ARTIFICIAL INTELLIGENCE FOR TRADING STRATEGIES

A PREPRINT

Danijel Jevtic
School of Engineering
Zurich University of Applied Sciences
Winterthur, Switzerland
jevtidan@students.zhaw.ch

Romain Délèze
School of Engineering
Zurich University of Applied Sciences
Winterthur, Switzerland
delezrom@students.zhaw.ch

Joerg Osterrieder*
School of Engineering
Zurich University of Applied Sciences
Winterthur, Switzerland
joerg.osterrieder@zhaw.ch

June 5, 2022

*Financial support by the Swiss National Science Foundation within the project "Mathematics and Fintech - the next revolution in the digital transformation of the Finance industry" is gratefully acknowledged by the corresponding author. This research has also received funding from the European Union's Horizon 2020 research and innovation program FIN-TECH: A Financial supervision and Technology compliance training programme under the grant agreement No 825215 (Topic: ICT-35-2018, Type of action: CSA). Furthermore, this article is based upon work from the COST Action 19130 Fintech and Artificial Intelligence in Finance, supported by COST (European Cooperation in Science and Technology), www.cost.eu (Action Chair: Joerg Osterrieder).
The authors are grateful to Stephan Sturm, Moritz Pfenninger, Samuel Rikli, Bigler Daniel, Antonio Rosolia, management committee members of the COST (Cooperation in Science and Technology) Action Fintech and Artificial Intelligence in Finance as well as speakers and participants of the 5th and 6th European COST Conference on Artificial Intelligence in Finance and Industry, which took place at Zurich University of Applied Sciences, Switzerland, in September 2020 and 2021.

DECLARATION OF ORIGINALITY

Bachelor's Thesis at the School of Engineering

By submitting this Bachelor's thesis, the undersigned student confirms that this thesis is his/her own work and was written without the help of a third party. (Group works: the performance of the other group members are not considered as third party).

The student declares that all sources in the text (including Internet pages) and appendices have been correctly disclosed. This means that there has been no plagiarism, i.e. no sections of the Bachelor thesis have been partially or wholly taken from other texts and represented as the student's own work or included without being correctly referenced.

Any misconduct will be dealt with according to paragraphs 39 and 40 of the General Academic Regulations for Bachelor's and Master's Degree courses at the Zurich University of Applied Sciences (Rahmenprüfungsordnung ZHAW (RPO)) and subject to the provisions for disciplinary action stipulated in the University regulations.

City, Date:	**Name Student:**
Affoltern am Albis, 05.06.2022	Danijel Jevtic
Thalwil, 05.06.2022	Romain Délèze

ABSTRACT

In recent years, much research has been done in predicting stock price trends using machine learning algorithms. However, machine learning methods are often criticized by financial practitioners. They argue that neural networks are a "black box." In this bachelor thesis, we show how four different machine learning methods (Long Short-Term Memory, Random Forest, Support Vector Machine Regression, and k-Nearest Neighbor) perform compared to already successfully applied trading strategies such as Cross Signal Trading and a conventional statistical time series model ARMA-GARCH. The aim is to show that machine learning methods perform better than conventional methods in the crude oil market when used correctly. A more detailed performance analysis was made, showing the performance of the different models in different market phases so that the robustness of individual models in high and low volatility phases could be examined more closely. The focus was on the robustness and performance of the machine learning methods. It was essential to keep complexity in proportion to performance. All analysis work was realized with the software Python. It was shown that the machine learning methods such as Random Forest, on average, made better decisions in highly volatile market periods than ordinary statistical models such as ARMA-GARCH. The Random Forest was characterized by a very high true-positive rate (recall), which set it apart from the other models. Likewise, the final performance result provided conclusions about the robustness of the individual models. It could be shown in which market phases the models made good or bad trading decisions. In the case of the k-Nearest Neighbor model, which failed to detect the Corona crisis, it was shown that not every machine learning model responded well to different market phases. Comparing this with the Random Forest model, it can be seen that the most significant monthly gain was achieved there across all models. The conventional statistical models - ARMA-GARCH - performed surprisingly well and were on par with the machine learning models. Nevertheless, the extreme values were analyzed in more detail. It was found that the Random Forest model made the best decisions at the 95% interval level. This shows that the Random Forest model always made better decisions than all other models during periods when a lot is going on in markets, which is reflected in the total return. Surprisingly, the ARMA-GARCH model made the best decisions at the 99% level but performed worse on average when the market was "calmer." It was shown that better results could be achieved with proper machine learning methods than with conventional statistical methods. For further investigation, these models would also have to be analyzed in other markets.

***K*eywords** Machine Learning, Trading strategies, Financial time-series, Time-series analysis, Financial data

Contents

1 Introduction

Our research combines three main building blocks: Financial Time Series, Machine Learning, and Trading Strategies. We briefly reviewed existing research in each of these three areas. We used a classical approach with an ARMA-GARCH model and Cross Signal Trading as a benchmark in this work. We then compared the performance of different machine learning models such as Random Forest (RF), k-nearest neighbors (kNN), Support Vector Machines (SVM), and a long short-term memory (LSTM) Network

1.1 Financial Markets and Data

Less than 70 years after the term was coined, artificial intelligence has become an integral part of the most sophisticated and fast-moving industries, with significant impacts for the financial industry. The impact of AI and ML tools on financial markets cannot be underestimated. Strong computational power allows large amounts of data to be processed in a short period of time, while cognitive computing helps manage structured and unstructured data. Algorithms analyze financial data and identify early signs of potential future problems. Financial markets, like the economy, are highly complex systems, and it is often impossible to explain macroscopic phenomena by simple summaries of microscopic processes or events. The outcomes of actions in the system are difficult to predict, and sometimes it is impossible to find the cause of significant anomalies or even the factors that influence the event. [1] AI in finance is a powerful ally in analyzing real-time activity in a given market or environment, providing accurate and detailed forecasts based on multiple variables that are critical to business planning. For example, we mentioned Equbot's AI-driven equity fund ETF (AIEQ), which is based on IBM's Watson AI program. AIEQ is the first ETF to use artificial intelligence for stock selection. AI autonomously analyzes data, processes it, and then makes investment decisions. AI can beat the market without the deployment delays of traditional data science methods. [2]

Financial data consists of pieces or sets of information related to the financial health of a business. The pieces of data are used by internal management to analyze business performance and determine whether tactics and strategies must be altered. People and organizations outside a business will also use financial data reported by the business to judge its credentials, decide whether to invest in the business and determine whether it complies with government regulations. [3]

1.2 Financial time-series modelling

The financial industry has always been interested in successfully predicting financial time series. Numerous studies based on conventional time series analysis and ML models have been published. Due to the successful research in AI, more and more conventional models are combined with ML models, which have been met with success.

There are several methods of modeling financial time series. In particular, the ARCH/GARCH models, which are based on classical statistics, model the change in variance over time in a time series by describing the variance of the current error term as a function of past errors. Often, the variance is related to the squares of the previous innovations as AR (Auto-Regressive) models rely heavily on past information to make a prediction. They are often used in time series models where time-varying volatility and volatility clusters are present. We will primarily rely on the ARCH/GARCH models as the statistics are trackable and, therefore, can be considered a good benchmark. [4]

1.3 Neural networks and Machine learning

Neural networks are a subset of machine learning techniques and are at the heart of deep learning algorithms. Deep neural networks consist of node layers that contain three different layers that are input layers, hidden layers, and output layers. In the simplest case, each node is connected to another and has an associated weight and threshold. If the output of a single perceptron is above the specified threshold value, that perceptron is activated and sends data to the next layer of the network. Otherwise, no data is forwarded to the next layer of the network.

Neural networks rely on training data to learn and improve their accuracy over time. Once these learning algorithms achieve high accuracy, they are potent tools in computer science and artificial intelligence, allowing us to classify cluster data and serve as universal approximation tools for unknown functions. Machine learning models such as RF are typically treated as black boxes. Due to the fact that a forest consists of many deep trees, where each tree is trained on bagged data using a random selection of features, gaining a complete understanding by examining each tree would be close to impossible. [5]

1.4 Research Contribution

This paper provides an overview of the current machine learning models used in finance. The aim is to explain the information gained from trading strategies using different ML models. The data of Brent crude Oil to different ML models will be analyzed and applied to trading strategies. Furthermore, the functionality of ML models will be taught, and the current state of research and practice will be presented in applications.

1.5 Structure of this paper

The continuation of the thesis is structured as follows: Section 2 will present the background literature on artificial intelligence for trading strategies, current research, and similar approaches while section 3 introduces the experimental work (a weak form of EMH). Lastly, section 5 will conclude the thesis and provide an outlook at further research and applications.

2 Literature Overview

In the literature overview, the used literature is briefly summarized. In the beginning, the Efficient Markets and Rational Expectations are discussed. Afterward, machine learning in Finance is introduced. In the chapter Background, the used models are discussed to overview the upcoming work.

2.1 Efficient markets and Rational Expectations

The theory of efficient markets was established in 1970 by Eugene F. Fama (Efficient Capital Markets: A Review of Theory and Empirical Work). Fama described efficiency as "Markets are efficient only if prices always "fully reflect" all available information." Fama divided the efficient market theory into three different forms: the weak form, which includes historical prices, and the semi-strong form, which identifies with public information such as profits/losses, stock splits, etc. The strong form includes insider information. Further in itself, the question was whether the market prices would last exactly. However, this stands in conflict with whether a constant outperformance would be possible; this is not possible alone by using already known information. Other conditions were defined so that the capital market efficiency was fulfilled. The first condition: no transaction costs, the second: all information is available to all market participants at no cost, and the third: all agree on the impact of current information on the current price and distribution of future prices of each security. [6]

Charlie Munger quotes: "I think it is broadly true that the market is efficient, which makes it very hard to beat. But I do not think it is completely efficient." [7]

Therefore, a perfectly efficient market can be unpredictable because the more efficient it is, the more random and unpredictable the returns. Due to this, most economists agree that the EMH is not absolute, which Eugen Fama also agreed with. The participants, like the investors or simple traders, form the market, so everyone constantly influences the efficiency of the market. The public tries to gain a profit from information advantages. This is why the thesis is that prices contain all information reflecting the intrinsic value of the underlying asset. Additionally, the EMH does not work perfectly in the real world because people are irrational. Therefore, papers were also written regarding rational expectations. In 1961, John F. Muth said that "Our hypothesis is based on that dynamic economic models do not assume sufficient rationality." This remark should be understood that not perfect efficiency of the markets as a conclusion. [8]

2.2 Machine Learning in Finance

The financial industry is most simply described as an information-processing industry, and each sector processes information differently. For example, investment funds make their investment decisions by processing information, while insurance companies use the information to determine the price of future insurance policies. In the 21st century, large amounts of data are generated daily. Therefore, it is obvious to use machine learning or neural networks for information processing. Now the question arises, what is machine learning? To answer this, we cite Arthur Samuel's quote published in 1959: "Machine learning is the subfield of computer science that gives computers the ability to learn without being explicitly programmed." In other words, machine learning does not require conventional rules, which humans have programmed in all software. It gives the computer the ability to learn on its own. We let the computer develop its own rules through pattern recognition. There are four different types of learning: supervised, unsupervised, reinforcement learning, and deep learning.

In supervised learning, functions are determined based on training data whose output is known (classification and regression). In unsupervised learning, patterns are recognized from unknown data sets, and rules are derived (clustering as one of the primary forms). In reinforcement learning, the algorithm is given parameters that it can control and vary, and it optimizes towards this goal independently through simulations. Finally, deep learning - is a system of artificial software neurons based on the human brain, which communicate with each other and learn independently from data. [9]

2.3 Background

In this chapter, the used models are brought closer. The basic ideas of the process or algorithm are explained and further discussed in the chapter titled Experiment.

ARMA-GARCH Models

One of the most challenging and complex topics in finance is modeling financial time series. The problems can be attributed to the complexity and variety of stocks, currency exchange rates, interest rates, and the importance of observation frequencies such as seconds, minutes, hours, or days. Often the availability of large data sets is also a problem. However, the biggest problem is attributed to the existence of statistical regularities (stylized facts) that are common to many financial series and are difficult to reproduce artificially with stochastic models. Most of these stylized facts were presented in a scholarly article by Mandelbrot (1963).

In general, the objective of time series analysis is to find a model for the underlying stochastic process. This model is then used to analyze the process's structure or make optimal predictions. The class of ARMA models is most commonly used to predict steady-state processes. The problem is that the returns of financial time series depends on time and prove volatility clusters. For this problem, the ARCH model was developed by Engel in 1982, which can model autoregressive, conditionally heteroskedastic patterns in the data. A few years later, the generalized ARCH model GARCH was developed by Bollerslev (1986). In these models, the key concept is the conditional variance, i.e., the variance that depends on the past. The conditional variance is expressed as a linear function of the squared past values of the series. Now one can capture the vital stylized facts that characterize the financial series. [10]

Random Forest

RF is an algorithm used for classification and regression tasks. To have a more analyzed view of what RF is, we refer to the book written by James et al. (An Introduction to Statistical Learning), published in 2013. It combines the results of many different decision trees to make the best possible decisions. The learning algorithm belongs to the supervised learning methods. This uses the results of many different decision trees to make the best possible decisions or predictions. The decision trees were created randomly in an uncorrelated manner. Each tree makes individual decisions for itself. From the set of individual decisions, the algorithm delivers a final decision.

In order to fully understand the functional principle of RF, the terms decision tree and bagging must first be explained in more detail. Decision trees form the basis for RF and a single tree consists of several branches. The branches are then created by assigning data to a class based on their properties using rules. Beginning from the first decision, more and more branches are created until a certain result level is reached. Bagging is a specific method of combining the individual predictions of different classification models. The individual results of the decision trees are included in the overall result with a specified weighting. [11]

Support Vector Machines

SVM are generally used for classification and regression problems. Furthermore, SVM can easily handle multiple continuous and categorical variables. In order to separate the data set into different classes, SVM constructs a hyperplane in multidimensional space. To guarantee the minimization of the error, SVM iteratively generates an optimal hyperplane. The core idea is to find a maximum marginal hyperplane that best classifies the data-set. The support vectors, hyperplane, and the margin have to be explained to understand the procedure better.

Support vectors are the data points that are closest to the hyperplane. These points have the most significant impact on the classification of the data. Through these points, the dividing line is better defined by calculating the margins. The hyperplane can be seen as a decision plane. This is between the set of objects lies, with which the different class memberships are separated. The margin is the gap between the two lines at the closest class points. The margin is the perpendicular distance between the line and the support vectors. The bigger the gap between the classes, the better. A smaller gap is considered flawed. [12]

K-nearest-neighbors

The kNN algorithm belongs to the supervised machine learning methods and is one of the topmost machine learning algorithms. kNN is used in various applications such as finance, healthcare, political science, and credit ratings. It can be used to solve classification and regression problems. Furthermore, the kNN algorithm is based on the feature similarity approach.

The focus of this section is to explain how the algorithm works and how it is defined. In supervised learning, the algorithm is given a data-set labeled with appropriate output values with which it can train and define predictive models. Subsequently, the trained model will be applied to a test set to predict the corresponding output values. The intuition of the kNN algorithm is simple to understand. kNN is a non-parametric learning algorithm. Non-parametric means that there is no assumption for underlying data distribution. The first step is to select the number of k neighbors. The number of nearest neighbors in the kNN algorithm is denoted by K, which is also the deciding factor. If the number of classes is 2, K is generally odd. We get the simplest case of the algorithm when $K = 1$. Then the algorithm is called nearest neighbor. Suppose P1 is the point for which the label must be predicted. In the first step, we need to find the k points nearest to P1 and then classify the points by voting their k neighbors. Each object votes for its class and the class with the most votes is used as the prediction. The closest Points can be found if we use distance measures like Euclidean or Manhattan distance. [13] [14]

LSTM

In 1997, LSTM networks were introduced by Hochreiter and Schmidhuber. LSTM networks belong to the family of recurrent neural networks (RNN). These are the most widely used nets. LSTM networks are capable of learning short-term dependency structures. Their standard behavior is to remember information over long periods of time. Therefore such networks are often used in applications such as classification of time series, generation of sentences, speech recognition, and handwriting recognition. In recent years, the idea of using such networks in forecasting financial time series has become more common [15]. With that being said, they have been increasingly used as a forecasting model. Over time, more LSTM networks have been developed, such as in the work of Z. Li (Financial time series forecasting model based on CEEMDAN and LSTM), which was published in 2018 [16]. They improved the forecasting accuracy of various stock indices such as S&P500 or the DAX by combining the CEEMDAN signal decomposition algorithm with the LSTM model. This should serve as an example for the continuous development of the LSTM networks. A simpler LSTM network was developed through this intuition, which can predict the Brent crude oil price. The prediction is then converted into signals, deciding when to buy and sell.

3 Data

The following chapter aims to analyze the data obtained by the Brent crude oil future data set by the following topics: data description, Brent crude oil, crude oil price determination, financial indicators, log-returns, trading signals, and descriptive statistics. By looking at these various topics, one can fully understand and grasp the full understanding that goes behind the data set.

3.1 Data description

For this experiment, one used the daily data of the Brend-Crude-Oil future from 30.07.2007 to 26.04.22. The data used for the experiment was obtained by Yahoo Finance, which can be accessed directly with the yfinance package on Python. The format of the data is called pandas.core.frameData Frame, which is a tabular form of the data. When accessing the data, the following price values were downloaded: Open, High, Low, Close, Adj Close, and the volume. It was found that the price values of a financial instrument show very similar patterns. One first had to process the data and extend it with financial indicators, such as RSI, MACD, Price Rate Of Change, and others that will be discussed later on. Many machine learning methods are classification methods; for such methods, we must create a signal that can be interpreted as a trading signal.

3.1.1 Brent Crude Oil

Brent oil is a type of oil that consists of four various fields known as Brent, Forties, Oseberg, and Ekofisk. Brent oil has a limited production of 170'000 barrels per day, which makes up only 0.2 percent of the current worldwide consumption. These barrels will be transported via a subsea pipeline to the Sullom Voe oil terminal on Mainland Shetland, together with the oil from the Ninian field to the final destination of a tanker. Prior to the completion of the pipeline and oil terminal, oil will be loaded onto tankers in the North Sea from loading platforms such as the Brent Spar. Norway's Statfjord field is found on the East and is connected by two fields by a transfer pipeline. The Brent and Ninian oil fields have now exceeded their production peaks. The traded product is Brent Blend, which is originally a blend derived mainly from the Brent and Ninian fields. The product comes from the North Sea between the Shetland Islands and Norway. The Brent oil field is being developed by Shell UK Ltd.

Brent is arguably the most important type of crude oil for Europe due to its low-boiling and light oil texture, which allows it to be more valuable than other crude oils. In addition, it also has a lower sulfur content than other crude oils. It was discovered in 1971 and has been produced since 1976, in the Brent South subfield. It is traded on the ICE Futures commodity exchange in London. [17]

3.1.1.1 Crude oil price determination

How is the price of crude oil determined, and how does crude oil trade? The relation of supply and demand determines crude oil prices. When decisions are made because of speculation about possible future developments, psychology plays a crucial role in the stock market. Crude oil prices depend heavily on how traders assess the short- and medium-term direction of the sales market. Fear of political unrest can manifest itself in a price premium that does not in the least reflect real supply conditions. Other actual or expected changes in supply or demand are also reflected in the prices offered on the exchange. The onset of the "driving season" in the spring in the United States of America also makes itself feel like an exceptionally frosty winter in Europe. Incredibly high or low stock levels in important consumer countries lead to decreased or increased product prices. Moreover, of course, the dollar exchange rate immediately impacts the oil price. Oil barrels or tank farms will not be found on these trading floors. In principle, it is possible to deliver or collect the traded commodity, but this is rarely done in practice. Crude oil is traded on the stock exchange mainly in futures contracts. A futures contract contains a formal, legally binding commitment to buy or sell a predefined quantity of a commodity at a specific time in the future (hence the term "futures"). These very contracts are standardized to ensure that all market participants assume the same qualities, quantities, and delivery terms. As a result, the traded oil futures include only a limited range of products: Crude oil, gasoline, diesel, and heating oil. The exchanges primarily hedge against future price fluctuations for traders when dealing with tangible, "physical" commodities. This strategy is called hedging. It is exciting for oil traders, producers, refiners, and large consumers such as airlines. All these companies have to buy or sell large quantities of petroleum products several months in advance. They are therefore highly exposed to the risk of price fluctuations. Hedgers are not looking for short-term profits on the stock exchange

but instead try to balance their positions in the market for the physical commodity and better distribute their risk situations. The majority of the stock market participants pursue a different goal: they try to make short-term gains because they estimate the future market development. These speculators help the market to greater liquidity and take over the dangerous situations that the hedgers try to pass on. However, they can also contribute to the amplification and acceleration of price fluctuations, especially if they are not familiar with the business and technical environment of the oil industry. Thanks to centralized, open business transactions with standardized contracts on the exchange, the prices paid there are exceptionally transparent. [18]

3.1.2 Financial indicators

There are various financial indicators that can help investors to make buy or sell decisions. The most important financial indicators are KPIs which belong to fundamental analysis and technical analysis.

KPIs are used to register how successful specific actions in companies are. All processes in companies can be monitored based on those key performance indicators. With the help of key performance indicators, management and controlling can gain knowledge and analyze processes in companies. With the help of this consistent monitoring, stakeholders can faithfully adapt and improve processes and measures. There is a significant number of selected KPIs. Depending on the company and the area, different KPIs are relevant for measuring performance. For example, marketing focuses on different KPIs than sales, accounting, or management. Which KPIs are the correct ones depends continuously on which actions are to be checked. For example, the KPIs used to study customers' behavior play an exceptionally fundamental role in marketing. On the other hand, management wants to benefit from the project or department performance KPIs. [19]

In distinction to fundamental analysis, which deals with essential company data, chart analysis is often called technical analysis. This is because it is not the evaluation of business reports in the foreground. Nevertheless, the interpretation of "technical" aids such as chart progressions or indicators of the trends, oscillations, or patterns in the time series that are to be uncovered. Technical analysis is thus a tool, a method for predicting the likely upcoming price movements of a financial asset - such as a stock or a currency pair - based on market data. Behind the validity of technical elicitation is the demonstration that market participants' collective activities (buying and selling) specifically reflect all relevant data about a traded asset and, therefore, continually assign it a fair market value. In other words, technical traders believe that the market's current or past price action under consideration is the most reliable indicator of future prices. Thus, the technical study can be used for any financial instrument with historic price data. Therefore, chart analysts focus on patterns of price movements, trading signals, and various other analytical charting tools to evaluate the strength or weakness of an asset. All chart analysis product packages have one thing in common: the assumption that observable occurrences repeat with the probability that can be predicted within limits. It becomes arguable to use specific patterns in charts or the values of indicators as a guide for the trader. Technical analysis is not only used by chart analysts. Plenty of investors use fundamental analysis to determine whether they should buy into a market or not. Nevertheless, after this decision is made, they also use chart analysis to register excellent entry price levels. [20]

3.1.2.1 Returns and Log Returns

Rates of return and logarithmic rates of return are the net percentage gains or losses of the past cost of an investment over a specified period. They calculate the percentage rate of change over a defined period such as a day, month, or year. Returns can be used to measure the increase in value of any asset, including stocks, bonds, mutual funds, real estate, collectibles, etc. Investors can also use the returns to compare past periods or other investments. Two inputs are required to calculate the rate of return: i) the purchase amount of the investment and ii) the current value or terminal value of the investment in the period being measured. In some cases, any income generated from the asset such as interest and dividends is also included in the calculation. A rate of return is calculated by subtracting the initial value of the investment from its current value and then dividing it by the initial value. To express the return in percent, the result is multiplied by 100. A simple logarithmized rate of return is calculated by taking the quotient of the initial value of the investment by the current value of the investment and logarithmizing. To express the return in percent, multiply the result by 100. [21] [22]

$$\text{Returns in percent} = \frac{r_t - r_{t-1}}{r_{t-1}} * 100 \tag{1}$$

$$\text{Logarithmic Returns in percent} = log(\frac{r_t}{r_{t-1}}) * 100 \tag{2}$$

3.1.2.2 RSI

The Relative Strength Index or RSI is a momentum trading indicator often used in technical analysis. The RSI can take values between 0 and 100 and display them as an oscillator. It measures the impact of the price change in a given period. The default period is 14 days but can be changed to any. The RSI is used to measure overbought or oversold conditions in a stock or other asset price. The usual interpretation of the RSI is that values greater than or equal to 70 indicate an overbought condition (overvaluation of the asset) and will cause the price to fall. Values less than or equal to 30 indicate an oversold condition (undervaluation of the asset), and a price slope follows. [23] The RSI is calculated in the following three steps:

First, the sum of all positive and negative price changes is calculated:

$$\text{sum}_{\text{up}}(t) := \sum_{i=1}^{n} \max\left\{P(t-i+1) - P(t-i); 0\right\} \tag{3}$$

$$\text{sum}_{\text{down}}(t) := -\sum_{i=1}^{n} \min\left\{P(t-i+1) - P(t-i); 0\right\} \tag{4}$$

Then the mean value of the sums is taken:

$$\text{avg}_{\text{up}} := \frac{\text{sum}_{\text{up}}}{n} \tag{5}$$

$$\text{avg}_{\text{down}} := \frac{\text{sum}_{\text{down}}}{n} \tag{6}$$

The RSI then results with:

$$\text{RSI} := 100 - (\frac{100}{1 - (\frac{\text{avg}_{\text{up}}}{\text{avg}_{\text{down}}})}) \tag{7}$$

3.1.2.3 Stochastic Oscillator

The stochastic oscillator, also known as the stochastic indicator, is a popular trading helpful indicator for predicting trend reversals. The indicator works by focusing on the location of an instrument's closing price relative to the high-low range of the price over a given period. Typically, 14 prior periods are used. By comparing the closing price to previous price movements, the indicator predicts price reversal points. The stochastic indicator is a two-line indicator applied to any chart. It fluctuates between 0 and 100. The indicator shows how the current price behaves compared to the highest and lowest price levels in a predefined period. If the stochastic indicator is at a high level, the instrument's price has closed near the upper end of the period range. If the indicator is at a low level, the price has closed near the lower end of the period range. The general rule for the stochastic indicator is that prices in an upmarket close near the high. In contrast, in a down market, prices close near the low. When the closing price deviates from the high or low, it indicates a slowdown in momentum. The formula of the Stochastic Oscillator is as follows: [24]

$$\begin{aligned} C &= Current\ Close\ Price \\ H_x &= The\ Highest\ High\ value\ of\ the\ last\ x\ periods \\ L_x &= The\ Lowest\ Low\ value\ of\ the\ last\ x\ periods \end{aligned}$$

$$Percent_K = (\frac{C - L_x}{H_x - L_x}) * 100 \tag{8}$$

The idea behind the stochastic indicator is that the dynamics of the price of an instrument often change before the instrument's price movement changes direction. It also focuses on price momentum and can be used to identify overbought and oversold levels in stocks, indexes, currencies, and many other assets.

3.1.2.4 MACD

The moving average convergence divergence (MACD) indicator calculates the change or speed of price movement of the asset. It is a momentum indicator that follows a trend and shows the correlation between two moving averages of the asset. The formula for calculating the MACD is simple. MACD is a subtraction of the 26-period exponential moving average (EMA) from the 12-period EMA.

$$MACD = \mathrm{EMA}_{12} \; - \; EMA_{26} \tag{9}$$

EMA is the moving average that gives more weight to the current price points. This allows this moving average to be more responsive to recent price changes. [25]

3.1.2.5 Price Rate Of Change

The Price Rate of Change or (ROC) conveys exactly the same statements as the "Momentum". Momentum measures the speed, force or strength of a price movement by subtracting the closing price n periods ago from today's closing price. The difference between the two indicators is simply that in the ROC the difference calculated is divided by the closing price n periods ago and this quotient is then multiplied by 100. The current ROC value thus shows by how many percent today's price is above or below the price n periods ago. Some programs omit the multiplication by 100, which means that the results are not displayed at a 100 percent line, but at the zero line. This is purely program-related, the statement is the same. The ROC shows the momentum of the price movement. Accordingly, the position of the ROC line is important. (ROC in the positive area) A rising ROC indicates an (increasing) positive momentum, i.e. a continuation of the upward trend. A falling ROC indicates a decrease in momentum and thus a possible end of the upward movement. (ROC in negative range) A falling ROC indicates a (increasing) negative momentum, thus a continuation of the downward trend. A rising ROC indicates an easing of the negative momentum and thus a possible end of the downward movement. Thus, if recent price gains are less than previous ones, the ROC will fall in positive territory, and analogously, if recent price declines are less than previous ones, the ROC will rise in negative territory. Thus, the ROC can turn even though the stock price is forming new highs or lows. The ROC can be calculated as follows [26]:

The Price Rate of Change (ROC) conveys the same statements as the "Momentum." Momentum measures a price movement's speed, force, or strength by subtracting the closing price n periods ago from today's closing price. The difference between the two indicators is simply that in the ROC, the difference calculated is divided by the closing price n periods ago. This quotient is then multiplied by 100. The current ROC value thus shows by how many percent today's price is above or below the price n periods ago. Some programs omit the multiplication by 100, which means that the results are not displayed at a 100 percent line, but at the zero line. This is purely program-related; the statement is the same. The ROC shows the momentum of the price movement. Accordingly, the position of the ROC line is essential. (ROC in the positive area) A rising ROC indicates an (increasing) positive momentum, i.e., upward trend continuation. A falling ROC indicates a decrease in momentum and thus a possible end of the upward movement. (ROC in negative range) A falling ROC indicates an (increasing) negative momentum, thus continuing the downward trend. A rising ROC indicates an easing of the negative momentum and thus a possible end of the downward movement. Thus, if recent price gains are less than previous ones, the ROC will fall in positive territory, and vice versa, if recent price declines are less volatile than previous ones, the ROC will rise in negative territory. The ROC can turn even though the stock price forms new highs or lows. The ROC can be calculated as follows [26]:

$$\begin{aligned} n &= \text{Number } of \; periods \\ C_t &= \text{Current } close \; price \\ C_{t-n} &= \text{Close } price \; n \; periods \; before \end{aligned}$$

$$ROC_t = \frac{C_t - C_{t-n}}{C_{t-n}} * 100 \tag{10}$$

3.1.2.6 Trading Signal

A user can freely define the trading signal. The returns are used for coding signals (binary signals) in this work. Positive returns are coded as the number one, and negative returns are coded as zero. After encoding the signal, the signal is shifted back to one line to realize a one-day prediction. This was done in the context of classification algorithms like kNN.

$$Signal_t = \begin{cases} 1 & , \text{if } return_{t+1} > 0 \\ 0 & , \text{if } return_{t+1} <= 0 \end{cases} \tag{11}$$

3.2 Descriptive statistics

The Brent Crude Oil dataset was extended with the financial indicators, and all irrelevant values were filtered out. After that, all rows with NA values were deleted to clean up the dataset. After editing, the dataset consists of the following columns: Close Price, Log Returns, RSI, MACD, K Percent, ROC, and Signal. A closer look at the data will show various outcomes such as how the data is distributed, which are the most critical parameters, which columns contain outliers, and can already be made certain statements about how the data will behave in the future.

3.2.1 Time Series Brent Crude Oil

In periods of high volatility, the prices can change very rapidly, and the prices also tend to fall in such cases. The time series has only shown a clear upward or downward trend in specific time intervals in recent years. In figure 1, the Brent Crude Oil feature closing price with a simple moving average over 200 days, and the Log Returns can be seen.

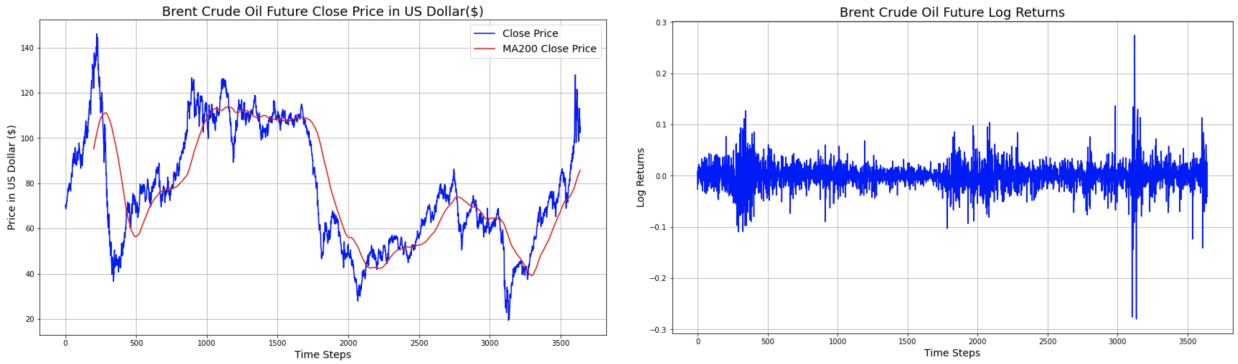


Figure 1: Close Price from Brent Crude Oil Future (left) and Log Returns from Brent Crude Oil Future (right)

As depicted in table 1, the essential position measures are shown. The average closing price is USD 77.4, as shown in figure 1. The average of the log-returns is more interesting because it is a positive number, which means that the Brent Crude Oil Future gives a positive return on average, which is a good measure for investments. In addition, the quantiles shows that the chance of making a profit is more remarkable than making a loss, which provides us with information that the buy-and-hold strategy is profitable.

Table 1: Position measure (Close Price and Log Returns)

Statistic	Close	Log Returns
Mean	77.384	0.000105
Standard deviation	25.919	0.025
Minimum	19.330	-0.280
5%	41.290	-0.037
25%	55.980	-0.010
50%	73.360	0.0004
75%	103.810	0.011
95%	117.100	0.035
Maximum	146.080	0.274

The distribution of the log-returns (figure 2) shows that the distribution is symmetric around zero. It is visible that the distribution is not normally distributed with the help of a quantile-quantile normal plot (figure 3); it is possible to verify this. The distribution's tails are immense; for this reason, a t-Students distribution is to be assumed here.

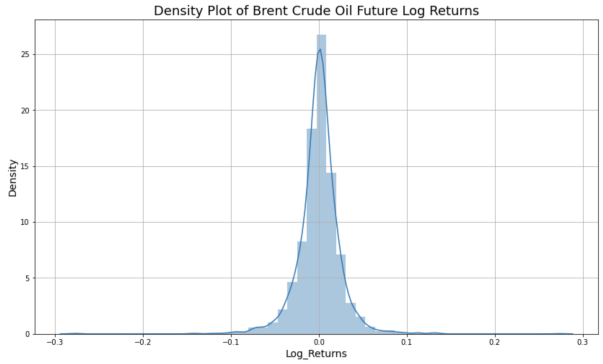


Figure 2: Density Log Returns

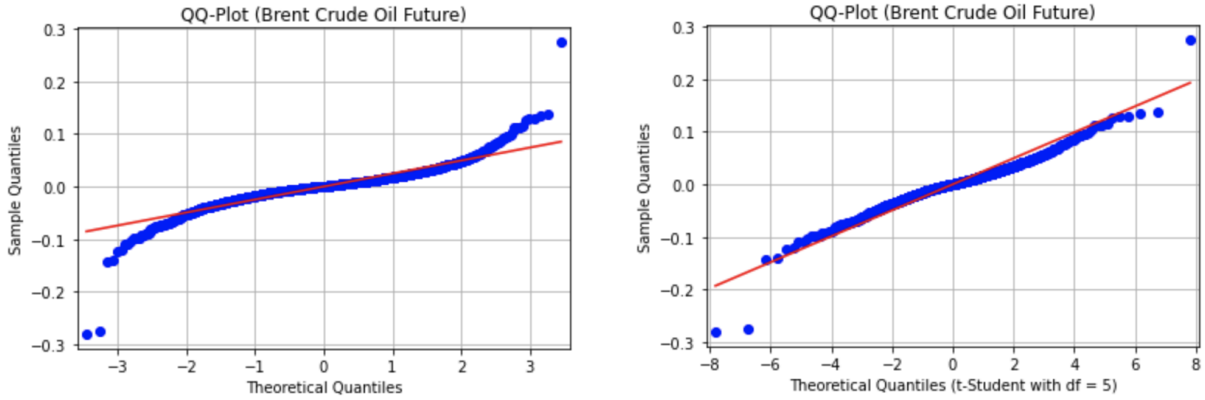


Figure 3: Normal QQ-Plots (left) and t-Student QQ-Plot with df = 5 (right)

To check this, another quantile-quantile plot was used. But this time with the distribution assumption of a t-Students distribution with 5 degrees of freedom. See figure 3. It is obvious that the points fit better around the red line. The 3 points that are not on the red line can be considered as outliers. The autocorrelation of the log returns look like this figure 4.

The distribution assumption of a t-Students distribution was used with 5 degrees of freedom (see figure 3). The points fit better around the red line. The 3 points which are not on the red line are outliers. The autocorrelation of the log-returns looks like in figure 4. The dependency structure is shallow.

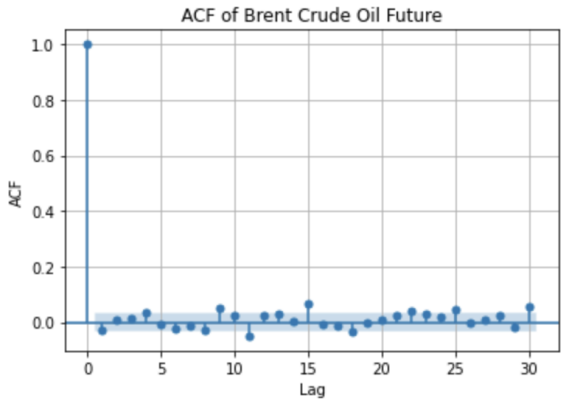


Figure 4: Autocorrelation Plot

The maximum drawdown (MDD) is a measure of the risk of an investment fund. It represents the maximum cumulative loss from the highest price within a period under consideration and is usually presented as a percentage value. Figure 5 shows the MDD of the whole available data and its distribution.

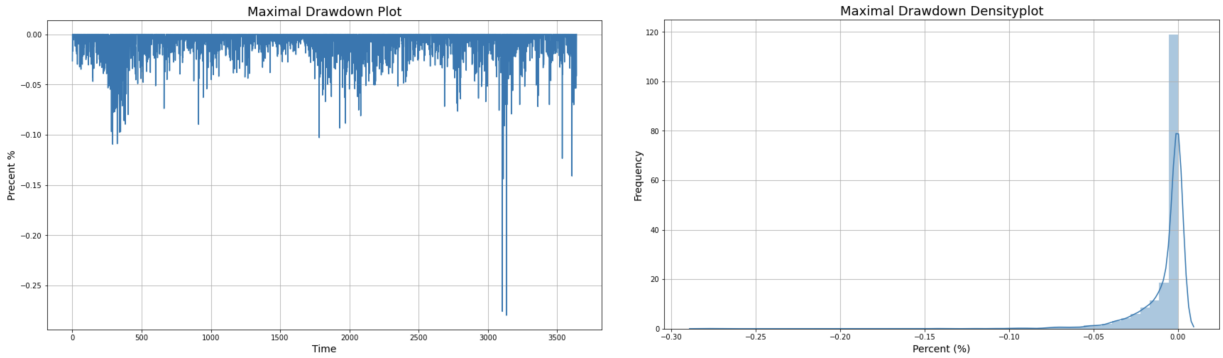


Figure 5: MDD Plot (left) and MDD Density Plot (right)

Table 2 is an overview of the most important parameters of the total MDD. The quantiles show that no losses are generated in most cases, which confirms the assumption made based on the table (Log Returns). It can be seen (except for a few outliers) that the maximum losses are around -0.1 percent and that the losses are usually minimal.

Table 2: Maximum Drawdown Position measure

Statistic	**MDD**
Observations	3641
Mean	-0.008
Standard deviation	0.016
Minimum	-0.280
5%	-0.037
25%	-0.010
50%	0
75%	0
95%	0
Maximum	0

The distributions of the features in figure 6 have a bell shape with the exception of the K percent. This feature is a bimodal distribution, and depending on the Machine Learning algorithm, it is advisable to transform the data to be as normally distributed as possible.

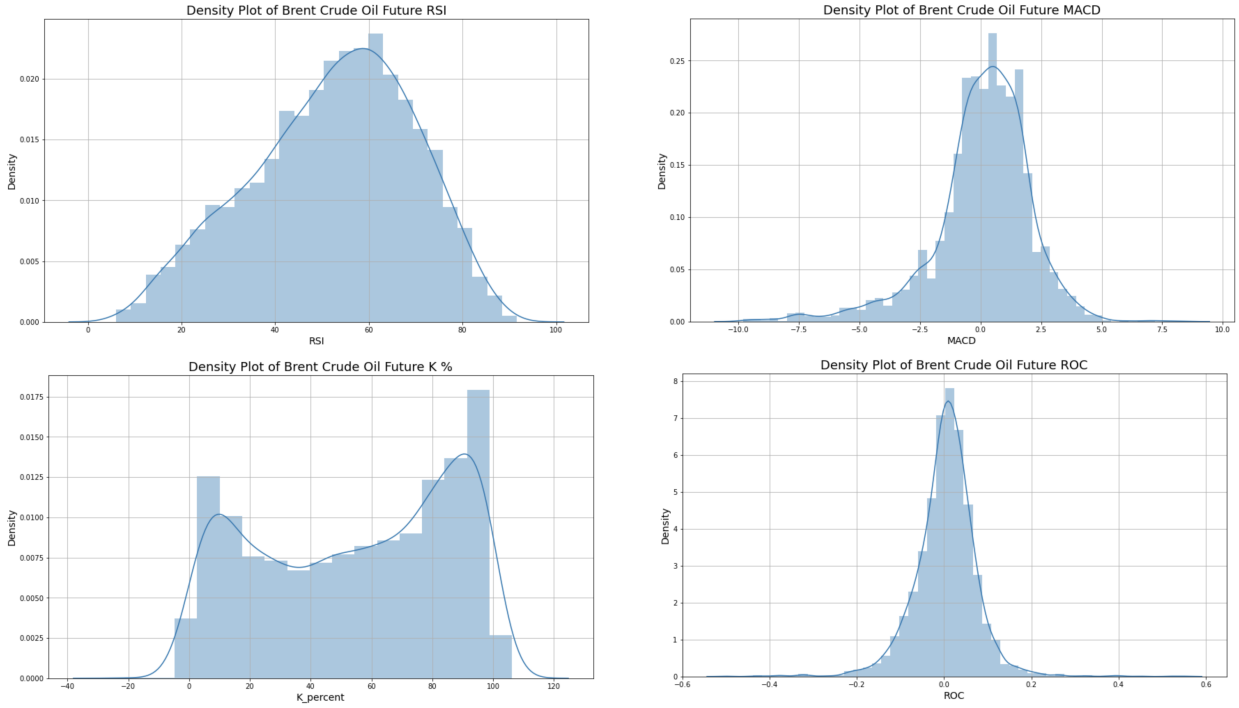


Figure 6: Feature Density Plots

4 Experiment

Within this chapter, there were six different models presented that had different applications and results. For each model, two possible strategies were simulated. The first strategy focuses on a constant long strategy and the second strategy focuses on a long-short strategy. In the long strategy, a signal vector was generated, coded with 1 and 0. Code 1 is a buy signal, and code 0 is a sell signal. In this strategy, either future asset was bought or sold. Moving onto the long-short strategy that was coded with the signal 1 or -1, this strategy allows short selling. Short-selling strategies have a higher risk appetite but can yield significantly higher returns when investing accurately. It is important to note that both strategies mentioned above were compared with the buy and hold. Additionally, two other models from the classical time series analysis and chart analysis were used to compare the machine learning models, the ARMA-GARCH, and cross signal, models. The ARMA-GARCH model belongs to the classical time series models, while the cross signal strategy is found in the chart trader. All models were experimented and tested with the same data to enable a complete and correct comparison, not excluding the prediction horizon is harmonious for all models. Financial indicators are used individually or combined in the chart analysis to develop a trading strategy by using trading rules that include these financial indicators. The theory behind the machine learning models is to give exact financial indicators in the hope that the models find patterns in the data to predict the up and down signal for the next day. By looking at 3, one can find all the input variables used for all machine learning models. The first four input variables are features used uniformly for all ML models. The label "signal" was used for classification and Close Price for regression. The labels were each shifted back one row to allow for one-day prediction. Since the values, "features" should predict tomorrow's value of the label $Label_{t+1}$.

Table 3: Description of input variable

S. no.	Input variable formulation	Type
1	RSI	Feature
2	ROC	Feature
3	MACD	Feature
4	K-percent	Feature
5	Signal	Label
6	Close Price	Label

Other input data were used for the LSTM, ARMA-GARCH, and the Cross Signal Model. This is explained in more detail in the respective section.

4.1 Architecture of the Algorithms

The architecture of the algorithms used in this work is shown in figure 7. The process flow for all ML algorithms as well as for the traditional time series methods can be divided into five categories. First comes the data acquisition, where considerations must be made such as what software will be used to process the data, where will the data come from, and what type of data will be needed. In this work all computations and analyses were compiled with the programming language Python. The data was downloaded directly as a data frame via a package integrated in Python (yfinance).

In the second step comes the data preparation, here it is important to examine the data carefully because often the data are incomplete and filled with NA values. If this is the case it is necessary to check where the affected rows are and delete them if possible. If there are many NA values, it is advisable to repeat the first step and load the data in another database. If this is also not possible, the NA values can be processed with missing-data algorithms. Once the data has been processed so that there are no more NA values, the data can be manipulated and extended with new features. In trading, financial indicators are often used to gain a better understanding of the time series. For certain ML algorithms it is better if the data is normalized. That means you scale the data for example with the minmax scaler. If it is not clear in advance which ML algorithm will be used for the analysis, the normalization can be done later. Because many ML algorithms are classification algorithms it is important to create a feature that has a class. In this work a signal was created as a class that encodes the returns with zero and one. Third, a model is chosen and then created

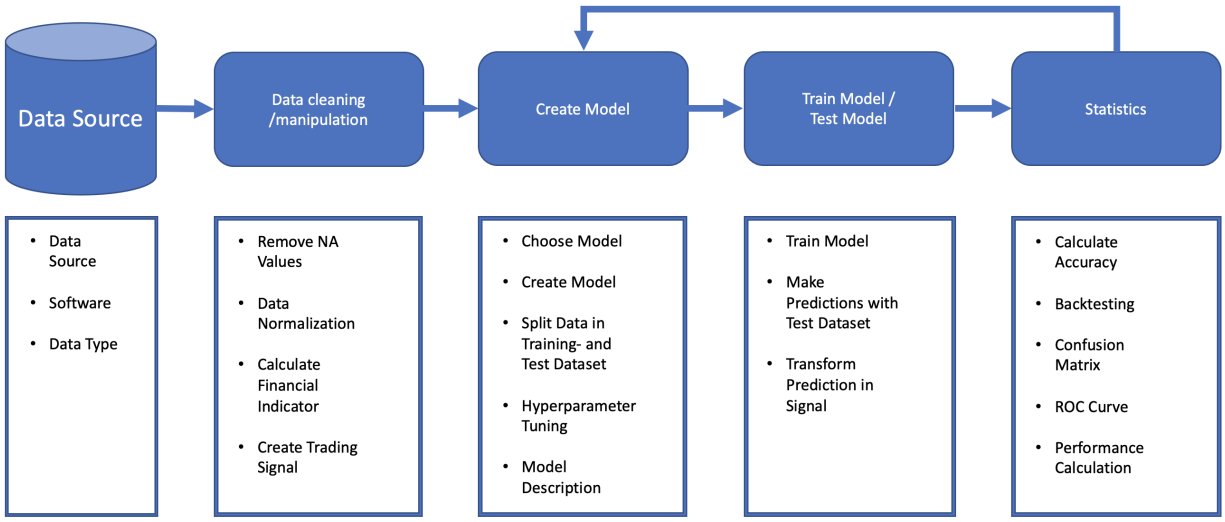


Figure 7: Architecture

in a programming language. It is advisable to look closely at the functions of each algorithm. Most programming languages have web pages where you can read about how the functions work, what parameters exist and examples of use. With this information you can then select the relevant features and also the target variable also called label. With most algorithms there are functions with which one can realize the hyperparameter tuning simply. To realize hyperparameter tuning, the data set must first be split into a training and a test data set. Since hyperparameter tuning needs the data to find the optimal parameters for the given data set. Once these steps are done, the model can be built. For the fourth step you have to pass the whole training data-set to the model, i.e. all features and the label. With this data the model is trained. Depending on the model this can take some time. After the model is trained, we can make a prediction with the test data-set. The model is passed only the features of the test data-set. The output of the model is then the prediction of the label values. At last comes the fifth step, here we compare the results of the prediction and the actual values. In this work, we have always used the same comparison method, whether regression or classification. Here we had to convert the results of the regression back into a signal. The usual methods are, the calculation of the accuracy, the confusion matrix. If the results of the backtesting are satisfactory, the performance of the different trading strategies can be calculated. If the results are bad, you have to go back to the third step. [27]

4.2 Benchmark Models

In this chapter, the results of the ARMA-GARCH model and the cross signal strategy are presented. A detailed procedure of how everything was done can be seen in the appendix, in this chapter only the final result are presented.

4.2.1 ARMA-GARCH

The application of ARCH/GARCH models has become unthinkable in the financial industry. First, a detailed analysis of the financial time series was done, which showed that the data are not normally distributed and that they follow a Students-t distribution. This was supported through a QQ-Plot analysis. The Jarque-Bera test confirms the assumption of no normal distribution. Figure 1 shows the prices and log returns of the Brent crude oil time series. By chart analysis, no significant trend is apparent on average for the time being. And performing the Augmented Dickey-Fuller test revealed no significant evidence of non-stationarity. Subsequently, the autocorrelation functions ACF and PACF are performed to determine the model order. Here we rely on the Bayesian information criterion BIC. The best model was an ARMA(1,1), which had the lowest BIC. However, if we look at the log returns, we see that there are various volatility clusters, which is due to the heteroscedasticity. With the GARCH(1,1) we could capture the heteroskedasticity well which led to a very good performance. For training and testing the model, log returns from Brent crude oil are used. In the next step, the data were split into 80:20 proportions with which the ARMA(1,1)-GARCH(1,1) model was trained (80% of data) and the predictions were converted into binary numbers -1 and 1 using the signum operator. With a positive signal you are long, with a negative signal you are short. Thus we get the signals with which the trading performance can be calculated. The signals can be seen in the figure 8. The model was able to recognize the volatile market phases well. The accuracy of the predicted signals was 54.27%, which is good. Afterwards a detailed extreme value analysis was don, to see how the model have performed in high volatility phases and in low volatility phases. This analysis can be seen in chapter 4.3.5, which compares all used models.

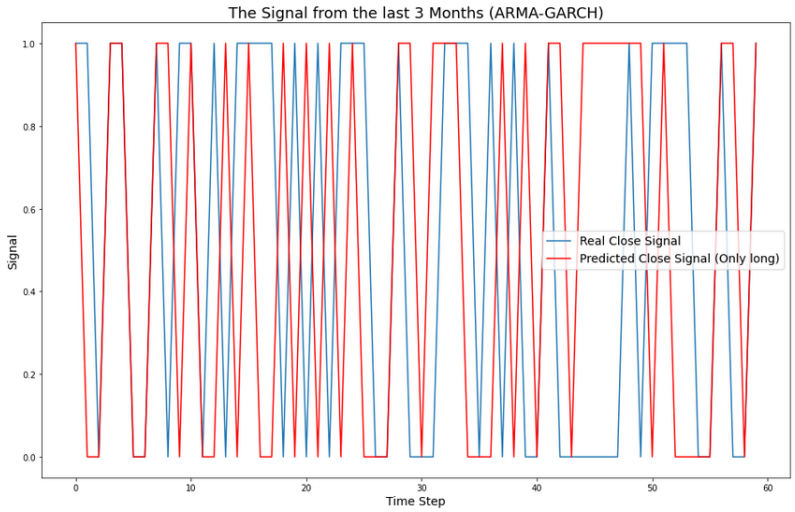


Figure 8: Signals of the last three months

If we now look at the signal plot, we see that the predicted signals often fluctuate, this means that the model is very sensitive to the high volatility. Occasionally, longer phases are also visible which the model remains long.

Figure 9 shows the performance plot of the three strategies (Only Long, Long-Short and Buy and Hold). Over a period of approximately 2 years, the ARMA-GARCH strategy has performed significantly better than the Buy and Hold strategy. Furthermore, in the case of the Long-Short strategy, it can be seen that it has produced the largest gain between 2020-02 and 2020-06. This shows that the ARMA-GARCH model is good at capturing high volatility phases in a turbulent market such as the Covid 19 crisis.

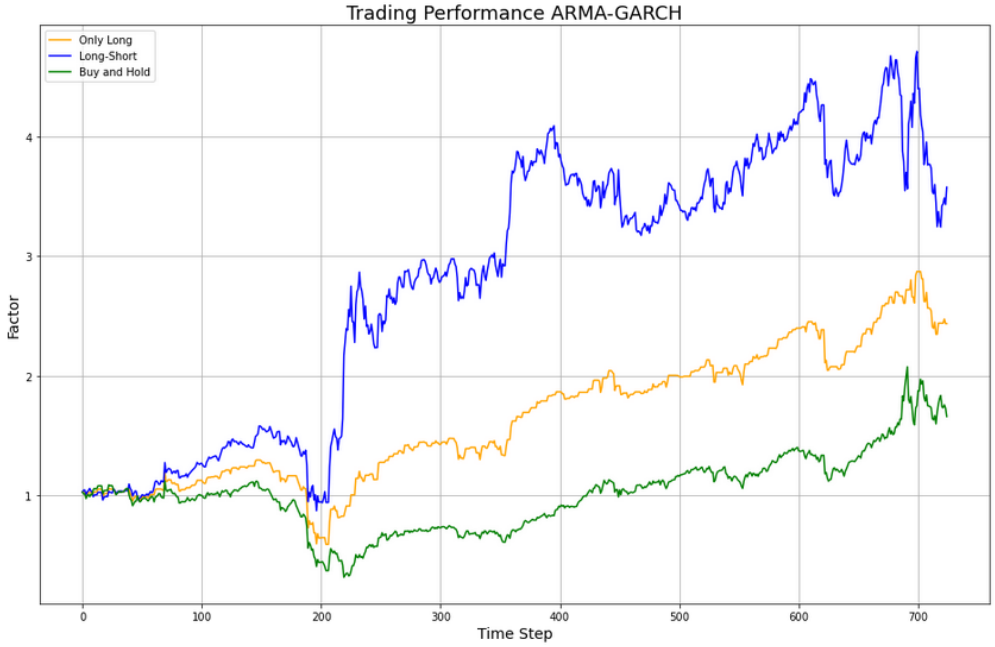


Figure 9: ARCH-GARCH Performance-Plot

The long-short strategy performed best with a Sharpe ratio of 0.88 and a profit factor of 3.82. The strategy significantly outperformed the buy and hold, which achieved a Sharpe Ratio of 0.36 and a Profit Factor of 1.73. The Only Long strategy with a Sharpe Ratio of 0.77 and a Profit Factor of 2.55 has also significantly outperformed the Buy and Hold strategy. Thus, one last comment to make about this model is that the ARMA-GARCH model performed better in choppy markets than in calm ones.

Table 4: Performance of ARMA(1,1)-GARCH(1,1)

Trading strategy	**Sharpe Ratio**	**Profit Factor**
Buy and Hold	0.36	1.73
Only Long	0.77	2.55
Long Short	0.88	3.82

4.2.2 Cross Signal

Cross signal trading is a very common method used by chart traders. In this example, a very simple model was developed. Using the close price, two simple moving averages (SMA) were formed. The first with a 3 week interval (15 days) and the second with a 9 week interval (60 days). SMA are so called smoother's which makes sure that the time series are not so volatile. The larger the SMA the smoother the time series. Thus, it is possible that strong trends become apparent. The problem with large SMA is that the time series takes a long time to recognize a change in direction. Therefore it is important to work with two such SMA. After creating two SMA's, the trading rules have to be determined. In this example very simple rules were determined.

The trading rules are: If the MA15 crosses the MA60 from the bottom to the top, this is a buy signal. However, if the MA15 crosses the MA60 from the top to the bottom, it is a sell signal. This method belongs to the trend strategies. In the first model only buy or sell was enabled, in the second model also the shortselling.

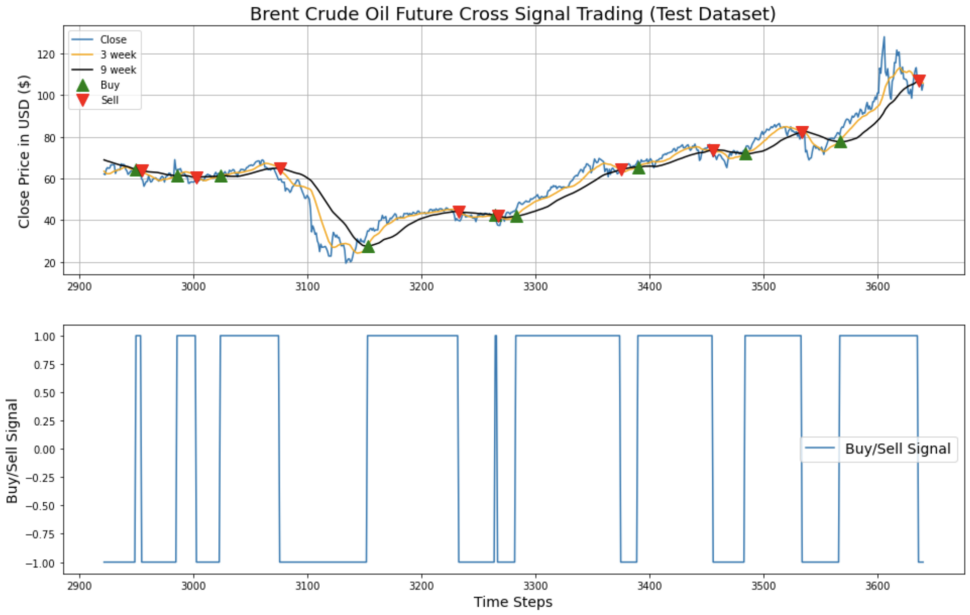


Figure 10: Cross Signal Trading with Testdata

In figure 10 you can see the original time series in blue and the MA15 and MA60 in orange and black. The green arrows signal a buy signal and the red a sell signal. In the first phase of the original time series there is no clear trend, here we have an up and down movement of the time series with an average of 60 USD. In such phases, cross signal trading is not very performant because this trading strategy needs trends to achieve good results. After this first phase, the time series starts to have clear trends and here the cross signal trading can make good decisions.

Also in the performance plot (figure 11) the initial phase is well visible. Only-Long as well as Long Short perform poorly. After that, a clear upward movement in the performance is evident and here you can also generate good returns. If you look closely, you can see that the only-long strategy in certain phases completely exits. This can have a positive or negative impact on performance. Nevertheless, it is a good strategy because it has a lower risk than the long short strategy.

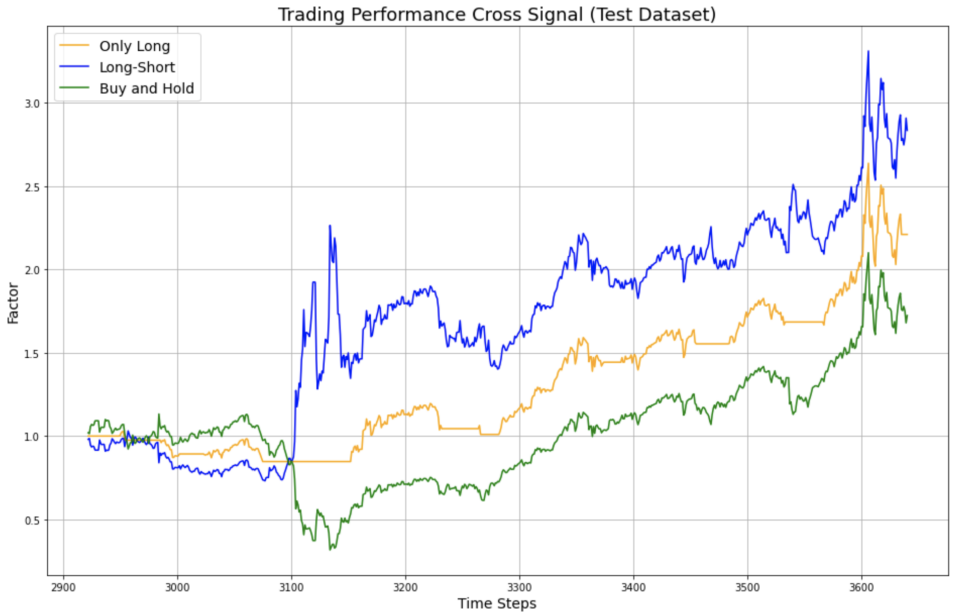


Figure 11: Cross Signal Trading Performance Plot

Both strategies perform better than the buy and hold which can be seen in table 5. Only Long achieves a better Sharp Ratio than Long Short which confirms the statement that the Only Long strategy has a lower risk than Long Short. The profit factor is highest with the Long Short strategy, this is because this strategy is also profitable when the original time series has a downward movement.

Table 5: Perforance Table of Cross Signal trading

Trading strategy	Sharpe Ratio	Profit Factor
Buy and Hold	0.36	1.73
Only Long	0.95	2.21
Long Short	0.69	2.83

4.3 Machine Learning Models

This chapter contains the results of the machine learning models LSTM, Random Forest, Support Vector Machine Regression (SVM) and k-Nearest-Neighbor (kNN). First, all predictions of the respective models are presented, followed by a classification report, which evaluates the accuracy of the predictions. Then the Feature Importance is performed, which shows how much each feature contributes to the model. Then the performance of the individual models is presented and finally the cross validation and a extreme value analysis were performed.

4.3.1 The structure of the models and their predictions

In the first step, we have to determine the model structure and their predictions. This will be shown in this section.

Long short-term memory

The LSTM networks can solve the problem of vanishing gradient, which ordinary RNN networks have. This thanks to the property that these networks can track the long-term dependencies in data. For more detailed examples see "Deep Learning; Adaptive Computation and Machine Learning Series" which was published by Goodfellow et al. 2016 [28]. We first fit a GARCH(1,1) model to find the number of lags using the standardized residuals. In the previous chapters, the distribution of Brent crude oil prices was shown to follow a Students-t distribution, so the GARCH model was fitted with this assumption. The standardized residuals should not show any dependence structure, if this should be the case, then we know how many lags are needed to explain $t + 1$. The ACF analysis showed that the data must be lagged for 39 days. For the LSTM algorithm we create lagged data-sets so that we have a kind of a linear function who train the neural network. The LSTM creates the optimal parameters in each node to predict the next day with the past data. After we have to split the Close Price in a train and test data-set with a 80:20 ratio. The data used has been normalized. This is to improve the performance of the model, bringing the values of numerical columns to a common scale. Hence, the training data set was transformed to a scale of zero and one. The architecture of the LSTM model consists of two LSTM layers and two Dense layers. The LSTM layers each have 128/64 neurons and the Dense layers each have 25/1 neuron, the latter being the output neuron. Each of these LSTM layers uses the activation function tanh and the recurrent activation function sigmoid. The tanh function allows the negative values to be retained. Additionally, "return_sequences=True" was set in the programming, this is required for stacking LSTM layers so that the subsequent LSTM layer has a three dimensional sequence input. The optimization of the model was optimized with the Adam optimizer (Kingma and Ba, 2017) [29] , which is a stochastic gradient descent optimizer with momentum and the loss was set as mean square error. Then the model is trained 5 epochs, the epochs describe the number of times the learning algorithm is run through the entire training set. The batch size is set to one so that the optimizer can work more efficiently. The hyperparameters are listed in the following table 6:

Table 6: Values of hyperparameter

Hyperparameter	Selected Value
No. hidden layers	4
No. neurons	128 / 64 / 25 / 1
Activation function	tanh
Recurrent activation function	sigmoid
Optimizer	Adam
Dropout rate	0
Batch size	1
Loss function	MSE
Learning rate	0.001

The predictions of the LSTM model can be seen in figure 12. To get the predicted signals, we had to convert the predicted closing price into log-returns. Afterwards these log-returns were converted into 0 and 1 signals with the signum operator.

