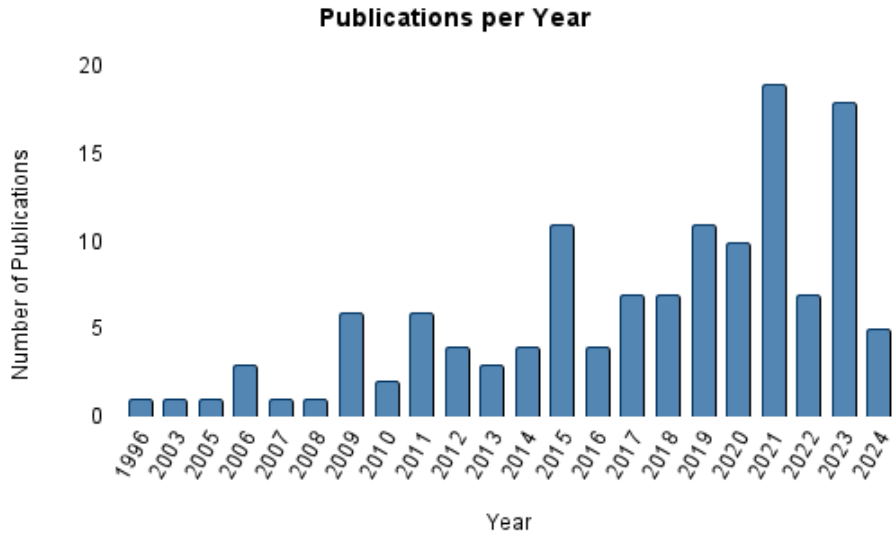

Figure 2.4 Evolution of Papers by Publication Year

From the 132 relevant publications identified, we found that they were distributed across 107 different journals.

However, to better understand the key journals in the field, table 2.1 highlights only those journals with more than one publication.

Table 2.1 Publishing Journals

JOURNALS	NUMBER OF ARTICLES
INTERNATIONAL JOURNAL OF SPORT FINANCE	4
APPLIED ECONOMICS	4
PLOS ONE	4
SOCIAL SCIENCE JOURNAL	3
JOURNAL OF SPORTS ECONOMICS	3
PHYSICA A: STATISTICAL MECHANICS AND ITS APPLICATIONS	3
RESEARCH IN INTERNATIONAL BUSINESS AND FINANCE	3
INTERNATIONAL JOURNAL OF FORECASTING	2
JOURNAL OF SPORT MANAGEMENT	2
SPORT, BUSINESS AND MANAGEMENT: AND INTERNATIONAL JOURNAL	2
EMERGING MARKETS FINANCE AND TRADE	2
APPLIED ECONOMICS LETTERS	2
FINANCIAL MANAGEMENT	2
EUROPEAN SPORT MANAGEMENT QUARTERLY	2
FINANCE RESEARCH LETTERS	2

The journals with the most published articles were the International Journal of Sport Finance, Applied Economics and PLOS ONE, with 4 articles published.

2.3. CONTENT ANALYSIS

Football Clubs and the Stock Market Price

Throughout football's history, numerous football clubs have sought external funding by entering the stock market. Various studies have examined the variables influencing the stock prices of football clubs, providing a foundation for exploring the impact of Google Searches on these prices (Morrow, 2023).

One significant factor affecting stock prices is on-pitch performance. On-pitch performance and the importance of the game was studied on 17 British clubs over three seasons. Their findings showed that victory gave a positive abnormal return, while a draw and loss gave a negative abnormal loss. Furthermore, their study showed that the importance of the game also affected the outcome. The effect on the stock price was greater for relegation or promotion decisive matches and European games (Renneboog & Vanbrabant, 2000).

Betting Odds were also used to understand to which extent expected and unexpected match results affect the price, indicating that market sentiment and expectations play a role in stock price fluctuations (Castellani et al., 2015). This suggests that external interest and attention, potentially measured by Google Searches, could similarly influence stock prices.

Off-field factors have also been shown to influence stock prices. Revenue growth and book value of equity growth were found to positively influence stock prices (Hagen & Cunha, 2019). Additionally, financial performance metrics such as the liquidity ratio measured by the Current Ratio, and the solvency ratio measured by the debt to asset ratio, significantly affect stock prices. However, the profitability ratio, measured by the Net Profit Margin indicator, showed no significant effect on the stock prices of football clubs (Prayoga et al., 2022). Understanding how financial indicators affect stock prices can help contextualize the potential impact of economic-related search trends.

Google Trends/Searches

In Portugal, Google dominates the search engine market with a share exceeding 95% since 2014 (Search Engine Market Share Worldwide, 2023). This dominance explains the relevance of using Google Trends data in our analysis, as it reflects most of the search behaviour in the country.

Moreover, Google Trends has evolved into a tool utilized by the OECD for weekly tracking of economic activity. This involves a two-step model to estimate weekly GDP growth based on Google Trends data (Woloszko, 2020). This application highlights the potential of Google search data to provide timely and accurate economic indicators. By leveraging Google Trends data related to economic topics, we aim to assess its predictive power in the context of football club stock prices.

Additionally, Google searches have proven valuable in predicting housing market trends and prices. The use of this search data produced more accurate out-of-sample predictions than the baseline model that uses conventional data but lacks search data (Brynjolfsson & Wu, 2015). This demonstrates the enhanced predictive capability of incorporating Google search data, supporting our research questions.

Furthermore, Google Trends were used to illustrate the increase of public attention that Saudi Arabia's pro League has received over the recent years (Mutz, 2024). This study just shows the utility of search data in tracking public interest in specific sectors, such as football.

These examples underscore the multifaceted utility of Google's search data in various domains, from economic forecasting to tracking public interest in specific sectors like football.

However, Google's search data has some limitations.

Among these limitations, the most obvious one is that all the search data available through Google Trends reflects those with internet access, potentially excluding vulnerable groups or regions where internet access could be low.

Google trends were used to predict the number of cases of influenza, but quickly was reported that Google Flu Trends was predicting more than double the proportion of doctor

visits for influenza illness than the Centres for Disease Control and Prevention, which bases its estimates on surveillance reports. One of the reasons for the error was the media panic, that led to spike the predictions. Another was the wrong choice of search terms, that although correlated with flu data, were not directly related to influenza flu. And the change in algorithm dynamics, that was done might also have affected the results across time, since the algorithm producing the data has been modified (Lazer et al., 2014).

The assumption that the sheer volume of data will naturally lead to accurate and reliable insights is wrong. The quality, relevance, and accuracy of the data are crucial for a meaningful analysis.

In a systematic review of the use of google trends in health care research, that surely can be applied to every field, showed that lack of documentation precludes the reproducibility of the findings. The recommended checklist for documentation would be the access date, the time period of the data, the query category, the search input, and the search strategy (Nuti et al., 2014).

Google Trends in the Stock Market

With special focus on the stock market, Google Trends has emerged as a valuable tool, with several notable developments already in place.

Studies conducted in various markets shed light on the relationship between online search activity and stock-trading behaviour. For instance, research on the Japanese stock market revealed that increased search activity correlates with higher trading volumes (Takeda & Wakao, 2014). This suggests that search activity can be a strong indicator of investor interest and market movements.

Similarly, in a major stock exchange of India, using google search volume as a measure of invertors information demand has proven effective in driving stock market liquidity and increasing trading activity (Jha et al., 2019). In the same study, using Google Search data provided more accurate predictions, aligning with our investigation into how Google search trends can impact the stock market, potentially improving our predictive models.

It was also proved to exist a positive correlation between the volume of searches in Google for the term ‘S&P500’ with the overall market index liquidity, volatility, and return (Água, 2014). These findings demonstrate the predictive power of Google Trends data in capturing market dynamics.

In addition, it was also already proved with clear evidence that weekly transaction volumes of S&P500 companies are correlated with weekly search volume of the corresponding company names (Preis et al., 2010).

Further supporting this, a study published in ‘The Online Journal of Science and Technology’ analysed the relationship between stock market indices and Google Trends data, finding

robust correlations between search volume and stock market index rates. These findings suggest that stock market indices are more frequently searched for as market values increase (Bozanta et al., 2017).

In the football field, researchers have studied the influence of emotions on the financial performance of football clubs listed on the stock market. They utilized Google Trends data as an explanatory variable and an indicator of sentiment, with the STOXX Europe Football Index serving as a benchmark for profitability. Applying an econometric model, the study revealed that investors sentiment, as measured by the interest in football-related topics, holds statistical significance (Martínez et al., 2020). This is particularly relevant to our study, as we investigate how Google searches reflect public sentiment and interest and its impact on the stock prices of Portuguese football clubs.

On an economic level, it was found that a relative increase in the search of the word 'debt' tended to precede stock market decline (Preis et al., 2013). This is an indicator of the relevance of search words in the economic environment that is projected to the stock market, directly linked to our second research question regarding the impact of Google Searches of economic topics on the stock prices of Portuguese football clubs.

In conclusion, the integration of Google Trends data into stock market analysis has provided promising results. It provides valuable insights into investor sentiment, information demand, and market dynamics, enhancing the understanding and prediction of stock market behaviour.

Predicting Stock Market

The stock market is affected by many factors, such as political events, general economic conditions, and traders expectations.

Recent studies have explored the integration of Google Trends data into stock market prediction models.

A study investigated the predictive power of Google Trends in forecasting contagion from the US stock market to international markets amid the 2008 financial crisis. By incorporating Google Trends data into dynamic models, they found that it significantly enhanced contagion prediction, particularly during volatile market periods, showcasing its potential as a valuable leading indicator for financial risk management (Maneejuk and Yamaka, 2019). By leveraging Google Trends data, we aim to predict stock price movements during periods of high volatility and market stress.

Additionally, results from empirical analysis highlight the substantial role of event impact in volatility prediction, emphasizing the effectiveness of integrating event data with macroeconomic indicators. Notably, combining Google Trends data with PPI and GDP provides optimal results in predicting volatility (Xu et al., 2019). This proves that by

combining search data with traditional economic indicators, we can enhance our understanding of how economic factors affect stock prices.

Furthermore, research utilizing text mining techniques on 10,000 Central Bank speeches developed a financial dictionary, which was then employed to explore public interest in financial news using Google Trends indices. The research focuses on examining the relationship between these indices and financial market turbulence, employing Deep Learning techniques, and benchmarking them against various Machine Learning algorithms and traditional statistical methods. The key finding indicates that Google queries effectively predict future market turbulence within a short timeframe, with Deep Learning algorithms outperforming traditional techniques (Petropoulos et al., 2021). We will use a similar methodology, where we compare machine learning models accuracy, and we will also leverage the power of google trends data of financial variables to explore market dynamics.

It was already proved, in companies listed in the Ghana stock exchange, with the use of a Multilayer Perceptron, that the stock market price is predictable using public sentiments. The increase in accuracy recorded by the model in predicting multiple days ahead based on google searches and the combination of other data points, shows the relevance of the public interest (Nti et al., 2020). This finding shows that by integrating search data with other relevant features, we aim to improve the accuracy of our predictions, and that Multilayer Perceptron is currently used for stock price predictions.

To forecast the direction of the opening prices of the S&P 500 and DJIA indices, Google Trends data was utilized in neural network models. The results indicate that integrating Google Trends data can greatly enhance the accuracy of stock market predictions. This study suggests that future research should investigate the impact of Google Trends on stock prices and trading volumes (Hu et al., 2018). This directly supports our exploration into how Google search data can be used to predict stock prices of Portuguese football clubs, and which machine learning models and feature combinations could give the best results.

In summary, recent papers indicate that integrating Google Trends data into stock market prediction models has shown promising results, offering valuable opportunities for enhancing our understanding and predictive capabilities in financial markets.

2.4. RESEARCH GAP

The predictive power of Google Trends in the stock market has been extensively explored in academic literature. However, a notable research gap exists in its application specifically to the domain of football clubs.

While numerous studies have investigated the relationship between Google search data and stock market trends, there is a lack of research focusing on the predictive capabilities of Google Trends for football clubs stock market.

Football, being a prominent field of investment, presents a unique environment where sentiment and public interest converge to influence market dynamics.

Thus, examining the predictive power of Google Trends for the stock performance of these clubs holds significant potential for enriching our understanding of investor behaviour in the context of sports-related investments.

We focused on the top 3 Portuguese football clubs, integrating two levels of layers to the Google Trends data. One layer reflects public interest in the clubs, while the other layer expresses the public interest in certain financial topics, that might reflect the economic state of the country and consequently the stock market price.

In conclusion, this research aims to fill this gap by providing insights into the relationship between online search activity, market sentiment, and stock market price within the context of Portuguese football clubs.

3. DATA

A comprehensive review examined 138 journal articles published between 2000 and 2009, shedding light on the application of machine learning techniques for stock market prediction, providing important insights of the main features, predicted variables, metrics and models utilized (Kumbure et al., 2022). The most frequently employed stock market variables include the lagged closing price from the previous day, as well as lagged high, low, open prices, and volume traded. Additionally, popular technical indicators such as the returns or the relative strength index over various periods that capture the momentum, simple moving averages over different periods that capture the trend and the bias or the volatility over a specific period to capture the volatility itself. Macroeconomic variables play a crucial role in market prediction literature, with exchange rates, particularly with the US dollar, being the most prevalent currency considered. Commodities such as crude oil and gold also feature prominently. Economic performance indicators are also frequently included in predictive models, that we will incorporate with the use of Google Financial Trends.

3.1. SAMPLE SIZE

We will have a sample of five years of data, which will be the last five seasons played (2019/2020, 2020/2021, 2021/2022, 2022/2023 and 2023/2024), which means we will have data since 1 of July of 2019 until 30 of June of 2024.

The sample includes the three biggest football clubs in Portugal.

The sample of five years was chosen because the Google Trends frequency, over five years periods, goes from weekly to monthly, and since we want to measure its predictive power, it is more interesting to have weekly data.

3.2. BENCHMARK INDEPENDENT VARIABLES

The independent variables used in this paper are the 1-day lag of closing price, 1-day lag of high price, 1-day lag of low price and 1-day lag of open price, 1-day lag of the returns, the crude oil price and gold price, the euro to dollar exchange rate, the relative strength index with a 30 days window, the simple moving average of price and volume traded with a 10 and 20 days window, price volatility with a 10 days window and commodity channel index for a 30 days window.

Not all these variables were used in the model, since we only kept the more relevant, after applying feature selection techniques.

The granularity of our data is daily for all the stock market related variables such as closing price, volume traded, high price, low price and open price, for the returns, for the commodities crude oil price and the gold price, for the euro to dollar exchange rate, for the interest rate and for all the technical indicators.

Some descriptive statistics of the macroeconomic variables can be reviewed on table 3.1.

Table 3.1 Macroeconomic Variables Descriptive Statistics

VARIABLE	MEAN	STD	MEDIAN	MIN	MAX	VARIANCE	IQR	CV	SKEW	KURTOSIS	SE
OIL PRICE	69	21	72.5	-38	124	430	26	0.3	-0.28	0.41	0.59
GOLD PRICE	1827	182	18278	1420	24334	33307	198	0.1	0.29	0.98	5.18
EURO/DOLLAR EXCHANGE RATE	1.1	0.06	1.1	0.96	1.23	0.003	0.1	0.1	0.068	-0.44	0.002

3.3. GOOGLE-BASED INDEPENDENT VARIABLES

The exogenous variables that are specific to this study are the weekly Google Index that summarizes the searches performed through the Google search engine website, collected through Google Trends.

The Google Index (GI) represents the number of web searches that have been made for a particular keyword in a specific geographical area (*r*) within a given time, in our case the geographical area is Portugal, and the time frame is between '2019-07-29' to '2024-06-02'. The data was downloaded on '2024-06-05'.

The search share for a particular keyword on day (*d*) is given by the number of web searches containing that keyword ($V_{d,r}$), normalized by division by the total number of web searches performed through Google for the same day and area ($T_{d,r}$):

$$S_{d,r} = \frac{V_{d,r}}{T_{d,r}} \quad (1)$$

The search share for week *t* is given by the simple average:

$$S_{t,r} = \frac{1}{7} * \sum_{k}^{Saturday}_{d = Sunday} S_{d,r} \quad (2)$$

Google also scales the index $GI_{t,r}$ to 100 in the week in which it reaches the maximum level. Thus, the Google index for week *t,* is given by:

$$GI_{t,r} = \frac{100}{\max_t (S_{t,r})} * S_{t,r} \quad (3)$$

The searched keywords of football clubs are 'Sport Lisboa e Benfica', 'Sporting Clube de Portugal' and 'Futebol Clube do Porto', with the respective trends illustrated in figure 3.1.

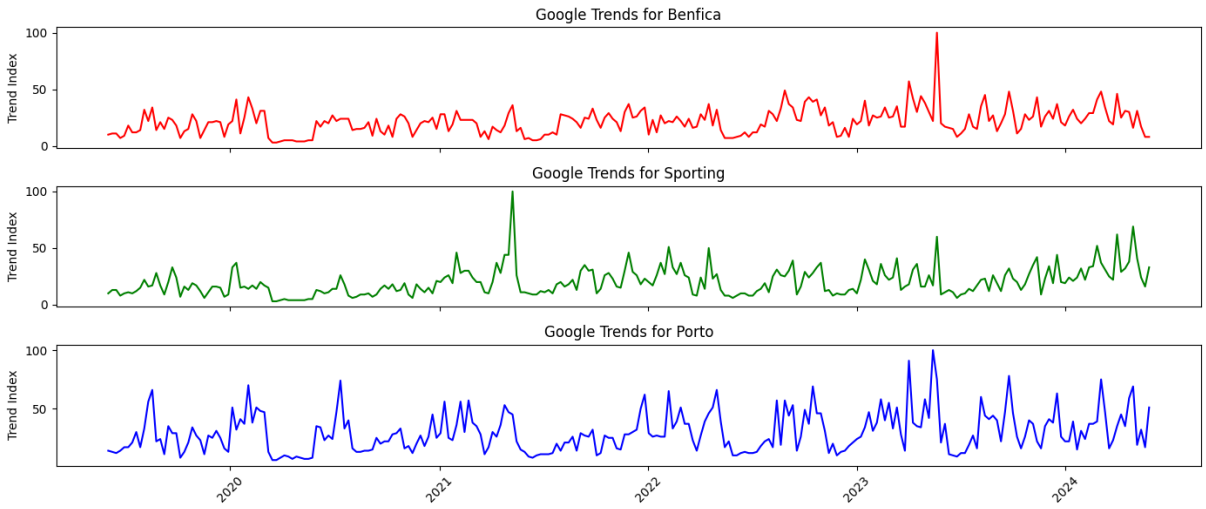

Figure 3.1 Football Club Google Search Index

Table 3.2 highlights some descriptive statistics about club trends.

Table 3.2 Club Trends Descriptive Statistics

VARIABLE	MEAN	STD	MEDIAN	MIN	MAX	VARIANCE	IQR	CV	SKEW	KURTOSIS	SE
BENFICA TREND	21.8	11.3	22	3	100	128.6	14	0.52	1.57	8.1	0.32
SPORTING TREND	20.4	12.4	18	3	100	152.5	15	0.61	1.83	7	0.35
PORTO TREND	29.9	16.9	27	6	100	287.6	23	0.57	1.02	1.1	0.48

Choosing which search terms to download is a crucial ingredient of every study utilizing the Google Trends tool, so for an accurate representation of the economic environment, we selected some of the most trended words in Portugal, related to it.

The searched keywords that represent the economic indicators are ‘Empréstimo’, ‘Preço’, ‘Taxas’, ‘Emprego’, ‘Investimento’, ‘Mercado de Ações’, ‘Salário’, ‘Finanças’ and ‘Segurança Social’. A review of its descriptive statistics can be found in table 3.3.

Table 3.3 Financial Trends Descriptive Statistics

VARIABLE	MEAN	STD	MEDIAN	MIN	MAX	VARIANCE	IQR	CV	SKEW	KURTOSIS	SE
EMPRÉSTIMO	48	10.5	47	22	100	109.5	13	0.22	0.52	2.1	0.3
PREÇO	72	8.7	71	44	100	75.6	14	0.12	0.12	0.2	0.25
TAXAS	51	11.9	49	26	100	143.2	16	0.24	0.71	0.75	0.34
EMPREGO	70	9.7	69	45	100	93.1	14	0.14	0.19	-0.06	0.27

INVESTIMENTO	70	11.7	69	36	100	135.6	16	0.17	-0.01	-0.17	0.33
MERCADO DE AÇÕES	23	10.7	21	0	100	113.6	8	0.46	3.1	16.9	0.3
SALÁRIO	33	12.8	32	7	100	163.5	16	0.40	1.1	2.9	0.36
FINANÇAS	21	11.1	17	10	100	123.8	8	0.54	3.18	15.3	0.32
SEGURANÇA SOCIAL	33	11.3	33	8	100	128.4	14	0.35	0.91	4.2	0.32

All the Google Search features will have a weekly frequency.

3.4. TARGET VARIABLE

The variable we are going to predict is the stock market daily closing price, a numerical variable. The target variable evolution, per club, can be seen in figure 3.2.

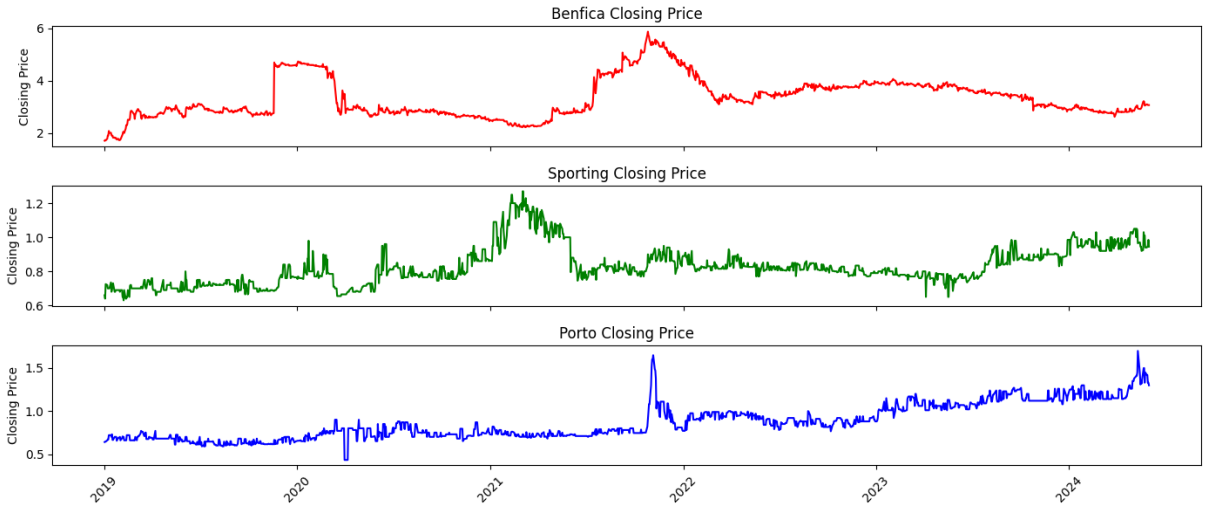

Figure 3.2 Target Variables per Football Club

Table 3.4 provides a summary of its descriptive statistics.

Table 3.4 Target Variables Descriptive Statistics

VARIABLE	MEAN	STD	MEDIAN	MIN	MAX	VARIANCE	IQR	CV	SKEW	KURTOSIS	SE
BENFICA CLOSING PRICE	3.4	0.75	3.3	2.2	5.9	0.56	1.03	0.22	0.74	-0.05	0.02
SPORTING CLOSING PRICE	0.89	0.2	0.85	0.43	1.7	0.04	0.33	0.23	0.66	-0.12	0.01
PORTO CLOSING PRICE	0.84	0.11	0.82	0.65	1.3	0.01	0.12	0.13	1.17	1.7	0.003

4. METHODOLOGY

In the first phase we will extract the data from multiple sources and import them all into 'Google Colaboratory', a hosted Jupyter Notebook service.

The stock market data, crude oil price, gold oil price and the euro to dollar exchange rate will be imported from 'yfinance' library, that is a Python package, from Yahoo Finance. The Google search data will be extracted from 'trends.google.com'.

Then we start with the data treatment, where we handle missing values, since the stock market is closed during holidays and weekends.

We will proceed to generate other features for our predictions, such as:

1. Relative Strength Index (RSI) with a 30-day window

$$RS_{30} = \frac{Average\ Gain\ over\ the\ last\ 30\ periods}{Average\ Loss\ over\ the\ last\ 30\ periods} \quad (4)$$

Where average gain over the last 30 periods is the average of positive price changes observed in the last 30 periods, representing the magnitude of price increases during that time frame. The opposite applies to average loss over the last 30 periods.

$$RSI = 100 - \left(\frac{100}{1+RS_{30}}\right) \quad (5)$$

2. Simple Moving Average with 10-day and 20-day window

$$SMA_{10} = \frac{\sum Closing\ Prices\ over\ the\ last\ 10\ days}{10} \quad (6)$$

3. Volatility over a 10-day window

$$Volatility_{10} = \sqrt{\frac{1}{10-1}\sum_{i=1}^{10}(R_i - \bar{R})} \quad (7)$$

where R_i represents each individual return within the 10-day window and $\bar{R}$ is the mean return over the 10-day window.

4. Commodity Channel Index

$$CCI = \frac{\sum_{i=1}^{n}\left(\frac{High+Low+Close}{3}\right) - SMA_n}{0.015 * \left(\frac{\sum_{i=1}^{n}\left|\sum_{i=1}^{n}\left(\frac{High+Low+Close}{3}\right) - SMA_n\right|}{n}\right)} \quad (8)$$

5. Returns

$$R_t = \frac{P_t - P_{t\_1}}{P_{t\_1}} \quad (9)$$

Where P_t is the closing price of the stock at time t and P_{t-1} is the closing price at time t-1.

6. And 1 day lag of the open price, close price, low price, high price, volume traded and returns.

After the feature engineering, the numerical data will be scaled, applying a min-max normalization, with a range of [0,1]. MinMax scaler is a normalization technique already incorporated in many stock market price prediction models (Priyatno et al., 2024).

$$X_{Scaled} = \frac{X - X_{min}}{X_{max} - X_{min}} \quad (10)$$

Additionally, a research of feature selection methods for stock market prediction showed that Random Forest was one of the most used methods, which is an ensemble learning technique with two methodologies for calculating the feature importance: mean decrease accuracy (MDA) and mean decrease impurity (MDI). MDA describes how much prediction accuracy the model loses after removing each feature, and MDI is a measure of how each feature contributes to the homogeneity of the nodes and leaves for each decision-tree model (Htun, et al, 2023).

We will apply it to each model and extract the most important features, that will be the 'Base features'.

The models used are a Support Vector Regression (SVR) and a Neural Network.

For the train and test division, we will apply a method known as the recursive scheme or expanding window, which is a type of cross-validation method useful for time-series data that gives a realistic perspective of time (Katsafados, 2022). In this technique, the training window starts with an initial size and gradually expands to include more data points. The model is trained on each window and evaluated on the subsequent window, as exemplified in figure 4.1. We applied a 10-fold split.

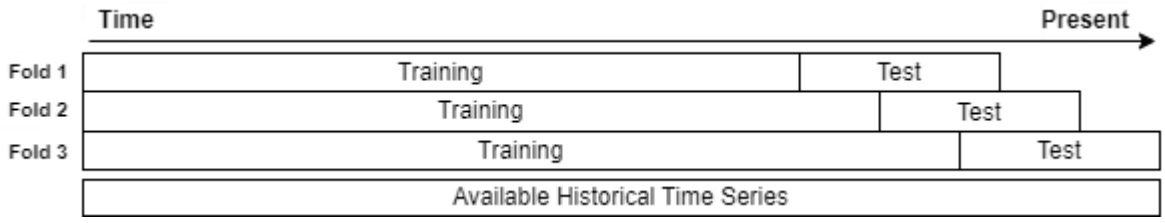

Figure 4.1 3-Fold Cross Validation Expanding Window Example

We will make predictions for the target variable, with each model, using only the 'Base features', using both the 'Base Features' and the Google Search Index of the three clubs, and using the 'Base Features', the Google Search Index of the three clubs and a Google Search Index for Economic terms, as it can be seen in figure 4.2.

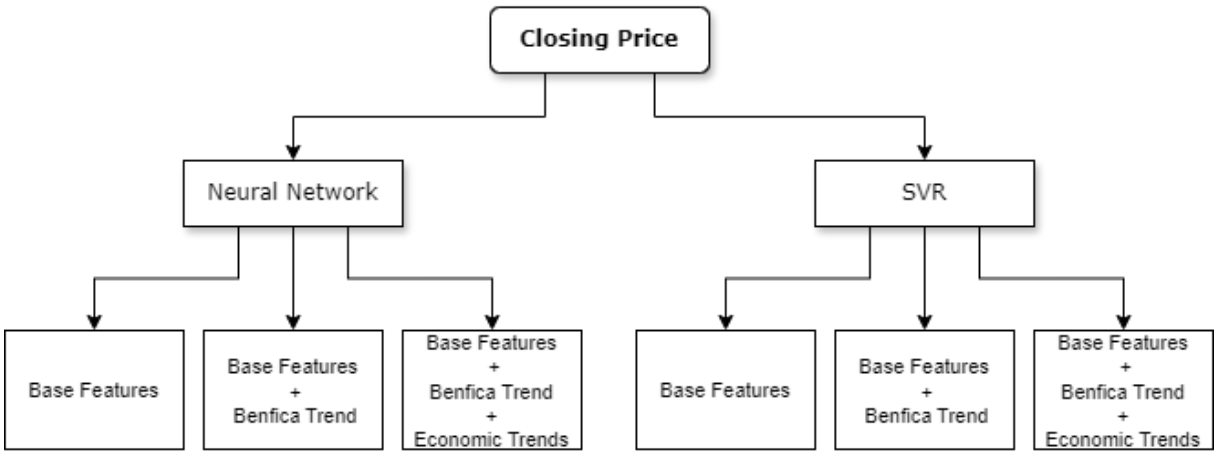

Figure 4.2 Prediction Flow per Club

After that we will analyse which machine learning model got the best results, and we will start tuning the hyperparameters. Since hyperparameter tunning is computationally and time expensive, we will only move forward with the model that gets the best results.

In the end, we will analyse if the use of google search data had better performance than the one using only the base features metrics.

In terms of performance metrics, the evaluation of predictive models leans heavily on several key measures, but Root Mean Squared Error (RMSE) emerges as the predominant metric (Kumbure et al., 2022). We will use the Root Mean Squared Error:

$$\text{RMSE} = \sqrt{\frac{\Sigma(y_i - \bar{y}_i)^2}{N-P}} \tag{11}$$

We will compare the results across various combinations of features and different football clubs to identify the most effective feature set for each club, and then we will discuss the findings.

An overview of the methodology can be seen in figure 4.3.

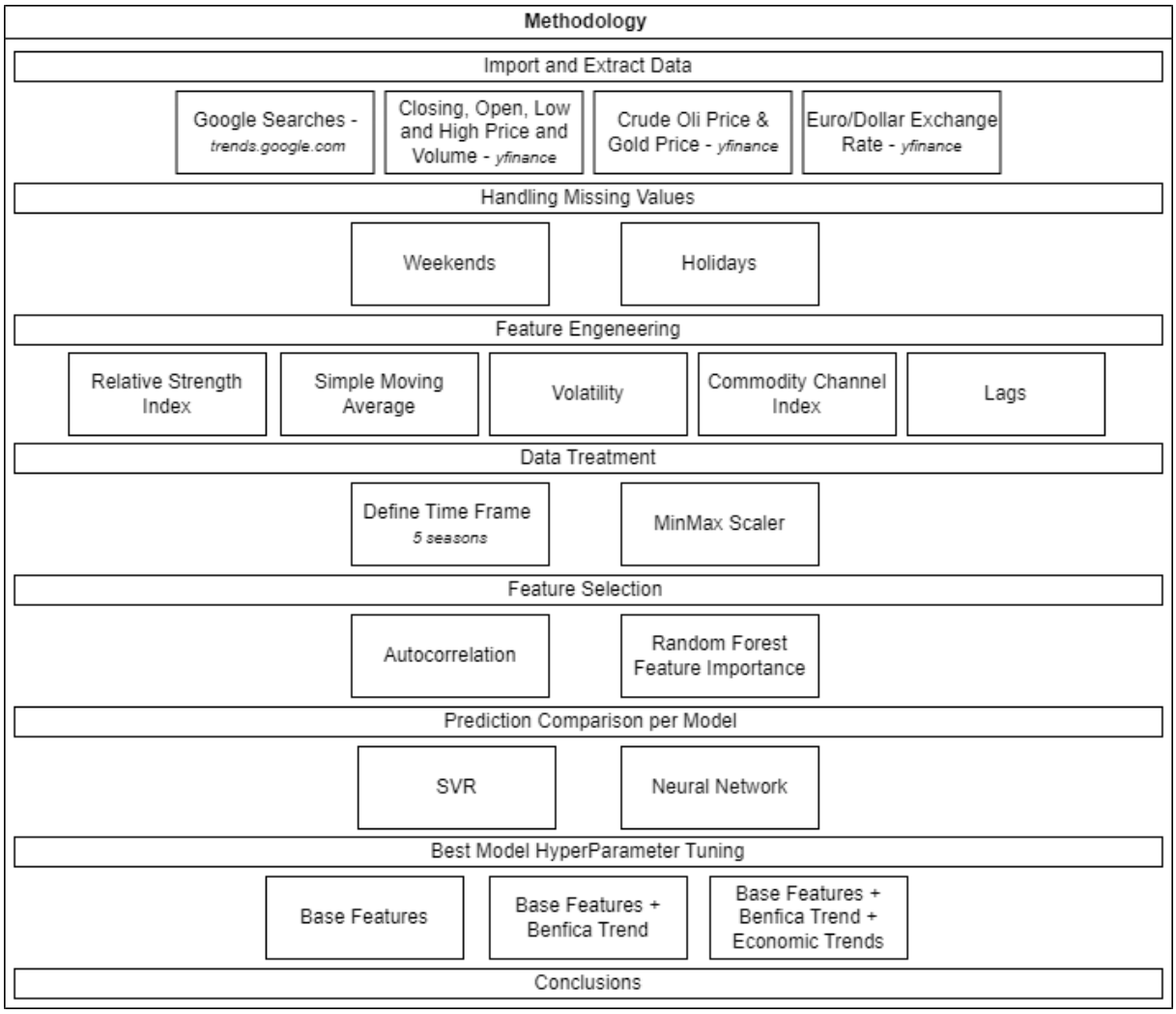

Figure 4.3 Overview of Methodology

5. MODELS

Still on the literature review done by Kumbure et al. (2022), analysing the use of machine learning techniques for stock market prediction, it highlighted the prevalence of certain models.

Neural Networks (NN) emerged as the most frequently employed machine learning model, closely trailed by Support Vector Regression (SVR).

Consequently, the models we will use are a Support Vector Regression and a Neural Network.

Initially, we will use both models without any hyperparameter tuning. However, because of the significant computational cost and lengthy runtime, only the superior model will proceed to the second phase of hyperparameter tuning, which was the Neural Network.

5.1. SUPPORT VECTOR REGRESSION

The Support Vector Machine concept was introduced by Vapnik (1998). The approach was originally designed for binary classification and then adapted for the prediction of numerical values (Support Vector Regression).

In the SVR model the goal is to find a function f(x) that has at most ε deviation from the actually obtained targets y_i for all the training data, and at the same time is as flat as possible (Smola & Scholkopf, 2004).

The f(x) kernel function can be:

1. Linear, when the relation between the input features and the target variable is linear.
2. Non-Linear, when the input features and the target variable do not have a linear relation. Input features are transformed into a higher dimensional space using a kernel function, enabling the model to capture complex relations.

The most general function f(x) is typically a linear function in the feature space:

$$f(x) = \langle w,x \rangle + b \text{ with } b \in X,\ b \in R \tag{12}$$

where

$$\langle w, x \rangle = \sum_i w_i x_i \tag{13}$$

and w is the weight vector, x is the feature vector, and b is the bias term.

Flatness means that one seeks a small w. One way to ensure this is to minimize the cost function:

$$\frac{1}{2} * \|w\|^2 + C * \sum_{i=1}^{n} L\epsilon(y_i, f(x_i)) \tag{14}$$

where $\frac{1}{2}$ * $\|w\|^2$ is the regularization term that ensure flatness, the constant C is a regularization parameter that determines the trade-off between the flatness of f(x) and the amount up to which deviations larger than ε are tolerated and $L\epsilon(yi, f(xi))$ is the ε-insensitive loss function.

The ε-insensitive loss function is defined as:

$$\begin{cases} 0 & ,if\ |y - f(x) \leq\ \varepsilon \\ |y - f(x) -\ \varepsilon & \text{otherwise} \end{cases} \tag{15}$$

This loss function indicates that errors within the epsilon margin are not penalized, making the model robust to small variations.

In figure 5.1 it can be seen summarized.

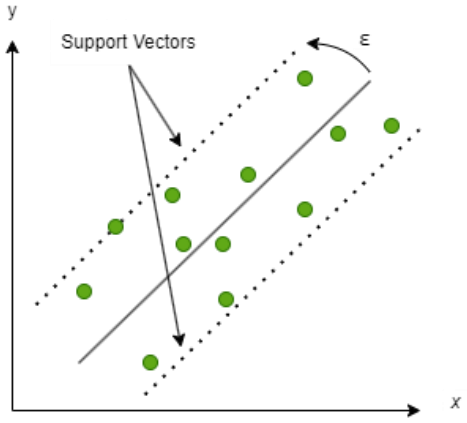

Figure 5.1 Univariate Linear Support Vector Regression

This is simply the most generic model. There are multiple combinations of parameters that can be adjusted, from the type of kernel function to the epsilon parameter (ε), the regularization parameter (C), and specific parameters of each kernel function such as the degree, the gamma, or the coefficient.

5.2. Artificial neural Network

An Artificial Neural Network (ANN) is a signal processing system composed of many simple processing elements which are interconnected by direct links, and which cooperate to perform parallel distributed processing to solve a desired computational task. A neural network can be thought of as a network of neurons organized in layers. (Macukow, 2016).

Since most of financial data, including stock market data, are nonlinear and asymmetric, Artificial Neural Network models can be helpful to recognize that relationship (Farahani & Hajiaga, 2021).

Perceptron is a type of artificial neuron and one of the simplest forms of artificial neural networks (Rosenblatt, 1958).

The perceptron is fed with real-valued inputs, each paired with a specific weight. This is like the dendrites in neurons that collect input. These inputs, which can be expressed as a vector, are multiplied by their corresponding weight, and summed up. The weights determine the importance of each input in determining the outcome. The weighted sum goes through a non-linear function, which determines the output of the Perceptron. For the original Perceptron, it was a step function with a threshold activation. This is illustrated in figure 5.2.

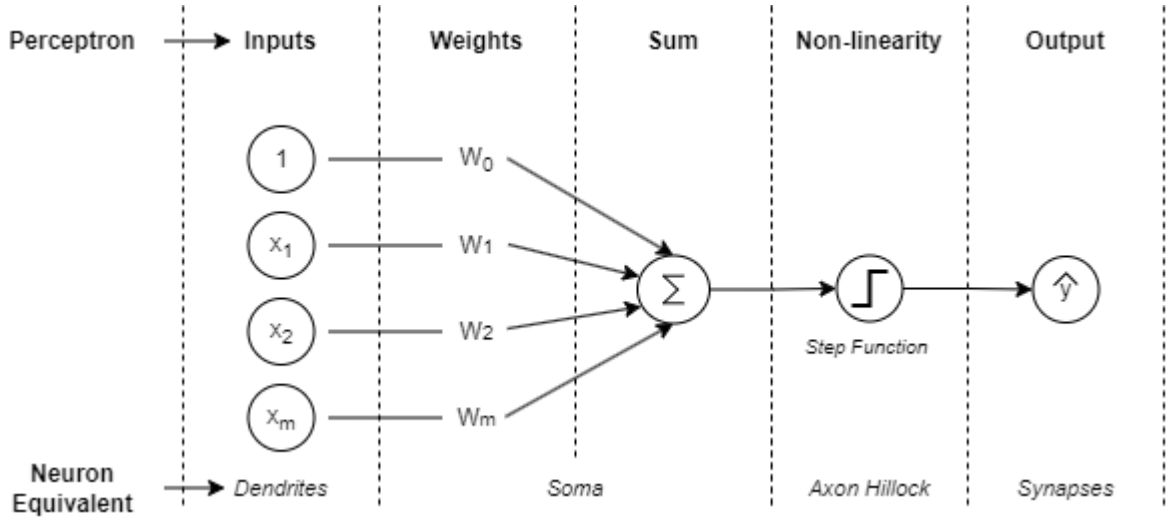

Figure 5.2 Perceptron

The perceptron output is defined by the weighted sum of inputs, which is passed in through a non-linear function.

$$\hat{y} = g\,(w_0 + \sum_{i=1}^{m} x_i\, w_i) \tag{14}$$

The Multi-Layer Perceptron, which is what we apply in our model and is represented in figure 5.3, is a feedforward artificial neural network with multiple layers of perceptron's, capable of modelling complex non-linear relationships between inputs and outputs.

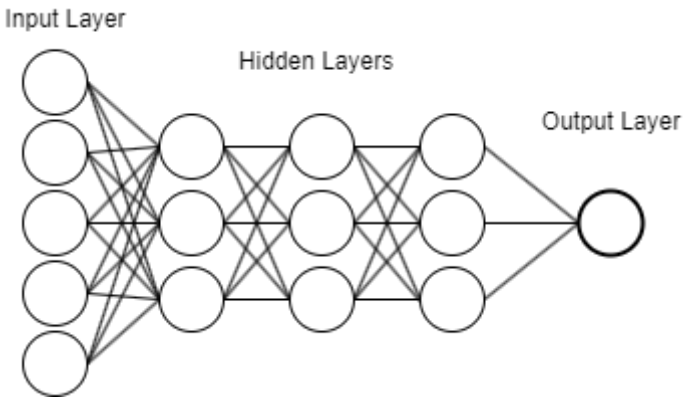

Figure 5.3 Multilayer Perceptron

The first layer of these models is the input layer of neurons, where the input features are introduced to the network. Each neuron corresponds to one feature of the input data.

After the input layer, there are the hidden layers, which are intermediate layers. A network can have multiple layers, each potentially with different number of neurons. Extra layers allow for the capacity to model more complex data.

The output layer generates predictions using the outputs of the hidden layers as its inputs. Backpropagation is done for all weights from input layer to output layer.

This model has many hyperparameters that can be changed to achieve better results.

We generate random intermediate layer configurations ranging from 3 to 5 layers, with each layer having between 10 to 120 neurons. This randomness allows for exploration of diverse architectures, and faster optimization times.

The activation function introduces nonlinearity to the model, enabling it to learn complex patterns and relationships in the data.

We tested the Tanh activation function, that squashes the input values into the range [-1,1], according to the following equation:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{15}$$

Neurons saturate at large negative and positive values, and the derivative of the function approaches zero, as seen in figure 5.4.

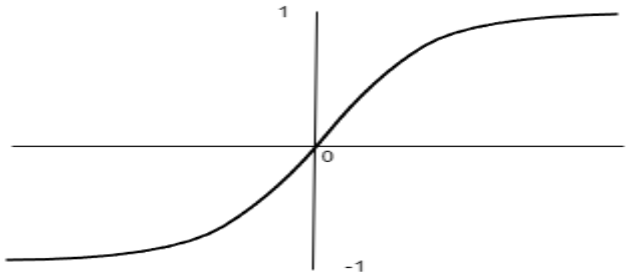

Figure 5.4 Tanh function

We also tested the Rectified Linear Unit activation function (figure 5.5), or simply Relu, which is linear for input values greater than zero, according to the following equation:

$$R(x) = \max(0, x) \tag{16}$$

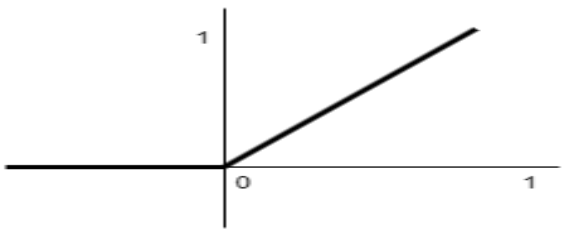

Figure 5.5 Relu Function

Another hyperparameter was the solver, which determines the optimization algorithm used to update the weights of the model during training.

The Stochastic Gradient Algorithm, or 'sgd', updates the weights of the network using the gradient of the loss function with respect to the weights. It updates the weights after computing the gradient of each training sample.