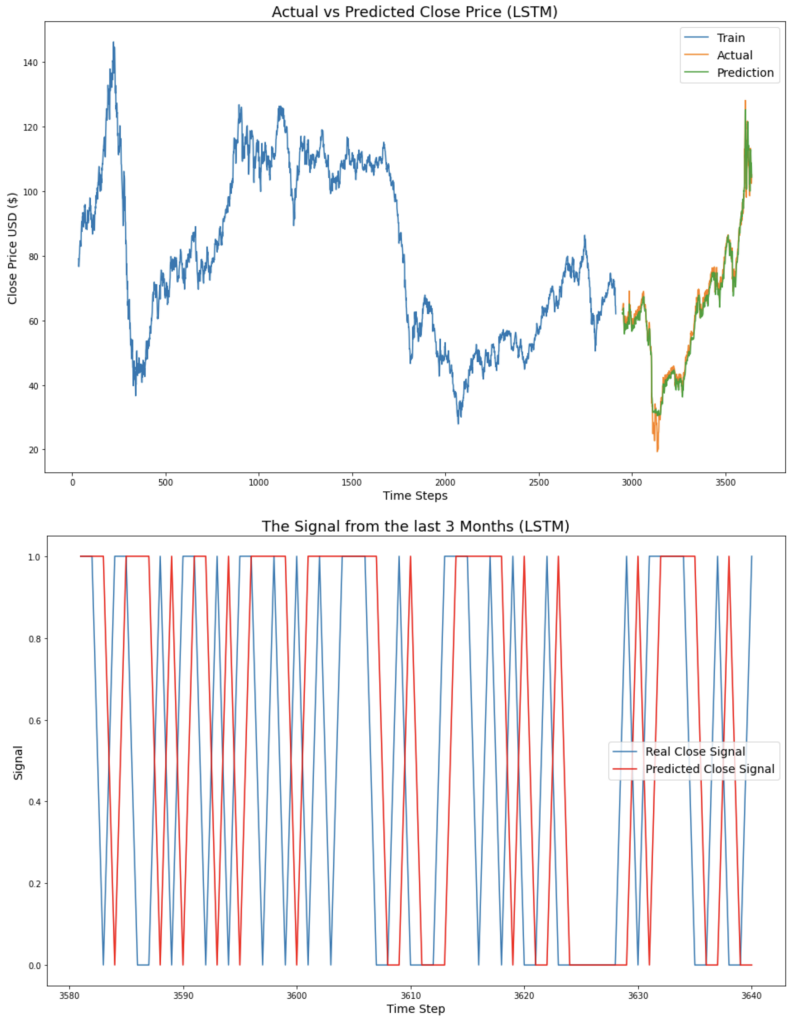


Figure 12: LSTM Prediction and LSTM Signals

The model correctly predicted the signals with a accuracy of 51.81%. Furthermore, it can be seen that the model tends to be positioned long in calm market phases and sings between long and short in volatile phases.

Random Forest

This section presents the results of predicting trading signals for Brent crude oil. The Random Forest classifier works very simply: it creates a set of decision trees from a randomly selected subset of the training set. Then, the votes of the different decision trees are combined to determine the final class of the test object. In this work, a Random Forest was used for classification. Financial indicators were defined as features to capture the characteristics of Brent crude oil, while the Random Forest (RF) model is used to train on the training data-sets according to a certain classification criterion and make predictions for the next trading day. [30]

For the Random Forest model, the data set was split into training and test data in the ratio of 80/20. Then the tuning of the hyperparameters was performed in Python using the RandomizedSearchCV command. The following table 7 shows what the command outputted.

Table 7: Values of hyperparameter (Random Forest)

Hyperparameter	Selected Value
No. of trees	169
No. of features to consider at every split	2
Maximum number of levels in tree	4
Minimum number of samples required to split a node	49
Minimum number of samples required at each leaf node	1
Criterion	entropy

The Random Forest model was initialized with these parameters and then trained. Then a prediction of the labels was made with the test data. In figure 13 you can see the signal predicted by the random forest model and the confusion matrix. It is obvious that the model predicts up days in the largest part and only a few down days. This also explains the high recall value for the Up Days, which can be seen in a later chapter.The Random Forest model achieves the highest accuracy with 54.73%.

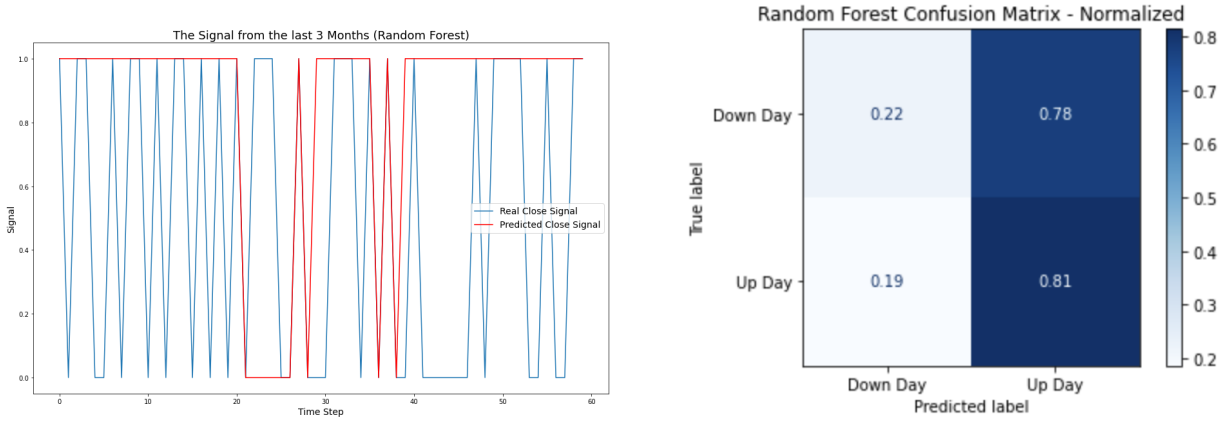


Figure 13: Signal Plot (left) and Confusion Matrix (right)

Support Vector Machine

For signal prediction, a SVM regression model with a radial basis kernel was used. The SVM can solve the same problems as ordinary linear regression. The main difference is that SVM can also solve non-linear problems. By tuning hyperparameters, it is possible to develop very flexible and robust models. In the SVM algorithm, there are two specific parameters that must be estimated. These are C and ϵ. The parameter C is a regularization parameter, which regulates the distance ξ to the points outside the hyperplane. ϵ determines the width of the hyperplane. To find the best value for C, a list of 100 values from 1 to 100 was generated in Python and for ϵ a list of 0.1 to 50.1 with a sequence of 0.1. Subsequently, the best values for C and ϵ are searched with the hyperparameter Tuning. For this model, the features from chapter 4 were used. The main difference to the other models is that the label is now based on the $ClosingPrice_{t-1}$. For this model the data was again splitted into training and test data with a 80:20 ratio. Afterwards a hyperparameter tuning with cross validation was done, this was done in Python with the command RandomizedSearchCV. The following table 8 shows the best parameters that lead to the best model.

Table 8: Values of hyperparameter (SVR)

Hyperparameter	**Selected Value**
Shrinking	True
Gamma	Auto
ϵ	22.4
C	19
Cache size	41

With the hyperparameters from the table 8 the SVM algorithm was trained. After this a prediction was made of the labels with the test data. In the figure 14 the predicted signals and the confusion matrix are shown. The SVM model achieves an accuracy of 49.18%. The SVM model rarely has long periods in which it is in the same class (0 or 1), it fluctuates constantly. This is also shown by the confusion matrix, the two classes are approximately equally distributed, the class Up day (i.e. 1) is predicted a bit more often than the class Down day (i.e. 0).

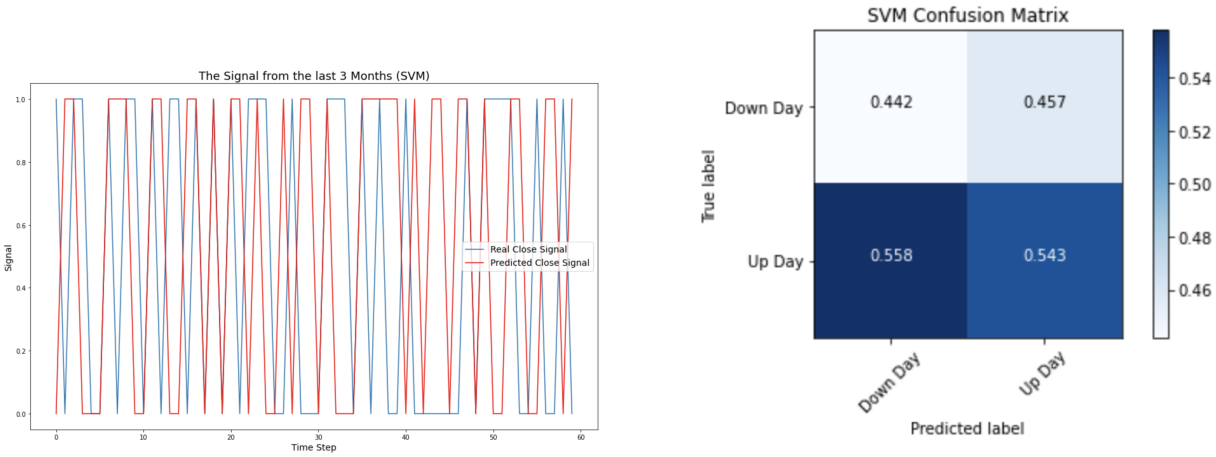


Figure 14: Signal Plot (left) and Confusion Matrix (right)

k-Nearest Neighbor

In this section, the K-Nearest Neighbor (kNN) model is applied to Brent crude oil futures to predict a trading signal for a sample of test data. The kNN was applied as a classification model. All financial indicators presented in chapter 3.1.2 were used as feature inputs. The generated signals from the whole data-set were used as labels. Furthermore, the input features were normalized, these were scaled between 0 and 1, so that the algorithm can work more efficiently and the calculation of the distances is simplified. The Manhattan distance was used, then the data was splitted into test and training data. Hyperparameter tuning table 9 was done to see which hyperparameter leads to the best fit of the model. Then the model was validated and checked how well it performed. This was done using accuracy score, which we rely on it in a later step. To get more details about the performance a classifications report was made. Also the distribution of the predictions is looked at with the confusion matrix. A Feature Importance analysis shows the explanatory power of the input features on the model, this will also be looked at later in chapter 4.3.3. Finally, the final results are discussed. For the model, the data set was split into training and test data in an 80:20 ratio. Then the hyperparameter tuning was done in Python with the RandomizedSearchCV command. Therefor, lists were created for the number of leafs, the distances, for the weights and for the determination of K, which were then tested to obtain the best possible parameters. The final result is shown in the table below.

Table 9: Hyperparameters after tuning the kNN Model

Tuning Parameter	Selected Value
Leaf size	15
Distance	Manhattan
Weight	Distance
No. K	5
Algorithm	Auto

In the figure 15 the predicted Signals vs. the real signals and the confusion matrix are shown. The signals fluctuate very firmly with this model. Occasionally, there are periods in which the model remains in the same class for a certain time. However, the confusion matrix shows that the two classes (0 and 1) are very balanced.

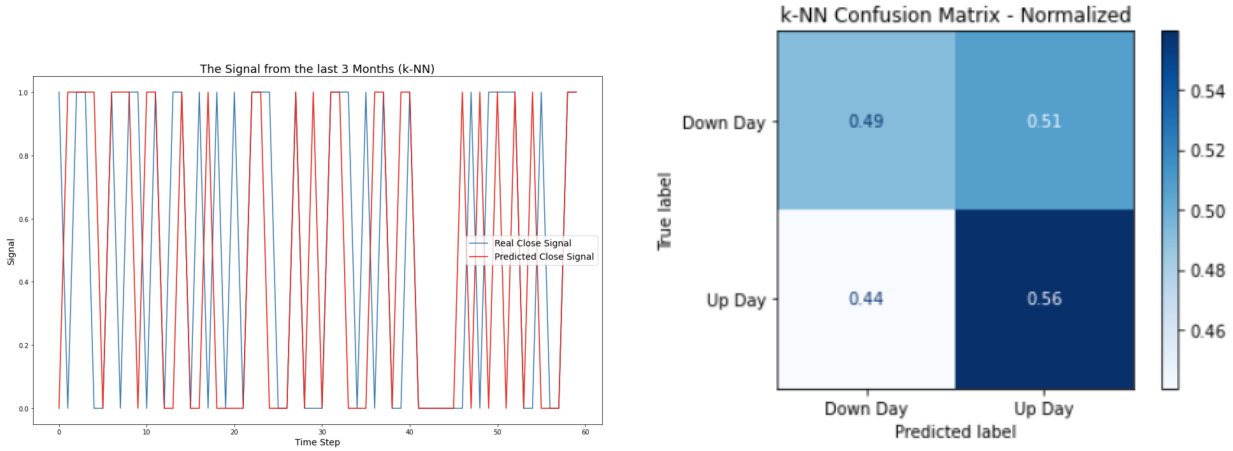


Figure 15: Signal Plot (left) and Confusion Matrix (right)

4.3.2 Classification Report

To measure the quality of predictions of the Machine Learning Models a classification report was made. This is shown in the table 10. The report shows the main classification metrics accuracy, recall, f1 score and support per class. The metrics are calculated using true and false positives, true and false negatives. The precision tells how many of all positively classified instances were classified correctly. Now consider the recall, this describes what percentage of all cases that were actually positive were correctly classified. For the F1-score, the best value is 1 and the worst 0. How all this was calculated can be seen in the appendix.

The precision of the Random Forest model is for Down Days slightly better then for Up Days, which doese not mean, that the model can better predict the Down Days, if we rely on the recall, which significantly better performed for UP days than for Down days, which means that the Random Forest model can predict the Up days better than the Down days. If we are looking on the precision of the SVM model we see that the precision is bit better for the Up days than for the Down days, which means that the model predicts the Up days better than the Down days. Because the recall makes no significantly better difference between the Up and Down Days.

If we look at the kNN Model, the class Down Day, which corresponds to class 0, we see that it was classified correctly 47.6% of the time, compared to Up Day, which corresponds to class 1. The precision was classified 57.5% correctly. This means that if the model predicts class 1, we can be more confident that it has been correctly predicted. Now consider the recall, for class 0 the recall was 49.2% and for class 1 56%, this means that for class 1 56% of the positive cases were actually positive. When looking on the f1-score th class 1 is better predicted than class 0.

The conclusion of this classification report is that if the model predicts class 1, we can trust it more than if it predicts class 0.

Table 10: Classification Report of all ML models

	Precision	**Recall**	**F1 score**	**Support**
Down Day				
LSTM	0.460	0.389	0.422	311
RF	**0.490**	0.223	0.307	337
SVM	0.441	**0.492**	0.465	327
kNN	0.476	**0.492**	**0.484**	327
Up Day				
LSTM	0.554	0.624	0.587	378
RF	0.562	**0.811**	**0.664**	402
SVM	0.543	0.491	0.516	401
kNN	**0.575**	0.560	0.567	402

4.3.3 Feature Importance

Feature Importance was determined using permutation. The permutation works as follows. The model is run without modification to determine the accuracy. In a second step, the values of one feature are changed at a time to find out how sensitive the model reacts to changes in a specific feature. In this way it can be determined which feature has a large influence on the accuracy.

In figure 16 you can see the result of the feature importance of the RF, kNN and SVM model. If we are looking on the RF Plot, all features have an influence on the accuracy with the feature MACD having the largest influence with 44% and K-Percent with 10.6% the smallest. It is difficult to say why MACD has the biggest influence on the accuracy. When looking at the distribution of the MACD feature (Section descriptive Statistics) we can see that the distribution is normally distributed with a large tail in the negative direction. Looking at the feature K-Percent it is obvious that the distribution is bimodal. This could also explain why this feature performs worst here. If we have a look on feature importance plot of the SVM model. RSI has the an influence on the model with 35.1%, followed by K-percent, which has the biggest influence on the model with 35.1%. MACD also has an influence on the model with 29.5%. ROC has the smallest influence with 0.1%. Last, the feature importance plot of kNN is considered. K-percent contributes the most to the model with 28.8%, followed by RSI with 27.7%. MACD has 25.7% share and ROC with a share of 17.9%. kNN has the most balanced feature distribution. These proportions are similar for the kNN Model.

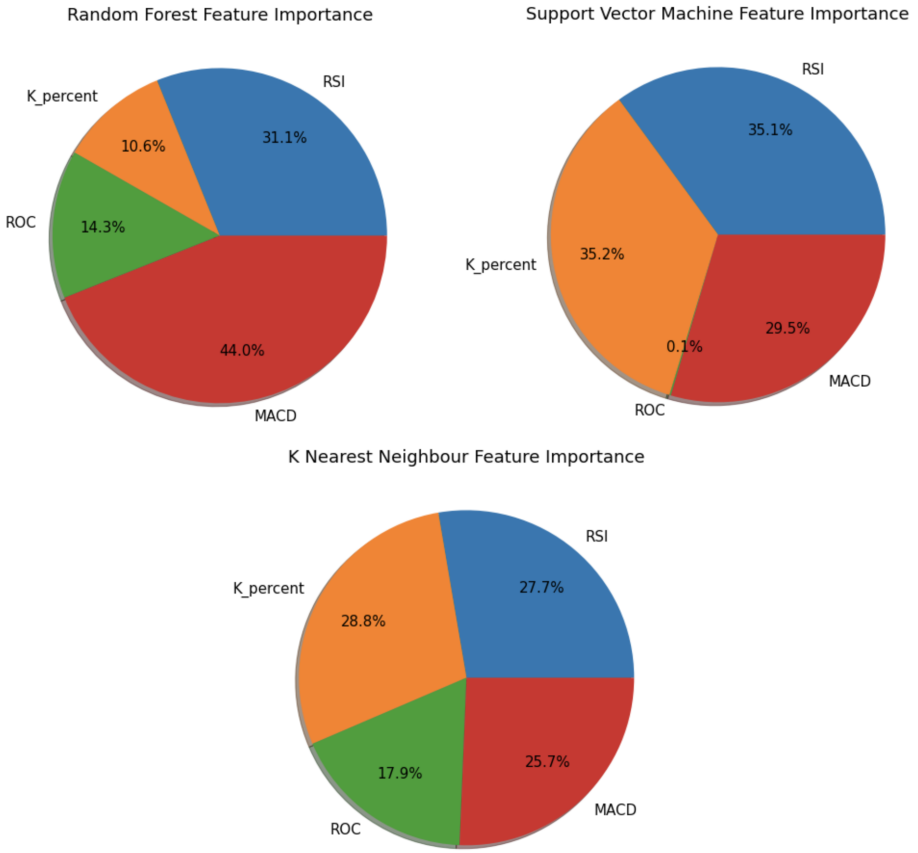


Figure 16: Feature Importance

4.3.4 Trading Performance

In this chapter, all performance results, including the benchmark models, are visualized and discussed. In addition, the monthly returns from the test phase are visualized to see in which market phases the different models performed best. In this analysis the profit factor is considered, but the risk of each strategy and performance evaluation is essential. This will be discussed in later chapters, but for now, only the Profit Factor will be discussed. The first thing to see is that the ARMA-GARCH model performed very well with both the long-short and long-only strategies. This gives us to understand that the ordinary statistical models are still competitive. Again, it gives us to understand that the ML methods still need to be further developed. Nevertheless, we consider the figure 17 with the results. Let's start with the long short strategy. It is pleasing to see that every model outperformed the Buy and Hold strategy. The model that performed the best was the RF model. The profit factor gap with the next best model, namely the ARCH-GARCH, is clearly visible. Surprisingly, the kNN model did not perform as well as hoped. Furthermore, the LSTM model rash at the beginning of the Corona crisis is visible. This shows that the model has recognized the characteristics of the time series at this time very well. However, this positive trend suddenly collapses at the beginning of the Ukraine war and loses almost 50% of the returns. Although the Brend Crude price had a positive trend at most of the time, except for an outlier at the very end of the time series, the LSTM model could not capture and detect this. Looking at the Only Long strategy, we see that the performance has a much more constant positive trend and there are very rare performance dips. Again, the RF model has performed the best. And each model has outperformed the buy and hold. Consequently, the risk assessment of the Only Long strategy will be better than that of the Long Short strategy, which intuitively makes sense.

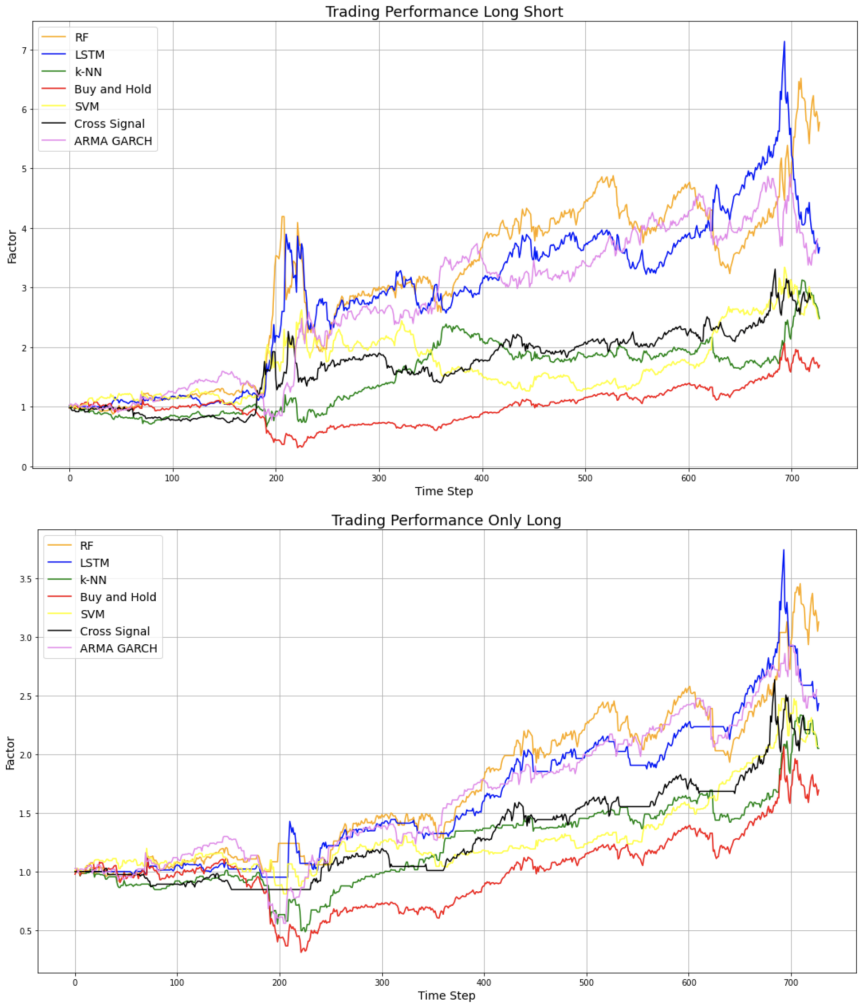


Figure 17: Performance Plot

Next, we look at the monthly returns of the individual models. Here the characteristics of the models can be analyzed. Often it is important to know in which market phases (high volatility / low volatility phases) the models perform best or worst. This can be seen in figure 18 . The monthly returns of the individual models reflect the global political situation on earth. In the period from 2019-12 to 2020-06 the panic on the financial markets was extremely noticeable. The large investors and private investors did not know at that time how the governments would position themselves regarding the Corona policy. The ARMA-GARCH model reflects this uncertain period very well. It can be seen that returns plummeted sharply in 2019-12. This is also seen in all other models apart from Cross Signal Trading. In this regard, the ARMA-GARCH model went downhill and then came the biggest gain. On 2020-04, the ARMA-GARCH model made almost 50% of the total gain. The difference between the machine learning models and the conventional statistical models is that the machine learning models detected the pattern one month earlier (as early as 2020-03) than the ARMA-GARCH model. Nevertheless, the benchmark model performed very well throughout the out-of-sample period. Cross signal trading also performed very well during the Corona crisis. If we now look at the Random Forest model, we see that there were only a few losses. The model recognized very well in almost all market situations the trend patterns and also achieved the largest profit with 125% on 2020-03. The LSTM model performed very well at the beginning of the period. It recognized the Corona pebbles very well and achieved the biggest profit of all models. However, the model loses performance in the final period. Why this happened is difficult to explain. The kNN model performed moderately. It is obvious that it did not perform well during the Corona crisis. It seems to perform better in calmer market phases than in highly volatile phases. The SVM model, on the other hand, seems to be the most balanced model. This is because it either makes high losses or profits in every market phase. If you now look at the table 11 you can see the biggest gains and the maximum drawdown of the models.

Table 11: Maximum drawdown and maximum profits

Strategy	Model	Maximum Drawdown	Maximum Profits
Long-Short	ARMA-GARCH	-0.33	0.96
	Cross Signal	**-0.17**	0.8
	LSTM	-0.33	0.8
	RF	-0.42	**1.23**
	SVM	-0.22	0.37
	kNN	-0.23	0.59
Only Long	ARMA-GARCH	-0.56	**0.53**
	Cross Signal	**-0.11**	0.17
	LSTM	-0.13	0.17
	RF	-0.22	0.24
	SVM	-0.21	0.15
	kNN	-0.42	0.33

If we now look at the largest losses and the largest profits, we see that the RF model has made the largest profit and the largest loss in the long-short strategy. Despite the loss, the RF model has the highest profit factor overall. In the Only Long strategy, the ARMA-GARCH model made the highest monthly profit. However, the model also suffered the largest loss. The risk can be considered in different ways, if we look at the model with the lowest loss, we see that it is the Cross Signal strategy, in the Long-Short strategy it has reached a maximum loss of -0.17 and in the Only Long strategy it has reached a maximum loss of -0.11.

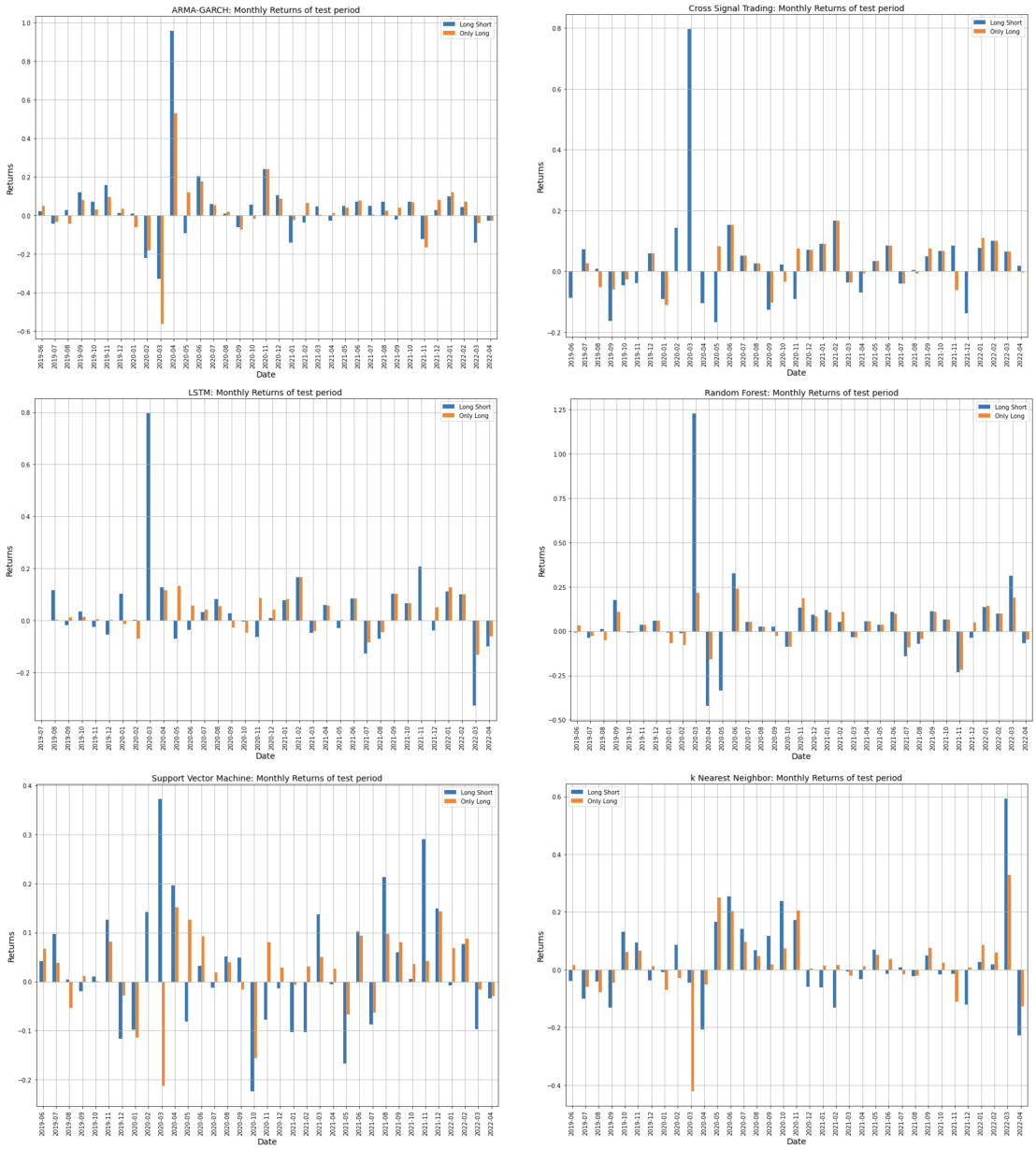


Figure 18: Monthly Returns

4.3.5 Extreme value analysis

In order to explain the performance of the individual models, one has to take a closer look at the accuracy of the extreme values. To see if a specific model performs better in the extreme values or, in other words, makes better decisions. In figure 19 the log returns of the prediction horizon are shown. In addition two intervals are visible. They can be approximated as (in black) the 95% and (in red) the 99% quantile. To analyze how the models perform in these quantiles, the values outside these quantiles have to be filtered out and then the accuracy has to be determined with the filtered values.

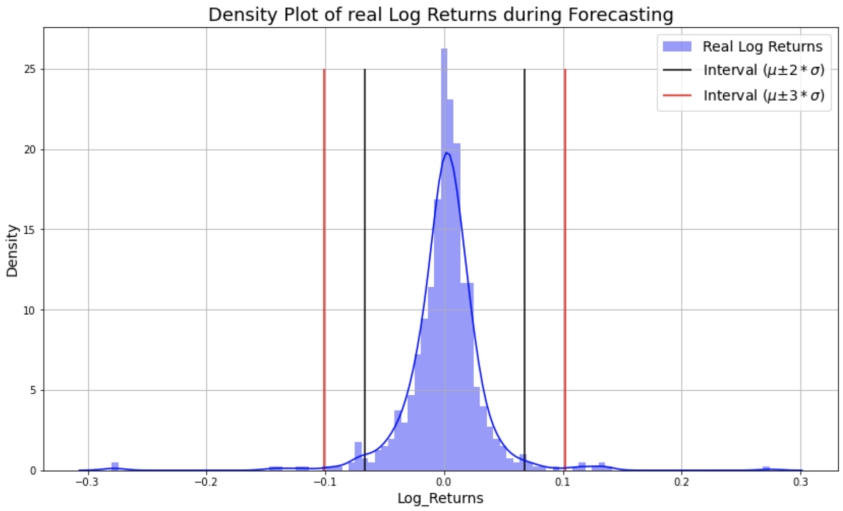


Figure 19: Log Return of Prediction Horizont

In the table 12 the respective accuracy's of the models and their quantiles are visible. With this analysis it is now possible to explain why the Random Forest model performs best. It can be seen that the Random Forest model performs very well in the extreme values. This means that if a good return value is seen the next day, the model is more likely to make a correct prediction. The model allows better entries with large returns, so days with extreme return values are very profitable. The LSTM and ARCH GARCH models also perform very well in these quantiles. This is also evident in the performance. Models that do not perform well at these extreme values are therefore also less profitable.

Table 12: Extreme values accuracy

Model	Accuracy regular values	Accuracy extreme values (5%)	Accuracy extreme values (1%)
ARMA-GARCH	51.3%	52.78%	**66.67%**
LSTM	51.52%	54.84%	64.29%
RF	**54.26%**	**61.11%**	60.0%
SVM	48.91%	47.22%	46.67%
kNN	52.53%	58.33%	46.67%

4.3.6 Cross Validation

Cross validation is used to obtain meaningful misclassification rate. The procedure is the same as for the original model. One determines a training and a test data set, trains the model with the training data and then makes a prediction using the test data set. For cross validation, the data is cut into k pieces of equal size. In each run, one of these pieces is defined as the test data set and the remaining data as the training data set. Then the model is trained k times and tested using test data. Thus, k predictions are obtained, which can then be quantified using a measure such as (MSE, Accuracy, MAPE, etc.). Finally, all these values can be averaged out to get a better measure and to avoid that the choice of data for test and training data was not randomly good or bad. In figure 20 is an illustration of the k-Fold Cross Validation. [31]

Figure 20: k-Fold Cross Validation [32]

In this work, the k-fold cross validation was performed with $k = 5$. It should be noted that the data is time dependent, since they are time series data. The cross validation can be performed if the k-folds keep their order. With the cross validation the robustness of the model can be tested. Nevertheless, the practical relevance is not given because with time series one wants to predict the future data by means of past data. For all models, the Sharpe ratios of the two strategies were calculated. This was also done for the CV, in addition the accuracy of the predictions for each run was determined.

Table 13: k-Fold Cross Validation values

	k = 1	**k = 2**	**k = 3**	**k = 4**	**k = 5**	**Mean**
RF						
Sharpe Ratio Only Long	1.18	0.4	-0.18	0.21	1.18	**0.57**
Sharpe Ratio Long Short	1.7	0.27	0.5	0.06	1.15	**0.74**
Accuracy in %	53.22	52.13	49.66	48.7	54.73	**51.69**
kNN						
Sharpe Ratio Only Long	1.59	0.65	-0.34	0.85	0.57	**0.66**
Sharpe Ratio Long Short	2.01	0.51	0.33	0.99	0.6	**0.89**
Accuracy in %	53.36	52.54	50.75	51.99	52.95	**52.32**
SVM						
Sharpe Ratio Only Long	0.06	1.02	-0.58	0.4	0.78	**0.33**
Sharpe Ratio Long Short	-0.02	0.87	-0.07	0.21	0.58	**0.32**
Accuracy in %	51.17	51.31	52.96	50.81	49.24	**51.11**
LSTM						
Sharpe Ratio Only Long	-0.19	0.72	0.41	1.55	1.03	**0.7**
Sharpe Ratio Long Short	-0.61	0.76	0.17	1.86	0.96	**0.63**
Accuracy in %	50.49	48.68	50.28	47.71	45.91	**48.61**

In the table 13 all results of the CV are shown including the arithmetic mean. Comparing the different models with each other, it is obvious that the classification models perform better than the regression models. The k-NN model with the long short strategy has the best mean Sharpe Ratio and this model also has the best mean Accuracy. You can see that the Accuracy is robust for all models, but the Sharpe Ratios are very different. Many Sharpe Ratios are negative, which indicates a mean loss. If we look at the Sharpe Ratios of k = 5, we see that they all have positive values. The k-fold k $= 5$ is also the regular way of working with time series. The model gets past data to train to predict future data.

4.4 Analysis of results

The Random Forest model shows better results than other models as it is able to correctly predict extreme returns as well as smaller returns. ARMA-GARCH and LSTM also showed very good results. These three models are able to capture the long-term dependencies in the time series and are more suitable for predicting Brent Crude Oil Future. The table 14 shows the Experimental Sharpe Ratios and the Profit Factors. This table confirms that machine learning models are able to perform very well on time series.

Table 14: Performance analysis

Strategy	Model	Sharpe Ratio	Profit Factor
Buy and Hold	-	0.33	1.62
Long-Short	ARMA-GARCH	0.88	3.82
	Cross Signal	0.69	2.83
	LSTM	0.89	3.67
	RF	**1.15**	**5.77**
	SVM	0.59	2.47
	kNN	0.60	2.48
Only Long	ARMA-GARCH	0.77	2.55
	Cross Signal	0.95	2.21
	LSTM	0.95	2.43
	RF	**1.18**	**3.13**
	SVM	0.76	2.07
	kNN	0.57	2.05

5 Conclusion and outlook

This thesis explored how ML methods compare to already successfully established standard methods such as cross signal trading and ARMA-GARCH models in the crude oil market. First, in our literature review, we found that different machine learning methods have their strengths in different areas. Based on these findings, we evaluated the most popular methods in time series forecasting and applied them to Brent crude oil futures. These were compared with the cross signal trading strategy and the ARMA-GARCH model. While the properties of ARMA-GARCH models are suitable for modeling a variable variance, the ML methods can be beneficial for pattern recognition in the data. To this end, measurements were made on the quality of the predictions of each model. The Random Forest was characterized by a very high true-positive rate (recall), which set it apart from the other models. Likewise, the final performance result provided conclusions about the robustness of the individual models. It could be shown in which market phases the models made strong or poor trading decisions. In the case of the kNN model, which failed to detect the Covid-19 crisis, it was shown that not every ML model responded well to different market phases. Comparing this with the RF model, it can be seen that the most significant monthly gain was achieved there across all models. The conventional statistical model - ARMA-GARCH - performed surprisingly well and was on par with the ML models. Nevertheless, the extreme values were analyzed in more detail. It was found that the RF model made the best decisions at the 95% interval level. This shows that the RF model continuously made better decisions than all other models during a volatile market period, which is reflected in the total return. Surprisingly, the ARMA-GARCH model made the best decisions at the 99% level but performed worse on average during periods when the market was "calmer." One direction for future work will be the volatility of stock time series. One difficulty in predicting stock markets arises from their non-stationary behavior. It would be interesting to see how machine learning models perform on denoised data or how these models perform in other crude oil markets.

6 Appendix

6.1 ARMA-GARCH Model

The Augmented Dickey-Fuller test revealed no significant evidence of non-stationarity, with a p-value of 0 and a negative ADF-Statistic of -11.93. Now, let us have a look at the table 15.

Table 15: Descriptive statistics of Brent crude Oil closing Price

Statistic	**Values**
Mean	77.384
Median	73.290
Maximum	146.080
Minimum	19.330
Standard deviation	25.870
Skewness	0.229
Kurtosis	-1.029
Jarque-Bera	193.091 p-Val: 0.0
Observations	3653

Based on the descriptive statistics, the assumption of a normal distribution can refute. The skewness of 0.229 indicates a right-skewed distribution. A further indication of no normal distribution is the negative kurtosis value of -1.029. The Jarque-Bera test confirms the assumption of no normal distribution. Since the p-value is 0, the null hypothesis that the distribution is normal can be rejected. Now let us have a look at the log returns. The QQ-Plot confirms the non-normality implied by the descriptive statistics. Fat tails can be seen at the right and left ends. This can also be seen when looking at the distribution of the Log Returns. However, the question arises of how the data is now distributed. We assume that the data follow a Students-t distribution, which the QQ-plot confirms. Nevertheless, outliers must be expected. This is shown in the figure 21.

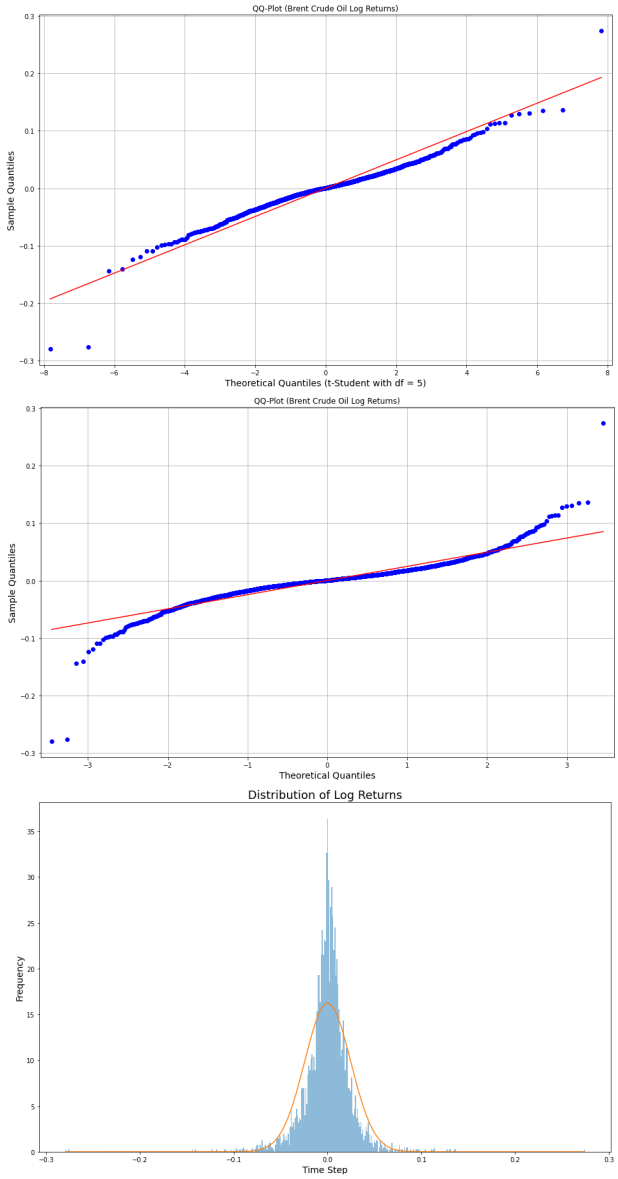


Figure 21: Log Return Distribution and QQ-Plot's

Dependency structure

Now the dependency structure is considered in more detail. The model order is determined with the help of the autocorrelation functions ACF and PACF. To examine this in more detail, one considers the following figure 22 :

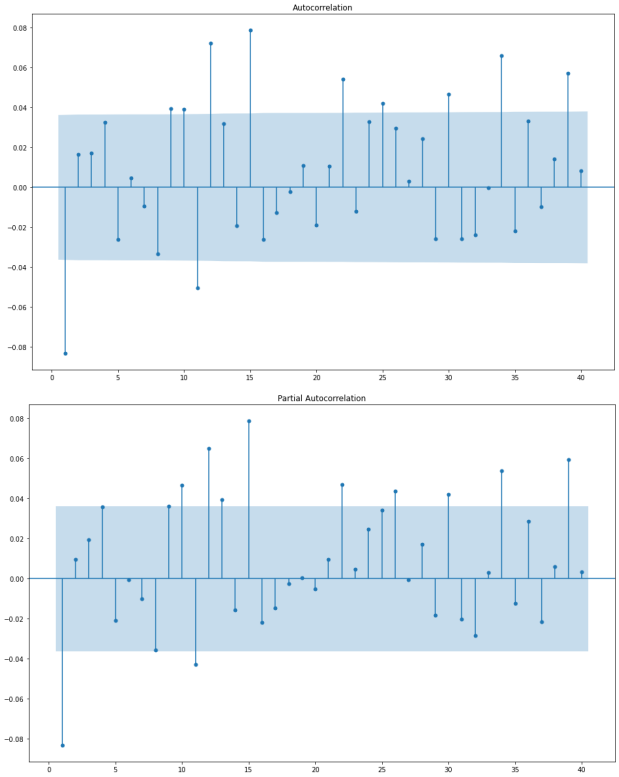


Figure 22: AFC and PACF

The ACF shows dependence structures up to lag 49, but these are not considered because it is not assumed that data 49 days in the past have a significant influence on tomorrow's return. The same is also evident for the PACF at lag 44. Another reason for using minor possible model orders is model complexity. It was essential to keep the Model as simple as possible to explainable, and the result is not overfitted. Therefore, one assumes the model order ARMA(1,1). This will be examined in more detail in the next chapter using the Bayesian information criterion.

Building the ARMA(1,1)-GARCH(1,1) Model

The ARMA(1,1)-GARCH(1,1) model is built in this section. GARCH processes are stationary (but not iid) white noise. In case the data is autocorrelated, the processes are misspecified. If this case happens, the pure ARMA and GARCH processes are defined as follows:

$$y_t = \mu + \sum_{i=1}^{p} a_p y_{t-p} + \epsilon_t + \sum_{i=1}^{q} b_q \epsilon_{t-q} \tag{12}$$

$$\epsilon_t = \sigma_t u_t \tag{13}$$

$${\sigma_t}^2 = d + \sum_{i=1}^{n} \alpha_n \sigma_{t-n}^2 + \sum_{i=1}^{m} \beta_m \epsilon_{t-m}^2 \tag{14}$$

where y_t describes the original data (typically non-stationary) and ϵ_t corresponds to the stationary log-return. This Model generalizes the ARMA family to the case of conditional heteroskedasticity (volatility clustering) and generalizes GARCH type models to processes with non-vanishing autocorrelation: y_t is called the $ARMA(p,q)-GARCH(n,m)$-process. The identification of the model order now involves p, q, m, n. In many cases $n = m = 1$ is used for financial data and in the case that $p, q \leq 1$, this serves as a good benchmark.

If one relies on the ACF/PACF analysis, one can see that residuals are significantly above the level at the lags 1, 4, 8, 10, and 15. One can use the BIC criteria to estimate the best possible Model Order. The ARMA(1,1) model has the lowest BIC of -14002. This confirms that we should use as simple an ARMA model as possible. To say something about the diagnostics of the Model, one can look at the Ljung box statistics in the table 16.

Table 16: Ljung-Box Statitic of ARMA(1,1)-GARCH(1,1)

Lag	LB-Statistic	p-Value
10	18.917444	4.132388e-02
15	61.224745	1.551223e-07
20	64.187634	1.571243e-06

The Ljung box Statistic looks good, as all p-values are below the 0.05 level.

Parameter erstimation of ARMA(1,1)-GARCH(1,1)

All estimated parameters are shown here (see table 17).

Table 17: Parameter Estimation of ARMA(1,1)-GARCH(1,1)

Parameter	**ARMA(1,1)**	**GARCH(1,1)**
a_1	-0.1805	-
α_1	-	0.0607
b_1	0.993	-
β_1	-	0.9386
Log Likelihood	7020.987	-5868.3
BIC	-14002.09	11776.5

After modeling the ARMA(1,1) Model, the residuals are passed to the GARCH(1,1) model. Thus, the resulting conditional volatility can be calculated. This can be seen in the figure 23.

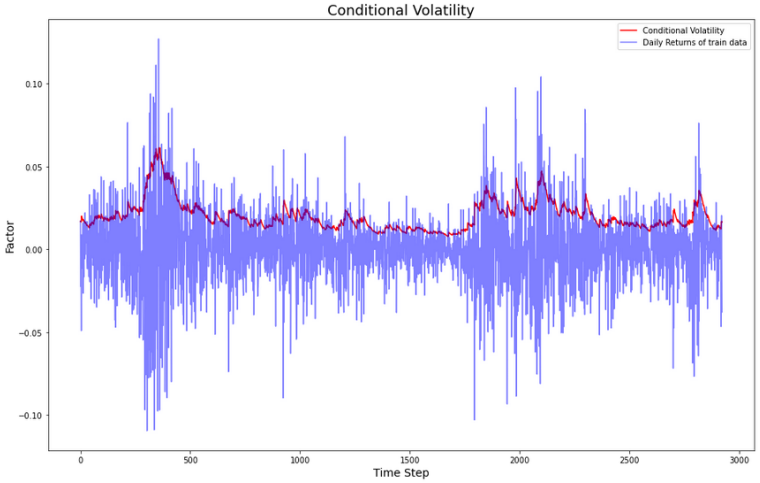


Figure 23: Conditional Volatility

ARMA-GARCH Model Prediction

The prediction of the model can now be calculated using the formula 15.

$$y_t = \mu + a_1 y_{t-1} + \epsilon_t + b_1 \epsilon_{t-1} \tag{15}$$

Figure 24 shows the test data with the predicted data.

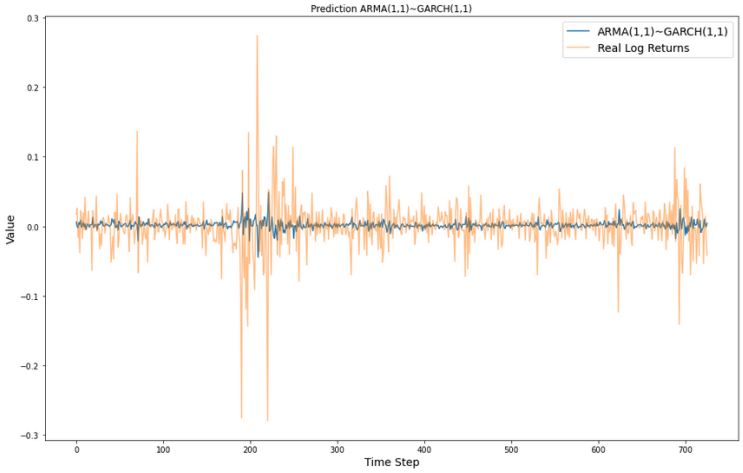


Figure 24: ARMA-GARCH Prediciton

6.2 Classification Report

To get a more detailed overview of how the models performed, one can build a classification report to compute the F1 Score, precision, recall, and support. The meaning of the variables can be interpreted as follows:

$$t_p = TruePositiv$$
$$f_p = FalsePositiv$$
$$f_n = FalseNegativ$$

Precision measures the proportion of all correctly identified samples in a population of samples that are classified as positive labels and are defined as the following:

$$\text{Precision} = \frac{t_p}{(t_p + f_p)} \tag{16}$$

The precision is intuitively the ability of the classifier not to label as positive a sample that is negative. The best value is 1, and the worst value is 0.

Recall (also known as sensitivity) measures the ability of a classifier to identify positive labels correctly and is defined as the following:

$$Recall = \frac{t_p}{(t_p + f_n)} \tag{17}$$

The recall is intuitively the ability of the classifier to find all the positive samples. The best value is 1, and the worst value is 0.

Support is the number of actual occurrences of the class in the specified data-set. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or re-balancing. Support does not change between models but instead diagnoses the evaluation process. In some cases, we will have models that may have low precision or high recall. It is difficult to compare two models with low precision and high recall or vice versa. To make results comparable, we use a metric called the F-Score. The F-score helps to measure Recall and Precision at the same time. It uses Harmonic Mean in place of Arithmetic Mean by punishing the extreme values more. The traditional F-measure or balanced F-score (F1 score) is the harmonic mean of precision and recall:

$$F1Score = \frac{t_p}{(t_p + \frac{1}{2}(f_p + f_n))} \tag{18}$$

6.3 Python Codes

Data_Cleaning_and_Manipulation

June 5, 2022

1 Data Cleaning, Manipulation

1.0.1 At first we have to look at the data and decide if the data is good if they exists NaN values. Then we have to grow up the date with new values. The new values are financial indicators that will help to better understand the data and that will help for many machine learning methods to perform better. At the end we will creat a new csv file as a workfile and save this file on the desktop.

1.0.2 Imports

[1]:
```
# Basic Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math

# Data upload Import
import yfinance as yf
```

[2]:
```
# Download dataset from Yahoo Finance
bco = yf.download('BZ=F', start='2007-07-30', end = '2022-04-26',period='1d') #␣
 ↪Only trading days
```

```
[**********************100%**********************]  1 of 1 completed
```

[3]:
```
# few Info of the dataset
bco.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3653 entries, 2007-07-30 to 2022-04-25
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Open       3653 non-null   float64
 1   High       3653 non-null   float64
 2   Low        3653 non-null   float64
 3   Close      3653 non-null   float64
 4   Adj Close  3653 non-null   float64
 5   Volume     3653 non-null   int64
```

```
dtypes: float64(5), int64(1)
memory usage: 199.8 KB
```

2 Financial Indicator

For a good classification model we have to calculate some financial indicators like returns, signal flag for the lable, Relative Strength Index (RSI), Stochastic Oscillator, Williams %R, Moving Average Convergence Divergnece (MACD), Price Rate Of Change, On Balance Volume. Maybe we also have to smoothe the data. At the end we will make a forecast with the model. But insted of values we will predict a sell or buy binary signal.

2.1 Returns

Returns are the differenz between the value today and the value from yesterday. So we can see if the price increase or decrease. In this cast we want see the behavior of the Close value.

[4]:
```
bco['Returns']= bco['Close'].pct_change()
bco.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3653 entries, 2007-07-30 to 2022-04-25
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Open       3653 non-null   float64
 1   High       3653 non-null   float64
 2   Low        3653 non-null   float64
 3   Close      3653 non-null   float64
 4   Adj Close  3653 non-null   float64
 5   Volume     3653 non-null   int64
 6   Returns    3652 non-null   float64
dtypes: float64(6), int64(1)
memory usage: 228.3 KB
```

[5]:
```
print(bco.head())
```

```
                Open       High        Low      Close  Adj Close  Volume  \
Date
2007-07-30  75.849998  76.529999  75.440002  75.739998  75.739998    2575
2007-07-31  75.699997  77.169998  75.669998  77.050003  77.050003    3513
2007-08-01  77.000000  77.059998  74.860001  75.349998  75.349998    3930
2007-08-02  75.220001  76.209999  74.269997  75.760002  75.760002    6180
2007-08-03  75.389999  76.000000  74.529999  74.750000  74.750000    4387

            Returns
Date
2007-07-30      NaN
```

```
2007-07-31  0.017296
2007-08-01 -0.022064
2007-08-02  0.005441
2007-08-03 -0.013332
```

2.2 Log Returns

Log Returns are the same as the Returns but in a Loghatitmic Scale

[6]:
```
bco['Log_Returns'] = np.log(1 + bco['Close'].pct_change())
```

2.3 Signal

For the trading signal we use the returns but for a Random Forest we have to creat a binary signal for a clean classification. We look at the return values and indicate all values smaller then 0 with 0 and all values bigger then 0 with 1.

[7]:
```
bco['Signal'] = (bco['Returns'] > 0).astype(int)
```

[8]:
```
bco.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3653 entries, 2007-07-30 to 2022-04-25
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Open         3653 non-null   float64
 1   High         3653 non-null   float64
 2   Low          3653 non-null   float64
 3   Close        3653 non-null   float64
 4   Adj Close    3653 non-null   float64
 5   Volume       3653 non-null   int64  
 6   Returns      3652 non-null   float64
 7   Log_Returns  3652 non-null   float64
 8   Signal       3653 non-null   int64  
dtypes: float64(7), int64(2)
memory usage: 285.4 KB
```

2.4 Predict Signal

The predict signal is the same as the signal before. However, it has been shifted back by one.

[9]:
```
bco['Pred_Signal'] = bco['Signal'].shift(-1)
bco.tail()
```

[9]:

Date	Open	High	Low	Close	Adj Close	\
2022-04-19	112.650002	114.050003	106.769997	107.250000	107.250000	
2022-04-20	107.720001	108.980003	104.669998	106.800003	106.800003	

```
2022-04-21  107.010002  109.790001  106.779999  108.330002  108.330002
2022-04-22  108.639999  108.730003  105.529999  106.650002  106.650002
2022-04-25  105.739998  105.949997   99.470001  102.320000  102.320000

            Volume   Returns  Log_Returns  Signal  Pred_Signal
Date
2022-04-19   18972 -0.052227    -0.053640       0          0.0
2022-04-20   15985 -0.004196    -0.004205       0          1.0
2022-04-21   13203  0.014326     0.014224       1          0.0
2022-04-22   11040 -0.015508    -0.015630       0          0.0
2022-04-25   15464 -0.040600    -0.041447       0          NaN
```

2.5 Relative Strength Index (RSI)

Definition From Paper:

RSI is a popular momentum indicator that determines whether the stock is overbought or oversold. A stock is said to be overbought when the demand unjustifiably pushes the price upwards. This condition is generally interpreted as a sign that the stock is overvalued, and the price is likely to go down. A stock is said to be oversold when the price goes down sharply to a level below its true value. This is a result caused due to panic selling. RSI ranges from 0 to 100, and generally, when RSI is above 70, it may indicate that the stock is overbought and when RSI is below 30, it may indicate the stock is oversold.

Formula

RSI = 100 - 100/(1+RS)

[10]:
```
# Calculate the 14 day RSI
n = 14

# First make a copy of the data frame twice
# For up days, if the change is less than 0 set to 0.
up_df, down_df = bco[['Returns']].copy(), bco[['Returns']].copy()

# For down days, if the change is greater than 0 set to 0.
up_df.loc['Returns'] = up_df.loc[(up_df['Returns'] < 0), 'Returns'] = 0

# We need change in price to be absolute.
down_df.loc['Returns'] = down_df.loc[(down_df['Returns'] > 0), 'Returns'] = 0
down_df['Returns'] = down_df['Returns'].abs()

# Calculate the EWMA (Exponential Weighted Moving Average), meaning older␣
 ↪values are
# given less weight compared to newer values.
ewma_up = up_df['Returns'].transform(lambda x: x.ewm(span = n).mean())
```

```
ewma_down = down_df['Returns'].transform(lambda x: x.ewm(span = n).mean())

# Calculate the Relative Strength
relative_strength = ewma_up / ewma_down # Calculate the Relative Strength Index
relative_strength_index = 100.0 - (100.0 / (1.0 + relative_strength))

# Add the info to the data frame.
bco['RSI'] = relative_strength_index
```

2.6 Stochastic Oscillator

Definition From Paper:

Stochastic Oscillator follows the speed or the momentum of the price. As a rule, momentum changes before the price changes. It measures the level of the closing price relative to the low-high range over a period of time.

Formula:

C = Current Closing Price

L14 = Lowest low over the past 14 days

H14 = Highest high over the past 14 days

K = 100 * (C-L14)/(H14-L14)

[11]:
```
# Calculate the Stochastic Oscillator
n = 14

# Make a copy of the high and low column.
low_14, high_14 = bco[['Low']].copy(), bco[['High']].copy()

# Group by symbol, then apply the rolling function and grab the Min and Max.
low_14 = low_14['Low'].transform(lambda x: x.rolling(window = n).min())
high_14 = high_14['High'].transform(lambda x: x.rolling(window = n).max())

# Calculate the Stochastic Oscillator.
k_percent = 100 * ((bco['Close'] - low_14) / (high_14 - low_14))

# Add the info to the data frame.
bco['K_percent'] = k_percent
```

2.7 Moving Average Convergence Divergnece (MACD)

Definition From Paper:

EMA stands for Exponential Moving Average. When the MACD goes below the SingalLine, it indicates a sell signal. When it goes above the Signal Line, it indicates a buy signal.

Formula:

MACD = Moving Average Convergence Divergence

C = Closing Price

EMAn = n day Exponential Moving Average

MACD = EMA12(C) - EMA26(C)

SignalLine = EMA9(MACD)

[12]:
```
# Calculate the MACD
ema_26 = bco['Close'].transform(lambda x: x.ewm(span = 26).mean())
ema_12 = bco['Close'].transform(lambda x: x.ewm(span = 12).mean())
macd = ema_12 - ema_26

# Calculate the EMA
ema_9_macd = macd.ewm(span = 9).mean()

# Store the data in the data frame.
bco['MACD'] = macd
bco['MACD_EMA'] = ema_9_macd
```

2.8 Price Rate Of Change

Definition From Paper:

It measures the most recent change in price with respect to the price in n days ago.

PROCt = Return at time t

Ct = Closing Price at time t

PROCt = Ct-Ct-n/Ct-n

[13]:
```
# Calculate the Price Rate of Change
n=9

# Calculate the Rate of Change in the Price, and store it in the Data Frame.
bco['ROC'] = bco['Close'].transform(lambda x: x.pct_change(periods = n))
```

2.9 Removing NaN Values

Classification algorithms can't accept Nan values, so we will need to remove them before feeding the data in. The code below prints the number of rows before dropping the NaN values, use the dropna method to remove any rows NaN values and then displays the number of rows after dropping the NaN values.

[14]:
```
# We need to remove all rows that have an NaN value.
print('Before NaN Drop we have {} rows and {} columns'.format(bco.shape[0], bco.
 ↪shape[1]))

# Any row that has a `NaN` value will be dropped.
bco = bco.dropna()

# Display how much we have left now.
print('After NaN Drop we have {} rows and {} columns'.format(bco.shape[0], bco.
 ↪shape[1]))
```

```
Before NaN Drop we have 3653 rows and 15 columns
After NaN Drop we have 3639 rows and 15 columns
```

2.10 Transform Predict Signal in Integer

[15]:
```
bco['Pred_Signal'] = bco['Pred_Signal'].astype(int)
```

```
<ipython-input-15-4a7a810e7cc2>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  bco['Pred_Signal'] = bco['Pred_Signal'].astype(int)
```

[16]:
```
bco.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3639 entries, 2007-08-16 to 2022-04-22
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Open         3639 non-null   float64
 1   High         3639 non-null   float64
 2   Low          3639 non-null   float64
 3   Close        3639 non-null   float64
 4   Adj Close    3639 non-null   float64
 5   Volume       3639 non-null   int64
 6   Returns      3639 non-null   float64
 7   Log_Returns  3639 non-null   float64
 8   Signal       3639 non-null   int64
```

```
 9   Pred_Signal  3639 non-null   int64
 10  RSI          3639 non-null   float64
 11  K_percent    3639 non-null   float64
 12  MACD         3639 non-null   float64
 13  MACD_EMA     3639 non-null   float64
 14  ROC          3639 non-null   float64
dtypes: float64(12), int64(3)
memory usage: 454.9 KB
```

2.11 Drop all columns we do not need!

[17]:
```
bco = bco.drop(['Open','High','Low','Adj Close','Volume','Signal','MACD_EMA'],␣
↪axis = 1)
```

2.12 Create a new csv file and save on Desk

[18]:
```
# df.to_csv("<path to desktop and filename>")
# If you just use file name then it will save CSV file in working directory.
# Example path : C:/Users/<>/Desktop/<file name>.csv
```

[ ]:
```

```

Deskriptiv_Statistic

June 4, 2022

1 Deskriptiv Statistic

Imports

[1]:
```
# Basic Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from itertools import repeat
```

[2]:
```
bco = pd.read_csv('/Users/romaindeleze/Desktop/BSc/Romain_WI_TZ/8.Semester/BA/
↪Daten/Brent_Crude_Oil')
```

[3]:
```
# Data Frame with the Classification relevant Columns
df = pd.DataFrame(data=(bco['RSI'],bco['K_percent'],bco['MACD'],
                                bco['ROC'], bco['Close'], bco['Log_Returns']))
df = df.T
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3641 entries, 0 to 3640
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   RSI          3641 non-null   float64
 1   K_percent    3641 non-null   float64
 2   MACD         3641 non-null   float64
 3   ROC          3641 non-null   float64
 4   Close        3641 non-null   float64
 5   Log_Returns  3641 non-null   float64
dtypes: float64(6)
memory usage: 170.8 KB
```

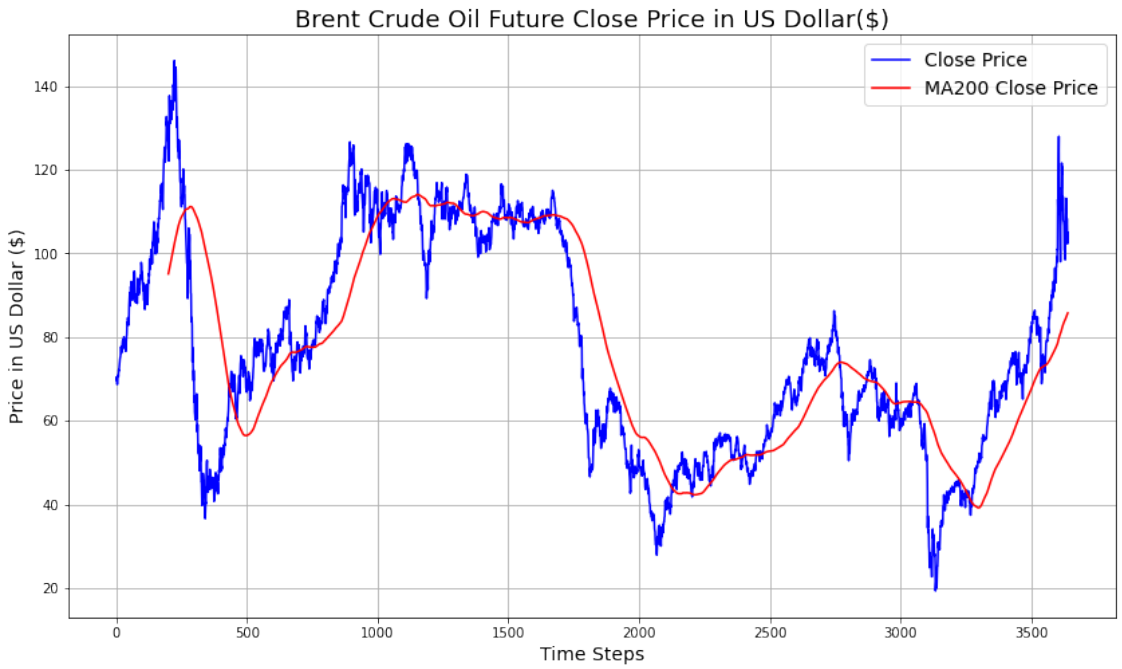
2 Time Serie

2.1 Descriptiv Statistik

2.2 Close Price Plot

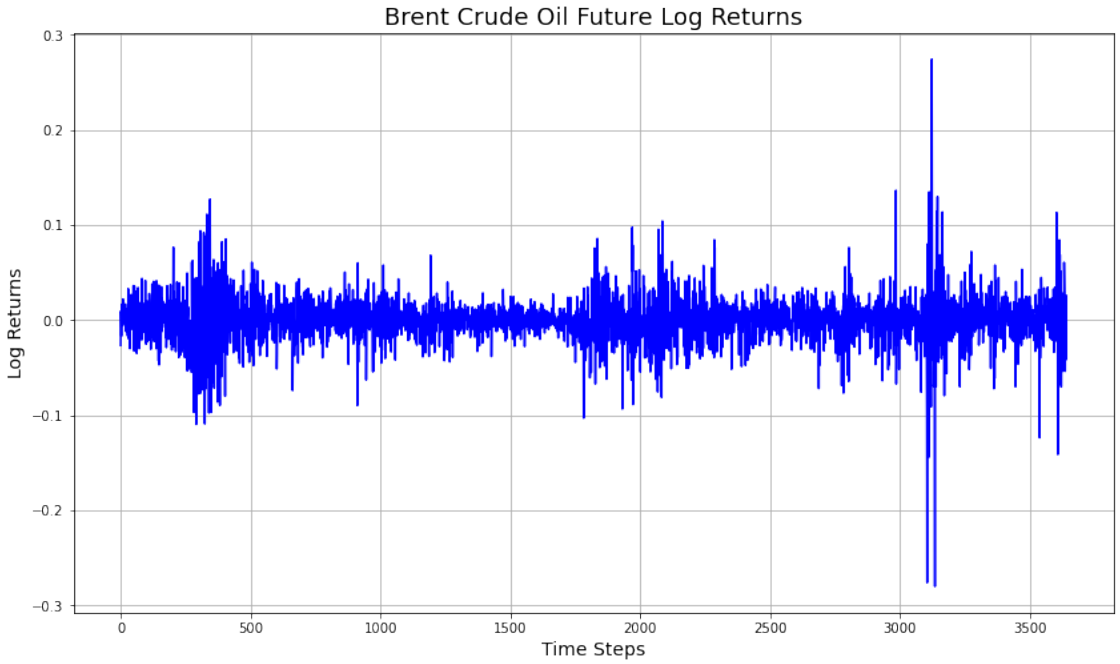
[4]:
```
plt.figure(figsize=(14,8))
plt.title('Brent Crude Oil Future Close Price in US Dollar($)', fontsize = 18)
plt.ylabel('Price in US Dollar ($)', fontsize = 14)
plt.xlabel('Time Steps', fontsize = 14)
plt.grid(True)
plt.plot(df.Close, label = 'Close Price', color = 'blue')
plt.plot(df.Close.rolling(200).mean(), label = 'MA200 Close Price', color =␣
↪'red')
plt.legend(loc='best', fontsize = 14)
```

[4]:
```
<matplotlib.legend.Legend at 0x7fd7e4abcfa0>
```



[5]:
```
plt.figure(figsize=(14,8))
plt.title('Brent Crude Oil Future Log Returns', fontsize = 18)
plt.ylabel('Log Returns', fontsize = 14)
plt.xlabel('Time Steps', fontsize = 14)
plt.grid(True)
plt.plot(df.Log_Returns, color = 'blue')
```

[5]:
```
[<matplotlib.lines.Line2D at 0x7fd7e4d0dfa0>]
```

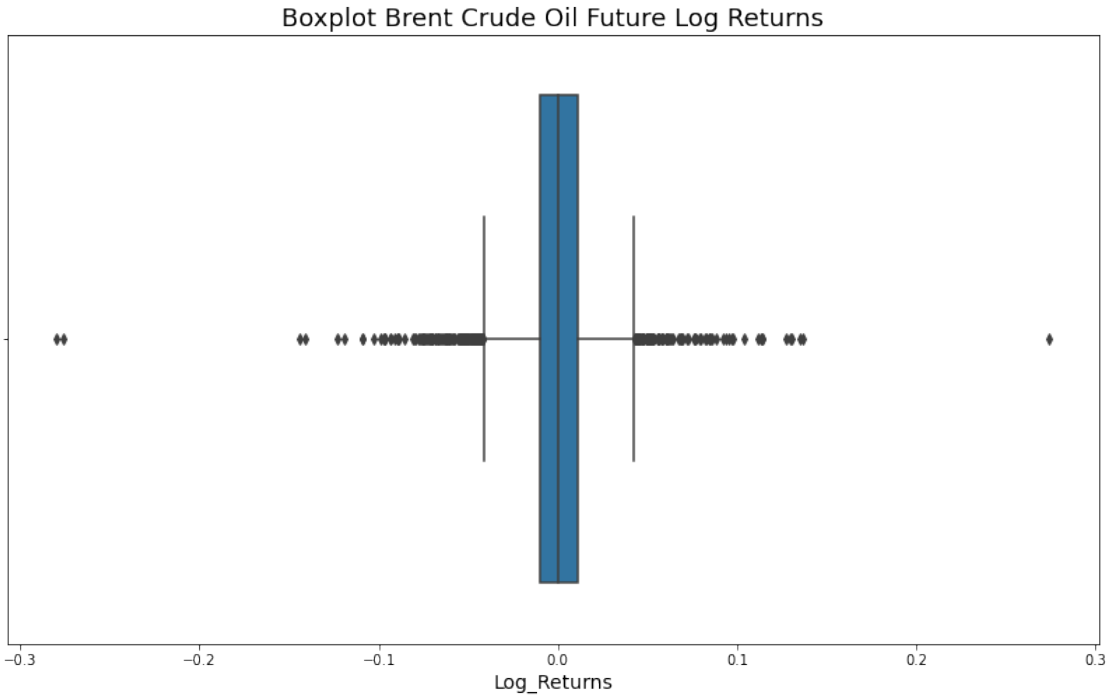


2.2.1 Price and Log Returns Box Plots

[6]:
```
plt.figure(figsize=(14,8))
plt.title('Boxplot Brent Crude Oil Future Log Returns', fontsize = 18)
plt.xlabel('Log Returns', fontsize = 14)
sns.boxplot(df['Log_Returns'])
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

[6]:
```
<AxesSubplot:title={'center':'Boxplot Brent Crude Oil Future Log Returns'},
xlabel='Log_Returns'>
```

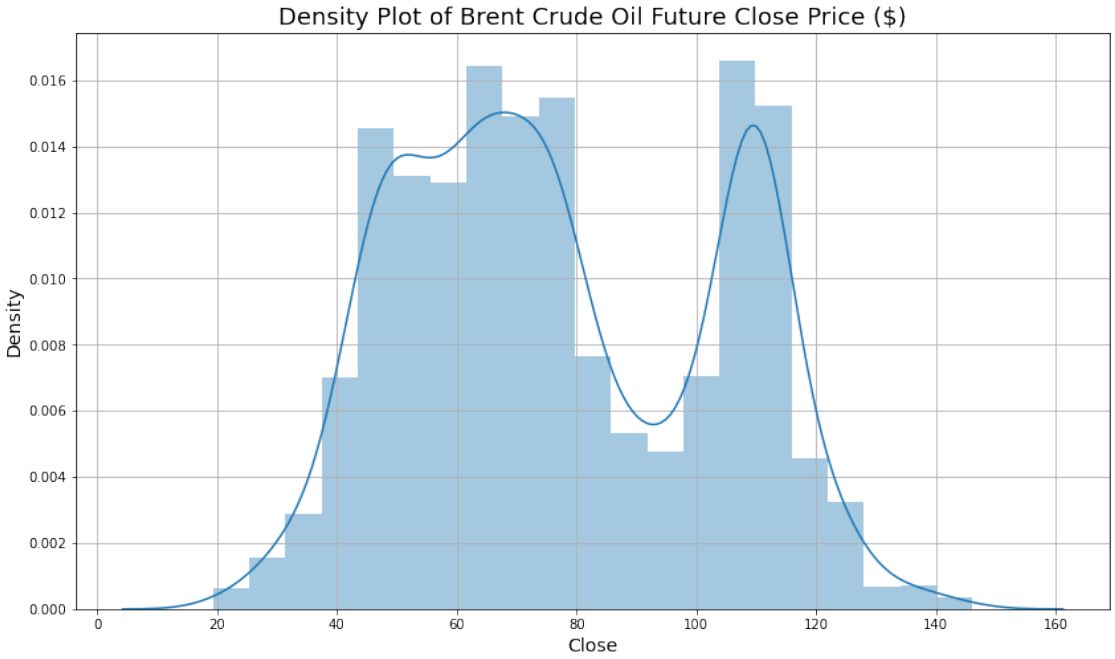


2.2.2 Price and Log Returns Density Plots

[7]:
```
plt.figure(figsize=(14,8))
plt.title('Density Plot of Brent Crude Oil Future Close Price ($)', fontsize =␣
↪18)
plt.ylabel('Density', fontsize = 14)
plt.xlabel('Close Price ($)', fontsize = 14)
plt.grid(True)
sns.distplot(df['Close'])
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

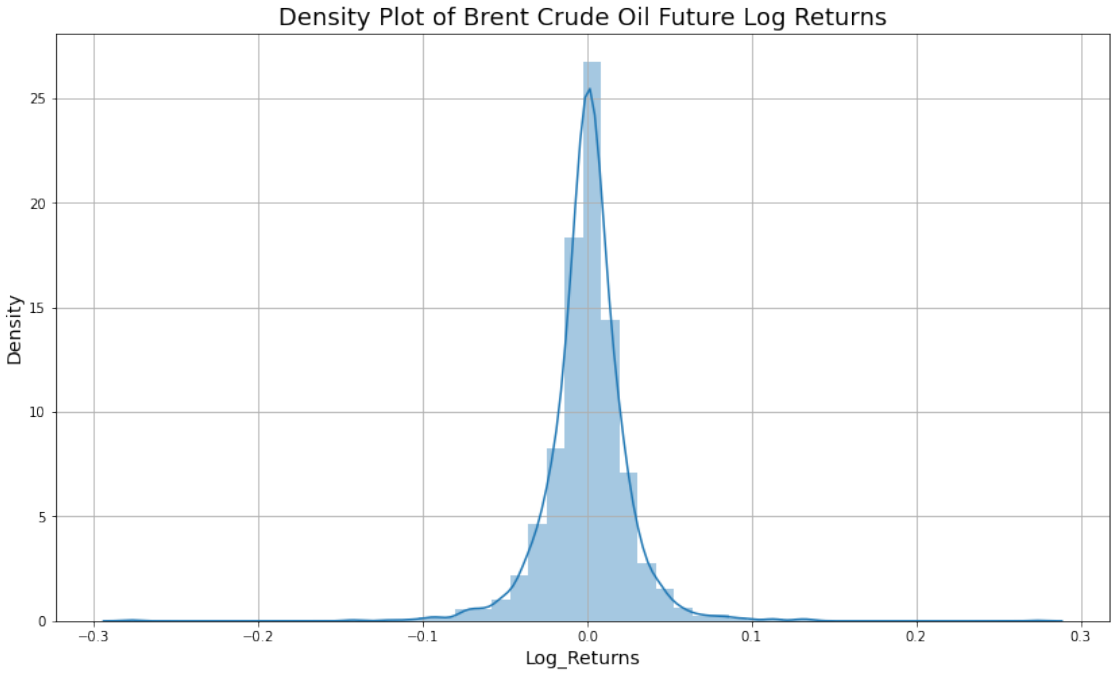
[7]:
```
<AxesSubplot:title={'center':'Density Plot of Brent Crude Oil Future Close Price
($)'}, xlabel='Close', ylabel='Density'>
```



[8]:
```
plt.figure(figsize=(14,8))
plt.title('Density Plot of Brent Crude Oil Future Log Returns', fontsize = 18)
plt.ylabel('Density', fontsize = 14)
plt.xlabel('Log Returns', fontsize = 14)
plt.grid(True)
sns.distplot(df['Log_Returns'])
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[8]:
```
<AxesSubplot:title={'center':'Density Plot of Brent Crude Oil Future Log
Returns'}, xlabel='Log_Returns', ylabel='Density'>
```



2.2.3 Price and Log Returns Position measure

[9]:
```
df_label = df.drop(['RSI','MACD', 'K_percent','ROC'], axis = 1)
print(df_label.describe(percentiles=[.05,.25,.5,.75,.95]))
```

```
              Close  Log_Returns
count   3641.000000  3641.000000
mean      77.408053     0.000105
std       25.918687     0.024656
min       19.330000    -0.279761
5%        41.290001    -0.037248
25%       55.980000    -0.009985
50%       73.360001     0.000435
75%      103.809998     0.011025
95%      117.099998     0.034635
max      146.080002     0.274191
```

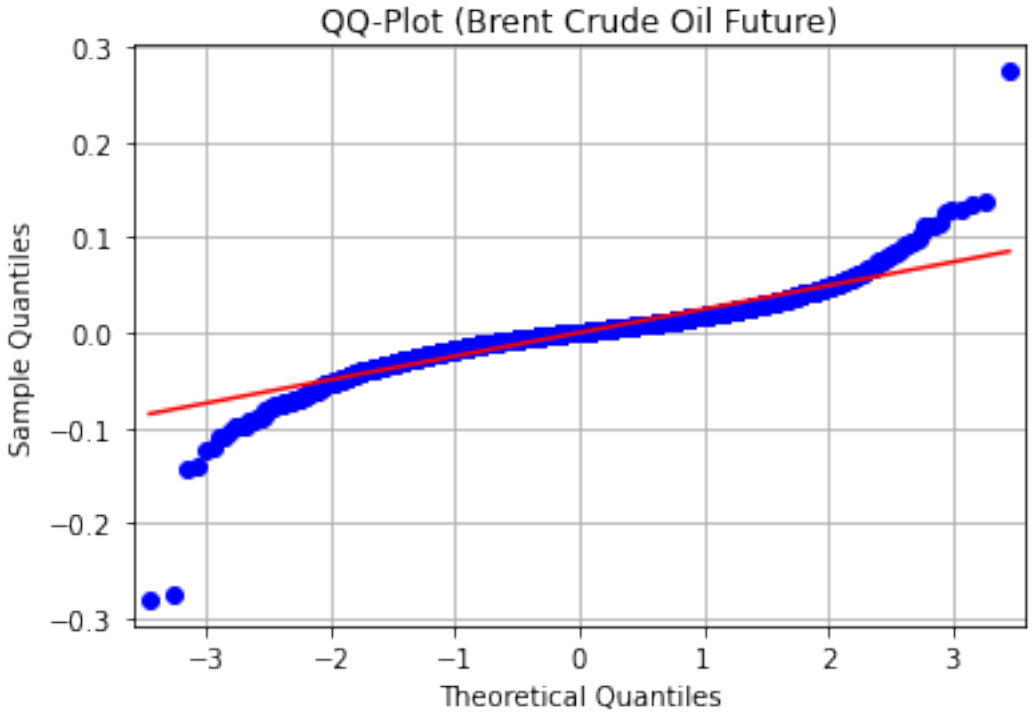
2.2.4 Quantile-Quantile Plot

[10]:
```
import statsmodels.api as sm
import scipy.stats as stats
```

[11]:
```
plt.figure(figsize=(18,10))
sm.qqplot(df.Log_Returns, line='s')
plt.title('QQ-Plot (Brent Crude Oil Future)')
```

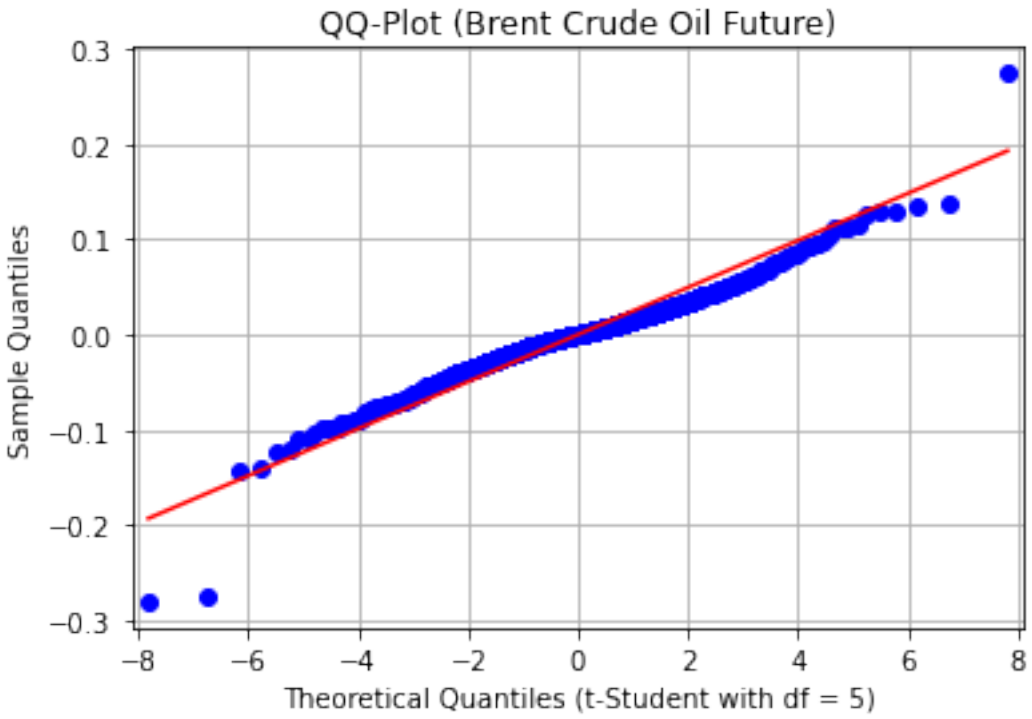
```
plt.grid(True)
plt.show()
```

```
<Figure size 1296x720 with 0 Axes>
```



[12]:
```
# QQ-Plot with T-Student Distribution
plt.figure(figsize=(18,10))
sm.qqplot(df.Log_Returns, dist = stats.t, distargs=(5,), line='s')# Seems like␣
 ↪a t-Student Distribution with df = 5
plt.title('QQ-Plot (Brent Crude Oil Future)')
plt.xlabel('Theoretical Quantiles (t-Student with df = 5)')
plt.grid(True)
plt.show()
```

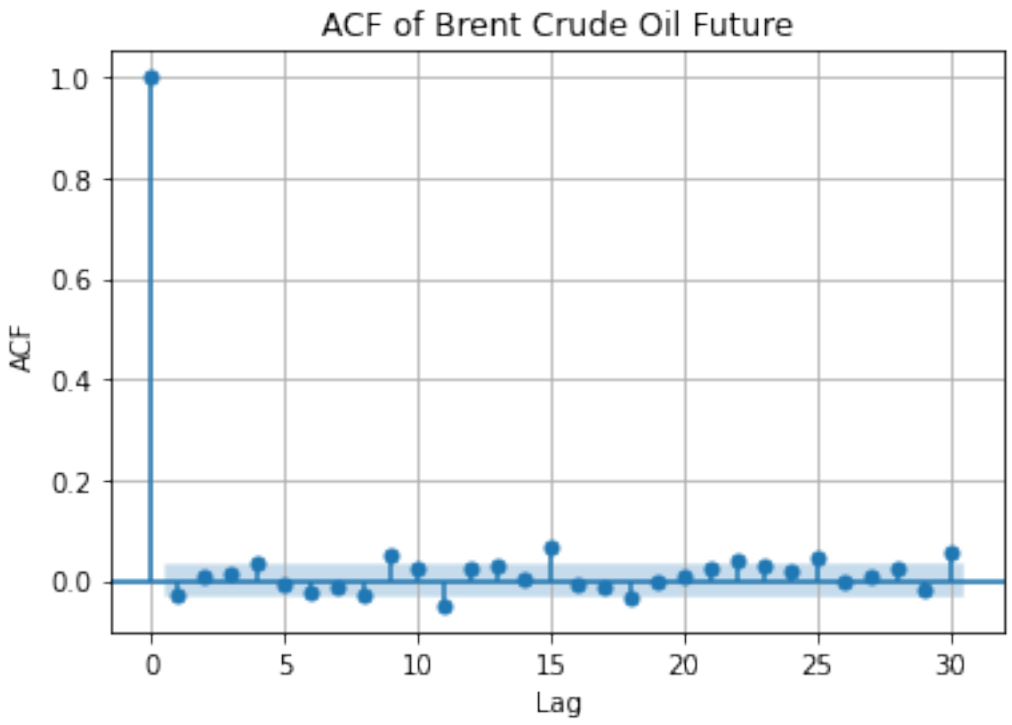
```
<Figure size 1296x720 with 0 Axes>
```



2.2.5 ACF Plot

[13]:
```
from statsmodels.graphics.tsaplots import plot_acf
```

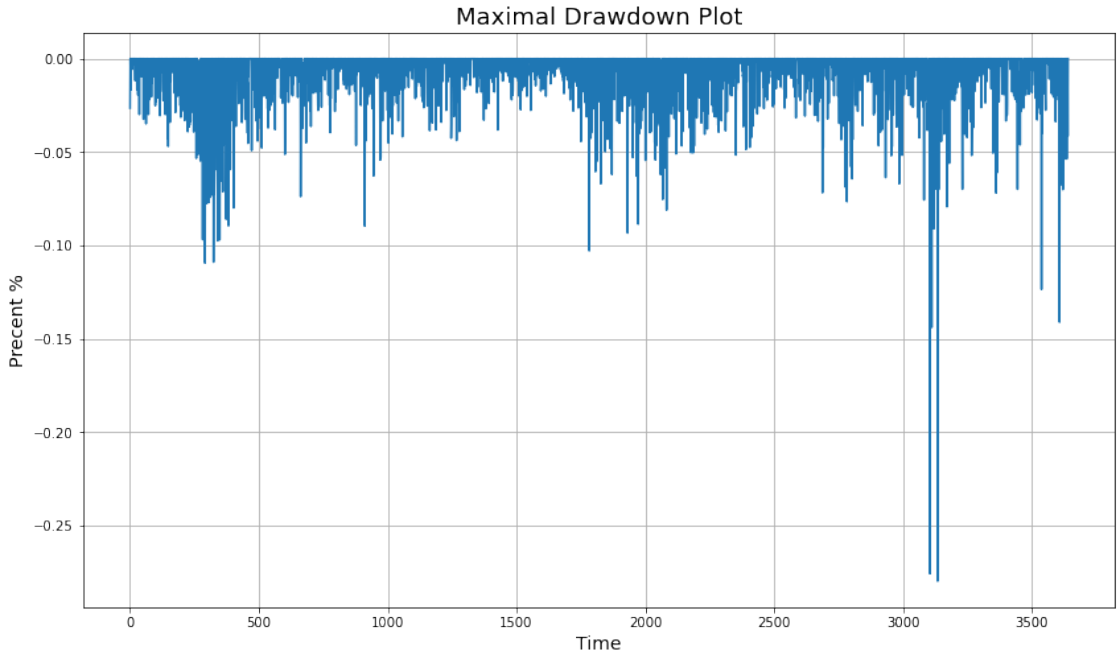
[14]:
```
#plt.figure(figsize=(10,8))
plot_acf(df.Log_Returns, lags=30)
plt.title('ACF of Brent Crude Oil Future')
plt.ylabel('ACF')
plt.xlabel('Lag')
plt.grid(True)
```



2.2.6 Maximal Drawdown Plot

[15]:
```
# Own MDD Function for GAN's Output
def own_MDD_GANs_plt(returns):

    # INPUT are RETURS values !!!!
    ##################################################
    # Cumulative Returns
    cum_returns = [returns[0]]
    for i in range(0, len(returns)-1):
        i = (1 + returns[i]) * (1 + returns[i+1]) - 1
        cum_returns.append(i)
    ##################################################
    # Drawdown
    drawdown = []
    for i in range(0,len(returns)):
        if returns[i] < 0:
            drawdown.append(returns[i])
        else:
            drawdown.append(0)

    return [drawdown]
    ##################################################
```

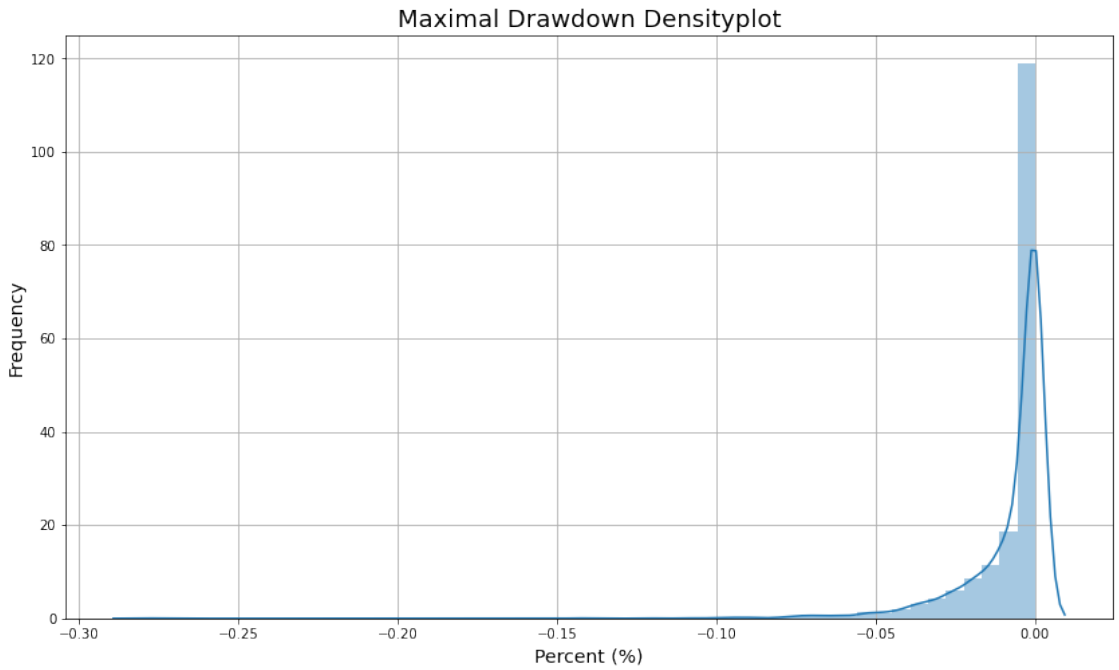
```
MDD = own_MDD_GANs_plt(df.Log_Returns)

plt.figure(figsize=(14,8))
plt.title('Maximal Drawdown Plot', fontsize = 18)
plt.ylabel('Precent %', fontsize = 14)
plt.xlabel('Time', fontsize = 14)
plt.grid(True)
plt.plot(MDD[0])
plt.show()
```



[16]:
```
# MDD Densityplot
plt.figure(figsize=(14,8))
plt.title('Maximal Drawdown Densityplot', fontsize = 18)
plt.ylabel('Frequency', fontsize = 14)
plt.xlabel('Percent (%)', fontsize = 14)
plt.grid(True)
sns.distplot(MDD[0])
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[16]: <AxesSubplot:title={'center':'Maximal Drawdown Densityplot'}, xlabel='Percent (%)', ylabel='Frequency'>



[17]:
```
mdd = {'MDD':MDD[0]}
df_MDD = pd.DataFrame(data=mdd)

print(df_MDD.describe(percentiles=[.05,.25,.5,.75,.95]))
```

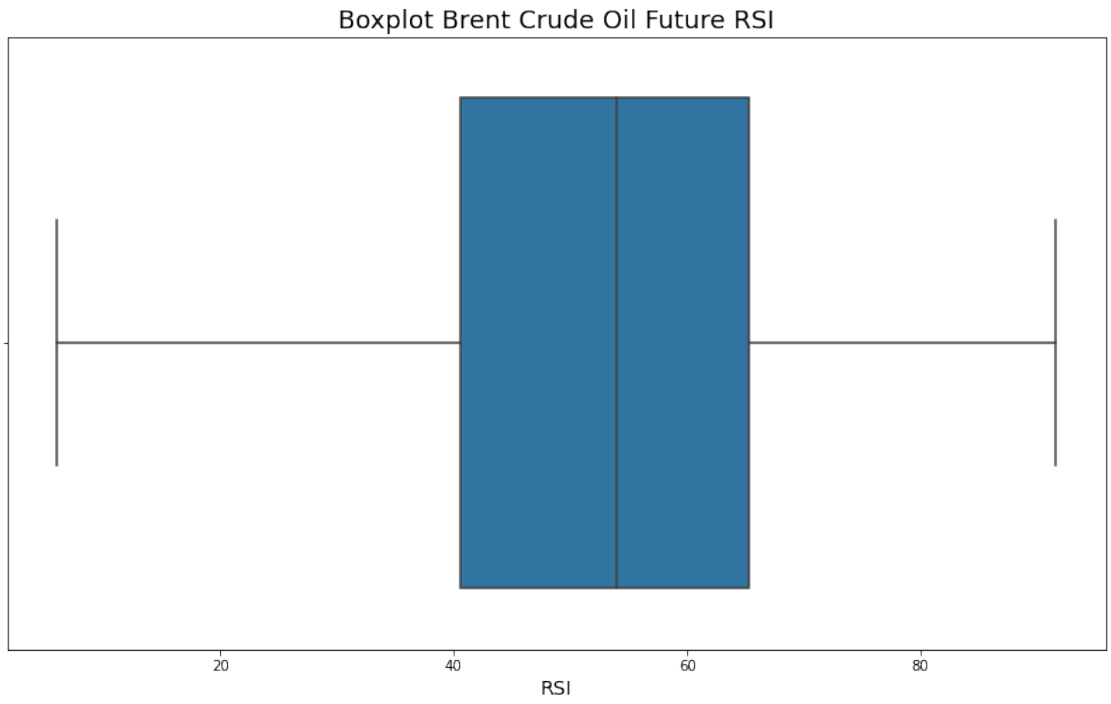
```
                MDD
count  3641.000000
mean     -0.007991
std       0.016021
min      -0.279761
5%       -0.037248
25%      -0.009985
50%       0.000000
75%       0.000000
95%       0.000000
max       0.000000
```

2.2.7 Features Box Plots

[18]:
```
plt.figure(figsize=(14,8))
plt.title('Boxplot Brent Crude Oil Future RSI', fontsize = 18)
plt.xlabel('RSI', fontsize = 14)
sns.boxplot(df['RSI'])
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

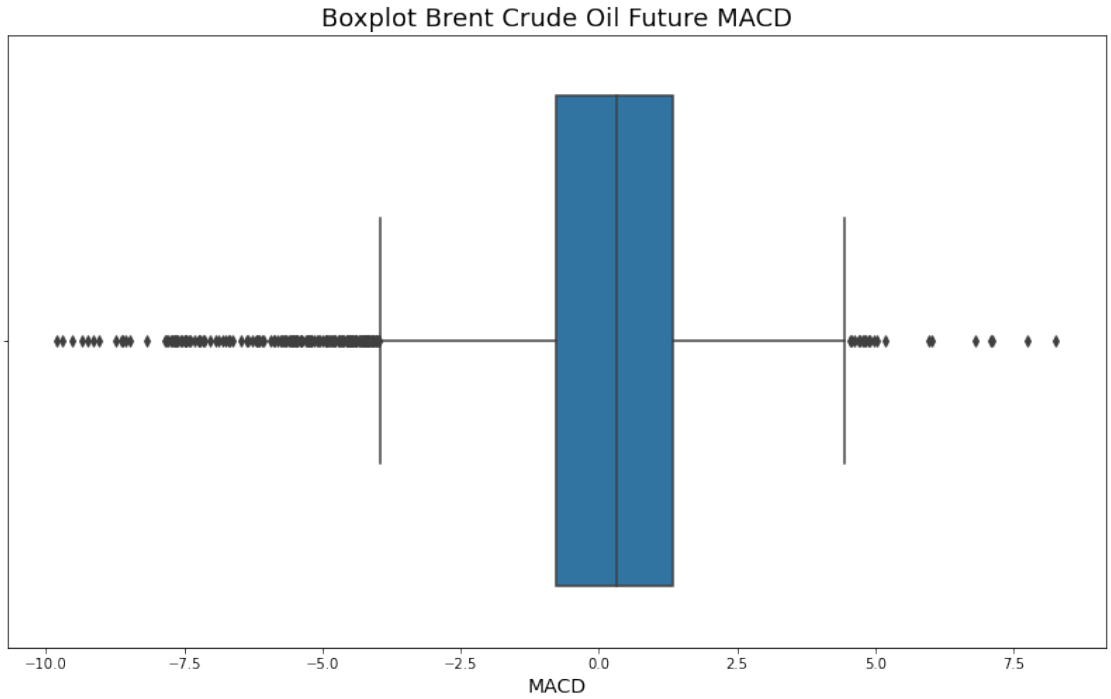
[18]: <AxesSubplot:title={'center':'Boxplot Brent Crude Oil Future RSI'}, xlabel='RSI'>



[19]:
```
plt.figure(figsize=(14,8))
plt.title('Boxplot Brent Crude Oil Future MACD', fontsize = 18)
plt.xlabel('MACD', fontsize = 14)
sns.boxplot(df['MACD'])
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

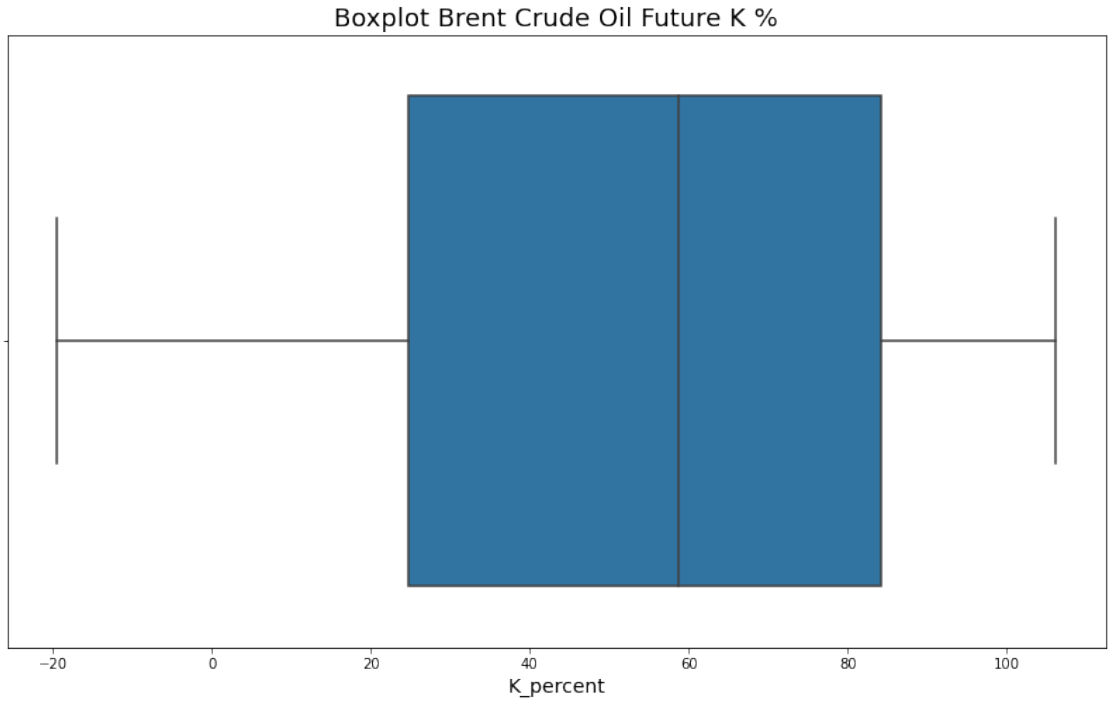
[19]: <AxesSubplot:title={'center':'Boxplot Brent Crude Oil Future MACD'}, xlabel='MACD'>



[20]:
```
plt.figure(figsize=(14,8))
plt.title('Boxplot Brent Crude Oil Future K %', fontsize = 18)
plt.xlabel('K %', fontsize = 14)
sns.boxplot(df['K_percent'])
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

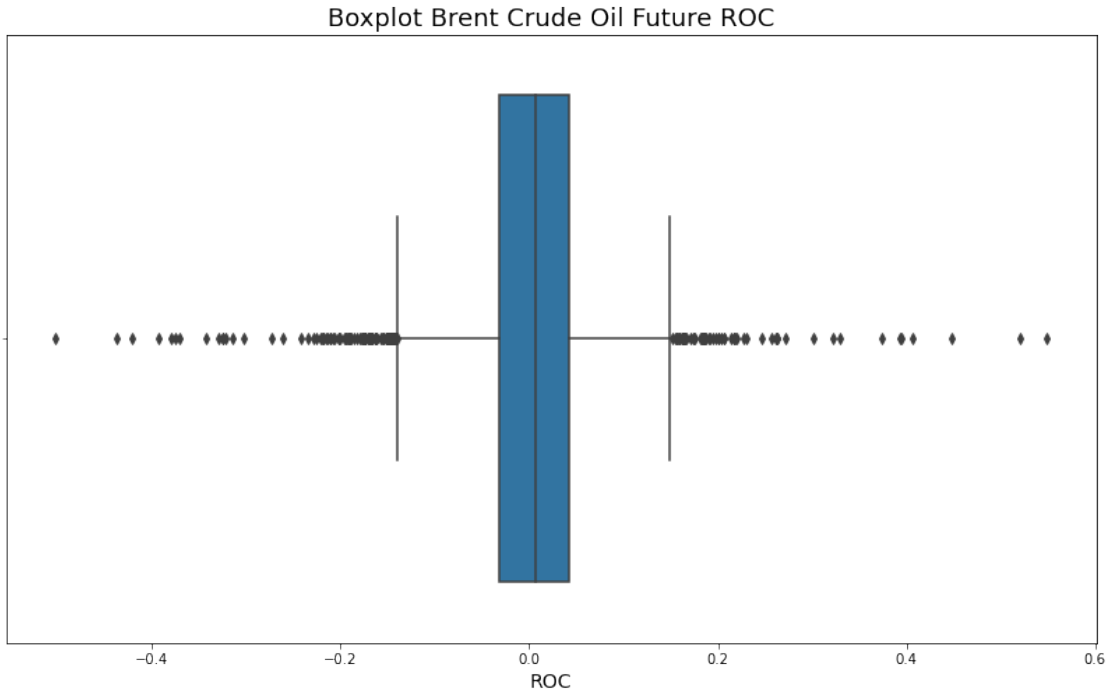
[20]:
```
<AxesSubplot:title={'center':'Boxplot Brent Crude Oil Future K %'},
xlabel='K_percent'>
```



[21]:
```
plt.figure(figsize=(14,8))
plt.title('Boxplot Brent Crude Oil Future ROC', fontsize = 18)
plt.xlabel('ROC', fontsize = 14)
sns.boxplot(df['ROC'])
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

[21]:
```
<AxesSubplot:title={'center':'Boxplot Brent Crude Oil Future ROC'},
xlabel='ROC'>
```

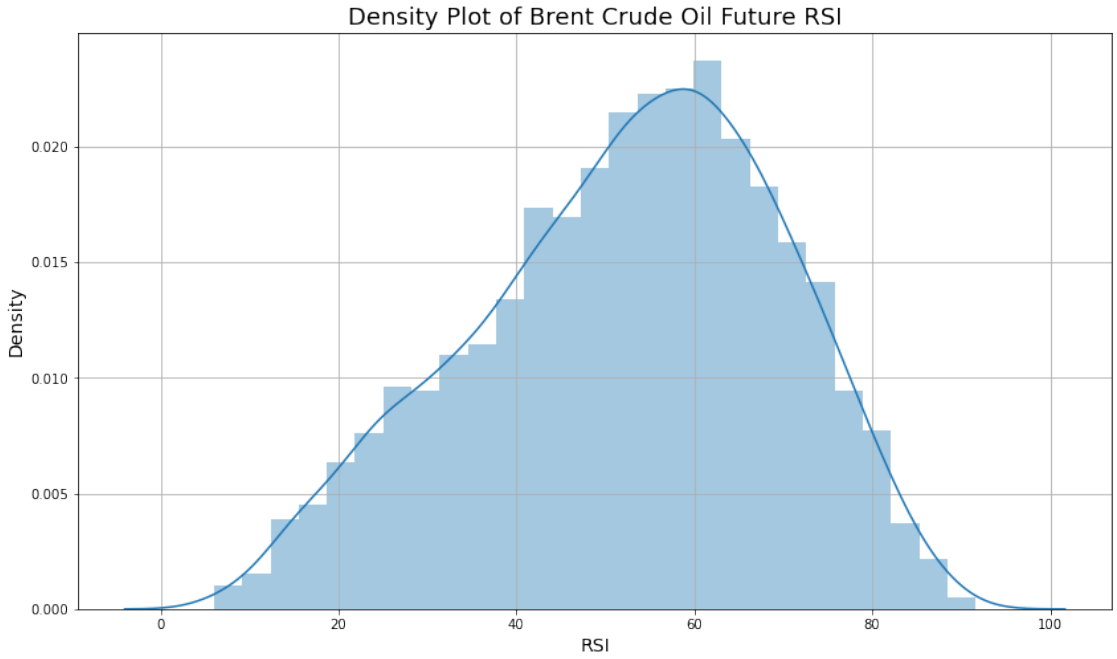


2.2.8 Features Density Plots

[22]:
```
plt.figure(figsize=(14,8))
plt.title('Density Plot of Brent Crude Oil Future RSI', fontsize = 18)
plt.ylabel('Density', fontsize = 14)
plt.xlabel('RSI', fontsize = 14)
plt.grid(True)
sns.distplot(df['RSI'])
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

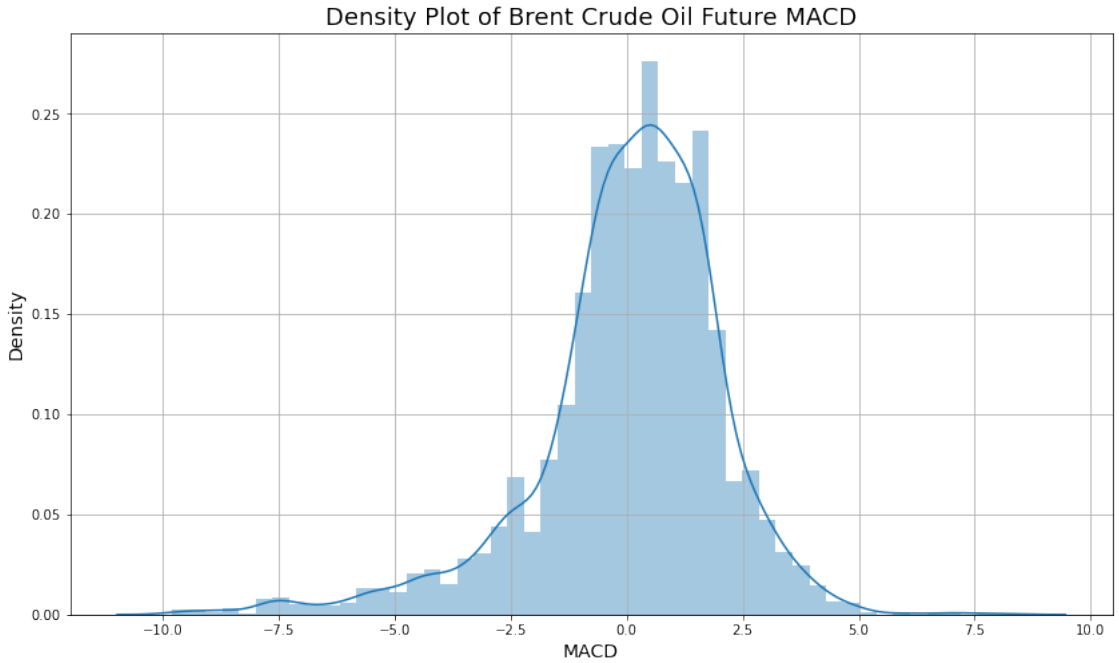
[22]:
```
<AxesSubplot:title={'center':'Density Plot of Brent Crude Oil Future RSI'},
xlabel='RSI', ylabel='Density'>
```



[23]:
```
plt.figure(figsize=(14,8))
plt.title('Density Plot of Brent Crude Oil Future MACD', fontsize = 18)
plt.ylabel('Density', fontsize = 14)
plt.xlabel('MACD', fontsize = 14)
plt.grid(True)
sns.distplot(df['MACD'])
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

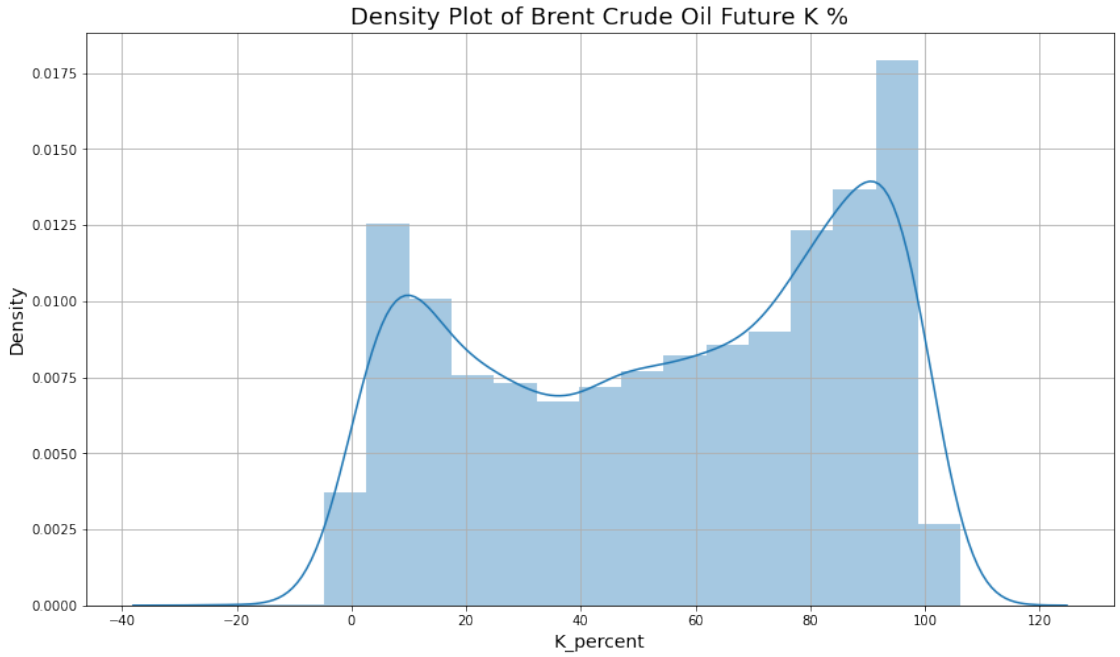
[23]:
```
<AxesSubplot:title={'center':'Density Plot of Brent Crude Oil Future MACD'},
xlabel='MACD', ylabel='Density'>
```



[24]:
```
plt.figure(figsize=(14,8))
plt.title('Density Plot of Brent Crude Oil Future K %', fontsize = 18)
plt.ylabel('Density', fontsize = 14)
plt.xlabel('K %', fontsize = 14)
plt.grid(True)
sns.distplot(df['K_percent'])
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

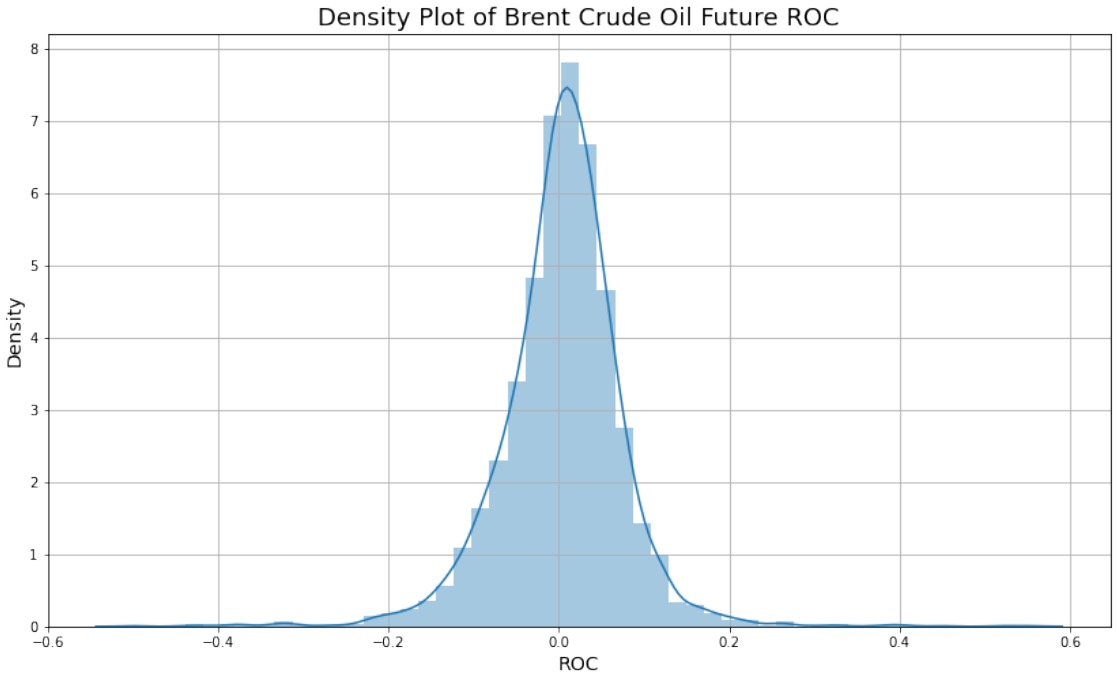
[24]:
```
<AxesSubplot:title={'center':'Density Plot of Brent Crude Oil Future K %'},
xlabel='K_percent', ylabel='Density'>
```



[25]:
```
plt.figure(figsize=(14,8))
plt.title('Density Plot of Brent Crude Oil Future ROC', fontsize = 18)
plt.ylabel('Density', fontsize = 14)
plt.xlabel('ROC', fontsize = 14)
plt.grid(True)
sns.distplot(df['ROC'])
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[25]:
```
<AxesSubplot:title={'center':'Density Plot of Brent Crude Oil Future ROC'},
xlabel='ROC', ylabel='Density'>
```



[26]:
```
df_feature = df.drop(['Close','Log_Returns'], axis = 1)

print(df_feature.describe(percentiles=[.05,.25,.5,.75,.95]))
```

```
                RSI    K_percent         MACD          ROC
count   3641.000000  3641.000000  3641.000000  3641.000000
mean      52.275425    54.728841     0.065906     0.003645
std       17.264053    31.862231     2.022799     0.071429
min        5.991848   -19.512141    -9.802775    -0.502300
5%        21.264264     4.352861    -3.694584    -0.109340
25%       40.587059    24.686536    -0.771373    -0.031050
50%       53.986244    58.669804     0.313472     0.006683
75%       65.289606    84.218291     1.352979     0.041847
95%       78.270572    97.386590     2.835581     0.104124
max       91.607333   106.200867     8.275563     0.549275
```

3 Performance Plot of all Models

[27]:
```
# Imports
RF_Perf_data = pd.read_csv('/Users/romaindeleze/Desktop/BSc/Romain_WI_TZ/8.
 ↪Semester/BA/Daten/RF_Perf_data')
LSTM_Perf_data = pd.read_csv('/Users/romaindeleze/Desktop/BSc/Romain_WI_TZ/8.
 ↪Semester/BA/Daten/LSTM_Perf_data')
```

```
knn_Perf_data = pd.read_csv('//Users/romaindeleze/Desktop/BSc/Romain_WI_TZ/8.
 ↪Semester/BA/Daten/knn_Perf_data')
SVM_Perf_data = pd.read_csv('/Users/romaindeleze/Desktop/BSc/Romain_WI_TZ/8.
 ↪Semester/BA/Daten/SVM_Perf_data')
CS_Perf_data = pd.read_csv('/Users/romaindeleze/Desktop/BSc/Romain_WI_TZ/8.
 ↪Semester/BA/Daten/CS_Perf_data')
AG_Perf_data = pd.read_csv('/Users/romaindeleze/Desktop/BSc/Romain_WI_TZ/8.
 ↪Semester/BA/Daten/arma_garch_Perf_data')
```

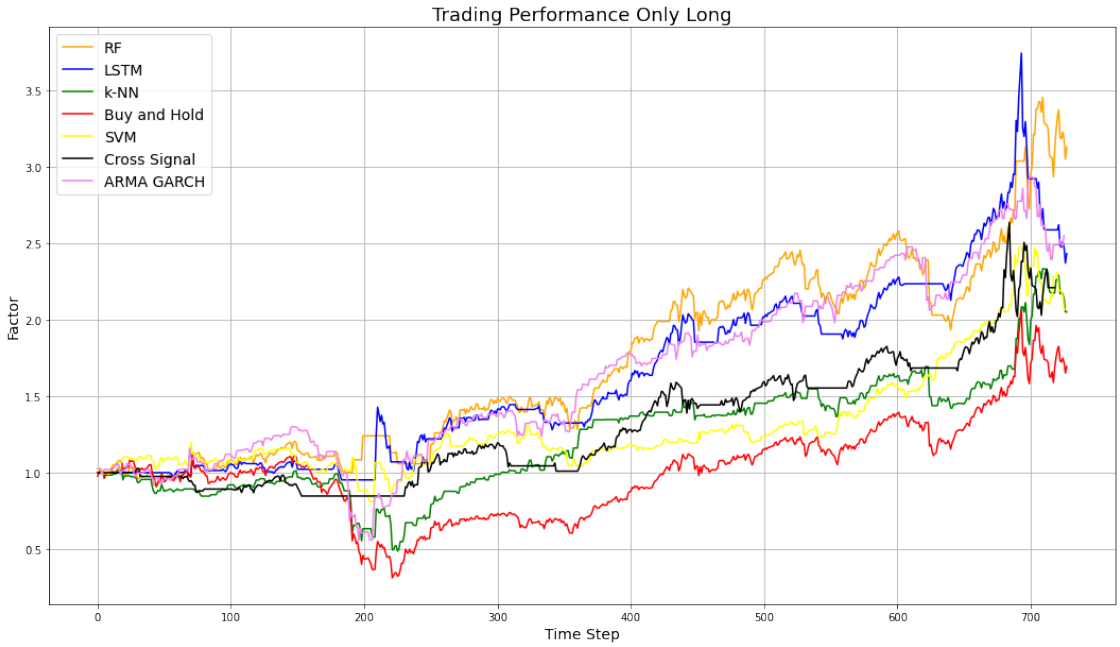
[28]:
```
CS_Perf_data = CS_Perf_data.dropna()
```

[29]:
```
a = list(repeat(1,␣
 ↪len(RF_Perf_data['RF_Perf_OL'])-len(LSTM_Perf_data['LSTM_Perf_OL'])))
a = np.array(a)

b = LSTM_Perf_data['LSTM_Perf_OL']
b = np.array(b)

c = LSTM_Perf_data['LSTM_Perf_LS']
c = np.array(c)

d = np.append(a,b)
e = np.append(a,c)
```

3.1 Only Long Plot

[30]:
```
plt.figure(figsize=(18,10))
plt.title('Trading Performance Only Long', fontsize = 18)
plt.ylabel('Factor', fontsize = 14)
plt.xlabel('Time Step', fontsize = 14)
plt.grid(True)
plt.plot(RF_Perf_data['RF_Perf_OL'],label='RF', color = 'orange')
plt.plot(d, label='LSTM', color = 'blue')
plt.plot(knn_Perf_data['knn_Perf_OL'], label='k-NN', color = 'green')
plt.plot(knn_Perf_data['BaH_Perf'],label='Buy and Hold', color = 'red')
plt.plot(SVM_Perf_data['SVM_Perf_OL'],label='SVM', color = 'yellow')
plt.plot(CS_Perf_data['CS_Perf_OL'].values,label='Cross Signal', color =␣
 ↪'black')
plt.plot(AG_Perf_data['only_long'],label='ARMA GARCH', color = 'violet')
plt.legend(loc='best', fontsize = 14)
```

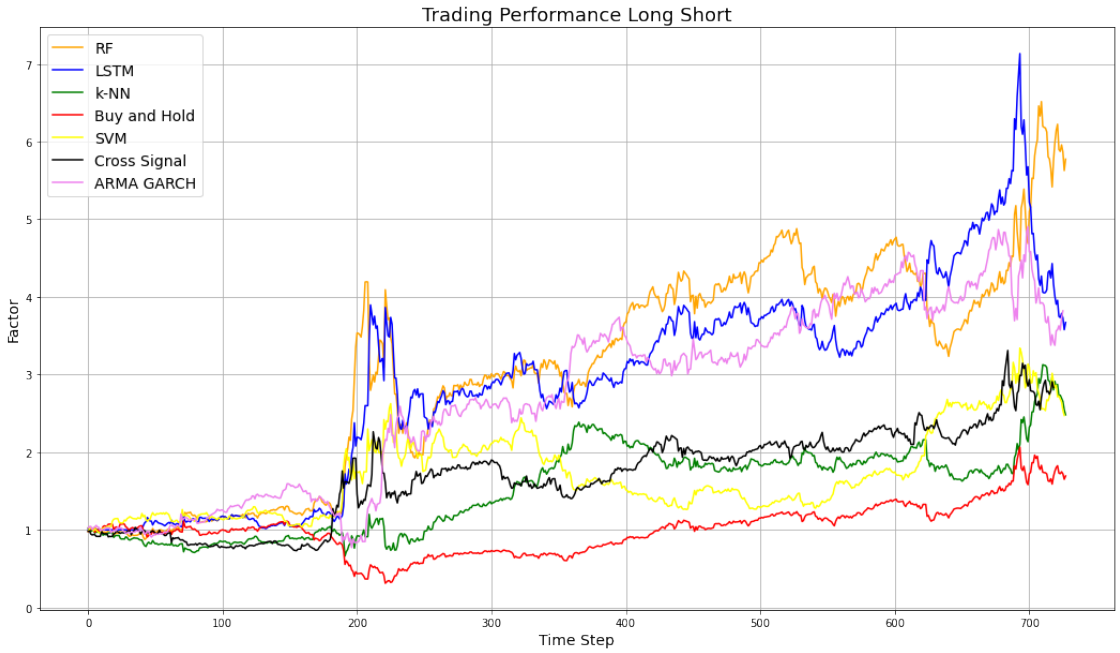
[30]:
```
<matplotlib.legend.Legend at 0x7fd7e667b580>
```



3.2 Long Short Plot

[31]:
```
plt.figure(figsize=(18,10))
plt.title('Trading Performance Long Short', fontsize = 18)
plt.ylabel('Factor', fontsize = 14)
plt.xlabel('Time Step', fontsize = 14)
plt.grid(True)
plt.plot(RF_Perf_data['RF_Perf_LS'],label='RF', color = 'orange')
plt.plot(e, label='LSTM', color = 'blue')
plt.plot(knn_Perf_data['knn_Perf_LS'], label='k-NN', color = 'green')
plt.plot(knn_Perf_data['BaH_Perf'],label='Buy and Hold', color = 'red')
plt.plot(SVM_Perf_data['SVM_Perf_LS'],label='SVM', color = 'yellow')
plt.plot(CS_Perf_data['CS_Perf_LS'].values,label='Cross Signal', color =␣
↪'black')
plt.plot(AG_Perf_data['long_short'],label='ARMA GARCH', color = 'violet')
plt.legend(loc='best', fontsize = 14)
```

[31]:
```
<matplotlib.legend.Legend at 0x7fd7e680ba60>
```


Trading Performance Long Short
RF
LSTM
k-NN
Buy and Hold
SVM
Cross Signal
ARMA GARCH
Factor
Time Step
0
1
2
3
4
5
6
7
0
100
200
300
400
500
600
700

ARMA-GARCH

June 4, 2022

1 ARCH-GARCH Model

1.0.1 Imports

[1]:
```
#Standard imports
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
from statsmodels import *
import statistics as stat
from scipy.stats import *
import yfinance as yf

#Model Imports
from sklearn.metrics import accuracy_score
from arch import arch_model
from statsmodels.tsa.stattools import q_stat
import statsmodels.graphics.tsaplots as sgt
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from pandas.plotting import autocorrelation_plot
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm
import statsmodels.api as smi
import seaborn as sns
import requests
from io import BytesIO
```

[2]:
```
# Importing Dataset
bco = pd.read_csv('/Users/romaindeleze/Desktop/BSc/Romain_WI_TZ/8.Semester/BA/
↪Daten/Brent_Crude_Oil')
```

[3]:
```
bco.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3641 entries, 0 to 3640
```

```
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Date         3641 non-null   object
 1   Close        3641 non-null   float64
 2   Log_Returns  3641 non-null   float64
 3   Pred_Signal  3641 non-null   int64
 4   RSI          3641 non-null   float64
 5   K_percent    3641 non-null   float64
 6   MACD         3641 non-null   float64
 7   ROC          3641 non-null   float64
dtypes: float64(6), int64(1), object(1)
memory usage: 227.7+ KB
```

2 Descriptive Statistics

[4]:
```
def descriptive_statistic(original_data):
    import scipy.stats as ss
    from scipy.stats import skew, kurtosis, jarque_bera
    import numpy as np

    #Original Data
    mean = np.mean(original_data)
    median = np.median(original_data)
    maximum = np.max(original_data)
    minimum = np.min(original_data)
    sd = np.std(original_data)
    skewness = ss.skew(original_data)
    kurtosis = ss.kurtosis(original_data)
    jarqu_bera = ss.jarque_bera(original_data)
    Observations = len(original_data)
    print("Original Data:\n","Mean: ",round(mean,3), "\n", "Median: ",␣
 ↪round(median,3), "\n", "Maximum: ", round(maximum,3), "\n", "Minimum: ",␣
 ↪round(minimum,3),
              "\n", "Standard deviation: ", round(sd,3), "\n", "Skewness: ",␣
 ↪round(skewness,3), "\n", "Kurtosis: ", round(kurtosis,3), "\n"
              "Jarque-Bera: ", jarqu_bera, "\n", "Observations: ",␣
 ↪round(Observations,3))
```

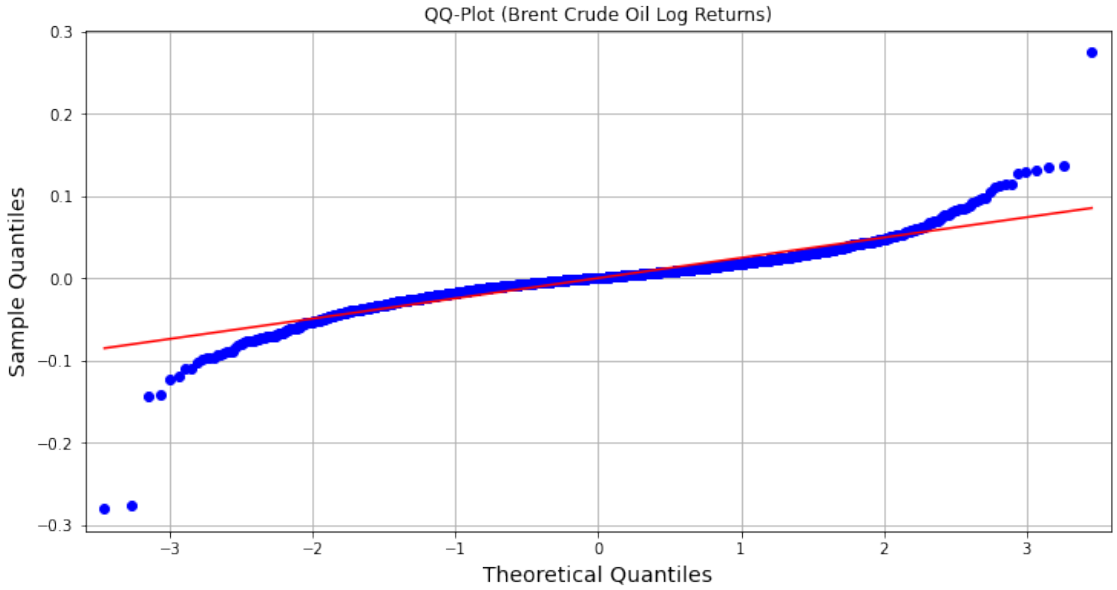
[5]:
```
# a few info about the time series
descriptive_statistic(bco["Close"])
```

```
Original Data:
 Mean:  77.408
 Median:  73.36
 Maximum:  146.08
```

```
 Minimum:  19.33
 Standard deviation:  25.915
 Skewness:  0.226
 Kurtosis:  -1.037
Jarque-Bera:  Jarque_beraResult(statistic=194.05488357708583, pvalue=0.0)
 Observations:  3641
```