It can be represented by the following equation:

$$\Theta_{t+1} = \Theta_t - \eta * \nabla_{\Theta} J(\Theta_t) \quad (17)$$

where Θ_t are the weights at iteration t, η is the learning rate and $\nabla_{\Theta} J(\Theta_t)$ is the gradient of the loss function J with respect to the weights Θ_t.

The Adaptative Moment Estimation, or 'adam', adapts the learning rates for each parameter, allowing different weights to update at different rates. It often converges faster and is less sensitive to the choice of learning rate compared to the Stochastic Gradient Algorithm. The Adam optimizer's adaptive learning rate and momentum properties make it particularly well-suited for training deep neural networks.

Another parameter was the Alpha, a regularization parameter that controls the amount of regularization applied to the model during training. This parameter helps prevent overfitting by penalizing large weights in the network. It was sampled from a range between [0.001,0.1], determining how strongly the weights will be penalized. The higher the alpha, the stronger the penalization.

The last parameter that was adjusted was the Learning Rate that controls the step size during optimization, determining how much the weights are updated in response to the estimated gradient of the loss function.

We tested a constant learning rate, that maintains a constant rate throughout training. And we also tested the adaptative rate, which adapts the learning rate during training based on the validation score. This learning rate comparison is observed in figure 5.6.

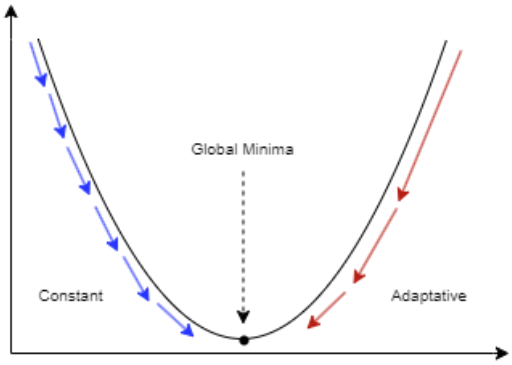

Figure 5.6 Constant vs Adaptative Learning Rate

The loss function used in regression our task is the Root Mean Squared Error:

$$\text{RMSE} = \sqrt{\frac{\sum(y_i - \bar{y}_i)^2}{N-P}} \tag{20}$$

It is defined as the square root of the mean of the squared differences between predicted and actual values. The lower the score, the better the results.

Properly tuning these parameters can significantly improve the performance of the model.

6. RESULTS AND DISCUSSION

Before any tunning of parameters two models were compared for all clubs and for each combination of features: Support Vector Regression and Neural Network.

Each model performed three times per football club, one with the base features, one with the base features and the club trend and the last one with the base features, the club trend and economic trend variables.

To select the base features, it was first created a correlation heatmap, per club. The decision, based on a club analysis, was to include the same features for all clubs, since they all presented similar patterns and for simplicity of comparison reasons. The correlation heat maps can be seen in figure A.1, A.2 and A.3 in Appendix.

We noticed, for all clubs, a high correlation between closing price and 1-day lag of closing, open, high and low price.

We applied a Random Forest model per club to analyse feature importance, associated with the target variable, and we end up including in the model the 1-day lag of closing price. It can be found in figure A.4 in Appendix.

It was also seen a high correlation between the simple moving average of 10 and 20 days of the closing price, but they were both kept in the model since they are important tools used to identify trends and potential trend changes.

When choosing which financial variables to incorporate in the model, we also included the same variables for all clubs, with the same reasoning. The variables used for financial trends are ´preço´, ´mercado de ações' and 'salário', and their trends are represented in figure 6.1.

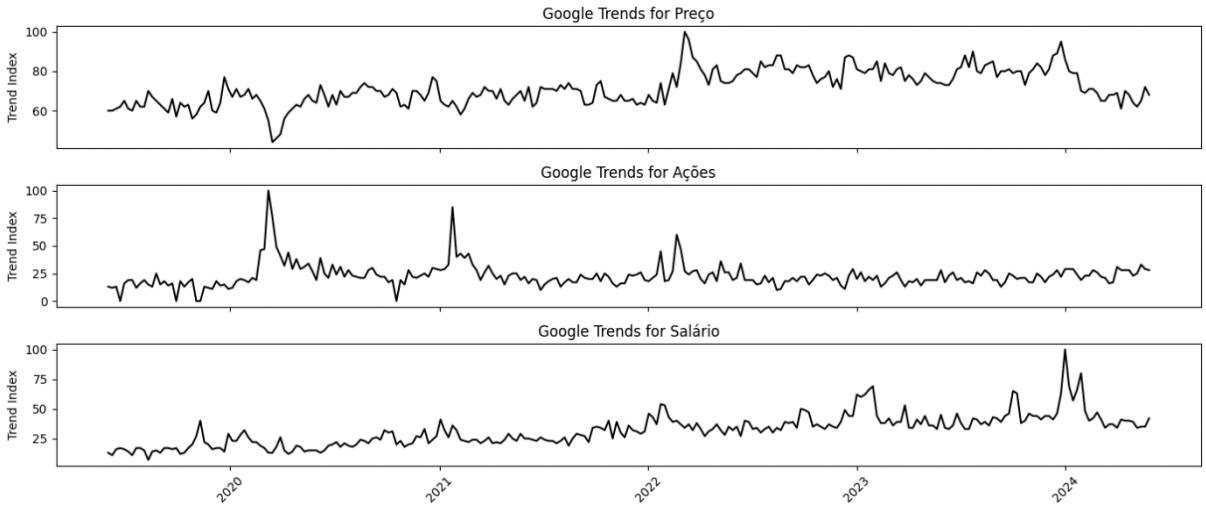

Figure 6.1 Financial Keywords Google Search Index

The feature importance can be found in figure A.5 in Appendix.

The observed was that the Neural Network was able to better understand the complexity and the relationships in the data, which got him better results.

Consequent of these results, the parameter tuning was only performed on the Neural Network model, more precisely, in a Multilayer Perceptron.

The initial phase involved generating randomly 75 configurations for the layers of the neural network. This was accomplished using a function that generates a specific number of random configurations, and in this case the parameters were between 3 and 5 hidden layers and the number of neurons in each layer was between 10 and 120.

Once these configurations were created, they were integrated into the model's hyperparameters. The hyperparameter distributions for the neural network included several key elements, such as the activation functions, the optimization functions, the alpha and the learning rate.

The maximum number of iterations of this model was set up to 2000, which means the model will train up to 2000 epochs unless any stopping criteria is met. While the maximum number of epochs specified is 2000, the actual number of epochs the model trains for may be less due to early stopping if the validation performance does not improve for 10 consecutive epochs. We defined that 10% of the data was used for validation during the training process to monitor the model performance and apply the early stopping.

By employing this approach, the model was prepared with a robust set of configurations and hyperparameters, ready for the subsequent steps of training and evaluation using a 10-fold time series cross-validation.

The achievements per club are seen below.

6.1. SPORT LISBOA E BENFICA

The results achieved by 'Sport Lisboa e Benfica', for the Support Vector Regression and for the Neural Network, before any hyperparameter tuning, presented in table 6.1, were the following:

Table 6.1 S. L. Benfica before Hyperparameter Tuning – SVR vs Neural Network

	RMSE Before Hyperparameter Tuning		
	Base Features	Base Features + Benfica Trend	Base Features + Benfica Trend + Economic Trends
SVR	0.221	**0.212**	0.216
Neural Network	0.187	**0.169**	0.195

Using the SVR model the best results were obtained using the 'Base Features + Benfica Trend', followed by 'Base Features + Benfica Trend + Economic Trends', and the worst results were obtained using simply the 'Base Features'.

The Neural Network model got the same results in terms of the best feature combination. The best results were obtained using the 'Base Features + Benfica Trend', followed by 'Base Features', and the worst results were obtained using the 'Base Features + Benfica Trend + Economic Trends'.

However, in all combination of features, the neural network model got better results.

Table 6.2 illustrates the results achieved by S. L. Benfica, focusing on Neural Networks, after the hyperparameter tuning.

Table 6.2 S. L. Benfica After Hyperparameter Tuning – Neural Network

	RMSE After Hyperparameter Tuning		
	Base Features	Base Features + Benfica Trend	Base Features + Benfica Trend + Economic Trends
Neural Network	0.166	0.154	**0.134**

From a first view from figure 6.2, the results achieved with hyperparameter tuning improved in all the combinations.

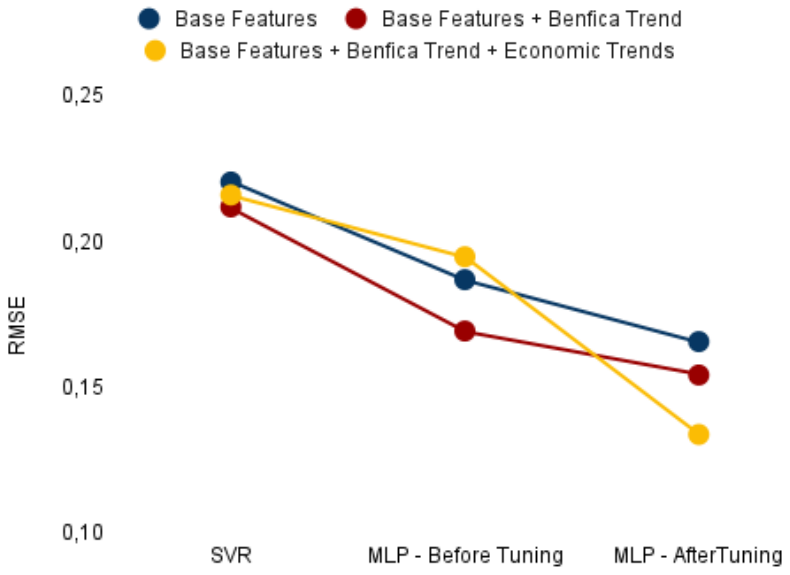

Figure 6.2 S. L. Benfica Models Comparison

The Multilayer Perceptron with 'Base Features + Benfica Trend + Economic Trends' got the best results, achieving a RMSE of 0,134. The second-best result was from 'Base Features +

Benfica Trend' with a RMSE of 0,154, and the worst result was with the use of the 'Base features', achieving only a RMSE of 0,166.

Table 6.3 summarizes the best set of hyperparameters obtained for each model.

Table 6.3 S. L. Benfica Hyperparameter Tuning

	Activation Function	alpha	Hidden layers	Neurons	Learning Rate	Solver
Base Features	tanh	0.018436642900499147	4	(41, 112, 36, 115)	adaptive	adam
Base Features + Benfica Trend	tanh	0.004142918568673425	5	(46, 60, 99, 96, 68)	constant	adam
Base Features + Benfica Trend + Economic Trends	tanh	0.004142918568673425	5	(46, 60, 99, 96, 68)	constant	adam

It is unanimous that the activation function with best results was the Tanh, and the optimization algorithm was the adam.

In figure 6.3 we can observe the actual values versus the predicted values for all the feature combinations, using a Multilayer Perceptron model.

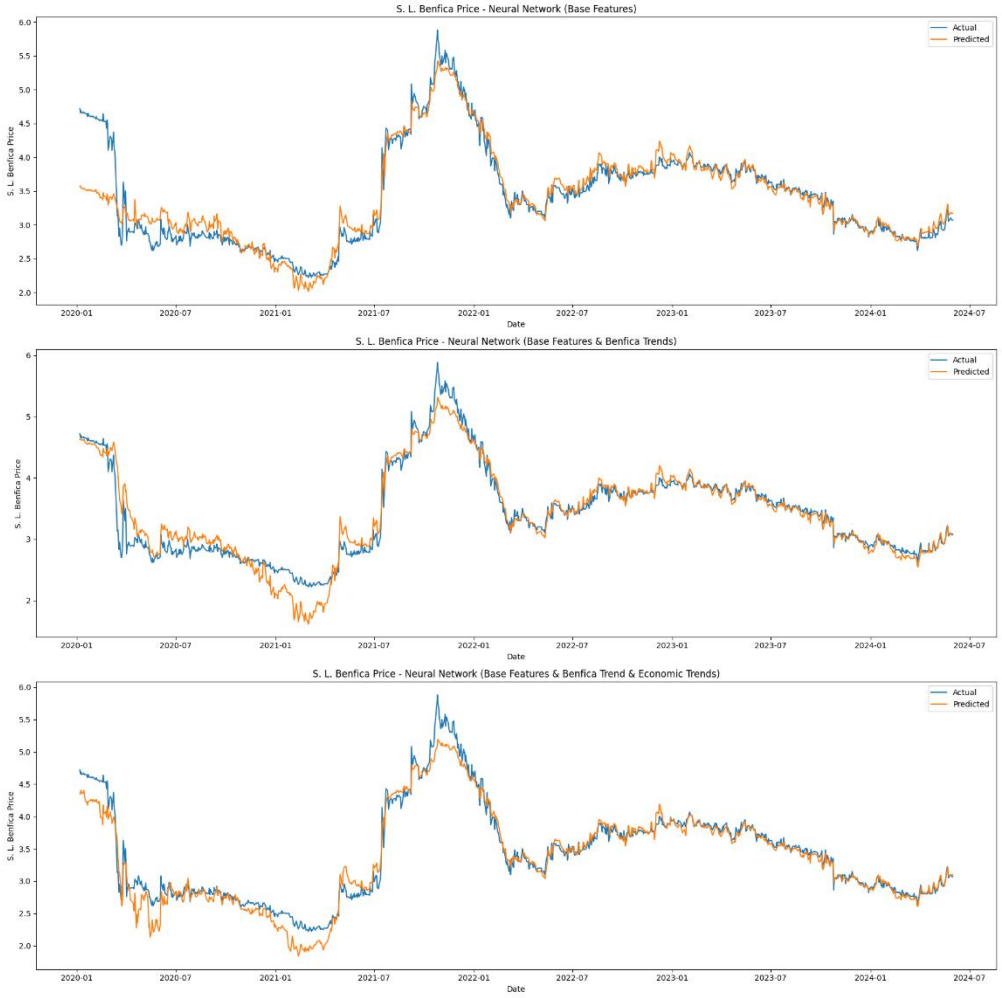

Figure 6.3 S. L. Benfica after hyper parameter tuning

6.2. SPORTING CLUBE DE PORTUGAL

The results achieved by 'Sporting Clube de Portugal', presented in table 6.4, for the Support Vector Regression and for the Neural Network, before any hyperparameter tuning were the following:

Table 6.4 Sporting C. P. before Hyperparameter Tuning – SVR vs Neural Network

	RMSE Before Hyperparameter Tuning		
	Base Features	Base Features + Sporting Trend	Base Features + Sporting Trend + Economic Trends
SVR	0.064	**0.063**	0.068
Neural Network	0.073	**0.058**	0.088

Using the SVR model the best results were obtained using the 'Base Features + Sporting Trend' with a RMSE of 0,063, followed by 'Base Features', and the worst results were obtained by 'Base Features + Sporting Trend + Economic Trends'.

The Neural Network model got the same results in terms of feature combinations. The best results were obtained using the 'Base Features + Sporting Trend' with a RMSE of 0,058, followed by 'Base Features', and the worst results we obtained with 'Base Features + Sporting Trend + Economic Trends'.

In this case the SVR got better results using 'Base features' and 'Base Features + Sporting Trend + Economic Trends', but on the best combination the Neural Network model got better results.

The results achieved by Sporting C. P., seen in table 6.5, focusing on Neural Networks, after the hyperparameter tuning were the following:

Table 6.5 Sporting C. P. After Hyperparameter Tuning – Neural Network

	RMSE After Hyperparameter Tuning		
	Base Features	Base Features + Sporting Trend	Base Features + Sporting Trend + Economic Trends
Neural Network	**0.027**	0.040	0.035

From a first view, the results achieved with hyperparameter tuning improved in all the combinations and can be seen in figure 6.4.

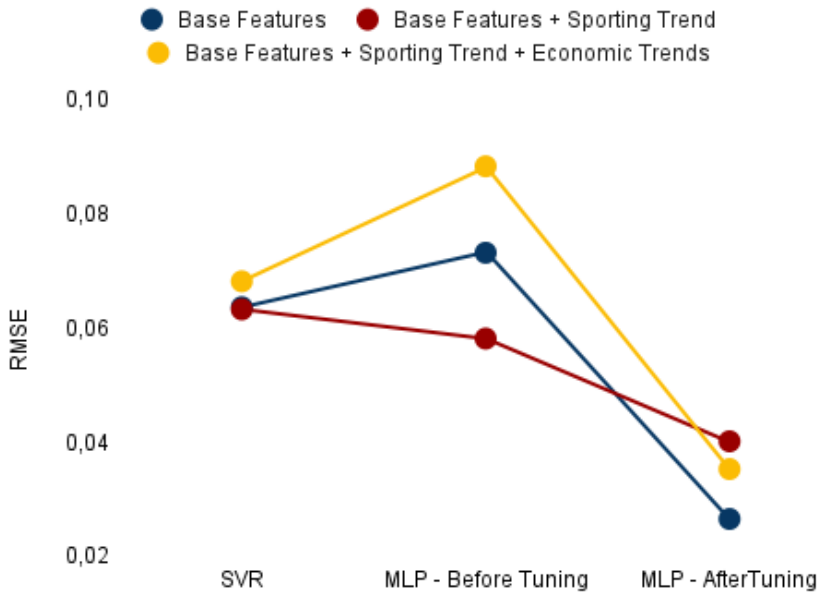

Figure 6.4 Sporting C. P. Models Comparison

The Multilayer Perceptron with 'Base Features' got the best results, achieving a RMSE of 0,027. The second-best result was from 'Base Features + Sporting Trend + Economic Trends' with a RMSE of 0,040, and the worst result was with the use of the 'Base features + Sporting Trend', achieving only a RMSE of 0,040.

To denote the great improvements after parameter tuning of the model using simple 'Base Features', that decreased the RMSE from 0,073 to 0,027.

Table 6.6 compares the best set of hyperparameters obtained for each model.

Table 6.6 Sporting C. P. Hyperparameter Tuning

	Activation Function	alpha	Hidden layers	Neurons	Learning Rate	Solver
Base Features	relu	0.01006064345328208	4	(22, 98, 43, 81)	constant	adam
Base Features + Sporting Trend	tanh	0.018052412368729155	4	(75, 86, 70, 62)	adaptative	adam
Base Features + Sporting Trend + Economic Trends	relu	0.018711067940704897	4	(115, 78, 95, 57)	constant	adam

It is unanimous that the number of layers used was 4, and the optimization algorithm was the adam.

Illustrated in figure 6.5, we can observe the actual values versus the predicted values for all the feature combinations, using a Multilayer Perceptron model.

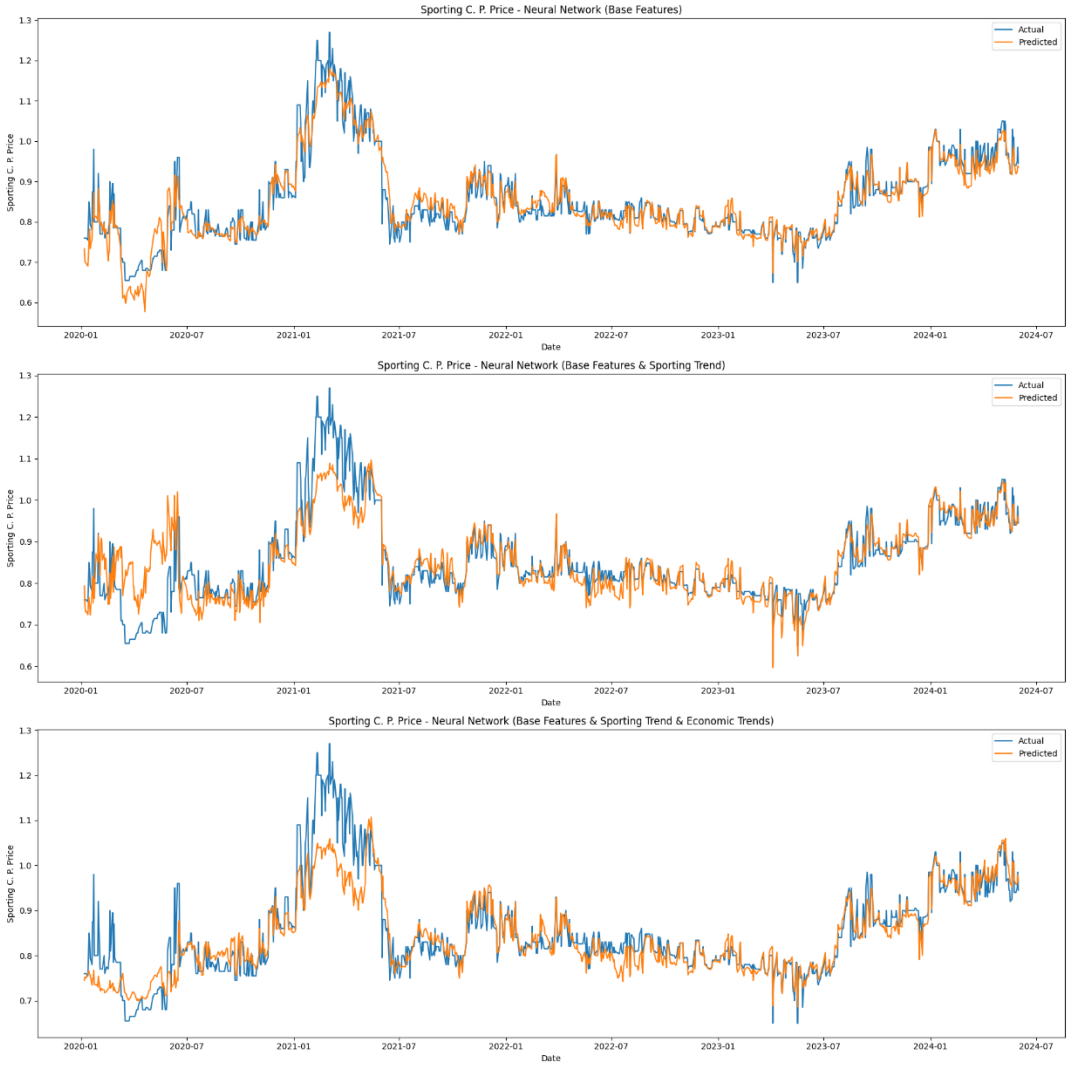

Figure 6.5 Sporting C. P. after hyper parameter tuning

6.3. FUTEBOL CLUBE DO PORTO

The results achieved by 'Futebol Clube do Porto', provided in table 6.7, for the Support Vector Regression and for the Neural Network, before any hyperparameter tuning were the following:

Table 6.7 F. C. Porto before Hyperparameter Tuning – SVR vs Neural Network

	RMSE Before Hyperparameter Tuning		
	Base Features	Base Features + Porto Trend	Base Features + Porto Trend + Economic Trends
SVR	0.094	**0.093**	0.094
Neural Network	0.086	**0.067**	0.092

Using the SVR model the best results were obtained using the 'Base Features + Porto Trend' achieving 0,093, followed by 'Base Features + Porto Trend + Economic Trends' and 'Base Features'. In this case the results are very similar across all the combinations.

With the Neural Network model, the lowest RMSE was obtained using the 'Base Features + Porto Trend' with the best in both models, followed by 'Base Features', and the worst results were obtained with the combination of 'Base Features + Porto Trend + Economic Trends'.

However, in all combination of features, the neural network model got better results.

Table 6.8 shows the results achieved by F. C. Porto, focusing on Neural Networks, after the hyperparameter tuning.

Table 6.8 F. C. Porto After Hyperparameter Tuning – Neural Network

	RMSE After Hyperparameter Tuning		
	Base Features	Base Features + Porto Trend	Base Features + Porto Trend + Economic Trends
Neural Network	0.053	**0.052**	0.056

From a first view in figure 6.6, the results achieved with hyperparameter tuning improved in all the combinations.

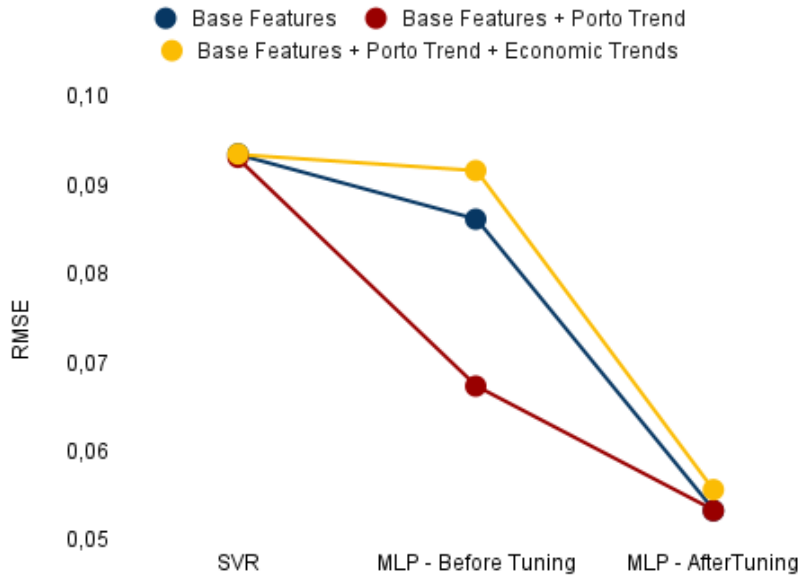

Figure 6.6 F. C. Porto Models Comparison

The Multilayer Perceptron with 'Base Features + Porto Trend' got the best results, achieving a RMSE of 0,052. The second-best result was from 'Base Features' with a RMSE of 0,053, and

the worst result was with the use of the 'Base Features + Porto Trend + Economic Trends', achieving only a RMSE of 0,055.

The best set of hyperparameters obtained for each model is represented in table 6.9.

Table 6.9 F. C. Porto Hyperparameter Tuning

	Activation Function	alpha	Hidden layers	Neurons	Learning Rate	Solver
Base Features	relu	0.026794162771515563	3	(118, 85, 116)	adaptive	adam
Base Features + Porto Trend	tanh	0.0606850157946487	3	(114, 74, 49)	constant	adam
Base Features + Porto Trend + Economic Trends	relu	0.022876421957307026	3	(55, 100, 64)	constant	adam

In F. C. Porto models, the number of hidden layers was 3 in all models and the optimization algorithm was the adam.

In figure 6.7 we can observe the actual values versus the predicted values for all the feature combinations, using a Multilayer Perceptron model.

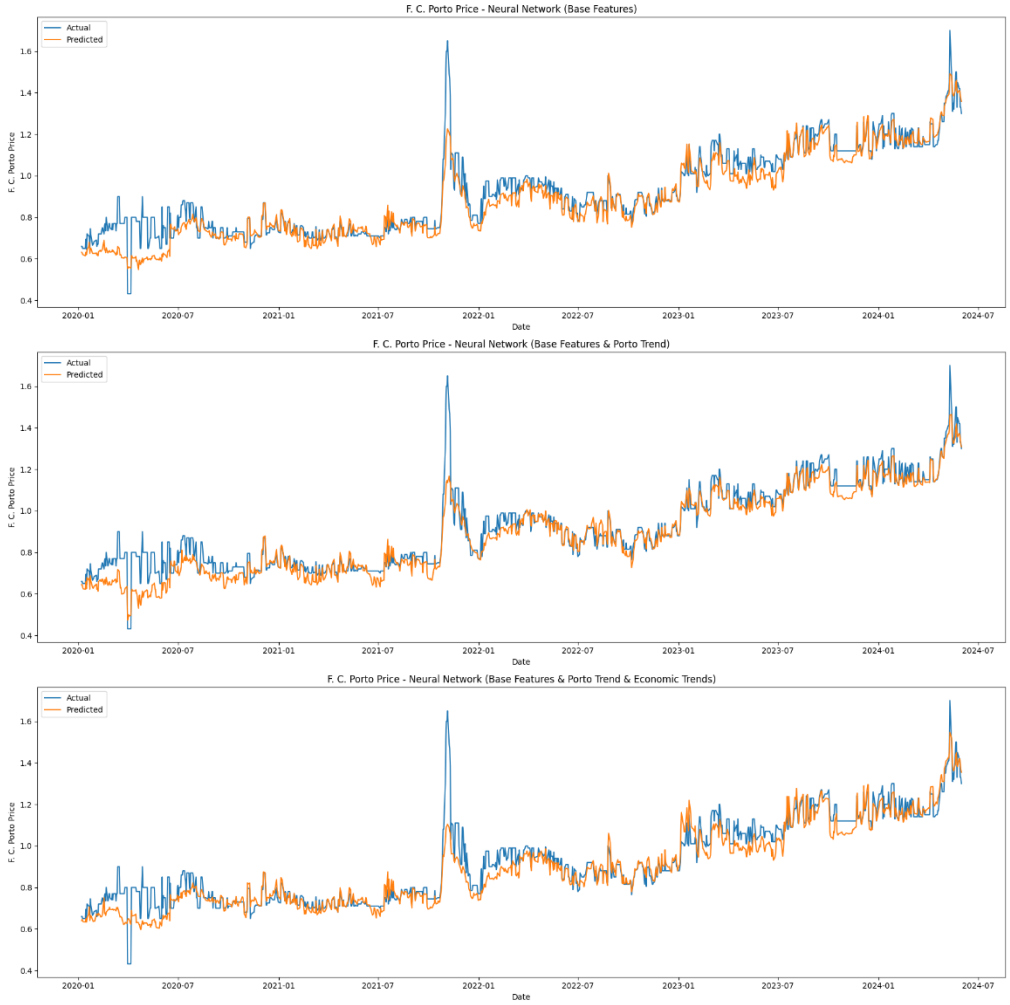

Figure 6.7 F. C. Porto after hyper parameter tuning

7. CONCLUSIONS

The main objective of this study was to understand the relationship and the predictive influence of google trends data with the stock market price of football clubs, in the Portuguese context.

It is important to underline that the improvements on the model's performance, were done in good predictive models, with a benchmark model of 10 features: oil price, gold price, euro to dollar exchange rate, 30 day window relative strength index, 10 and 20 day closing price simple moving average, 10 day window volatility, 30 day window commodity channel index, 1 day lag of closing price and 1 day lag of volume traded.

Also to keep in perspective we are measuring the predictive power of weekly variables in a daily target.

Before any hyperparameter tuning, there is the need to evaluate and compare the performance of the Support Vector Regression and the Neural Network, and to evaluate and compare the results achieved with different combination of features.

Prior to any tuning, simply applying the baseline models with the inclusion of the google search index of football clubs in the features, we achieved better predictions for all clubs and models. The combination of the base features with google search index of football clubs and of financial terms, did not proved to be an improvement in most cases, with exception of the SVR model for S. L. Benfica and F. C. Porto, maybe due to the complexity of the relationships.

Regarding model performance comparison, the Neural Network out of 9 models, it achieved a lower RMSE in 7. The Support Vector Regression only performed better for Sporting C. P., with the use of 'Base Features' and 'Base Features + Sporting Trend + Economic Trends'.

After the tunning, focusing on the Multilayer Perceptron, it is insightful to measure the improvements and the comparison between feature combinations.

The Multilayer Perceptron achieved improvements for all combination of features.

After tuning the number of layers, the number of neurons, the activation function, the alpha, the learning rate and the optimization function, from the three clubs, we achieved the best results for 'Sport Lisboa e Benfica' and for 'Futebol Clube do Porto' including Google Trends in the model. For 'Futebol Clube do Porto' we also included both the Porto trend and economic trends. These results are highlighted in table 7.1.

Table 7.1 Results per Club

Best Models	S. L. Benfica	Sporting C. P.	F. C. Porto
Base Features		**X**	

Base Features + Club Trend			**X**
Base Features + Club Trend + Economic Trends	**X**		

The best combination of features is not linear. It was different for every club, so it is always important to do an analysis on which combination gives us the best results.

The best machine learning model, that achieved greater results was the Multilayer Perceptron.

The findings in this study validate and extend previous research indicating that online search activity correlates with stock market behaviour, where Google search trends specific to Portuguese football clubs can significantly enhance the accuracy of stock price predictions. This finding bridges the gap by demonstrating that the predictive capabilities of Google Trends data, well-documented in broader stock market contexts, are also applicable to the niche domain of football clubs.

The results also show that while club-specific Google Trends data is valuable, Google searches related to general economic topics do not significantly improve predictive accuracy for football club stocks. This finding enriches the understanding of which types of search data are most relevant in the context of sport related investments. This might suggest that sentiment and public perception around football clubs play a more important role in shaping investor decisions and stock market outcomes, than the financial perception of the economy.

The consistent superiority of the Multilayer Perceptron over Support Vector Regression aligns with findings from recent studies advocating for advanced machine learning techniques in financial prediction. This reinforces the potential of neural networks to provide more accurate predictions compared to traditional models.

The study also demonstrates that the best feature combinations vary across different football clubs, emphasizing the need for tailored analysis. This insight is crucial for researchers focusing on the intersection of investor behaviour and sports investments, as it suggests that predictive models should be customized to the specific dynamics of each club.

In conclusion, this research enhances our understanding of how online search activity, sentiment analysis, and advanced machine learning contribute to predicting stock market behaviour in the unique context of football clubs. It emphasizes the need for tailored approaches in modelling and forecasting within the sports investment domain, thereby guiding future research and practical applications in financial analytics.

The limitations of this thesis include the relatively small market size of Portuguese football clubs, which may affect the robustness of the findings.

The study also does not account for other influential factors like management decisions or sponsorship deals, which could impact stock prices.

Additionally, the temporal mismatch between weekly Google Trends data and daily stock prices may limit predictive accuracy, which means that incorporating a weekly target variable could significantly improve its predictive power.

Despite these limitations, this study aims to provide a valuable assessment of using Google Trends data to predict the stock prices of Portuguese football clubs.

Future research could improve predictions by incorporating additional variables and more advanced techniques.

8. FUTURE WORK

Study the predictive power of google trends in the stock market of football clubs comparing different time horizons. The 2016-2017 award winning paper of the 'International Journal of Forecasting', conducted by D'Amuri and Marcucci, showed that Google-based models improve more consistently their out-of-sample forecasting performances for US unemployment, when it comes to a medium to long-term horizon. It would be interesting to see if this also applies to the stock market and even to the stock market of football clubs.

Another interesting topic of future work would be to implement other keywords that are representative of the social state in Portugal, not just the names of the football clubs or economic keywords. Ideally sentiment analysis extracted from Twitter, that gives an idea, not only of the public interest, but also of the public sentiment.

One last research topic would be to measure the predictive power of google trends on a weekly target, for example, the weekly returns of football clubs.

Lastly, it would also be interesting to compare the strength of google searches of football clubs between periods of positive price movements and negative price movements. Maybe they are more useful to predict negative price movements, when a club is facing difficulties.

BIBLIOGRAPHICAL REFERENCES

Água, J. (2014). *Exploring the Predictive Power of Google Searches over the US Stock Market. [Dissertação de mestrado, NSBE - UNL]. Repositório Universidade Nova. http://hdl.handle.net/10362/11694*

Bozanta, A., Coskun, M., Kutlu, B., & Ozturan, M. (2017). *RELATIONSHIP BETWEEN STOCK MARKET INDICES AND GOOGLE TRENDS*. In *The Online Journal of Science and Technology* (Vol. 7, Issue 4). https://tojsat.net/

Castellani, M., Pattitoni, P., & Patuelli, R. (2015). Abnormal Returns of Soccer Teams: Reassessing the Informational Value of Betting Odds. In *Journal of Sports Economics* (Vol. 16, Issue 7). https://doi.org/10.1177/1527002513505285

D'Amuri, F., & Marcucci, J. (2017). The predictive power of Google searches in forecasting US unemployment. *International Journal of Forecasting*, *33*(4), 801–816. https://doi.org/10.1016/j.ijforecast.2017.03.004

Farahani, M. & Hajiagha, S. (2021). Forecasting stock price using integrated artificial neural network and metaheuristic algorithms compared to time series models. Methodologies and Application. Springer. https://doi.org/10.1007/s00500-021-05775-5

Martínez, R., Bastos, F., & Amaral, C. (2020). Google trends as an explanatory variable of the football stocks index. In *Journal of Sports Economics & Management* (Vol. 10, Issue 1). http://sportsem.uv.es/j_sports_and_em/index.php/JSEM/article/view/173

Hagen, J. and Cunha, M. (2019). The History of Infesting in Football and Factors Affecting Stock Price of Listed Football Clubs. In International Journal of Financial Management. http://publishingindia.com/ijfm/

Htun, H., Biehl, M. & Petkov, N. (2023). In Financial Innovation Journal. Springer. https://doi.org/10.1186/s40854-022-00441-7

Hu, H., Tang, L., Zhang, S., & Wang, H. (2018). Predicting the direction of stock markets using optimized neural networks with Google Trends. *Neurocomputing*, *285*, 188–195. https://doi.org/10.1016/j.neucom.2018.01.038

Jha, S., Bhattacharya, & M., Bhattacharya. (2019). *Google Search Volume and Stock Market Liquidity*. In *Indian Journal of Finance.* https://doi.org/ 10.17010/ijf/2019/v13i8/146304

Katsafados, A. (2022). Stock Market Prediction: The Power of Machine Learning and Social Media. Eliva Press 978-1636485898. https://www.elivapress.com/en/authors/author-9284089831/

Lazer, D., Kennedy, R., King, G., & Vespignani, A. (2014). The Parable of Google Flu: Traps in Big Data Analysis. Science. https://doi.org/10.1126/science.1248506

Kumbure, M., Lohrmann, C., Luukka, P., & Porras, J. (2022). Machine learning techniques and data for stock market forecasting: A literature review. In *Expert Systems with Applications* (Vol. 197). Elsevier Ltd. https://doi.org/10.1016/j.eswa.2022.116659

Macukow, B. (2016). Neural Networks – State of Art, Brief History, Basic Models and Architecture. In K. Saeed, W. Homenda, (Eds.), Computer Information Systems and Industrial Management. Lecture Notes in Computer Science, vol 9842. Springer. https://doi.org/10.1007/978-3-319-45378-1_1

Maneejuk, P., & Yamaka, W. (2019). Predicting contagion from the US financial crisis to international stock markets using dynamic copula with google trends. *Mathematics*, *7*(11). https://doi.org/10.3390/math7111032

Morrow, S. (2023). *The People's Game? Football, Finance and Society*. Springer. https://doi.org/10.1057/9780230288393

Mutz, M. (2024). A new flagship of global football: the rise of global attention towards Saudi Arabia's pro league. *Frontiers in Sports and Active Living*, *6*. https://doi.org/10.3389/fspor.2024.1293751

Nti, I., Adekoya, A. & Weyori, B. (2020). Predicting Stock Market Price Movement Using Sentiment Analysis: Evidence from Ghana. Applied Computer Systems. https://doi.org/10.2478/acss-2020-0004

Nuti, S., Wayda, B., Ranasinghe, I., Wang, I., Dreyer, R., Chen, S., & Murugiah, K. (2014). The Use of Google Trends in Health Care Research: A Systematic Review. PLoS ONE 9(10): e109583. https://doi.org/10.1371/journal.pone.0109583

Petropoulos, A., Siakoulis, V., Stavroulakis, E., Lazaris, P., & Vlachogiannakis, N. (2021). Employing Google Trends and Deep Learning in Forecasting Financial Market Turbulence. *Journal of Behavioral Finance*, *23*(3), 353–365. https://doi.org/10.1080/15427560.2021.1913160

Prayoga, H., Dharma, F., & Sukmasari, D. (2022). The effect of sports performance and financial performance on European soccer club stock prices. *Asian Journal of Economics and Business Management*, *1*(2), 92–99. https://doi.org/10.53402/ajebm.v1i2.83

Preis, T., Moat, H. S., & Eugene Stanley, H. (2013). Quantifying trading behavior in financial markets using google trends. *Scientific Reports*, *3*. https://doi.org/10.1038/srep01684

Preis, T., Reith, D., & Stanley, H. E. (2010). Complex dynamics of our economic life on different scales: Insights from search engine query data. *Philosophical Transactions of*

the Royal Society A: Mathematical, Physical and Engineering Sciences, *368*(1933), 5707–5719. https://doi.org/10.1098/rsta.2010.0284

Priyatno, A. M., Ningsih, L., & Noor, M. (2024). Harnessing Machine Learning for Stock Price Prediction with Random Forest and Simple Moving Average Techniques. Journal of Engineering and Science Application, 1(1), 1–8. https://doi.org/10.69693/jesa.v1i1.1

Xu, Q., Bo, Z., Jiang, C. & Liu, Y. (2019). Does Google Search Index really help predicting stock market volatility? Evidence from a modified mixed data sampling model on volatility. Elsevier. https://doi.org/10.1016/j.knosys.2018.12.025

Renneboog, L. D. R., & Vanbrabant, P. (2000). Share Price Reactions to Sporty Performances of Soccer Clubs listed on the London Stock Exchange and the AIM. (CentER Discussion Paper; Vol. 2000-19). Finance. https://research.tilburguniversity.edu/en/publications/share-price-reactions-to-sporty-performances-of-soccer-clubs-list

Rosenblatt, F. (1958). THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN 1. In *Psychological Review* (Vol. 65, Issue 6). https://doi.org/10.1037/h0042519

Shahvaroughi Farahani, M., & Razavi Hajiagha, S. H. (2021). Forecasting stock price using integrated artificial neural network and metaheuristic algorithms compared to time series models. *Soft Computing*, *25*(13), 8483–8513. https://doi.org/10.1007/s00500-021-05775-5

Smola, A., Schölkopf, B. (2004). A tutorial on support vector regression. Statistics and Computing 14, 199–222. https://doi.org/10.1023/B:STCO.0000035301.49549.88

Takeda, F., & Wakao, T. (2014). *Google search intensity and its relationship with returns and trading volume of Japanese stocks*. https://doi.org/10.1016/j.pacfin.2014.01.003

Woloszko, N. (2020). *Tracking activity in real time with Google Trends*. OECD Economics Department Working Paper. https://doi.org/10.1787/6b9c7518-en

Wu, L. & Brynjolfsson, E. (2015). *The Future of Prediction: How Google Searches Foreshadow Housing Prices and Sales*. https://doi.org/10.7208/chicago/9780226206981.003.0003