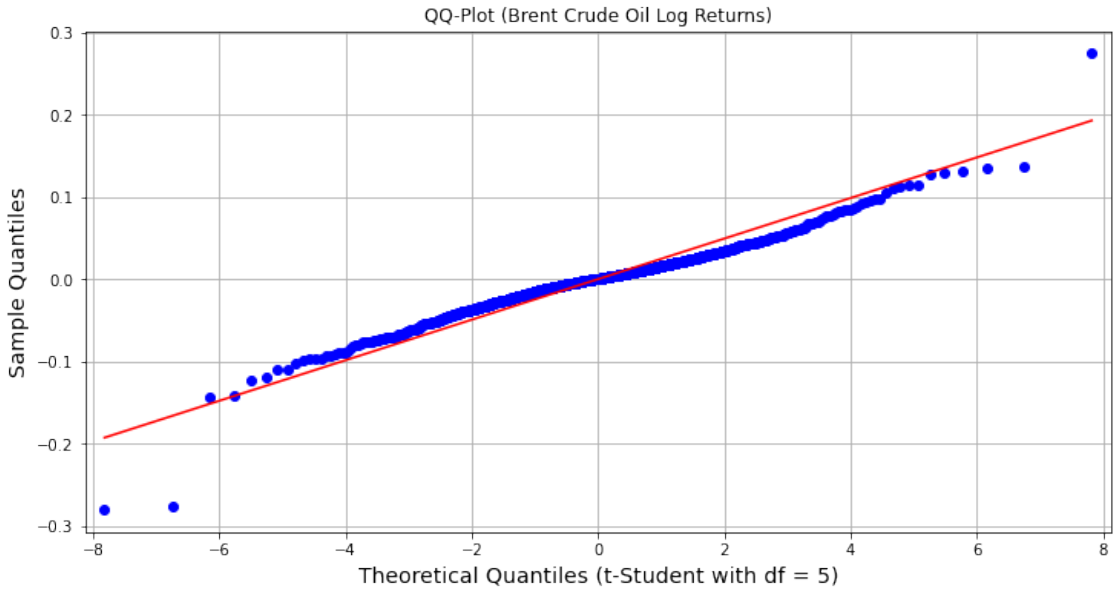
[6]:
```
#QQ Plot of Log Returns Data
plt.figure(figsize=(18,10))
data =  np.squeeze(bco['Log_Returns'])
N, M = 12, 6
fig, ax = plt.subplots(figsize=(N, M))
sm.qqplot(data, line='s',ax=ax)
plt.title('QQ-Plot (Brent Crude Oil Log Returns)')
plt.grid(True)
plt.ylabel('Sample Quantiles', fontsize=14)
plt.xlabel('Theoretical Quantiles', fontsize=14)
plt.show()
plt.show()
```

```
<Figure size 1296x720 with 0 Axes>
```



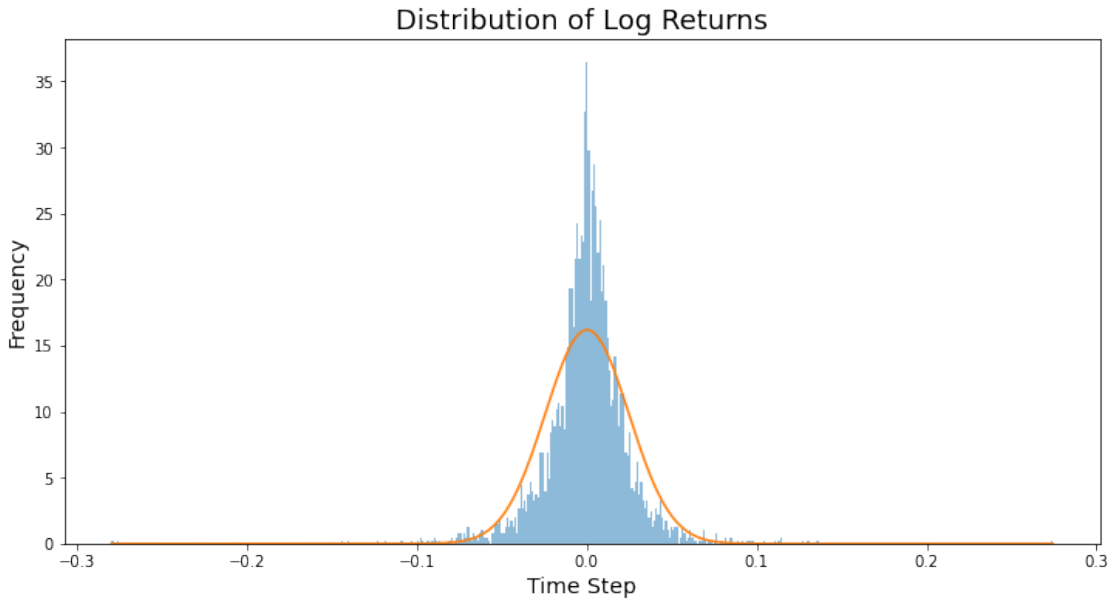
[7]:
```
# QQ-Plot with T-Student Distribution
# Lets check the distribution of the data
import statsmodels.api as sm
import scipy.stats as stats
data =   np.squeeze(bco['Log_Returns'])
```

```
N, M = 12, 6
fig, ax = plt.subplots(figsize=(N, M))
sm.qqplot(data, dist = stats.t, distargs=(5,), line='s',ax=ax)
plt.title('QQ-Plot (Brent Crude Oil Log Returns)')
plt.grid(True)
plt.ylabel('Sample Quantiles', fontsize=14)
plt.xlabel('Theoretical Quantiles (t-Student with df = 5)', fontsize=14)
plt.show()
```



[8]:
```
# Distribution of the log returns
dat = bco["Log_Returns"]
import scipy.stats as ss
plt.figure(figsize=(12,6))
plt.title("Distribution of Log Returns", fontsize=18)
plt.ylabel('Frequency', fontsize=14)
plt.xlabel('Time Step', fontsize=14)
_, bins, _ = plt.hist(dat, 500, density=1, alpha=0.5)

mu, sigma = ss.norm.fit(dat)

best_fit_line = ss.norm.pdf(bins, mu, sigma)

plt.plot(bins, best_fit_line)
```

[8]: [<matplotlib.lines.Line2D at 0x7f82caf7f250>]



3 Train- and Testdata

[9]:
```
Log_Returns = pd.DataFrame(bco['Log_Returns'].loc[1:])
```

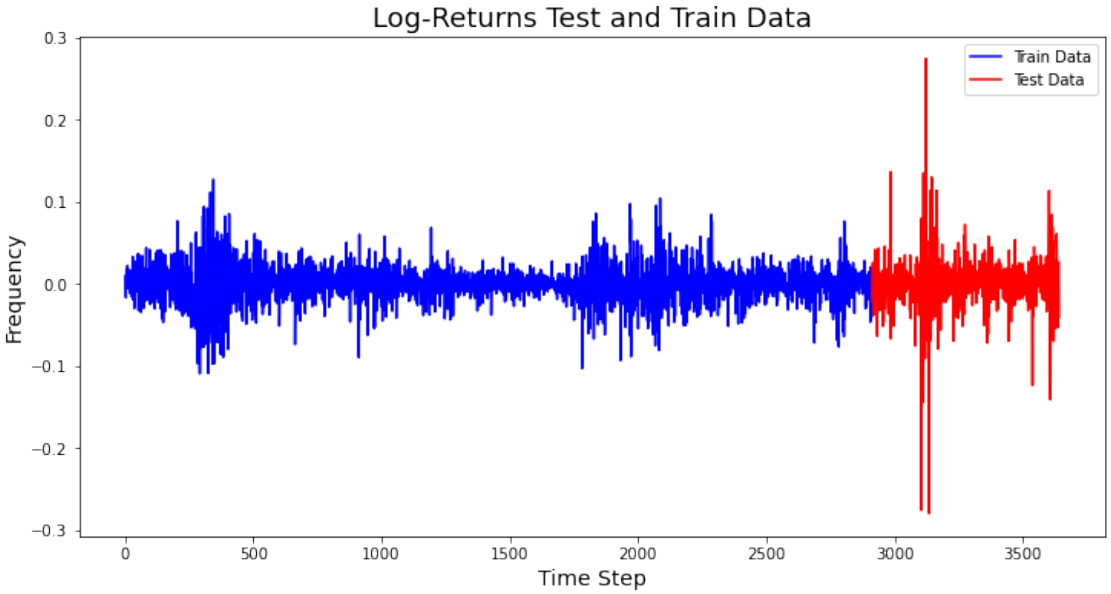
[10]:
```
n = round(0.2 * len(Log_Returns))
split_index = len(Log_Returns) - n
train_df = Log_Returns.iloc[:split_index]
test_df = Log_Returns.iloc[split_index:]

print("Training Set Shape - ", train_df.shape)
print("Testing Set Shape - ", test_df.shape)
```

```
Training Set Shape -  (2912, 1)
Testing Set Shape -  (728, 1)
```

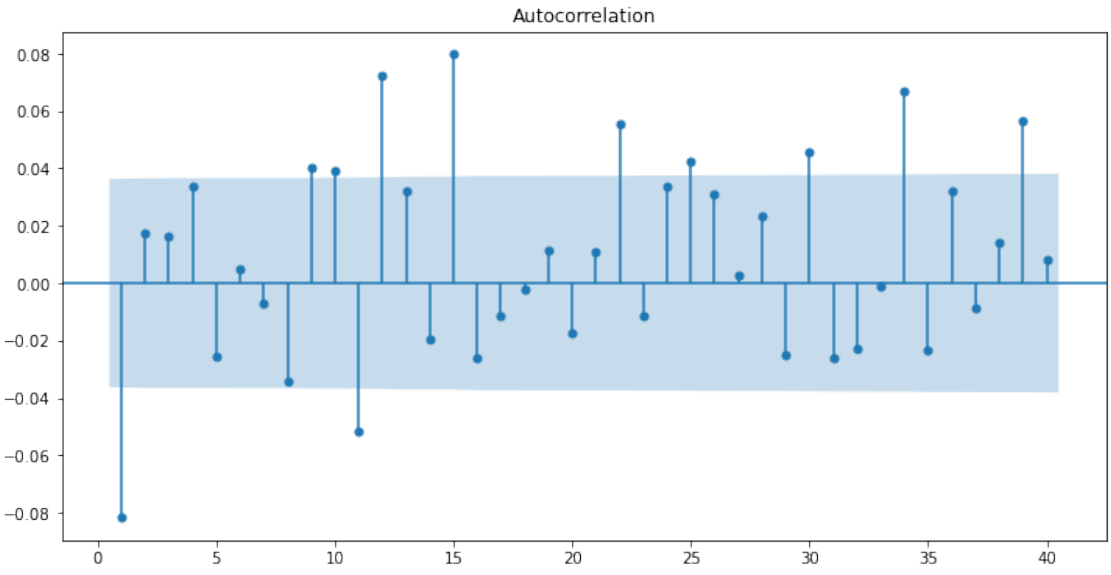
[11]:
```
#Plotting TRain and Test Data
plt.figure(figsize=(12,6))
plt.title("Log-Returns Test and Train Data", fontsize=18)
plt.ylabel('Frequency', fontsize=14)
plt.xlabel('Time Step', fontsize=14)

# Plot the train log returns
plt.plot(train_df, color = "blue", label = "Train Data")

# Plot the test log returns
plt.plot(test_df, color = "red", label = "Test Data")
```

```
plt.legend()
plt.plot()
```

[11]: []

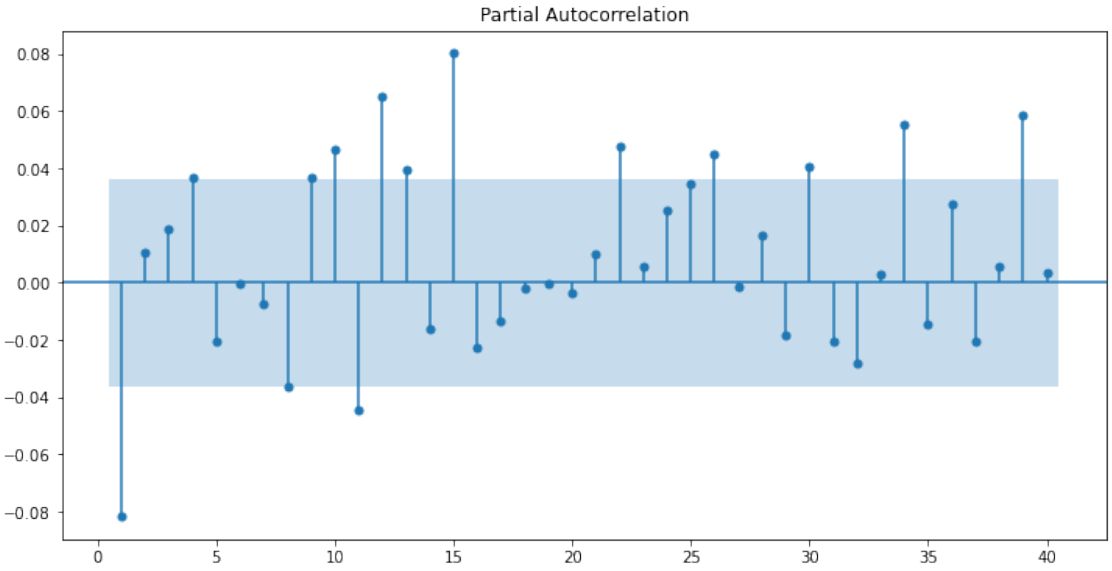


[12]:
```
# Plot autocorrelation
N, M = 12, 6
fig, ax = plt.subplots(figsize=(N, M))
plot_acf(train_df, lags=40, zero = False,ax=ax);
plt.show()
```



For lag 1,4,10,11,12,13,15,22,... is the acf out of the 0.05 level to minimize the model complexety, we decided to use lag 1.

[13]:
```
# Plot partial autocorrelation
N, M = 12, 6
fig, ax = plt.subplots(figsize=(N, M))
plot_pacf(train_df, lags=40, zero = False,ax=ax);
```



For lag 1,4,10,11,12,13,15,22,... is the pacf out of the 0.05 level to minimize the model complexety, we decided to use lag 1.

4 Stationarty test With ADF

[14]:
```
# Checking if time seires is stationary
from statsmodels.tsa.stattools import adfuller

def check_stationarity(series):
    result = adfuller(series.values)

    print('ADF Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))

    if (result[1] <= 0.05) & (result[4]['5%'] > result[0]):
        print("\u001b[32mStationary\u001b[0m")
    else:
```

```
        print("\x1b[31mNon-stationary\x1b[0m")
```

[15]:
```
check_stationarity(train_df)
```

```
ADF Statistic: -11.823858
p-value: 0.000000
Critical Values:
        1%: -3.433
        5%: -2.863
        10%: -2.567
Stationary
```

5 Lagging data

[16]:
```
# Lag Data
def buildLaggedFeatures(s,lag=2,dropna=True):

    if type(s) is pd.DataFrame:
        new_dict={}
        for col_name in s:
            new_dict[col_name]=s[col_name]
            # create lagged Series
            for l in range(1,lag+1):
                new_dict['%s_lag%d' %(col_name,l)]=s[col_name].shift(l)
        res=pd.DataFrame(new_dict,index=s.index)

    elif type(s) is pd.Series:
        the_range=range(lag+1)
        res=pd.concat([s.shift(i) for i in the_range],axis=1)
        res.columns=['lag_%d' %i for i in the_range]
    else:
        print('Only works for DataFrame or Series')
        return None
    if dropna:
        return res.dropna()
    else:
        return res
```

[17]:
```
# Call data lag function
res = buildLaggedFeatures(test_df,lag=1,dropna=False)

# Drop all columns with NA values
test_lag = res.dropna()

# Convert the data to a numpy array
test_lag = np.array(test_lag)
```

```
# Put the data in a data frame
test_lag = pd.DataFrame(test_lag)
```

6 ARMA(1,1)~GARCH(1,1)

[18]:
```
import pmdarima
import arch
from statsmodels.tsa.arima.model import ARIMA

# fit ARIMA on returns
arima = ARIMA(train_df, order=(1,0,1),trend = "ct")
arima_model = arima.fit()
print(arima_model.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:            Log_Returns   No. Observations:                 2912
Model:                 ARIMA(1, 0, 1)   Log Likelihood                7020.987
Date:                Sat, 04 Jun 2022   AIC                         -14031.973
Time:                        17:54:11   BIC                         -14002.090
Sample:                             0   HQIC                        -14021.208
                                - 2912
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0003      0.001      0.420      0.674      -0.001       0.002
x1         -2.174e-07   4.23e-07     -0.514      0.607   -1.05e-06    6.11e-07
ar.L1         -0.1805      0.142     -1.267      0.205      -0.460       0.099
ma.L1          0.0993      0.145      0.685      0.493      -0.185       0.384
sigma2         0.0005   7.33e-06     64.400      0.000       0.000       0.000
===================================================================================
===
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
1710.82
Prob(Q):                              0.99   Prob(JB):
0.00
Heteroskedasticity (H):               0.70   Skew:
-0.15
Prob(H) (two-sided):                  0.00   Kurtosis:
6.74
===================================================================================
===

Warnings:
```

```
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).

/opt/anaconda3/lib/python3.8/site-packages/statsmodels/base/model.py:566:
ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check
mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

[19]:
```
arima_residuals = arima_model.resid
```

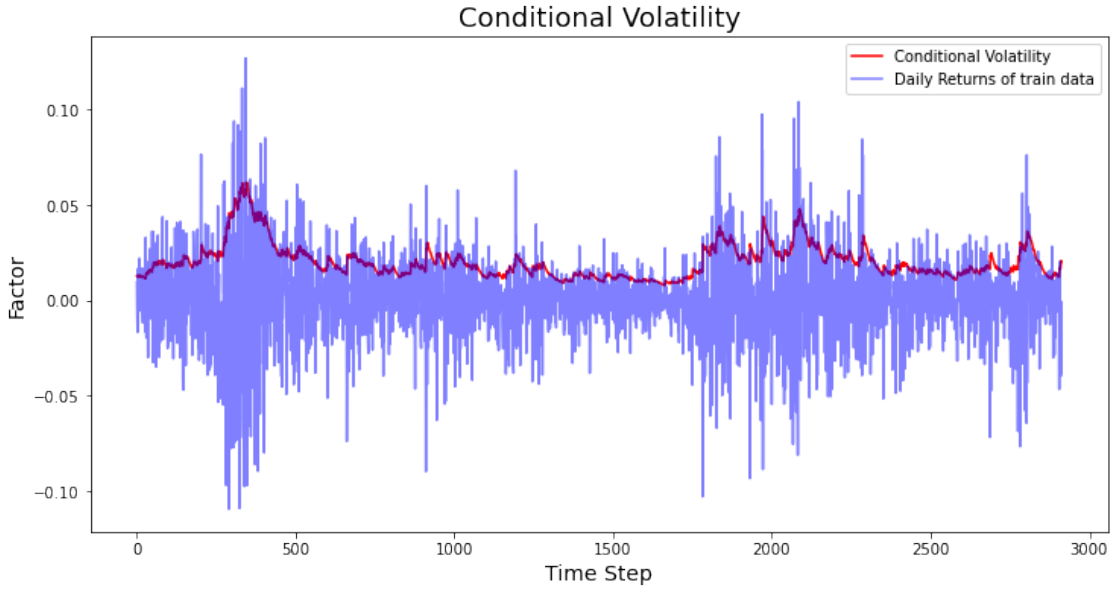
[20]:
```
# fit a ARMA(1,0,1)~GARCH(1,1) model on the residuals of the ARIMA model
garch = arch.arch_model(100*arima_residuals,vol = "GARCH" ,p=1 ,q=1,mean =␣
 ↪"Constant",dist="t")
garch_fitted = garch.fit(disp="off")
print(garch_fitted.summary())
```

```
                        Constant Mean - GARCH Model Results
==============================================================================
====
Dep. Variable:                          None   R-squared:
0.000
Mean Model:                  Constant Mean   Adj. R-squared:
0.000
Vol Model:                           GARCH   Log-Likelihood:
-5868.30
Distribution:      Standardized Student's t   AIC:
11746.6
Method:                  Maximum Likelihood   BIC:
11776.5
                                             No. Observations:
2912
Date:                      Sat, Jun 04 2022   Df Residuals:
2911
Time:                             17:54:11   Df Model:
1
                            Mean Model
==============================================================================
                 coef    std err          t      P>|t|     95.0% Conf. Int.
------------------------------------------------------------------------------
mu             0.0527  2.817e-02      1.872  6.126e-02 [-2.489e-03,  0.108]
                          Volatility Model
==============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
------------------------------------------------------------------------------
omega          0.0185  8.954e-03      2.063  3.909e-02 [9.251e-04,3.603e-02]
alpha[1]       0.0607  9.936e-03      6.112  9.821e-10 [4.126e-02,8.021e-02]
beta[1]        0.9386  9.706e-03     96.696      0.000     [  0.920,  0.958]
                          Distribution
```

```
==============================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
------------------------------------------------------------------------------
nu             5.8290      0.640      9.103  8.774e-20 [  4.574,  7.084]
==============================================================================

Covariance estimator: robust
```

[21]:
```
# Compute conditional volatility
conditional_volatility = garch_fitted.conditional_volatility/100 # rescale the␣
↪cond vola
garch_residuals = garch_fitted.resid
arma_residuals = arima_residuals
```

[22]:
```
# Plot model fitting results
plt.figure(figsize=(12,6))
plt.title('Conditional Volatility', fontsize=18)
plt.ylabel('Factor', fontsize=14)
plt.xlabel('Time Step', fontsize=14)
plt.plot(conditional_volatility, color = 'red', label = 'Conditional␣
↪Volatility')
plt.plot(train_df, color = "blue", label = 'Daily Returns of train data', alpha␣
↪= 0.5)
plt.legend(loc = 'upper right')
plt.show()
```



[23]:
```
# Define test index
ind = test_df.index

# lower index
m = ind.start

# upper index
n = ind.stop

# see the values
m,n
```

[23]:
```
(2913, 3641)
```

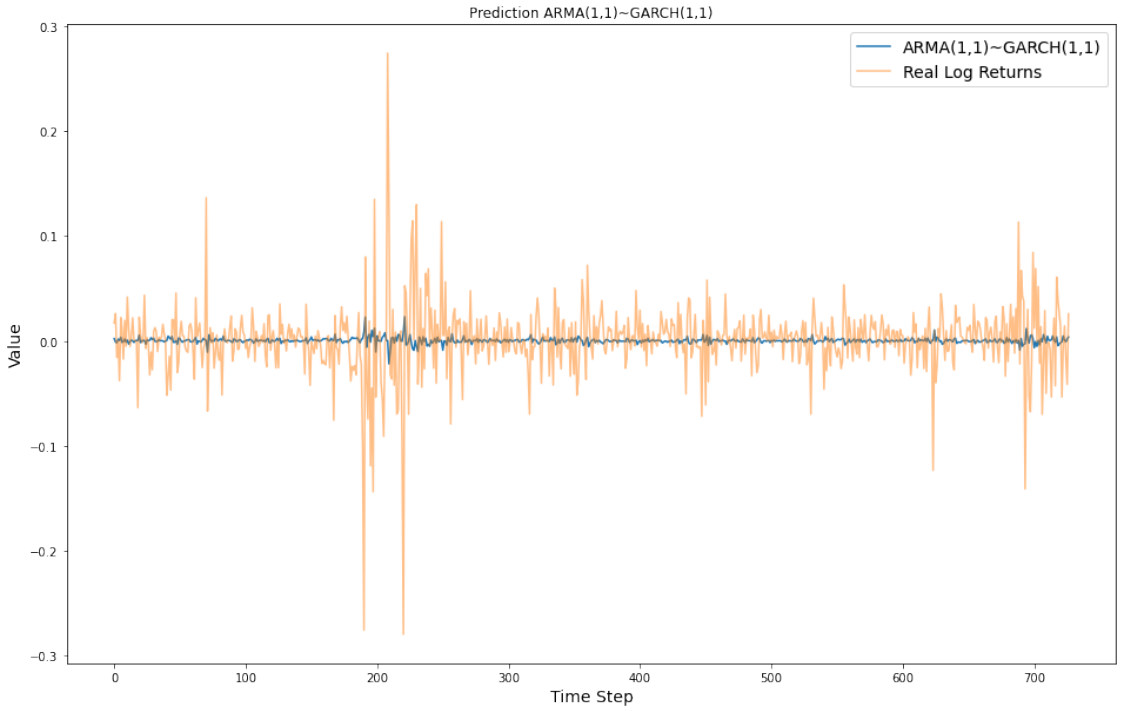
[24]:
```
mean = np.mean(train_df)
std = np.var(train_df)
num_samples = len(test_df)
eps_1 = np.random.normal(mean, std, size=num_samples)
```

[25]:
```
cond_vola = np.array(conditional_volatility)
cond_vola =pd.DataFrame(cond_vola)
cond_vola = np.array(cond_vola[m:n])
cond_vola = pd.DataFrame(cond_vola)

# Call data lag function
res = buildLaggedFeatures(cond_vola,lag=4,dropna=False)

# Drop all columns with NA values
cond_vola = res.dropna()

# Convert the data to a numpy array
cond_vola = np.array(cond_vola)

# Put data in a data frame
eps = pd.DataFrame(cond_vola)
```

[26]:
```
# Compute ARMA(1,1)~GARCH(1,1) prediction
c,x1,ar1,ma1,sigma= arima_model.params
y_t = c + ar1 * test_lag.iloc[:,1]+ ma1 * test_lag.iloc[:,1]

# Plot results
plt.figure(figsize=(16,10))
plt.plot(y_t,label = 'ARMA(1,1)~GARCH(1,1)'),plt.plot(test_lag[0], alpha = 0.
↪5,label = 'Real Log Returns')
plt.legend(loc = 'upper right')
plt.title("Prediction ARMA(1,1)~GARCH(1,1)")
plt.ylabel('Value', fontsize=14)
```

```
plt.xlabel('Time Step', fontsize=14)
plt.legend(loc='best', fontsize=14)
```

[26]: <matplotlib.legend.Legend at 0x7f82ca210fd0>



6.1 Trading Performance

[27]:
```
y_pred = np.sign(y_t)
log_ret = bco["Log_Returns"]
n = len(y_pred)
y_pred = y_pred[:n-1]
X = log_ret.shift(-1)
X = X[len(X)-n:]
X = pd.DataFrame(X).dropna()
```

[28]:
```
# Transform -1,1 Signal in a 0,1 Signal
y_pred_1 = np.array(y_pred)
y_pred1 = []
for i in range(0,len(y_pred_1)):
    if y_pred_1[i] == -1:
        y_pred1.append(0)
    else:
        y_pred1.append(1)
```

```
y_pred1 = pd.DataFrame(y_pred1)
```

[29]:
```
m3 = 4*5*3
real_signal = np.sign(X)
real_signal = np.array(real_signal)
y_pred_Array = np.array(y_pred)
```

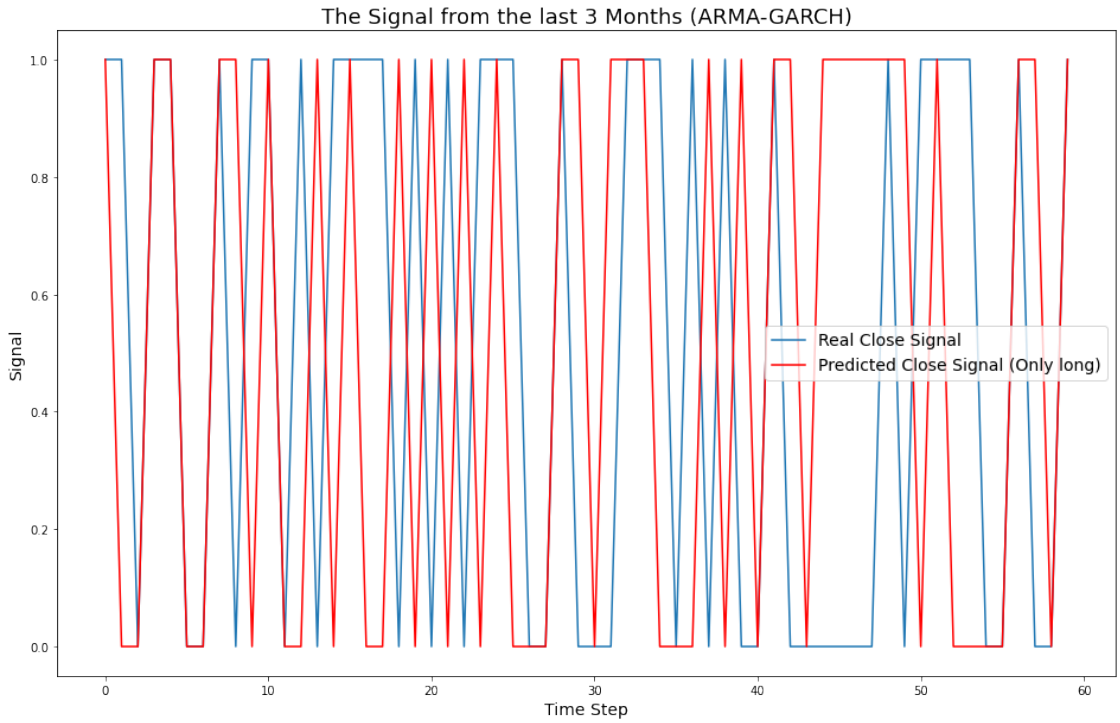
[30]:
```
# Compute the accuracy
print('Accuracy in % : ', accuracy_score(real_signal, y_pred_Array) * 100.0)
print()
```

```
Accuracy in % :  54.269972451790636
```

[31]:
```
# Transform -1,1 Signal in a 0,1 Signal
real_signal_long = np.array(real_signal)
real_signal_L = []
for i in range(0,len(real_signal_long)):
    if real_signal_long[i] == -1:
        real_signal_L.append(0)
    else:
        real_signal_L.append(1)

real_signal_L = pd.DataFrame(real_signal_L)
real_signal_L = np.array(real_signal_L)
```

[32]:
```
# Plot the Predicted Signal with the actual signal
m3 = 4*5*3
y_pred_Array_l = np.array(y_pred1)
plt.figure(figsize=(16,10))
plt.title('The Signal from the last 3 Months (ARMA-GARCH)', fontsize=18)
plt.plot(real_signal_L[-m3:], label = 'Real Close Signal')
plt.plot(y_pred_Array_l[-m3:], color = 'red', label = 'Predicted Close Signal␣
↪(Only long)')
plt.ylabel('Signal', fontsize=14)
plt.xlabel('Time Step', fontsize=14)
plt.legend(loc='best', fontsize=14)
```

[32]:
```
<matplotlib.legend.Legend at 0x7f82ca502280>
```



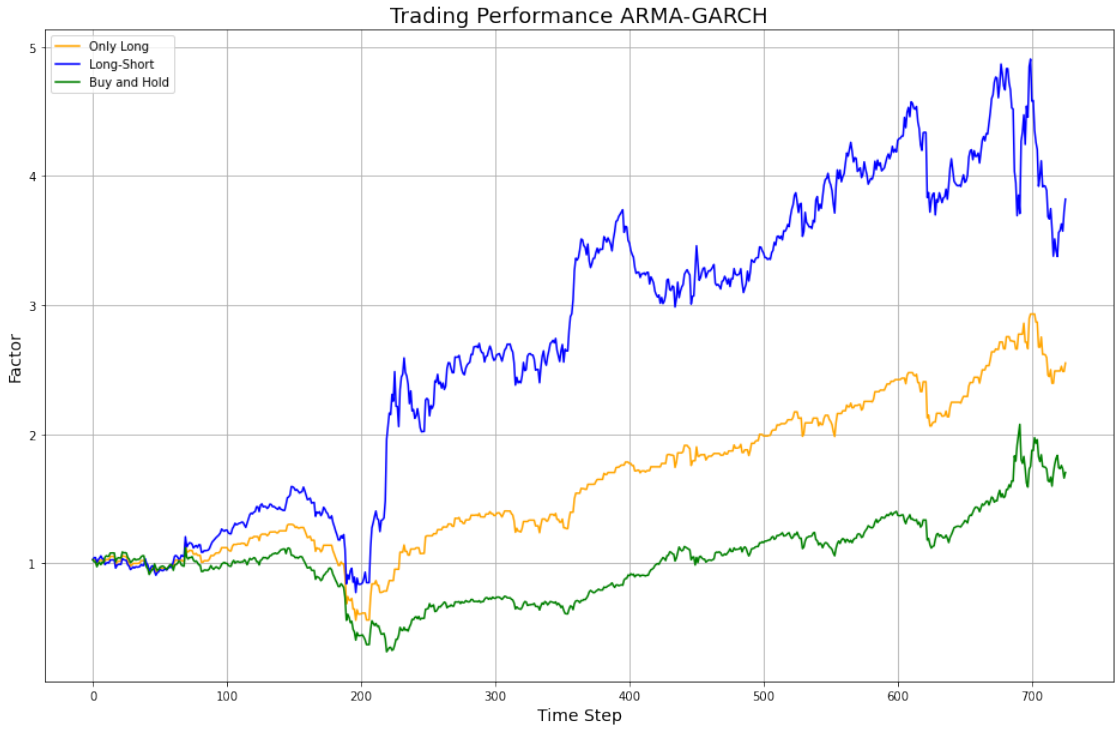
[33]:
```
# Only long
perf = y_pred1.values * X.values
perf = pd.DataFrame(perf)
perf = perf.dropna()

trading_days_Y = 5*52

perf_sharpe = np.sqrt(trading_days_Y) * np.mean(perf) / (np.sqrt(np.var(perf)))
```

[34]:
```
# Long short
y_pred = pd.DataFrame(y_pred)
perf1 = y_pred.values * X.values
perf1 = pd.DataFrame(perf1)
perf1 = perf1.dropna()

trading_days_Y = 5*52

perf_sharpe1 = np.sqrt(trading_days_Y) * np.mean(perf1) / (np.sqrt(np.
↪var(perf1)))
```

[35]:
```
# Buy and Hold
perf2 = np.array(X)

trading_days_Y = 5*52
```

```
perf_sharpe2 = np.sqrt(trading_days_Y) * np.mean(perf2) / (np.sqrt(np.
↪var(perf2)))
```

[36]:
```
# Lets see the performances
plt.figure(figsize=(16,10))
plt.title('Trading Performance ARMA-GARCH', fontsize=18)
plt.ylabel('Factor', fontsize=14)
plt.xlabel('Time Step', fontsize=14)
plt.grid(True)
plt.plot(np.exp(perf).cumprod() , label='Only Long', color = 'orange')
plt.plot(np.exp(perf1).cumprod() , label='Long-Short', color = 'blue')
plt.plot(np.exp(perf2).cumprod() , label='Buy and Hold', color = 'green')
plt.legend(loc='best')
plt.show()
```



[37]:
```
from tabulate import tabulate
# Compute Performances
perf2 = pd.DataFrame(perf2)
perf_0 = np.exp(perf).cumprod().iloc[-1]
perf_1 = np.exp(perf1).cumprod().iloc[-1]
perf_2 = np.exp(perf2).cumprod().iloc[-1]

# Create data
```

```
data = [["Buy and Hold", round(perf_sharpe2,2),round(perf_2,2)],
        ["Only Long", round(perf_sharpe,2),round(perf_0,2)],
        ["Long Short", round(perf_sharpe1,2),round(perf_1,2)]]

# Define header names
col_names = ["Strategie", "Sharp Ratio", "Profit Factor"]

# Display table
print("ARMA-GARCH")
print(tabulate(data, headers=col_names))
```

```
ARMA-GARCH
Strategie       Sharp Ratio    Profit Factor
------------  -------------  ---------------
Buy and Hold           0.35              1.7
Only Long              0.77              2.55
Long Short             0.88              3.82
```

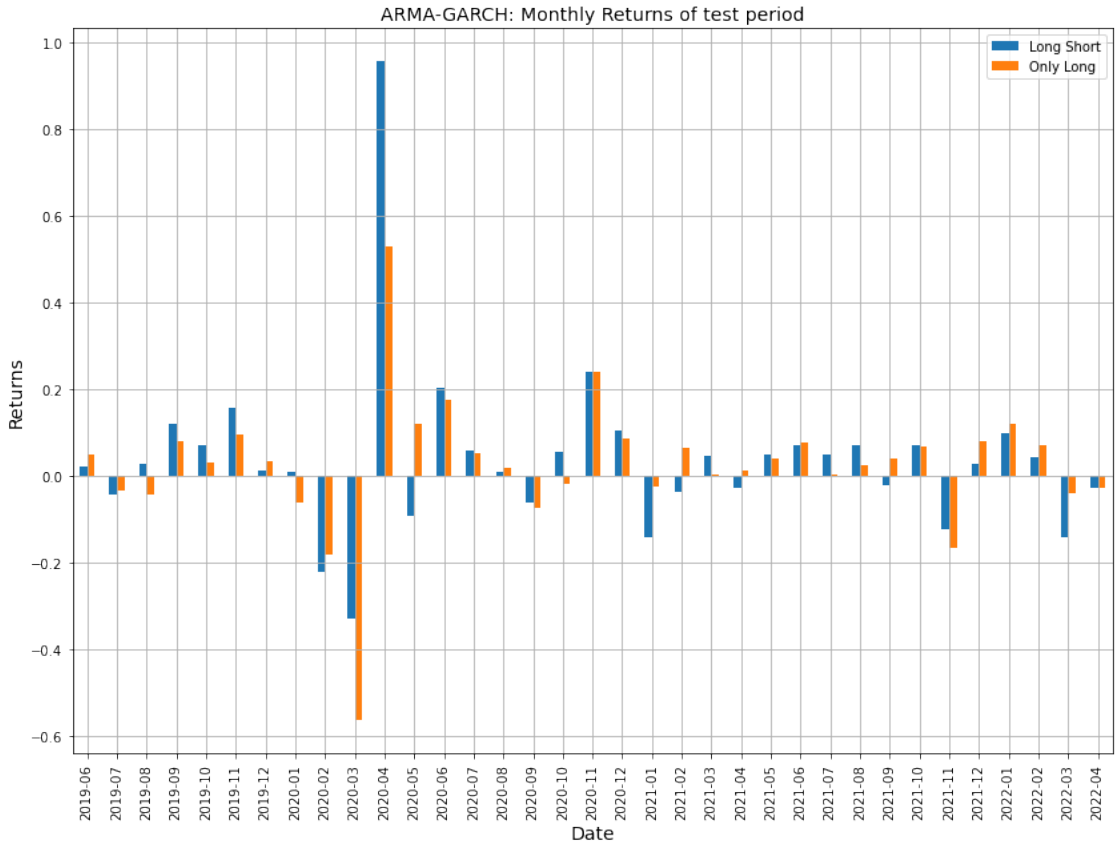
[38]:
```
# Compute the performances
only_long = np.exp(perf).cumprod()
long_short = np.exp(perf1).cumprod()
log_ret = np.array(Log_Returns[m:n])
logret = pd.DataFrame(log_ret)

# Load in data frame
arma_garch_perf_data = pd.concat([only_long, long_short], axis=1)
arma_garch_perf_data.columns  = ["only_long","long_short"]

# Create a csv file
arma_garch_perf_data.to_csv('/Users/romaindeleze/Desktop/BSc/Romain_WI_TZ/8.
↪Semester/BA/Daten/arma_garch_Perf_data')
```

7 Analysis of Performance Plot

[39]:
```
# Indexing
dat = yf.download('BZ=F', start='2007-08-16',end = "2022-04-26",period='1d')
end_len = len(dat["Close"])
long_short = np.exp(perf1).cumprod()
start_len = end_len - len(long_short)
index = dat["Close"][start_len:end_len].index
perf1.index = index
perf.index = index

# Create Monthly Returns Long Short
da = perf1.groupby([(perf1.index.year),(perf1.index.month)]).sum()
monthly_ret_LS = da.values
```

```
n = len(monthly_ret_LS)
monthly_ret_LS = pd.DataFrame(monthly_ret_LS,index = pd.date_range(start='6/1/
 ↪2019', freq='M', periods=n))

# Create Monthly Returns Long
da1 = perf.groupby([(perf.index.year),(perf.index.month)]).sum()
monthly_ret_OL = da1.values
monthly_ret_OL = pd.DataFrame(monthly_ret_OL,index = pd.date_range(start='6/1/
 ↪2019', freq='M', periods=n))

# Connect DF
mon_ret = pd.concat([monthly_ret_LS, monthly_ret_OL], axis=1)
mon_ret.columns = ["Long Short","Only Long"]

# x Axis
x_ax = pd.period_range('6/1/2019', '2022-04-30', freq='M').strftime('%Y-%m')
mon_ret.index = x_ax

# Plot the barplot
ax = mon_ret.plot.bar(figsize=(14,10), rot = 0)
plt.title("ARMA-GARCH: Monthly Returns of test period",fontsize=14)
plt.xlabel('Date', fontsize=14)
plt.xticks(rotation=90)
plt.ylabel('Returns', fontsize=14)
plt.legend(loc='best')
plt.grid()
```

```
[**********100%**********]  1 of 1 completed
```



[40]:
```
# MDD and Maximum Profit
max_profit_LS = np.max(da)
max_prift_OL = np.max(da1)
MDD_LS = np.min(da)
MDD_OL = np.min(da1)

# Create data
data = [["Only Long", round(MDD_OL,2), round(max_prift_OL,2)],
        ["Long Short", round(MDD_LS,2), round(max_profit_LS,2)]]

# Define header names
col_names = ["Strategie", "MDD", "Max. Profit"]

# Display table
print('MDD and MAX.Profti of RF')
print(tabulate(data, headers=col_names))
```

```
MDD and MAX.Profti of RF
Strategie      MDD    Max. Profit
-----------  -----  -------------
Only Long    -0.56           0.53
```

```
Long Short   -0.33           0.96
```

8 Extreme Values

[41]:
```
# Creat Data Frame with Predicted Signal and Log Returns
y = y_pred_Array_1
n = len(y)
X = bco['Log_Returns'].shift(-1)
X = X[len(X)-n:]
y2 = np.squeeze(y) # Dimension Reduction
d = {'Predicted Signal':y2, 'Log Returns':X}
df = pd.DataFrame(data=d)
```

[42]:
```
# Creat the upperbound and the lowerbound
mu = np.mean(X)
sig = np.std(X)
lb = mu-2*sig
ub = mu+2*sig

# Creat Data Frame with only Values in the interval [lb,ub]
reg_val = df[(df['Log Returns'] >= lb) & (df['Log Returns'] <= ub)] # DataFrame␣
 ↪with Regular Values = [lb,ub]

#print(len(reg_val))# Creat Data Frame with only Values outside the interval␣
 ↪[lb,ub]
ext_val = df.drop(index=reg_val.index)

# Add real Signal with the Log Returns for each Data Frame
reg_val['Real Signal'] = (reg_val['Log Returns'] > 0).astype(int)
ext_val['Real Signal'] = (ext_val['Log Returns'] > 0).astype(int)

# Compute accuracy of the Extrem Values and the Regular Values
acc_reg = accuracy_score(reg_val['Real Signal'], reg_val['Predicted Signal'],␣
 ↪normalize = True) * 100.0
acc_ext = accuracy_score(ext_val['Real Signal'], ext_val['Predicted Signal'],␣
 ↪normalize = True) * 100.0
print('Accuracy Regular Values = ', round(acc_reg,2))
print('Accuracy Exrtem Values (95%) = ', round(acc_ext,2))
```

```
Accuracy Regular Values =  51.3
Accuracy Exrtem Values (95%) =  52.78
```

```
<ipython-input-42-83d37511bd47>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
```

```
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  reg_val['Real Signal'] = (reg_val['Log Returns'] > 0).astype(int)
```

[43]:
```
# Creat the upperbound and the lowerbound
mu = np.mean(X)
sig = np.std(X)
lb = mu-3*sig
ub = mu+3*sig

# Creat Data Frame with only Values in the interval [lb,ub]
reg_val = df[(df['Log Returns'] >= lb) & (df['Log Returns'] <= ub)] # DataFrame␣
 ↪with Regular Values = [lb,ub]

#print(len(reg_val))# Creat Data Frame with only Values outside the interval␣
 ↪[lb,ub]
ext_val = df.drop(index=reg_val.index)

# Add real Signal with the Log Returns for each Data Frame
reg_val['Real Signal'] = (reg_val['Log Returns'] > 0).astype(int)
ext_val['Real Signal'] = (ext_val['Log Returns'] > 0).astype(int)

# Compute accuracy of the Extrem Values and the Regular Values
acc_reg = accuracy_score(reg_val['Real Signal'], reg_val['Predicted Signal'],␣
 ↪normalize = True) * 100.0
acc_ext = accuracy_score(ext_val['Real Signal'], ext_val['Predicted Signal'],␣
 ↪normalize = True) * 100.0
print('Accuracy Regular Values = ', round(acc_reg,2))
print('Accuracy Exrtem Values (99%) = ', round(acc_ext,2))
```

```
Accuracy Regular Values =  51.05
Accuracy Exrtem Values (99%) =  66.67

<ipython-input-43-d2a3f5473bce>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  reg_val['Real Signal'] = (reg_val['Log Returns'] > 0).astype(int)
```

[ ]:

Cross_Signal_Trade

June 4, 2022

1 Cross Signal Trade

1.0.1 Imports

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tabulate import tabulate
import yfinance as yf
```

[2]:
```
# From 16.08.2007 to 26.04.2022
bco = pd.read_csv('/Users/romaindeleze/Desktop/BSc/Romain_WI_TZ/8.Semester/BA/
 ↪Daten/Brent_Crude_Oil')
```

[3]:
```
bco.head()
```

[3]:

	Date	Close	Log_Returns	Pred_Signal	RSI	K_percent	MACD	ROC
0	2007-08-16	69.769997	-0.026449	1	29.797228	9.756045	-0.572856	-0.066622
1	2007-08-17	70.440002	0.009557	0	37.710864	17.926851	-0.563372	-0.010257
2	2007-08-20	69.849998	-0.008411	0	33.868557	11.206854	-0.584877	-0.027159
3	2007-08-21	68.690002	-0.016746	1	27.466358	6.351211	-0.665806	-0.032399
4	2007-08-22	68.699997	0.000145	1	27.604705	6.649574	-0.718980	-0.021507

[4]:
```
# Add data / transform data
```

[5]:
```
# Add moving avarage to the dataset
bco['3_week'] = bco['Close'].rolling(15).mean().shift() # 5*3 Trading days = 15␣
 ↪Days
bco['9_week'] = bco['Close'].rolling(45).mean().shift() # 5*9 Trading days = 45␣
 ↪Days
bco[44:50]
```

[5]:

	Date	Close	Log_Returns	Pred_Signal	RSI	K_percent	\
44	2007-10-18	84.599998	0.017529	0	78.073526	98.892597	
45	2007-10-19	83.790001	-0.009621	0	70.295228	89.922523	
46	2007-10-22	83.269997	-0.006225	0	65.420567	84.163893	
47	2007-10-23	82.849998	-0.005057	1	61.426060	79.512749	
48	2007-10-24	84.370003	0.018180	1	69.294757	96.345579	
49	2007-10-25	87.480003	0.036198	1	79.153322	99.243728	

	MACD	ROC	3_week	9_week
44	1.880058	0.072243	79.484666	NaN
45	1.955213	0.094150	79.789333	76.208667
46	1.950672	0.074590	80.097333	76.520222
47	1.891635	0.054071	80.472666	76.805333
48	1.944102	0.052651	80.837333	77.094222
49	2.209307	0.086034	81.303334	77.442667

[6]:
```
# Add long-short Signal Column
bco['LS_signal'] = np.where(bco['3_week'] > bco['9_week'],1,0) # Long Signal
bco['LS_signal'] = np.where(bco['3_week'] < bco['9_week'],-1,bco['LS_signal'])␣
↪# Short Signal

# Add only long Signal
bco['L_signal'] = np.where(bco['3_week'] > bco['9_week'],1,0) # Long Signal

# drop na values
bco.dropna(inplace=True)
```

[7]:
```
# Calculate return/system retruns

# System returns from long-short System
bco['LS_system_return'] = bco['LS_signal'] * bco['Log_Returns']

# System returns from only long System
bco['L_system_return'] = bco['L_signal'] * bco['Log_Returns']

bco.head()
```

[7]:

	Date	Close	Log_Returns	Pred_Signal	RSI	K_percent	\
45	2007-10-19	83.790001	-0.009621	0	70.295228	89.922523	
46	2007-10-22	83.269997	-0.006225	0	65.420567	84.163893	
47	2007-10-23	82.849998	-0.005057	1	61.426060	79.512749	
48	2007-10-24	84.370003	0.018180	1	69.294757	96.345579	
49	2007-10-25	87.480003	0.036198	1	79.153322	99.243728	

	MACD	ROC	3_week	9_week	LS_signal	L_signal	\
45	1.955213	0.094150	79.789333	76.208667	1	1	
46	1.950672	0.074590	80.097333	76.520222	1	1	

```
47  1.891635  0.054071  80.472666  76.805333          1         1
48  1.944102  0.052651  80.837333  77.094222          1         1
49  2.209307  0.086034  81.303334  77.442667          1         1

    LS_system_return  L_system_return
45         -0.009621        -0.009621
46         -0.006225        -0.006225
47         -0.005057        -0.005057
48          0.018180         0.018180
49          0.036198         0.036198
```

[8]:
```
# Calculate return/system retruns Test set

h = round(len(bco['Close'])*0.2)

bco['B_a_H_system_test'] = bco.iloc[-h:]['Log_Returns']

# System returns from 1 year long-short System
bco['LS_system_return_test'] = bco.iloc[-h:]['LS_signal'] * bco.iloc[-h:
 ↪]['Log_Returns']

# System returns from 1 year only long System
bco['L_system_return_test'] = bco.iloc[-h:]['L_signal'] * bco.iloc[-h:
 ↪]['Log_Returns']

bco.head()
```

[8]:
```
           Date      Close  Log_Returns  Pred_Signal        RSI  K_percent  \
45  2007-10-19  83.790001    -0.009621            0  70.295228  89.922523
46  2007-10-22  83.269997    -0.006225            0  65.420567  84.163893
47  2007-10-23  82.849998    -0.005057            1  61.426060  79.512749
48  2007-10-24  84.370003     0.018180            1  69.294757  96.345579
49  2007-10-25  87.480003     0.036198            1  79.153322  99.243728

        MACD       ROC     3_week     9_week  LS_signal  L_signal  \
45  1.955213  0.094150  79.789333  76.208667          1         1
46  1.950672  0.074590  80.097333  76.520222          1         1
47  1.891635  0.054071  80.472666  76.805333          1         1
48  1.944102  0.052651  80.837333  77.094222          1         1
49  2.209307  0.086034  81.303334  77.442667          1         1

    LS_system_return  L_system_return  B_a_H_system_test  \
45         -0.009621        -0.009621                NaN
46         -0.006225        -0.006225                NaN
47         -0.005057        -0.005057                NaN
48          0.018180         0.018180                NaN
49          0.036198         0.036198                NaN
```

```
    LS_system_return_test  L_system_return_test
45                    NaN                   NaN
46                    NaN                   NaN
47                    NaN                   NaN
48                    NaN                   NaN
49                    NaN                   NaN
```

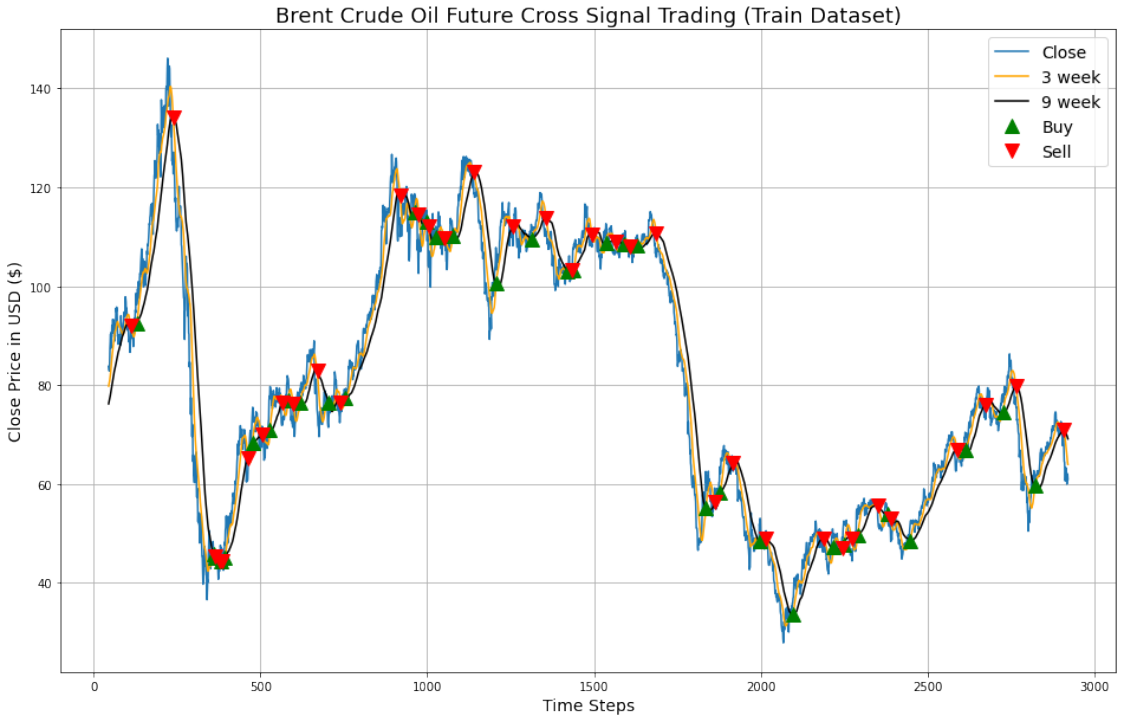
[9]:
```
bco['LS_entry'] = bco.LS_signal.diff()

bco['L_entry'] = bco.L_signal.diff()

bco.head()
```

[9]:
```
          Date      Close  Log_Returns  Pred_Signal        RSI  K_percent  \
45  2007-10-19  83.790001    -0.009621            0  70.295228  89.922523
46  2007-10-22  83.269997    -0.006225            0  65.420567  84.163893
47  2007-10-23  82.849998    -0.005057            1  61.426060  79.512749
48  2007-10-24  84.370003     0.018180            1  69.294757  96.345579
49  2007-10-25  87.480003     0.036198            1  79.153322  99.243728

        MACD       ROC     3_week     9_week  LS_signal  L_signal  \
45  1.955213  0.094150  79.789333  76.208667          1         1
46  1.950672  0.074590  80.097333  76.520222          1         1
47  1.891635  0.054071  80.472666  76.805333          1         1
48  1.944102  0.052651  80.837333  77.094222          1         1
49  2.209307  0.086034  81.303334  77.442667          1         1

    LS_system_return  L_system_return  B_a_H_system_test  \
45         -0.009621       -0.009621                NaN
46         -0.006225       -0.006225                NaN
47         -0.005057       -0.005057                NaN
48          0.018180        0.018180                NaN
49          0.036198        0.036198                NaN

    LS_system_return_test  L_system_return_test  LS_entry  L_entry
45                    NaN                   NaN       NaN      NaN
46                    NaN                   NaN       0.0      0.0
47                    NaN                   NaN       0.0      0.0
48                    NaN                   NaN       0.0      0.0
49                    NaN                   NaN       0.0      0.0
```

2 Buy Sell Chart (Full Dataset)

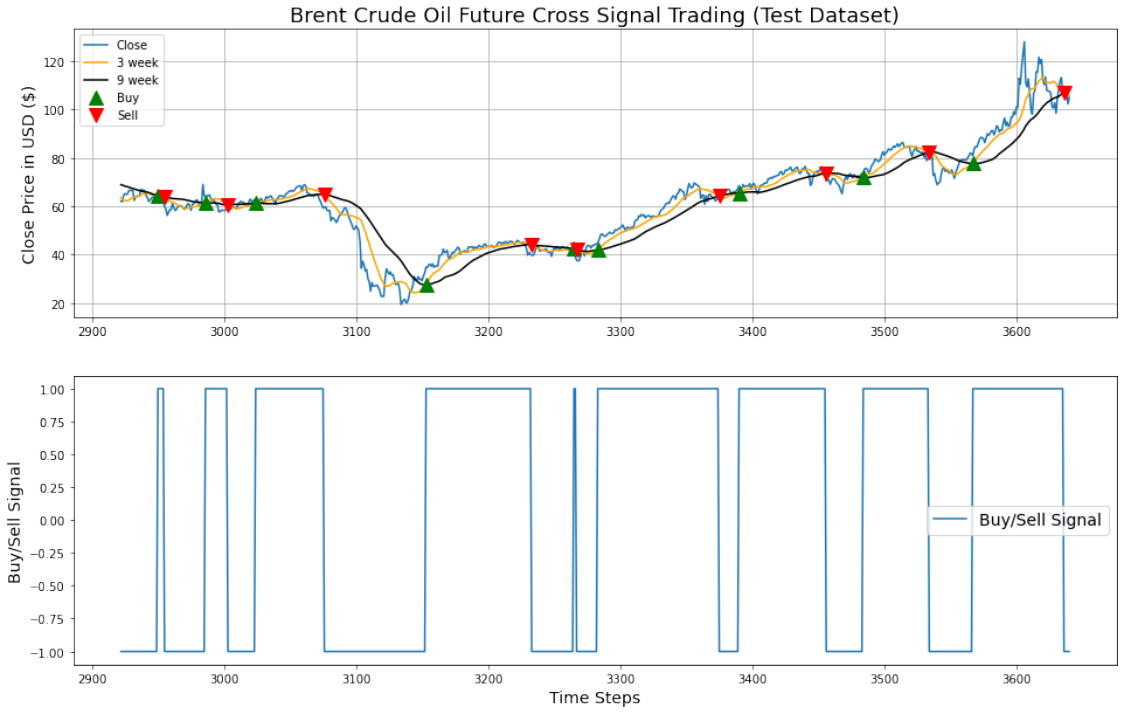
[10]:
```
plt.figure(figsize=(16,10))
plt.grid(True)
plt.title('Brent Crude Oil Future Cross Signal Trading (Train Dataset)',␣
 ↪fontsize=18)
plt.xlabel('Time Steps', fontsize=14)
plt.ylabel('Close Price in USD ($)', fontsize=14)
plt.plot(bco.iloc[:-h]['Close'], label='Close')
plt.plot(bco.iloc[:-h]['3_week'], label='3 week', color = 'orange')
plt.plot(bco.iloc[:-h]['9_week'], label='9 week', color = 'black')
plt.plot(bco[:-h].loc[bco.LS_entry == 2].index, bco[:-h]['3_week'][bco.LS_entry␣
 ↪== 2],'^',
        color = 'green', markersize=12, label='Buy')
plt.plot(bco[:-h].loc[bco.LS_entry == -2].index, bco[:-h]['9_week'][bco.
 ↪LS_entry == -2],'v',
        color = 'red', markersize=12, label='Sell')
plt.legend(loc='best',fontsize=14)
```

[10]:
```
<matplotlib.legend.Legend at 0x7feceba90910>
```



3 Buy Sell Chart (Testset)

[11]:
```
fig, axs = plt.subplots(2, 1, figsize=(16,10))
axs[0].set_title('Brent Crude Oil Future Cross Signal Trading (Test Dataset)',␣
 ↪fontsize=18)
axs[0].set_ylabel('Close Price in USD ($)', fontsize=14)
axs[0].grid(True)
axs[0].plot(bco.iloc[-h:]['Close'], label='Close')
axs[0].plot(bco.iloc[-h:]['3_week'], label='3 week', color = 'orange')
axs[0].plot(bco.iloc[-h:]['9_week'], label='9 week', color = 'black')
axs[0].plot(bco[-h:].loc[bco.LS_entry == 2].index, bco[-h:]['3_week'][bco.
 ↪LS_entry == 2],'^',
        color = 'green', markersize=12, label='Buy')
axs[0].plot(bco[-h:].loc[bco.LS_entry == -2].index, bco[-h:]['9_week'][bco.
 ↪LS_entry == -2],'v',
        color = 'red', markersize=12, label='Sell')
axs[0].legend(loc='best')


axs[1].plot(bco.iloc[-h:]['LS_signal'], label ='Buy/Sell Signal')
axs[1].set_ylabel('Buy/Sell Signal', fontsize=14)
axs[1].set_xlabel('Time Steps', fontsize=14)
axs[1].legend(loc='best', fontsize=14)

plt.show()
```



4 Traging Rules

4.0.1 The trading strategy is very simple to understand. The following rules are used:

1. The 3 weeks moving average crosses the 9 weeks moving average from the bottom to the top so it is a buy signal.

2. The 3 weeks moving average crosses the 9 weeks moving average from top to bottom, it is a sell signal.

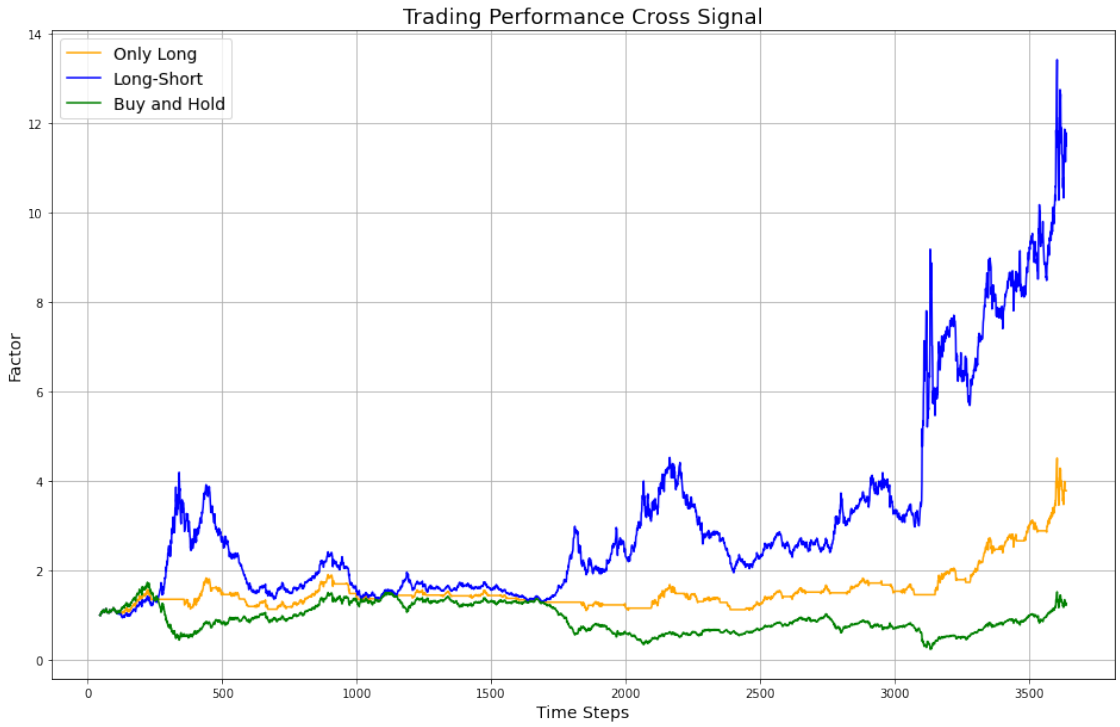
We have tried this strategy with two different signals. The first signal long-short buys at the buy signal and makes shortselling at the sell signal. The second signal only long buys at the buy signal and sells at the sell signal.