APPENDIX A

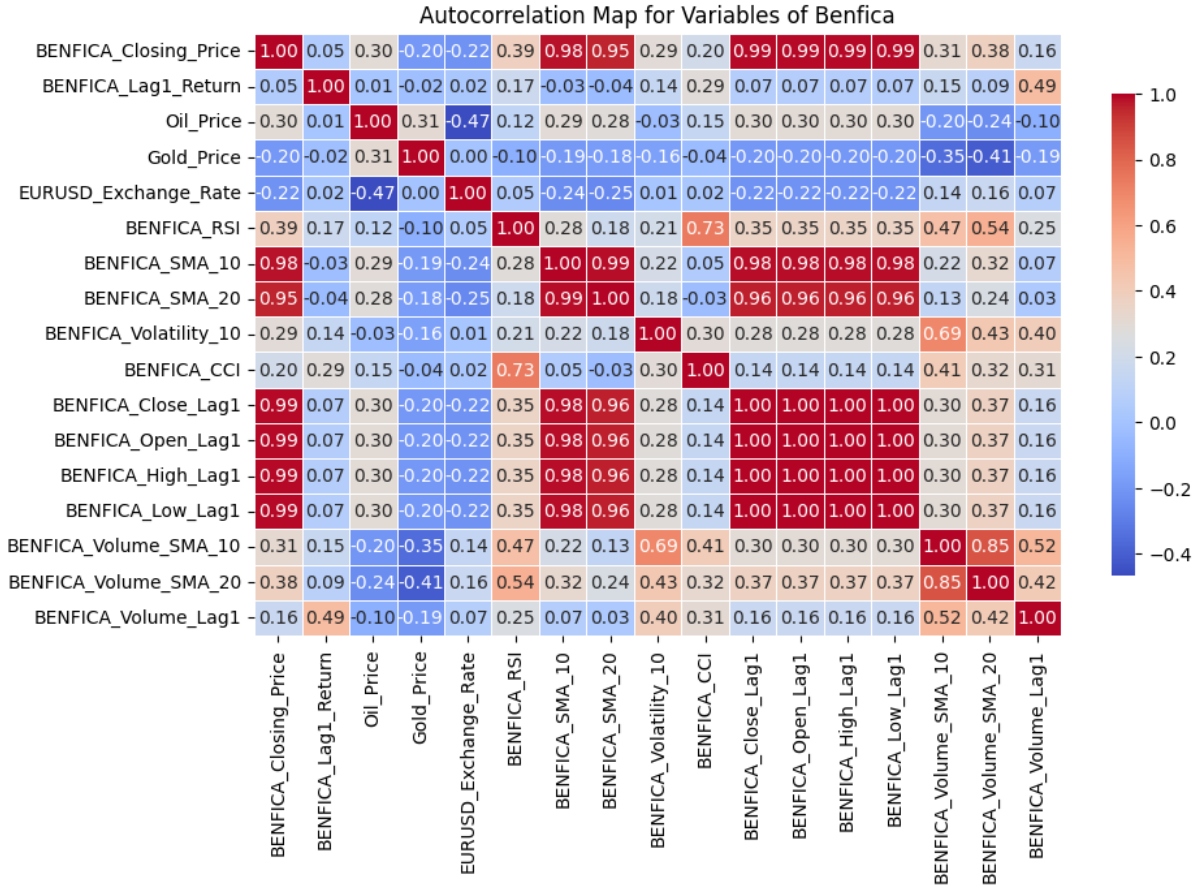

Figure A.1 – Benfica Autocorrelation Heatmap

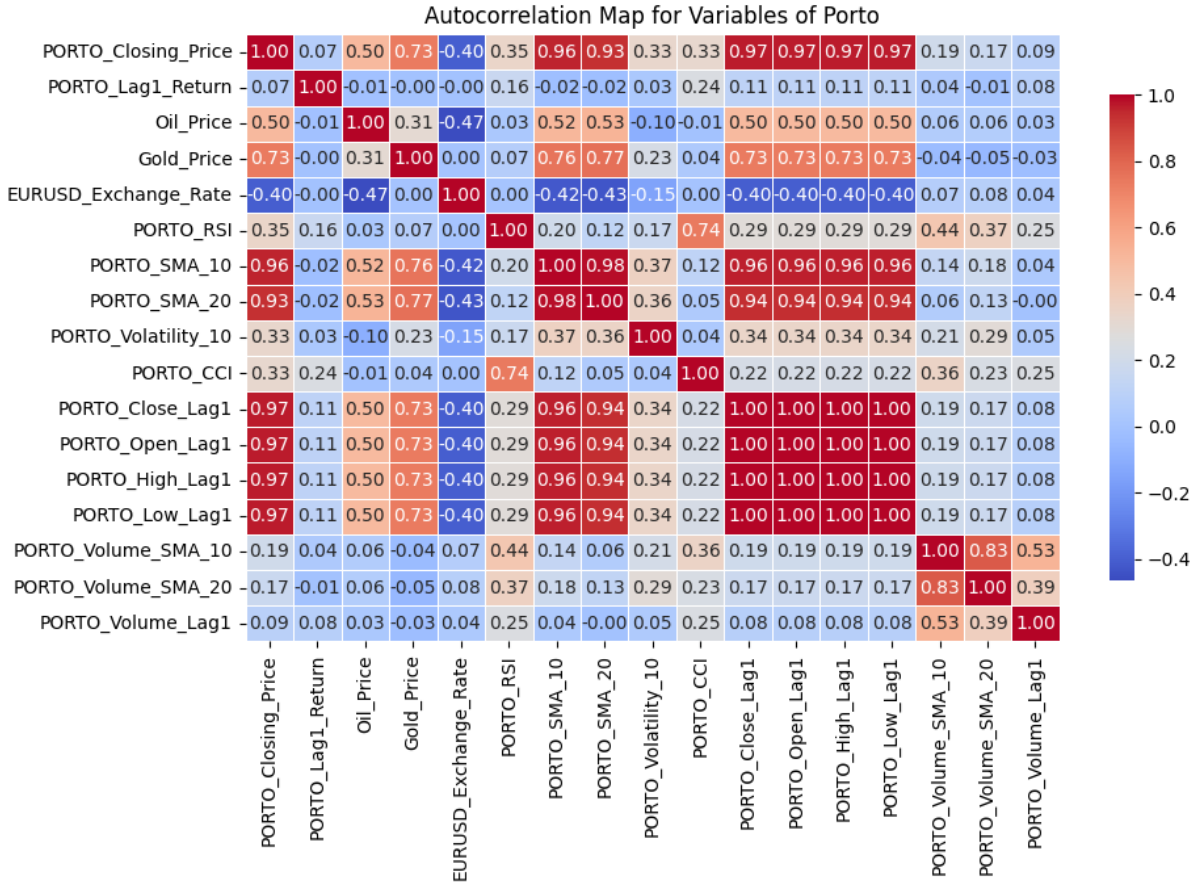

Figure A.2 – Porto Autocorrelation Heatmap

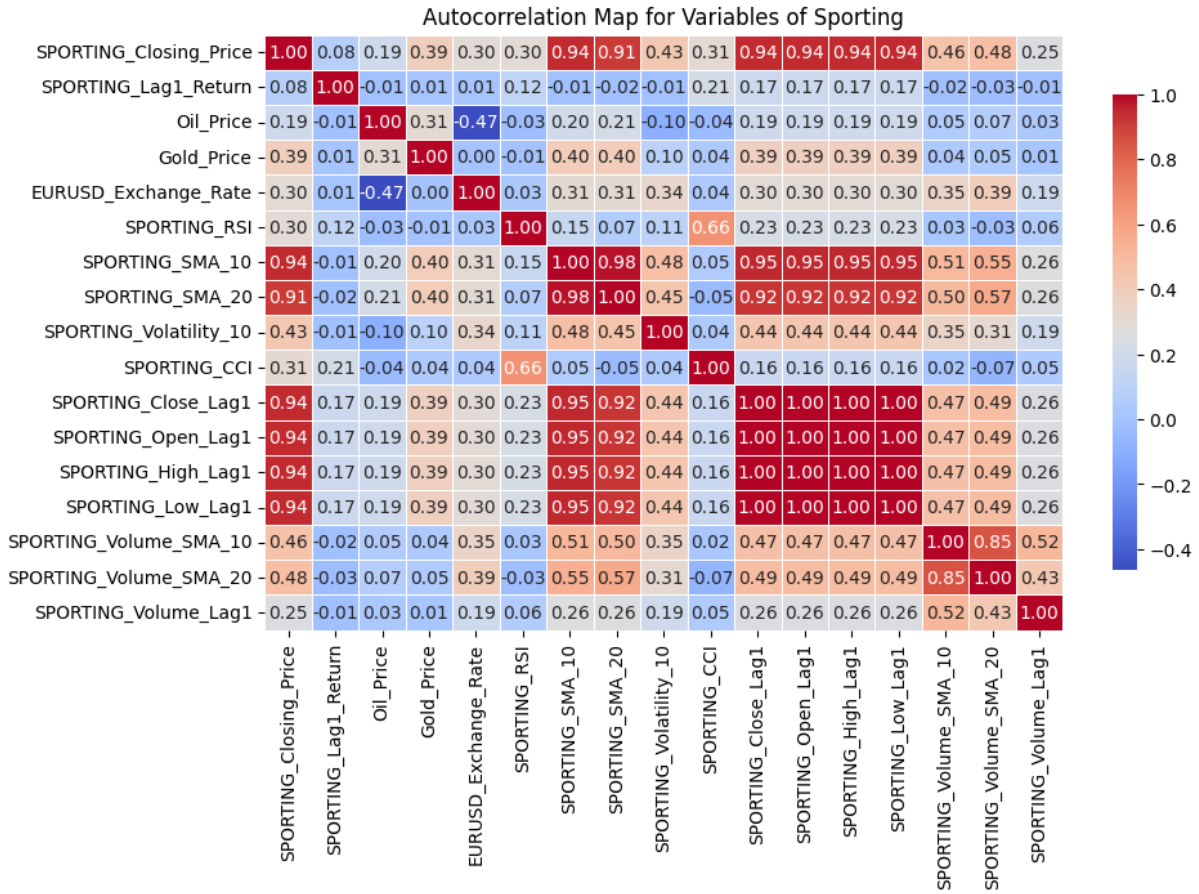

Figure A.3 – Sporting Autocorrelation Heatmap

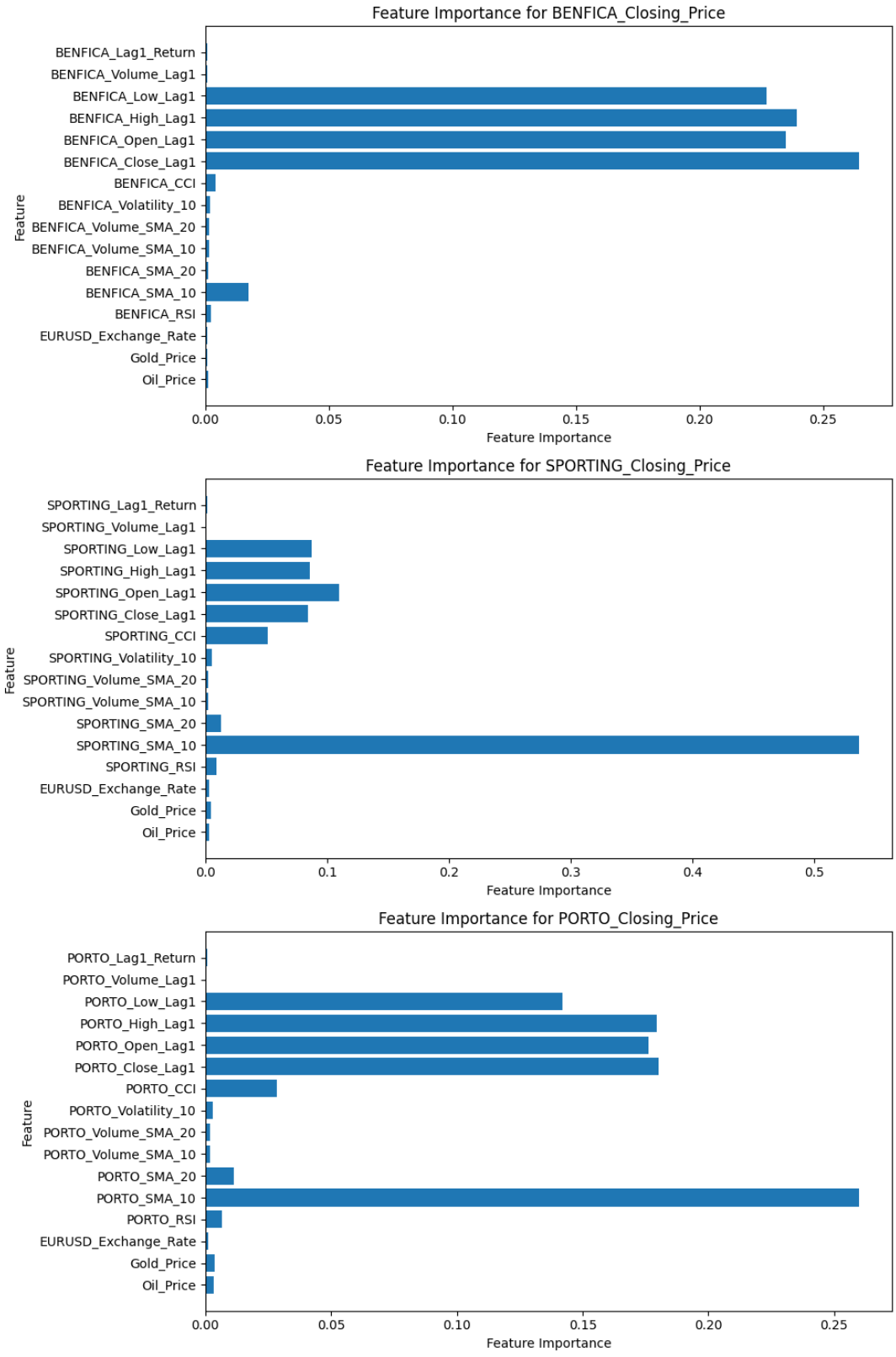

Figure A.4 – Feature Importance per Club

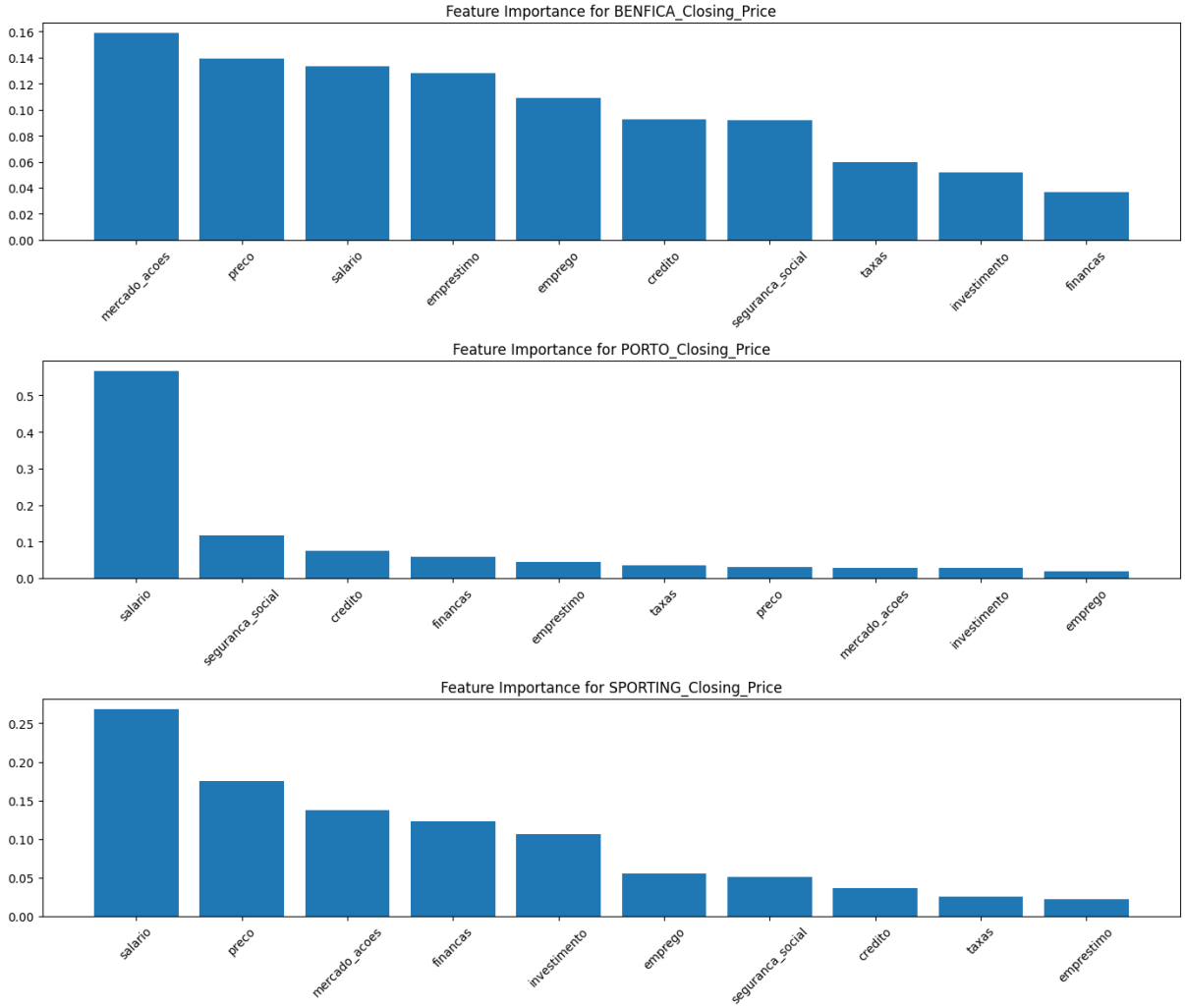

Figure A.5 – Feature Importance per Club for Economic Features

APPENDIX B

Google Colab Code

```
# -*- coding: utf-8 -*-
"""Final Data Set

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/1orCS7hz5shhw0UfwgO9BrKn-
H6ZIlA4U

# **1. Configurations**
"""

pip install yfinance

## Importing python packages

!pip install category_encoders

import warnings
warnings.filterwarnings('ignore')
import time
import graphviz
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
from scipy.stats import ttest_rel
from sklearn.model_selection import TimeSeriesSplit, GridSearchCV,
RandomizedSearchCV
from scipy.stats import ttest_rel, wilcoxon
from xgboost import XGBRegressor
from sklearn.neural_network import MLPRegressor
from scipy.stats import uniform, randint

"""# **2. Importing Data**"""

import pandas as pd
from datetime import datetime, timedelta
```

```
# Define start and end dates
start_date = datetime(2019, 1, 1)
end_date = datetime(2024, 6, 1)

# Generate date range
date_range = pd.date_range(start=start_date, end=end_date, freq='D')

# Create DataFrame with just dates
df = pd.DataFrame(date_range, columns=['Days'])

# Display DataFrame
print(df.tail())

# Read the CSV file into a DataFrame
data_path =
"https://raw.githubusercontent.com/JoaoMirandaJB/JB/main/Folha%20de%
20c%C3%A1lculo%20sem%20nome%20-%20Folha1.csv"

trends = pd.read_csv(data_path)

trends.head()

# Convert 'Week' column to datetime if it's not already
trends['Week'] = pd.to_datetime(trends['Week'])

# Create a single figure with three subplots
fig, axs = plt.subplots(3, 1, figsize=(14, 6), sharex=True)

# Plot the trend data for Benfica
axs[0].plot(trends['Week'], trends['BENFICA_trend'], color='red')
axs[0].set_ylabel('Trend Index')
axs[0].set_title('Google Trends for Benfica')

# Plot the trend data for Sporting
axs[1].plot(trends['Week'], trends['SPORTING_trend'], color='green')
axs[1].set_ylabel('Trend Index')
axs[1].set_title('Google Trends for Sporting')

# Plot the trend data for Porto
axs[2].plot(trends['Week'], trends['PORTO_trend'], color='blue')

axs[2].set_ylabel('Trend Index')
axs[2].set_title('Google Trends for Porto')

# Adjust layout
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Convert 'Week' column to datetime if it's not already
trends['Week'] = pd.to_datetime(trends['Week'])

# Create a single figure with three subplots
fig, axs = plt.subplots(3, 1, figsize=(14, 6), sharex=True)
```

```
# Plot the trend data for Benfica
axs[0].plot(trends['Week'], trends['preco'], color='black')
axs[0].set_ylabel('Trend Index')
axs[0].set_title('Google Trends for PreÃ§o')

# Plot the trend data for Sporting
axs[1].plot(trends['Week'], trends['mercado_acoes'], color='black')
axs[1].set_ylabel('Trend Index')
axs[1].set_title('Google Trends for AÃ§Ãµes')

# Plot the trend data for Porto
axs[2].plot(trends['Week'], trends['salario'], color='black')
axs[2].set_ylabel('Trend Index')
axs[2].set_title('Google Trends for SalÃ¡rio')

# Adjust layout
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Convert 'Week' to datetime
trends['Week'] = pd.to_datetime(trends['Week'])

# Generate daily trends DataFrame
daily_trends_list = []

for i, row in trends.iterrows():
    week_start = row['Week']
    for j in range(7):  # Loop over the 7 days of the week
        day = week_start + timedelta(days=j)
        daily_trends_list.append({
            'Days': day,
            'BENFICA_trend': row['BENFICA_trend'],
            'SPORTING_trend': row['SPORTING_trend'],
            'PORTO_trend': row['PORTO_trend'],
            'emprestimo': row['emprestimo'],
            'credito': row['credito'],
            'preco': row['preco'],
            'taxas': row['taxas'],
            'emprego': row['emprego'],
            'investimento': row['investimento'],
            'mercado_acoes': row['mercado_acoes'],
            'salario': row['salario'],
            'financas': row['financas'],
            'seguranca_social': row['seguranca_social'],
        })

daily_trends = pd.DataFrame(daily_trends_list)

# Merge the daily trends with df based on 'Days'
df = pd.merge(df, daily_trends, on='Days', how='left')

# Display the first few rows of the merged DataFrame
print(df.head(30))

# Convert the "Days" column to datetime format
```

```
df['Days'] = pd.to_datetime(df['Days'])

# Set the "Days" column as the index
df.set_index('Days', inplace=True)

# Get the minimum and maximum dates
min_date = df.index.min()
max_date = df.index.max()

print("Minimum Date:", min_date)
print("Maximum Date:", max_date)

"""## **2.1 Stock Market Data**"""

import yfinance as yf

# Define the ticker symbols and their corresponding names
ticker_symbols = {'SLBEN.LS': 'BENFICA', 'FCP.LS': 'PORTO',
'SCP.LS': 'SPORTING'}

# Fetch historical data for each stock and set the index to 'Date'
historical_data = {}
for symbol, name in ticker_symbols.items():
    stock_data = yf.download(symbol, start=min_date, end=max_date)
    stock_data.index.rename('Date', inplace=True)  # Rename index to
'Date'
    historical_data[name] = stock_data

# Join data to the original DataFrame based on index
for name, data in historical_data.items():
    # Extract required columns
    closing_price = data['Close']
    volume_traded = data['Volume']
    open_price = data['Open']
    high_price = data['High']
    low_price = data['Low']

    # Join data to original DataFrame
    df = df.join(closing_price.rename(f'{name}_Closing_Price'))
    df = df.join(volume_traded.rename(f'{name}_Volume'))
    df = df.join(closing_price.rename(f'{name}_Open_Price'))
    df = df.join(closing_price.rename(f'{name}_High_Price'))
    df = df.join(closing_price.rename(f'{name}_Low_Price'))

df.head()

"""## **2.2 Crude Oil and Gold Price**"""

# Extract historical data for WTI crude oil prices (symbol: CL=F)
oil_prices = yf.download("CL=F", start=min_date, end=max_date)

# Extract historical data for gold prices (symbol: GC=F)
gold_prices = yf.download("GC=F", start=min_date, end=max_date)

# Join oil prices to the original DataFrame based on index
df = df.join(oil_prices['Close'].rename('Oil_Price'))
```

```
# Join gold prices to the original DataFrame based on index
df = df.join(gold_prices['Close'].rename('Gold_Price'))

"""## **2.3 EU/US Exchange Rate**"""

# Extract historical data for Euro to US Dollar exchange rate
(symbol: EURUSD=X)
eurusd_exchange_rate = yf.download("EURUSD=X", start=min_date,
end=max_date)

# Join Euro to US Dollar exchange rate to the original DataFrame
based on index
df =
df.join(eurusd_exchange_rate['Close'].rename('EURUSD_Exchange_Rate')
)

"""# **3. Data Understanding**"""

# Define the names of the clubs
clubs = ['BENFICA', 'PORTO', 'SPORTING']

# Plot the Closing Price for each club
for club in clubs:
    column_name = f'{club}_Closing_Price'
    plt.figure(figsize=(10, 6))
    plt.plot(df.index, df[column_name])
    plt.title(f'Daily Closing Price for {club}')
    plt.xlabel('Date')
    plt.ylabel('Daily Closing Price')
    plt.grid(True)
    plt.show()

# Define the names of the clubs
clubs = ['BENFICA', 'PORTO', 'SPORTING']

# Plot the returns for each club
for club in clubs:
    column_name = f'{club}_Volume'
    plt.figure(figsize=(10, 6))
    plt.plot(df.index, df[column_name])
    plt.title(f'Daily Volume Traded for {club}')
    plt.xlabel('Date')
    plt.ylabel('Daily Volume')
    plt.grid(True)
    plt.show()

df.describe()

"""# **4. Handling Missing Values**

### **4.1. Target Variables**
"""

# Assuming your DataFrame is named 'df'
null_counts = df.isnull().sum()
```

```
print(null_counts)

# Assuming your DataFrame is named 'df'
columns_to_check = ['BENFICA_Closing_Price', 'PORTO_Closing_Price',
'SPORTING_Closing_Price']
df = df.dropna(subset=columns_to_check)

# Create a single figure with three subplots
fig, axs = plt.subplots(3, 1, figsize=(14, 6), sharex=True)

# Plot the closing prices for Benfica
axs[0].plot(df.index, df['BENFICA_Closing_Price'], color='red')
axs[0].set_ylabel('Closing Price')
axs[0].set_title('Benfica Closing Price')

# Plot the closing prices for Sporting
axs[1].plot(df.index, df['SPORTING_Closing_Price'], color='green')
axs[1].set_ylabel('Closing Price')
axs[1].set_title('Sporting Closing Price')

# Plot the closing prices for Porto
axs[2].plot(df.index, df['PORTO_Closing_Price'], color='blue')
axs[2].set_ylabel('Closing Price')
axs[2].set_title('Porto Closing Price')

# Adjust layout
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

"""### **4.2. Gold and Oil Price**"""

# Fill null values in 'Oil_Price' and 'Gold_Price' columns with the
last valid observation
df['Oil_Price'].fillna(method='ffill', inplace=True)
df['Gold_Price'].fillna(method='ffill', inplace=True)

"""### **4.3. EU/US Exchange Rate**"""

df['EURUSD_Exchange_Rate'].fillna(method='ffill', inplace=True)

"""# **5. Feature Engeneering**"""

tiv

"""## **5.1 Relative Strength Index**

The relative strength index (RSI) is a momentum indicator used in
technical analysis. RSI measures the speed and magnitude of a
security's recent price changes to evaluate overvalued or
undervalued conditions in the price of that security.
"""

rsi_dict = {}

for club in clubs:
```

```
    closing_prices = df[f'{club}_Closing_Price'].sort_index()  #
Ensure data is sorted by date

    if len(closing_prices) >= 30:  # Check if there are enough data
points
        price_changes = closing_prices.diff()
        gains = price_changes.where(price_changes > 0, 0)
        losses = -price_changes.where(price_changes < 0, 0)

        average_gains = gains.rolling(window=30).mean()
        average_losses = losses.rolling(window=30).mean()

        relative_strength = average_gains / average_losses
        rsi = 100 - (100 / (1 + relative_strength))

        rsi_dict[club] = rsi
        df[f'{club}_RSI'] = rsi

# Define the start date
start_date = '2019-07-29'

# Define the names of the clubs
clubs = ['BENFICA', 'PORTO', 'SPORTING']

# Plot the returns for each club
for club in clubs:
    column_name = f'{club}_RSI'
    plt.figure(figsize=(10, 6))
    plt.plot(df.index, df[column_name])
    plt.title(f'Daily RSI for {club}')
    plt.xlabel('Date')
    plt.ylabel('Daily RSI')
    plt.grid(True)

    # Get the index location of the start date
    start_index = df.index.get_loc(start_date)

    # Set the x-axis limits to start from the specified start date
    plt.xlim(df.index[start_index])

    plt.show()

"""## **5.2.1 Price Moving Average**

A simple moving average (SMA) calculates the average of a selected
range of prices, usually closing prices, by the number of periods in
that range. Moving averages are an important analytical tool used to
identify current price trends and the potential for a change in an
established trend.
"""

# Dictionary to store SMA for each club
sma_dict = {}

# Iterate over each club
for club in clubs:
```

```
    # Extract the closing prices for the club
    closing_prices = df[f'{club}_Closing_Price'].sort_index()  #
Ensure data is sorted by date

    # Calculate the 10-day simple moving average
    sma = closing_prices.rolling(window=10).mean()

    # Store the SMA in the dictionary
    sma_dict[club] = sma

    # Add SMA to df DataFrame
    df[f'{club}_SMA_10'] = sma

# Plot the returns for each club
for club in clubs:
    column_name = f'{club}_SMA_10'
    plt.figure(figsize=(10, 6))
    plt.plot(df.index, df[column_name])
    plt.title(f'Daily SMA for {club}')
    plt.xlabel('Date')
    plt.ylabel('Daily SMA')
    plt.grid(True)

    # Get the index location of the start date
    start_index = df.index.get_loc(start_date)

    # Set the x-axis limits to start from the specified start date
    plt.xlim(df.index[start_index])

    plt.show()

# Dictionary to store SMA for each club
sma_dict = {}

# Iterate over each club
for club in clubs:
    # Extract the closing prices for the club
    closing_prices = df[f'{club}_Closing_Price'].sort_index()  #
Ensure data is sorted by date

    # Calculate the 20-day simple moving average
    sma = closing_prices.rolling(window=20).mean()

    # Store the SMA in the dictionary
    sma_dict[club] = sma

    # Add SMA to df DataFrame
    df[f'{club}_SMA_20'] = sma

# Plot the returns for each club
for club in clubs:
    column_name = f'{club}_SMA_20'
    plt.figure(figsize=(10, 6))
    plt.plot(df.index, df[column_name])
    plt.title(f'Daily SMA for {club}')
    plt.xlabel('Date')
```

```
    plt.ylabel('Daily SMA')
    plt.grid(True)

    # Get the index location of the start date
    start_index = df.index.get_loc(start_date)

    # Set the x-axis limits to start from the specified start date
    plt.xlim(df.index[start_index])

    plt.show()

"""## **5.2.2 Volume Moving Average**"""

# Dictionary to store SMA for each club
sma_dict = {}

# Iterate over each club
for club in clubs:
    # Extract the _Volume for the club
    Volume = df[f'{club}_Volume'].sort_index()  # Ensure data is
sorted by date

    # Calculate the 10-day simple moving average
    sma = Volume.rolling(window=10).mean()

    # Store the SMA in the dictionary
    sma_dict[club] = sma

    # Add SMA to df DataFrame
    df[f'{club}_Volume_SMA_10'] = sma

# Plot the returns for each club
for club in clubs:
    column_name = f'{club}_Volume_SMA_10'
    plt.figure(figsize=(10, 6))
    plt.plot(df.index, df[column_name])
    plt.title(f'Daily SMA for {club}')
    plt.xlabel('Date')
    plt.ylabel('Daily SMA')
    plt.grid(True)

    # Get the index location of the start date
    start_index = df.index.get_loc(start_date)

    # Set the x-axis limits to start from the specified start date
    plt.xlim(df.index[start_index])

    plt.show()

# Dictionary to store SMA for each club
sma_dict = {}

# Iterate over each club
for club in clubs:
    # Extract the _Volume for the club
```

```
    Volume = df[f'{club}_Volume'].sort_index()  # Ensure data is
sorted by date

    # Calculate the 20day simple moving average
    sma = Volume.rolling(window=20).mean()

    # Store the SMA in the dictionary
    sma_dict[club] = sma

    # Add SMA to df DataFrame
    df[f'{club}_Volume_SMA_20'] = sma

"""## **5.3 Volatility**

Volatility measures the degree of variation or dispersion of price
for a financial instrument over a certain period of time.
It reflects the magnitude of price fluctuations, regardless of the
direction of those fluctuations.
Higher volatility implies larger price swings and greater
uncertainty about future price movements.
"""

# Dictionary to store volatility for each club
volatility_dict = {}

# Iterate over each club
for club in clubs:
    # Extract the closing prices for the club
    closing_prices = df[f'{club}_Closing_Price'].sort_index()  #
Ensure data is sorted by date

    # Calculate the price changes
    price_changes = closing_prices.diff()

    # Calculate the rolling standard deviation over a 10-day window
    volatility = price_changes.rolling(window=10).std()

    # Store the volatility in the dictionary
    volatility_dict[club] = volatility

    # Add volatility to df DataFrame
    df[f'{club}_Volatility_10'] = volatility

# Plot the returns for each club
for club in clubs:
    column_name = f'{club}_Volatility_10'
    plt.figure(figsize=(10, 6))
    plt.plot(df.index, df[column_name])
    plt.title(f'Daily Volatility for {club}')
    plt.xlabel('Date')
    plt.ylabel('Daily Volatility')
    plt.grid(True)

    # Get the index location of the start date
    start_index = df.index.get_loc(start_date)
```

```
    # Set the x-axis limits to start from the specified start date
    plt.xlim(df.index[start_index])

    plt.show()

"""## **5.4 Commodity Channel Index**"""

# Function to calculate Commodity Channel Index (CCI)
def calculate_cci(close, high, low, period=30):
    # Sort the data by index to ensure chronological order
    close = close.sort_index()
    high = high.sort_index()
    low = low.sort_index()

    # Calculate Typical Price (TP)
    tp = (high + low + close) / 3

    # Calculate Simple Moving Average (SMA) of Typical Price over
the specified period
    sma_tp = tp.rolling(window=period).mean()

    # Calculate Mean Deviation (MD)
    md = (tp - sma_tp).abs().rolling(window=period).mean()

    # Calculate Commodity Channel Index (CCI)
    cci = (tp - sma_tp) / (0.015 * md)

    return cci

# Calculate CCI for each club
for club in clubs:
    close = df[f'{club}_Closing_Price']
    high = df[f'{club}_High_Price']
    low = df[f'{club}_Low_Price']

    # Calculate CCI
    df[f'{club}_CCI'] = calculate_cci(close, high, low)

# Iterate through each club's CCI column
for club in clubs:
    cci_column = f"{club}_CCI"

    # Find null values in CCI column
    null_dates = df[df[cci_column].isnull()].index

    # Print club name and null dates
    print(f"Null values in CCI for {club}:")
    print(null_dates)
    print()

# Plot CCI for each club
for club in clubs:
    column_name = f'{club}_CCI'
    plt.figure(figsize=(10, 6))
    plt.plot(df.index, df[column_name], label=f'{club} CCI')
    plt.title(f'Commodity Channel Index for {club}')
```

```
    plt.xlabel('Date')
    plt.ylabel('CCI')
    plt.legend()
    plt.grid(True)

    # Optionally, set the x-axis limits to start from a specified
start date
    start_date = '2020-01-01'  # Adjust this date as needed
    if start_date in df.index:
        start_index = df.index.get_loc(start_date)
        plt.xlim(df.index[start_index:])

    plt.show()

"""## **5.5 Lags**"""

# Create lagged features and calculate CCI for each club
for club in clubs:
    # Shift the closing, opening, high, and low prices by 1 day to
create lagged features
    df[f'{club}_Close_Lag1'] =
df[f'{club}_Closing_Price'].sort_index().shift(1)
    df[f'{club}_Open_Lag1'] =
df[f'{club}_Open_Price'].sort_index().shift(1)
    df[f'{club}_High_Lag1'] =
df[f'{club}_High_Price'].sort_index().shift(1)
    df[f'{club}_Low_Lag1'] =
df[f'{club}_Low_Price'].sort_index().shift(1)
    df[f'{club}_Volume_Lag1'] =
df[f'{club}_Volume'].sort_index().shift(1)

df.head()

"""## **5.6 Return Lag**"""

# Initialize dictionary to store returns
returns_dict = {}
lagged_returns_dict = {}

# Calculate daily simple returns for each club
for club in clubs:
    closing_prices = df[f'{club}_Closing_Price'].sort_index()  #
Ensure data is sorted by date

    if len(closing_prices) >= 2:  # Check if there are enough data
points for returns calculation
        # Calculate simple returns
        simple_returns = closing_prices.pct_change()
        # Calculate lagged returns (lag of 1)
        lagged_returns = simple_returns.shift(1)

        # Store the returns in the dictionary
        returns_dict[club] = simple_returns
        lagged_returns_dict[club] = lagged_returns

        # Add the returns to the DataFrame
```

```
        df[f'{club}_Return'] = simple_returns
        # Add the returns to the DataFrame
        df[f'{club}_Lag1_Return'] = lagged_returns

# Plot the simple returns for each club in separate plots
for club in clubs:
    plt.figure(figsize=(12, 6))
    plt.plot(df.index, df[f'{club}_Return'], label=f'{club} Return')
    plt.title(f'Daily Returns for {club}')
    plt.xlabel('Date')
    plt.ylabel('Return')
    plt.legend()
    plt.grid(True)
    plt.show()

"""# **6. Data Treatment**

## **6.1 Define Time Frame - 5 seasons**
"""

# Limit the DataFrame from July 29, 2019, onwards
df = df['2019-07-29':]

"""## **6.2 Checking Data**"""

df.describe()

"""## **6.3 Scaling Data**"""

# List of target variables
target_variables = ['BENFICA_Closing_Price',
                    'PORTO_Closing_Price',
                    'SPORTING_Closing_Price',]

# List of columns to scale (excluding stock and target variables)
columns_to_scale = [col for col in df.columns if col not in
target_variables]

# Initialize StandardScaler
scaler = MinMaxScaler()

# Fit scaler to the data
scaler.fit(df[columns_to_scale])

# Transform the data
df_scaled = df.copy()
df_scaled[columns_to_scale] = scaler.transform(df[columns_to_scale])

df_scaled.describe()

df = df_scaled.copy()

"""# **7. Feature Importance**"""

# Define the features and target variables
```

```
features = ['emprestimo', 'credito', 'preco', 'taxas', 'emprego',
'investimento',
            'mercado_acoes', 'salario', 'financas',
'seguranca_social']
target_variables = ['BENFICA_Closing_Price', 'PORTO_Closing_Price',
'SPORTING_Closing_Price']

# Extract features and target variables from the DataFrame
X = df[features]
y_benfica = df['BENFICA_Closing_Price']
y_porto = df['PORTO_Closing_Price']
y_sporting = df['SPORTING_Closing_Price']

# Function to plot feature importance
def plot_feature_importance(importances, features, title):
    indices = np.argsort(importances)[::-1]
    plt.figure()
    plt.title(title)
    plt.bar(range(len(importances)), importances[indices],
align='center')
    plt.xticks(range(len(importances)), [features[i] for i in
indices], rotation=90)
    plt.tight_layout()
    plt.show()

# Initialize Random Forest models
rf_benfica = RandomForestRegressor(n_estimators=100,
random_state=42)
rf_porto = RandomForestRegressor(n_estimators=100, random_state=42)
rf_sporting = RandomForestRegressor(n_estimators=100,
random_state=42)

# Fit the models
rf_benfica.fit(X, y_benfica)
rf_porto.fit(X, y_porto)
rf_sporting.fit(X, y_sporting)

# Get feature importances
importances_benfica = rf_benfica.feature_importances_
importances_porto = rf_porto.feature_importances_
importances_sporting = rf_sporting.feature_importances_

# Plot feature importances
plot_feature_importance(importances_benfica, features, 'Feature
Importance for BENFICA Price')
plot_feature_importance(importances_porto, features, 'Feature
Importance for PORTO Price')
plot_feature_importance(importances_sporting, features, 'Feature
Importance for SPORTING Price')

# Define the features and target variables
features = ['emprestimo', 'credito', 'preco', 'taxas', 'emprego',
'investimento',
            'mercado_acoes', 'salario', 'financas',
'seguranca_social']
```

```
target_variables = ['BENFICA_Closing_Price', 'PORTO_Closing_Price',
'SPORTING_Closing_Price']

# Extract features and target variables from the DataFrame
X = df[features]
y = df[target_variables]

# Function to plot feature importance
def plot_feature_importance(importances, features, title, ax):
    indices = np.argsort(importances)[::-1]
    ax.set_title(title)
    ax.bar(range(len(importances)), importances[indices],
align='center')
    ax.set_xticks(range(len(importances)))
    ax.set_xticklabels([features[i] for i in indices], rotation=45)

# Initialize Random Forest models
rf = [RandomForestRegressor(n_estimators=100, random_state=42) for _
in range(len(target_variables))]

# Create a single figure with three subplots
fig, axs = plt.subplots(3, 1, figsize=(14, 12))

# Fit the models and plot feature importances
for i, (rf_model, target) in enumerate(zip(rf, target_variables)):
    rf_model.fit(X, y.iloc[:, i])
    importances = rf_model.feature_importances_
    plot_feature_importance(importances, features, f'Feature
Importance for {target}', axs[i])

# Adjust layout and show plots
plt.tight_layout()
plt.show()

benfica_variables = ['BENFICA_Closing_Price', 'BENFICA_Lag1_Return',
'Oil_Price', 'Gold_Price', 'EURUSD_Exchange_Rate', 'BENFICA_RSI',
'BENFICA_SMA_10', 'BENFICA_SMA_20', 'BENFICA_Volatility_10',
'BENFICA_CCI', 'BENFICA_Close_Lag1', 'BENFICA_Open_Lag1',
'BENFICA_High_Lag1', 'BENFICA_Low_Lag1','BENFICA_Volume_SMA_10',
'BENFICA_Volume_SMA_20','BENFICA_Volume_Lag1']
porto_variables = ['PORTO_Closing_Price', 'PORTO_Lag1_Return',
'Oil_Price', 'Gold_Price', 'EURUSD_Exchange_Rate', 'PORTO_RSI',
'PORTO_SMA_10', 'PORTO_SMA_20', 'PORTO_Volatility_10', 'PORTO_CCI',
'PORTO_Close_Lag1', 'PORTO_Open_Lag1', 'PORTO_High_Lag1',
'PORTO_Low_Lag1','PORTO_Volume_SMA_10',
'PORTO_Volume_SMA_20','PORTO_Volume_Lag1']
sporting_variables = ['SPORTING_Closing_Price',
'SPORTING_Lag1_Return', 'Oil_Price', 'Gold_Price',
'EURUSD_Exchange_Rate', 'SPORTING_RSI', 'SPORTING_SMA_10',
'SPORTING_SMA_20', 'SPORTING_Volatility_10', 'SPORTING_CCI',
'SPORTING_Close_Lag1', 'SPORTING_Open_Lag1', 'SPORTING_High_Lag1',
'SPORTING_Low_Lag1','SPORTING_Volume_SMA_10',
'SPORTING_Volume_SMA_20','SPORTING_Volume_Lag1']

# Compute autocorrelation for each set of exogenous variables
def plot_autocorrelation(exogenous_variables, title):
```

```
    plt.figure(figsize=(10, 6))
    plt.title(title)
    sns.heatmap(df[exogenous_variables].corr(), annot=True,
cmap='coolwarm', fmt=".2f", linewidths=0.5, cbar_kws={"shrink":
0.8})
    plt.show()

plot_autocorrelation(benfica_variables, 'Autocorrelation Map for
Variables of Benfica')
plot_autocorrelation(porto_variables, 'Autocorrelation Map for
Variables of Porto')
plot_autocorrelation(sporting_variables, 'Autocorrelation Map for
Variables of Sporting')

# Define a function to plot feature importance
def plot_feature_importance(features, importances, target_variable,
ax):
    ax.barh(features, importances)
    ax.set_xlabel('Feature Importance')
    ax.set_ylabel('Feature')
    ax.set_title(f'Feature Importance for {target_variable}')

# Define target variables
target_variables = ['BENFICA_Closing_Price',
'SPORTING_Closing_Price', 'PORTO_Closing_Price']

exogenous_variables_benfica = ['BENFICA_Lag1_Return', 'Oil_Price',
'Gold_Price', 'EURUSD_Exchange_Rate', 'BENFICA_RSI',
'BENFICA_SMA_10', 'BENFICA_SMA_20', 'BENFICA_Volatility_10',
'BENFICA_CCI', 'BENFICA_Close_Lag1', 'BENFICA_Open_Lag1',
'BENFICA_High_Lag1', 'BENFICA_Low_Lag1','BENFICA_Volume_SMA_10',
'BENFICA_Volume_SMA_20','BENFICA_Volume_Lag1']
exogenous_variables_porto = ['PORTO_Lag1_Return', 'Oil_Price',
'Gold_Price', 'EURUSD_Exchange_Rate', 'PORTO_RSI', 'PORTO_SMA_10',
'PORTO_SMA_20', 'PORTO_Volatility_10', 'PORTO_CCI',
'PORTO_Close_Lag1', 'PORTO_Open_Lag1', 'PORTO_High_Lag1',
'PORTO_Low_Lag1','PORTO_Volume_SMA_10',
'PORTO_Volume_SMA_20','PORTO_Volume_Lag1']
exogenous_variables_sporting = ['SPORTING_Lag1_Return', 'Oil_Price',
'Gold_Price', 'EURUSD_Exchange_Rate', 'SPORTING_RSI',
'SPORTING_SMA_10', 'SPORTING_SMA_20', 'SPORTING_Volatility_10',
'SPORTING_CCI', 'SPORTING_Close_Lag1', 'SPORTING_Open_Lag1',
'SPORTING_High_Lag1', 'SPORTING_Low_Lag1','SPORTING_Volume_SMA_10',
'SPORTING_Volume_SMA_20','SPORTING_Volume_Lag1']

# Initialize RandomForestRegressor
rf_selector = RandomForestRegressor(n_estimators=200,
random_state=42)

# Create a single figure with three subplots
fig, axs = plt.subplots(len(target_variables), 1, figsize=(10, 15))

# Iterate over target variables
for i, target_variable in enumerate(target_variables):
    # Select exogenous variables based on the club
    if 'BENFICA' in target_variable:
```

```
        exogenous_variables = exogenous_variables_benfica
    elif 'PORTO' in target_variable:
        exogenous_variables = exogenous_variables_porto
    elif 'SPORTING' in target_variable:
        exogenous_variables = exogenous_variables_sporting

    # Select features for current target variable
    features = [col for col in df.columns if col in
exogenous_variables]
    X = df[features]
    y = df[target_variable]

    # Fit random forest for feature selection
    rf_selector.fit(X, y)

    # Get feature importances
    feature_importances = rf_selector.feature_importances_

    # Plot feature importance
    plot_feature_importance(features, feature_importances,
target_variable, axs[i])

# Adjust layout and show plots
plt.tight_layout()
plt.show()

"""# **8. Compare Predictive Models**

## **8.1 Before Hyper Parameter Tuning Benfica**
"""

# Define target variable
target_variables = ['BENFICA_Closing_Price']

# Define exogenous variables for the target variable
exogenous_variables_price = ['Oil_Price', 'Gold_Price',
'EURUSD_Exchange_Rate',
                             'BENFICA_RSI', 'BENFICA_SMA_10',
'BENFICA_SMA_20', 'BENFICA_Volatility_10', 'BENFICA_CCI',
                             'BENFICA_Close_Lag1',
'BENFICA_Volume_Lag1'
                             ]

# Trend variables
trend_variables_benfica = ['BENFICA_trend']

# Define economic trend variables
economic_trend_variables = ['preco', 'mercado_acoes', 'salario']

# Combine exogenous variables and trend variables
selected_variables_with_trend_price = exogenous_variables_price +
trend_variables_benfica
selected_variables_with_all_trends_price = exogenous_variables_price
+ trend_variables_benfica + economic_trend_variables
selected_variables_without_trend_price = exogenous_variables_price
```

```
# Function to calculate RMSE and store predictions
def calculate_rmse_and_predictions(X_train, y_train, model):
    rmse_scores = []
    actual_values = np.full_like(y_train, np.nan)
    predicted_values = np.full_like(y_train, np.nan)

    for train_index, test_index in tscv.split(X_train):
        X_train_fold, X_test_fold = X_train.iloc[train_index],
X_train.iloc[test_index]
        y_train_fold, y_test_fold = y_train.iloc[train_index],
y_train.iloc[test_index]

        # Train the model
        model.fit(X_train_fold, y_train_fold)

        # Make predictions
        y_pred = model.predict(X_test_fold)

        # Store actual and predicted values
        actual_values[test_index] = y_test_fold
        predicted_values[test_index] = y_pred

        # Calculate RMSE
        rmse = mean_squared_error(y_test_fold, y_pred,
squared=False)
        rmse_scores.append(rmse)

    # Calculate average RMSE
    average_rmse = sum(rmse_scores) / len(rmse_scores)
    return average_rmse, actual_values, predicted_values

# Initialize models
models = {
    'SVR': SVR(),
    'Neural Network': MLPRegressor(max_iter=1000)
}

# Perform time series cross-validation
tscv = TimeSeriesSplit(n_splits=10)

# Dictionary to store RMSE results and predictions
rmse_results = {}
predictions = {}

# Assuming df is already defined with the necessary data
for target_variable in target_variables:
    # Extract target variable and selected features from df
    y_train = df[target_variable]

    rmse_results[target_variable] = {}
    predictions[target_variable] = {}

    for model_name, model in models.items():
        # Calculate RMSE with base features
        X_train_base = df[selected_variables_without_trend_price]
```

```
        average_rmse_base, actual_base, predicted_base =
calculate_rmse_and_predictions(X_train_base, y_train, model)

        # Calculate RMSE with trend variables
        X_train_with_trend = df[selected_variables_with_trend_price]
        average_rmse_with_trend, actual_with_trend,
predicted_with_trend =
calculate_rmse_and_predictions(X_train_with_trend, y_train, model)

        # Calculate RMSE with all trends (base + benfica trends +
economic trends)
        X_train_with_all_trends =
df[selected_variables_with_all_trends_price]
        average_rmse_with_all_trends, actual_with_all_trends,
predicted_with_all_trends =
calculate_rmse_and_predictions(X_train_with_all_trends, y_train,
model)

        # Store RMSE results
        rmse_results[target_variable][model_name] = {
            'base': average_rmse_base,
            'with_trend': average_rmse_with_trend,
            'with_all_trends': average_rmse_with_all_trends
        }

        # Store predictions
        predictions[target_variable][model_name] = {
            'base': (actual_base, predicted_base),
            'with_trend': (actual_with_trend, predicted_with_trend),
            'with_all_trends': (actual_with_all_trends,
predicted_with_all_trends)
        }

# Print RMSE results
for target_variable, model_rmse in rmse_results.items():
    print(f"Target Variable: {target_variable}")
    for model_name, rmse in model_rmse.items():
        print(f"Model: {model_name}")
        print(f"  Average RMSE with base features: {rmse['base']}")
        print(f"  Average RMSE with base features + Benfica trend:
{rmse['with_trend']}")
        print(f"  Average RMSE with base features + Benfica trend +
Economic trends: {rmse['with_all_trends']}\n")

# Define a dictionary to map original target variable names to
desired titles
title_mapping = {
    'BENFICA_Closing_Price': 'S. L. Benfica Price',
}

# Plot actual vs predicted values
dates = df.index

for target_variable, model_preds in predictions.items():
    for model_name, pred in model_preds.items():
        actual_base, predicted_base = pred['base']
```

```
        actual_with_trend, predicted_with_trend = pred['with_trend']
        actual_with_all_trends, predicted_with_all_trends =
pred['with_all_trends']

        plt.figure(figsize=(18, 18))

        plt.subplot(3, 1, 1)
        plt.plot(dates, actual_base, label='Actual')
        plt.plot(dates, predicted_base, label='Predicted')
        plt.title(f'{title_mapping[target_variable]} - {model_name}
(Base Features)')
        plt.xlabel('Date')
        plt.ylabel(title_mapping[target_variable])
        plt.legend()

        plt.subplot(3, 1, 2)
        plt.plot(dates, actual_with_trend, label='Actual')
        plt.plot(dates, predicted_with_trend, label='Predicted')
        plt.title(f'{title_mapping[target_variable]} - {model_name}
(Base Features & Benfica Trend)')
        plt.xlabel('Date')
        plt.ylabel(title_mapping[target_variable])
        plt.legend()

        plt.subplot(3, 1, 3)
        plt.plot(dates, actual_with_all_trends, label='Actual')
        plt.plot(dates, predicted_with_all_trends,
label='Predicted')
        plt.title(f'{title_mapping[target_variable]} - {model_name}
(Base Features & Benfica Trend & Economic Trends)')
        plt.xlabel('Date')
        plt.ylabel(title_mapping[target_variable])
        plt.legend()

        plt.tight_layout()
        plt.show()

"""## **8.2 After Hyper Parameter Tuning Benfica**"""

# Define target and feature variables
target_variables = ['BENFICA_Closing_Price']

# Define exogenous variables for the target variable
exogenous_variables_price = ['Oil_Price', 'Gold_Price',
'EURUSD_Exchange_Rate',
                             'BENFICA_RSI', 'BENFICA_SMA_10',
'BENFICA_SMA_20', 'BENFICA_Volatility_10', 'BENFICA_CCI',
                             'BENFICA_Close_Lag1',
'BENFICA_Volume_Lag1']

# Trend variables
trend_variables_benfica = ['BENFICA_trend']

# Define economic trend variables
economic_trend_variables = ['preco', 'mercado_acoes', 'salario']
```

```
# Combine exogenous variables, economic trend variables, and trend
variables for each target variable
selected_variables_with_trend_price = exogenous_variables_price +
trend_variables_benfica + economic_trend_variables
selected_variables_with_benfica_trend = exogenous_variables_price +
trend_variables_benfica
selected_variables_without_trend_price = exogenous_variables_price

# Function to calculate RMSE and store predictions
def calculate_rmse_and_predictions(X_train, y_train, model):
    rmse_scores = []
    actual_values = np.full(len(y_train), np.nan)
    predicted_values = np.full(len(y_train), np.nan)

    for train_index, test_index in tscv.split(X_train):
        X_train_fold, X_test_fold = X_train.iloc[train_index],
X_train.iloc[test_index]
        y_train_fold, y_test_fold = y_train.iloc[train_index],
y_train.iloc[test_index]

        # Train the model
        model.fit(X_train_fold, y_train_fold)

        # Make predictions
        y_pred = model.predict(X_test_fold)

        # Store actual and predicted values
        actual_values[test_index] = y_test_fold
        predicted_values[test_index] = y_pred

        # Calculate RMSE
        rmse = mean_squared_error(y_test_fold, y_pred,
squared=False)
        rmse_scores.append(rmse)

    # Calculate average RMSE
    average_rmse = sum(rmse_scores) / len(rmse_scores)
    return average_rmse, actual_values, predicted_values

# Function to generate random layer configurations with the number
of layers between 3 and 5
def generate_layer_configs(num_configs, min_layers=3, max_layers=5,
min_neurons=10, max_neurons=120):
    configs = []
    for _ in range(num_configs):
        num_layers = np.random.randint(min_layers, max_layers + 1)
        layers = tuple(np.random.randint(min_neurons, max_neurons +
1) for _ in range(num_layers))
        configs.append(layers)
    return configs

# Generate random layer configurations
layer_configs = generate_layer_configs(75)

# Define hyperparameter distributions
param_dist_nn = {
```

```
    'hidden_layer_sizes': layer_configs,
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': uniform(0.001, 0.1),
    'learning_rate': ['constant', 'adaptive']
}

# Initialize models
models = {
    'Neural Network': (MLPRegressor(max_iter=2000,
early_stopping=True, validation_fraction=0.1, n_iter_no_change=10),
param_dist_nn)
}

# Perform time series cross-validation
tscv = TimeSeriesSplit(n_splits=10)