5 Comparison Cross_Signal_Strategie vs. Buy and Hold (Full Dataset)

[12]:
```
# Calculat Performance of whole Dataset
B_a_H = np.exp(bco['Log_Returns']).cumprod()
L_perf = np.exp(bco['L_system_return']).cumprod()
LS_perf = np.exp(bco['LS_system_return']).cumprod()
```

[13]:
```
plt.figure(figsize=(16,10))
plt.title('Trading Performance Cross Signal', fontsize=18)
plt.ylabel('Factor', fontsize=14)
plt.xlabel('Time Steps', fontsize=14)
plt.grid(True)
plt.plot(L_perf, label='Only Long', color = 'orange')
plt.plot(LS_perf, label='Long-Short', color = 'blue')
plt.plot(B_a_H, label='Buy and Hold', color = 'green')
plt.legend(loc='best', fontsize=14)
```

[13]:
```
<matplotlib.legend.Legend at 0x7fece765b130>
```

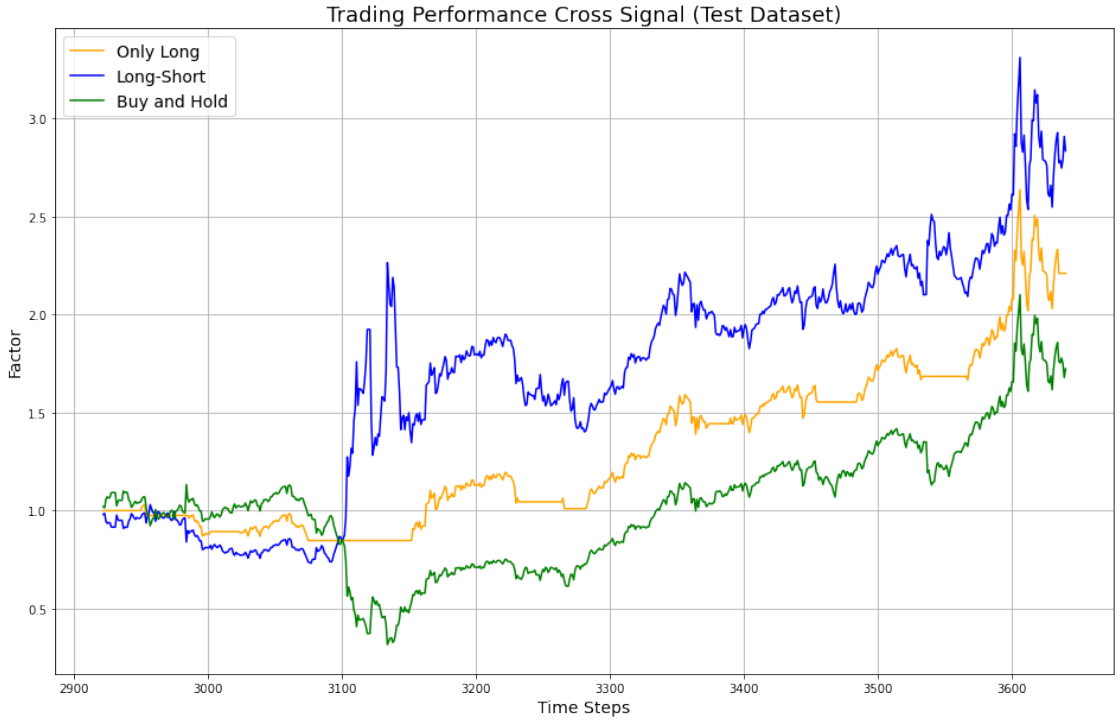


6 Comparison Cross Signal Strategie vs. Buy and Hold (Testset)

[14]:
```
# Calculat Performance Testset
B_a_H_y1 = np.exp(bco['B_a_H_system_test']).cumprod()
L_perf_y1 = np.exp(bco['L_system_return_test']).cumprod()
LS_perf_y1 = np.exp(bco['LS_system_return_test']).cumprod()
```

[15]:
```
plt.figure(figsize=(16,10))
plt.title('Trading Performance Cross Signal (Test Dataset)', fontsize=18)
plt.ylabel('Factor', fontsize=14)
plt.xlabel('Time Steps', fontsize=14)
plt.grid(True)
plt.plot(L_perf_y1, label='Only Long', color = 'orange')
plt.plot(LS_perf_y1, label='Long-Short', color = 'blue')
plt.plot(B_a_H_y1, label='Buy and Hold', color = 'green')
plt.legend(loc='best',fontsize=14)
```

[15]: <matplotlib.legend.Legend at 0x7fece7679af0>



7 Sharpe Ratios (Testset)

[16]:
```
# Shapre Ratio Calculation (Testdata)
trading_days_Y = 5*52

# Only Long
perf = bco['L_system_return_test']
perf_sharpe = np.sqrt(trading_days_Y) * np.mean(perf) / (np.sqrt(np.var(perf)))

# Long Short
perf1 = bco['LS_system_return_test']
perf_sharpe1 = np.sqrt(trading_days_Y) * np.mean(perf1) / (np.sqrt(np.
 ↪var(perf1)))

# Buy and Hold
perf2 = bco['B_a_H_system_test']
perf_sharpe2 = np.sqrt(trading_days_Y) * np.mean(perf2) / (np.sqrt(np.
 ↪var(perf2)))
```

[17]:
```
# Performance Dataset
CS_Perf_OL = L_perf_y1
CS_Perf_LS = LS_perf_y1
```

```
CS_Perf_data = pd.DataFrame()
CS_Perf_data['CS_Perf_OL'] = CS_Perf_OL
CS_Perf_data['CS_Perf_LS'] = CS_Perf_LS

CS_Perf_data.to_csv('/Users/romaindeleze/Desktop/BSc/Romain_WI_TZ/8.Semester/BA/
 ↪Daten/CS_Perf_data')
```

8 Profit Factor (Testset)

[18]:
```
L_profit = L_perf_y1.iloc[-1]

LS_profit = LS_perf_y1.iloc[-1]

B_a_H_profit = B_a_H_y1.iloc[-1]
```

9 Performance Table

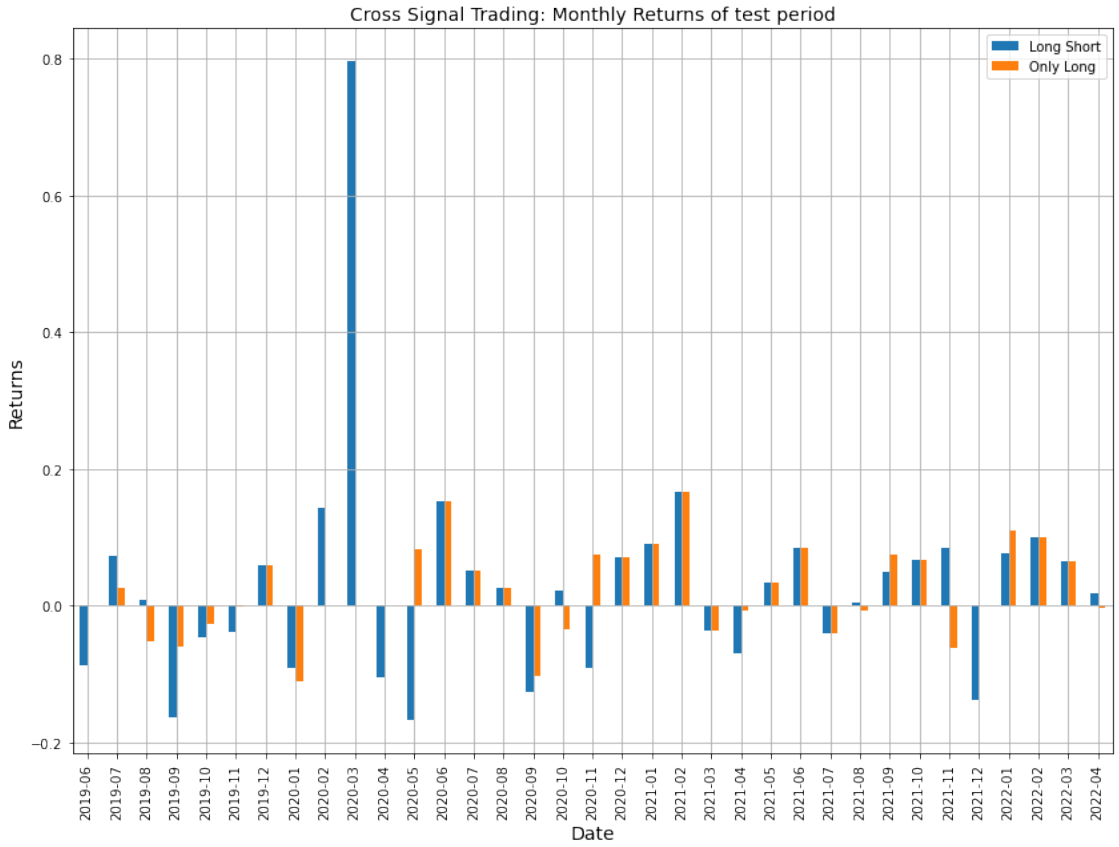
[19]:
```
#create data
data = [["Buy and Hold", round(perf_sharpe2,2), round(B_a_H_profit,2)],
        ["Only Long", round(perf_sharpe,2), round(L_profit,2)],
        ["Long Short", round(perf_sharpe1,2), round(LS_profit,2)]]

#define header names
col_names = ["Strategie", "Sharp Ratio", "Profit Factor"]

#display table
print('Cross Signal Trade')
print(tabulate(data, headers=col_names))
```

```
Cross Signal Trade
Strategie       Sharp Ratio    Profit Factor
------------  -------------  ---------------
Buy and Hold           0.36             1.72
Only Long              0.95             2.21
Long Short             0.69             2.83
```

[20]:
```
# Indexing
dat = yf.download('BZ=F', start='2007-08-16',end = "2022-04-26",period='1d')
end_len = len(dat["Close"])
long_short = perf.dropna()
start_len = end_len - len(long_short)
index = dat["Close"][start_len:end_len].index
perf1 = perf1.dropna()
perf = perf.dropna()
perf1.index = index
```

```
perf.index = index

# Create Monthly Returns Long Short
da = perf1.groupby([(perf1.index.year),(perf1.index.month)]).sum()
monthly_ret_LS = da.values
n = len(monthly_ret_LS)
monthly_ret_LS = pd.DataFrame(monthly_ret_LS,index = pd.date_range(start='6/1/
 ↪2019', freq='M', periods=n))

# Create Monthly Returns Long
da1 = perf.groupby([(perf.index.year),(perf.index.month)]).sum()
monthly_ret_OL = da1.values
monthly_ret_OL = pd.DataFrame(monthly_ret_OL,index = pd.date_range(start='6/1/
 ↪2019', freq='M', periods=n))

# Connect DF
mon_ret = pd.concat([monthly_ret_LS, monthly_ret_OL], axis=1)
mon_ret.columns = ["Long Short","Only Long"]

# x Axis
x_ax = pd.period_range('6/1/2019', '2022-04-30', freq='M').strftime('%Y-%m')
mon_ret.index = x_ax

# Plot the barplot
ax = mon_ret.plot.bar(figsize=(14,10), rot = 0)
plt.title("Cross Signal Trading: Monthly Returns of test period",fontsize=14)
plt.xlabel('Date', fontsize=14)
plt.xticks(rotation=90)
plt.ylabel('Returns', fontsize=14)
plt.legend(loc='best')
plt.grid()
```

```
[**********************100%**********************]  1 of 1 completed
```



[21]:
```
# MDD and Maximum Profit
max_profit_LS = np.max(da)
max_prift_OL = np.max(da1)
MDD_LS = np.min(da)
MDD_OL = np.min(da1)

# Create data
data = [["Only Long", round(MDD_OL,2), round(max_prift_OL,2)],
["Long Short", round(MDD_LS,2), round(max_profit_LS,2)]]

# Define header names
col_names = ["Strategie", "MDD", "Max. Profit"]

# Display table
print('MDD and MAX.Profti of Cross Signal')
print(tabulate(data, headers=col_names))
```

```
MDD and MAX.Profti of Cross Signal
Strategie      MDD    Max. Profit
-----------  -----  -------------
Only Long    -0.11           0.17
```

```
Long Short   -0.17           0.8
```

[ ]:

Random_Forest

June 4, 2022

1 Random Forest

1.0.1 Imports

[1]:
```
# Basic Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pprint import pprint
import seaborn as sns
from matplotlib import pyplot
import yfinance as yf

# Machine Learning Imports
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import plot_roc_curve, roc_auc_score
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import plot_confusion_matrix
from sklearn.preprocessing import MinMaxScaler
from sklearn.inspection import permutation_importance
from tabulate import tabulate
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
```

2 Load CSV File

[2]:
```
# From 16.08.2007 to 26.04.2022
bco = pd.read_csv('/Users/romaindeleze/Desktop/BSc/Romain_WI_TZ/8.Semester/BA/
 ↪Daten/Brent_Crude_Oil')
```

[3]:
```
print(bco.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3641 entries, 0 to 3640
```

```
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Date         3641 non-null   object
 1   Close        3641 non-null   float64
 2   Log_Returns  3641 non-null   float64
 3   Pred_Signal  3641 non-null   int64
 4   RSI          3641 non-null   float64
 5   K_percent    3641 non-null   float64
 6   MACD         3641 non-null   float64
 7   ROC          3641 non-null   float64
dtypes: float64(6), int64(1), object(1)
memory usage: 227.7+ KB
None
```

3 Building the Random Forest Model

We have split our data into a training set and testing set, so we need to identify our input columns which are the following:

RSI, Stochastic Oscillator (K_percent), Price Rate of Change (ROC), MACD

Those columns will serve as our X , and our Y column will be the Signal column, the column that specifies whether the future closed up or down compared to the previous day.

Once we've selected our columns, we need to split the data into a training and test set. Which will take our features and labels and split them based on the size we input. In our case, let's have the test_size be '20 % . After we've split the data, we can create our RandomForestClassifier model. Once we've created it, we can fit the training data to the model using the fit method. Finally, with our "trained" model, we can make predictions. Take the X_test data set and use it to make predictions.

[4]:
```
# Choose the features we want in the Random Forest model
features = bco[['RSI','K_percent','ROC','MACD']]
labels = bco['Pred_Signal']

# Split the data in a 80%, 20% proportion
X_train, X_test, y_train, y_test = train_test_split(features,labels,␣
 ↪test_size=0.2, shuffle=False)

# Print lengths to see if its splitted right
print('Features lenght = ',len(features))
print('Label lenght = ',len(labels))
print()
print('X_train lenght = ',len(X_train))
print('y_train lenght = ',len(y_train))
```

```
print('X_test lenght = ',len(X_test))
print('y_test lenght = ',len(y_test))
print('Backtesting: ',len(X_train)+len(X_test))
print('Backtesting: ',len(y_train)+len(y_test))
```

```
Features lenght =  3641
Label lenght =  3641

X_train lenght =  2912
y_train lenght =  2912
X_test lenght =  729
y_test lenght =  729
Backtesting:  3641
Backtesting:  3641
```

4 Hyperparameter Tuning

To do the hyperparameter tuning we need to optimize the tuning parameters.

The optimisation follows via the RandomizedSearchCV command and then we fit the optimisation model with the training data.

[5]:
```
# Create a empty model
rf = RandomForestClassifier(random_state = 8)

# Look at parameters used by our current RandomForestClassifier
print('Parameters currently in use:\n')
pprint(rf.get_params())
```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 8,
```

```
 'verbose': 0,
 'warm_start': False}
```

4.0.1 With the RandomizedSearchCV command we can achive the hyperparameter tuning

We have to choose which hyperparameter we want tune. Then we have to creat lists with the parameter values for which we are looking for. The RandomizedSearchCV command will compute different combinations and then we can access to the best parameter values.

[6]:
```
# Number of trees in random forest
n_estimators = list(range(1,501,1))

# Number of features to consider at every split
max_features = ['sqrt', 'log2']

# Maximum number of levels in tree
max_depth = list(range(1,51))

# Minimum number of samples required to split a node
min_samples_split = list(range(1,51))

# Minimum number of samples required at each leaf node
min_samples_leaf = list(range(1,51))

# Criterion
criterion = ['gini', 'entropy']

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'criterion': criterion}
```

[7]:
```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestClassifier()

# Random search of parameters, using 3 fold cross validation,
# search across 45 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions =␣
 ↪random_grid, n_iter = 15,
                               cv = 3, verbose=1, random_state=8, n_jobs = 1,␣
 ↪scoring = 'r2')
```

```
# Fit the random search model
rf_random.fit(X_train, y_train) # The optimisation may take a few minutes
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
```

[7]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_iter=15,
                   n_jobs=1,
                   param_distributions={'criterion': ['gini', 'entropy'],
                                        'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9,
                                                      10, 11, 12, 13, 14, 15,
                                                      16, 17, 18, 19, 20, 21,
                                                      22, 23, 24, 25, 26, 27,
                                                      28, 29, 30, …],
                                        'max_features': ['sqrt', 'log2'],
                                        'min_samples_leaf': [1, 2, 3, 4, 5, 6,
                                                             7, 8, 9, 10, 11,
                                                             12, 13, 14, 15, 16,
                                                             17, 18, 19, 20, 21,
                                                             22, 23, 24, 25, 26,
                                                             27, 28, 29, 30,
…],
                                        'min_samples_split': [1, 2, 3, 4, 5, 6,
                                                              7, 8, 9, 10, 11,
                                                              12, 13, 14, 15,
                                                              16, 17, 18, 19,
                                                              20, 21, 22, 23,
                                                              24, 25, 26, 27,
                                                              28, 29, 30, …],
                                        'n_estimators': [1, 2, 3, 4, 5, 6, 7, 8,
                                                         9, 10, 11, 12, 13, 14,
                                                         15, 16, 17, 18, 19, 20,
                                                         21, 22, 23, 24, 25, 26,
                                                         27, 28, 29, 30, …]},
                   random_state=8, scoring='r2', verbose=1)
```

[8]:
```
# This is the Output command to see the best Parameters
param = rf_random.best_params_
print(param)
```

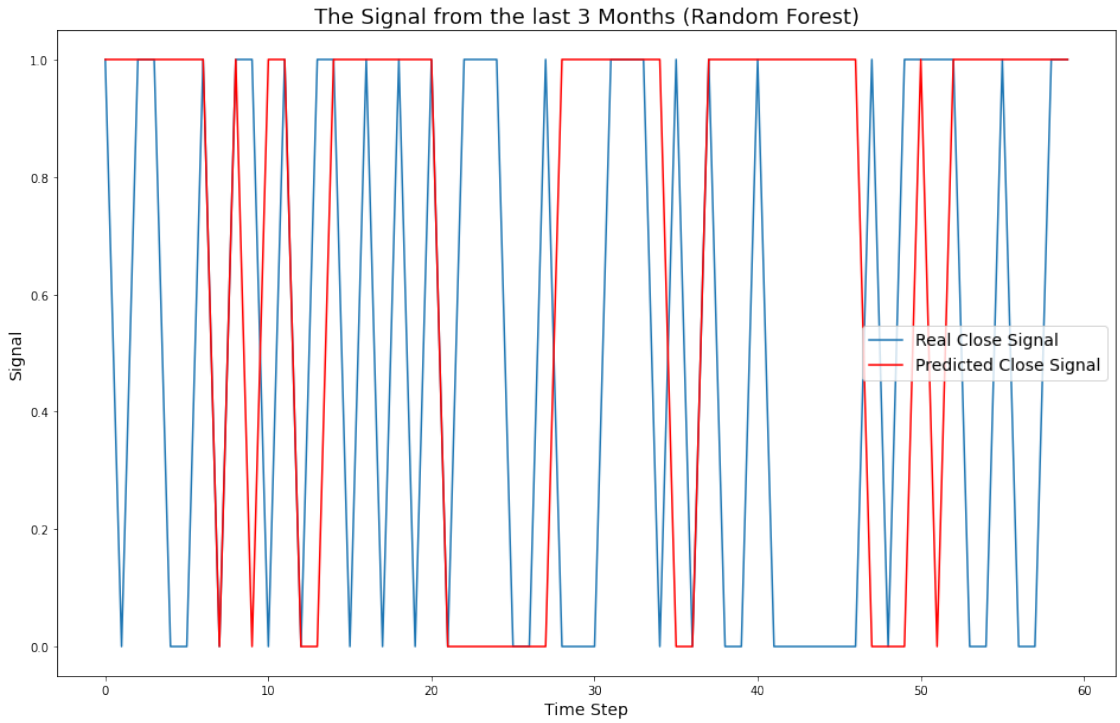
```
{'n_estimators': 46, 'min_samples_split': 39, 'min_samples_leaf': 10,
'max_features': 'sqrt', 'max_depth': 36, 'criterion': 'entropy'}
```

4.1 After the optimisation we can create the random forest model and make the predictions

[9]:
```
# Create a Random Forest Classifier with the Hyperparameters from the Tuning
rand_frst_clf = RandomForestClassifier(oob_score = True,
                                       n_estimators = param['n_estimators'],
                                       min_samples_split =␣
 ↪param['min_samples_split'],
                                       min_samples_leaf =␣
 ↪param['min_samples_leaf'],
                                       max_features = param['max_features'],
                                       max_depth = param['max_depth'],
                                       criterion = param['criterion'],
                                       bootstrap = True,
                                       random_state = 8)

# Fit the data to the model
rand_frst_clf.fit(X_train, y_train)

# Make predictions
y_pred = rand_frst_clf.predict(X_test)
```

[10]:
```
# Lets see the actual and the predicted Signal from the last 3 months
m3 = 3*4*5
plt.figure(figsize=(16,10))
plt.title('The Signal from the last 3 Months (Random Forest)', fontsize=18)
plt.plot(y_test[len(y_test)-m3:].values, label = 'Real Close Signal')
plt.plot(y_pred[len(y_pred)-m3:], color = 'red', label = 'Predicted Close␣
 ↪Signal')
plt.ylabel('Signal', fontsize=14)
plt.xlabel('Time Step', fontsize=14)
plt.legend(loc='best', fontsize=14)
```

[10]:
```
<matplotlib.legend.Legend at 0x7f9ef113d820>
```



5 Accuracy

We've built our model, so now we can see how accurate it is. SciKit learn, makes the process of evaluating our model very easy by providing a bunch of built-in metrics that we can call. One of those metrics is the accuracy_score. The accuracy_score function computes the accuracy, by calculating the sum of the correctly predicted signals and then dividing it by the total number of predictions. Imagine we had three TRUE values [1, 2, 3] , and our model predicted the following values [1, 2, 4] we would say the accuracy of our model is $2/3 = 66\ \%$.

6 AUC

AUC curve is a performance measurement for the classification problems at various threshold settings. AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1. By analogy, the Higher the AUC, the better the model is at distinguishing between patients with the disease and no disease.

7 RMSE

Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are;

RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit.

[11]:
```
# Print the Accuracy of our Model.
print('Accuracy % : ', accuracy_score(y_test, y_pred, normalize = True) * 100.0)
print()

# Checking performance our model with AUC Score.
print('Area under the curve: ',roc_auc_score(y_test, y_pred))
print()

# Get the root mean squared error (RMSE)
print('RMSE = ' , np.sqrt(np.mean(((y_pred - y_test) ** 2))))
# Don't forget the worst value in this case is 1!
```

```
Accuracy % :  50.06858710562414

Area under the curve:  0.4962001917020402

RMSE =  0.706621630679219
```

8 Classification Report

To get a more detailed overview of how the model performed, we can build a classification report that will compute the F1_Score , the Precision , the Recall , and the Support .

8.0.1 Precision

Precision measures the proportion of all correctly identified samples in a population of samples which are classified as positive labels and is defined as the following:

tp = True Positiv

fp = False Positiv

Precision = tp/(tp+fp)

The precision is intuitively the ability of the classifier not to label as positive a sample that is negative. The best value is 1, and the worst value is 0.

8.0.2 Recall

Recall (also known as sensitivity) measures the ability of a classifier to correctly identify positive labels and is defined as the following:

tp = True Positiv

fn = False Negativ

Recall = tp/(tp+fn)

The recall is intuitively the ability of the classifier to find all the positive samples. The best value is 1, and the worst value is 0.

8.0.3 Support

Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing. Support doesn't change between models but instead diagnoses the evaluation process.

8.0.4 F1 Score

In some cases, we will have models that may have low precision or high recall. It's difficult to compare two models with low precision and high recall or vice versa. To make results comparable, we use a metric called the F-Score. The F-score helps to measure Recall and Precision at the same time. It uses Harmonic Mean in place of Arithmetic Mean by punishing the extreme values more.

The traditional F-measure or balanced F-score (F1 score) is the harmonic mean of precision and recall:

tp = True Positiv

fp = False Positiv

fn = False Negativ

F1 Score = tp/(tp+0.5*(fp+fn))

[12]:
```
# Define the traget names
target_names = ['Down Day', 'Up Day']

# Build a classifcation report
report = classification_report(y_true = y_test, y_pred = y_pred, target_names =␣
↪target_names, output_dict = True)

# Add it to a data frame, transpose it for readability.
report_df = pd.DataFrame(report).transpose()
report_df
```

[12]:

	precision	recall	f1-score	support
Down Day	0.444444	0.452599	0.448485	327.000000
Up Day	0.547980	0.539801	0.543860	402.000000

```
accuracy       0.500686  0.500686  0.500686    0.500686
macro avg      0.496212  0.496200  0.496172  729.000000
weighted avg   0.501538  0.500686  0.501078  729.000000
```

9 Confusion Matrix

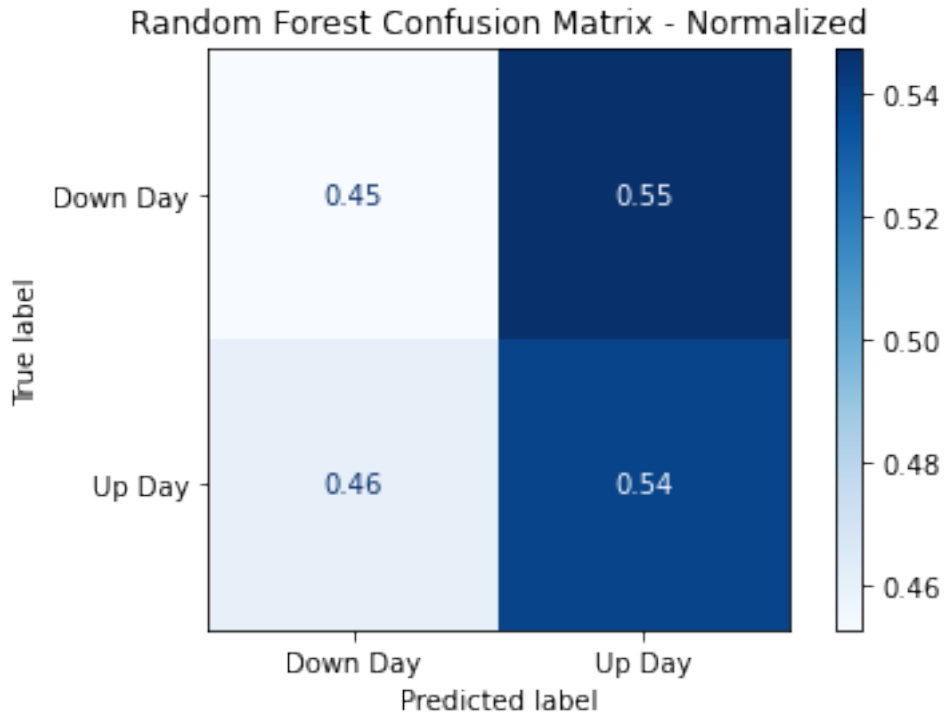
A confusion matrix is a technique for summarizing the performance of a classification algorithm. Classification accuracy alone can be misleading if you have an unequal number of observations in each class or if you have more than two classes in your dataset. Calculating a confusion matrix can give you a better idea of what your classification model is getting right and what types of errors it is making.

[13]:
```
# Compute Confusion Matrix
rf_matrix = confusion_matrix(y_test, y_pred)

# Save individual values
true_negatives = rf_matrix[0][0]
false_negatives = rf_matrix[1][0]
true_positives = rf_matrix[1][1]
false_positives = rf_matrix[0][1]

# Compute performance values
accuracy = (true_negatives + true_positives) / (true_negatives + true_positives␣
↪+ false_negatives + false_positives)
percision = true_positives / (true_positives + false_positives)
recall = true_positives / (true_positives + false_negatives)
specificity = true_negatives / (true_negatives + false_positives)

# Print values
print('Accuracy: {}'.format(float(accuracy)))
print('Percision: {}'.format(float(percision)))
print('Recall: {}'.format(float(recall)))
print('Specificity: {}'.format(float(specificity)))

# Visualise the Confusion Matrix
disp = plot_confusion_matrix(rand_frst_clf, X_test, y_test, display_labels =␣
↪['Down Day', 'Up Day']
                             , normalize = 'true', cmap=plt.cm.Blues)
disp.ax_.set_title('Random Forest Confusion Matrix - Normalized')
plt.show()
```

```
Accuracy: 0.5006858710562414
Percision: 0.547979797979798
Recall: 0.5398009950248757
Specificity: 0.4525993883792049
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function plot_confusion_matrix is deprecated; Function
```

```
`plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one
of the class methods: ConfusionMatrixDisplay.from_predictions or
ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



9.0.1 With the information from the confusion matrix we want know now how the proportion is.

How is the prediction distributed, and how ist the real values distributed

[14]:
```
# Propotion of the real values
unique, counts = np.unique(y_test, return_counts=True)
print('Real Proportion')
print(dict(zip(unique, counts/len(y_test))))
print()

# Propotion of the predicted values
unique, counts = np.unique(y_pred, return_counts=True)
print('Predicted Proportion')
print(dict(zip(unique, counts/len(y_pred))))
```

```
Real Proportion
{0: 0.448559670781893, 1: 0.551440329218107}

Predicted Proportion
```

```
{0: 0.4567901234567901, 1: 0.5432098765432098}
```

10 Feature Importance

With any model, you want to have an idea of what features are helping explain most of the model, as this can give you insight as to why you're getting the results you are. With Random Forest, we can identify some of our most important features or, in other words, the features that help explain most of the model. In some cases, some of our features might not be very important, or in other words, when compared to additional features, don't explain much of the model.

10.0.1 Why Do We Care About Feature Importance?

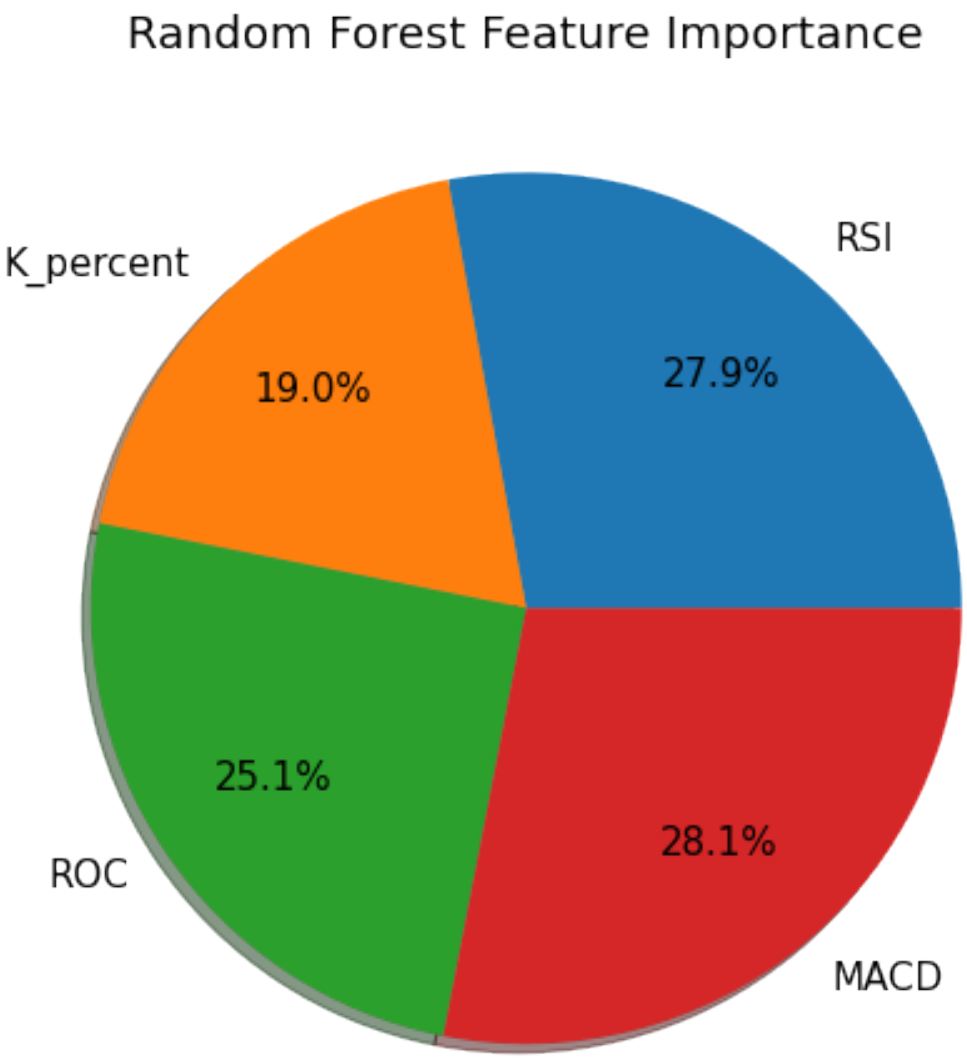
What that means is if we were to get rid of those features, our accuracy will go down a little, hopefully, but not significantly. You might be asking, "Why would I want to get rid of a feature if it lowers my accuracy?" Well, it depends, in some cases, you don't care if your model is 95% accurate or 92% accurate. To you, a 92% accurate model is just as good as a 95% accurate model. However, if you wanted to get a 95% accurate model, you would, in this hypothetical case, have to train your model twice as long. Now, I'm a little extreme in this case, but the idea is the same. The cost doesn't justify the benefit. In the real world, we have to make these decisions all the time, and in some cases, it just doesn't warrant the extra cost for such a minimal increase in the accuracy.

10.0.2 Calculating the Feature Importance

Like all the previous steps, SkLearn makes this process very easy. Take your ML model and call the permutation_importance. This will return all of our features and their importance measurement.

[15]:
```
# Perform permutation importance
results = permutation_importance(rand_frst_clf, X_train, y_train)

# Get importance for accuracy
importance = results.importances_mean

# Create Pie Plot
imp_scale = sum(importance)
imp = importance/imp_scale
def func(pct, allvalues):
    return "{:.1f}%".format(pct)
plt.figure(figsize=(8,8))
plt.title('Random Forest Feature Importance', fontsize = 18)
plt.pie(imp, labels=['RSI','K_percent','ROC','MACD'], autopct = lambda pct:␣
 ↪func(pct, imp),
        shadow=True, pctdistance=.7, textprops={'fontsize': 15})
```

[15]: ([<matplotlib.patches.Wedge at 0x7f9ef0db4a90>,
<matplotlib.patches.Wedge at 0x7f9ef0dc3400>,
<matplotlib.patches.Wedge at 0x7f9ef0dc3cd0>,
<matplotlib.patches.Wedge at 0x7f9ef0dd15e0>],
[Text(0.7043645240172034, 0.8449086443551271, 'RSI'),
Text(-0.7718361188936859, 0.7837531534680625, 'K_percent'),
Text(-0.9132208854812488, -0.6132109052527068, 'ROC'),
Text(0.6997307277377152, -0.8487502039231848, 'MACD')],
[Text(0.4482319698291294, 0.537669137316899, '27.9%'),
Text(-0.49116843929598186, 0.49875200675240333, '19.0%'),
Text(-0.5811405634880673, -0.3902251215244497, '25.1%'),
Text(0.4452831903785459, -0.5401137661329357, '28.1%')])

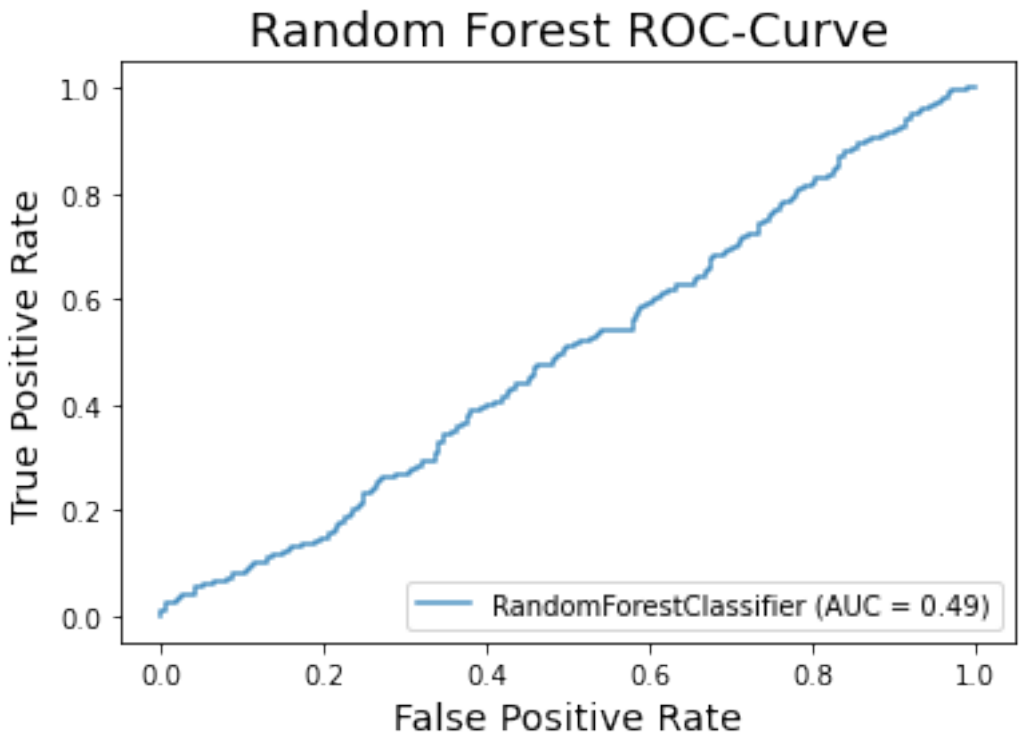


11 ROC Curve

The Receiver Operating Characteristic is a graphical method to evaluate the performance of a binary classifier. A curve is drawn by plotting True Positive Rate (sensitivity) against False Positive Rate (1 - specificity) at various threshold values. ROC curve shows the trade-off between sensitivity and specificity. When the curve comes closer to the left-hand border and the top border of the ROC space, it indicates that the test is accurate. The closer the curve is to the top and left-hand border, the more accurate the test is. If the curve is close to the 45 degrees diagonal of the ROC space, it means that the test is not accurate. ROC curves can be used to select the optimal model and discard the suboptimal ones.

[16]:
```
# Create an ROC Curve plot.
rfc_disp = plot_roc_curve(rand_frst_clf, X_test, y_test, alpha = 0.8)
plt.title('Random Forest ROC-Curve', fontsize=18)
plt.xlabel('False Positive Rate', fontsize=14)
plt.ylabel('True Positive Rate', fontsize=14)
plt.show()
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function plot_roc_curve is deprecated; Function
:func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one
of the class methods: :meth:`sklearn.metric.RocCurveDisplay.from_predictions` or
:meth:`sklearn.metric.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```



12 Tranding Performance

[17]:
```
# Define the length of the prediction
n = len(y_pred)
X = bco['Log_Returns'].shift(-1)
X = X[len(X)-n:]
len(X)
```

[17]: 729

12.1 Only Long

[18]:
```
# Compute the Only Long Performance: Signal * Log Return
perf = y_pred * X
perf = perf.dropna()

trading_days_Y = 5*52

perf_sharpe = np.sqrt(trading_days_Y) * np.mean(perf) / (np.sqrt(np.var(perf)))
```

12.2 Long-Short

[19]:
```
# Transform 0,1 Signal in a -1,1 Signal
y_pred1 = []
for i in range(0,len(y_pred)):
    if y_pred[i] == 0:
        y_pred1.append(-1)
    else:
        y_pred1.append(1)
```

[20]:
```
# Compute the Long Short Performance: Signal * Log Returns
perf1 = y_pred1 * X
perf1 = perf1.dropna()

trading_days_Y = 5*52

perf_sharpe1 = np.sqrt(trading_days_Y) * np.mean(perf1) / (np.sqrt(np.
 ↪var(perf1)))
```

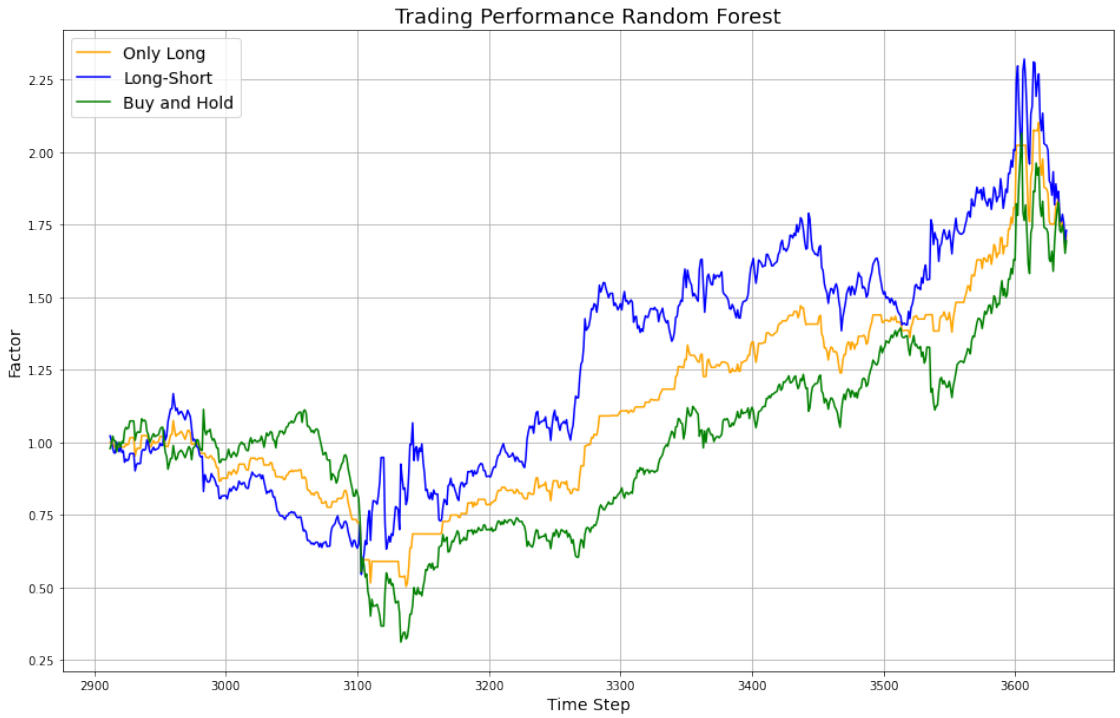
12.3 Buy and Hold

[21]:
```
perf2 = X

trading_days_Y = 5*52

perf_sharpe2 = np.sqrt(trading_days_Y) * np.mean(perf2) / (np.sqrt(np.
 ↪var(perf2)))
```

13 Comparison ML vs. Buy and Hold

[22]:
```
# Lets compare the performances
plt.figure(figsize=(16,10))
plt.title('Trading Performance Random Forest', fontsize=18)
plt.ylabel('Factor', fontsize=14)
plt.xlabel('Time Step', fontsize=14)
plt.grid(True)
plt.plot(np.exp(perf).cumprod() , label='Only Long', color = 'orange')
plt.plot(np.exp(perf1).cumprod() , label='Long-Short', color = 'blue')
plt.plot(np.exp(perf2).cumprod() , label='Buy and Hold', color = 'green')
plt.legend(loc='best', fontsize=14)
```

[22]:
```
<matplotlib.legend.Legend at 0x7f9ef0d866d0>
```



14 Profit Factor

[23]:
```
# Profit Factor is the last value of the performance computation
L_profit = np.exp(perf).cumprod().iloc[-1]

LS_profit = np.exp(perf1).cumprod().iloc[-1]

BaH_profit = np.exp(perf2).cumprod().iloc[-2]
```

15 Performance Table

[24]:
```
# Create data
data = [["Buy and Hold", round(perf_sharpe2,2), round(BaH_profit,2)],
        ["Only Long", round(perf_sharpe,2), round(L_profit,2)],
        ["Long Short", round(perf_sharpe1,2), round(LS_profit,2)]]

# Define header names
col_names = ["Strategie", "Sharp Ratio", "Profit Factor"]

# Display table
print('Random Forest Classification')
print(tabulate(data, headers=col_names))
```

```
Random Forest Classification
Strategie       Sharp Ratio    Profit Factor
------------  -------------  ---------------
Buy and Hold           0.35             1.69
Only Long              0.55             1.71
Long Short             0.36             1.73
```

[25]:
```
# Performance Dataset
# Compute performance
RF_Perf_OL = np.exp(perf).cumprod()
RF_Perf_LS = np.exp(perf1).cumprod()

# Load in a Data Frame
RF_Perf_data = pd.DataFrame()
RF_Perf_data['RF_Perf_OL'] = RF_Perf_OL
RF_Perf_data['RF_Perf_LS'] = RF_Perf_LS

# Create a csv File
RF_Perf_data.to_csv('/Users/romaindeleze/Desktop/BSc/Romain_WI_TZ/8.Semester/BA/
 ↪Daten/RF_Perf_data')
```

16 Extrem and Regular Value Accuracy

[26]:
```
# Create Data Frame with Predicted Signal and Log Returns
n = len(y_pred)
X = bco['Log_Returns'].shift(-1)
X = X[len(X)-n:]
d = {'Predicted Signal':y_pred, 'Log Returns':X}
df = pd.DataFrame(data = d)
```

[27]:
```
# Create the upperbound and the lowerbound
mu = np.mean(X)
sig = np.std(X)
lb = mu-2*sig
ub = mu+2*sig

# Creat Data Frame with only Values in the interval [lb,ub]
reg_val = df[(df['Log Returns'] >= lb) & (df['Log Returns'] <= ub)] # DataFrame␣
 ↪with Regular Values = [lb,ub]

# Creat Data Frame with only Values outside the interval [lb,ub]
ext_val = df.drop(index=reg_val.index)

# Add real Signal with the Log Returns for each Data Frame
reg_val['Real Signal'] = (reg_val['Log Returns'] > 0).astype(int)
ext_val['Real Signal'] = (ext_val['Log Returns'] > 0).astype(int)
```

```
# Compute accuracy of the Extrem Values and the Regular Values
acc_reg = accuracy_score(reg_val['Real Signal'], reg_val['Predicted Signal'],␣
 ↪normalize = True) * 100.0
acc_ext = accuracy_score(ext_val['Real Signal'], ext_val['Predicted Signal'],␣
 ↪normalize = True) * 100.0
print('Accuracy Regular Values = ', round(acc_reg,2))
print('Accuracy Exrtem Values (95%) = ', round(acc_ext,2))
```

```
Accuracy Regular Values =  49.21
Accuracy Exrtem Values (95%) =  63.89
```

```
<ipython-input-27-19962f7716fb>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  reg_val['Real Signal'] = (reg_val['Log Returns'] > 0).astype(int)
```

[28]:
```
# Create the upperbound and the lowerbound
mu = np.mean(X)
sig = np.std(X)
lb1 = mu-3*sig
ub1 = mu+3*sig

# Create Data Frame with only Values in the interval [lb,ub]
reg_val = df[(df['Log Returns'] >= lb1) & (df['Log Returns'] <= ub1)] #␣
 ↪DataFrame with Regular Values = [lb,ub]

# Create Data Frame with only Values outside the interval [lb,ub]
ext_val = df.drop(index=reg_val.index)

# Add real Signal with the Log Returns for each Data Frame
reg_val['Real Signal'] = (reg_val['Log Returns'] > 0).astype(int)
ext_val['Real Signal'] = (ext_val['Log Returns'] > 0).astype(int)

# Compute accuracy of the Extrem Values and the Regular Values
acc_reg = accuracy_score(reg_val['Real Signal'], reg_val['Predicted Signal'],␣
 ↪normalize = True) * 100.0
acc_ext = accuracy_score(ext_val['Real Signal'], ext_val['Predicted Signal'],␣
 ↪normalize = True) * 100.0
print('Accuracy Regular Values = ', round(acc_reg,2))
print('Accuracy Exrtem Values (99%) = ', round(acc_ext,2))
```

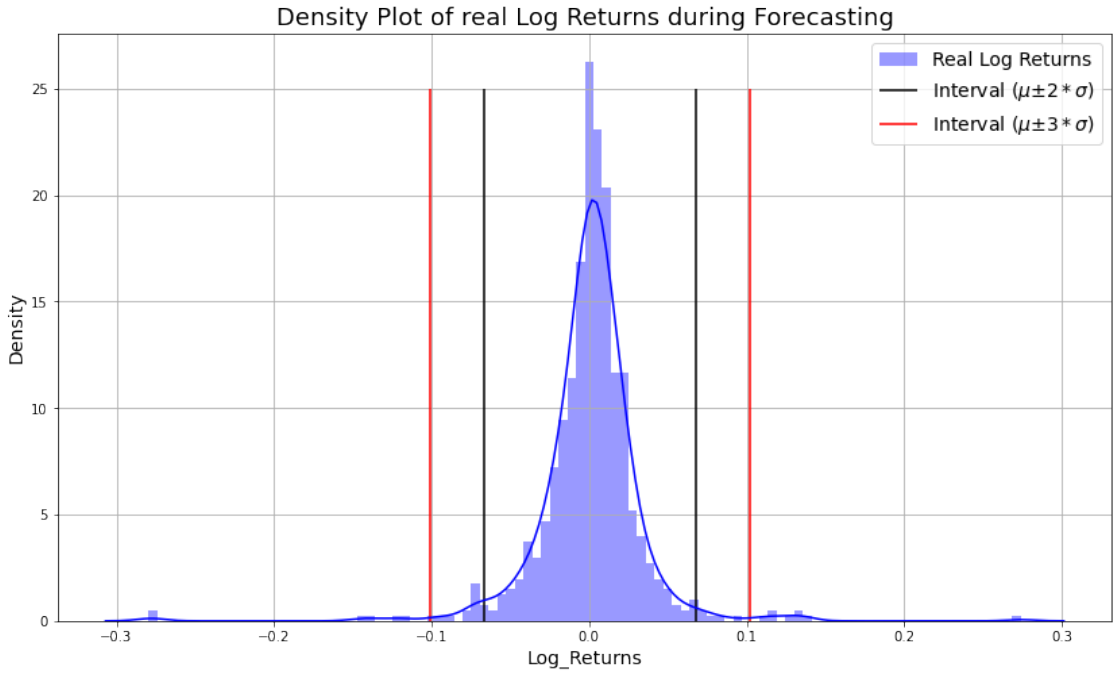
```
Accuracy Regular Values =  50.0
Accuracy Exrtem Values (99%) =  46.67
```

```
<ipython-input-28-6ea3d8971ebc>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  reg_val['Real Signal'] = (reg_val['Log Returns'] > 0).astype(int)
```

[29]:
```
# Lets see how the log returns are distributed during the testphase
plt.figure(figsize=(14,8))
plt.title('Density Plot of real Log Returns during Forecasting', fontsize = 18)
plt.ylabel('Density', fontsize = 14)
plt.xlabel('Log Returns', fontsize = 14)
plt.grid(True)
sns.distplot(X, bins = 100, color = 'blue', label = 'Real Log Returns')
plt.vlines(x=lb, ymin=0,ymax=25, color = 'black', label = 'Interval ($ ± 2* $)')
plt.vlines(x=ub, ymin=0,ymax=25, color = 'black')
plt.vlines(x=lb1, ymin=0,ymax=25, color = 'red', label = 'Interval ($ ± 3* $)')
plt.vlines(x=ub1, ymin=0,ymax=25, color = 'red')
plt.legend(loc='best', fontsize = 14)
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

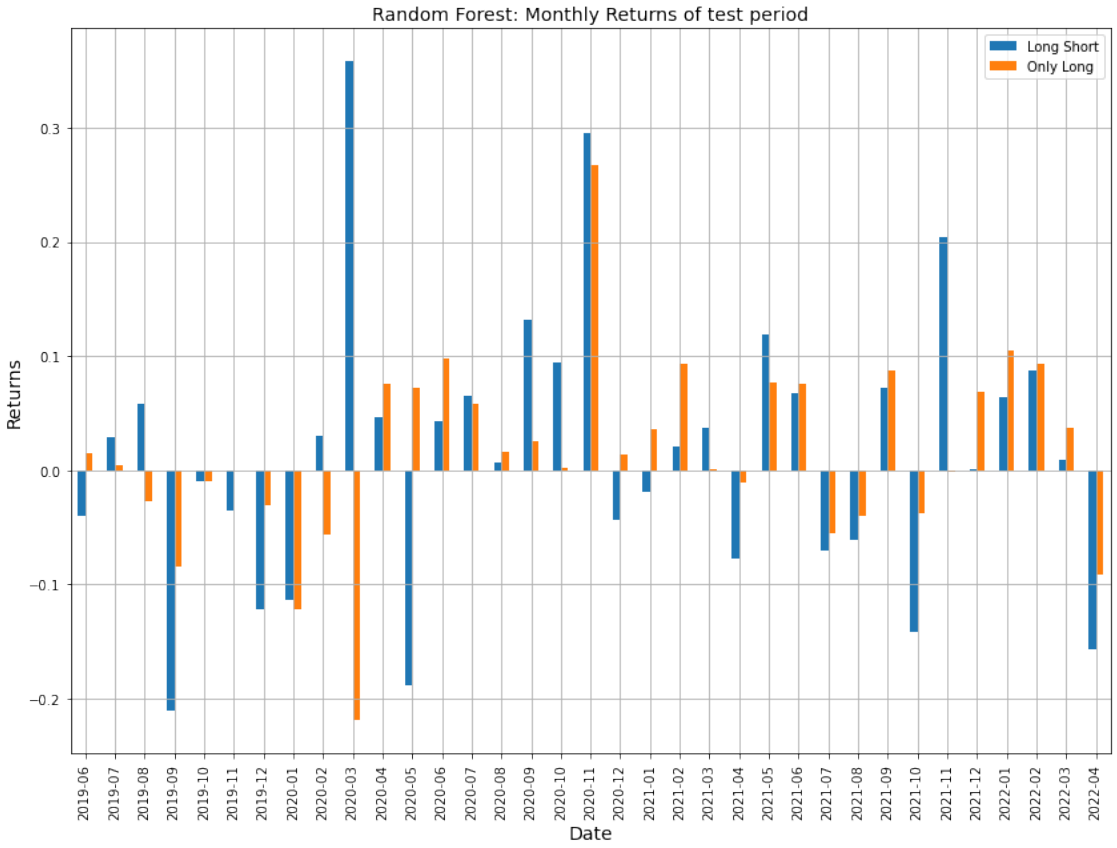
[29]:
```
<matplotlib.legend.Legend at 0x7f9ef0905700>
```



[30]:
```
# Indexing
dat = yf.download('BZ=F', start='2007-08-16',end = "2022-04-26",period='1d')
end_len = len(dat["Close"])
long_short = np.exp(perf1).cumprod()
start_len = end_len - len(long_short)
index = dat["Close"][start_len:end_len].index
perf1.index = index
perf.index = index

# Create Monthly Returns Long Short
da = perf1.groupby([(perf1.index.year),(perf1.index.month)]).sum()
monthly_ret_LS = da.values
n = len(monthly_ret_LS)
monthly_ret_LS = pd.DataFrame(monthly_ret_LS,index = pd.date_range(start='6/1/
↪2019', freq='M', periods=n))

# Create Monthly Returns Long
da1 = perf.groupby([(perf.index.year),(perf.index.month)]).sum()
monthly_ret_OL = da1.values
monthly_ret_OL = pd.DataFrame(monthly_ret_OL,index = pd.date_range(start='6/1/
↪2019', freq='M', periods=n))

# Connect DF
mon_ret = pd.concat([monthly_ret_LS, monthly_ret_OL], axis=1)
mon_ret.columns = ["Long Short","Only Long"]
```

```
# x Axis
x_ax = pd.period_range('6/1/2019', '2022-04-30', freq='M').strftime('%Y-%m')
mon_ret.index = x_ax

# Plot the barplot
ax = mon_ret.plot.bar(figsize=(14,10), rot = 0)
plt.title("Random Forest: Monthly Returns of test period",fontsize=14)
plt.xlabel('Date', fontsize=14)
plt.xticks(rotation=90)
plt.ylabel('Returns', fontsize=14)
plt.legend(loc='best')
plt.grid()
```

```
[**********************100%**********************]  1 of 1 completed
```



[31]:
```
# Lets see which are the MDD and Maximum Profit grouped by months
max_profit_LS = np.max(da)
max_prift_OL = np.max(da1)
MDD_LS = np.min(da)
```

```
MDD_OL = np.min(da1)

# Create data
data = [["Only Long", round(MDD_OL,2), round(max_prift_OL,2)],
["Long Short", round(MDD_LS,2), round(max_profit_LS,2)]]

# Define header names
col_names = ["Strategie", "MDD", "Max. Profit"]

# Display table
print('MDD and MAX.Profti of RF')
print(tabulate(data, headers=col_names))
```

```
MDD and MAX.Profti of RF
Strategie      MDD    Max. Profit
-----------  -----  -------------
Only Long    -0.22           0.27
Long Short   -0.21           0.36
```

17 Time Series Cross Validation Backwards

[32]:
```
#-------------------------------------------------------------------------#
#                               Full Dataset                              |
#-------------------------------------------------------------------------#
# t_0 <-----------------------------------------------------------> t_n #

# K = 1
#-------------------------------------------------------------------------#
#              |               |              |    Train      |     Test     |
#-------------------------------------------------------------------------#

# K = 2
#-------------------------------------------------------------------------#
#              |               |     Train    |    Train      |     Test     |
#-------------------------------------------------------------------------#

# K = 3
#-------------------------------------------------------------------------#
#              |     Train     |     Train    |    Train      |     Test     |
#-------------------------------------------------------------------------#

# K = 4
#-------------------------------------------------------------------------#
#     Train    |     Train     |     Train    |    Train      |     Test     |
#-------------------------------------------------------------------------#

# Create two empty lists
```

```
L_Sharpe_Ratios = []
LS_Sharpe_Ratios = []

# For loop who create model, fit model, test model and compute performance and␣
 ↪Sharpe Ratio
for k in range (1,5):

    # Choose of the features we want in the Random Forest model
    features = bco[['RSI','K_percent','ROC','MACD']]
    labels = bco['Pred_Signal']

    X_train, X_test, y_train, y_test = train_test_split(features,labels,␣
 ↪test_size=0.2, shuffle=False)

    for i in range(1,2):
        feature_len = int(round(len(X_train))*0.25*k)
        X_train = X_train[-feature_len:]
        label_len = int(round(len(y_train))*0.25*k)
        y_train = y_train[-label_len:]

        # Create model with hyperparameter tuning
        rf_random.fit(X_train, y_train)
        param = rf_random.best_params_
        rand_frst_clf = RandomForestClassifier(oob_score = True,
                                               n_estimators =␣
 ↪param['n_estimators'],
                                               min_samples_split =␣
 ↪param['min_samples_split'],
                                               min_samples_leaf =␣
 ↪param['min_samples_leaf'],
                                               max_features =␣
 ↪param['max_features'],
                                               max_depth = param['max_depth'],
                                               criterion = param['criterion'],
                                               bootstrap = True,
                                               random_state = 8)

        # Fit the data to the model
        rand_frst_clf.fit(X_train, y_train)

        # Make predictions
        y_pred = rand_frst_clf.predict(X_test)

        # Trading Performance
        n = len(y_pred)
        X = bco['Log_Returns'].shift(-1)
```

```
        X = X[len(X)-n:]

        trading_days_Y = 5*52

        # Only Long
        perf = y_pred * X
        perf = perf.dropna()
        # Sharpe
        perf_sharpe = np.sqrt(trading_days_Y) * np.mean(perf) / (np.sqrt(np.
↪var(perf)))
        L_Sharpe_Ratios.append(perf_sharpe)

        # Long Short
        # Transform 0,1 Signal in a -1,1 Signal
        y_pred1 = []
        for i in range(0,len(y_pred)):
            if y_pred[i] == 0:
                y_pred1.append(-1)
            else:
                y_pred1.append(1)

        perf1 = y_pred1 * X
        perf1 = perf1.dropna()
        # Sharpe
        perf_sharpe1 = np.sqrt(trading_days_Y) * np.mean(perf1) / (np.sqrt(np.
↪var(perf1)))
        LS_Sharpe_Ratios.append(perf_sharpe1)
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
Fitting 3 folds for each of 15 candidates, totalling 45 fits
Fitting 3 folds for each of 15 candidates, totalling 45 fits
Fitting 3 folds for each of 15 candidates, totalling 45 fits
```

[33]:
```
# Create data
data = [["Only Long", round(L_Sharpe_Ratios[0],2), round(L_Sharpe_Ratios[1],2),
         round(L_Sharpe_Ratios[2],2), round(L_Sharpe_Ratios[3],2)],
        ["Long Short", round(LS_Sharpe_Ratios[0],2),␣
↪round(LS_Sharpe_Ratios[1],2),
         round(LS_Sharpe_Ratios[2],2), round(LS_Sharpe_Ratios[3],2)]]

# Define header names
col_names = ["K = 1", "K = 2", "K = 3", "K = 4"]

# Display table
print('Random Forest')
print('Cross Validation with Time Series Split Backward')
print('Sharpe Ratio')
```

```
print(tabulate(data, headers=col_names))
```

```
Random Forest
Cross Validation with Time Series Split Backward
Sharpe Ratio
              K = 1    K = 2    K = 3    K = 4
----------  -------  -------  -------  -------
Only Long      0.29     0.17     0.28     0.38
Long Short     0.17    -0.2     -0.05     0.2
```

[34]:
```
# Print all results
print('Mean Only Long Sharpe Ratio = ',round(np.mean(L_Sharpe_Ratios),2))
print('Mean Long Short Sharpe Ratio = ',round(np.mean(LS_Sharpe_Ratios),2))
```

```
Mean Only Long Sharpe Ratio =  0.28
Mean Long Short Sharpe Ratio =  0.03
```

18 Time Series Cross Validation Forwards

[35]:
```
#-------------------------------------------------------------------------#
#                              Full Dataset                               |
#-------------------------------------------------------------------------#
# t_0 <-----------------------------------------------------------> t_n #

# K = 1
#-------------------------------------------------------------------------#
#     Train    |     Test      |               |               |          |
#-------------------------------------------------------------------------#

# K = 2
#-------------------------------------------------------------------------#
#     Train    |     Train     |     Test      |               |          |
#-------------------------------------------------------------------------#

# K = 3
#-------------------------------------------------------------------------#
#     Train    |     Train     |     Train     |    Test       |          |
#-------------------------------------------------------------------------#

# K = 4
#-------------------------------------------------------------------------#
#     Train    |     Train     |     Train     |    Train      |     Test     |
#-------------------------------------------------------------------------#

# Create two empty lists
L_Sharpe_Ratios = []
LS_Sharpe_Ratios = []
```

```
# For loop who create model, fit model, test model and compute performance and␣
 ↪Sharpe Ratio
for k in range (2,6):
    # Choose of the features we want in the Random Forest model
    features = bco[['RSI','K_percent','ROC','MACD']]
    labels = bco['Pred_Signal']

    feature_len = int(round(len(features))*0.2*k)
    label_len = int(round(len(labels))*0.2*k)

    features = features[:feature_len]
    labels = labels[:label_len]

    X_train = features[:-729]
    X_test = features[-729:]
    y_train = labels[:-729]
    y_test = labels[-729:]

    for i in range(1,2):
        rf_random.fit(X_train, y_train)
        param = rf_random.best_params_
        rand_frst_clf = RandomForestClassifier(oob_score = True,
                                               n_estimators =␣
 ↪param['n_estimators'],
                                               min_samples_split =␣
 ↪param['min_samples_split'],
                                               min_samples_leaf =␣
 ↪param['min_samples_leaf'],
                                               max_features =␣
 ↪param['max_features'],
                                               max_depth = param['max_depth'],
                                               criterion = param['criterion'],
                                               bootstrap = True,
                                               random_state = 8)

        # Fit the data to the model
        rand_frst_clf.fit(X_train, y_train)

        # Make predictions
        y_pred = rand_frst_clf.predict(X_test)

        # Trading Performance
        X = bco['Log_Returns'].shift(-1)
        X = X[:label_len]
        X = X[-729:]
```

```
        trading_days_Y = 5*52

        # Only Long
        perf = y_pred * X
        perf = perf.dropna()
        # Sharpe
        perf_sharpe = np.sqrt(trading_days_Y) * np.mean(perf) / (np.sqrt(np.
↪var(perf)))
        L_Sharpe_Ratios.append(perf_sharpe)

        # Long Short
        # Transform 0,1 Signal in a -1,1 Signal
        y_pred1 = []
        for i in range(0,len(y_pred)):
            if y_pred[i] == 0:
                y_pred1.append(-1)
            else:
                y_pred1.append(1)

        perf1 = y_pred1 * X
        perf1 = perf1.dropna()
        # Sharpe
        perf_sharpe1 = np.sqrt(trading_days_Y) * np.mean(perf1) / (np.sqrt(np.
↪var(perf1)))
        LS_Sharpe_Ratios.append(perf_sharpe1)
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
Fitting 3 folds for each of 15 candidates, totalling 45 fits
Fitting 3 folds for each of 15 candidates, totalling 45 fits
Fitting 3 folds for each of 15 candidates, totalling 45 fits
```

[36]:
```
# create data
data = [["Only Long", round(L_Sharpe_Ratios[0],2), round(L_Sharpe_Ratios[1],2),
         round(L_Sharpe_Ratios[2],2), round(L_Sharpe_Ratios[3],2)],
        ["Long Short", round(LS_Sharpe_Ratios[0],2),␣
↪round(LS_Sharpe_Ratios[1],2),
         round(LS_Sharpe_Ratios[2],2), round(LS_Sharpe_Ratios[3],2)]]

#define header names
col_names = ["K = 1", "K = 2", "K = 3", "K = 4"]

#display table
print('Random Forest')
print('Cross Validation Time Series Split Forward')
print('Sharpe Ratio')
print(tabulate(data, headers=col_names))
```

```
Random Forest
Cross Validation Time Series Split Forward
Sharpe Ratio
              K = 1    K = 2    K = 3    K = 4
----------  -------  -------  -------  -------
Only Long       0.2    -0.06     0.38     0.78
Long Short     -0.07    0.68     0.22     0.54
```

[37]:
```
# Print all results
print('Mean Only Long Sharpe Ratio = ',round(np.mean(L_Sharpe_Ratios),2))
print('Mean Long Short Sharpe Ratio = ',round(np.mean(LS_Sharpe_Ratios),2))
```

```
Mean Only Long Sharpe Ratio =  0.32
Mean Long Short Sharpe Ratio =  0.34
```

19 K-Fold Cross Validation

[38]:
```
#-------------------------------------------------------------------------------#
#                                 Full Dataset                                  |
#-------------------------------------------------------------------------------#
# t_0 <-------------------------------------------------------------> t_n #

# K = 1
#-------------------------------------------------------------------------------#
#     Test      |     Train     |     Train     |     Train      |     Train    |
#-------------------------------------------------------------------------------#

# K = 2
#-------------------------------------------------------------------------------#
#     Train     |     Test      |     Train     |     Train      |     Train    |
#-------------------------------------------------------------------------------#

# K = 3
#-------------------------------------------------------------------------------#
#     Train     |     Train     |     Test      |     Train      |     Train    |
#-------------------------------------------------------------------------------#

# K = 4
#-------------------------------------------------------------------------------#
#     Train     |     Train     |     Train     |     Test       |     Train    |
#-------------------------------------------------------------------------------#

# K = 5
#-------------------------------------------------------------------------------#
#     Train     |     Train     |     Train     |     Train      |     Test     |
#-------------------------------------------------------------------------------#
```

```
# Create three empty lists
L_Sharpe_Ratios = []
LS_Sharpe_Ratios = []
Acc_score = []

# For loop who create model, fit model, test model and compute performance and␣
 ↪Sharpe Ratio
for k in range (1,6):
    # Choose of the features we want in the Random Forest model
    features = bco[['RSI','K_percent','ROC','MACD']]
    labels = bco['Pred_Signal']

    L1 = int(round(len(features))*0.20*k)+1
    L2 = int(round(len(features))*0.20*(k-1))

    X_test = features[L2:L1]
    X_train = features.drop(index=X_test.index)

    y_test = labels[L2:L1]
    y_train = labels.drop(index=y_test.index)

    for i in range(1,2):
        rf_random.fit(X_train, y_train)
        param = rf_random.best_params_
        rand_frst_clf = RandomForestClassifier(oob_score = True,
                                               n_estimators =␣
 ↪param['n_estimators'],
                                               min_samples_split =␣
 ↪param['min_samples_split'],
                                               min_samples_leaf =␣
 ↪param['min_samples_leaf'],
                                               max_features =␣
 ↪param['max_features'],
                                               max_depth = param['max_depth'],
                                               criterion = param['criterion'],
                                               bootstrap = True,
                                               random_state = 8)

        # Fit the data to the model
        rand_frst_clf.fit(X_train, y_train)

        # Make predictions
        y_pred = rand_frst_clf.predict(X_test)

        Acc_score.append(accuracy_score(y_test, y_pred, normalize = True) * 100.
 ↪0)
```

```
        # Trading Performance
        X = bco['Log_Returns'].shift(-1)
        X = X[L2:L1]

        trading_days_Y = 5*52

        # Only Long
        perf = y_pred * X
        perf = perf.dropna()
        # Sharpe
        perf_sharpe = np.sqrt(trading_days_Y) * np.mean(perf) / (np.sqrt(np.
↪var(perf)))
        L_Sharpe_Ratios.append(perf_sharpe)

        # Long Short
        # Transform 0,1 Signal in a -1,1 Signal
        y_pred1 = []
        for i in range(0,len(y_pred)):
            if y_pred[i] == 0:
                y_pred1.append(-1)
            else:
                y_pred1.append(1)

        perf1 = y_pred1 * X
        perf1 = perf1.dropna()
        # Sharpe
        perf_sharpe1 = np.sqrt(trading_days_Y) * np.mean(perf1) / (np.sqrt(np.
↪var(perf1)))
        LS_Sharpe_Ratios.append(perf_sharpe1)
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
Fitting 3 folds for each of 15 candidates, totalling 45 fits
Fitting 3 folds for each of 15 candidates, totalling 45 fits
Fitting 3 folds for each of 15 candidates, totalling 45 fits
Fitting 3 folds for each of 15 candidates, totalling 45 fits
```

[39]:
```
# Create data
data = [["Only Long", round(L_Sharpe_Ratios[0],2), round(L_Sharpe_Ratios[1],2),
         round(L_Sharpe_Ratios[2],2), round(L_Sharpe_Ratios[3],2),␣
↪round(L_Sharpe_Ratios[4],2)],
        ["Long Short", round(LS_Sharpe_Ratios[0],2),␣
↪round(LS_Sharpe_Ratios[1],2),
         round(LS_Sharpe_Ratios[2],2), round(LS_Sharpe_Ratios[3],2),␣
↪round(LS_Sharpe_Ratios[4],2)],
        ["Accuracy", round(Acc_score[0],2), round(Acc_score[1],2),
         round(Acc_score[2],2), round(Acc_score[3],2), round(Acc_score[4],2)]]
```

```
# Define header names
col_names = ["K = 1", "K = 2", "K = 3", "K = 4", "K = 5"]

# Display table
print('Random Forest')
print('K-Fold Cross Validation')
print('Sharpe Ratio')
print(tabulate(data, headers=col_names))
```

```
Random Forest
K-Fold Cross Validation
Sharpe Ratio
              K = 1    K = 2    K = 3    K = 4    K = 5
----------  -------  -------  -------  -------  -------
Only Long      1.18     0.4     -0.18     0.39     0.55
Long Short     1.7      0.27     0.5      0.34     0.36
Accuracy      53.22    52.13    49.66    51.44    50.07
```

[40]:
```
# Print all results
print('Mean Only Long Sharpe Ratio = ',round(np.mean(L_Sharpe_Ratios),2))
print('Mean Long Short Sharpe Ratio = ',round(np.mean(LS_Sharpe_Ratios),2))
print('Mean Accuracy Score = ',round(np.mean(Acc_score),2))
```

```
Mean Only Long Sharpe Ratio =  0.47
Mean Long Short Sharpe Ratio =  0.63
Mean Accuracy Score =  51.3
```

[ ]:
```

```

LSTM

June 4, 2022

1 LSTM model

1.0.1 Imports

[1]:
```
# Basic Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pprint import pprint
import yfinance as yf

# Deep Learning / Machine Learning Imoprts
import random
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import plot_confusion_matrix
import itertools
from sklearn.metrics import plot_roc_curve, roc_auc_score
from sklearn.metrics import roc_curve, auc
from keras.wrappers.scikit_learn import KerasClassifier
from statsmodels import *
from scipy.stats import *
import scipy
from arch import arch_model
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from tabulate import tabulate
from numpy.random import seed
```

2 Load CSV File

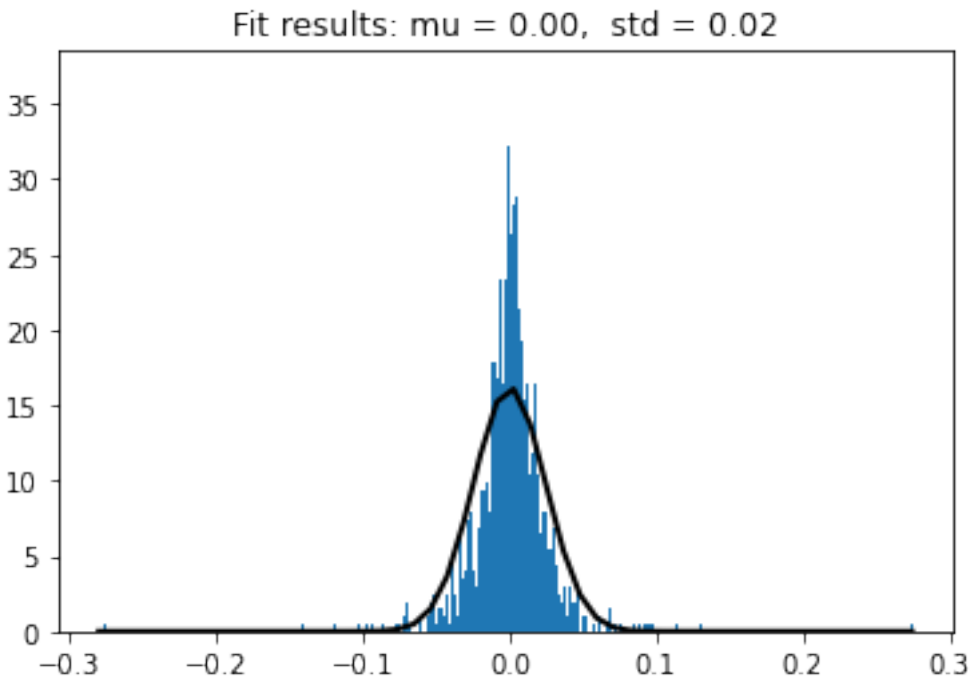
[2]:
```
# From 16.08.2007 to 26.04.2022
bco = pd.read_csv('/Users/romaindeleze/Desktop/BSc/Romain_WI_TZ/8.Semester/BA/
 ↪Daten/Brent_Crude_Oil')
```

[3]:
```
bco.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3641 entries, 0 to 3640
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Date         3641 non-null   object
 1   Close        3641 non-null   float64
 2   Log_Returns  3641 non-null   float64
 3   Pred_Signal  3641 non-null   int64
 4   RSI          3641 non-null   float64
 5   K_percent    3641 non-null   float64
 6   MACD         3641 non-null   float64
 7   ROC          3641 non-null   float64
dtypes: float64(6), int64(1), object(1)
memory usage: 227.7+ KB
```

2.1 Fit a GARCH(1,1) Model to see how many lags are needet to explain t+1

[4]:
```
#Lets check the distribution of the data
# Generate some data for this demonstration.
data = bco['Log_Returns']

# Fit a normal distribution to the data:
mu, std = scipy.stats.norm.fit(data)

# Plot the histogram.
plt.hist(data, bins=1000, density=True)

# Plot the PDF.
xmin, xmax = np.min(data), np.max(data)
x = np.linspace(xmin, xmax)
p = norm.pdf(x, mu, std)
plt.plot(x, p, 'k', linewidth=2)
title = "Fit results: mu = %.2f,  std = %.2f" % (mu, std)
plt.title(title)
plt.show()
```

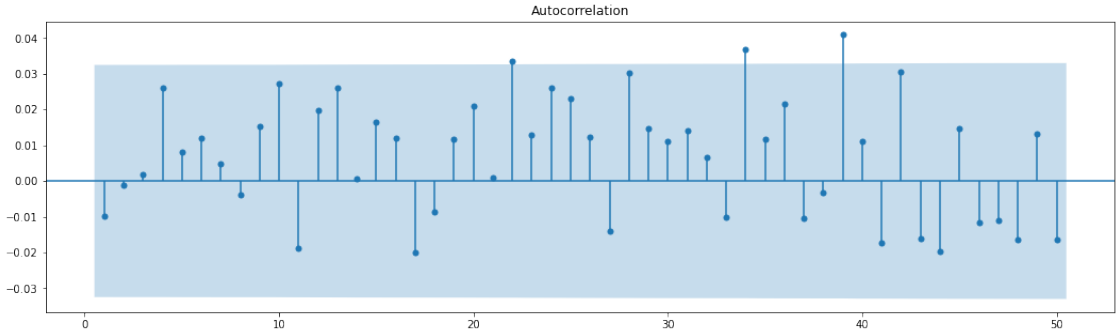


[5]:
```
# Defining the Model
garch_model = arch_model(bco['Log_Returns'], mean = "Zero", vol="GARCH", p=1,␣
↪q=1, dist='t')

# Fitting the Model
garch_model_results = garch_model.fit()

# Printing the model summary
print(garch_model_results.summary())
```

```
Iteration:      1,   Func. Count:      5,   Neg. LLF: -9213.148204331432
Optimization terminated successfully    (Exit mode 0)
            Current function value: -9213.148204396279
            Iterations: 5
            Function evaluations: 5
            Gradient evaluations: 1
                      Zero Mean - GARCH Model Results
==============================================================================
====
Dep. Variable:                  Log_Returns   R-squared:
0.000
Mean Model:                       Zero Mean   Adj. R-squared:
0.000
Vol Model:                            GARCH   Log-Likelihood:
9213.15
```

```
Distribution:      Standardized Student's t   AIC:
-18418.3
Method:                  Maximum Likelihood   BIC:
-18393.5
                                              No. Observations:
3641
Date:                      Sat, Jun 04 2022   Df Residuals:
3641
Time:                             18:15:48   Df Model:
0
                            Volatility Model
=============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
-----------------------------------------------------------------------------
omega      1.2155e-05  4.133e-08    294.089      0.000 [1.207e-05,1.224e-05]
alpha[1]       0.1000  1.124e-02      8.893  5.932e-19   [7.796e-02,  0.122]
beta[1]        0.8800  8.023e-03    109.681      0.000     [  0.864,  0.896]
                            Distribution
=============================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
-----------------------------------------------------------------------------
nu             5.4563      0.974      5.604  2.090e-08 [  3.548,  7.364]
=============================================================================

Covariance estimator: robust

/opt/anaconda3/lib/python3.8/site-packages/arch/univariate/base.py:309:
DataScaleWarning: y is poorly scaled, which may affect convergence of the
optimizer when
estimating the model parameters. The scale of y is 0.0006077. Parameter
estimation work better when this value is between 1 and 1000. The recommended
rescaling is 100 * y.

This warning can be disabled by either rescaling y before initializing the
model or by setting rescale=False.

  warnings.warn(
```

[6]:
```python
# Fixing plot size
plt.rcParams["figure.figsize"] = 18, 5

# Calculate standardized residuals
standard_residuals = garch_model_results.resid/garch_model_results.
 ↪conditional_volatility

# Plot ACF of the stanadtized residuals
plot_acf(standard_residuals, zero = False, lags = 50)
plt.show()
```



3 Building the Long Short Term Memory Model

We have split our data into a training set and testing set, so we need to identify our input which are the following:

Close price

We have to split the Close Price in a train and test dataset. For the LSTM algorithm we creat laged datasets so that we have a kind of a linear function who train the neural network. The LSTM creates the optimal parameters in each node to predict the next day with the past data.

For the LSTM model we are only interested in the Close column. We filter the Close column and save in a new object

[7]:
```
# Create a new dataframe object with only the Close column
data = bco.filter(['Close'])

# Extract only the values
close = data.values

# Creat object with the training data lenght
training_data_len = round(len(close)*0.8)

# Backtesting the lenghts
print('Lenght of the Whole dataset = ',len(close))
print('Training data lenght = ',training_data_len)
print('Test data lenght = ',len(close)-training_data_len)
print(training_data_len,' + ',len(close)-training_data_len,' = ',␣
↪training_data_len + len(close)-training_data_len)
```

```
Lenght of the Whole dataset =  3641
Training data lenght =  2913
Test data lenght =  728
2913  +  728  =  3641
```

For better results we have to scale the dataset with the MinMaxScaler(). This is a function who Scale the values between 0 and 1.

[8]:
```
# We sacale the data with the MinMaxSacaler in values between 0 and 1
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(close)
```

[9]:
```
# Transform the sacled data in a data frame
df = pd.DataFrame(scaled_data, columns=['Scaled_Close'])
```

[10]:
```
# Lag Data
def buildLaggedFeatures(s,lag=2,dropna=True):

    if type(s) is pd.DataFrame:
        new_dict={}
        for col_name in s:
            new_dict[col_name]=s[col_name]
            # create lagged Series
            for l in range(1,lag+1):
                new_dict['%s_lag%d' %(col_name,l)]=s[col_name].shift(l)
        res=pd.DataFrame(new_dict,index=s.index)

    elif type(s) is pd.Series:
        the_range=range(lag+1)
        res=pd.concat([s.shift(i) for i in the_range],axis=1)
        res.columns=['lag_%d' %i for i in the_range]
    else:
        print('Only works for DataFrame or Series')
        return None
    if dropna:
        return res.dropna()
    else:
        return res

res=buildLaggedFeatures(df,lag=39,dropna=False)

# Drop all columns with NA values
res = res.dropna()
```

[11]:
```
# Split Data in X_train, X_test, y_train, y_test

X = res.iloc[:,1:]

y = res.iloc[:,0:1]

# Split Features
X_train = X[:training_data_len]
X_test = X[training_data_len:]
```

```
# Split Label
y_train = y[:training_data_len]
y_test = y[training_data_len:]
```

[12]:
```
# Convert the X_train and y_train to numpy arrays
X_train, y_train = np.array(X_train), np.array(y_train)
print(X_train.shape),print(y_train.shape)

# Reshape the data
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))

# Convert the data to a numpy array
X_test = np.array(X_test)

# Reshape the data
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1 ))
```

```
(2913, 39)
(2913, 1)
```

4 We build the LSTM model and use it to make the prediction

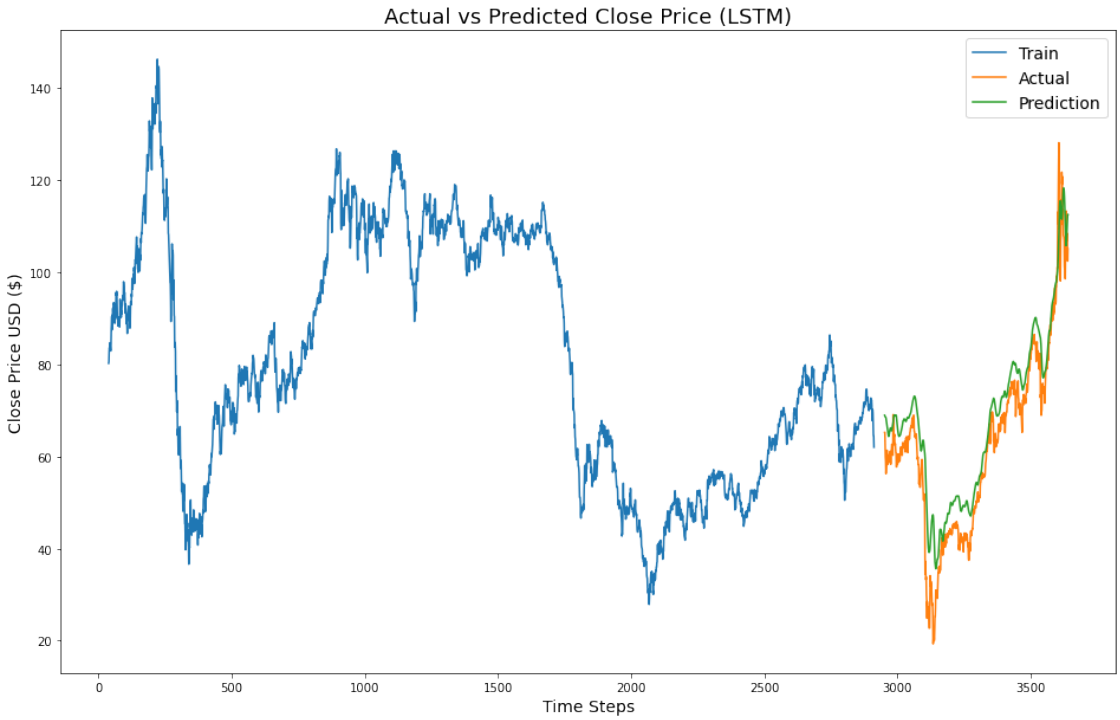
[13]:
```
# Build the LSTM model
seed(8)
tf.random.set_seed(8)
def build_model():
    model = Sequential()
    model.add(LSTM(128, return_sequences=True, input_shape= (X_train.shape[1],␣
 ↪1)))
    model.add(LSTM(64, return_sequences=False))
    model.add(Dense(25))
    model.add(Dense(1))

    # Compile the model
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model

model = build_model()
history = model.fit(X_train, y_train, batch_size=1, epochs=5)

# Get the models predicted price values
predictions = model.predict(X_test)
predictions = scaler.inverse_transform(predictions) # back-transform Prediction
```

```
Epoch 1/5
2913/2913 [==============================] - 42s 14ms/step - loss: 0.0063
```

```
Epoch 2/5
2913/2913 [==============================] - 52s 18ms/step - loss: 0.0037
Epoch 3/5
2913/2913 [==============================] - 56s 19ms/step - loss: 0.0030
Epoch 4/5
2913/2913 [==============================] - 52s 18ms/step - loss: 0.0019
Epoch 5/5
2913/2913 [==============================] - 45s 16ms/step - loss: 0.0016
```

[14]:
```python
# Plot the data
train = data[X_train.shape[1]:training_data_len]
test = data[training_data_len+X_train.shape[1]:]
test['Prediction'] = predictions

# Visualize the data
plt.figure(figsize=(16,10))
plt.title('Actual vs Predicted Close Price (LSTM)', fontsize=18)
plt.xlabel('Time Steps', fontsize=14)
plt.ylabel('Close Price USD ($)', fontsize=14)
plt.plot(train['Close'])
plt.plot(test[['Close', 'Prediction']])
plt.legend(['Train', 'Actual', 'Prediction'], loc='best', fontsize=14)
plt.show()
```

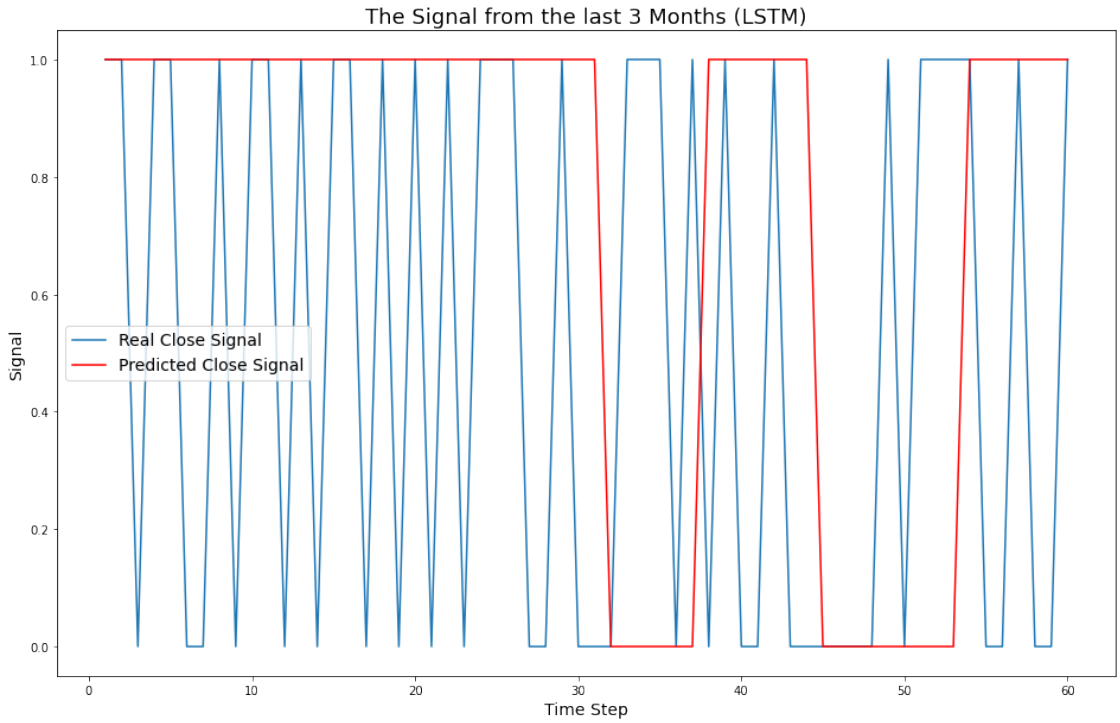
```
<ipython-input-14-ca7e21da7781>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test['Prediction'] = predictions
```



[15]:
```
# Transform real and predicted close price in a Signal
X_test = test['Close']
y_pred = test['Prediction']

X_r = X_test.pct_change() # Real Returns in Test data
X1 = (X_r > 0).astype(int) # Real low risk Signal

y_r = y_pred.pct_change() # Predicted Returns from Test data
y1 = (y_r > 0).astype(int) # Predicted low risk Signal
```

[16]:
```
# Lets see the actual and the predicted Signal from the last 3 months
m3 = 3*4*5
x = np.linspace(1,m3,m3)
plt.figure(figsize=(16,10))
plt.title('The Signal from the last 3 Months (LSTM)', fontsize=18)
plt.plot(x,X1[len(X1)-m3:],label = 'Real Close Signal')
plt.plot(x,y1[len(y1)-m3:],color = 'red', label = 'Predicted Close Signal')
plt.ylabel('Signal', fontsize=14)
plt.xlabel('Time Step', fontsize=14)
plt.legend(loc='best', fontsize=14)
```

[16]:
```
<matplotlib.legend.Legend at 0x7fb3ab5d5cd0>
```



5 Accuracy

We've built our model, so now we can see how accurate it is. SciKit learn, makes the process of evaluating our model very easy by providing a bunch of built-in metrics that we can call. One of those metrics is the accuracy_score. The accuracy_score function computes the accuracy, by calculating the sum of the correctly predicted signals and then dividing it by the total number of predictions. Imagine we had three TRUE values [1, 2, 3] , and our model predicted the following values [1, 2, 4] we would say the accuracy of our model is $2/3 = 66\ \%$.

6 AUC

AUC curve is a performance measurement for the classification problems at various threshold settings. AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1. By analogy, the Higher the AUC, the better the model is at distinguishing between patients with the disease and no disease.

7 RMSE

Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are;

RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit.

[17]:
```
# Now we can compare the accuracy from the prediction
print('Accuracy in % : ', accuracy_score(X1, y1) * 100.0) # Hit Ratio low risk␣
 ↪strategie!
print()

# Checking performance our model with AUC Score.
print('Area under the curve: ',roc_auc_score(X1, y1))
print()

# Get the root mean squared error (RMSE)
print('Close Price RMSE = ' ,np.sqrt(np.mean(((predictions - y_test) ** 2))))
print('Signal RMSE = ' ,np.sqrt(np.mean(((y1 - X1) ** 2))))
```

```
Accuracy in % :  51.814223512336724

Area under the curve:  0.50670307422719

Close Price RMSE =  Scaled_Close    70.170485
dtype: float64
Signal RMSE =  0.6941597545786077
```

8 Classification Report

To get a more detailed overview of how the model performed, we can build a classification report that will compute the F1_Score , the Precision , the Recall , and the Support .

8.0.1 Precision

Precision measures the proportion of all correctly identified samples in a population of samples which are classified as positive labels and is defined as the following:

tp = True Positiv

fp = False Positiv

Precision = tp/(tp+fp)

The precision is intuitively the ability of the classifier not to label as positive a sample that is negative. The best value is 1, and the worst value is 0.

8.0.2 Recall

Recall (also known as sensitivity) measures the ability of a classifier to correctly identify positive labels and is defined as the following:

tp = True Positiv

fn = False Negativ

Recall = tp/(tp+fn)

The recall is intuitively the ability of the classifier to find all the positive samples. The best value is 1, and the worst value is 0.

8.0.3 Support

Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing. Support doesn't change between models but instead diagnoses the evaluation process.

8.0.4 F1 Score

In some cases, we will have models that may have low precision or high recall. It's difficult to compare two models with low precision and high recall or vice versa. To make results comparable, we use a metric called the F-Score. The F-score helps to measure Recall and Precision at the same time. It uses Harmonic Mean in place of Arithmetic Mean by punishing the extreme values more.

The traditional F-measure or balanced F-score (F1 score) is the harmonic mean of precision and recall:

tp = True Positiv

fp = False Positiv

fn = False Negativ

F1 Score = tp/(tp+0.5*(fp+fn))

[18]:
```
# Define the traget names
target_names = ['Down Day', 'Up Day']

# Build a classifcation report
report = classification_report(y_true = X1, y_pred = y1, target_names =␣
↪target_names, output_dict = True)

# Add it to a data frame, transpose it for readability.
report_df = pd.DataFrame(report).transpose()
report_df
```

[18]:

	precision	recall	f1-score	support
Down Day	0.460076	0.389068	0.421603	311.000000
Up Day	0.553991	0.624339	0.587065	378.000000
accuracy	0.518142	0.518142	0.518142	0.518142
macro avg	0.507033	0.506703	0.504334	689.000000
weighted avg	0.511600	0.518142	0.512379	689.000000

9 Confusion Matrix

A confusion matrix is a technique for summarizing the performance of a classification algorithm. Classification accuracy alone can be misleading if you have an unequal number of observations in each class or if you have more than two classes in your dataset. Calculating a confusion matrix can give you a better idea of what your classification model is getting right and what types of errors it is making.

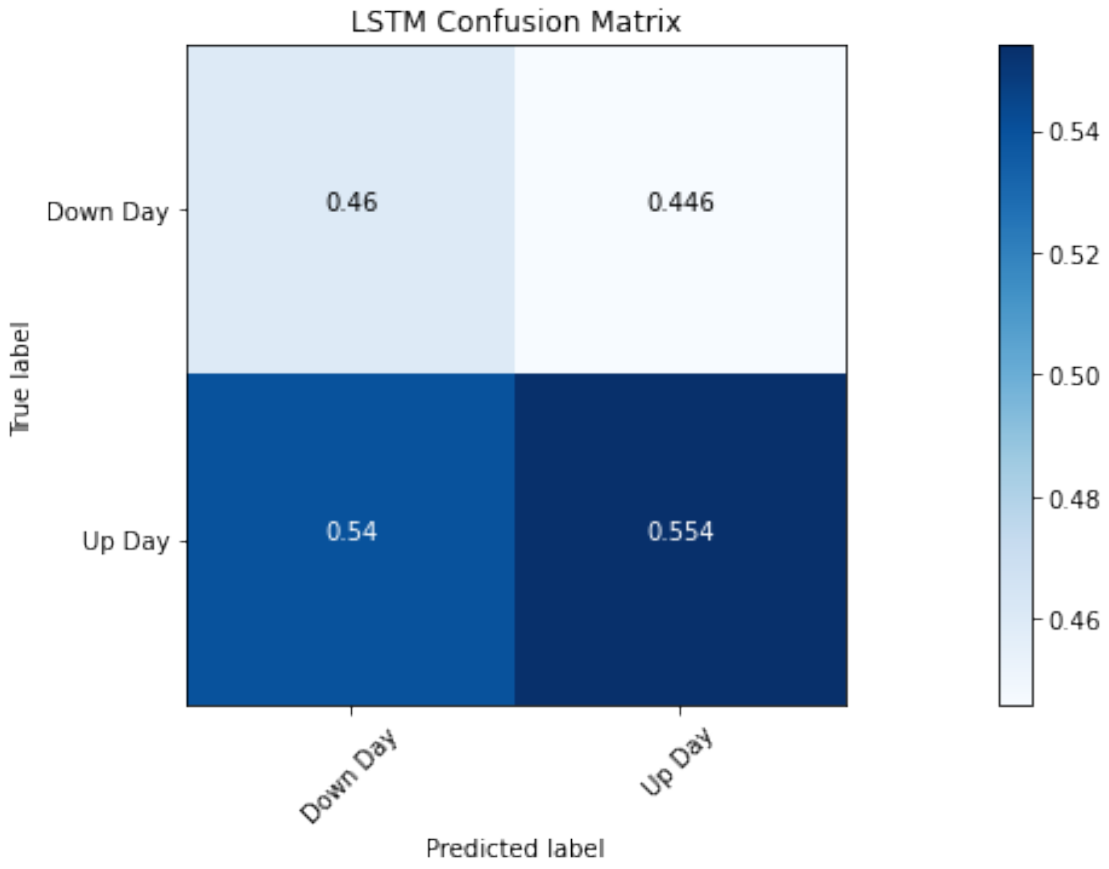
[19]:
```
cm = confusion_matrix(X1, y1, normalize='pred')
```

[20]:
```
# Function: Confusion Matrix Plot for Regression, Neural Network
def plot_confusion_matrix_regression(cm, classes,
                        normalize=False,
                        title='Confusion matrix',
                        cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 1.2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
            horizontalalignment="center",
            color="white" if cm[i, j] > thresh else "black")
```

```
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

[21]:
```
# Compute Confusion Matrix
LSTM_matrix = confusion_matrix(y1, X1)

# Save individual values
true_negatives = LSTM_matrix[0][0]
false_negatives = LSTM_matrix[1][0]
true_positives = LSTM_matrix[1][1]
false_positives = LSTM_matrix[0][1]

# Compute performance values
accuracy = (true_negatives + true_positives) / (true_negatives + true_positives␣
 ↪+ false_negatives + false_positives)
percision = true_positives / (true_positives + false_positives)
recall = true_positives / (true_positives + false_negatives)
specificity = true_negatives / (true_negatives + false_positives)

# Print values
print('Accuracy: {}'.format(round(float(accuracy),5)))
print('Percision: {}'.format(round(float(percision),5)) )
print('Recall: {}'.format(round(float(recall),5)) )
print('Specificity: {}'.format(round(float(specificity),5)))
print()

# Visualise the Confusion Matrix
cm_plot_labels = ['Down Day','Up Day']
disp = plot_confusion_matrix_regression(cm=np.round(cm,3),␣
 ↪classes=cm_plot_labels, title='LSTM Confusion Matrix',
                                 normalize=False)
```

```
Accuracy: 0.51814
Percision: 0.62434
Recall: 0.55399
Specificity: 0.46008

Confusion matrix, without normalization
[[0.46  0.446]
 [0.54  0.554]]
```



9.0.1 With the information from the confusion matrix we want know now how the proportion is.

How is the prediction distributed, and how ist the real values distributed

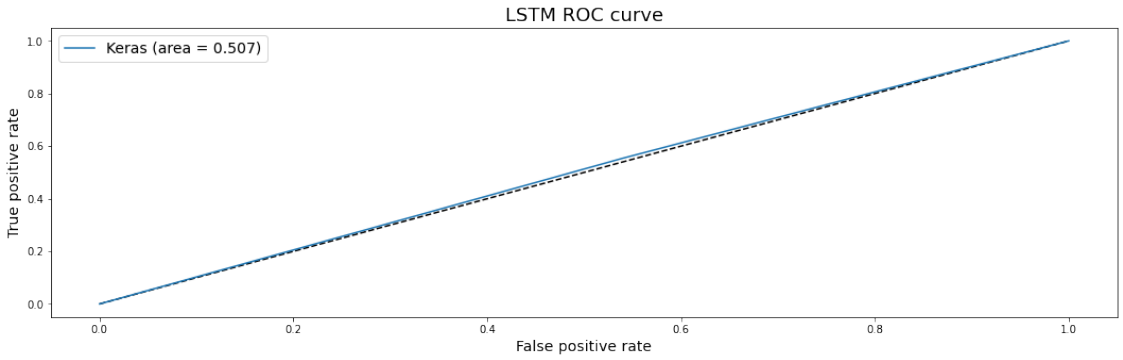
[22]:
```
# Propotion of the real values
unique, counts = np.unique(X1, return_counts=True)
print('Real Proportion')
print(dict(zip(unique, counts/len(X1))))
print()

# Propotion of the predicted values
unique, counts = np.unique(y1, return_counts=True)
print('Predicted Proportion')
print(dict(zip(unique, counts/len(y1))))
```

```
Real Proportion
{0: 0.4513788098693759, 1: 0.548621190130624}

Predicted Proportion
{0: 0.38171262699564584, 1: 0.6182873730043541}
```

10 ROC Curve

The Receiver Operating Characteristic is a graphical method to evaluate the performance of a binary classifier. A curve is drawn by plotting True Positive Rate (sensitivity) against False Positive Rate (1 - specificity) at various threshold values. ROC curve shows the trade-off between sensitivity and specificity. When the curve comes closer to the left-hand border and the top border of the ROC space, it indicates that the test is accurate. The closer the curve is to the top and left-hand border, the more accurate the test is. If the curve is close to the 45 degrees diagonal of the ROC space, it means that the test is not accurate. ROC curves can be used to select the optimal model and discard the suboptimal ones.

[23]:
```
y_pred_Signal = y1
y_test_Signal = X1

fpr_keras, tpr_keras, thresholds_keras = roc_curve(y_pred_Signal, y_test_Signal)

auc_keras = auc(fpr_keras, tpr_keras)

plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_keras, tpr_keras, label='Keras (area = {:.3f})'.format(auc_keras))
plt.xlabel('False positive rate', fontsize=14)
plt.ylabel('True positive rate', fontsize=14)
plt.title('LSTM ROC curve', fontsize=18)
plt.legend(loc='best', fontsize=14)
plt.show()
```



11 Trading Performance

[24]:
```
# Creat Data Frame with Predicted Signal and Log Returns
n = len(y_pred)
X2 = bco['Log_Returns']
X2 = X2[len(X2)-n:]
```

```
len(X2)
```

[24]: 689

11.1 Only Long

[25]:
```
# Compute Only Long performance: Signal * Log Returns
perf = y1 * X2
perf = perf.dropna()

trading_days_Y = 5*52

perf_sharpe = np.sqrt(trading_days_Y) * np.mean(perf) / (np.sqrt(np.var(perf)))
```

11.2 Long-Short

[26]:
```
# Compute Long Short performance: Signal * Log Returns
perf1 = np.sign(y_r) * X2
perf1 = perf1.dropna()

trading_days_Y = 5*52

perf_sharpe1 = np.sqrt(trading_days_Y) * np.mean(perf1) / (np.sqrt(np.
 ↪var(perf1)))
```

11.3 Buy and Hold

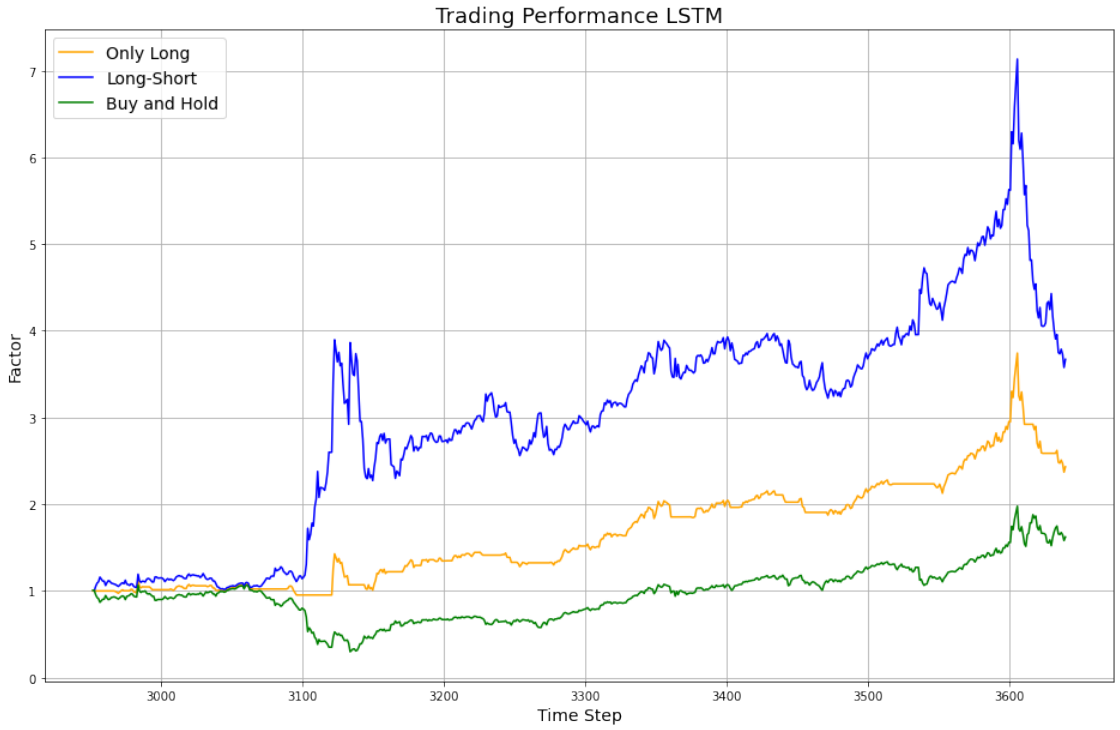
[27]:
```
perf2 = X2
perf2 = perf2.dropna()

trading_days_Y = 5*52

perf_sharpe2 = np.sqrt(trading_days_Y) * np.mean(perf2) / (np.sqrt(np.
 ↪var(perf2)))
```

12 Comparison ML vs. Buy and Hold

[28]:
```
# Lets compare the performances
plt.figure(figsize=(16,10))
plt.title('Trading Performance LSTM', fontsize=18)
plt.ylabel('Factor', fontsize=14)
plt.xlabel('Time Step', fontsize=14)
plt.grid(True)
plt.plot(np.exp(perf).cumprod() , label='Only Long', color = 'orange')
plt.plot(np.exp(perf1).cumprod() , label='Long-Short', color = 'blue')
```

```
plt.plot(np.exp(perf2).cumprod() , label='Buy and Hold', color = 'green')
plt.legend(loc='best', fontsize=14)
```

[28]: <matplotlib.legend.Legend at 0x7fb3abe8ffa0>



13 Profit Factor

[29]:
```
# Profit Factor is the last value of the performance computation
L_profit = np.exp(perf).cumprod().iloc[-1]

LS_profit = np.exp(perf1).cumprod().iloc[-1]

BaH_profit = np.exp(perf2).cumprod().iloc[-1]
```

14 Performance Table

[30]:
```
# Create data
data = [["Buy and Hold", round(perf_sharpe2,2), round(BaH_profit,2)],
        ["Only Long", round(perf_sharpe,2), round(L_profit,2)],
        ["Long Short", round(perf_sharpe1,2), round(LS_profit,2)]]

# Define header names
```

```
col_names = ["Strategie", "Sharp Ratio", "Profit Factor"]

# Display table
print('LSTM')
print(tabulate(data, headers=col_names))
```

```
LSTM
Strategie       Sharp Ratio    Profit Factor
------------  -------------  ---------------
Buy and Hold           0.33             1.62
Only Long              0.95             2.43
Long Short             0.89             3.67
```

[31]:
```
# Performance Dataset
# Compute performance
LSTM_Perf_OL = np.exp(perf).cumprod()
LSTM_Perf_LS = np.exp(perf1).cumprod()

# Load in a Data Frame
LSTM_Perf_data = pd.DataFrame()
LSTM_Perf_data['LSTM_Perf_OL'] = LSTM_Perf_OL[:720]
LSTM_Perf_data['LSTM_Perf_LS'] = LSTM_Perf_LS

# Create a csv File
LSTM_Perf_data.to_csv('/Users/romaindeleze/Desktop/BSc/Romain_WI_TZ/8.Semester/
↪BA/Daten/LSTM_Perf_data')
```

15 Extrem and Regular Value Accuracy

[32]:
```
# Creat Data Frame with Predicted Signal and Log Returns
n = len(y_pred)
X3 = bco['Log_Returns']
X3 = X3[len(X3)-n:]
y2 = np.squeeze(y1) # Dimension Reduction
d = {'Predicted Signal':y2.values, 'Log Returns':X3.values}
df = pd.DataFrame(data=d)
```

[33]:
```
# Creat the upperbound and the lowerbound
mu = np.mean(X3)
sig = np.std(X3)
lb = mu-2*sig
ub = mu+2*sig

# Creat Data Frame with only Values in the interval [lb,ub]
reg_val = df[(df['Log Returns'] >= lb) & (df['Log Returns'] <= ub)] # DataFrame␣
↪with Regular Values = [lb,ub]
```

```
# Creat Data Frame with only Values outside the interval [lb,ub]
ext_val = df.drop(index=reg_val.index)

# Add real Signal with the Log Returns for each Data Frame
reg_val['Real Signal'] = (reg_val['Log Returns'] > 0).astype(int)
ext_val['Real Signal'] = (ext_val['Log Returns'] > 0).astype(int)

# Compute accuracy of the Extrem Values and the Regular Values
acc_reg = accuracy_score(reg_val['Real Signal'], reg_val['Predicted Signal'],␣
 ↪normalize = True) * 100.0
acc_ext = accuracy_score(ext_val['Real Signal'], ext_val['Predicted Signal'],␣
 ↪normalize = True) * 100.0
print('Accuracy Regular Values = ', round(acc_reg,2))
print('Accuracy Exrtem Values (95%) = ', round(acc_ext,2))
```

```
Accuracy Regular Values =  51.52
Accuracy Exrtem Values (95%) =  54.84
```

```
<ipython-input-33-d401731cabac>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  reg_val['Real Signal'] = (reg_val['Log Returns'] > 0).astype(int)
```

[34]:
```
# Creat the upperbound and the lowerbound
mu = np.mean(X3)
sig = np.std(X3)
lb = mu-3*sig
ub = mu+3*sig

# Creat Data Frame with only Values in the interval [lb,ub]
reg_val = df[(df['Log Returns'] >= lb) & (df['Log Returns'] <= ub)] # DataFrame␣
 ↪with Regular Values = [lb,ub]

# Creat Data Frame with only Values outside the interval [lb,ub]
ext_val = df.drop(index=reg_val.index)

# Add real Signal with the Log Returns for each Data Frame
reg_val['Real Signal'] = (reg_val['Log Returns'] > 0).astype(int)
ext_val['Real Signal'] = (ext_val['Log Returns'] > 0).astype(int)

# Compute accuracy of the Extrem Values and the Regular Values
acc_reg = accuracy_score(reg_val['Real Signal'], reg_val['Predicted Signal'],␣
 ↪normalize = True) * 100.0
```

```
acc_ext = accuracy_score(ext_val['Real Signal'], ext_val['Predicted Signal'],␣
 ↪normalize = True) * 100.0
print('Accuracy Regular Values = ', round(acc_reg,2))
print('Accuracy Exrtem Values (99%) = ', round(acc_ext,2))
```

```
Accuracy Regular Values =  51.41
Accuracy Exrtem Values (99%) =  64.29
```

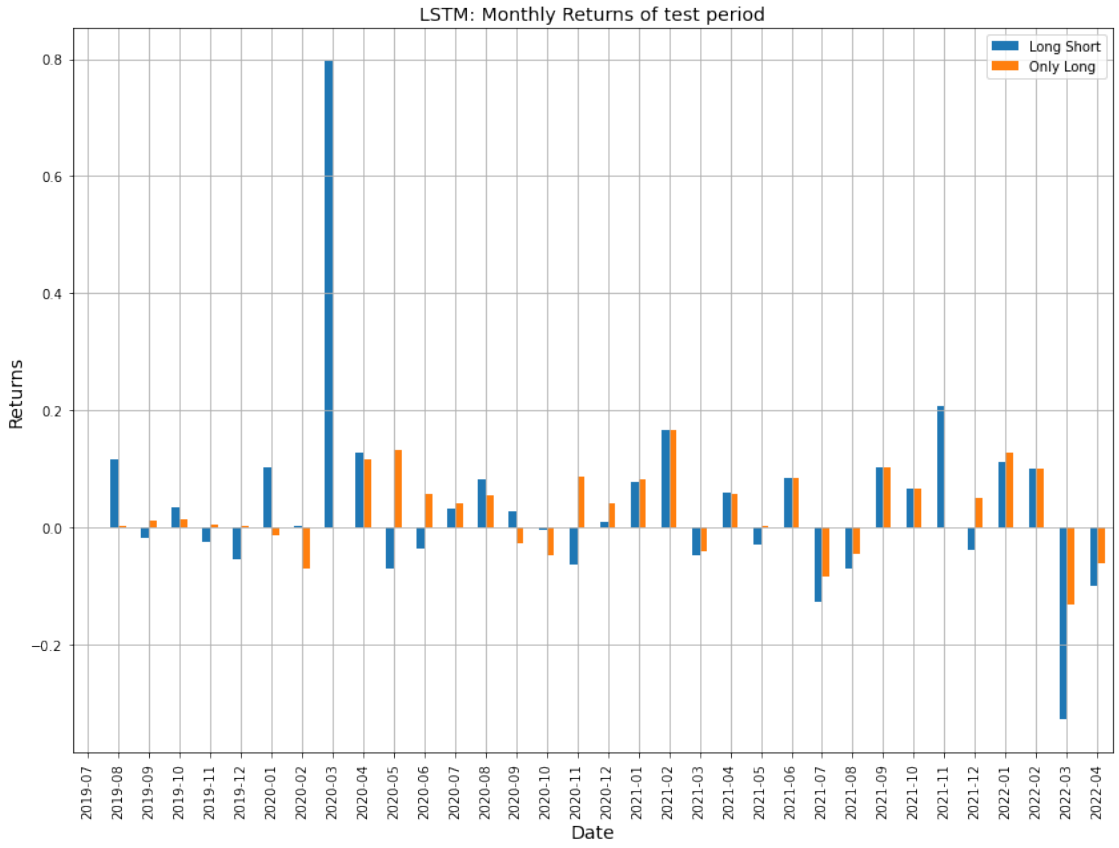
```
<ipython-input-34-6cda256fddbd>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  reg_val['Real Signal'] = (reg_val['Log Returns'] > 0).astype(int)
```

[35]:
```
perf1 = np.squeeze(perf1) # Dimension Reduction
perf = np.squeeze(perf) # Dimension Reduction
perf1 = pd.Series(perf1)
perf = pd.Series(perf[1:])
```

[36]:
```
# Indexing
dat = yf.download('BZ=F', start='2007-08-16',end = "2022-04-26",period='1d')
end_len = len(dat["Close"])
long_short = np.exp(perf1).cumprod()
start_len = end_len - len(long_short)
index = dat["Close"][start_len:end_len].index
perf1.index = index
perf.index = index

# Create Monthly Returns Long Short
da = perf1.groupby([(perf1.index.year),(perf1.index.month)]).sum()
monthly_ret_LS = da.values
n = len(monthly_ret_LS)
monthly_ret_LS = pd.DataFrame(monthly_ret_LS,index = pd.date_range(start='6/1/
 ↪2019', freq='M', periods=n))

# Create Monthly Returns Long
da1 = perf.groupby([(perf.index.year),(perf.index.month)]).sum()
monthly_ret_OL = da1.values
monthly_ret_OL = pd.DataFrame(monthly_ret_OL,index = pd.date_range(start='6/1/
 ↪2019', freq='M', periods=n))

# Connect DF
mon_ret = pd.concat([monthly_ret_LS, monthly_ret_OL], axis=1)
mon_ret.columns = ["Long Short","Only Long"]
```

```
# x Axis
x_ax = pd.period_range('6/1/2019', '2022-04-30', freq='M').strftime('%Y-%m')
mon_ret.index = x_ax[1:]

#Plot the barplot
ax = mon_ret.plot.bar(figsize=(14,10), rot = 0)
plt.title("LSTM: Monthly Returns of test period",fontsize=14)
plt.xlabel('Date', fontsize=14)
plt.xticks(rotation=90)
plt.ylabel('Returns', fontsize=14)
plt.legend(loc='best')
plt.grid()
```

```
[**********************100%**********************]  1 of 1 completed
```



[37]:
```
#MDD and Maximum Profit
max_profit_LS = np.max(da)
max_prift_OL = np.max(da1)
MDD_LS = np.min(da)
MDD_OL = np.min(da1)
```

```
# Create data
data = [["Only Long", round(MDD_OL,2), round(max_prift_OL,2)],
["Long Short", round(MDD_LS,2), round(max_profit_LS,2)]]

# Define header names
col_names = ["Strategie", "MDD", "Max. Profit"]

# Display table
print('MDD and MAX.Profti of LSTM')
print(tabulate(data, headers=col_names))
```

```
MDD and MAX.Profti of LSTM
Strategie      MDD    Max. Profit
-----------  -----  -------------
Only Long    -0.13           0.17
Long Short   -0.33           0.8
```

16 Time Series Cross Validation Backwards

[38]:
```
#-------------------------------------------------------------------------#
#                              Full Dataset                               |
#-------------------------------------------------------------------------#
# t_0 <-------------------------------------------------------------> t_n #

# K = 1
#-------------------------------------------------------------------------#
#              |              |              |    Train      |     Test     |
#-------------------------------------------------------------------------#

# K = 2
#-------------------------------------------------------------------------#
#              |              |     Train    |    Train      |     Test     |
#-------------------------------------------------------------------------#

# K = 3
#-------------------------------------------------------------------------#
#              |     Train    |     Train    |    Train      |     Test     |
#-------------------------------------------------------------------------#

# K = 4
#-------------------------------------------------------------------------#
#     Train    |     Train    |     Train    |    Train      |     Test     |
#-------------------------------------------------------------------------#

# Create two empty lists
L_Sharpe_Ratios = []
```

```
LS_Sharpe_Ratios = []
seed(8)
tf.random.set_seed(8)

# For loop who create model, fit model, test model and compute performance and␣
 ↪Sharpe Ratio
for k in range (1,5):

    L1 = int(round(len(X)*0.2))
    X_test = X[-L1:] # Testdata
    y_test = y[-L1:] # Testdata
    train1 = X[:-L1] # Cut the Testdata from the whole Dataset
    train2 = y[:-L1]

    L2 = int(round(len(train1)*0.25*k))

    X_train = train1[-L2:] # Traindata
    y_train = train2[-L2:] # Traindata

    # Convert the X_train and y_train to numpy arrays
    X_train, y_train = np.array(X_train), np.array(y_train)

    # Reshape the data
    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))

    # Convert the data to a numpy array
    X_test = np.array(X_test)

    # Reshape the data
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1 ))

    # Fit the model
    model.fit(X_train, y_train, batch_size=1, epochs=5)

    # Get the models predicted price values
    predictions = model.predict(X_test)
    predictions = scaler.inverse_transform(predictions) # back-transform␣
 ↪Prediction

    y_pred = pd.DataFrame(predictions)

    y_r = y_pred.pct_change() # Predicted Returns from Test data
    y1 = (y_r > 0).astype(int) # Predicted low risk Signal

    n = len(y1)
    X2 = bco['Log_Returns']
    X2 = X2[len(X2)-n:]
```

```
    perf = y1.T.values * X2.values

    trading_days_Y = 5*52

    perf_sharpe = np.sqrt(trading_days_Y) * np.mean(perf) / (np.sqrt(np.
↪var(perf)))
    L_Sharpe_Ratios.append(perf_sharpe)

    y3 = np.sign(y_r)
    y3 = y3[1:]
    X2 = X2[1:]
    perf1 = y3.T.values * X2.values
    perf_sharpe1 = np.sqrt(trading_days_Y) * np.mean(perf1) / (np.sqrt(np.
↪var(perf1)))
    LS_Sharpe_Ratios.append(perf_sharpe1)
```

```
Epoch 1/5
720/720 [==============================] - 10s 14ms/step - loss: 4.2131e-04
Epoch 2/5
720/720 [==============================] - 10s 14ms/step - loss: 3.6295e-04
Epoch 3/5
720/720 [==============================] - 10s 14ms/step - loss: 3.9954e-04
Epoch 4/5
720/720 [==============================] - 10s 14ms/step - loss: 2.9476e-04
Epoch 5/5
720/720 [==============================] - 12s 17ms/step - loss: 3.8097e-04
Epoch 1/5
1441/1441 [==============================] - 25s 17ms/step - loss: 0.0012
Epoch 2/5
1441/1441 [==============================] - 27s 18ms/step - loss: 7.5089e-04
Epoch 3/5
1441/1441 [==============================] - 26s 18ms/step - loss: 7.4213e-04
Epoch 4/5
1441/1441 [==============================] - 21s 14ms/step - loss: 5.9706e-04
Epoch 5/5
1441/1441 [==============================] - 21s 14ms/step - loss: 7.1261e-04
Epoch 1/5
2162/2162 [==============================] - 31s 15ms/step - loss: 8.5101e-04
Epoch 2/5
2162/2162 [==============================] - 35s 16ms/step - loss: 8.8014e-04
Epoch 3/5
2162/2162 [==============================] - 41s 19ms/step - loss: 7.0299e-04
Epoch 4/5
2162/2162 [==============================] - 38s 18ms/step - loss: 6.5038e-04
Epoch 5/5
2162/2162 [==============================] - 36s 17ms/step - loss: 7.0368e-04
```

```
Epoch 1/5
2882/2882 [==============================] - 45s 16ms/step - loss: 8.9996e-04
Epoch 2/5
2882/2882 [==============================] - 35s 12ms/step - loss: 6.9648e-04
Epoch 3/5
2882/2882 [==============================] - 37s 13ms/step - loss: 7.9339e-04
Epoch 4/5
2882/2882 [==============================] - 35s 12ms/step - loss: 7.2650e-04
Epoch 5/5
2882/2882 [==============================] - 35s 12ms/step - loss: 6.4189e-04
```

[39]:
```
# Create data
data = [["Only Long", round(L_Sharpe_Ratios[0],2), round(L_Sharpe_Ratios[1],2),
         round(L_Sharpe_Ratios[2],2), round(L_Sharpe_Ratios[3],2)],
        ["Long Short", round(LS_Sharpe_Ratios[0],2),␣
 ↪round(LS_Sharpe_Ratios[1],2),
         round(LS_Sharpe_Ratios[2],2), round(LS_Sharpe_Ratios[3],2)]]

# Define header names
col_names = ["K = 1", "K = 2", "K = 3", "K = 4"]

# Display table
print('LSTM')
print('Cross Validation Time Series Split Backward')
print('Sharpe Ratio')
print(tabulate(data, headers=col_names))
```

```
LSTM
Cross Validation Time Series Split Backward
Sharpe Ratio
             K = 1    K = 2    K = 3    K = 4
----------  -------  -------  -------  -------
Only Long       0.68     0.32     0.05      0.5
Long Short      0.52     0.09    -0.3       0.32
```

[40]:
```
# Print all results
print('Mean Only Long Sharpe Ratio = ',round(np.mean(L_Sharpe_Ratios),2))
print('Mean Long Short Sharpe Ratio = ',round(np.mean(LS_Sharpe_Ratios),2))
```

```
Mean Only Long Sharpe Ratio =  0.39
Mean Long Short Sharpe Ratio =  0.16
```

17 Time Series Cross Validation Forwards

[41]:
```
#-------------------------------------------------------------------------#
#                               Full Dataset                              |
#-------------------------------------------------------------------------#
# t_0 <-----------------------------------------------------------> t_n #

# K = 1
#-------------------------------------------------------------------------#
#     Train     |     Test      |               |               |         |
#-------------------------------------------------------------------------#

# K = 2
#-------------------------------------------------------------------------#
#     Train     |     Train     |     Test      |               |         |
#-------------------------------------------------------------------------#

# K = 3
#-------------------------------------------------------------------------#
#     Train     |     Train     |     Train     |    Test       |         |
#-------------------------------------------------------------------------#

# K = 4
#-------------------------------------------------------------------------#
#     Train     |     Train     |     Train     |    Train      |     Test     |
#-------------------------------------------------------------------------#

# Create two empty lists
L_Sharpe_Ratios = []
LS_Sharpe_Ratios = []
seed(8)
tf.random.set_seed(8)

# For loop who create model, fit model, test model and compute performance and␣
 ↪Sharpe Ratio
for k in range (2,6):

    L1 = round(len(X)*0.2*k)
    L2 = round(len(X)*0.2)
    X_t = X[:L1]
    y_t = y[:L1]

    X_train = X_t[:-L2]
    X_test = X_t[-L2:]
    y_train = y_t[:-L2]
    y_test = y_t[-L2:]
```

```
    # Convert the X_train and y_train to numpy arrays
    X_train, y_train = np.array(X_train), np.array(y_train)

    # Reshape the data
    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
    # X_train.shape

    # Convert the data to a numpy array
    X_test = np.array(X_test)

    # Reshape the data
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1 ))

    model.fit(X_train, y_train, batch_size=1, epochs=5)

    # Get the models predicted price values
    predictions = model.predict(X_test)
    predictions = scaler.inverse_transform(predictions) # back-transform␣
↪Prediction

    y_pred = pd.DataFrame(predictions)

    y_r = y_pred.pct_change() # Predicted Returns from Test data
    y1 = (y_r > 0).astype(int) # Predicted low risk Signal

    n = len(y1)
    X2 = bco['Log_Returns']
    X2 = X2[len(X2)-n:]

    perf = y1.T.values * X2.values

    trading_days_Y = 5*52

    perf_sharpe = np.sqrt(trading_days_Y) * np.mean(perf) / (np.sqrt(np.
↪var(perf)))
    L_Sharpe_Ratios.append(perf_sharpe)

    y3 = np.sign(y_r)
    y3 = y3[1:]
    X2 = X2[1:]
    perf1 = y3.T.values * X2.values
    perf_sharpe1 = np.sqrt(trading_days_Y) * np.mean(perf1) / (np.sqrt(np.
↪var(perf1)))
    LS_Sharpe_Ratios.append(perf_sharpe1)
```

```
Epoch 1/5
```

```
721/721 [==============================] - 9s 12ms/step - loss: 0.0012
Epoch 2/5
721/721 [==============================] - 9s 12ms/step - loss: 7.8987e-04
Epoch 3/5
721/721 [==============================] - 9s 12ms/step - loss: 8.1135e-04
Epoch 4/5
721/721 [==============================] - 8s 12ms/step - loss: 8.5928e-04
Epoch 5/5
721/721 [==============================] - 9s 12ms/step - loss: 8.1746e-04
Epoch 1/5
1441/1441 [==============================] - 17s 12ms/step - loss: 6.8453e-04
Epoch 2/5
1441/1441 [==============================] - 17s 12ms/step - loss: 6.0277e-04
Epoch 3/5
1441/1441 [==============================] - 17s 12ms/step - loss: 6.6007e-04
Epoch 4/5
1441/1441 [==============================] - 17s 12ms/step - loss: 6.3290e-04
Epoch 5/5
1441/1441 [==============================] - 17s 12ms/step - loss: 6.4989e-04
Epoch 1/5
2162/2162 [==============================] - 25s 12ms/step - loss: 5.5654e-04
Epoch 2/5
2162/2162 [==============================] - 25s 12ms/step - loss: 5.0937e-04
Epoch 3/5
2162/2162 [==============================] - 25s 12ms/step - loss: 5.0513e-04
Epoch 4/5
2162/2162 [==============================] - 26s 12ms/step - loss: 5.0889e-04
Epoch 5/5
2162/2162 [==============================] - 25s 12ms/step - loss: 4.6957e-04
Epoch 1/5
2882/2882 [==============================] - 34s 12ms/step - loss: 4.3135e-04
Epoch 2/5
2882/2882 [==============================] - 36s 12ms/step - loss: 4.7135e-04
Epoch 3/5
2882/2882 [==============================] - 36s 13ms/step - loss: 3.7474e-04
Epoch 4/5
2882/2882 [==============================] - 34s 12ms/step - loss: 4.2062e-04
Epoch 5/5
2882/2882 [==============================] - 34s 12ms/step - loss: 3.9108e-04
```

[42]:
```
# Create data
data = [["Only Long", round(L_Sharpe_Ratios[0],2), round(L_Sharpe_Ratios[1],2),
         round(L_Sharpe_Ratios[2],2), round(L_Sharpe_Ratios[3],2)],
        ["Long Short", round(LS_Sharpe_Ratios[0],2),␣
↪round(LS_Sharpe_Ratios[1],2),
         round(LS_Sharpe_Ratios[2],2), round(LS_Sharpe_Ratios[3],2)]]
```

```
# Define header names
col_names = ["K = 1", "K = 2", "K = 3", "K = 4"]

# Display table
print('LSTM')
print('Cross Validation Time Series Split Forward')
print('Sharpe Ratio')
print(tabulate(data, headers=col_names))
```

```
LSTM
Cross Validation Time Series Split Forward
Sharpe Ratio
              K = 1    K = 2    K = 3    K = 4
----------  -------  -------  -------  -------
Only Long      0.84    -0.55     1.39     0.95
Long Short     0.84    -1.14     1.44     0.86
```

[43]:
```
# Print all results
print('Mean Only Long Sharpe Ratio = ',round(np.mean(L_Sharpe_Ratios),2))
print('Mean Long Short Sharpe Ratio = ',round(np.mean(LS_Sharpe_Ratios),2))
```

```
Mean Only Long Sharpe Ratio =  0.66
Mean Long Short Sharpe Ratio =  0.5
```

18 K-Fold Cross Validation

[44]:
```
#-------------------------------------------------------------------------#
#                              Full Dataset                               |
#-------------------------------------------------------------------------#
# t_0 <-------------------------------------------------------------> t_n #

# K = 1
#-------------------------------------------------------------------------#
#     Test     |     Train     |     Train     |     Train     |     Train    |
#-------------------------------------------------------------------------#

# K = 2
#-------------------------------------------------------------------------#
#     Train    |     Test      |     Train     |     Train     |     Train    |
#-------------------------------------------------------------------------#

# K = 3
#-------------------------------------------------------------------------#
#     Train    |     Train     |     Test      |     Train     |     Train    |
#-------------------------------------------------------------------------#

# K = 4
```

```
#-------------------------------------------------------------------------------#
#     Train     |     Train     |     Train     |     Test      |     Train     |
#-------------------------------------------------------------------------------#

# K = 5
#-------------------------------------------------------------------------------#
#     Train     |     Train     |     Train     |     Train     |     Test      |
#-------------------------------------------------------------------------------#

# Create three empty lists
L_Sharpe_Ratios = []
LS_Sharpe_Ratios = []
Acc_score = []
seed(8)
tf.random.set_seed(8)

# For loop who create model, fit model, test model and compute performance and␣
 ↪Sharpe Ratio
for k in range (1,6):

    L1 = int(round(len(X))*0.20*k)+1 # Geben
    L2 = int(round(len(X))*0.20*(k-1)) # Entfernen

    X_test = X[L2:L1]
    X_train = X.drop(index=X_test.index)

    y_test = y[L2:L1]
    y_train = y.drop(index=y_test.index)

    # Convert the X_train and y_train to numpy arrays
    X_train, y_train = np.array(X_train), np.array(y_train)

    # Reshape the data
    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
    # X_train.shape

    # Convert the data to a numpy array
    X_test = np.array(X_test)

    # Reshape the data
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1 ))

    model.fit(X_train, y_train, batch_size=1, epochs=5)

    # Get the models predicted price values
    predictions = model.predict(X_test)
```

```
    predictions = scaler.inverse_transform(predictions) # back-transform␣
↪Prediction

    y_pred = pd.DataFrame(predictions)

    y_r = y_pred.pct_change() # Predicted Returns from Test data
    y1 = (y_r > 0).astype(int) # Predicted low risk Signal

    n = len(y1)
    X2 = bco['Log_Returns']
    X2 = X2[len(X2)-n:]

    X1 = (X2 > 0).astype(int) # Real low risk Signal

    Acc_score.append(accuracy_score(X1, y1) * 100.0)

    perf = y1.T.values * X2.values

    trading_days_Y = 5*52

    perf_sharpe = np.sqrt(trading_days_Y) * np.mean(perf) / (np.sqrt(np.
↪var(perf)))
    L_Sharpe_Ratios.append(perf_sharpe)

    y3 = np.sign(y_r)
    y3 = y3[1:]
    X2 = X2[1:]
    perf1 = y3.T.values * X2.values
    perf_sharpe1 = np.sqrt(trading_days_Y) * np.mean(perf1) / (np.sqrt(np.
↪var(perf1)))
    LS_Sharpe_Ratios.append(perf_sharpe1)
```

```
Epoch 1/5
2881/2881 [==============================] - 34s 12ms/step - loss: 3.7226e-04
Epoch 2/5
2881/2881 [==============================] - 38s 13ms/step - loss: 3.4206e-04
Epoch 3/5
2881/2881 [==============================] - 42s 14ms/step - loss: 3.3735e-04
Epoch 4/5
2881/2881 [==============================] - 36s 12ms/step - loss: 3.3777e-04
Epoch 5/5
2881/2881 [==============================] - 36s 13ms/step - loss: 3.1846e-04
Epoch 1/5
2881/2881 [==============================] - 36s 13ms/step - loss: 0.0010
Epoch 2/5
2881/2881 [==============================] - 36s 13ms/step - loss: 3.3329e-04
Epoch 3/5
```

```
2881/2881 [==============================] - 34s 12ms/step - loss: 3.2206e-04
Epoch 4/5
2881/2881 [==============================] - 37s 13ms/step - loss: 3.3582e-04
Epoch 5/5
2881/2881 [==============================] - 38s 13ms/step - loss: 8.5800e-04
Epoch 1/5
2880/2880 [==============================] - 37s 13ms/step - loss: 4.5286e-04
Epoch 2/5
2880/2880 [==============================] - 37s 13ms/step - loss: 3.9500e-04
Epoch 3/5
2880/2880 [==============================] - 39s 14ms/step - loss: 3.2635e-04
Epoch 4/5
2880/2880 [==============================] - 42s 15ms/step - loss: 3.5287e-04
Epoch 5/5
2880/2880 [==============================] - 38s 13ms/step - loss: 3.4994e-04
Epoch 1/5
2881/2881 [==============================] - 39s 14ms/step - loss: 3.7702e-04
Epoch 2/5
2881/2881 [==============================] - 38s 13ms/step - loss: 3.5025e-04
Epoch 3/5
2881/2881 [==============================] - 38s 13ms/step - loss: 3.7199e-04
Epoch 4/5
2881/2881 [==============================] - 38s 13ms/step - loss: 3.6625e-04
Epoch 5/5
2881/2881 [==============================] - 38s 13ms/step - loss: 3.6976e-04
Epoch 1/5
2881/2881 [==============================] - 38s 13ms/step - loss: 2.9502e-04
Epoch 2/5
2881/2881 [==============================] - 38s 13ms/step - loss: 2.8370e-04
Epoch 3/5
2881/2881 [==============================] - 38s 13ms/step - loss: 2.8495e-04
Epoch 4/5
2881/2881 [==============================] - 39s 13ms/step - loss: 2.9425e-04
Epoch 5/5
2881/2881 [==============================] - 39s 13ms/step - loss: 2.7136e-04
```

[45]:
```
# Create data
data = [["Only Long", round(L_Sharpe_Ratios[0],2), round(L_Sharpe_Ratios[1],2),
         round(L_Sharpe_Ratios[2],2), round(L_Sharpe_Ratios[3],2),␣
↪round(L_Sharpe_Ratios[4],2)],
        ["Long Short", round(LS_Sharpe_Ratios[0],2),␣
↪round(LS_Sharpe_Ratios[1],2),
         round(LS_Sharpe_Ratios[2],2), round(LS_Sharpe_Ratios[3],2),␣
↪round(LS_Sharpe_Ratios[4],2)],
        ["Accuracy", round(Acc_score[0],2), round(Acc_score[1],2),
         round(Acc_score[2],2), round(Acc_score[3],2), round(Acc_score[4],2)]]
```

```
# Define header names
col_names = ["K = 1", "K = 2", "K = 3", "K = 4", "K = 5"]

# Display table
print('LSTM')
print('K-Fold Cross Validation')
print('Sharpe Ratio')
print(tabulate(data, headers=col_names))
```

```
LSTM
K-Fold Cross Validation
Sharpe Ratio
              K = 1    K = 2    K = 3    K = 4    K = 5
----------  -------  -------  -------  -------  -------
Only Long       0.1     1.24    -0.04     1.58     0.87
Long Short    -0.22     1.37    -0.41     1.81     0.72
Accuracy      48.13    52.01    48.34    53.12    49.65
```

[46]:
```
# Print all results
print('Mean Only Long Sharpe Ratio = ',round(np.mean(L_Sharpe_Ratios),2))
print('Mean Long Short Sharpe Ratio = ',round(np.mean(LS_Sharpe_Ratios),2))
print('Mean Accuracy Score = ',round(np.mean(Acc_score),2))
```

```
Mean Only Long Sharpe Ratio =  0.75
Mean Long Short Sharpe Ratio =  0.65
Mean Accuracy Score =  50.25
```

[ ]:
```

```

k-NN

June 4, 2022

1 k-NN model

1.0.1 Imports

[1]:
```
# Basic Imports
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
import seaborn as sns
from pprint import pprint
from matplotlib import pyplot
import yfinance as yf

# Machine Learning Imports
import sklearn
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import plot_roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import r2_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.inspection import permutation_importance
from tabulate import tabulate
from sklearn.model_selection import train_test_split
```

2 Load CSV File

[2]:
```
# From 16.08.2007 to 26.04.2022
bco = pd.read_csv('/Users/romaindeleze/Desktop/BSc/Romain_WI_TZ/8.Semester/BA/
 ↪Daten/Brent_Crude_Oil')
```

[3]:
```
bco.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3641 entries, 0 to 3640
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Date         3641 non-null   object
 1   Close        3641 non-null   float64
 2   Log_Returns  3641 non-null   float64
 3   Pred_Signal  3641 non-null   int64
 4   RSI          3641 non-null   float64
 5   K_percent    3641 non-null   float64
 6   MACD         3641 non-null   float64
 7   ROC          3641 non-null   float64
dtypes: float64(6), int64(1), object(1)
memory usage: 227.7+ KB
```

3 Building the K-nearst Neighbors Model

We have split our data into a training set and testing set, so we need to identify our input columns which are the following:

RSI, Stochastic Oscillator, Price Rate of Change, MACD

Those columns will serve as our X , and our Y column will be the Signal column, the column that specifies whether the future closed up or down compared to the previous day.

Once we've selected our columns, we need to split the data into a training and test set. Which will take our features and labels and split them based on the size we input. In our case, let's have the test_size be '20 % . After we've split the data, we can create our KNeighborsClassifier model. Once we've created it, we can fit the training data to the model using the fit method. Finally, with our "trained" model, we can make predictions. Take the X_test data set and use it to make predictions.

3.0.1 First we have to choose the label and the features we are looking for. Second step is to scale the data because the k-NN works with distance.

[4]:
```
# Choose of the features we want in the knn model
features = bco[['RSI','K_percent','ROC','MACD']]
labels = bco['Pred_Signal']

# We sacale the data with the MinMaxSacaler in values between 0 and 1
scaler = MinMaxScaler(feature_range=(0,1))
scaled_features = scaler.fit_transform(features)

# Fill features dataset with Scaled Data
features['RSI'] = scaled_features[:,0]
```

```
features['K_percent'] = scaled_features[:,1]
features['ROC'] = scaled_features[:,2]
features['MACD'] = scaled_features[:,3]
# The label is allready between 0 and 1
```

```
<ipython-input-4-4708a1e9adee>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  features['RSI'] = scaled_features[:,0]
<ipython-input-4-4708a1e9adee>:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  features['K_percent'] = scaled_features[:,1]
<ipython-input-4-4708a1e9adee>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  features['ROC'] = scaled_features[:,2]
<ipython-input-4-4708a1e9adee>:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  features['MACD'] = scaled_features[:,3]
```

[5]:
```
# Split the data in a 80%, 20% proportion
X_train, X_test, y_train, y_test = train_test_split(features,labels,␣
 ↪test_size=0.2, shuffle=False)

# Print lengths to see if its splitted right
print('Features lenght = ',len(features))
print('Label lenght = ',len(labels))
print()
print('X_train lenght = ',len(X_train))
print('y_train lenght = ',len(y_train))
print('X_test lenght = ',len(X_test))
print('y_test lenght = ',len(y_test))
print('Backtesting: ',len(X_train)+len(X_test))
```

```
print('Backtesting: ',len(y_train)+len(y_test))
```

```
Features lenght =  3641
Label lenght =  3641

X_train lenght =  2912
y_train lenght =  2912
X_test lenght =  729
y_test lenght =  729
Backtesting:  3641
Backtesting:  3641
```

3.1 Hyperparameter Tuning

Lets get first a closer look which parameter exists for knn classification models

[6]:
```
# Create KNN Object.
knn = KNeighborsClassifier()

# Look at parameters used by our current KNeighborsClassifier
pprint(knn.get_params())
```

```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 5,
 'p': 2,
 'weights': 'uniform'}
```

3.1.1 With the RandomizedSearchCV command we can achive the hyperparameter tuning

We have to choose which hyperparameter we want tune. Then we have to creat lists with the parameter values for which we are looking for. The RandomizedSearchCV command will compute all combinations and then we can access to the best parameter values.

[7]:
```
# List hyperparameters that we want to tune.

# Algorithm used to compute the nearest neighbors
algorithm = ['auto','brute'] # 2 Possibilities

# number of each leaf size
leaf_size = list(range(1,50)) # 500 Possibilities

# Distanz metrics, 1 = Manhattan distanz, 2 = Euclidian distanz
p = [1,2] # 2 Possibilities
```

```
# How weighted are the data points
weights = ['uniform', 'distance'] # 2 Possibilities

# number of neighbors
n_neighbors = list(range(4,50)) # 200 Possibilities


# Create the random grid
random_grid = {'algorithm':algorithm,
               'leaf_size':leaf_size,
               'p':p,
               'weights':weights,
               'n_neighbors':n_neighbors}
```

[8]:
```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
knn_2 = KNeighborsClassifier()

# Random search of parameters, using 3 fold cross validation,
# search across 300 different combinations, and use all available cores
knn_random = RandomizedSearchCV(estimator = knn_2, param_distributions =␣
↪random_grid, n_iter = 100,
                                cv = 3, verbose = 1, random_state = 8, n_jobs =␣
↪1)

# Fit the random search model
knn_random.fit(X_train, y_train) # The optimisation may take a few minutes
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

[8]:
```
RandomizedSearchCV(cv=3, estimator=KNeighborsClassifier(), n_iter=100, n_jobs=1,
                   param_distributions={'algorithm': ['auto', 'brute'],
                                        'leaf_size': [1, 2, 3, 4, 5, 6, 7, 8, 9,
                                                      10, 11, 12, 13, 14, 15,
                                                      16, 17, 18, 19, 20, 21,
                                                      22, 23, 24, 25, 26, 27,
                                                      28, 29, 30, …],
                                        'n_neighbors': [4, 5, 6, 7, 8, 9, 10,
                                                        11, 12, 13, 14, 15, 16,
                                                        17, 18, 19, 20, 21, 22,
                                                        23, 24, 25, 26, 27, 28,
                                                        29, 30, 31, 32, 33,
…],
                                        'p': [1, 2],
                                        'weights': ['uniform', 'distance']},
                   random_state=8, verbose=1)
```

[9]:
```
# This is the Output command to see the best Parameters
param = knn_random.best_params_
print(param)
```

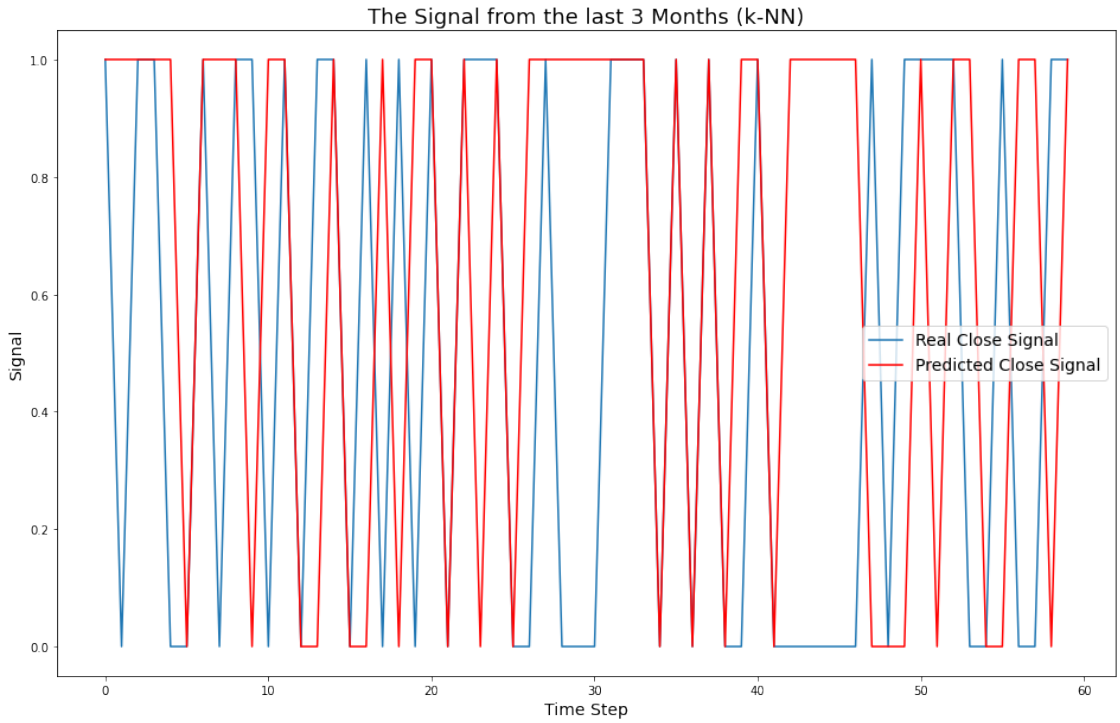
```
{'weights': 'distance', 'p': 1, 'n_neighbors': 5, 'leaf_size': 15, 'algorithm':
'auto'}
```

3.2 After the optimisation we can creat the k-NN model and make the predictions

[10]:
```
# Creat best knn model
best_knn = KNeighborsClassifier(weights = param['weights'],
                                p = param['p'],
                                n_neighbors = param['n_neighbors'],
                                leaf_size = param['leaf_size'],
                                algorithm = param['algorithm'])

# Fit the model with the training dataset
best_knn.fit(X_train, y_train)

# Predict test data set.
y_pred = best_knn.predict(X_test)
```

[11]:
```
# Lets see the actual and the predicted Signal from the last 3 months
m3 = 3*4*5
plt.figure(figsize=(16,10))
plt.title('The Signal from the last 3 Months (k-NN)', fontsize = 18)
plt.plot(y_test[len(y_test)-m3:].values, label = 'Real Close Signal')
plt.plot(y_pred[len(y_pred)-m3:], color = 'red', label = 'Predicted Close␣
↪Signal')
plt.ylabel('Signal', fontsize = 14)
plt.xlabel('Time Step', fontsize = 14)
plt.legend(loc='best', fontsize = 14)
```

[11]:
```
<matplotlib.legend.Legend at 0x7fe231b25b20>
```



4 Accuracy

We've built our model, so now we can see how accurate it is. SciKit learn, makes the process of evaluating our model very easy by providing a bunch of built-in metrics that we can call. One of those metrics is the accuracy_score. The accuracy_score function computes the accuracy, by calculating the sum of the correctly predicted signals and then dividing it by the total number of predictions. Imagine we had three TRUE values [1, 2, 3] , and our model predicted the following values [1, 2, 4] we would say the accuracy of our model is $2/3 = 66\ \%$.

5 AUC

AUC curve is a performance measurement for the classification problems at various threshold settings. AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1. By analogy, the Higher the AUC, the better the model is at distinguishing between patients with the disease and no disease.

6 RMSE

Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are;

RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit.

[12]:
```
# Print Accuracy
print('Accuracy (%): ', accuracy_score(y_test, y_pred, normalize = True) * 100.
 ↪0)
print()

# Checking performance our model with AUC Score.
print('Area under the curve: ',roc_auc_score(y_test, y_pred))
print()

# Get the root mean squared error (RMSE)
print('RMSE = ' , np.sqrt(np.mean(((y_pred - y_test) ** 2))))
# Don't forget the worst value in this case is 1!
```

```
Accuracy (%):  52.94924554183813

Area under the curve:  0.5260281162992377

RMSE =  0.6859355250908198
```

7 Classification Report

To get a more detailed overview of how the model performed, we can build a classification report that will compute the F1_Score , the Precision , the Recall , and the Support .

7.0.1 Precision

Precision measures the proportion of all correctly identified samples in a population of samples which are classified as positive labels and is defined as the following:

tp = True Positiv

fp = False Positiv

Precision = tp/(tp+fp)

The precision is intuitively the ability of the classifier not to label as positive a sample that is negative. The best value is 1, and the worst value is 0.

7.0.2 Recall

Recall (also known as sensitivity) measures the ability of a classifier to correctly identify positive labels and is defined as the following:

tp = True Positiv

fn = False Negativ

Recall = tp/(tp+fn)

The recall is intuitively the ability of the classifier to find all the positive samples. The best value is 1, and the worst value is 0.

7.0.3 Support

Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing. Support doesn't change between models but instead diagnoses the evaluation process.

7.0.4 F1 Score

In some cases, we will have models that may have low precision or high recall. It's difficult to compare two models with low precision and high recall or vice versa. To make results comparable, we use a metric called the F-Score. The F-score helps to measure Recall and Precision at the same time. It uses Harmonic Mean in place of Arithmetic Mean by punishing the extreme values more.

The traditional F-measure or balanced F-score (F1 score) is the harmonic mean of precision and recall:

tp = True Positiv

fp = False Positiv

fn = False Negativ

F1 Score = tp/(tp+0.5*(fp+fn))

[13]:
```
# Define the traget names
target_names = ['Down Day', 'Up Day']

# Build a classifcation report
report = classification_report(y_true = y_test, y_pred = y_pred, target_names =␣
↪target_names, output_dict = True)

# Add it to a data frame, transpose it for readability.
report_df = pd.DataFrame(report).transpose()
report_df
```

[13]:

	precision	recall	f1-score	support
Down Day	0.476331	0.492355	0.484211	327.000000
Up Day	0.575448	0.559701	0.567465	402.000000

```
accuracy       0.529492  0.529492  0.529492    0.529492
macro avg      0.525889  0.526028  0.525838  729.000000
weighted avg   0.530988  0.529492  0.530121  729.000000
```

8 Confusion Matrix

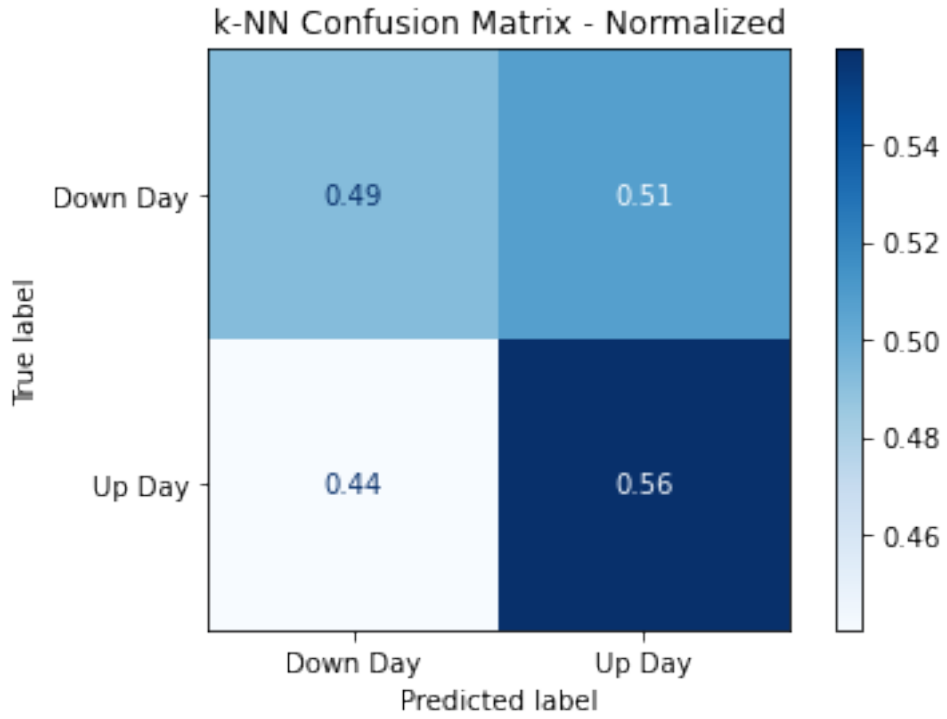
A confusion matrix is a technique for summarizing the performance of a classification algorithm. Classification accuracy alone can be misleading if you have an unequal number of observations in each class or if you have more than two classes in your dataset. Calculating a confusion matrix can give you a better idea of what your classification model is getting right and what types of errors it is making.

[14]:
```
# Compute Confusion Matrix
knn_matrix = confusion_matrix(y_test, y_pred)

# Save individual values
true_negatives = knn_matrix[0][0]
false_negatives = knn_matrix[1][0]
true_positives = knn_matrix[1][1]
false_positives = knn_matrix[0][1]

# Compute performance values
accuracy = (true_negatives + true_positives) / (true_negatives + true_positives␣
↪+ false_negatives + false_positives)
percision = true_positives / (true_positives + false_positives)
recall = true_positives / (true_positives + false_negatives)
specificity = true_negatives / (true_negatives + false_positives)

# Print values
print('Accuracy: {}'.format(float(accuracy)))
print('Percision: {}'.format(float(percision)))
print('Recall: {}'.format(float(recall)))
print('Specificity: {}'.format(float(specificity)))

# Visualise the Confusion Matrix
disp = plot_confusion_matrix(best_knn, X_test, y_test, display_labels = ['Down␣
↪Day', 'Up Day']
                             , normalize = 'true', cmap=plt.cm.Blues)
disp.ax_.set_title('k-NN Confusion Matrix - Normalized')
plt.show()
```

```
Accuracy: 0.5294924554183813
Percision: 0.5754475703324808
Recall: 0.5597014925373134
Specificity: 0.4923547400611621
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function plot_confusion_matrix is deprecated; Function
```

```
`plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one
of the class methods: ConfusionMatrixDisplay.from_predictions or
ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



8.0.1 With the information from the confusion matrix we want know now how the proportion is.

How is the prediction distributed, and how ist the real values distributed

[15]:
```
# Propotion of the real values
unique, counts = np.unique(y_test, return_counts=True)
print('Real Proportion')
print(dict(zip(unique, counts/len(y_test))))
print()

# Propotion of the predicted values
unique, counts = np.unique(y_pred, return_counts=True)
print('Predicted Proportion')
print(dict(zip(unique, counts/len(y_pred))))
```

```
Real Proportion
{0: 0.448559670781893, 1: 0.551440329218107}

Predicted Proportion
```

```
{0: 0.46364883401920437, 1: 0.5363511659807956}
```

9 Feature Importance

With any model, you want to have an idea of what features are helping explain most of the model, as this can give you insight as to why you're getting the results you are. With Random Forest, we can identify some of our most important features or, in other words, the features that help explain most of the model. In some cases, some of our features might not be very important, or in other words, when compared to additional features, don't explain much of the model.

9.0.1 Why Do We Care About Feature Importance?

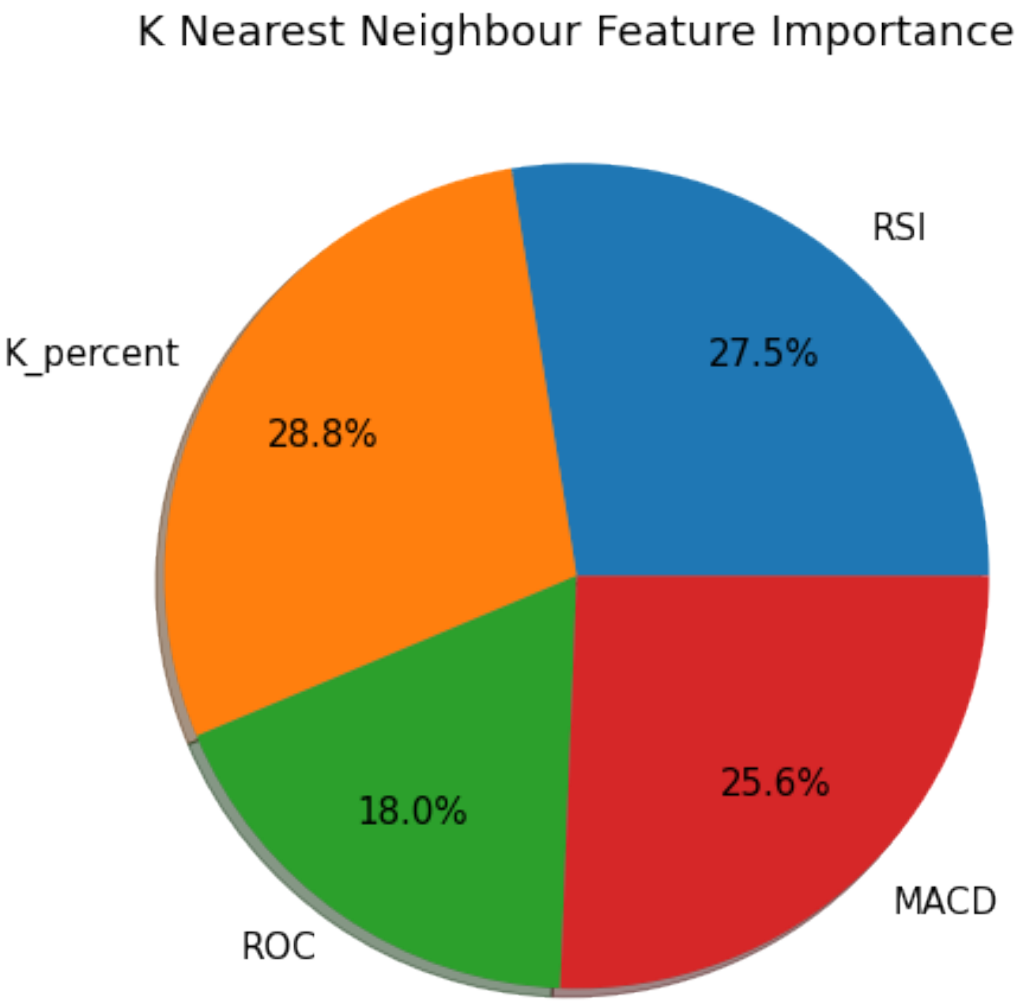
What that means is if we were to get rid of those features, our accuracy will go down a little, hopefully, but not significantly. You might be asking, "Why would I want to get rid of a feature if it lowers my accuracy?" Well, it depends, in some cases, you don't care if your model is 95% accurate or 92% accurate. To you, a 92% accurate model is just as good as a 95% accurate model. However, if you wanted to get a 95% accurate model, you would, in this hypothetical case, have to train your model twice as long. Now, I'm a little extreme in this case, but the idea is the same. The cost doesn't justify the benefit. In the real world, we have to make these decisions all the time, and in some cases, it just doesn't warrant the extra cost for such a minimal increase in the accuracy.

9.0.2 Calculating the Feature Importance

Like all the previous steps, SkLearn makes this process very easy. Take your ML model and call the permutation_importance. This will return all of our features and their importance measurement.

[16]:
```
# perform permutation importance
results = permutation_importance(best_knn, X_train, y_train)

# get importance for accuracy
importance = results.importances_mean

# Create Pie Plot
imp_scale = sum(importance)
imp = importance/imp_scale
def func(pct, allvalues):
    return "{:.1f}%".format(pct)
plt.figure(figsize=(8,8))
plt.title('K Nearest Neighbour Feature Importance', fontsize = 18)
plt.pie(imp, labels=['RSI','K_percent','ROC','MACD'], autopct = lambda pct:␣
 ↪func(pct, imp),
        shadow=True, pctdistance=.7, textprops={'fontsize': 15})
```

```
[16]: ([<matplotlib.patches.Wedge at 0x7fe23196a9a0>,
         <matplotlib.patches.Wedge at 0x7fe231976310>,
         <matplotlib.patches.Wedge at 0x7fe231976be0>,
         <matplotlib.patches.Wedge at 0x7fe2319844f0>],
        [Text(0.7133705448611826, 0.8373186166116574, 'RSI'),
         Text(-0.9619965256097449, 0.5334441720693737, 'K_percent'),
         Text(-0.6265813906909288, -0.9040994197762887, 'ROC'),
         Text(0.7621254633576059, -0.7931990784802733, 'MACD')],
        [Text(0.45396307400257063, 0.5328391196619637, '27.5%'),
         Text(-0.6121796072062012, 0.33946447313505596, '28.8%'),
         Text(-0.3987336122578637, -0.5753359944030927, '18.0%'),
         Text(0.4849889312275673, -0.5047630499419921, '25.6%')])
```

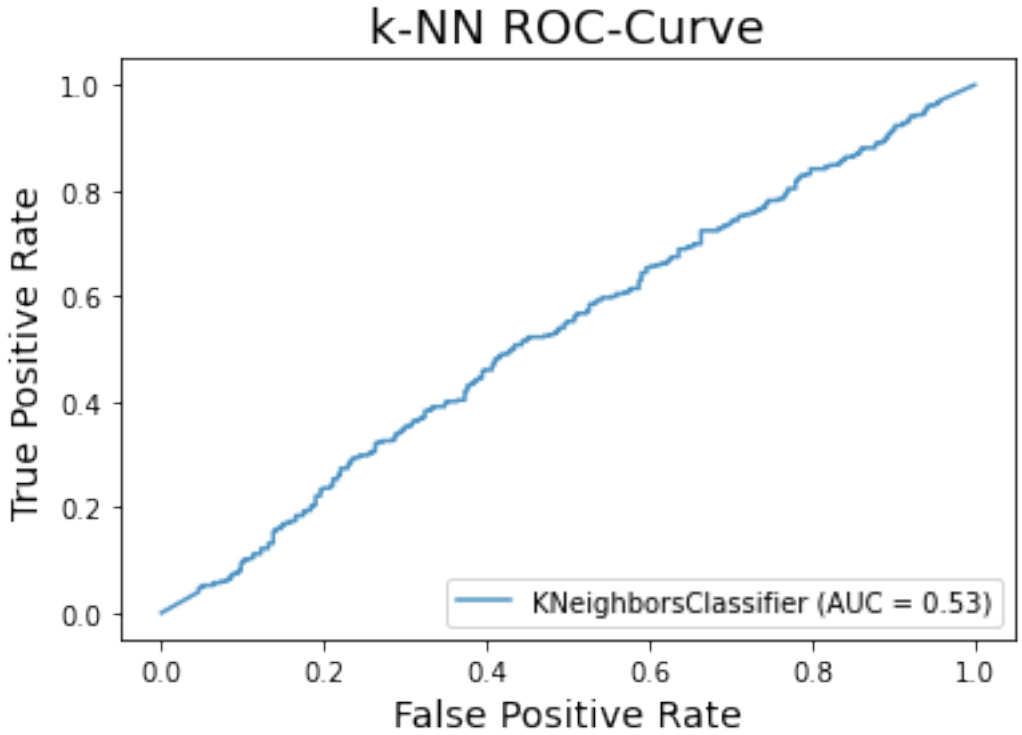


10 ROC Curve

The Receiver Operating Characteristic is a graphical method to evaluate the performance of a binary classifier. A curve is drawn by plotting True Positive Rate (sensitivity) against False Positive Rate (1 - specificity) at various threshold values. ROC curve shows the trade-off between sensitivity and specificity. When the curve comes closer to the left-hand border and the top border of the ROC space, it indicates that the test is accurate. The closer the curve is to the top and left-hand border, the more accurate the test is. If the curve is close to the 45 degrees diagonal of the ROC space, it means that the test is not accurate. ROC curves can be used to select the optimal model and discard the suboptimal ones.

[17]:
```
# Create an ROC Curve plot.
knn_disp = plot_roc_curve(best_knn, X_test, y_test, alpha = 0.8)
plt.title('k-NN ROC-Curve', fontsize=18)
plt.xlabel('False Positive Rate', fontsize=14)
plt.ylabel('True Positive Rate', fontsize=14)
plt.show()
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function plot_roc_curve is deprecated; Function
:func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one
of the class methods: :meth:`sklearn.metric.RocCurveDisplay.from_predictions` or
:meth:`sklearn.metric.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```



11 Trading Performance

[18]:
```
# Define the length of the prediction
n = len(y_pred)
X = bco['Log_Returns'].shift(-1)
X = X[len(X)-n:]
len(X)
```

[18]: 729

11.1 Long

[19]:
```
# Compute the Only Long Performance: Signal * Log Return
perf = y_pred * X
perf = perf.dropna()

trading_days_Y = 5*52

perf_sharpe = np.sqrt(trading_days_Y) * np.mean(perf) / (np.sqrt(np.var(perf)))
```

11.2 Long-Short

[20]:
```
# Transform 0,1 Signal in a -1,1 Signal
y_pred1 = []
for i in range(0,len(y_pred)):
    if y_pred[i] == 0:
        y_pred1.append(-1)
    else:
        y_pred1.append(1)
```

[21]:
```
# Compute the Long Short Performance: Signal * Log Return
perf1 = y_pred1 * X
perf1 = perf1.dropna()

trading_days_Y = 5*52

perf_sharpe1 = np.sqrt(trading_days_Y) * np.mean(perf1) / (np.sqrt(np.
↪var(perf1)))
```

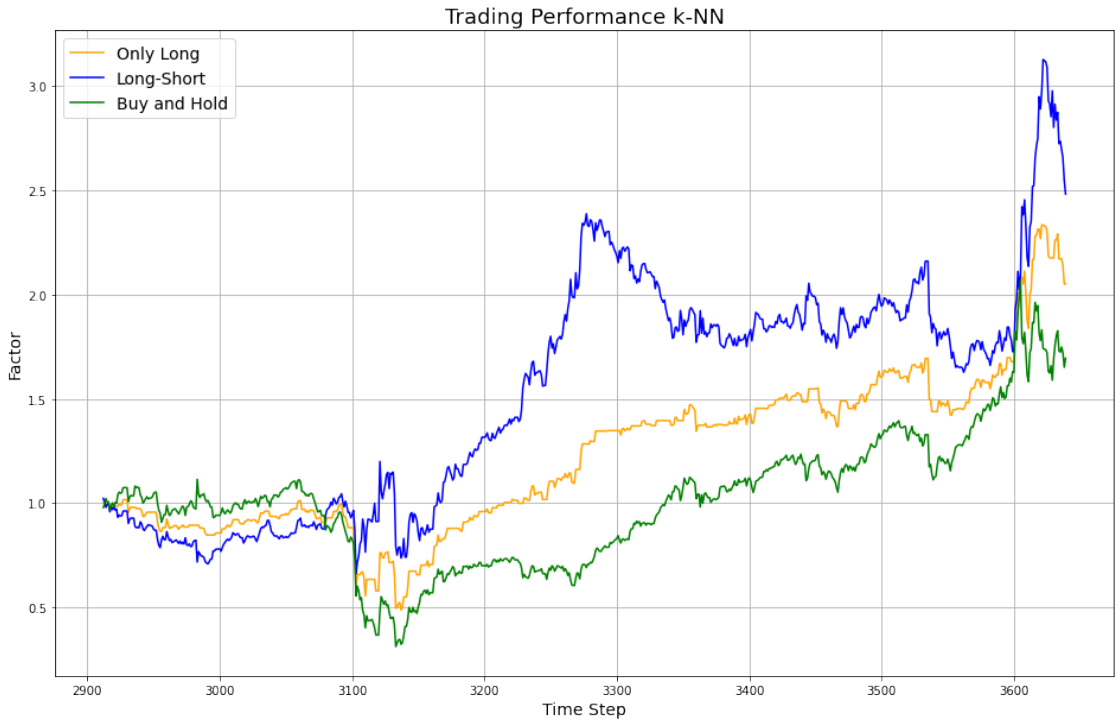
11.3 Buy and Hold

[22]:
```
perf2 = X
perf2 = perf2.dropna()

trading_days_Y = 5*52

perf_sharpe2 = np.sqrt(trading_days_Y) * np.mean(perf2) / (np.sqrt(np.
 ↪var(perf2)))
```

12 Comparison ML vs. Buy and Hold

[23]:
```
# Lets compare the performances
plt.figure(figsize=(16,10))
plt.title('Trading Performance k-NN', fontsize = 18)
plt.ylabel('Factor', fontsize = 14)
plt.xlabel('Time Step', fontsize = 14)
plt.grid(True)
plt.plot(np.exp(perf).cumprod() , label='Only Long', color = 'orange')
plt.plot(np.exp(perf1).cumprod() , label='Long-Short', color = 'blue')
plt.plot(np.exp(perf2).cumprod() , label='Buy and Hold', color = 'green')
plt.legend(loc='best', fontsize = 14)
```

[23]:
```
<matplotlib.legend.Legend at 0x7fe230effa00>
```



13 Profit Factor

[24]:
```
# Profit Factor is the last value of the performance computation
L_profit = np.exp(perf).cumprod().iloc[-1]

LS_profit = np.exp(perf1).cumprod().iloc[-1]

BaH_profit = np.exp(perf2).cumprod().iloc[-1]
```

14 Performance Table

[25]:
```
# Create data
data = [["Buy and Hold", round(perf_sharpe2,2), round(BaH_profit,2)],
        ["Only Long", round(perf_sharpe,2), round(L_profit,2)],
        ["Long Short", round(perf_sharpe1,2), round(LS_profit,2)]]

# Define header names
col_names = ["Strategie", "Sharp Ratio", "Profit Factor"]

# Display table
print('k-NN Classification')
print(tabulate(data, headers=col_names))
```

```
k-NN Classification
Strategie       Sharp Ratio    Profit Factor
------------  -------------  ---------------
Buy and Hold           0.35             1.69
Only Long              0.57             2.05
Long Short             0.6              2.48
```

[26]:
```
# Performance Dataset
# Compute performance
BaH_Perf = np.exp(perf2).cumprod()
knn_Perf_OL = np.exp(perf).cumprod()
knn_Perf_LS = np.exp(perf1).cumprod()

# Load in a Data Frame
knn_Perf_data = pd.DataFrame()
knn_Perf_data['BaH_Perf'] = BaH_Perf
knn_Perf_data['knn_Perf_OL'] = knn_Perf_OL
knn_Perf_data['knn_Perf_LS'] = knn_Perf_LS

# Create a csv File
```

```
knn_Perf_data.to_csv('/Users/romaindeleze/Desktop/BSc/Romain_WI_TZ/8.Semester/
↪BA/Daten/knn_Perf_data')
```

15 Extrem and Regular Value Accuracy

[27]:
```
# Creat Data Frame with Predicted Signal and Log Returns
n = len(y_pred)
X = bco['Log_Returns'].shift(-1)
X = X[len(X)-n:]
d = {'Predicted Signal':y_pred, 'Log Returns':X}
df = pd.DataFrame(data = d)
```

[28]:
```
# Creat the upperbound and the lowerbound
mu = np.mean(X)
sig = np.std(X)
lb = mu-2*sig
ub = mu+2*sig

# Creat Data Frame with only Values in the interval [lb,ub]
reg_val = df[(df['Log Returns'] >= lb) & (df['Log Returns'] <= ub)] # DataFrame␣
↪with Regular Values = [lb,ub]

# Creat Data Frame with only Values outside the interval [lb,ub]
ext_val = df.drop(index=reg_val.index)

# Add real Signal with the Log Returns for each Data Frame
reg_val['Real Signal'] = (reg_val['Log Returns'] > 0).astype(int)
ext_val['Real Signal'] = (ext_val['Log Returns'] > 0).astype(int)

# Compute accuracy of the Extrem Values and the Regular Values
acc_reg = accuracy_score(reg_val['Real Signal'], reg_val['Predicted Signal'],␣
↪normalize = True) * 100.0
acc_ext = accuracy_score(ext_val['Real Signal'], ext_val['Predicted Signal'],␣
↪normalize = True) * 100.0
print('Accuracy Regular Values = ', round(acc_reg,2))
print('Accuracy Exrtem Values (95%) = ', round(acc_ext,2))
```

```
Accuracy Regular Values =  52.53
Accuracy Exrtem Values (95%) =  58.33
```

```
<ipython-input-28-f97868f112e9>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  reg_val['Real Signal'] = (reg_val['Log Returns'] > 0).astype(int)
```

[29]:
```
# Creat the upperbound and the lowerbound
mu = np.mean(X)
sig = np.std(X)
lb = mu-3*sig
ub = mu+3*sig

# Creat Data Frame with only Values in the interval [lb,ub]
reg_val = df[(df['Log Returns'] >= lb) & (df['Log Returns'] <= ub)] # DataFrame␣
↪with Regular Values = [lb,ub]

# Creat Data Frame with only Values outside the interval [lb,ub]
ext_val = df.drop(index=reg_val.index)

# Add real Signal with the Log Returns for each Data Frame
reg_val['Real Signal'] = (reg_val['Log Returns'] > 0).astype(int)
ext_val['Real Signal'] = (ext_val['Log Returns'] > 0).astype(int)

# Compute accuracy of the Extrem Values and the Regular Values
acc_reg = accuracy_score(reg_val['Real Signal'], reg_val['Predicted Signal'],␣
↪normalize = True) * 100.0
acc_ext = accuracy_score(ext_val['Real Signal'], ext_val['Predicted Signal'],␣
↪normalize = True) * 100.0
print('Accuracy Regular Values = ', round(acc_reg,2))
print('Accuracy Exrtem Values (99%) = ', round(acc_ext,2))
```

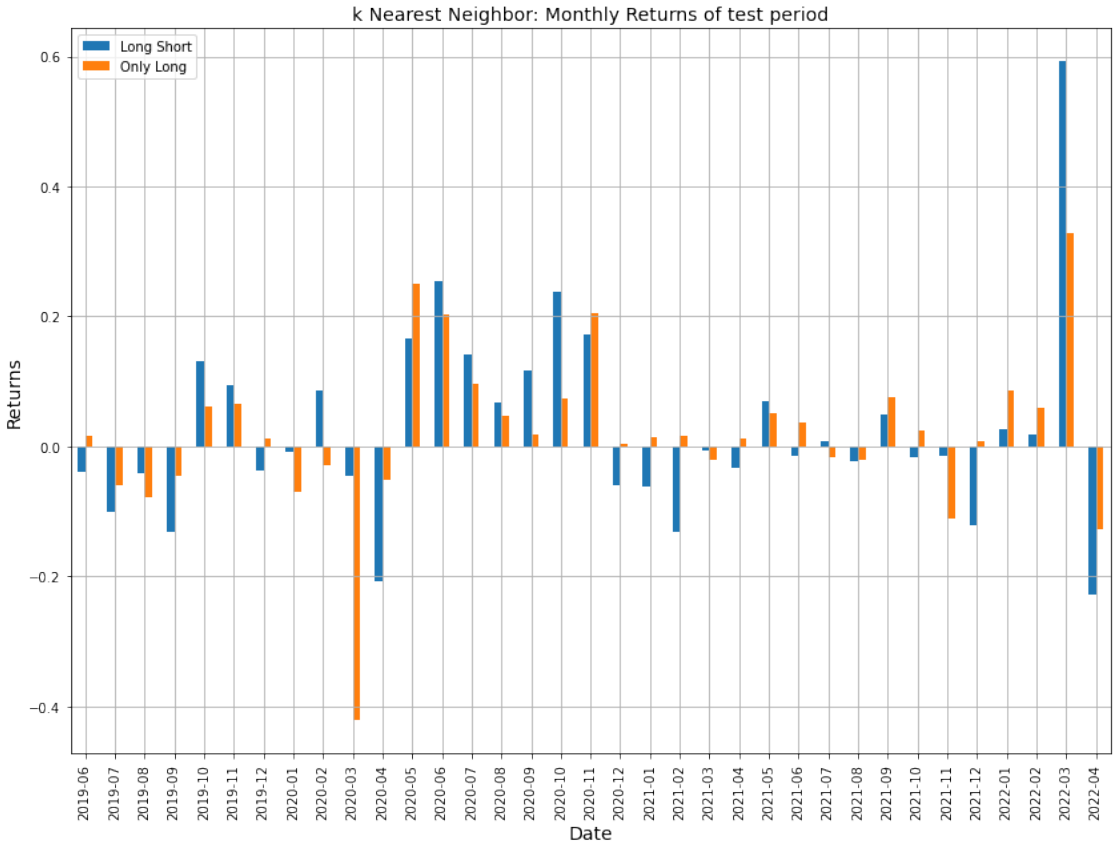
```
Accuracy Regular Values =  52.94
Accuracy Exrtem Values (99%) =  46.67
```

```
<ipython-input-29-a883c4847048>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  reg_val['Real Signal'] = (reg_val['Log Returns'] > 0).astype(int)
```

[30]:
```
# Indexing
dat = yf.download('BZ=F', start='2007-08-16',end = "2022-04-26",period='1d')
end_len = len(dat["Close"])
long_short = np.exp(perf1).cumprod()
start_len = end_len - len(long_short)
index = dat["Close"][start_len:end_len].index
perf1.index = index
perf.index = index

# Create Monthly Returns Long Short
da = perf1.groupby([(perf1.index.year),(perf1.index.month)]).sum()
```

```
monthly_ret_LS = da.values
n = len(monthly_ret_LS)
monthly_ret_LS = pd.DataFrame(monthly_ret_LS,index = pd.date_range(start='6/1/
 ↪2019', freq='M', periods=n))

# Create Monthly Returns Long
da1 = perf.groupby([(perf.index.year),(perf.index.month)]).sum()
monthly_ret_OL = da1.values
monthly_ret_OL = pd.DataFrame(monthly_ret_OL,index = pd.date_range(start='6/1/
 ↪2019', freq='M', periods=n))

# Connect Date Frames
mon_ret = pd.concat([monthly_ret_LS, monthly_ret_OL], axis=1)
mon_ret.columns = ["Long Short","Only Long"]

# x Axis
x_ax = pd.period_range('6/1/2019', '2022-04-30', freq='M').strftime('%Y-%m')
mon_ret.index = x_ax




# Plot the barplot
ax = mon_ret.plot.bar(figsize=(14,10), rot = 0)
plt.title("k Nearest Neighbor: Monthly Returns of test period",fontsize=14)
plt.xlabel('Date', fontsize=14)
plt.xticks(rotation=90)
plt.ylabel('Returns', fontsize=14)
plt.legend(loc='best')
plt.grid()
```

```
[**********************100%**********************]  1 of 1 completed
```



[31]:
```
# Lets see which are the MDD and Maximum Profit grouped by months
max_profit_LS = np.max(da)
max_prift_OL = np.max(da1)
MDD_LS = np.min(da)
MDD_OL = np.min(da1)

# Create data
data = [["Only Long", round(MDD_OL,2), round(max_prift_OL,2)],
["Long Short", round(MDD_LS,2), round(max_profit_LS,2)]]

# Define header names
col_names = ["Strategie", "MDD", "Max. Profit"]

# Display table
print('MDD and MAX.Profti of k-NN')
print(tabulate(data, headers=col_names))
```

```
MDD and MAX.Profti of k-NN
Strategie      MDD    Max. Profit
-----------  -----  -------------
Only Long    -0.42           0.33
```

```
Long Short   -0.23           0.59
```

16 Time Series Cross Validation Backwards

[32]:
```
#-------------------------------------------------------------------------#
#                              Full Dataset                               |
#-------------------------------------------------------------------------#
# t_0 <-------------------------------------------------------------> t_n #

# K = 1
#-------------------------------------------------------------------------#
#              |               |              |    Train      |     Test     |
#-------------------------------------------------------------------------#

# K = 2
#-------------------------------------------------------------------------#
#              |               |     Train    |    Train      |     Test     |
#-------------------------------------------------------------------------#

# K = 3
#-------------------------------------------------------------------------#
#              |     Train     |     Train    |    Train      |     Test     |
#-------------------------------------------------------------------------#

# K = 4
#-------------------------------------------------------------------------#
#     Train    |     Train     |     Train    |    Train      |     Test     |
#-------------------------------------------------------------------------#

# Create two empty lists
L_Sharpe_Ratios = []
LS_Sharpe_Ratios = []

# For loop who create model, fit model, test model and compute performance and␣
 ↪Sharpe Ratio
for k in range (1,5):

    X_train, X_test, y_train, y_test = train_test_split(features,labels,␣
 ↪test_size=0.2, shuffle=False)

    for i in range(1,2):
        feature_len = int(round(len(X_train))*0.25*k)
        X_train = X_train[-feature_len:]
        label_len = int(round(len(y_train))*0.25*k)
        y_train = y_train[-label_len:]
```

```
        knn_random.fit(X_train, y_train)
        param = knn_random.best_params_
        best_knn = KNeighborsClassifier(weights = param['weights'],
                                        p = param['p'],
                                        n_neighbors = param['n_neighbors'],
                                        leaf_size = param['leaf_size'],
                                        algorithm = param['algorithm'])

        # Fit the data to the model
        best_knn.fit(X_train, y_train)

        # Make predictions
        y_pred = best_knn.predict(X_test)

        # Trading Performance
        n = len(y_pred)
        X = bco['Log_Returns'].shift(-1)
        X = X[len(X)-n:]

        trading_days_Y = 5*52

        # Only Long
        perf = y_pred * X
        perf = perf.dropna()
        # Sharpe
        perf_sharpe = np.sqrt(trading_days_Y) * np.mean(perf) / (np.sqrt(np.
↪var(perf)))
        L_Sharpe_Ratios.append(perf_sharpe)

        # Long Short
        # Transform 0,1 Signal in a -1,1 Signal
        y_pred1 = []
        for i in range(0,len(y_pred)):
            if y_pred[i] == 0:
                y_pred1.append(-1)
            else:
                y_pred1.append(1)

        perf1 = y_pred1 * X
        perf1 = perf1.dropna()
        # Sharpe
        perf_sharpe1 = np.sqrt(trading_days_Y) * np.mean(perf1) / (np.sqrt(np.
↪var(perf1)))
        LS_Sharpe_Ratios.append(perf_sharpe1)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

[33]:
```
# Create data
data = [["Only Long", round(L_Sharpe_Ratios[0],2), round(L_Sharpe_Ratios[1],2),
         round(L_Sharpe_Ratios[2],2), round(L_Sharpe_Ratios[3],2)],
        ["Long Short", round(LS_Sharpe_Ratios[0],2),␣
 ↪round(LS_Sharpe_Ratios[1],2),
         round(LS_Sharpe_Ratios[2],2), round(LS_Sharpe_Ratios[3],2)]]

# Define header names
col_names = ["K = 1", "K = 2", "K = 3", "K = 4"]

# Display table
print('k Nearest Neighbour')
print('Cross Validation Time Series Split Backward')
print('Sharpe Ratio')
print(tabulate(data, headers=col_names))
```

```
k Nearest Neighbour
Cross Validation Time Series Split Backward
Sharpe Ratio
              K = 1    K = 2    K = 3    K = 4
----------  -------  -------  -------  -------
Only Long       1.1    -0.33     0.65     0.57
Long Short      1.13   -0.79     0.46     0.6
```

[34]:
```
# Print all results
print('Mean Only Long Sharpe Ratio = ',round(np.mean(L_Sharpe_Ratios),2))
print('Mean Long Short Sharpe Ratio = ',round(np.mean(LS_Sharpe_Ratios),2))
```

```
Mean Only Long Sharpe Ratio =  0.5
Mean Long Short Sharpe Ratio =  0.35
```

17 Time Series Cross Validation Forwards

[35]:
```
#-------------------------------------------------------------------------#
#                              Full Dataset                               |
#-------------------------------------------------------------------------#
# t_0 <-------------------------------------------------------------> t_n #

# K = 1
#-------------------------------------------------------------------------#
#     Train    |     Test      |               |               |           |
#-------------------------------------------------------------------------#

# K = 2
```

```
#-------------------------------------------------------------------------------#
#     Train     |     Train     |     Test     |                |               |
#-------------------------------------------------------------------------------#

# K = 3
#-------------------------------------------------------------------------------#
#     Train     |     Train     |     Train    |    Test        |               |
#-------------------------------------------------------------------------------#

# K = 4
#-------------------------------------------------------------------------------#
#     Train     |     Train     |     Train    |    Train       |     Test      |
#-------------------------------------------------------------------------------#

# Create two empty lists
L_Sharpe_Ratios = []
LS_Sharpe_Ratios = []

# For loop who create model, fit model, test model and compute performance and␣
 ↪Sharpe Ratio
for k in range (2,6):

    feature_len = int(round(len(features))*0.2*k)
    label_len = int(round(len(labels))*0.2*k)

    features1 = features[:feature_len]
    labels1 = labels[:label_len]

    X_train = features1[:-729]
    X_test = features1[-729:]
    y_train = labels1[:-729]
    y_test = labels1[-729:]


    for i in range(1,2):
        knn_random.fit(X_train, y_train)
        param = knn_random.best_params_
        best_knn = KNeighborsClassifier(weights = param['weights'],
                                        p = param['p'],
                                        n_neighbors = param['n_neighbors'],
                                        leaf_size = param['leaf_size'],
                                        algorithm = param['algorithm'])

        # Fit the data to the model
        best_knn.fit(X_train, y_train)

        # Make predictions
```

```
        y_pred = best_knn.predict(X_test)

        # Trading Performance
        X = bco['Log_Returns'].shift(-1)
        X = X[:label_len]
        X = X[-729:]

        trading_days_Y = 5*52

        # Only Long
        perf = y_pred * X
        perf = perf.dropna()
        # Sharpe
        perf_sharpe = np.sqrt(trading_days_Y) * np.mean(perf) / (np.sqrt(np.
↪var(perf)))
        L_Sharpe_Ratios.append(perf_sharpe)

        # Long Short
        # Transform 0,1 Signal in a -1,1 Signal
        y_pred1 = []
        for i in range(0,len(y_pred)):
            if y_pred[i] == 0:
                y_pred1.append(-1)
            else:
                y_pred1.append(1)

        perf1 = y_pred1 * X
        perf1 = perf1.dropna()
        # Sharpe
        perf_sharpe1 = np.sqrt(trading_days_Y) * np.mean(perf1) / (np.sqrt(np.
↪var(perf1)))
        LS_Sharpe_Ratios.append(perf_sharpe1)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
Fitting 3 folds for each of 100 candidates, totalling 300 fits
Fitting 3 folds for each of 100 candidates, totalling 300 fits
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

[36]:
```
# Create data
data = [["Only Long", round(L_Sharpe_Ratios[0],2), round(L_Sharpe_Ratios[1],2),
         round(L_Sharpe_Ratios[2],2), round(L_Sharpe_Ratios[3],2)],
        ["Long Short", round(LS_Sharpe_Ratios[0],2),␣
↪round(LS_Sharpe_Ratios[1],2),
         round(LS_Sharpe_Ratios[2],2), round(LS_Sharpe_Ratios[3],2)]]

# Define header names
col_names = ["K = 1", "K = 2", "K = 3", "K = 4"]
```

```
# Display table
print('k Nearest Neigbour')
print('Cross Validation Time Series Split Forward')
print('Sharpe Ratio')
print(tabulate(data, headers=col_names))
```

```
k Nearest Neigbour
Cross Validation Time Series Split Forward
Sharpe Ratio
              K = 1    K = 2    K = 3    K = 4
----------  -------  -------  -------  -------
Only Long      0.48     0.01      0.7      0.57
Long Short     0.25     0.79      0.62     0.6
```

[37]:
```
# Print all results
print('Mean Only Long Sharpe Ratio = ',round(np.mean(L_Sharpe_Ratios),2))
print('Mean Long Short Sharpe Ratio = ',round(np.mean(LS_Sharpe_Ratios),2))
```

```
Mean Only Long Sharpe Ratio =  0.44
Mean Long Short Sharpe Ratio =  0.56
```

18 K-Fold Cross Validation

[38]:
```
#-------------------------------------------------------------------------#
#                              Full Dataset                               |
#-------------------------------------------------------------------------#
# t_0 <-------------------------------------------------------------> t_n #

# K = 1
#-------------------------------------------------------------------------#
#     Test     |     Train     |     Train     |     Train     |     Train    |
#-------------------------------------------------------------------------#

# K = 2
#-------------------------------------------------------------------------#
#     Train    |     Test      |     Train     |     Train     |     Train    |
#-------------------------------------------------------------------------#

# K = 3
#-------------------------------------------------------------------------#
#     Train    |     Train     |     Test      |     Train     |     Train    |
#-------------------------------------------------------------------------#

# K = 4
#-------------------------------------------------------------------------#
#     Train    |     Train     |     Train     |     Test      |     Train    |
```

```
#-------------------------------------------------------------------------------#

# K = 5
#-------------------------------------------------------------------------------#
#     Train     |     Train     |     Train     |     Train     |     Test     |
#-------------------------------------------------------------------------------#


# Create three empty lists
L_Sharpe_Ratios = []
LS_Sharpe_Ratios = []
Acc_score = []

# For loop who create model, fit model, test model and compute performance and␣
 ↪Sharpe Ratio
for k in range (1,6):
    # Choose of the features we want in the Random Forest model

    L1 = int(round(len(features))*0.20*k)+1 # Geben
    L2 = int(round(len(features))*0.20*(k-1)) # Entfernen

    X_test = features[L2:L1]
    X_train = features.drop(index=X_test.index)

    y_test = labels[L2:L1]
    y_train = labels.drop(index=y_test.index)


    for i in range(1,2):
        knn_random.fit(X_train, y_train)
        param = knn_random.best_params_
        best_knn = KNeighborsClassifier(weights = param['weights'],
                                        p = param['p'],
                                        n_neighbors = param['n_neighbors'],
                                        leaf_size = param['leaf_size'],
                                        algorithm = param['algorithm'])

        # Fit the data to the model
        best_knn.fit(X_train, y_train)

        # Make predictions
        y_pred = best_knn.predict(X_test)
        Acc_score.append(accuracy_score(y_test, y_pred, normalize = True) * 100.
 ↪0)

        # Trading Performance
        X = bco['Log_Returns'].shift(-1)
```

```
        X = X[L2:L1]

        trading_days_Y = 5*52

        # Only Long
        perf = y_pred * X
        perf = perf.dropna()
        # Sharpe
        perf_sharpe = np.sqrt(trading_days_Y) * np.mean(perf) / (np.sqrt(np.
↪var(perf)))
        L_Sharpe_Ratios.append(perf_sharpe)

        # Long Short
        # Transform 0,1 Signal in a -1,1 Signal
        y_pred1 = []
        for i in range(0,len(y_pred)):
            if y_pred[i] == 0:
                y_pred1.append(-1)
            else:
                y_pred1.append(1)

        perf1 = y_pred1 * X
        perf1 = perf1.dropna()
        # Sharpe
        perf_sharpe1 = np.sqrt(trading_days_Y) * np.mean(perf1) / (np.sqrt(np.
↪var(perf1)))
        LS_Sharpe_Ratios.append(perf_sharpe1)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
Fitting 3 folds for each of 100 candidates, totalling 300 fits
Fitting 3 folds for each of 100 candidates, totalling 300 fits
Fitting 3 folds for each of 100 candidates, totalling 300 fits
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

[39]:
```
# Create data
data = [["Only Long", round(L_Sharpe_Ratios[0],2), round(L_Sharpe_Ratios[1],2),
         round(L_Sharpe_Ratios[2],2), round(L_Sharpe_Ratios[3],2),␣
↪round(L_Sharpe_Ratios[4],2)],
        ["Long Short", round(LS_Sharpe_Ratios[0],2),␣
↪round(LS_Sharpe_Ratios[1],2),
         round(LS_Sharpe_Ratios[2],2), round(LS_Sharpe_Ratios[3],2),␣
↪round(LS_Sharpe_Ratios[4],2)],
        ["Accuracy", round(Acc_score[0],2), round(Acc_score[1],2),
         round(Acc_score[2],2), round(Acc_score[3],2), round(Acc_score[4],2)]]

# Define header names
col_names = ["K = 1", "K = 2", "K = 3", "K = 4", "K = 5"]
```

```
# Display table
print('k Nearest Neighbour')
print('K-Fold Cross Validation')
print('Sharpe Ratio')
print(tabulate(data, headers=col_names))
```

```
k Nearest Neighbour
K-Fold Cross Validation
Sharpe Ratio
              K = 1    K = 2    K = 3    K = 4    K = 5
----------  -------  -------  -------  -------  -------
Only Long      1.59     0.65    -0.34     0.84     0.57
Long Short     2.01     0.51     0.33     0.99     0.6
Accuracy      53.36    52.54    50.75    51.99    52.95
```

[40]:
```
# Print all results
print('Mean Only Long Sharpe Ratio = ',round(np.mean(L_Sharpe_Ratios),2))
print('Mean Long Short Sharpe Ratio = ',round(np.mean(LS_Sharpe_Ratios),2))
print('Mean Accuracy Score = ',round(np.mean(Acc_score),2))
```

```
Mean Only Long Sharpe Ratio =  0.66
Mean Long Short Sharpe Ratio =  0.89
Mean Accuracy Score =  52.32
```

[ ]:
```

```

SVM_Regression

June 4, 2022

1 SVM model

1.0.1 Imports

[1]:
```
# Basic Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pprint import pprint
import seaborn as sns
from matplotlib import pyplot
import yfinance as yf

# Machine Learning Imports
import sklearn
from sklearn import svm
from sklearn.svm import SVR
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score, roc_auc_score
from sklearn.metrics import r2_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import RepeatedKFold
import itertools
from sklearn.metrics import roc_curve, auc
from sklearn.inspection import permutation_importance
from tabulate import tabulate
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
```

2 Load CSV File

[2]:
```
# From 16.08.2007 to 26.04.2022
bco = pd.read_csv('/Users/romaindeleze/Desktop/BSc/Romain_WI_TZ/8.Semester/BA/
↪Daten/Brent_Crude_Oil')
```

[3]:
```
bco.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3641 entries, 0 to 3640
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Date         3641 non-null   object
 1   Close        3641 non-null   float64
 2   Log_Returns  3641 non-null   float64
 3   Pred_Signal  3641 non-null   int64
 4   RSI          3641 non-null   float64
 5   K_percent    3641 non-null   float64
 6   MACD         3641 non-null   float64
 7   ROC          3641 non-null   float64
dtypes: float64(6), int64(1), object(1)
memory usage: 227.7+ KB
```

3 Building the Support Vector Machine Regession Model

We have split our data into a training set and testing set, so we need to identify our input columns which are the following:

RSI, Stochastic Oscillator, Price Rate of Change, MACD

Those columns will serve as our X , and our Y column will be the Close price column.

Once we've selected our columns, we need to split the data into a training and test set. Which will take our features and labels and split them based on the size we input. In our case, let's have the test_size be '20 % . After we've split the data, we can create our Support Vector Machine Regression (SVR) model. Once we've created it, we can fit the training data to the model using the fit method. Finally, with our "trained" model, we can make predictions. Take the X_test data set and use it to make predictions.

3.0.1 Split the dataset for SVM regression

[4]:
```
# Choose of the features we want in the SVM model
features = bco[['RSI','K_percent','ROC','MACD']]
labels = bco['Close'].shift(-1)

# Split the data in a 80%, 20% proportion
X_train, X_test, y_train, y_test = train_test_split(features,labels,␣
 ↪test_size=0.2, shuffle=False)

print('Features lenght = ',len(features))
print('Label lenght = ',len(labels))
print()
print('X_train lenght = ',len(X_train))
```

```
print('y_train lenght = ',len(y_train))
print('X_test lenght = ',len(X_test))
print('y_test lenght = ',len(y_test))

print('Backtesting: ',len(X_train)+len(X_test))
print('Backtesting: ',len(y_train)+len(y_test))
```

```
Features lenght =  3641
Label lenght =  3641

X_train lenght =  2912
y_train lenght =  2912
X_test lenght =  729
y_test lenght =  729
Backtesting:  3641
Backtesting:  3641
```

4 Hyperparameter Tuning

To do the hyperparameter tuning we need to optimize the tuning parameters.

The optimisation follows via the RandomizedSearchCV command and then we fit the optimisation model with the training data.

[5]:
```
model = SVR()

# Look at parameters used by our current SVR
print('Parameters currently in use:\n')
print(model.get_params())
```

```
Parameters currently in use:

{'C': 1.0, 'cache_size': 200, 'coef0': 0.0, 'degree': 3, 'epsilon': 0.1,
'gamma': 'scale', 'kernel': 'rbf', 'max_iter': -1, 'shrinking': True, 'tol':
0.001, 'verbose': False}
```

4.0.1 With the RandomizedSearchCV command we can achive the hyperparameter tuning

We have to choose which hyperparameter we want tune. Then we have to creat lists with the parameter values for which we are looking for. The RandomizedSearchCV command will compute different combinations and then we can access to the best parameter values.

[6]:
```
# Regularization parameter. The strength of the regularization is inversely␣
 ↪proportional to C
C = list(np.arange(1,101)) # 10000 Possibilities
```

```
# Epsilon in the epsilon-SVR model. It specifies the epsilon-tube within which␣
 ↪no penalty is
# associated in the training loss function with points predicted within a␣
 ↪distance epsilon
# from the actual value.
epsilon = list(np.arange(0.1,50.1,0.1))

# if gamma='scale' (default) is passed then it uses 1 / (n_features * X.var())␣
 ↪as value of gamma,
# if 'auto', uses 1 / n_features.
gamma = ['auto','scale']

shrinking =  [True,False]

cache_size = list(np.arange(1,51))

random_grid = {'C':C,
               'epsilon':epsilon,
               'gamma':gamma,
               'cache_size':cache_size,
               'shrinking':shrinking }

#pprint(random_grid)
```

[7]:
```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
svr = SVR()

cv = RepeatedKFold(n_splits=15, n_repeats=2, random_state=8)

# Random search of parameters, using 10 fold cross validation,
# search across 720 different combinations, and use all available cores
svr_random = RandomizedSearchCV(estimator = svr, param_distributions =␣
 ↪random_grid, n_iter = 24,
                                cv = cv, verbose = 1, random_state = 8, n_jobs␣
 ↪= 1,
                                scoring='r2')

# Fit the SVR search model
svr_random.fit(X_train, y_train) # The optimisation may take a few minutes
print('Best Score = ', svr_random.best_score_)
```

```
Fitting 30 folds for each of 24 candidates, totalling 720 fits
Best Score =  0.054789405629539836
```

[8]:
```
# This is the Output command to see the best Parameters
param = svr_random.best_params_
print(param)
```

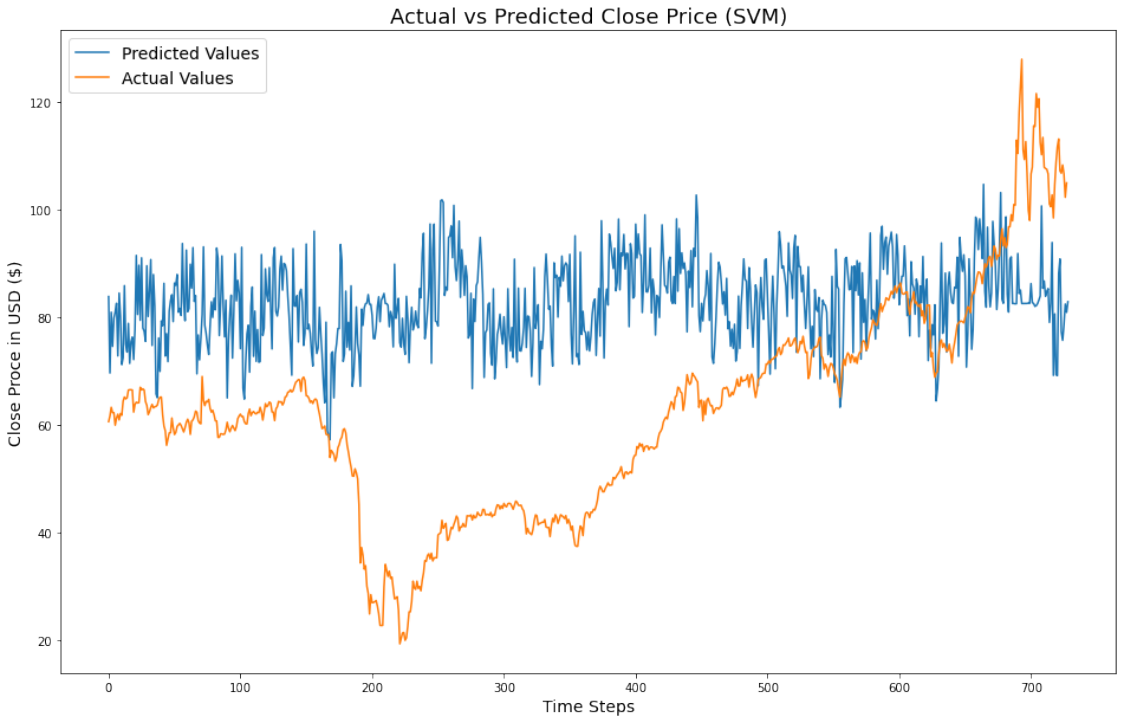
```
{'shrinking': True, 'gamma': 'auto', 'epsilon': 22.400000000000002,
'cache_size': 41, 'C': 19}
```

4.1 We create a SVR model with the parameter from the cross validation optimisation.

[9]:
```
# Create a Support Vector Machine Regression with the Hyperparameters from the␣
↪Tuning
svr = SVR(shrinking = param['shrinking'],
          gamma = param['gamma'],
          epsilon = param['epsilon'],
          cache_size = param['cache_size'],
          C = param['C'],
          kernel = 'rbf')

# Fit the data to the model
svr.fit(X_train, y_train)

# Make predictions
y_pred = svr.predict(X_test)
```

4.2 Plot the real and predicted close price

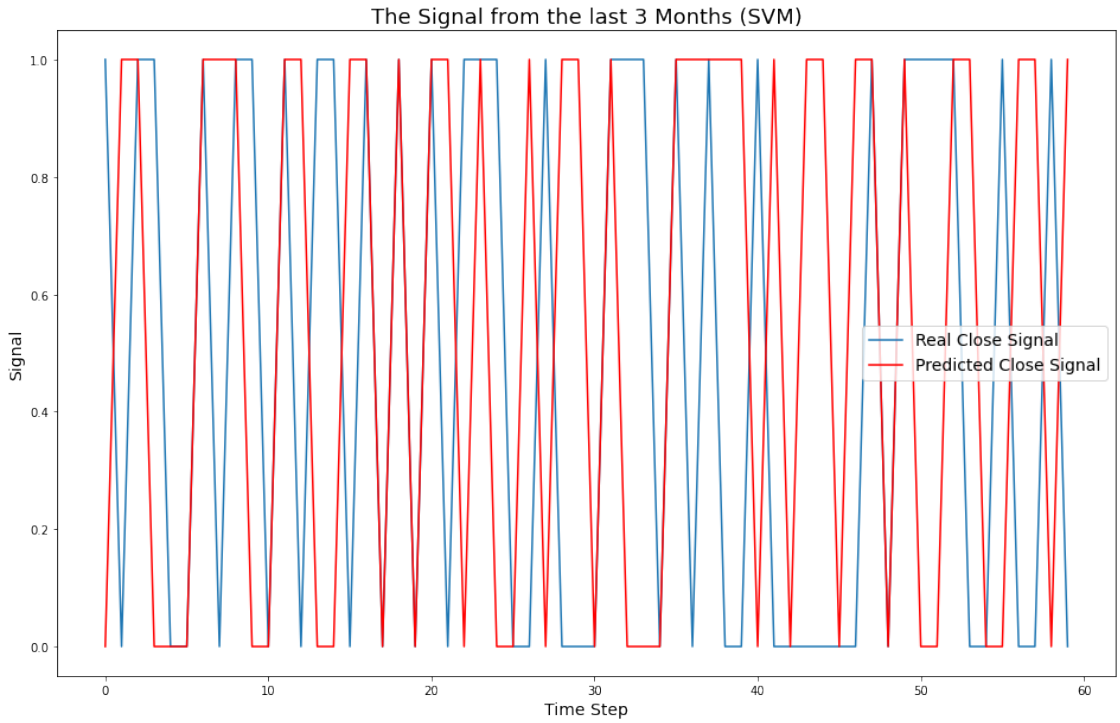
[10]:
```
# Plot the fit from the Regression
plt.figure(figsize=(16, 10))
plt.title('Actual vs Predicted Close Price (SVM)', fontsize = 18)
plt.plot(y_pred, label = 'Predicted Values')
plt.plot(y_test.values, label = 'Actual Values')
plt.ylabel('Close Proce in USD ($)', fontsize = 14)
plt.xlabel('Time Steps', fontsize = 14)
plt.legend(loc='best', fontsize = 14)
plt.show()
```



[11]:
```
# Transform real close and predicted close in a signal
X_returns = np.diff(np.log(y_test))
X = (X_returns > 0).astype(int)

y_returns = np.diff(np.log(y_pred))
y = (y_returns > 0).astype(int)
```

[12]:
```
# Lets see the actual and the predicted Signal from the last 3 months
m3 = 3*4*5
plt.figure(figsize=(16,10))
plt.title('The Signal from the last 3 Months (SVM)', fontsize = 18)
plt.plot(X[len(X)-m3:], label = 'Real Close Signal')
plt.plot(y[len(X)-m3:], color = 'red', label = 'Predicted Close Signal')
plt.ylabel('Signal', fontsize = 14)
plt.xlabel('Time Step', fontsize = 14)
plt.legend(loc='best', fontsize = 14)
```

[12]: <matplotlib.legend.Legend at 0x7fb471f29910>



5 Accuracy

We've built our model, so now we can see how accurate it is. SciKit learn, makes the process of evaluating our model very easy by providing a bunch of built-in metrics that we can call. One of those metrics is the accuracy_score. The accuracy_score function computes the accuracy, by calculating the sum of the correctly predicted signals and then dividing it by the total number of predictions. Imagine we had three TRUE values [1, 2, 3] , and our model predicted the following values [1, 2, 4] we would say the accuracy of our model is $2/3 = 66\ \%$.

6 AUC

AUC curve is a performance measurement for the classification problems at various threshold settings. AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1. By analogy, the Higher the AUC, the better the model is at distinguishing between patients with the disease and no disease.

7 RMSE

Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are;

RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit.

[13]:
```
# Print Accuracy
print('Accuracy in %: ', accuracy_score(X, y, normalize = True) * 100.0)
print()

# Checking performance our model with AUC Score.
print('Area under the curve: ',roc_auc_score(X, y))
print()

# Get the root mean squared error (RMSE)
print('RMSE Close Price = ' , np.sqrt(np.mean(((y_pred - y_test) ** 2))))
print('RMSE Signal = ' , np.sqrt(np.mean(((y - X) ** 2))))
```

```
Accuracy in %:  49.175824175824175

Area under the curve:  0.49181328025501997

RMSE Close Price =  28.23035378960343
RMSE Signal =  0.712910764571386
```

8 Classification Report

To get a more detailed overview of how the model performed, we can build a classification report that will compute the F1_Score , the Precision , the Recall , and the Support .

8.0.1 Precision

Precision measures the proportion of all correctly identified samples in a population of samples which are classified as positive labels and is defined as the following:

tp = True Positiv

fp = False Positiv

Precision = tp/(tp+fp)

The precision is intuitively the ability of the classifier not to label as positive a sample that is negative. The best value is 1, and the worst value is 0.

8.0.2 Recall

Recall (also known as sensitivity) measures the ability of a classifier to correctly identify positive labels and is defined as the following:

tp = True Positiv

fn = False Negativ

Recall = tp/(tp+fn)

The recall is intuitively the ability of the classifier to find all the positive samples. The best value is 1, and the worst value is 0.

8.0.3 Support

Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing. Support doesn't change between models but instead diagnoses the evaluation process.

8.0.4 F1 Score

In some cases, we will have models that may have low precision or high recall. It's difficult to compare two models with low precision and high recall or vice versa. To make results comparable, we use a metric called the F-Score. The F-score helps to measure Recall and Precision at the same time. It uses Harmonic Mean in place of Arithmetic Mean by punishing the extreme values more.

The traditional F-measure or balanced F-score (F1 score) is the harmonic mean of precision and recall:

tp = True Positiv

fp = False Positiv

fn = False Negativ

F1 Score = tp/(tp+0.5*(fp+fn))

[14]:
```
# Define the traget names
target_names = ['Down Day', 'Up Day']

# Build a classifcation report
report = classification_report(y_true = X, y_pred = y, target_names =␣
↪target_names, output_dict = True)

# Add it to a data frame, transpose it for readability.
report_df = pd.DataFrame(report).transpose()
report_df
```

[14]:

	precision	recall	f1-score	support
Down Day	0.441096	0.492355	0.465318	327.000000
Up Day	0.542700	0.491272	0.515707	401.000000

```
accuracy       0.491758  0.491758  0.491758    0.491758
macro avg      0.491898  0.491813  0.490512  728.000000
weighted avg   0.497062  0.491758  0.493073  728.000000
```

9 Confusion Matrix

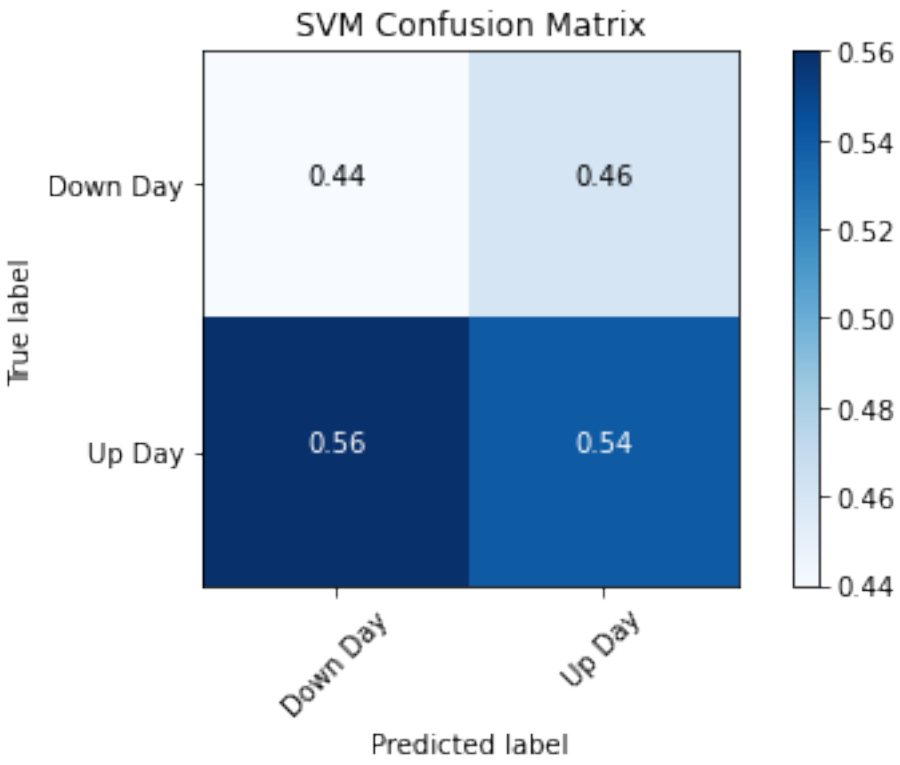
A confusion matrix is a technique for summarizing the performance of a classification algorithm. Classification accuracy alone can be misleading if you have an unequal number of observations in each class or if you have more than two classes in your dataset. Calculating a confusion matrix can give you a better idea of what your classification model is getting right and what types of errors it is making.

[15]:
```
cm = confusion_matrix(X, y, normalize='pred')
```

[16]:
```
# Function: Confusion Matrix Plot for Regression, Neural Network
def plot_confusion_matrix_regression(cm, classes,
                        normalize=False,
                        title='Confusion matrix',
                        cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 1.2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
            horizontalalignment="center",
            color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

[17]:
```
# Compute Confusion Matrix
svr_matrix = confusion_matrix(y, X)

# Save individual values
true_negatives = svr_matrix[0][0]
false_negatives = svr_matrix[1][0]
true_positives = svr_matrix[1][1]
false_positives = svr_matrix[0][1]

# Compute performance values
accuracy = (true_negatives + true_positives) / (true_negatives + true_positives␣
↪+ false_negatives + false_positives)
percision = true_positives / (true_positives + false_positives)
recall = true_positives / (true_positives + false_negatives)
specificity = true_negatives / (true_negatives + false_positives)

# Print values
print('Accuracy: {}'.format(round(float(accuracy),5)))
print('Percision: {}'.format(round(float(percision),5)) )
print('Recall: {}'.format(round(float(recall),5)) )
print('Specificity: {}'.format(round(float(specificity),5)))
print()

# Visualise the Confusion Matrix
cm_plot_labels = ['Down Day','Up Day']
disp = plot_confusion_matrix_regression(cm=np.round(cm,2),␣
↪classes=cm_plot_labels, title='SVM Confusion Matrix',
                                normalize=False)
```

```
Accuracy: 0.49176
Percision: 0.49127
Recall: 0.5427
Specificity: 0.4411

Confusion matrix, without normalization
[[0.44 0.46]
 [0.56 0.54]]
```



9.0.1 With the information from the confusion matrix we want know now how the proportion is.

How is the prediction distributed, and how ist the real values distributed

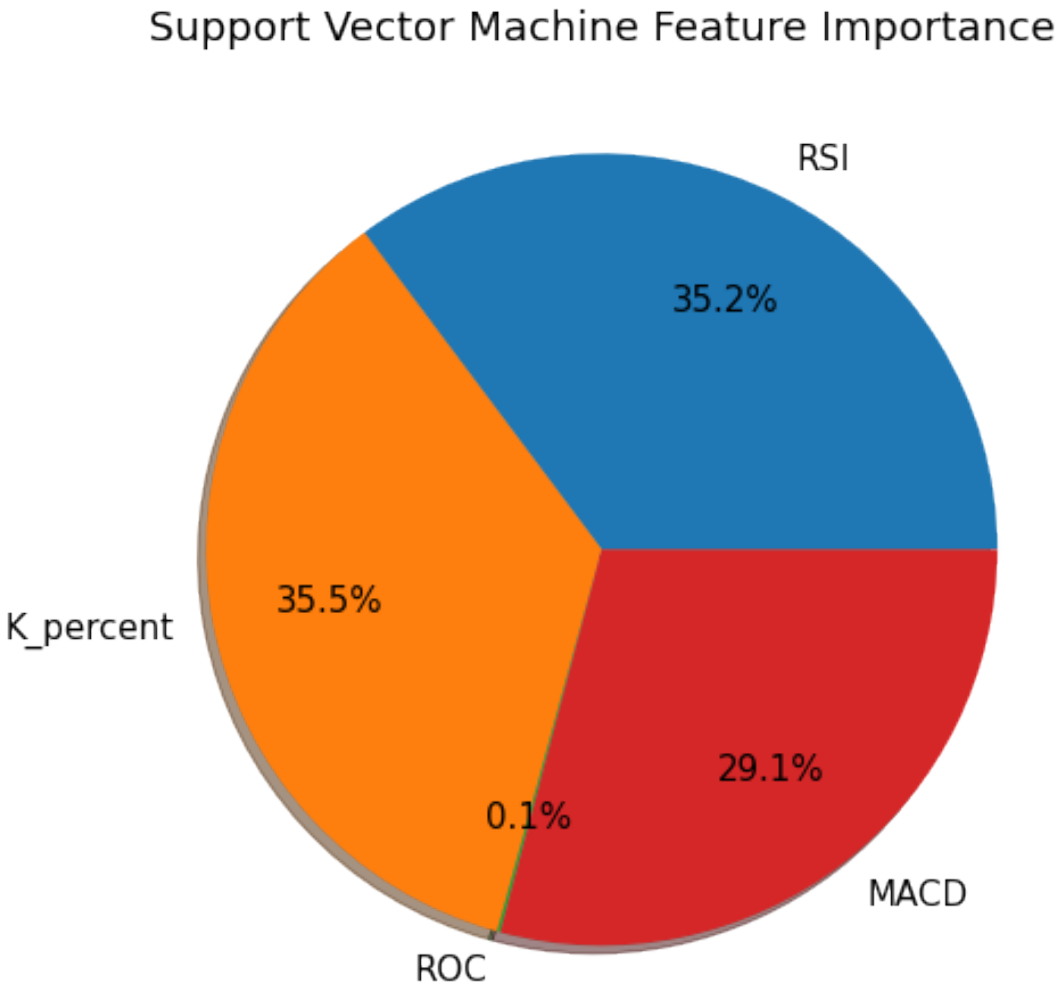
[18]:
```
# Propotion of the real values
unique, counts = np.unique(X, return_counts=True)
print('Real Proportion')
print(dict(zip(unique, counts/len(X))))
print()

# Propotion of the predicted values
unique, counts = np.unique(y, return_counts=True)
print('Predicted Proportion')
print(dict(zip(unique, counts/len(y))))
```

```
Real Proportion
{0: 0.4491758241758242, 1: 0.5508241758241759}

Predicted Proportion
{0: 0.5013736263736264, 1: 0.4986263736263736}
```

10 Feature Importance

[19]:
```
# Perform permutation importance
results = permutation_importance(svr, X_train, y_train)

# Get importance for r_squared
importance = results.importances_mean

# Create Pie Plot
imp_scale = sum(importance)
imp = importance/imp_scale
def func(pct, allvalues):
    return "{:.1f}%".format(pct)
plt.figure(figsize=(8,8))
plt.title('Support Vector Machine Feature Importance', fontsize = 18)
plt.pie(imp, labels=['RSI','K_percent','ROC','MACD'], autopct = lambda pct:␣
 ↪func(pct, imp),
        shadow=True, pctdistance=.7, textprops={'fontsize': 15})
```

[19]:
```
([<matplotlib.patches.Wedge at 0x7fb473d96340>,
  <matplotlib.patches.Wedge at 0x7fb473d96c70>,
  <matplotlib.patches.Wedge at 0x7fb473db1580>,
  <matplotlib.patches.Wedge at 0x7fb473db1e50>],
 [Text(0.49201956571092104, 0.9838276002215209, 'RSI'),
  Text(-1.080641559342738, -0.20546002098047175, 'K_percent'),
  Text(-0.28634235856097134, -1.0620772352770491, 'ROC'),
  Text(0.6708014985798142, -0.8717943275240301, 'MACD')],
 [Text(0.31310335999785877, 0.626072109231877, '35.2%'),
  Text(-0.687680992309015, -0.130747286078482, '35.5%'),
  Text(-0.18221786453879993, -0.6758673315399403, '0.1%'),
  Text(0.4268736809144272, -0.5547782084243826, '29.1%')])
```

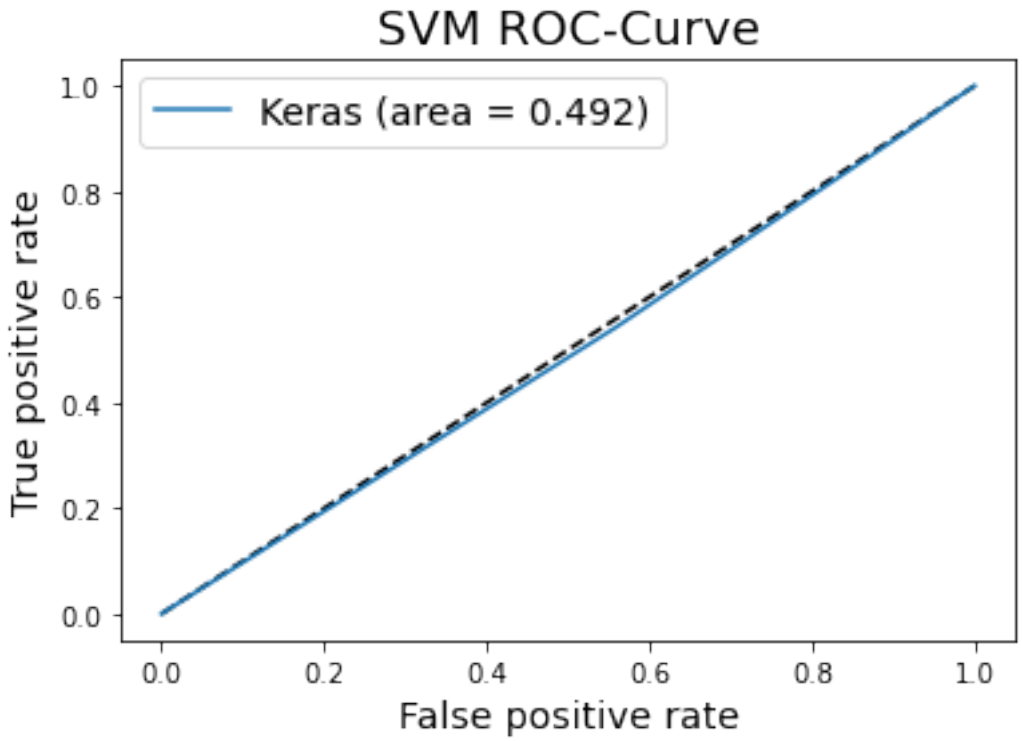


11 ROC Curve Plot

The Receiver Operating Characteristic is a graphical method to evaluate the performance of a binary classifier. A curve is drawn by plotting True Positive Rate (sensitivity) against False Positive Rate (1 - specificity) at various threshold values. ROC curve shows the trade-off between sensitivity and specificity. When the curve comes closer to the left-hand border and the top border of the ROC space, it indicates that the test is accurate. The closer the curve is to the top and left-hand border, the more accurate the test is. If the curve is close to the 45 degrees diagonal of the ROC space, it means that the test is not accurate. ROC curves can be used to select the optimal model and discard the suboptimal ones.

[20]:
```
y_pred_Signal = y
y_test_Signal = X
```

```
fpr_keras, tpr_keras, thresholds_keras = roc_curve(y_pred_Signal, y_test_Signal)

auc_keras = auc(fpr_keras, tpr_keras)

plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_keras, tpr_keras, label='Keras (area = {:.3f})'.format(auc_keras))
plt.xlabel('False positive rate', fontsize = 14)
plt.ylabel('True positive rate', fontsize = 14)
plt.title('SVM ROC-Curve', fontsize = 18)
plt.legend(loc='best', fontsize = 14)
plt.show()
```



12 Trading Performance

[21]:
```
# Creat Data Frame with Predicted Signal and Log Returns
n = len(y_pred)
X = bco['Log_Returns'].shift(-1)
X = X[len(X)-n+1:]
len(X)
```

[21]: 728

12.1 Long

[22]:
```
# Compute Only Long performance: Signal * Log Returns
perf = y * X

perf = perf.dropna()

trading_days_Y = 5*52

perf_sharpe = np.sqrt(trading_days_Y) * np.mean(perf) / (np.sqrt(np.var(perf)))
```

12.2 Long-Short

[23]:
```
# Transform 0,1 Signal in a -1,1 Signal
y_array = np.array(y)
y1 = []
for i in range(0,len(y)):
    if y_array[i] == 0:
        y1.append(-1)
    else:
        y1.append(1)
```

[24]:
```
# Compute Long Short performance: Signal * Log Returns
perf1 = y1 * X

perf1 = perf1.dropna()

trading_days_Y = 5*52

perf_sharpe1 = np.sqrt(trading_days_Y) * np.mean(perf1) / (np.sqrt(np.
 ↪var(perf1)))
```

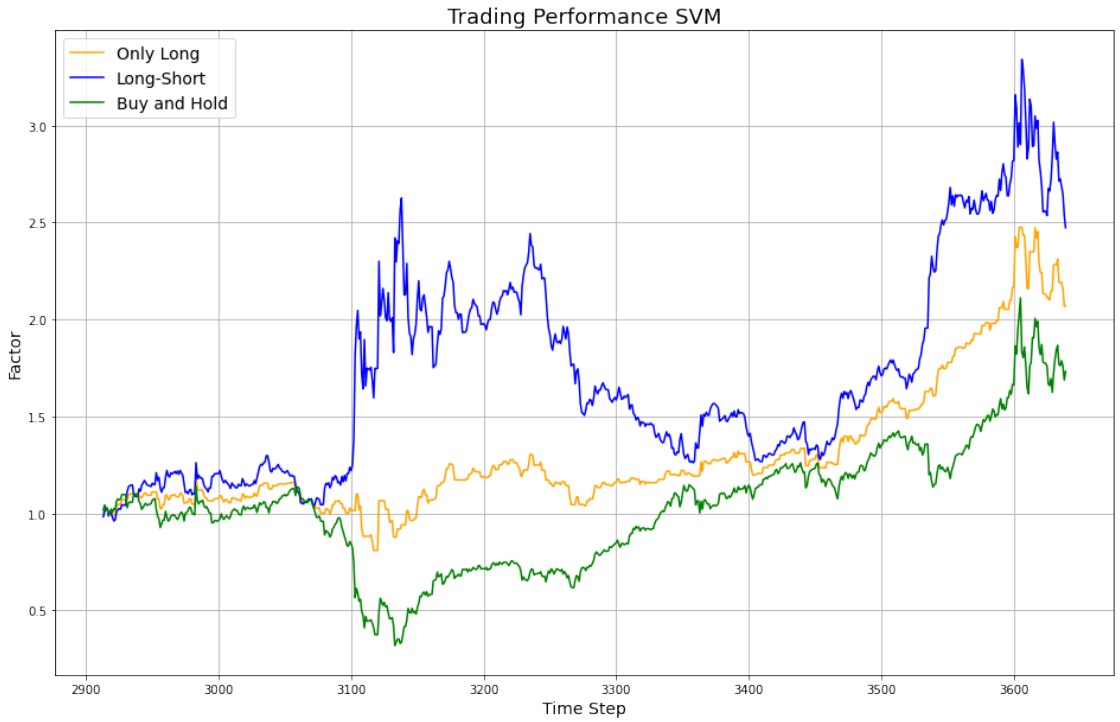
12.3 Buy and Hold

[25]:
```
perf2 = X
trading_days_Y = 5*52

perf_sharpe2 = np.sqrt(trading_days_Y) * np.mean(perf2) / (np.sqrt(np.
 ↪var(perf2)))
```

13 Comparison ML vs. Buy and Hold

[26]:
```
# Lets compare the performances
plt.figure(figsize=(16,10))
plt.title('Trading Performance SVM', fontsize = 18)
plt.ylabel('Factor', fontsize = 14)
plt.xlabel('Time Step', fontsize = 14)
plt.grid(True)
plt.plot(np.exp(perf).cumprod() , label='Only Long', color = 'orange')
plt.plot(np.exp(perf1).cumprod() , label='Long-Short', color = 'blue')
plt.plot(np.exp(perf2).cumprod() , label='Buy and Hold', color = 'green')
plt.legend(loc='best', fontsize = 14)
```

[26]:
```
<matplotlib.legend.Legend at 0x7fb4739b8f40>
```



14 Profit Factor

[27]:
```
# Profit Factor is the last value of the performance computation
L_profit = np.exp(perf).cumprod().iloc[-1]

LS_profit = np.exp(perf1).cumprod().iloc[-1]

BaH_profit = np.exp(perf2).cumprod().iloc[-2]
```

15 Performance Table

[28]:
```
# Create data
data = [["Buy and Hold", round(perf_sharpe2,2), round(BaH_profit,2)],
        ["Only Long", round(perf_sharpe,2), round(L_profit,2)],
        ["Long Short", round(perf_sharpe1,2), round(LS_profit,2)]]

# Define header names
col_names = ["Strategie", "Sharp Ratio", "Profit Factor"]

# Display table
print('SVM Regression')
print(tabulate(data, headers=col_names))
```

```
SVM Regression
Strategie       Sharp Ratio    Profit Factor
------------  -------------  ---------------
Buy and Hold           0.36             1.73
Only Long              0.76             2.07
Long Short             0.59             2.47
```

[29]:
```
# Performance Dataset
# Compute performance
SVM_Perf_OL = np.exp(perf).cumprod()
SVM_Perf_LS = np.exp(perf1).cumprod()

# Load in a Data Frame
SVM_Perf_data = pd.DataFrame()
SVM_Perf_data['SVM_Perf_OL'] = SVM_Perf_OL
SVM_Perf_data['SVM_Perf_LS'] = SVM_Perf_LS

# Create a csv File
SVM_Perf_data.to_csv('/Users/romaindeleze/Desktop/BSc/Romain_WI_TZ/8.Semester/
 ↪BA/Daten/SVM_Perf_data')
```

16 Extrem and Regular Value Accuracy

[30]:
```
# Creat Data Frame with Predicted Signal and Log Returns
n = len(y)
X = bco['Log_Returns'].shift(-1)
X = X[len(X)-n:]
y2 = np.squeeze(y) # Dimension Reduction
d = {'Predicted Signal':y2, 'Log Returns':X}
df = pd.DataFrame(data=d)
```

[31]:
```
# Creat the upperbound and the lowerbound
mu = np.mean(X)
sig = np.std(X)
lb = mu-2*sig
ub = mu+2*sig

# Creat Data Frame with only Values in the interval [lb,ub]
reg_val = df[(df['Log Returns'] >= lb) & (df['Log Returns'] <= ub)] # DataFrame␣
↪with Regular Values = [lb,ub]

# Creat Data Frame with only Values outside the interval [lb,ub]
ext_val = df.drop(index=reg_val.index)

# Add real Signal with the Log Returns for each Data Frame
reg_val['Real Signal'] = (reg_val['Log Returns'] > 0).astype(int)
ext_val['Real Signal'] = (ext_val['Log Returns'] > 0).astype(int)

# Compute accuracy of the Extrem Values and the Regular Values
acc_reg = accuracy_score(reg_val['Real Signal'], reg_val['Predicted Signal'],␣
↪normalize = True) * 100.0
acc_ext = accuracy_score(ext_val['Real Signal'], ext_val['Predicted Signal'],␣
↪normalize = True) * 100.0
print('Accuracy Regular Values = ', round(acc_reg,2))
print('Accuracy Exrtem Values (95%) = ', round(acc_ext,2))
```

```
Accuracy Regular Values =  48.99
Accuracy Exrtem Values (95%) =  52.78
```

```
<ipython-input-31-f97868f112e9>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  reg_val['Real Signal'] = (reg_val['Log Returns'] > 0).astype(int)
```

[32]:
```
# Creat the upperbound and the lowerbound
mu = np.mean(X)
sig = np.std(X)
lb = mu-3*sig
ub = mu+3*sig

# Creat Data Frame with only Values in the interval [lb,ub]
reg_val = df[(df['Log Returns'] >= lb) & (df['Log Returns'] <= ub)] # DataFrame␣
↪with Regular Values = [lb,ub]

# Creat Data Frame with only Values outside the interval [lb,ub]
```

```
ext_val = df.drop(index=reg_val.index)

# Add real Signal with the Log Returns for each Data Frame
reg_val['Real Signal'] = (reg_val['Log Returns'] > 0).astype(int)
ext_val['Real Signal'] = (ext_val['Log Returns'] > 0).astype(int)

# Compute accuracy of the Extrem Values and the Regular Values
acc_reg = accuracy_score(reg_val['Real Signal'], reg_val['Predicted Signal'],␣
 ↪normalize = True) * 100.0
acc_ext = accuracy_score(ext_val['Real Signal'], ext_val['Predicted Signal'],␣
 ↪normalize = True) * 100.0
print('Accuracy Regular Values = ', round(acc_reg,2))
print('Accuracy Exrtem Values (99%) = ', round(acc_ext,2))
```

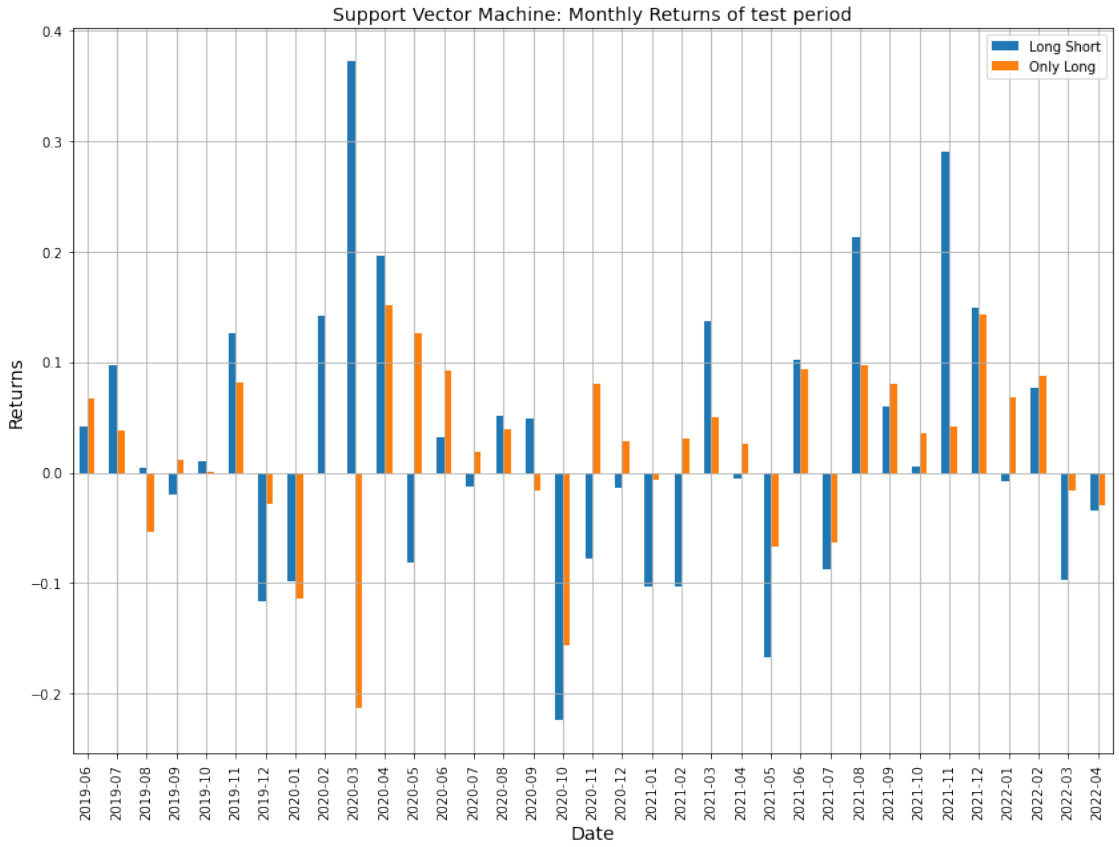
```
Accuracy Regular Values =  49.09
Accuracy Exrtem Values (99%) =  53.33
```

```
<ipython-input-32-a883c4847048>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  reg_val['Real Signal'] = (reg_val['Log Returns'] > 0).astype(int)
```

[33]:
```
# Indexing
dat = yf.download('BZ=F', start='2007-08-16',end = "2022-04-26",period='1d')
end_len = len(dat["Close"])
long_short = np.exp(perf1).cumprod()
start_len = end_len - len(long_short)
index = dat["Close"][start_len:end_len].index
perf1.index = index
perf.index = index

# Create Monthly Returns Long Short
da = perf1.groupby([(perf1.index.year),(perf1.index.month)]).sum()
monthly_ret_LS = da.values
n = len(monthly_ret_LS)
monthly_ret_LS = pd.DataFrame(monthly_ret_LS,index = pd.date_range(start='6/1/
 ↪2019', freq='M', periods=n))

# Create Monthly Returns Long
da1 = perf.groupby([(perf.index.year),(perf.index.month)]).sum()
monthly_ret_OL = da1.values
monthly_ret_OL = pd.DataFrame(monthly_ret_OL,index = pd.date_range(start='6/1/
 ↪2019', freq='M', periods=n))
```

```
# Connect DF
mon_ret = pd.concat([monthly_ret_LS, monthly_ret_OL], axis=1)
mon_ret.columns = ["Long Short","Only Long"]

# x Axis
x_ax = pd.period_range('6/1/2019', '2022-04-30', freq='M').strftime('%Y-%m')
mon_ret.index = x_ax

#Plot the barplot
ax = mon_ret.plot.bar(figsize=(14,10), rot = 0)
plt.title("Support Vector Machine: Monthly Returns of test period",fontsize=14)
plt.xlabel('Date', fontsize=14)
plt.xticks(rotation=90)
plt.ylabel('Returns', fontsize=14)
plt.legend(loc='best')
plt.grid()
```

```
[**********100%**********]  1 of 1 completed
```



[34]:
```
# MDD and Maximum Profit
max_profit_LS = np.max(da)
max_prift_OL = np.max(da1)
MDD_LS = np.min(da)
MDD_OL = np.min(da1)

# Create data
data = [["Only Long", round(MDD_OL,2), round(max_prift_OL,2)],
["Long Short", round(MDD_LS,2), round(max_profit_LS,2)]]

# Define header names
col_names = ["Strategie", "MDD", "Max. Profit"]

# Display table
print('MDD and MAX.Profti of SVM')
print(tabulate(data, headers=col_names))
```

```
MDD and MAX.Profti of SVM
Strategie      MDD    Max. Profit
-----------  -----  -------------
Only Long    -0.21           0.15
Long Short   -0.22           0.37
```

17 Time Series Cross Validation Backwards

[35]:
```
#-------------------------------------------------------------------------#
#                               Full Dataset                              |
#-------------------------------------------------------------------------#
# t_0 <-------------------------------------------------------------> t_n #

# K = 1
#-------------------------------------------------------------------------#
#              |               |              |    Train      |     Test     |
#-------------------------------------------------------------------------#

# K = 2
#-------------------------------------------------------------------------#
#              |               |     Train    |    Train      |     Test     |
#-------------------------------------------------------------------------#

# K = 3
#-------------------------------------------------------------------------#
#              |     Train     |     Train    |    Train      |     Test     |
#-------------------------------------------------------------------------#

# K = 4
#-------------------------------------------------------------------------#
```

```
#     Train     |     Train     |     Train     |     Train       |     Test     |
#-------------------------------------------------------------------------------#

# Create two empty lists
L_Sharpe_Ratios = []
LS_Sharpe_Ratios = []

# For loop who create model, fit model, test model and compute performance and␣
 ↪Sharpe Ratio
for k in range (1,5):
    # Choose of the features we want in the Random Forest model
    features = bco[['RSI','K_percent','ROC','MACD']]
    labels = bco['Close'].shift(-1)

    X_train, X_test, y_train, y_test = train_test_split(features,labels,␣
 ↪test_size=0.2, shuffle=False)

    for i in range(1,2):
        feature_len = int(round(len(X_train))*0.25*k)
        X_train = X_train[-feature_len:]
        label_len = int(round(len(y_train))*0.25*k)
        y_train = y_train[-label_len:]

        svr_random.fit(X_train, y_train)
        param = svr_random.best_params_
        svr = SVR(shrinking = param['shrinking'],
                  gamma = param['gamma'],
                  epsilon = param['epsilon'],
                  cache_size = param['cache_size'],
                  C = param['C'],
                  kernel = 'rbf')

        # Fit the data to the model
        svr.fit(X_train, y_train)

        # Make predictions
        y_pred = svr.predict(X_test)

        # Transform prediction in a signal
        y_returns = np.diff(np.log(y_pred))
        y = (y_returns > 0).astype(int)

        # Trading Performance
        n = len(y)
        X = bco['Log_Returns'].shift(-1)
        X = X[len(X)-n:]
```

```
        trading_days_Y = 5*52

        # Only Long
        perf = y * X
        perf = perf.dropna()
        # Sharpe
        perf_sharpe = np.sqrt(trading_days_Y) * np.mean(perf) / (np.sqrt(np.
↪var(perf)))
        L_Sharpe_Ratios.append(perf_sharpe)

        # Long Short
        # Transform 0,1 Signal in a -1,1 Signal
        y_pred1 = []
        for i in range(0,len(y)):
            if y[i] == 0:
                y_pred1.append(-1)
            else:
                y_pred1.append(1)

        perf1 = y_pred1 * X
        perf1 = perf1.dropna()
        # Sharpe
        perf_sharpe1 = np.sqrt(trading_days_Y) * np.mean(perf1) / (np.sqrt(np.
↪var(perf1)))
        LS_Sharpe_Ratios.append(perf_sharpe1)
```

```
Fitting 30 folds for each of 24 candidates, totalling 720 fits
Fitting 30 folds for each of 24 candidates, totalling 720 fits
Fitting 30 folds for each of 24 candidates, totalling 720 fits
Fitting 30 folds for each of 24 candidates, totalling 720 fits
```

[36]:
```
# Create data
data = [["Only Long", round(L_Sharpe_Ratios[0],2), round(L_Sharpe_Ratios[1],2),
         round(L_Sharpe_Ratios[2],2), round(L_Sharpe_Ratios[3],2)],
        ["Long Short", round(LS_Sharpe_Ratios[0],2),␣
↪round(LS_Sharpe_Ratios[1],2),
         round(LS_Sharpe_Ratios[2],2), round(LS_Sharpe_Ratios[3],2)]]

# Define header names
col_names = ["K = 1", "K = 2", "K = 3", "K = 4"]

# Display table
print('Support Vector Machine')
print('Cross Validation Time Series Split Backward')
print('Sharpe Ratio')
print(tabulate(data, headers=col_names))
```

```
Support Vector Machine
Cross Validation Time Series Split Backward
Sharpe Ratio
              K = 1    K = 2    K = 3    K = 4
----------  -------  -------  -------  -------
Only Long      0.38     0.44     0.68     0.76
Long Short     0.2      0.26     0.52     0.59
```

[37]:
```
# Print all results
print('Mean Only Long Sharpe Ratio = ',round(np.mean(L_Sharpe_Ratios),2))
print('Mean Long Short Sharpe Ratio = ',round(np.mean(LS_Sharpe_Ratios),2))
```

```
Mean Only Long Sharpe Ratio =  0.56
Mean Long Short Sharpe Ratio =  0.39
```

18 Time Series Cross Validation Forwards

[38]:
```
#-------------------------------------------------------------------------------#
#                                 Full Dataset                                  |
#-------------------------------------------------------------------------------#
# t_0 <-------------------------------------------------------------> t_n #

# K = 1
#-------------------------------------------------------------------------------#
#     Train     |     Test      |               |               |               |
#-------------------------------------------------------------------------------#

# K = 2
#-------------------------------------------------------------------------------#
#     Train     |     Train     |     Test      |               |               |
#-------------------------------------------------------------------------------#

# K = 3
#-------------------------------------------------------------------------------#
#     Train     |     Train     |     Train     |    Test       |               |
#-------------------------------------------------------------------------------#

# K = 4
#-------------------------------------------------------------------------------#
#     Train     |     Train     |     Train     |    Train      |     Test      |
#-------------------------------------------------------------------------------#

# Create two epmty lists
L_Sharpe_Ratios = []
LS_Sharpe_Ratios = []
```

```
# For loop who create model, fit model, test model and compute performance and␣
 ↪Sharpe Ratio
for k in range (2,6):
    # Choose of the features we want in the Random Forest model
    features = bco[['RSI','K_percent','ROC','MACD']]
    labels = bco['Close'].shift(-1)

    feature_len = int(round(len(features))*0.2*k)
    label_len = int(round(len(labels))*0.2*k)

    features = features[:feature_len]
    labels = labels[:label_len]

    X_train = features[:-729]
    X_test = features[-729:]
    y_train = labels[:-729]
    y_test = labels[-729:]

    for i in range(1,2):
        svr_random.fit(X_train, y_train)
        param = svr_random.best_params_
        svr = SVR(shrinking = param['shrinking'],
                  gamma = param['gamma'],
                  epsilon = param['epsilon'],
                  cache_size = param['cache_size'],
                  C = param['C'],
                  kernel = 'rbf')

        # Fit the data to the model
        svr.fit(X_train, y_train)

        # Make predictions
        y_pred = svr.predict(X_test)

        # Transform prediction in a signal
        y_returns = np.diff(np.log(y_pred))
        y = (y_returns > 0).astype(int)

        # Trading Performance
        X = bco['Log_Returns'].shift(-1)
        X = X[:label_len]
        X = X[-729+1:]

        trading_days_Y = 5*52

        # Only Long
        perf = y * X
```

```
        perf = perf.dropna()
        # Sharpe
        perf_sharpe = np.sqrt(trading_days_Y) * np.mean(perf) / (np.sqrt(np.
↪var(perf)))
        L_Sharpe_Ratios.append(perf_sharpe)

        # Long Short
        # Transform 0,1 Signal in a -1,1 Signal
        y_pred1 = []
        for i in range(0,len(y)):
            if y[i] == 0:
                y_pred1.append(-1)
            else:
                y_pred1.append(1)

        perf1 = y_pred1 * X
        perf1 = perf1.dropna()
        # Sharpe
        perf_sharpe1 = np.sqrt(trading_days_Y) * np.mean(perf1) / (np.sqrt(np.
↪var(perf1)))
        LS_Sharpe_Ratios.append(perf_sharpe1)
```

```
Fitting 30 folds for each of 24 candidates, totalling 720 fits
Fitting 30 folds for each of 24 candidates, totalling 720 fits
Fitting 30 folds for each of 24 candidates, totalling 720 fits
Fitting 30 folds for each of 24 candidates, totalling 720 fits
```

[39]:
```
# Create data
data = [["Only Long", round(L_Sharpe_Ratios[0],2), round(L_Sharpe_Ratios[1],2),
         round(L_Sharpe_Ratios[2],2), round(L_Sharpe_Ratios[3],2)],
        ["Long Short", round(LS_Sharpe_Ratios[0],2),␣
↪round(LS_Sharpe_Ratios[1],2),
         round(LS_Sharpe_Ratios[2],2), round(LS_Sharpe_Ratios[3],2)]]

# Define header names
col_names = ["K = 1", "K = 2", "K = 3", "K = 4"]

# Display table
print('Support Vector Machine')
print('Cross Validation Time Series Split Forward')
print('Sharpe Ratio')
print(tabulate(data, headers=col_names))
```

```
Support Vector Machine
Cross Validation Time Series Split Forward
Sharpe Ratio
              K = 1    K = 2    K = 3    K = 4
```

```
----------  -------  -------  -------  -------
Only Long      0.19    -0.93    -0.25     0.76
Long Short    -0.16    -0.49    -0.66     0.59
```

[40]:
```
# Print all results
print('Mean Only Long Sharpe Ratio = ',round(np.mean(L_Sharpe_Ratios),2))
print('Mean Long Short Sharpe Ratio = ',round(np.mean(LS_Sharpe_Ratios),2))
```

```
Mean Only Long Sharpe Ratio =  -0.06
Mean Long Short Sharpe Ratio =  -0.18
```

19 K-Fold Cross Validation

[41]:
```
#-------------------------------------------------------------------------#
#                              Full Dataset                               |
#-------------------------------------------------------------------------#
# t_0 <-------------------------------------------------------------> t_n #

# K = 1
#-------------------------------------------------------------------------#
#     Test     |     Train     |     Train     |     Train     |     Train     |
#-------------------------------------------------------------------------#

# K = 2
#-------------------------------------------------------------------------#
#     Train    |     Test      |     Train     |     Train     |     Train     |
#-------------------------------------------------------------------------#

# K = 3
#-------------------------------------------------------------------------#
#     Train    |     Train     |     Test      |     Train     |     Train     |
#-------------------------------------------------------------------------#

# K = 4
#-------------------------------------------------------------------------#
#     Train    |     Train     |     Train     |     Test      |     Train     |
#-------------------------------------------------------------------------#

# K = 5
#-------------------------------------------------------------------------#
#     Train    |     Train     |     Train     |     Train     |     Test      |
#-------------------------------------------------------------------------#

# Create three empty lists
L_Sharpe_Ratios = []
LS_Sharpe_Ratios = []
Acc_score = []
```

```
# For loop who create model, fit model, test model and compute performance and␣
 ↪Sharpe Ratio
for k in range (1,6):
    # Choose of the features we want in the Random Forest model
    features = bco[['RSI','K_percent','ROC','MACD']]
    labels = bco['Close'].shift(-1)
    features = features[:-1]
    labels = labels[:-1]

    L1 = int(round(len(features))*0.20*k)
    L2 = int(round(len(features))*0.20*(k-1))

    X_test = features[L2:L1]
    X_train = features.drop(index=X_test.index)

    y_test = labels[L2:L1]
    y_train = labels.drop(index=y_test.index)

    for i in range(1,2):
        svr_random.fit(X_train, y_train)
        param = svr_random.best_params_
        svr = SVR(shrinking = param['shrinking'],
                  gamma = param['gamma'],
                  epsilon = param['epsilon'],
                  cache_size = param['cache_size'],
                  C = param['C'],
                  kernel = 'rbf')

        # Fit the data to the model
        svr.fit(X_train, y_train)

        # Make predictions
        y_pred = svr.predict(X_test)

        # Transform prediction in a signal
        y_returns = np.diff(np.log(y_pred))
        y = (y_returns > 0).astype(int)

        # Trading Performance
        n = len(y)
        X = bco['Log_Returns'].shift(-1)
        X = X[L2+1:L1]

        X1 = (X > 0).astype(int)

        Acc_score.append(accuracy_score(X1, y, normalize = True) * 100.0)
```

```
        trading_days_Y = 5*52

        # Only Long
        perf = y * X
        perf = perf.dropna()
        # Sharpe
        perf_sharpe = np.sqrt(trading_days_Y) * np.mean(perf) / (np.sqrt(np.
↪var(perf)))
        L_Sharpe_Ratios.append(perf_sharpe)

        # Long Short
        # Transform 0,1 Signal in a -1,1 Signal
        y_pred1 = []
        for i in range(0,len(y)):
            if y[i] == 0:
                y_pred1.append(-1)
            else:
                y_pred1.append(1)

        perf1 = y_pred1 * X
        perf1 = perf1.dropna()
        # Sharpe
        perf_sharpe1 = np.sqrt(trading_days_Y) * np.mean(perf1) / (np.sqrt(np.
↪var(perf1)))
        LS_Sharpe_Ratios.append(perf_sharpe1)
```

```
Fitting 30 folds for each of 24 candidates, totalling 720 fits
Fitting 30 folds for each of 24 candidates, totalling 720 fits
Fitting 30 folds for each of 24 candidates, totalling 720 fits
Fitting 30 folds for each of 24 candidates, totalling 720 fits
Fitting 30 folds for each of 24 candidates, totalling 720 fits
```

[42]:
```
# Create data
data = [["Only Long", round(L_Sharpe_Ratios[0],2), round(L_Sharpe_Ratios[1],2),
         round(L_Sharpe_Ratios[2],2), round(L_Sharpe_Ratios[3],2),␣
↪round(L_Sharpe_Ratios[4],2)],
        ["Long Short", round(LS_Sharpe_Ratios[0],2),␣
↪round(LS_Sharpe_Ratios[1],2),
         round(LS_Sharpe_Ratios[2],2), round(LS_Sharpe_Ratios[3],2),␣
↪round(LS_Sharpe_Ratios[4],2)],
        ["Accuracy", round(Acc_score[0],2), round(Acc_score[1],2),
         round(Acc_score[2],2), round(Acc_score[3],2), round(Acc_score[4],2)]]

# Define header names
col_names = ["K = 1", "K = 2", "K = 3", "K = 4", "K = 5"]
```

```
# Display table
print('Support Vector Machine')
print('K-Fold Cross Validation')
print('Sharpe Ratio')
print(tabulate(data, headers=col_names))
```

```
Support Vector Machine
K-Fold Cross Validation
Sharpe Ratio
              K = 1    K = 2    K = 3    K = 4    K = 5
----------  -------  -------  -------  -------  -------
Only Long      0.06     1.02    -0.58     0.4      0.76
Long Short    -0.02     0.87    -0.07     0.21     0.59
Accuracy      51.17    51.31    52.96    50.89    49.24
```

[43]:
```
# Print all resulta
print('Mean Only Long Sharpe Ratio = ',round(np.mean(L_Sharpe_Ratios),2))
print('Mean Long Short Sharpe Ratio = ',round(np.mean(LS_Sharpe_Ratios),2))
print('Mean Accuracy Score = ',round(np.mean(Acc_score),2))
```

```
Mean Only Long Sharpe Ratio =  0.33
Mean Long Short Sharpe Ratio =  0.32
Mean Accuracy Score =  51.11
```

[ ]:
```

```

References

[1] L. Vaughan, *Flash crash: A trading savant, a global manhunt, and the most mysterious market crash in history*. Doubleday, 2020.

[2] O. Baron, "Diese aktien hat der supercomputer," url: https://www.godmode-trader.de/artikel/diese-aktien-hat-der-supercomputer-2,5824912, August 2021.

[3] S. Grimsley, "What is financial data? - definition & concept," url: https://study.com/academy/lesson/what-is-financial-data-definition-lesson-quiz.html, September 2021.

[4] C. Francq and J.-M. Zakoian, *GARCH models: Structure, statistical inference and Financial Applications*. Wiley, 2019.

[5] C. Y. Wijaya, "Machine learning explainability introduction via eli5," url: https://medium.com/towards-data-science/machine-learning-explainability-introduction-via-eli5-99c767f017e2, October 2021.

[6] E. F. Fama, "Efficient capital markets: A review of theory and empirical work," *The Journal of Finance*, vol. 25, no. 2, pp. 383—-417, December 1970.

[7] R. J. Shiller, "From efficient markets theory to behavioral finance," *Journal of Economic Perspectives*, vol. 17, no. 1, p. 83–104, 2003.

[8] E. Sinclair, *The EfficientMarket Hypothesisand Its Limitations*, 1st ed. Wiley, 2008, p. 11–27.

[9] CFI, "Machine learning (in finance)," url: https://corporatefinanceinstitute.com/resources/knowledge/other/machine-learning-in-finance/, Jan 2022.

[10] C. Francq and J.-M. Zakoian, *GARCH models: Structure, statistical inference and Financial Applications*. Wiley, 2019.

[11] A. Liaw, M. Wiener *et al.*, "Classification and regression by random forest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[12] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, no. 3, p. 199–222, 2004.

[13] A. Mucherino, P. J. Papajorgji, and P. M. Pardalos, "K-nearest neighbor classification," in *Data mining in agriculture*. Springer, 2009, pp. 83–106.

[14] A. Navlani, "Knn classification tutorial using sklearn python," url: https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn, August 2018.

[15] J. Patterson and A. Gibson, *Deep learning - A practitioner's approach*. O'Reilly, 2017.

[16] J. Cao, Z. Li, and J. Li, "Financial time series forecasting model based on ceemdan and lstm," *Physica A: Statistical Mechanics and its Applications*, vol. 519, p. 127–139, 2019.

[17] S. Su, "North sea forties weakens after loadings increase for february," url: http://www.bloomberg.com/news/2011-01-10/north-sea-forties-weakens-after-loadings-increase-for-february.html, January 2011. [Online]. Available: http://www.bloomberg.com/news/2011-01-10/north-sea-forties-weakens-after-loadings-increase-for-february.html

[18] Erdöl-Vereinigung, 2005.

[19] S. Luber, "Was ist ein kpi (key performance indicator)?" url: https://www.bigdata-insider.de/was-ist-ein-kpi-key-performance-indicator-a-887319/, November 2019.

[20] R. Gehrt, "Technische analyse – markttechnische indikatoren," url: https://www.lynxbroker.ch/boerse/trading/technische-analyse/technische-indikatoren/technische-analyse-markttechnische-indikatoren/, October 2020.

[21] R. Best, "Rate of return: Formula, calculation & amp; examples," url: https://seekingalpha.com/article/4467051-rate-of-return, May 2022.

[22] "Why log returns," url: https://quantivity.wordpress.com/2011/02/21/why-log-returns/, February 2011.

[23] Admirals, "Überkauft oder überverkauft? der rsi indikator schafft klarheit," url: https://admiralmarkets.com/de/wissen/articles/forex-indicators/der-relative-strength-index-trading-mit-dem-rsi-indikator, February 2022.

[24] A. Hayes, "Stochastic oscillator," url: https://www.investopedia.com/terms/s/stochasticoscillator.asp, December 2021.

[25] W. Gürtel, "Cmc nachgefragt: Der macd-indikator erklärt," url: https://www.cmcmarkets.com/de-at/hilfe/glossar/m/macd-indikator, 2022.

[26] Elearnmarkets, "Roc indicator - how to trade with rate of change indicator ?" url: https://www.elearnmarkets.com/blog/rate-of-change/, February 2022.

[27] D. Lv, S. Yuan, M. Li, and Y. Xiang, "An empirical study of machine learning algorithms for stock daily trading strategy," *Mathematical Problems in Engineering*, vol. 2019, p. 1–30, 2019.

[28] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[30] S. Luber and N. Litzel, "Was ist random forest?" url:https://www.bigdata-insider.de/was-ist-random-forest-a-913937/, March 2020.

[31] Z. Stern, "K-fold cross validation explained:," url: https://medium.com/@zstern/k-fold-cross-validation-explained-5aeba90ebb, December 2018.

[32] R. Patro, "Cross validation - k fold vs. monte carlo," url:https://towardsdatascience.com/cross-validation-k-fold-vs-monte-carlo-e54df2fc179b, February 2021.