# Dictionary to store RMSE results and predictions
rmse_results = {}
predictions = {}
best_params = {}

for target_variable in target_variables:
    # Extract target variable and selected features from df
    y_train = df[target_variable]

    rmse_results[target_variable] = {}
    predictions[target_variable] = {}
    best_params[target_variable] = {}

    for model_name, (model, param_dist) in models.items():
        # Calculate RMSE with trend variables for price
        X_train_with_all_trend =
df[selected_variables_with_trend_price]
        X_train_with_trend =
df[selected_variables_with_benfica_trend]
        X_train_without_trend =
df[selected_variables_without_trend_price]

        # Hyperparameter tuning with RandomizedSearchCV
        random_search_with_all_trend =
RandomizedSearchCV(estimator=model, param_distributions=param_dist,
n_iter=75, cv=tscv, scoring='neg_mean_squared_error', n_jobs=-1,
verbose=2, random_state=42)
        random_search_with_trend =
RandomizedSearchCV(estimator=model, param_distributions=param_dist,
n_iter=75, cv=tscv, scoring='neg_mean_squared_error', n_jobs=-1,
verbose=2, random_state=42)
        random_search_without_trend =
RandomizedSearchCV(estimator=model, param_distributions=param_dist,
n_iter=75, cv=tscv, scoring='neg_mean_squared_error', n_jobs=-1,
verbose=2, random_state=42)

        # Fit and predict with all trend variables
        random_search_with_all_trend.fit(X_train_with_all_trend,
y_train)
```

```
        best_model_with_all_trend =
random_search_with_all_trend.best_estimator_
        average_rmse_with_all_trend, actual_with_all_trend,
predicted_with_all_trend =
calculate_rmse_and_predictions(X_train_with_all_trend, y_train,
best_model_with_all_trend)

        # Fit and predict with trend variables
        random_search_with_trend.fit(X_train_with_trend, y_train)
        best_model_with_trend =
random_search_with_trend.best_estimator_
        average_rmse_with_trend, actual_with_trend,
predicted_with_trend =
calculate_rmse_and_predictions(X_train_with_trend, y_train,
best_model_with_trend)

        # Fit and predict without trend variables
        random_search_without_trend.fit(X_train_without_trend,
y_train)
        best_model_without_trend =
random_search_without_trend.best_estimator_
        average_rmse_without_trend, actual_without_trend,
predicted_without_trend =
calculate_rmse_and_predictions(X_train_without_trend, y_train,
best_model_without_trend)

        # Fit and predict without trend variables
        random_search_without_trend.fit(X_train_without_trend,
y_train)
        best_model_without_trend =
random_search_without_trend.best_estimator_
        average_rmse_without_trend, actual_without_trend,
predicted_without_trend =
calculate_rmse_and_predictions(X_train_without_trend, y_train,
best_model_without_trend)

        # Store RMSE results
        rmse_results[target_variable][model_name] = {
            'with_all_trend': average_rmse_with_all_trend,
            'with_trend': average_rmse_with_trend,
            'without_trend': average_rmse_without_trend
        }

        # Store predictions
        predictions[target_variable][model_name] = {
            'with_all_trend': (actual_with_all_trend,
predicted_with_all_trend),
            'with_trend': (actual_with_trend, predicted_with_trend),
            'without_trend': (actual_without_trend,
predicted_without_trend)
        }

        # Store best parameters
        best_params[target_variable][model_name] = {
            'with_all_trend':
random_search_with_all_trend.best_params_,
```

```
            'with_trend': random_search_with_trend.best_params_,
            'without_trend':
random_search_without_trend.best_params_
        }

# Print RMSE results and best parameters
for target_variable, model_rmse in rmse_results.items():
    print(f"Target Variable: {target_variable}")
    for model_name, rmse in model_rmse.items():
        print(f"Model: {model_name}")
        print(f"  Average RMSE with all trend variables:
{rmse['with_all_trend']}")
        print(f"  Best parameters with all trend:
{best_params[target_variable][model_name]['with_all_trend']}")
        print(f"  Average RMSE with trend variables:
{rmse['with_trend']}")
        print(f"  Best parameters with trend:
{best_params[target_variable][model_name]['with_trend']}")
        print(f"  Average RMSE without trend variables:
{rmse['without_trend']}")
        print(f"  Best parameters without trend:
{best_params[target_variable][model_name]['without_trend']}\n")

# Define a dictionary to map original target variable names to
desired titles
title_mapping = {
    'BENFICA_Closing_Price': 'S. L. Benfica Price',
}

# Plot actual vs predicted values
dates = df.index

for target_variable, model_preds in predictions.items():
    for model_name, pred in model_preds.items():
        actual_base, predicted_without_trend = pred['without_trend']
        actual_with_trend, predicted_with_trend = pred['with_trend']
        actual_with_all_trends, predicted_with_all_trend =
pred['with_all_trend']

        plt.figure(figsize=(18, 18))

        plt.subplot(3, 1, 1)
        plt.plot(dates, actual_base, label='Actual')
        plt.plot(dates, predicted_without_trend, label='Predicted')
        plt.title(f'{title_mapping[target_variable]} - {model_name}
(Base Features)')
        plt.xlabel('Date')
        plt.ylabel(title_mapping[target_variable])
        plt.legend()

        plt.subplot(3, 1, 2)
        plt.plot(dates, actual_with_trend, label='Actual')
        plt.plot(dates, predicted_with_trend, label='Predicted')
        plt.title(f'{title_mapping[target_variable]} - {model_name}
(Base Features & Benfica Trends)')
        plt.xlabel('Date')
```

```
        plt.ylabel(title_mapping[target_variable])
        plt.legend()

        plt.subplot(3, 1, 3)
        plt.plot(dates, actual_with_all_trends, label='Actual')
        plt.plot(dates, predicted_with_all_trend, label='Predicted')
        plt.title(f'{title_mapping[target_variable]} - {model_name}
(Base Features & Benfica Trend & Economic Trends)')
        plt.xlabel('Date')
        plt.ylabel(title_mapping[target_variable])
        plt.legend()

        plt.tight_layout()
        plt.show()

"""## **8.3 Before Hyper Parameter Tuning Sporting**"""

# Define target variable
target_variables = ['SPORTING_Closing_Price']

# Define exogenous variables for the target variable
exogenous_variables_price = ['Oil_Price', 'Gold_Price',
'EURUSD_Exchange_Rate',
                             'SPORTING_RSI', 'SPORTING_SMA_10',
'SPORTING_SMA_20', 'SPORTING_Volatility_10', 'SPORTING_CCI',
                             'SPORTING_Close_Lag1',
'SPORTING_Volume_Lag1']

# Trend variables
trend_variables_sporting = ['SPORTING_trend']

# Define economic trend variables
economic_trend_variables = ['preco', 'mercado_acoes', 'salario']

# Combine exogenous variables and trend variables
selected_variables_with_trend_price = exogenous_variables_price +
trend_variables_sporting
selected_variables_with_all_trends_price = exogenous_variables_price
+ trend_variables_sporting + economic_trend_variables
selected_variables_without_trend_price = exogenous_variables_price

# Function to calculate RMSE and store predictions
def calculate_rmse_and_predictions(X_train, y_train, model):
    rmse_scores = []
    actual_values = np.full_like(y_train, np.nan)
    predicted_values = np.full_like(y_train, np.nan)

    for train_index, test_index in tscv.split(X_train):
        X_train_fold, X_test_fold = X_train.iloc[train_index],
X_train.iloc[test_index]
        y_train_fold, y_test_fold = y_train.iloc[train_index],
y_train.iloc[test_index]

        # Train the model
        model.fit(X_train_fold, y_train_fold)
```

```
        # Make predictions
        y_pred = model.predict(X_test_fold)

        # Store actual and predicted values
        actual_values[test_index] = y_test_fold
        predicted_values[test_index] = y_pred

        # Calculate RMSE
        rmse = mean_squared_error(y_test_fold, y_pred,
squared=False)
        rmse_scores.append(rmse)

    # Calculate average RMSE
    average_rmse = sum(rmse_scores) / len(rmse_scores)
    return average_rmse, actual_values, predicted_values

# Initialize models
models = {
    'SVR': SVR(),
    'Neural Network': MLPRegressor(max_iter=2000)
}

# Perform time series cross-validation
tscv = TimeSeriesSplit(n_splits=10)

# Dictionary to store RMSE results and predictions
rmse_results = {}
predictions = {}

# Assuming df is already defined with the necessary data
for target_variable in target_variables:
    # Extract target variable and selected features from df
    y_train = df[target_variable]

    rmse_results[target_variable] = {}
    predictions[target_variable] = {}

    for model_name, model in models.items():
        # Calculate RMSE with base features
        X_train_base = df[selected_variables_without_trend_price]
        average_rmse_base, actual_base, predicted_base =
calculate_rmse_and_predictions(X_train_base, y_train, model)

        # Calculate RMSE with trend variables
        X_train_with_trend = df[selected_variables_with_trend_price]
        average_rmse_with_trend, actual_with_trend,
predicted_with_trend =
calculate_rmse_and_predictions(X_train_with_trend, y_train, model)

        # Calculate RMSE with all trends (base + Sporting trends +
economic trends)
        X_train_with_all_trends =
df[selected_variables_with_all_trends_price]
        average_rmse_with_all_trends, actual_with_all_trends,
predicted_with_all_trends =
```

```
calculate_rmse_and_predictions(X_train_with_all_trends, y_train,
model)

        # Store RMSE results
        rmse_results[target_variable][model_name] = {
            'base': average_rmse_base,
            'with_trend': average_rmse_with_trend,
            'with_all_trends': average_rmse_with_all_trends
        }

        # Store predictions
        predictions[target_variable][model_name] = {
            'base': (actual_base, predicted_base),
            'with_trend': (actual_with_trend, predicted_with_trend),
            'with_all_trends': (actual_with_all_trends,
predicted_with_all_trends)
        }

# Print RMSE results
for target_variable, model_rmse in rmse_results.items():
    print(f"Target Variable: {target_variable}")
    for model_name, rmse in model_rmse.items():
        print(f"Model: {model_name}")
        print(f"  Average RMSE with base features: {rmse['base']}")
        print(f"  Average RMSE with base features + Sporting trend:
{rmse['with_trend']}")
        print(f"  Average RMSE with base features + Sporting trend +
Economic trends: {rmse['with_all_trends']}\n")

# Define a dictionary to map original target variable names to
desired titles
title_mapping = {
    'SPORTING_Closing_Price': 'Sporting C. P. Price',
}

# Plot actual vs predicted values
dates = df.index

for target_variable, model_preds in predictions.items():
    for model_name, pred in model_preds.items():
        actual_base, predicted_base = pred['base']
        actual_with_trend, predicted_with_trend = pred['with_trend']
        actual_with_all_trends, predicted_with_all_trends =
pred['with_all_trends']

        plt.figure(figsize=(18, 18))

        plt.subplot(3, 1, 1)
        plt.plot(dates, actual_base, label='Actual')
        plt.plot(dates, predicted_base, label='Predicted')
        plt.title(f'{title_mapping[target_variable]} - {model_name}
(Base Features)')
        plt.xlabel('Date')
        plt.ylabel(title_mapping[target_variable])
        plt.legend()
```

```
        plt.subplot(3, 1, 2)
        plt.plot(dates, actual_with_trend, label='Actual')
        plt.plot(dates, predicted_with_trend, label='Predicted')
        plt.title(f'{title_mapping[target_variable]} - {model_name}
(Base Features & Sporting Trend)')
        plt.xlabel('Date')
        plt.ylabel(title_mapping[target_variable])
        plt.legend()

        plt.subplot(3, 1, 3)
        plt.plot(dates, actual_with_all_trends, label='Actual')
        plt.plot(dates, predicted_with_all_trends,
label='Predicted')
        plt.title(f'{title_mapping[target_variable]} - {model_name}
(Base Features & Sporting Trend & Economic Trends)')
        plt.xlabel('Date')
        plt.ylabel(title_mapping[target_variable])
        plt.legend()

        plt.tight_layout()
        plt.show()

"""## **8.4 After Hyper Parameter Tuning Sporting**"""

# Define target variable
target_variables = ['SPORTING_Closing_Price']

# Define exogenous variables for the target variable
exogenous_variables_price = ['Oil_Price', 'Gold_Price',
'EURUSD_Exchange_Rate',
                             'SPORTING_RSI', 'SPORTING_SMA_10',
'SPORTING_SMA_20', 'SPORTING_Volatility_10', 'SPORTING_CCI',
                             'SPORTING_Close_Lag1',
                             'SPORTING_Volume_Lag1']

# Trend variables
trend_variables_sporting = ['SPORTING_trend']

# Define economic trend variables
economic_trend_variables = ['preco', 'mercado_acoes', 'salario']

# Combine exogenous variables, economic trend variables, and trend
variables for each target variable
selected_variables_with_trend_price = exogenous_variables_price +
trend_variables_sporting + economic_trend_variables
selected_variables_with_sporting_trend = exogenous_variables_price +
trend_variables_sporting
selected_variables_without_trend_price = exogenous_variables_price

# Function to calculate RMSE and store predictions
def calculate_rmse_and_predictions(X_train, y_train, model):
    rmse_scores = []
    actual_values = np.full(len(y_train), np.nan)
    predicted_values = np.full(len(y_train), np.nan)

    for train_index, test_index in tscv.split(X_train):
```

```
        X_train_fold, X_test_fold = X_train.iloc[train_index],
X_train.iloc[test_index]
        y_train_fold, y_test_fold = y_train.iloc[train_index],
y_train.iloc[test_index]

        # Train the model
        model.fit(X_train_fold, y_train_fold)

        # Make predictions
        y_pred = model.predict(X_test_fold)

        # Store actual and predicted values
        actual_values[test_index] = y_test_fold
        predicted_values[test_index] = y_pred

        # Calculate RMSE
        rmse = mean_squared_error(y_test_fold, y_pred,
squared=False)
        rmse_scores.append(rmse)

    # Calculate average RMSE
    average_rmse = sum(rmse_scores) / len(rmse_scores)
    return average_rmse, actual_values, predicted_values

# Function to generate random layer configurations with the number
of layers between 3 and 5
def generate_layer_configs(num_configs, min_layers=3, max_layers=5,
min_neurons=10, max_neurons=120):
    configs = []
    for _ in range(num_configs):
        num_layers = np.random.randint(min_layers, max_layers + 1)
        layers = tuple(np.random.randint(min_neurons, max_neurons +
1) for _ in range(num_layers))
        configs.append(layers)
    return configs

# Generate random layer configurations
layer_configs = generate_layer_configs(75)

# Define hyperparameter distributions
param_dist_nn = {
    'hidden_layer_sizes': layer_configs,
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': uniform(0.001, 0.1),
    'learning_rate': ['constant', 'adaptive']
}

# Initialize models
models = {
    'Neural Network': (MLPRegressor(max_iter=2000,
early_stopping=True, validation_fraction=0.1, n_iter_no_change=10),
param_dist_nn)
}

# Perform time series cross-validation
```

```
tscv = TimeSeriesSplit(n_splits=10)

# Dictionary to store RMSE results and predictions
rmse_results = {}
predictions = {}
best_params = {}

for target_variable in target_variables:
    # Extract target variable and selected features from df
    y_train = df[target_variable]

    rmse_results[target_variable] = {}
    predictions[target_variable] = {}
    best_params[target_variable] = {}

    for model_name, (model, param_dist) in models.items():
        # Calculate RMSE with trend variables for price
        X_train_with_all_trend =
df[selected_variables_with_trend_price]
        X_train_with_trend =
df[selected_variables_with_sporting_trend]
        X_train_without_trend =
df[selected_variables_without_trend_price]

        # Hyperparameter tuning with RandomizedSearchCV
        random_search_with_all_trend =
RandomizedSearchCV(estimator=model, param_distributions=param_dist,
n_iter=75, cv=tscv, scoring='neg_mean_squared_error', n_jobs=-1,
verbose=2, random_state=42)
        random_search_with_trend =
RandomizedSearchCV(estimator=model, param_distributions=param_dist,
n_iter=75, cv=tscv, scoring='neg_mean_squared_error', n_jobs=-1,
verbose=2, random_state=42)
        random_search_without_trend =
RandomizedSearchCV(estimator=model, param_distributions=param_dist,
n_iter=75, cv=tscv, scoring='neg_mean_squared_error', n_jobs=-1,
verbose=2, random_state=42)

        # Fit and predict with all trend variables
        random_search_with_all_trend.fit(X_train_with_all_trend,
y_train)
        best_model_with_all_trend =
random_search_with_all_trend.best_estimator_
        average_rmse_with_all_trend, actual_with_all_trend,
predicted_with_all_trend =
calculate_rmse_and_predictions(X_train_with_all_trend, y_train,
best_model_with_all_trend)

        # Fit and predict with trend variables
        random_search_with_trend.fit(X_train_with_trend, y_train)
        best_model_with_trend =
random_search_with_trend.best_estimator_
        average_rmse_with_trend, actual_with_trend,
predicted_with_trend =
calculate_rmse_and_predictions(X_train_with_trend, y_train,
best_model_with_trend)
```

```
        # Fit and predict without trend variables
        random_search_without_trend.fit(X_train_without_trend,
y_train)
        best_model_without_trend =
random_search_without_trend.best_estimator_
        average_rmse_without_trend, actual_without_trend,
predicted_without_trend =
calculate_rmse_and_predictions(X_train_without_trend, y_train,
best_model_without_trend)

        # Fit and predict without trend variables
        random_search_without_trend.fit(X_train_without_trend,
y_train)
        best_model_without_trend =
random_search_without_trend.best_estimator_
        average_rmse_without_trend, actual_without_trend,
predicted_without_trend =
calculate_rmse_and_predictions(X_train_without_trend, y_train,
best_model_without_trend)

        # Store RMSE results
        rmse_results[target_variable][model_name] = {
            'with_all_trend': average_rmse_with_all_trend,
            'with_trend': average_rmse_with_trend,
            'without_trend': average_rmse_without_trend
        }

        # Store predictions
        predictions[target_variable][model_name] = {
            'with_all_trend': (actual_with_all_trend,
predicted_with_all_trend),
            'with_trend': (actual_with_trend, predicted_with_trend),
            'without_trend': (actual_without_trend,
predicted_without_trend)
        }

        # Store best parameters
        best_params[target_variable][model_name] = {
            'with_all_trend':
random_search_with_all_trend.best_params_,
            'with_trend': random_search_with_trend.best_params_,
            'without_trend':
random_search_without_trend.best_params_
        }

# Print RMSE results and best parameters
for target_variable, model_rmse in rmse_results.items():
    print(f"Target Variable: {target_variable}")
    for model_name, rmse in model_rmse.items():
        print(f"Model: {model_name}")
        print(f"  Average RMSE with all trend variables:
{rmse['with_all_trend']}")
        print(f"  Best parameters with all trend:
{best_params[target_variable][model_name]['with_all_trend']}")
```

```
        print(f"  Average RMSE with trend variables:
{rmse['with_trend']}")
        print(f"  Best parameters with trend:
{best_params[target_variable][model_name]['with_trend']}")
        print(f"  Average RMSE without trend variables:
{rmse['without_trend']}")
        print(f"  Best parameters without trend:
{best_params[target_variable][model_name]['without_trend']}\n")

# Define a dictionary to map original target variable names to
desired titles
title_mapping = {
    'SPORTING_Closing_Price': 'Sporting C. P. Price',
}

# Plot actual vs predicted values
dates = df.index

for target_variable, model_preds in predictions.items():
    for model_name, pred in model_preds.items():
        actual_base, predicted_without_trend = pred['without_trend']
        actual_with_trend, predicted_with_trend = pred['with_trend']
        actual_with_all_trends, predicted_with_all_trend =
pred['with_all_trend']

        plt.figure(figsize=(18, 18))

        plt.subplot(3, 1, 1)
        plt.plot(dates, actual_base, label='Actual')
        plt.plot(dates, predicted_without_trend, label='Predicted')
        plt.title(f'{title_mapping[target_variable]} - {model_name}
(Base Features)')
        plt.xlabel('Date')
        plt.ylabel(title_mapping[target_variable])
        plt.legend()

        plt.subplot(3, 1, 2)
        plt.plot(dates, actual_with_trend, label='Actual')
        plt.plot(dates, predicted_with_trend, label='Predicted')
        plt.title(f'{title_mapping[target_variable]} - {model_name}
(Base Features & Sporting Trend)')
        plt.xlabel('Date')
        plt.ylabel(title_mapping[target_variable])
        plt.legend()

        plt.subplot(3, 1, 3)
        plt.plot(dates, actual_with_all_trends, label='Actual')
        plt.plot(dates, predicted_with_all_trend, label='Predicted')
        plt.title(f'{title_mapping[target_variable]} - {model_name}
(Base Features & Sporting Trend & Economic Trends)')
        plt.xlabel('Date')
        plt.ylabel(title_mapping[target_variable])
        plt.legend()

        plt.tight_layout()
        plt.show()
```

```
"""## **8.5 Before Hyper Parameter Tuning Porto**"""

# Define target variable
target_variables = ['PORTO_Closing_Price']

# Define exogenous variables for the target variable
exogenous_variables_price = ['Oil_Price', 'Gold_Price',
'EURUSD_Exchange_Rate',
                             'PORTO_RSI', 'PORTO_SMA_10',
'PORTO_SMA_20', 'PORTO_Volatility_10', 'PORTO_CCI',
                             'PORTO_Close_Lag1',
'PORTO_Volume_Lag1']

# Trend variables
trend_variables_porto = ['PORTO_trend']

# Define economic trend variables
economic_trend_variables = ['preco', 'mercado_acoes', 'salario']

# Combine exogenous variables and trend variables
selected_variables_with_trend_price = exogenous_variables_price +
trend_variables_porto
selected_variables_with_all_trends_price = exogenous_variables_price
+ trend_variables_porto + economic_trend_variables
selected_variables_without_trend_price = exogenous_variables_price

# Function to calculate RMSE and store predictions
def calculate_rmse_and_predictions(X_train, y_train, model):
    rmse_scores = []
    actual_values = np.full_like(y_train, np.nan)
    predicted_values = np.full_like(y_train, np.nan)

    for train_index, test_index in tscv.split(X_train):
        X_train_fold, X_test_fold = X_train.iloc[train_index],
X_train.iloc[test_index]
        y_train_fold, y_test_fold = y_train.iloc[train_index],
y_train.iloc[test_index]

        # Train the model
        model.fit(X_train_fold, y_train_fold)

        # Make predictions
        y_pred = model.predict(X_test_fold)

        # Store actual and predicted values
        actual_values[test_index] = y_test_fold
        predicted_values[test_index] = y_pred

        # Calculate RMSE
        rmse = mean_squared_error(y_test_fold, y_pred,
squared=False)
        rmse_scores.append(rmse)

    # Calculate average RMSE
    average_rmse = sum(rmse_scores) / len(rmse_scores)
```

```
    return average_rmse, actual_values, predicted_values

# Initialize models
models = {
    'SVR': SVR(),
    'Neural Network': MLPRegressor(max_iter=2000)
}

# Perform time series cross-validation
tscv = TimeSeriesSplit(n_splits=10)

# Dictionary to store RMSE results and predictions
rmse_results = {}
predictions = {}

# Assuming df is already defined with the necessary data
for target_variable in target_variables:
    # Extract target variable and selected features from df
    y_train = df[target_variable]

    rmse_results[target_variable] = {}
    predictions[target_variable] = {}

    for model_name, model in models.items():
        # Calculate RMSE with base features
        X_train_base = df[selected_variables_without_trend_price]
        average_rmse_base, actual_base, predicted_base =
calculate_rmse_and_predictions(X_train_base, y_train, model)

        # Calculate RMSE with trend variables
        X_train_with_trend = df[selected_variables_with_trend_price]
        average_rmse_with_trend, actual_with_trend,
predicted_with_trend =
calculate_rmse_and_predictions(X_train_with_trend, y_train, model)

        # Calculate RMSE with all trends (base + porto trends +
economic trends)
        X_train_with_all_trends =
df[selected_variables_with_all_trends_price]
        average_rmse_with_all_trends, actual_with_all_trends,
predicted_with_all_trends =
calculate_rmse_and_predictions(X_train_with_all_trends, y_train,
model)

        # Store RMSE results
        rmse_results[target_variable][model_name] = {
            'base': average_rmse_base,
            'with_trend': average_rmse_with_trend,
            'with_all_trends': average_rmse_with_all_trends
        }

        # Store predictions
        predictions[target_variable][model_name] = {
            'base': (actual_base, predicted_base),
            'with_trend': (actual_with_trend, predicted_with_trend),
```

```
            'with_all_trends': (actual_with_all_trends,
predicted_with_all_trends)
        }

# Print RMSE results
for target_variable, model_rmse in rmse_results.items():
    print(f"Target Variable: {target_variable}")
    for model_name, rmse in model_rmse.items():
        print(f"Model: {model_name}")
        print(f"  Average RMSE with base features: {rmse['base']}")
        print(f"  Average RMSE with base features + Porto trend:
{rmse['with_trend']}")
        print(f"  Average RMSE with base features + Porto trend +
Economic trends: {rmse['with_all_trends']}\n")

# Define a dictionary to map original target variable names to
desired titles
title_mapping = {
    'PORTO_Closing_Price': 'F. C. Porto Price',
}

# Plot actual vs predicted values
dates = df.index

for target_variable, model_preds in predictions.items():
    for model_name, pred in model_preds.items():
        actual_base, predicted_base = pred['base']
        actual_with_trend, predicted_with_trend = pred['with_trend']
        actual_with_all_trends, predicted_with_all_trends =
pred['with_all_trends']

        plt.figure(figsize=(18, 18))

        plt.subplot(3, 1, 1)
        plt.plot(dates, actual_base, label='Actual')
        plt.plot(dates, predicted_base, label='Predicted')
        plt.title(f'{title_mapping[target_variable]} - {model_name}
(Base Features)')
        plt.xlabel('Date')
        plt.ylabel(title_mapping[target_variable])
        plt.legend()

        plt.subplot(3, 1, 2)
        plt.plot(dates, actual_with_trend, label='Actual')
        plt.plot(dates, predicted_with_trend, label='Predicted')
        plt.title(f'{title_mapping[target_variable]} - {model_name}
(Base Features & Porto Trend)')
        plt.xlabel('Date')
        plt.ylabel(title_mapping[target_variable])
        plt.legend()

        plt.subplot(3, 1, 3)
        plt.plot(dates, actual_with_all_trends, label='Actual')
        plt.plot(dates, predicted_with_all_trends,
label='Predicted')
```

```
        plt.title(f'{title_mapping[target_variable]} - {model_name}
(Base Features & Porto Trend & Economic Trends)')
        plt.xlabel('Date')
        plt.ylabel(title_mapping[target_variable])
        plt.legend()

        plt.tight_layout()
        plt.show()

"""## **8.6 After Hyper Parameter Tuning Porto**"""

# Define target variable
target_variables = ['PORTO_Closing_Price']

# Define exogenous variables for the target variable
exogenous_variables_price = ['Oil_Price', 'Gold_Price',
'EURUSD_Exchange_Rate',
                             'PORTO_RSI', 'PORTO_SMA_10',
'PORTO_SMA_20', 'PORTO_Volatility_10', 'PORTO_CCI',
                             'PORTO_Close_Lag1',
'PORTO_Volume_Lag1']

# Trend variables
trend_variables_porto = ['PORTO_trend']

# Define economic trend variables
economic_trend_variables = ['preco', 'mercado_acoes', 'salario']

# Combine exogenous variables and trend variables
selected_variables_with_trend_price = exogenous_variables_price +
trend_variables_porto + economic_trend_variables
selected_variables_with_porto_trend = exogenous_variables_price +
trend_variables_porto
selected_variables_without_trend_price = exogenous_variables_price

# Function to calculate RMSE and store predictions
def calculate_rmse_and_predictions(X_train, y_train, model):
    rmse_scores = []
    actual_values = np.full(len(y_train), np.nan)
    predicted_values = np.full(len(y_train), np.nan)

    for train_index, test_index in tscv.split(X_train):
        X_train_fold, X_test_fold = X_train.iloc[train_index],
X_train.iloc[test_index]
        y_train_fold, y_test_fold = y_train.iloc[train_index],
y_train.iloc[test_index]

        # Train the model
        model.fit(X_train_fold, y_train_fold)

        # Make predictions
        y_pred = model.predict(X_test_fold)

        # Store actual and predicted values
        actual_values[test_index] = y_test_fold
        predicted_values[test_index] = y_pred
```

```
        # Calculate RMSE
        rmse = mean_squared_error(y_test_fold, y_pred,
squared=False)
        rmse_scores.append(rmse)

    # Calculate average RMSE
    average_rmse = sum(rmse_scores) / len(rmse_scores)
    return average_rmse, actual_values, predicted_values

# Function to generate random layer configurations with the number
of layers between 3 and 5
def generate_layer_configs(num_configs, min_layers=3, max_layers=5,
min_neurons=10, max_neurons=120):
    configs = []
    for _ in range(num_configs):
        num_layers = np.random.randint(min_layers, max_layers + 1)
        layers = tuple(np.random.randint(min_neurons, max_neurons +
1) for _ in range(num_layers))
        configs.append(layers)
    return configs

# Generate random layer configurations
layer_configs = generate_layer_configs(75)

# Define hyperparameter distributions
param_dist_nn = {
    'hidden_layer_sizes': layer_configs,
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': uniform(0.001, 0.1),
    'learning_rate': ['constant', 'adaptive']
}

# Initialize models
models = {
    'Neural Network': (MLPRegressor(max_iter=2000,
early_stopping=True, validation_fraction=0.1, n_iter_no_change=10),
param_dist_nn)
}

# Perform time series cross-validation
tscv = TimeSeriesSplit(n_splits=10)

# Dictionary to store RMSE results and predictions
rmse_results = {}
predictions = {}
best_params = {}

for target_variable in target_variables:
    # Extract target variable and selected features from df
    y_train = df[target_variable]

    rmse_results[target_variable] = {}
    predictions[target_variable] = {}
    best_params[target_variable] = {}
```

```
    for model_name, (model, param_dist) in models.items():
        # Calculate RMSE with trend variables for price
        X_train_with_all_trend =
df[selected_variables_with_trend_price]
        X_train_with_trend = df[selected_variables_with_porto_trend]
        X_train_without_trend =
df[selected_variables_without_trend_price]

        # Hyperparameter tuning with RandomizedSearchCV
        random_search_with_all_trend =
RandomizedSearchCV(estimator=model, param_distributions=param_dist,
n_iter=75, cv=tscv, scoring='neg_mean_squared_error', n_jobs=-1,
verbose=2, random_state=42)
        random_search_with_trend =
RandomizedSearchCV(estimator=model, param_distributions=param_dist,
n_iter=75, cv=tscv, scoring='neg_mean_squared_error', n_jobs=-1,
verbose=2, random_state=42)
        random_search_without_trend =
RandomizedSearchCV(estimator=model, param_distributions=param_dist,
n_iter=75, cv=tscv, scoring='neg_mean_squared_error', n_jobs=-1,
verbose=2, random_state=42)

        # Fit and predict with all trend variables
        random_search_with_all_trend.fit(X_train_with_all_trend,
y_train)
        best_model_with_all_trend =
random_search_with_all_trend.best_estimator_
        average_rmse_with_all_trend, actual_with_all_trend,
predicted_with_all_trend =
calculate_rmse_and_predictions(X_train_with_all_trend, y_train,
best_model_with_all_trend)

        # Fit and predict with trend variables
        random_search_with_trend.fit(X_train_with_trend, y_train)
        best_model_with_trend =
random_search_with_trend.best_estimator_
        average_rmse_with_trend, actual_with_trend,
predicted_with_trend =
calculate_rmse_and_predictions(X_train_with_trend, y_train,
best_model_with_trend)

        # Fit and predict without trend variables
        random_search_without_trend.fit(X_train_without_trend,
y_train)
        best_model_without_trend =
random_search_without_trend.best_estimator_
        average_rmse_without_trend, actual_without_trend,
predicted_without_trend =
calculate_rmse_and_predictions(X_train_without_trend, y_train,
best_model_without_trend)

        # Fit and predict without trend variables
        random_search_without_trend.fit(X_train_without_trend,
y_train)
```

```
        best_model_without_trend =
random_search_without_trend.best_estimator_
        average_rmse_without_trend, actual_without_trend,
predicted_without_trend =
calculate_rmse_and_predictions(X_train_without_trend, y_train,
best_model_without_trend)

        # Store RMSE results
        rmse_results[target_variable][model_name] = {
            'with_all_trend': average_rmse_with_all_trend,
            'with_trend': average_rmse_with_trend,
            'without_trend': average_rmse_without_trend
        }

        # Store predictions
        predictions[target_variable][model_name] = {
            'with_all_trend': (actual_with_all_trend,
predicted_with_all_trend),
            'with_trend': (actual_with_trend, predicted_with_trend),
            'without_trend': (actual_without_trend,
predicted_without_trend)
        }

        # Store best parameters
        best_params[target_variable][model_name] = {
            'with_all_trend':
random_search_with_all_trend.best_params_,
            'with_trend': random_search_with_trend.best_params_,
            'without_trend':
random_search_without_trend.best_params_
        }

# Print RMSE results and best parameters
for target_variable, model_rmse in rmse_results.items():
    print(f"Target Variable: {target_variable}")
    for model_name, rmse in model_rmse.items():
        print(f"Model: {model_name}")
        print(f"  Average RMSE with all trend variables:
{rmse['with_all_trend']}")
        print(f"  Best parameters with all trend:
{best_params[target_variable][model_name]['with_all_trend']}")
        print(f"  Average RMSE with trend variables:
{rmse['with_trend']}")
        print(f"  Best parameters with trend:
{best_params[target_variable][model_name]['with_trend']}")
        print(f"  Average RMSE without trend variables:
{rmse['without_trend']}")
        print(f"  Best parameters without trend:
{best_params[target_variable][model_name]['without_trend']}\n")

# Define a dictionary to map original target variable names to
desired titles
title_mapping = {
    'PORTO_Closing_Price': 'F. C. Porto Price',
}
```

```
# Plot actual vs predicted values
dates = df.index

for target_variable, model_preds in predictions.items():
    for model_name, pred in model_preds.items():
        actual_base, predicted_without_trend = pred['without_trend']
        actual_with_trend, predicted_with_trend = pred['with_trend']
        actual_with_all_trends, predicted_with_all_trend =
pred['with_all_trend']

        plt.figure(figsize=(18, 18))

        plt.subplot(3, 1, 1)
        plt.plot(dates, actual_base, label='Actual')
        plt.plot(dates, predicted_without_trend, label='Predicted')
        plt.title(f'{title_mapping[target_variable]} - {model_name}
(Base Features)')
        plt.xlabel('Date')
        plt.ylabel(title_mapping[target_variable])
        plt.legend()

        plt.subplot(3, 1, 2)
        plt.plot(dates, actual_with_trend, label='Actual')
        plt.plot(dates, predicted_with_trend, label='Predicted')
        plt.title(f'{title_mapping[target_variable]} - {model_name}
(Base Features & Porto Trend)')
        plt.xlabel('Date')
        plt.ylabel(title_mapping[target_variable])
        plt.legend()

        plt.subplot(3, 1, 3)
        plt.plot(dates, actual_with_all_trends, label='Actual')
        plt.plot(dates, predicted_with_all_trend, label='Predicted')
        plt.title(f'{title_mapping[target_variable]} - {model_name}
(Base Features & Porto Trend & Economic Trends)')
        plt.xlabel('Date')
        plt.ylabel(title_mapping[target_variable])
        plt.legend()

        plt.tight_layout()
        plt.show()
```

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa