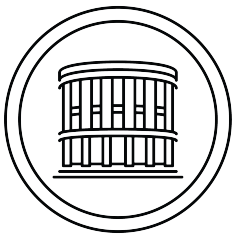
COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS



MACHINE LEARNING FOR PREDICTIVE MODELING IN TIME SERIES ANALYSIS

MASTER'S THESIS

2025 HANNA MUCHOVÁ

COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

MACHINE LEARNING FOR PREDICTIVE MODELING IN TIME SERIES ANALYSIS

MASTER'S THESIS

Study programme:	Mathematics of Economy, Finance and Modeling
Field of study:	1113 Matematics
Department:	Department of Applied Mathematics and Statistics
Supervisor:	CYRIL IZUCHUKWU UDEANI, PhD.

Bratislava 2023 **HANNA MUCHOVÁ**





Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Hanna Muchová
Študijný program: ekonomicko-finančná matematika a modelovanie (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: matematika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Machine learning for predictive modeling in time series analysis
Machine learning pre prediktívne modelovanie v analýze časových radov

Anotácia: Táto výskumná práca sa zameriava na vývoj prediktívnych modelov pomocou strojového učenia s cieľom predvídať cenu finančných nástrojov v čase. Analyzovaním historických údajov je cieľom vytvoriť presné a spoľahlivé modely, ktoré dokážu predpovedať budúcu cenu finančných nástrojov.

Vedúci: Cyril Izuchukwu Udeani, PhD.
Katedra: FMFI.KAMŠ - Katedra aplikovanej matematiky a štatistiky
Vedúci katedry: doc. Mgr. Igor Melicherčík, PhD.

Dátum zadania: 12.01.2024

Dátum schválenia: 25.01.2024 prof. RNDr. Daniel Ševčovič, DrSc.
garant študijného programu

.....
študent

.....
vedúci práce





Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Hanna Muchová
Study programme:	Mathematical Economics, Finance and Modelling (Single degree study, master II. deg., full time form)
Field of Study:	Mathematics
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Machine learning for predictive modeling in time series analysis

Annotation: This research focuses on developing predictive models using machine learning to anticipate the price of financial instruments over time. By analyzing historical data, the goal is to create accurate and reliable models that can forecast the future price of financial instruments.

Supervisor:	Cyril Izuchukwu Udeani, PhD.
Department:	FMFI.KAMŠ - Department of Applied Mathematics and Statistics
Head of department:	doc. Mgr. Igor Melicherčík, PhD.
Assigned:	12.01.2024
Approved:	25.01.2024

prof. RNDr. Daniel Ševčovič, DrSc.
Guarantor of Study Programme

.....
Student

.....
Supervisor

Acknowledgement First and foremost, I would like to express my sincere gratitude to my supervisor, Cyril Izuchukwu Udeani, PhD, for his professional guidance, valuable advice, and constant willingness to offer support, all of which were crucial to the completion of this thesis. I am also deeply grateful to everyone who stood by me and supported me during the writing of this thesis and throughout my studies.

Abstract

MUCHOVÁ, Hanna: Machine learning for predictive modeling in time series analysis [Master Thesis], Comenius University in Bratislava, Faculty of Mathematics, Physics and Informatics, Department of Applied Mathematics and Statistics; Supervisor: Cyril Izuchukwu Udeani, PhD., Bratislava, 2025, 74p.

In this master's thesis, we aim to develop predictive models capable of accurately forecasting the price of the SPY ETF. We employed three methods of varying complexity. The first is the Autoregressive Integrated Moving Average (ARIMA) model, a statistical method. The second is K-Nearest Neighbors (KNN), a model traditionally used for classification but also applicable to regression. The final method, and arguably the most complex, is Long Short-Term Memory (LSTM), a deep learning model. For each method, we first provide an overview of the theoretical background, followed by experimentation to identify the optimal model. This involves adjusting hyperparameters and selecting the best model based on evaluation metrics. Finally, we evaluate the performance of each selected model and identify the one that delivers the best results based on the hyperparameter combinations we tested.

Keywords: machine learning, predicting, regression, time series, ARIMA, KNN, LSTM

Abstrakt v štátnom jazyku

MUCHOVÁ, Hanna: Machine learning pre prediktívne modelovanie v analýze časových radov [Diplomová práca], Univerzita Komenského v Bratislave, Fakulta matematiky, fyziky a informatiky, Katedra aplikovanej matematiky a štatistiky; školiteľ: Cyril Izuchukwu Udeani, PhD., Bratislava, 2025, 74s.

V tejto diplomovej práci sa zameriavame na vývoj prediktívnych modelov schopných presne predpovedať cenu SPY ETF. Použili sme tri metódy rôznej zložitosti. Prvou je model Autoregresného integrovaného kĺzavého priemeru (ARIMA), štatistická metóda. Druhou je K-Najbližších susedov (KNN), model bežne používaný pre klasifikáciu, ale rovnako aplikovateľný aj na regresiu. Poslednou metódou, a pravdepodobne najkomplexnejšou, je Long Short-Term Memory (LSTM), model hlbokého učenia. Pre každú metódu najprv poskytujeme prehľad teoretického pozadia, následne vykonávame experimenty na identifikáciu optimálneho modelu. To zahŕňa nastavenie hyperparametrov a výber najlepšieho modelu na základe hodnotiacich metrík. Nakoniec vyhodnocujeme výkon každého vybraného modelu a identifikujeme ten, ktorý dosahuje najlepšie výsledky na základe nami skúšaných kombinácií hyperparametrov.

Kľúčové slová: strojové učenie, predikovanie, regresia, časový rad, ARIMA, KNN, LSTM

Contents

Table of Parameters **10**

Introduction **13**

1 Data Source and Description **16**

1.1 About the data 16

2 Model evaluation metrics **18**

2.1 Mean Absolute error (MAE) 18

2.2 Root Mean squared error (RMSE) 18

2.3 Mean absolute percentage error (MAPE) 19

2.4 R-squared (R^2) 19

2.5 Akaike Information Criterion (AIC) 20

2.6 Bayesian Information Criterion (BIC) 21

3 Autoregressive Integrated Moving Average model **22**

3.1 Autoregressive model 22

3.2 Moving average model 23

3.3 ARIMA(p,d,q) 23

3.4 Stationarity 24

3.5 Identifying the order of p and q 28

3.6 Function auto_arima() 28

3.7 Models 30

4 K-Nearest Neighbors model **39**

4.1 Steps of KNN algorithm 39

4.2 Data 40

4.3 Sequential feature selection (SFS) 45

4.4 Finding the best k 46

4.5 Models 47

5 Long short-term memory model **51**

5.1 How does the LSTM work 52

5.2 Data preparation 57
5.3 LSTM in Python 57
5.4 Models 59

6 Comparison of models **63**

Concusion **65**

References **67**

Table of Parameters for auto_arima()

start_p (int, optional)	starting value of Autoregressive (AR) term, positive integer
start_q (int, optional)	starting value of Moving average (MA) term, positive integer
d (int, optional)	order of differencing (I) term, if d=None statistical test will be used to determine order, positive integer or None
max_p (int, optional)	maximum value of p, inclusive, positive integer greater than or equal to start_p
max_q (int, optional)	maximum value of q, inclusive, positive integer greater or equal to start_q max_q
max_d (int, optional)	maximum number of differences, positive integer greater than or equal to d
start_P, start_Q (int, optional)	same as non-seasonal parameters start_p and start_q, terms P, Q correspond to seasonal SARIMA model
D (int, optional)	order of seasonal differencing in SARIMA model
max_P, max_Q (int, optional)	same as non-seasonal parameters max_p, max_q and max_d, terms P, Q correspond to seasonal SARIMA model
max_D (int, optional)	same as non-seasonal parameters max_d, term D corresponds to seasonal SARIMA model
max_order (int, optional)	Maximum value of p+q+P+Q if model selection is not stepwise, if the sum of p and q is >= max_order, a model will not be fit with those parameters and will progress to the next combination
m (int, optional)	period for seasonal differencing, whether the data has quarterly (m=4), monthly (m=12) or annual seasonality (m=1) which indicates there is non-seasonality in the data

seasonal (bool, optional)	whether to fit a seasonal ARIMA (True), if we want to fit non-seasonal ARIMA (False)
stationary (bool, optional)	whether the time series is stationary and d should be set to zero
information_criterion (str, optional)	information criterion used to select the best ARIMA model: 'aic'- Akaike Information Criterion, 'bic'- Bayesian Information Criterion, 'hqic'- Hannan-Quinn Information Criterion, 'oob'- Out-of-Bag Error
alpha (float, optional)	level of the test for testing significance
test (str, optional)	type of unit root test to use in order to detect stationarity, if stationary is False and d is None
seasonal_test (str, optional)	this determines which seasonal unit root test is used if seasonal is True and D is None
stepwise (bool, optional)	whether to use the stepwise algorithm to identify the optimal model parameters, it can be significantly faster than fitting all possible hyper-parameter combinations (or a random subset of), stepwise algorithm is less likely to over-fit the model
n_jobs (int, optional)	number of models to fit in case we set stepwise=False, a grid search will be used instead, default is 1, but -1 designates "as many models as possible"
start_params (array-like, optional)	starting parameters for ARMA(p,q), if set to None, the default is given by ARMA._fit_start_params

method (str, optional)	method determines which solver from scipy.optimize is used, and it can be chosen from among the following strings: ‘newton’ for Newton-Raphson, ‘nm’ for Nelder-Mead, ‘bfgs’ for Broyden-Fletcher-Goldfarb-Shanno (BFGS), ‘lbfgs’ for limited-memory BFGS with optional box constraints, ‘powell’ for modified Powell’s method, ‘cg’ for conjugate gradient, ‘ncg’ for Newton-conjugate gradient, ‘basinhopping’ for global basin-hopping solver
trend (str or None, optional)	trend parameter, if with_intercept is True, trend will be used and if with_intercept is False, the trend will be set to a no- intercept value
maxiter (int, optional)	maximum number of function evaluations
offset_test_args (dict, optional)	args to pass to the constructor of the offset (d) test
seasonal_test_args (dict, optional)	args to pass to the constructor of the seasonal offset (D) test
trace (bool or int, optional)	whether to print status on the fits, a value of False will print no debugging information, a value of True will print some and integer values exceeding 1 will print increasing amounts of debug information at each fit
return_valid_fits (bool, optional)	if True, will return all valid ARIMA fits in a list and if False (by default), will only return the best fit
with_intercept (bool or str, optional)	whether to include an intercept term, default is “auto” which behaves like True until a point in the search where the sum of differencing terms will explicitly set it to True or False

Introduction

A time series is a sequence of observations sorted by date. There are many examples of data sets that are classified as time series. Some include daily temperature measurements in a city, weekly sequences of a quantity of products made in a factory, and the monthly revenue of a company. Time series are used in various fields, amongst them economics, engineering, and natural sciences like meteorology and geophysics. There can also be time series with frequencies as small as minutes or even seconds, such as chemical or biological reactions. For instance, tracking glucose levels in blood can have a frequency measured in minutes, while electroencephalography (EEG) captures the electrical activity of the brain and can have a frequency of milliseconds or minutes [5].

A very substantial feature of time series is that, in most cases, the individual observations are dependent. The aim of time series analysis is to analyze these dependencies. This involves creating stochastic and dynamic models for time series data and applying these models in key areas of interest. One of the key areas is the forecasting of future values of a time series [5].

The main distinction from earlier efforts to build problem-solving machines is that modern algorithms learn decision-making rules from data rather than being explicitly programmed by humans, as was the case with expert systems popular in the 1980s [27].

At the start of time series analysis, several prediction methods were often used. These methods focused on the underlying patterns in the data, for example, trends and seasonality. Some of the most popular models used for time series include the Autoregressive (AR) model and the Moving Average (MA) model. These two models later became the base for the Autoregressive Integrated Moving Average (ARIMA) model and later also the Seasonal Autoregressive Integrated Moving Average (SARIMA) model, which dealt with seasonality in the data. Another popular model is the Exponential smoothing (ETS) model that forecasts future values by applying exponentially decreasing weights to past observations, making more recent observations more influential. Generalized Autoregressive Conditional Heteroskedasticity (GARCH) model that captures the pattern of periods that are exhibiting high volatility, while others show low volatility, providing a better estimate of future variances [60].

The models above are still largely used in forecasting of time series since they are rel-

atively easy to interpret. However, they have many limitations when it comes to handling complex, non-linear patterns in data because they rely only on assumptions about the stationarity, linearity, and patterns present in the data, which can prove to be a disadvantage. These limitations can be addressed using modern machine learning techniques.

Machine learning (ML) is a branch of artificial intelligence that focuses on helping computers learn from data and improve their accuracy without being explicitly programmed. Machine learning can solve a wide range of problems, including analyzing images, such as detecting tumors in brain scans, processing text, like automatically flagging offensive comments, enabling real-time speech-to-text translation across multiple languages, and handling time series data for tasks like forecasting trends [18].

They analyze large sets of data and look for potential patterns then use the knowledge for predictions. In time series, we usually want to predict future values based on previously observed values. That can be done by using machine learning due to its ability to process large datasets and very complex patterns. The key advantage of ML is that it continuously evolves based on new information.

There are three main types of Machine learning. The first is Supervised learning, which means the model trains on a labeled dataset where the expected output is paired with input. The model uses the input and output pairs for mapping and uses the information to predict the output for unseen data. A common application of supervised learning is classification. The second type is Unsupervised learning, where the model trains on unlabeled data. The model tries to identify patterns and relationships in the data and makes predictions based on those. Unsupervised learning is commonly used in clustering. The third is Reinforced learning where the learning system, called an agent, needs to pick an action at each time step, based on interactions with the environment, receiving rewards or penalties based on its actions. The agent aims to learn an optimal strategy therefore to choose the action that yields the highest reward over time through trial and error, based on a set of observations that describes the current state of the environment [18].

Our aim is to predict prices in the testing set as accurately as possible, which is, in reality, very hard since no model can predict economic and political changes that directly affect stock market prices. We will be using three models for our analysis and consequently,

compare their performance. The first is the Autoregressive Integrated Moving Average model (ARIMA), which is the simplest of the three models. The second is the K-Nearest Neighbors model (KNN). The last is the Long Short-Term Memory model (LSTM), which is the most complex of the three. Our final goal is to compare the performance of these three algorithms and identify the one with the highest prediction accuracy while taking into account the potential for overfitting.

When a model can make accurate predictions on unseen data, we say it is able to generalize from the training set to the testing set. Most often, we build a model to make good predictions on the training set, and if the training set and testing set share similar characteristics, we expect the model to perform well on unseen data, specifically on the test set. However, by making the model excessively complex, we can artificially increase the accuracy on the training set, thereby risking overfitting.

Increasing model complexity often means capturing every minor variation in the data, which is more likely to be noise rather than true signal. While the training accuracy may look appealing, the model will likely show a higher error rate on the test set due to overfitting, making the model useless in real life [39].

In the first section, we will discuss the source and description of the data we are using. The second section will provide theoretical background on the evaluation metrics used to assess model accuracy and facilitate model comparisons. The following three sections will focus on the three methods outlined above: the ARIMA model, the KNN model, and the LSTM model. The first will be the ARIMA model, the second will be the KNN model and the last will be the LSTM model. In each of these sections, we will cover the theoretical background of the method, data transformations to enable effective implementation, parameter selection for optimal performance, and choose models by comparisons in cases where we have more than one model within one method. The final section compares all the methods, discusses their strengths and limitations, and draws a conclusion on which models performed the best.

1 Data Source and Description

The data we used in this thesis is sourced from Alpha Vantage (https://www.alphavantage.co/documentation/). Our objective is to predict the prices of a financial instrument, specifically the SPDR S&P 500 ETF (Exchange Traded Funds) Trust (SPY). It was the first ETF traded on the U.S. market and was launched in January 1993. We chose the SPY ETF for its ability to mirror the stock market. It tracks the performance of the S&P 500 Index, representing the 500 largest publicly traded companies in the U.S. For the SPY to track the performance of the S&P 500 Index, it holds the same stocks in proportion to their weighting in the index. Since the S&P 500 is a cap-weighted index, which means that each company's stock is weighted according to its total market capitalization relative to the total market capitalization of all companies in the index, the top holdings receive heavier weights in the SPY portfolio. The SPY ETF is one of the most popular and widely traded ETFs, providing investors with exposure to a broad range of industries within the U.S. stock market [50].

The data consists of the date, open price, high price, low price, close price, and volume with daily frequency excluding weekends and public holidays when the stock market is closed. In this study, we focus on the closing price of the SPY. The first data point available is from November 11, 1999, though we will select a shorter time frame because the older the data point is the less relevant it is to present prices. The data has no missing values, so we do not need to adjust it in any way.

To prepare the data for applying any machine learning model, we split the 3258 data points into training and testing sets. We will be working with data from January 1, 2011, to January 1, 2023, which is 12 years, comprising 3,009 data points called training data. The goal is to predict the prices of the SPY ETF from January 1, 2023, to January 1, 2024, covering 249 data points called testing data.

1.1 About the data

If we want to analyze the time series and predict the price development, we first need to plot the time series, which will help us determine what we need to do with the data to prepare it for applying a machine learning model.

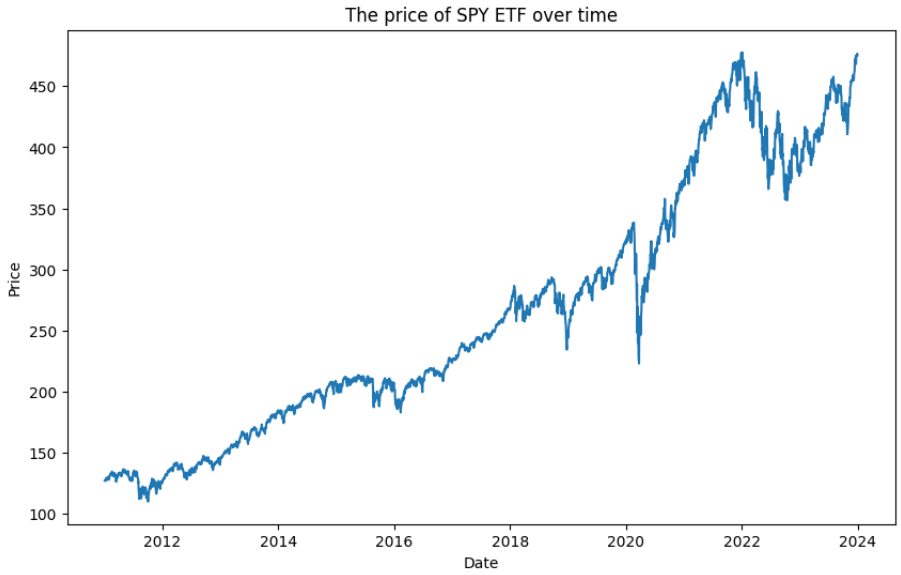


Figure 1: The prices of SPY ETF over time

The time series shows a clear upward trend with periods of increased volatility, specifically around the year 2020, corresponding to the onset of the COVID-19 pandemic. The intensity of price fluctuations appears to vary, with recent years exhibiting larger deviations from the trend, indicating increased volatility. This upward trend suggests that the time series is non-stationary, which is typical in financial data. For some of the models we will employ, this trend needs to be dealt with by transformations such as detrending or differencing to make the time series stationary, thereby ensuring reliable predictions. Additionally, any seasonality in the data would need to be addressed. However, there is no clear evidence of a seasonal pattern in the plot.

2 Model evaluation metrics

Since a crucial part of this thesis is comparing the three models we will be testing, the comparison can be done using various statistical evaluation metrics. We will use Mean absolute percentage error (MAPE), Root mean squared error (RMSE), Mean absolute error (MAE), and R-squared (R^2).

We must know how these metrics work to understand what the values for each test indicate and to compare the models effectively.

All metrics examine the difference between the actual value y and the predicted value $\hat{y}$. This difference is called forecast error and for the i^{th} data point is given by the following equation from [11]:

$$e_i = y_i - \hat{y}_i. \tag{1}$$

2.1 Mean Absolute error (MAE)

Since the difference in Equation (1) can be positive as well as negative, potentially canceling each other out, we calculate the absolute value of the forecast error for each predicted data point [11]. Subsequently, all residuals are aggregated, and their average is computed, resulting in the following equation as presented in [30]:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|e_i| \tag{2}$$

This metric is scale-dependent. A lower MAE indicates that the predictions are close to the actual values, suggesting a good fit of the model. Conversely, a higher MAE signifies a greater difference between the predictions and the actual values, indicating poor model fit [11].

2.2 Root Mean squared error (RMSE)

The Root Mean Squared Error (RMSE) is calculated as the square root of the Mean Squared Error (MSE). One advantage of using RMSE is that it shares the same scale as the original variable, making interpretation more intuitive [30]. However, RMSE remains scale-dependent, meaning its value is influenced by the scale of the data. Additionally, because it squares the differences between actual and predicted values (as shown in Equation (1)), RMSE places greater emphasis on larger errors, making it more sensitive to

significant deviations and outliers [11].

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(e_i)^2} \tag{3}$$

Lower RMSE values suggest that the predicted values closely align with the actual values, indicating good model performance. Conversely, higher RMSE values suggest poor model performance [30].

2.3 Mean absolute percentage error (MAPE)

The Mean Absolute Percentage Error (MAPE) is a scale-independent metric that calculates the percentage error of each data point using the following equation [11]:

$$p_i = 100 \times \frac{e_i}{y_i}, \tag{4}$$

where p_i represents the percentage error. The overall MAPE is then computed as:

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}|p_i|, \tag{5}$$

as detailed by [11].

This calculation obtains errors relative to the actual values, resulting in an error measure expressed as a percentage and, therefore, standardized. MAPE is commonly used to compare models across different forecasting applications. However, there are limitations, such as when evaluating a time series with possible zero values. A zero value in the time series results in an infinite error rate due to Equation (4). Despite these limitations, MAPE provides insight into how the model performs relative to the actual values and allows for easy comparison across different models, regardless of data scaling. The percentage indicates, on average, how much the predictions deviate from actual values in terms of percentage error [11].

2.4 R-squared (R^2)

R-squared (R^2), also known as the coefficient of determination, is a statistical measure that explains how well the independent variables explain the amount of variance that is in the dependent variable. It is commonly used to assess the goodness-of-fit of a regression model.

The sum of squared errors (SSE) measures the total deviation of the observed data points from the predicted values generated by the model. In other words, it quantifies the amount of unexplained variability in the actual values the model fails to account for [8]. It is defined in [8] as:

$$\text{SSE} = \sum_{i=1}^{n} {e_i}^2. \tag{6}$$

The total sum of squares (SST) which measures the overall variability present in the data, or in other words variance of the actual values from their mean, is defined in [8] by the equation:

$$\text{SST} = \sum_{i=1}^{n} (y_i - \overline{y_i})^2, \tag{7}$$

where $\overline{y_i}$ is the mean of the actual values. R^2 is then calculated as the ratio of the variance explained by the model to the total variance present in the actual values.

$$R^2 = 1 - \frac{SSE}{SST} \tag{8}$$

The value of the coefficient R^2 is typically from a range [0,1]. An R^2 value of 1 indicates that the model explains the variability in the actual values perfectly, while an R^2 value of 0 signifies that the model fails to explain any of the variability [8]. According to [14], in some cases, the value of the coefficient can have a negative value, it is only possible if SSE exceeds SST in Equation 8. This would mean that the model performs worse than a simple model that predicts the mean of the train data for all future values [14].

2.5 Akaike Information Criterion (AIC)

Another statistical metric we will employ to compare models is the Akaike Information Criterion (AIC). AIC provides an estimate of the prediction error and, consequently, the relative quality of each model used on the data. This metric will be considered when selecting the best model in Section 3.

$$\text{AIC} = 2k - 2\ln(\hat{L}), \tag{9}$$

where k represents the number of parameters estimated by the model, and $\hat{L}$ is the maximum likelihood estimate, which reflects the likelihood that the model could have produced the observed data. AIC considers both the accuracy of the model's fit and the simplicity of the model. With increasing k, a penalty for the number of parameters

increases. By penalizing complex models, it strives to prevent overfitting. While adding parameters may lead to improvement of the model's fit, it also increases the risk of making the model too specific, leading to the model not being simple enough to generalize to other data [1].

In the term $-2\ln(\hat{L})$ from Equation (9), the model is rewarded for fitting the data well. Better results in a higher likelihood $\hat{L}$, which leads to a smaller value for $-2\ln(\hat{L})$. When comparing models, we will prioritize those with lower AIC values, as it indicates a better balance between good model fit and simplicity [1].

2.6 Bayesian Information Criterion (BIC)

BIC, or Bayesian Information Criterion, is used for model selection. Similar to AIC, BIC includes a penalty for the number of parameters to mitigate the risk of overfitting. However, BIC imposes a more substantial penalty than AIC, depending on the sample size n and the relative magnitudes of n and k, which makes BIC more conservative and favors simpler models [19].

The BIC is defined in [19] as follows:

$$\text{BIC} = k\ln(n) - 2\ln(\hat{L}), \tag{10}$$

where k represents the number of parameters estimated by the model, n is the number of observations, and $\hat{L}$ is the maximum likelihood estimate of the model. A lower BIC value indicates a better model.

3 Autoregressive Integrated Moving Average model

The Autoregressive Integrated Moving Average (ARIMA) model is a linear model composed of two components: the Autoregressive (AR) model of order p and the Moving Average (MA) model of order q. These components are combined into a single ARIMA(p, d, q) model. The I in ARIMA stands for Integrated, indicating that the model can handle unit-root non-stationarity by differencing the time series d times [26].

3.1 Autoregressive model

The Autoregressive (AR) model is a stochastic univariate model that captures the linear dependence between the current values and its past values, known as lags. Lags are delayed values of a time series used to analyze its past behavior and predict future trends [27].

Definition 3.1. *[26] The AR(p) model of order p aims to capture the linear dependence between time series values at different lags and can be written as follows:*

$$AR(p) : y_t = \Phi_0 + \Phi_1 y_{t-1} + \Phi_2 y_{t-2} + \ldots + \Phi_p y_{t-p} + \epsilon_t, \quad \epsilon \sim i.i.d. \tag{11}$$

We define the values of a process at equally spaced times $t,\ t-1, ...,\ t-p$ as $y_t,\ y_{t-1}, ...,\ y_{t-p}$, where y_t is a dependent value we are predicting. Closely resembles multiple linear regression on lagged values of y_t. A linear regression would be as follows:

Definition 3.2. *[61] The general form of the multiple linear regression model is as follows:*

$$y = \Phi_0 + \Phi_1 x_1 + \Phi_2 x_2 + \ldots + \Phi_p x_p + \epsilon. \tag{12}$$

The process described in Definition 3.1 is called an autoregressive process because it is analogical to the regression model described in Definition 3.2, except in Definition 3.2, a dependent variable y is related to a set of independent variables $x_1,\ x_2, ...,\ x_p$ plus a random error term ϵ, which is called a regression model. In this context, y is said to be regressed on $x_1,\ x_2, ...,\ x_p$. Similarly, in Definition 3.1, the variable y is regressed on its previous values, which is why the model is called autoregressive [5].

Definition 3.3. *[27] The AR model follows the characteristic equation:*

$$1 - \Phi_1 z - \Phi_2 z^2 - \ldots - \Phi_p z^p = 0. \tag{13}$$

The inverses of the solution to this polynomial equation of degree p in z are the characteristic roots.

We can check if the AR(p) is a stationary process by checking if all the roots have an absolute value less than 1, otherwise, it is unstable, as discussed in [27]. In which case, we say that the process has a unit-root [26]. All $p + 1$ model parameters are estimated with the least squares method [26].

3.2 Moving average model

In the moving average model MA(q), the order q represents the number of lagged errors of the time series itself. The model expresses the current value y_t of the time series as a linear combination of past forecast errors.

According to [26], the model's name is related to the fact that each value of y_t is represented as a weighted moving average of the past q innovations. MA(p) processes are the weighted sum of white noise variables that are stationary, which is the reason why they are always stationary, as noted in [26].

Definition 3.4. *[26] An MA model of order q uses q past disturbances rather than lagged values of the time series in a regression-like model, as follows:*

$$MA(q) : y_t = \mu + \epsilon_t + \theta_1\epsilon_{t-1} + \theta_2\epsilon_{t-2} + \ldots + \theta_q\epsilon_{t-q}, \quad \epsilon \sim i.i.d.. \tag{14}$$

Where μ is the mean of the time series, ϵ_t is the white noise error term at time t, and θ_i, where $i = 1, ..., q$ are the parameters for lagged error terms [30].

The process contains $q + 1$ unknown parameters that need to be estimated from the data using maximum likelihood estimation (MLE) [27].

3.3 ARIMA(p,d,q)

Autoregressive integrated moving-average (ARIMA(p,d,q) model is a combination of AR(p) and MA(q) processes. Since these processes are complementary, using them in one model reduces the number of parameters used and the risk of overfitting. The model can also eliminate the problem of unit-root non-stationarity by differentiating the time series values d times. Differentiating the values once is equivalent to using the model ARMA(p,q) with the first differences of the series [27].

Definition 3.5. *[27] Using y' to denote the original series after non-seasonal differencing d times, then ARIMA(p,d,q) model is simply:*

$$\begin{aligned} ARIMA(p,d,q): \quad y'_t &= AR(p) + MA(q) \\ &= \Phi_0 + \Phi_1 y'_{t-1} + \ldots + \Phi_p y'_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} \\ &\quad + \ldots + \theta_q \epsilon_{t-q}, \quad \epsilon \sim i.i.d. \end{aligned}$$

If we denote variable L as a Lag operator with a formula defined in [49] as:

$$L^k y_t = y_{t-k}, \tag{15}$$

we can rewrite Formulas (11) and (14):

$$\text{AR(p): } (1 - \phi_1 L - \phi_2 L^2 - \cdots - \phi_p L^p) y_t = \epsilon_t \tag{16}$$

$$\text{MA(q): } y_t = (1 + \theta_1 L + \theta_2 L^2 + \cdots + \theta_q L^q) \epsilon_t \tag{17}$$

and consequently simplify formula in Definition 3.5:

$$(1 - \phi_1 L - \phi_2 L^2 - \cdots - \phi_p L^p)(1 - L)^d y_t = (1 + \theta_1 L + \theta_2 L^2 + \cdots + \theta_q L^q) \epsilon_t. \tag{18}$$

3.4 Stationarity

The ARIMA model requires stationarity, meaning the time series is required to have constant mean, variance, and autocorrelation over time, that is, the statistical properties do not change over time implying that if a time series is stationary then it does not have a trend or any seasonal effects [27].

As we can see 1, our series is not stationary because we have a growing trend, which needs to be fixed before using the ARIMA model.

The data in a time series often contains a mix of patterns like trend, seasonality, and unsystematic noise, all of which can be decomposed into several components. In Python, we will use the function *seasonal_decompose()* from the package *statsmodels.tsa.api* [27].

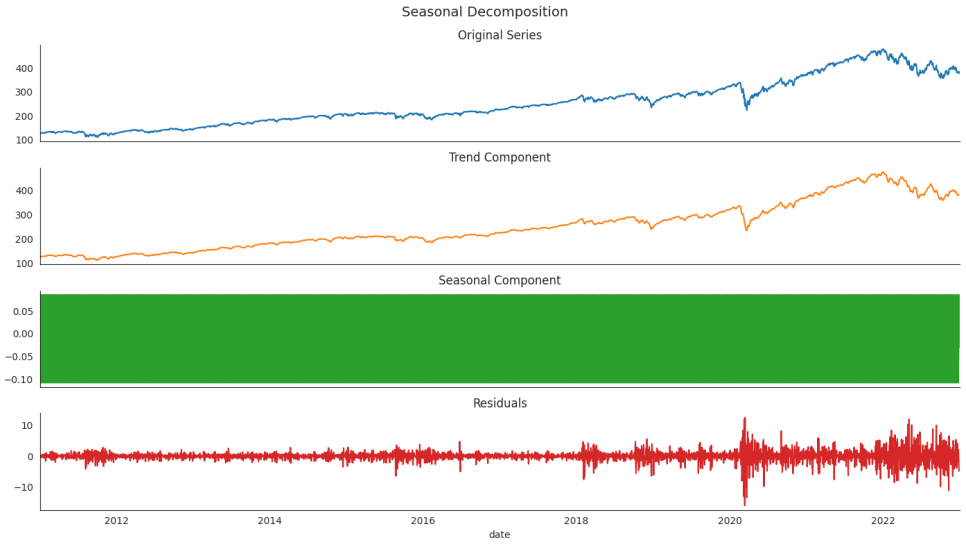


Figure 2: Decomposition of the time series

Figure 2 confirmed our assumption about the growing trend. In addition, it seems like there is no seasonality, which agrees with our assumption based on Figure 1.

3.4.1 How to make the time series stationary

Before we start transforming the time series, we should briefly remind ourselves how the time series originally looked by plotting it in Figure 3.



Figure 3: Original prices of SPY ETF

To achieve stationarity, we often need to transform the time series in several steps. First, we will make a logarithmic transformation that applies the natural logarithm on the data, stabilizing variance, therefore, making it more constant over time. The transformation also helps with exponential growth through its linearization. We say that a time series is trend-stationary if there is a long-run linear trend, which makes it easier for

ARIMA models since they assume linear trends [27].



Figure 4: Logarithmized prices of SPY ETF

Next, we need to determine whether a unit root is present, and for that, we will use the Augmented Dickey-Fuller (ADF) test.

3.4.2 Augmented Dickey-Fuller (ADF) test

Augmented Dickey-Fuller test is a statistical test, augmented for complex models by including lagged terms of the differenced series to account for autocorrelation [27]. The null hypothesis is that the time series has a unit-root signifying non-stationarity of the data, against the alternative of stationarity of the data. The ADF test equation can be written, as in [38], as:

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \sum_{i=1}^{p} \delta_i \Delta y_{t-i} + \epsilon_t, \tag{19}$$

where

$$\Delta y_t = y_t - y_{t-1}, \tag{20}$$

is the first difference of the time series, y_t, is the dependent time series value at time t. A constant α is an intercept term, and β is the coefficient of a time trend if there is any. The coefficient γy_{t-1} is for the lagged level of the time series used to test for unit-root, γ is a coefficient that determines if the time series is stationary consequently, the term y_{t-1} indicates whether the current value is dependent on the previous value. We can adjust the null hypothesis, if $\gamma = 0$, that means there is nothing that influences the previous value, then $H_0 : \ \gamma = 0$. Coefficients $\delta_1, \ldots, \delta_p$ for the lagged differences to account for higher order autocorrelation, and ϵ_t is for white noise, which is a random error term. In Equation (18), the $(1 - L)^d$ is a differencing operator that removes the unit-root.

If the p-value of the ADF test is greater than the chosen significance level, in our case 5% or the ADF test statistic is less than a corresponding critical value, we fail to reject the null hypothesis, suggesting the time series is non-stationary. Conversely, suppose the p-value is less than the chosen significance level or the test statistic is greater than the critical value. In that case, we reject the null hypothesis, indicating that the time series is stationary [38].

ADF Statistic:	-0.973347
p-value:	0.762861
Critical values:	
1%:	-3.432535
5%:	-2.862506
10%:	-2.567284

Table 1: Results of ADF test on logarithmized original data

As we can see in Table 1, the p-value is more than 5%, suggesting we have to reject the null hypothesis that states that the time series is stationary. If we compare ADF statistics with critical values, we come to the same conclusion no matter the chosen significance level, since the statistic is greater than the critical values. Based on the results of the ADF test, the time series needs to be differentiated at least once.



Figure 5: Logarithmized returns of SPY ETF

To determine whether a unit root is still present in the data, we perform the Augmented Dickey-Fuller (ADF) test on the logarithmized and once-differentiated time series.

As we can see in Table 2, the p-value is less than 5%, which means we cannot reject the

ADF Statistic:	-15.916475
p-value:	8.013790×10^{-29}
Critical values:	
1%:	-3.432535
5%:	-2.862506
10%:	-2.567284

Table 2: Results of ADF test on logarithmized and differentiated data

null hypothesis, so according to the test, the time series is stationary and differentiating once was enough. If we compare ADF statistics with critical values, we come to the same conclusion since the statistic is significantly smaller than all critical values. To summarize the test signified, there is no need to differentiate the data in time series again, so the order of integration is 1 or $I(1)$, where I is the I present in ARIMA with order $d = 1$.

We stabilized the variance of the time series with a logarithmic transformation. Next, we computed the differences between two consecutive data points, thereby stabilizing the mean by removing changes and eliminating trend. As shown in Figure 5, there is no longer any trend or any variance over time, meaning the statistical properties such as variance and mean remain constant over time. Consequently, we can now apply the ARIMA model effectively.

3.5 Identifying the order of p and q

We tend to look at the autocorrelation function (ACF) and partial autocorrelation function (PACF) to identify the best order of p and q.

As illustrated in Figure 6, the ACF and PACF do not provide substantial information for determining the optimal orders of p and q.

3.6 Function auto_arima()

To find the best ARIMA model, we will use a Python function *auto_arima()* from the package *pmdarima.arima*. The function simplifies the process of selecting the best ARIMA model by automatically searching through all possible parameter combinations and choos-

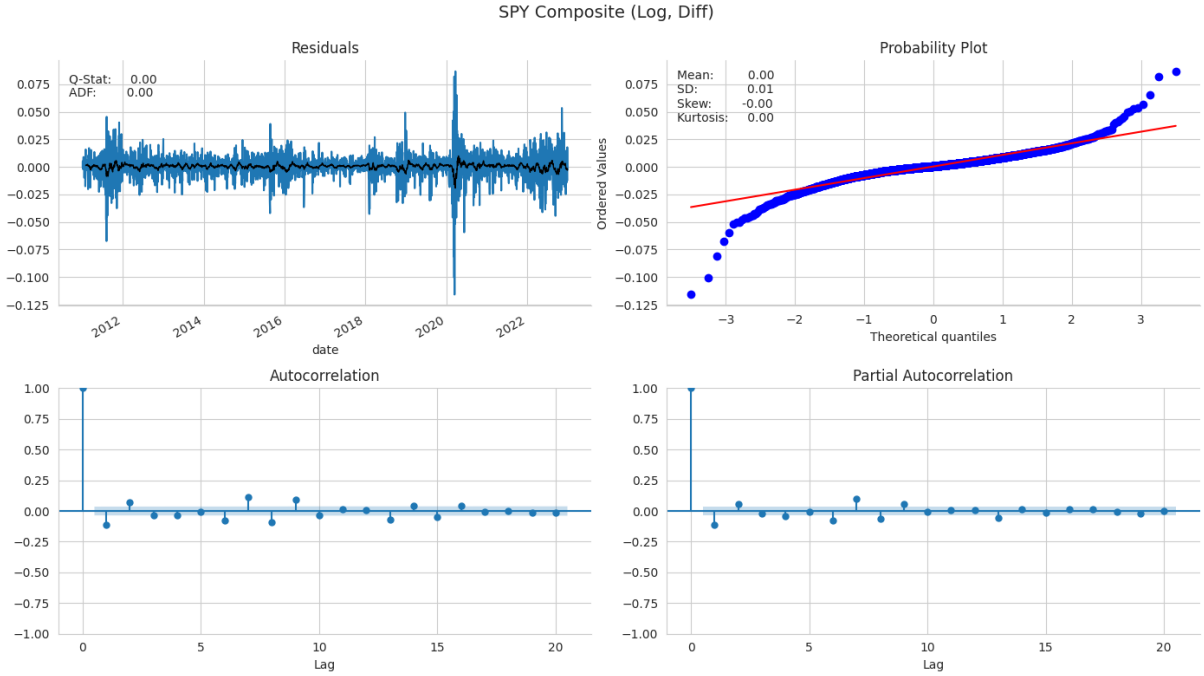


Figure 6: ACF and PACF

ing the one that optimizes a favored criterion, either the Akaike Information Criterion (AIC) or the Bayesian Information Criterion (BIC) [41]. The function *auto_arima()* has several parameters; if all are set to default, the function is, as in [41], as follows:

$$
\begin{aligned}
auto_arima(y, X = None, start_p = 2, d = None, start_q = 2, max_p = 5,\\
max_d = 2, max_q = 5, start_P = 1, D = None, start_Q = 1,\\
max_P = 2, max_D = 1, max_Q = 2, max_order = 5, m = 1,\\
seasonal = True, stationary = False, information_criterion =' aic',\\
alpha = 0.05, test =' kpss', seasonal_test =' ocsb', stepwise = True,\\
n_jobs = 1, start_params = None, trend = None, method =' lbfgs',\\
maxiter = 50, offset_test_args = None,\\
seasonal_test_args = None, suppress_warnings = True,\\
error_action =' trace', trace = False, random = False,\\
random_state = None, n_fits = 10, return_valid_fits = False,\\
out_of_sample_size = 0, scoring =' mse', scoring_args = None,\\
with_intercept =' auto', sarimax_kwargs = None, **fit_args)
\end{aligned}
$$

Before we can identify the optimal values for p and q, it is essential to understand the significance of these parameters in *auto_arima()* function. Most of them are in the Table of Parameters on page 9.

The function uses Kwiatkowski–Phillips–Schmidt–Shin, Augmented Dickey-Fuller, or Phillips–Perron test to find the order of differencing d, then searches through all possible combinations of parameters within the user-specified ranges max_p, max_q, start_p, and start_q and finds the best p and q. If there is seasonality in the data it identifies P and Q hyperparameters after conducting the Canova-Hansen test to determine the optimal order of seasonal differencing D. If we set start_p and start_q to some number, the algorithm might still search outside these initial values if it determines there might be better model outside the bounds, these values are more like guidelines where to start the search. The function uses stepwise search (if not defined otherwise) to minimize AIC or BIC, which are evaluation metrics used to compare models. They both consider model fit and at the same time penalize the model for complexity; the more parameters we have, the more it penalizes the model as stated in section 2.5 and 2.6 [41].

The stepwise algorithm starts with the initial model using the selected or default start_p and start_q and other parameters the user may have defined. Initially, four models are checked if there is no seasonality: ARIMA(0,d,1), ARIMA(1,d,0), ARIMA(0,d,0), ARIMA(2,d,2) the one with the smallest minimization criteria is selected as current best model and then the function evaluates different ARIMA models and increases or decreases the p and q parameters based on how each change impacts the chosen criterion (AIC or BIC) [54]. If adding more complexity does not significantly improve the model, the stepwise search allows the user to revert to simpler models. This method is faster than brute-force grid search, which would test all possible combinations of p and q parameters.

3.7 Models

3.7.1 No restrictions on p and q

We have already discussed that there is no seasonality, so we set seasonal=False. From the section 3.4.1, we know $d = 1$. Initially, we put no restrictions on parameters p and q.

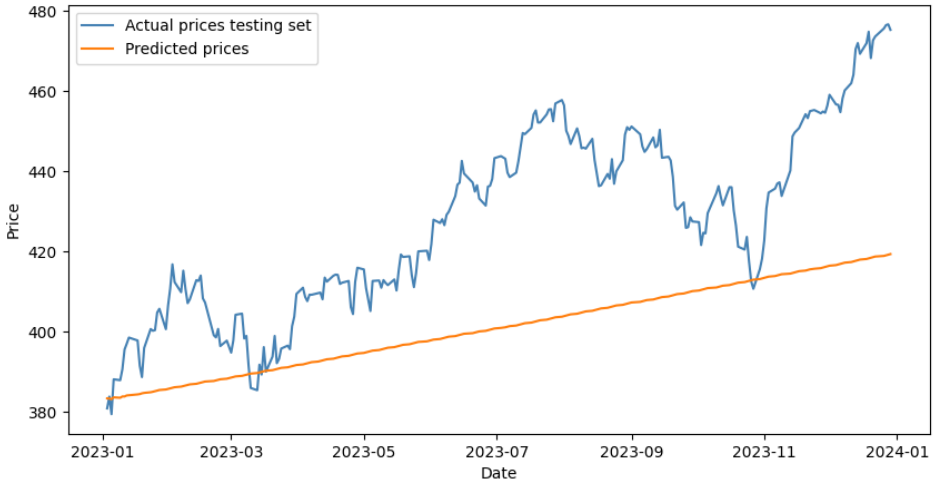
To find the optimal model, we use the following function with the specified attributes:

$$auto_arima(spy_log, seasonal = False, d = 1, trace = True, \\ supress_warnings = True) \quad ,$$

where *spy_log* is the logarithmically transformed time series of SPY ETF closing prices. The function *auto_arima()* determined ARIMA(1,1,5) intercept to be the best model with the given parameters. If we want to see how good the chosen model is, we have to evaluate it based on some metrics.



a) SPY ETF time series and prediction



b) Predicted prices from 1.1.2023 to 1.1.2024

Figure 9: Prediction of SPY ETF prices using ARIMA(1,1,5) intercept model

ARIMA(1,1,5) intercept:			
AIC:	-18 554.06	MAE:	26.4566
BIC:	-18 505.99	RMSE:	30.4542
Ljung-Box p-value:	100%	MAPE:	6.03%
Ljung-Box statistics:	0.049896	R^2:	-0.7476

Table 3: Evaluation metrics for ARIMA(1,1,5) intercept

3.7.2 Partial restrictions on p and q

For the second model, we apply partial restrictions on p and q. To find the optimal model, we use the following function with the specified attributes:

$$auto_arima(spy_log, max_p = 7, max_q = 7, seasonal = False, d = 1, trace = True,$$
$$error_action =' ignore', supress_warnings = True) \quad .$$

The function *auto_arima()* determined ARIMA(3,1,6) intercept to be the best model with restrictions max_p and max_q. If we want to see how good the chosen model is, we have to evaluate it based on some metrics.



Figure 10: a) SPY ETF time series and prediction

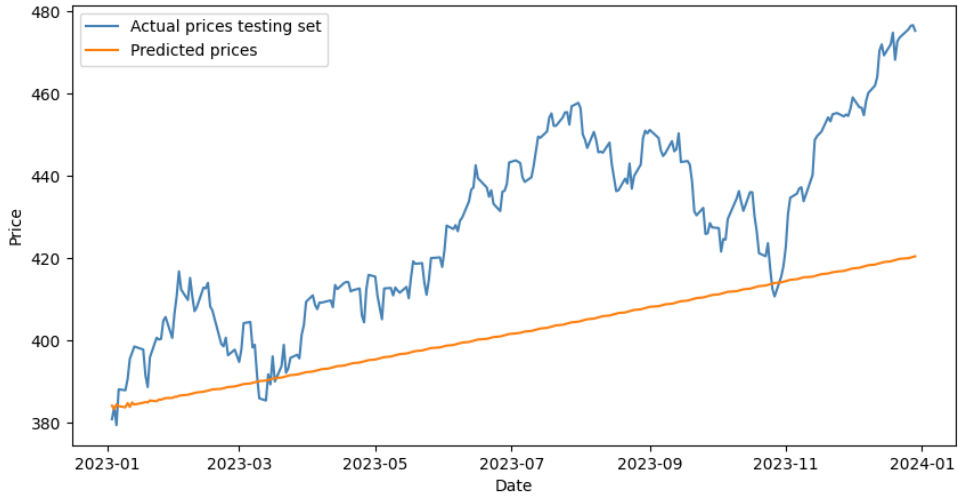


Figure 11: b) Predicted prices from 1.1.2023 to 1.1.2024

Figure 12: Prediction of SPY ETF prices using the ARIMA(3,1,6) intercept model

ARIMA(3,1,6) intercept:			
AIC:	-18 604.02	MAE:	25.7241
BIC:	-18 537.93	RMSE:	29.7269
Ljung-Box p-value:	100%	MAPE:	5.87%
Ljung-Box statistics:	0.027426	R^2:	-0.6651

Table 4: Evaluation metrics for ARIMA(3,1,6) intercept

3.7.3 Restrictions on start_p and start_q

For the next model, we apply additional restrictions on p and q, by specifying $start_p$, $start_q$. To find the optimal model, we use the following function with the specified attributes:

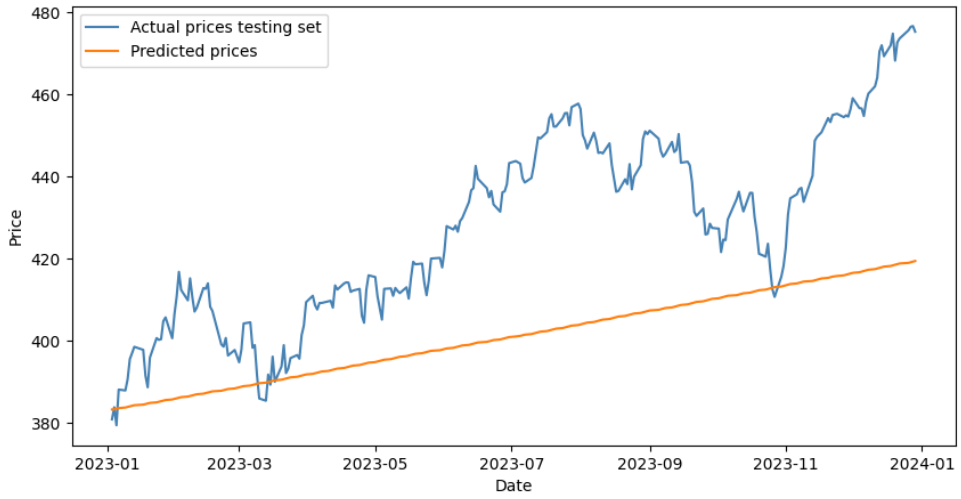
$$auto_arima(spy_log, start_p = 1, start_q = 1, max_p = 7, max_q = 7,$$
$$seasonal = False, d = 1, trace = True, error_action =' ignore',$$
$$supress_warnings = True) \quad .$$

The function *auto_arima()* determined ARIMA(2,1,0) intercept to be the best model with restrictions on the parameters p and q. If we want to see how good the chosen model

is, we have to evaluate it based on some metrics.



a) SPY ETF time series and prediction



b) Predicted prices from 1.1.2023 to 1.1.2024

Figure 15: Prediction of SPY ETF prices using ARIMA(2,1,0) intercept model

3.7.4 Stepwise algorithm set to False

For the next model, we set the parameter $stepwise = False$, and we will not impose restrictions on p and q. As a result, fitting this model may be slower. To identify the optimal model, we use the following function with the specified attributes:

$$auto_arima(spy_log, seasonal = False, d = 1, trace = True, stepwise = False,$$

ARIMA(2,1,0) intercept:			
AIC:	-18 540.53	MAE:	26.3575
BIC:	-18 516.50	RMSE:	30.3652
Ljung-Box p-value:	100%	MAPE:	6.01%
Ljung-Box statistics:	0.064823	R^2:	-0.7374

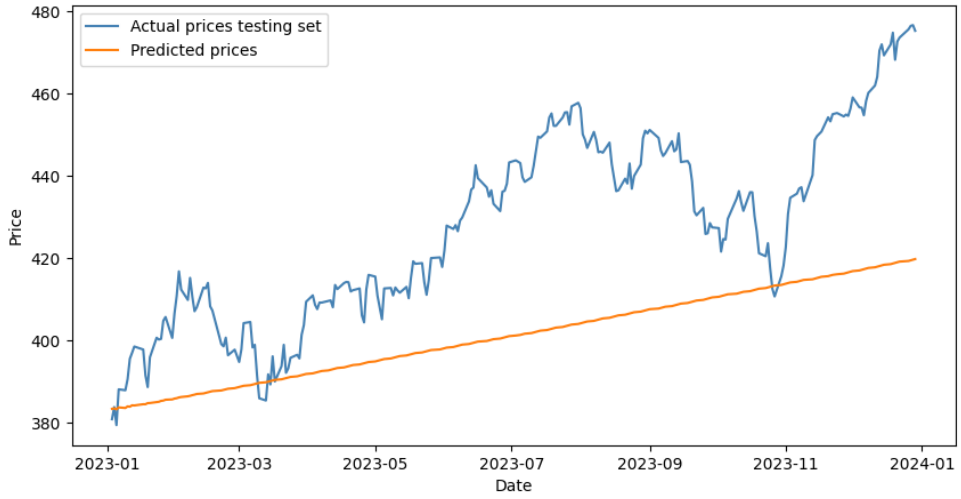
Table 5: Evaluation metrics for ARIMA(2,1,0) intercept

$$error_action =' ignore', supress_warnings = True) \quad .$$

The function *auto_arima()* determined the ARIMA(1,1,4) intercept to be the best model. If we want to assess the chosen model, we have to evaluate its performance using relevant metrics.



a) SPY ETF time series and prediction



b) Predicted prices from 1.1.2023 to 1.1.2024

Figure 18: Prediction of SPY ETF prices using ARIMA(1,1,4) intercept model

ARIMA(1,1,4) intercept:			
AIC:	-18 552.21	MAE:	26.2069
BIC:	-18 510.15	RMSE:	30.1997
Ljung-Box p-value:	100%	MAPE:	5.98%
Ljung-Box statistics:	0.049523	R^2:	-0.7185

Table 6: Evaluation metrics for ARIMA(1,1,4) intercept

3.7.5 Stepwise algorithm set to False and Random set to True

For the final model, we set the parameter $stepwise = False$ and additionally specified $random = True$, without imposing restrictions on p and q. Setting $random = True$ causes the algorithm to test a randomly selected set of hyperparameter combinations, while $stepwise = False$ ensures a purely random search rather than a guided one. To identify the optimal model, we use the following function with the specified attributes:

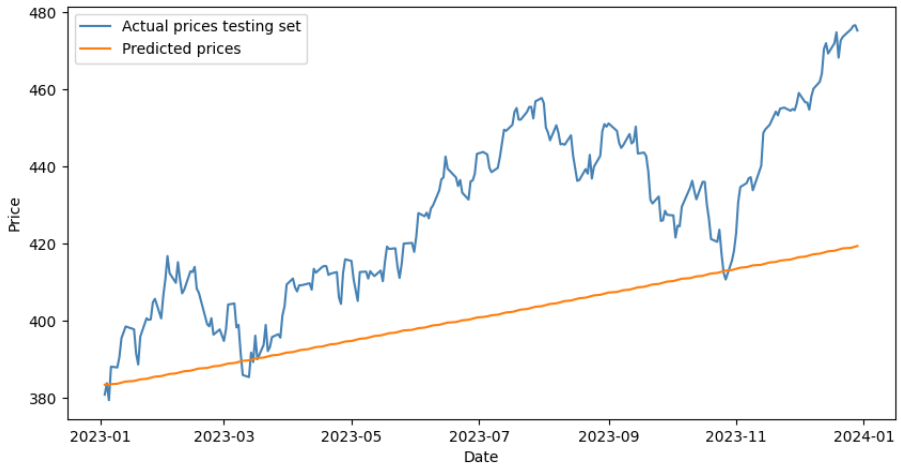
$$auto_arima(spy_log, seasonal = False, d = 1, trace = True, stepwise = False,$$
$$error_action =' ignore', supress_warnings = True, random = True,$$

$random_state = 3)$.

The function *auto_arima()* determined ARIMA(4,1,0) intercept to be the best model. If we want to see how good the chosen model is, we have to evaluate it based on evaluation metrics.



a) SPY ETF time series and prediction



b) Predicted prices from 1.1.2023 to 1.1.2024

Figure 21: Prediction of SPY ETF prices using ARIMA(4,1,0) intercept model

ARIMA(4,1,0) intercept:			
AIC:	-18 543.68	MAE:	26.4006
BIC:	-18 507.63	RMSE:	30.4050
Ljung-Box p-value:	100%	MAPE:	6.02%
Ljung-Box statistics:	0.05778	R^2:	-0.7420

Table 7: Evaluation metrics for ARIMA(4,1,0) intercept

3.7.6 Comparing the models

Since all the figures are very similar, we must compare all evaluation metrics and search for the lowest values of each one to compare all five models.

ARIMA	(1,1,5):	(3,1,6):	(2,1,0):	(1,1,4):	(4,1,0):
AIC:	-18 554.06	-18 604.02	-18 540.53	-18 552.21	-18 543.68
BIC:	-18 505.99	-18 537.93	-18 516.50	-18 510.15	-18 507.63
MAE:	26.4566	25.7241	26.3575	26.2069	26.4006
RMSE:	30.4542	29.7269	30.3652	30.1997	30.4050
MAPE:	6.03%	5.87%	6.01%	5.98%	6.02%
R^2:	-0.7476	-0.6651	0.7374	-0.7185	-0.7420

Table 8: Comparing evaluation metrics for ARIMA models

From Table 8 we can see that R^2 is negative and as we already mentioned in Subsection 2.4 that happens in cases where the model performs worse than a model that for all future values predicts the mean of training data [45]. The metric R^2 assumes the evaluation is done on the same data the model was trained on [14]. While comparing all the ARIMA models, we will rather focus on metrics like AIC, BIC, MAE, and RMSE. Based on the evaluation metrics, the best ARIMA model is ARIMA(3,1,6) intercept.

4 K-Nearest Neighbors model

Next, we use the K-Nearest Neighbors (KNN) model to predict the SPY ETF prices. K-Nearest Neighbors is a supervised non-parametric distance-based algorithm, first used for classification; later, it was extended to cover regression [3]. The algorithm has been effectively used in different research areas, such as financial modeling, image interpolation, and visual recognition. Despite being a simple algorithm, it has been successfully applied for time series forecasting.

In Python, we used the function *KNeighborsRegressor()* in *Scikit-Learn*, where (by default) the Euclidean distance (L_2 norm) is used to measure the distance of a new data point to a training set that contains target values. Euclidean distance is the most widely used distance measures; it can be mathematically expressed, as presented in [43], as follows:

$$d_{Euclidean}(\mathbf{X}, \mathbf{Y}) = ||\mathbf{X} - \mathbf{Y}||_2 = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}. \quad (21)$$

Here, the variables x and y in this formula represent two data points with n dimensions, indicating there are n explanatory variables in the dataset. Many other distances can be used, including Manhattan distance (L_1 norm), the sum of the pairwise distances, which can be expressed, as presented in [43], as follows:

$$d_{Manhattan}(\mathbf{X}, \mathbf{Y}) = ||\mathbf{X} - \mathbf{Y}||_1 = |x_1 - y_1| + |x_2 - y_2| + \cdots + |x_n - y_n|. \quad (22)$$

It is called the Manhattan distance because it represents the total street distance a person would have to travel in midtown Manhattan, where streets are arranged in a grid, to get from one point to another.

The parameter k, in KNN, indicates the number of neighbors the algorithm will use. The algorithm then computes the k nearest neighbors by computing the distances between a new data point and every data point in the training set; the k points with the lowest distance from the new point will be selected. Predictions are then made by either taking the average or weighted average of the target values of the k nearest neighbors [30].

4.1 Steps of KNN algorithm

- Choose the number of neighbors (k): Decide how many neighbors to look at when making a prediction.

- Calculate the distance: Measure the distance between the data point you want to classify and all the training data points.
- Sort and find the nearest neighbors: Sort all the distances in ascending order. Pick the top k closest data points.
- Vote for classification or average for regression. For classification, each of the k neighbors votes for its class. The class with the most votes wins. For regression, take the average (or weighted average) of the neighbors' values.
- Return the result: The final predicted class (or value) is returned based on the votes or average.

4.2 Data

We have already discussed the properties of our data in Section 1. However, since we are applying the KNN model, we need to transform the data differently, for example, by creating new features, checking stationarity, and scaling.

4.2.1 Feature engineering

For KNN to make good predictions, we need create new features, often called feature engineering. First, we will add new features that we extract from the column *Date*. To begin with, we introduce the following features: $Year$ (from 2011 to 2024), $Month$ (numbers from 1 to 12), and Day (numbers from 1 to 31). Additionally, we determine the day of the week for each date; this feature will be called Day_of_week and will use values from 0 for Monday to 4 for Friday. We will also calculate the day of the year, labeled Day_of_year, with values ranging from 1 to 366. All of these features are cyclical; they repeat after a period in time, e.g., a feature month with values from 1 to 31 repeat after every year. The issue with using cyclical features without transformation lies in the fact that even though January (1) and December (12) are numerically far apart, they are close time-wise. This applies to all numerical time features; the only exception is $Year$ since it does not have values that are cyclically repeating.

There are several options of how to deal with these features, one of which is to use one-hot encoding on features that have distinctive categories. It creates binary variables

for each category, which takes a value of 1 if the category is present and 0 otherwise [40]. This method creates as many features as we have categories, increasing dimensionality, which would considerably worsen the computability of the KNN model. This method does not capture the cyclical nature of the features; hence, we will not use this approach.

Another possibility is to use trigonometric coding. In this approach, the first step involves transforming the value to radians to represent every value on the circle's circumference, followed by calculating sine or cosine. The trigonometric transformations, as presented in [10], are as follows:

$$\text{Sine transformation} = sin\left(\frac{2\pi \times \text{Value}}{\text{Cycle length}}\right) \tag{23}$$

$$\text{Cosine transformation} = sin\left(\frac{2\pi \times \text{Value}}{\text{Cycle length}}\right) \tag{24}$$

The cycle length depends on a particular feature. The cycle length of feature $Month$ is 12; the feature $Week$ is 53, the Day is 31, the Day_of_week is 7, and the Day_of_year is 366.

Next, we include features that indicate whether a date is the last day or the first day of the month, which we name Is_month_start and Is_month_end, with values of either 1 for true or 0 for false. At the start of each month, fresh money inflows are directed to mutual funds, which can explain the surge in stock prices during this period. Another useful feature is identifying whether a date marks the beginning or end of a business week, specifically Mondays and Fridays. Some believe that Monday is the most favorable day to purchase stocks, as a significant amount of bad news typically emerges over the weekend; this phenomenon is known as the Monday effect or weekend effect. Therefore, we will add a new feature, Mon_Fri, which is 1 if the day is Monday or Friday, and 0 otherwise. In contrast, Tuesdays generally yield higher returns, so we will introduce a feature that assigns a value of 1 if the day of the week is Tuesday, and 0 otherwise, named $Tuesday$.

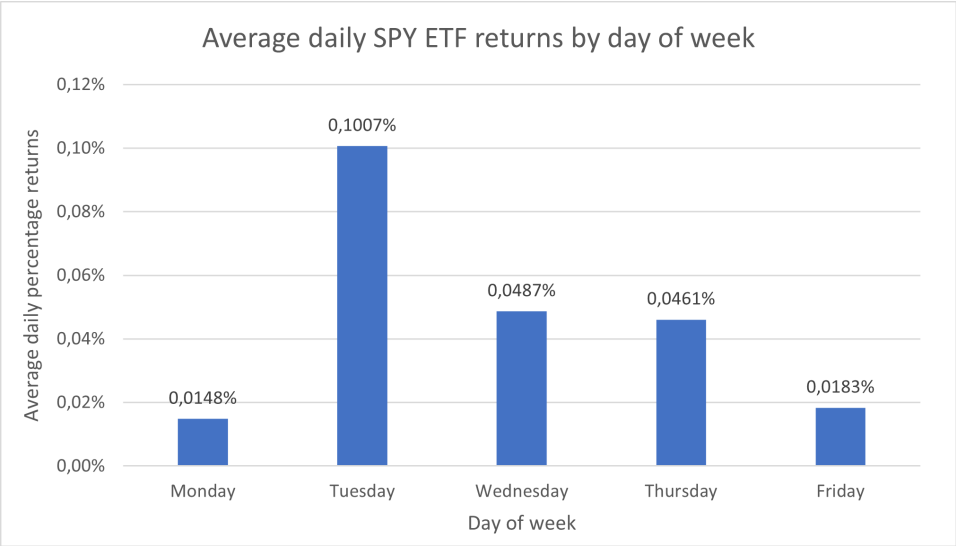


Figure 22: SPY ETF performance by day of week 2011 to 2023

As illustrated in Figure 22, the returns of the SPY ETF during our time frame tend to be lower on Mondays and Fridays and increase on Tuesdays.

Another useful feature could be to examine which months have the best and worst performances.

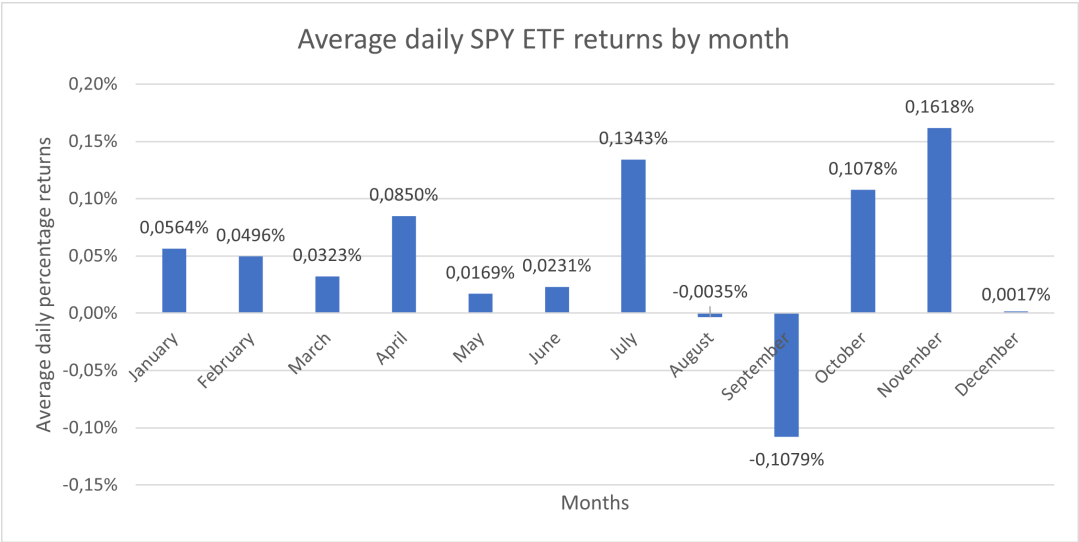


Figure 23: SPY ETF performance by months 2011 to 2023

As shown in Figure 23, *September* has a negative average daily percentage of returns; we added a new feature called *September*. We will add a feature *Aug_Dec* because these months have average daily returns close to 0, and *July_Nov* because they have the highest average daily returns. We also consider the start and end of a quarter since companies

release their earnings reports and financial statements at the end of each quarter. Institutional investors, such as pension funds, mutual funds, and hedge funds, frequently rebalance their portfolios at the end of the quarter, which can significantly affect the prices of stocks. We added the features $Is_quarter_start$ and $Is_quarter_end$, with values 1 if true or 0 otherwise.

The total number of stocks traded in a day is also essential information, which is why we add feature $Volume$ from the original data obtained from Alpha Vantage. Exponential Moving Average (EMA) could also be helpful, since it captures underlying trends and reduces noise, helping the model focus on recent and therefore relevant data. Adding feature Ema on different time horizons Ema_5 (one trading week), Ema_21 (one trading month), and Ema_63 (three trading months) will improve the accuracy of the model by differentiating short-term, medium-term, and long-term trends.

The last features we added include lagged values for the time series. This enables us to capture trends, seasonality, and the impact of past events, ultimately improving forecast accuracy. It also makes the model less sensitive to outliers and more robust to the complexities of real-life data. We will add Lag_1 to the closing prices from the previous day and Lag_2 from the day before, and use forward fill for missing values, which occurred during the shift, which will prevent data leakage to the testing set.

4.2.2 Stationarity

We need to check all numerical features for stationarity. In the KNN algorithm, distance plays a crucial role in predicting new data point values. If the time series has a global trend, there is a strong possibility that future data points will be out of range of the historical data. Since KNN calculates predicted values as an average of the target values of all k neighbors, it is theoretically impossible for the average to exceed the highest historical value, resulting in reduced model accuracy. Therefore, KNN is not suitable for forecasting with a trend [36]. A way to prevent that from happening is similar to fixing non-stationarity in Section 3.4, by differencing the values as in Equation (20).

Most of the features we created are categorical, such as Is_month_start and Mon_Fri. These features do not exhibit a trend and do not influence distance calculations in the KNN algorithm [20]. We need to perform the ADF test on all the numerical features

Ema_5, Ema_21, Ema_63, Lag_1, Lag_2, $Volume$ since we already know that the target variable is non-stationary. We checked cyclical features, such as Day_of_week and Day_of_year for stationarity, and all of them have p-values less than 5% and, therefore, are stationary.

	Ema_5	Ema_21	Ema_63	Lag_1	Lag_2	$Volume$
ADF Statistic:	1.6059	1.7994	1.8948	1.6059	1.5974	-4.4331
p-value:	0.9979	0.9983	0.9985	0.9979	0.9979	0.0003

Table 9: Results of ADF test on all numerical features

As can be seen in Table 9, the first five features are not stationary; however, the last one, $Volume$, has a p-value around 0%, meaning it is already stationary, indicating we do not have to transform this feature. After logarithmic transformation and differencing of the non-stationary features, according to the ADF test, all of them are now stationary.

4.2.3 Scaling

Since KNN is a distance-based model, scaling has to be done on numerical and cyclical features. If the ranges of values in these features are too far apart, the model will be biased towards attributes that have larger scales of measurement, and therefore will perform poorly [36]. Scaling inputs helps avoid situations where one or several features dominate others because of an apparent larger range and hardly pick up contributions of the smaller scale variables, even if they are strong.

There are several different scalers in the package *scikit-learn*. The first possibility is the min-max scaler (*MinMaxScaler*), otherwise called normalization. It is the simplest of them. The scaler scales the values to range from 0 to 1 by finding the maximum and minimum values in the feature, subtracting the minimum value, and dividing by the maximum minus the minimum value.

Using standardization (*StandardScaler*) is another possibility. Unlike the min-max scaler, it does not bind values to a specific range. Values are scaled by subtracting the feature's mean value, making the scale feature's mean zero, and then dividing by the standard deviation; the resulting distribution has unit variance. This scaling method is

less affected by outliers than the min-max scaler, but can still be significantly affected [18].

Unlike MinMaxScaler and StandardScaler, robust scaler (*RobustScaler*) is based on percentiles, so it scales data according to the quartile range, the default is Interquartile Range, which is between 25^{th} and 75^{th} quartile [44]. It is not affected by a small number of large marginal outliers. The transformed feature, when using RobustScaler, has a larger range than for the previous scalers. The outliers are still present in the transformed data [9].

We will use StandardScaler to scale the feature values. It is very important to fit the scalers only to the training set and not the whole dataset. After fitting the scaler, it can then be used to transform the testing set [18].

4.3 Sequential feature selection (SFS)

Sequential feature selection is a greedy search algorithm used to reduce the dimension of feature space from the original n-dimensional to a d-dimensional feature subset where $d<n$. The main goal of a feature selection algorithm is to keep only features that are most relevant to describing how the dependent variable changes in time. By using SFS, we want to improve computational efficiency as well as improve the model's generalization property by removing irrelevant features [47].

We use the function *SequentialFeatureSelector()* from Python library *sklearn*. In this function, we can choose different values for the parameter *direction*, forward, and backward.

If we choose the direction forward, the feature selection is called Sequential Forward Selection (SFS). The input $Y = \{y_1, y_2, ..., y_n\}$ is the original n-dimensional feature set and the output $X_d = \{x_j | j = 1, ..., d; x_j \in Y\}$ is the d-dimensional feature subset, where $d = (0, ..., n)$ is defined in advance and $d < n$. In the initialization phase, we start with an empty set $X_0 = \emptyset$, so the $d = 0$, d is the size of the subset. In the next step, we add features, $x^+ = argmax J(X_d + x)$, where $x \in Y - X_d$, associated with the best classifier performance if added to the subset X_d, one by one to the feature subset X_d until the subset has d elements. The algorithm of adding a new feature to subset X_d can be written as $X_{d+1} = X_d + x^+$ and then $d = d + 1$ [47]. This is the method we will be using.

Another possibility is to choose the direction backward, called Sequential Backward Selection (SBS). The input and output are the same as in the Sequential Forward Selection. The difference is in the starting set in the initialization phase of the algorithm. We start with a full set $X_0 = Y$ so that $d = n$. In the next step, we remove features, $x^- = argmax J(X_d - x)$, where $x \in X_d$, associated with the best classifier performance if removed from the subset X_d, one by one until it has d elements. The algorithm of removing features from subset X_d can be written as $X_{d-1} = X_d - x^-$ and then $d = d - 1$ [47].

4.4 Finding the best k

Finding the best value of hyperparameter k is crucial to tuning the model's performance, which is why we have tools to automate the process.

There are multiple ways of selecting the best k. One of them is to use Grid Search. The package *scikit-learn* has a function *GridSearchCV()* which has the default number of neighbors five. Grid search helps to see whether there is a better number that would provide better model accuracy [30].

Grid search is an automated process of manually changing hyperparameters until you find a good combination, which can take a long time and is inefficient. The input value to GridSearchCV() is a range of values that the GridSearchCV() will try and which hyperparameters to experiment with. It works by applying cross-validation to evaluate all the possible combinations of hyperparameter values. We used a custom rolling window cross-validation generator, where we define the size of the initial train set, that drops old data points and includes new ones as the window moves forward, $window_size = 500$, size of the $test_size = 100$, and the number of times the rolling window process is repeated $n_splits = 10$. It is crucial to preserve the order of data points since we work with time series.

Generally, selecting a small k makes the model prone to overfitting since the average of the target is done on a small sample, so it is not robust. Greater k, on the other hand, smooths the estimates among neighbors [43].

4.5 Models

In this part of the thesis, we will tune the hyperparameters and find the one with the best model accuracy. There are a few hyperparameters we selected. We already covered most hyperparameters in other parts of Section 4. The last one is the number of features we let SFS select.

The first five features selected by SFS are Is_year_start, Ema_5, Ema_63, Lag_1, Lag_2.

d	MAE	RMSE	MAPE	R^2	k
5	3.7688	5.1395	0.87%	0.9502	100
5	3.4571	4.6272	0.80%	0.9597	110
5	3.4522	4.2792	0.80%	0.9655	115
5	3.5017	4.2563	0.81%	0.9659	116
5	3.4915	4.2483	0.81%	0.9660	117
5	3.6597	4.3826	0.85%	0.9638	120
8	5.3738	7.8896	1.22%	0.8827	100
8	4.2539	6.3563	0.97%	0.9239	110
8	3.9875	5.8152	0.91%	0.9363	120
8	3.9781	5.8006	0.90%	0.9366	121
8	3.8902	5.6631	0.89%	0.9396	122
8	4.1562	5.9367	0.95%	0.9336	130
10	3.6073	5.2089	0.82%	0.9489	100
10	3.3824	4.4014	0.78%	0.9635	110
10	3.3492	4.2892	0.77%	0.9653	120
10	3.2924	4.2756	0.76%	0.9656	125
10	3.2648	4.1849	0.75%	0.9670	126
10	3.3451	4.2034	0.78%	0.9667	130
12	4.5327	5.2952	1.05%	0.9472	100
12	4.1362	4.9596	0.96%	0.9537	110
12	4.1606	4.9281	0.97%	0.9542	120

12	4.1590	4.9210	0.97%	0.9544	130
12	4.1305	4.9385	0.97%	0.9540	140
12	3.7407	4.5853	0.88%	0.9604	150
12	3.7159	4.5818	0.87%	0.9604	151
15	14.3376	17.5308	3.25%	0.4209	200
15	14.0125	17.0979	3.18%	0.4491	208
15	13.9814	17.0889	3.17%	0.4497	210
15	14.1395	17.2481	3.21%	0.4394	212
15	14.3101	17.3989	3.25%	0.4296	215
15	14.6430	17.7259	3.33%	0.4079	220

Table 10: Evaluation metrics and number of neighbors for different values of d

Here, d is the number of features selected and k is the number of neighbors.

d	Selected Features
5	Ema_21, Ema_63, Year, Lag_2, Is_year_start
8	Ema_21, Ema_63, Is_month_start, Year, Is_quarter_start, Is_year_end, Lag_2, Is_year_start
10	Is_year_start, Lag_2, Is_year_end, Year, Is_month_start, Is_quarter_start, Ema_63, Is_month_end, Ema_21, Is_quarter_end
12	Ema_21, July_Nov, Ema_63, Is_month_start, Year, Is_quarter_end, Is_quarter_start, Is_year_end, Lag_2, Is_year_start, September, Is_month_end
15	Ema_21, July_Nov, Ema_63, Is_month_start, Year, Volume, Is_quarter_end, Is_quarter_start, Is_year_end, Lag_2, Is_year_start, Mon_Fri, September, Aug_Dec, Is_month_end

Table 11: Selected features for different values of d

Based on the evaluation metrics the best model out of all the ones we tried is a model with 10 features and 126 neighbors. This model has the lowest MAE, RMSE, and MAPE,

and the highest R^2. Let us plot the prediction.



a) SPY ETF actual prices and predicted prices



b) Actual and predicted prices from 1.1.2023 to 1.1.2024

Figure 26: Prediction of SPY ETF prices using KNN model with 10 features selected by SFS and 126 neighbors

Figure 26 illustrates the predictive performance of the KNN model with a lag depth of d=10 and k=126 nearest neighbors. The predicted values (orange) closely follow the actual prices in the test set (blue), successfully capturing both the general upward trend and local fluctuations. The model achieves high accuracy across multiple evaluation metrics: a Mean Absolute Error (MAE) of 3.2648 and a Root Mean Squared Error (RMSE)

of 4.1849, indicating a low level of deviation from actual values. The Mean Absolute Percentage Error (MAPE) is particularly low at 0.75%, demonstrating strong relative accuracy. Furthermore, the model achieves an R^2 of 0.9670, indicating that nearly 97% of the variance in the test data is explained. The features selected by SFS include: Is_year_start, Lag_2, Is_year_end, Year, Is_month_start, Is_quarter_start, Ema_63, Is_month_end, Ema_21, and Is_quarter_end.

5 Long short-term memory model

Recurrent Neural Networks (RNNs) are a class of neural networks designed to process sequential data by maintaining a hidden state that captures information from previous time steps. Unlike feedforward networks, RNNs use recurrent connections, allowing the model to retain memory of past inputs, making them suitable for tasks like language modeling and time series prediction. However, standard RNN networks struggle with learning long-term dependencies due to the problem of vanishing and exploding gradient during training. This occurs when gradients are repeatedly multiplied over many time steps, causing them to shrink toward zero or grow uncontrollably, thereby limiting the network's ability to learn from distant inputs [33].

These limitations led to the development of more advanced recurrent architectures, the most commonly used being the Long Short-Term Memory (LSTM) network. Even though LSTM does not guarantee the complete elimination of vanishing or exploding gradients, it is still an improvement over traditional RNNs, as it facilitates the learning of long-distance dependencies [52]. LSTM networks are a type of deep learning architecture designed specifically to handle sequential data and overcome the shortcomings of traditional RNNs. Introduced by Hochreiter and Schmidhuber in 1997 ([22]), LSTM models address the vanishing and exploding gradient problems, which hinder the ability of RNNs to learn long-range temporal dependencies [57]. The name "Long Short-Term Memory" reflects the model's ability to retain information over both short and long durations through its gated memory cell mechanism. The LSTM architecture is based on a recurrent neural network but replaces the standard hidden units with memory cells. Each memory cell contains a node with a self-connected recurrent edge of weight 1, which ensures stable gradient flow across time. The flow of information is further controlled by three gates: input, forget, and output gates, which selectively update and preserve the internal cell state [33].

LSTMs have many applications, some of them being in fields such as natural language processing, handwriting and speech recognition, computer vision, and particularly, time series forecasting [57]. Their ability to model nonlinear and non-stationary sequences makes them especially suitable for forecasting financial markets, where data often exhibit complex temporal dependencies, where LSTM models have shown promising results. For

instance, in the study conducted by Fischer and Krauss [15], an LSTM-based model outperformed traditional approaches such as random forests, logistic regression, and standard deep neural networks in predicting stock price movements [57].

More broadly, the use of LSTM in predictive modeling aligns with the aim of machine learning, which aims to develop algorithms that automatically learn patterns from data and improve performance through experience [51].

5.1 How does the LSTM work

The LSTM cell is defined as in [33] by the following equations:

$$\text{Candidate Cell State: } g^{(t)} = \phi(W_{gx}x^{(t)} + W_{gh}h^{(t-1)} + b_g) \tag{25}$$

$$\text{Input Gate: } i^{(t)} = \sigma(W_{ix}x^{(t)} + W_{ih}h^{(t-1)} + b_i) \tag{26}$$

$$\text{Forget Gate: } f^{(t)} = \sigma(W_{fx}x^{(t)} + W_{fh}h^{(t-1)} + b_f) \tag{27}$$

$$\text{Output Gate: } o^{(t)} = \sigma(W_{ox}x^{(t)} + W_{oh}h^{(t-1)} + b_o) \tag{28}$$

$$\text{Cell State Update: } s^{(t)} = g^{(t)} \odot i^{(t)} + s^{(t-1)} \odot f^{(t)} \tag{29}$$

$$\text{Hidden State Update: } h^{(t)} = \phi(s^{(t)}) \odot o^{(t)}, \tag{30}$$

where, as noted in [15], the following notations apply:

- h is the number of hidden units (i.e., the size of the hidden state $h^{(t)}$).
- d is the number of input features (i.e., the dimensionality of $x^{(t)}$).
- $x^{(t)} \in \mathbb{R}^d$ denotes the input at time step t.
- $h^{(t-1)} \in (-1, 1)^h$ denotes the hidden state from the previous time step; the initial value $h^{(0)}$ is set to 0.
- $s^{(t-1)} \in (-1, 1)^h$ denotes the previous cell state; the initial value $s^{(0)}$ is set to 0.
- $W_{gx}, W_{ix}, W_{fx}, W_{ox} \in \mathbb{R}^{h \times d}$ are weight matrices for the input.
- $W_{gh}, W_{ih}, W_{fh}, W_{oh} \in \mathbb{R}^{h \times h}$ are recurrent weight matrices.
- $b_g, b_i, b_f, b_o \in \mathbb{R}^h$ are bias terms that allow each unit to learn an offset.
- ϕ denotes the tanh activation function.

- σ denotes the sigmoid activation function.
- $\odot$ denotes element-wise multiplication.

Weights $W_{gx}, W_{ix}, W_{fx}, W_{ox}$ determine how current input $x^{(t)}$ affects the gates. Weights $W_{gh}, W_{ih}, W_{fh}, W_{oh}$ determine how previous hidden state $h^{(t-1)}$ influences the gates at time step t.

The initial values of weights in a neural network significantly impact how quickly the model converges and which local optimum it reaches. While the network learns optimal weights during training, the starting point can influence training efficiency. Neural networks are trained using stochastic gradient descent, which adjusts weights to minimize a loss function, ideally leading to useful predictions. Weight initialization sets the starting values of a network's weights before training begins, typically using small random values. In Long Short-Term Memory (LSTM) networks, weights are often initialized using the Xavier initialization method. This method generates random weights from a uniform distribution

$$weight = U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right], \tag{31}$$

where n is the number of inputs to a node. Xavier initialization is especially effective for networks using sigmoid or tanh activations, as it prevents issues like vanishing or exploding gradients [58].

On the other hand, biases in the Dense layer are, by default, initialized to zero, while in the LSTM layer, certain biases, such as those in the forget gate, are commonly initialized to one as noted in [28]. It is also possible to define custom initializers for both weights and biases [48].

Now that we are familiar with the notation in the equations, we can examine the role each one plays, as discussed in [57], [15], and [56].

The Equation (25) defines a new candidate value, generated from the current input $x^{(t)}$, to update the cell state. This updates potential information that might be added to the memory cell. We use the activation function tanh (hyperbolic tangent), which has a formula as in [33] as follows:

$$\phi(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{32}$$

The output range is [-1,1], making this approach more balanced since it centers the data around zero. This activation function, unlike the Sigmoid function, maintains a steeper gradient in its center, which makes it easier for a neural network to learn in backpropagation [37]. Activation functions transform raw values into outputs that can be worked with.

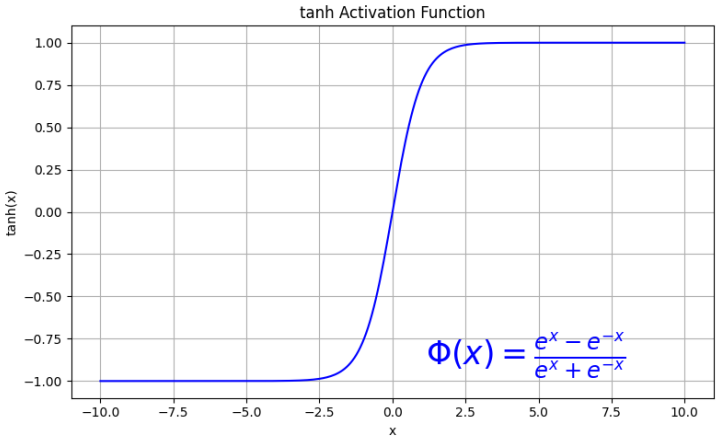


Figure 27: tanh Activation function

In Equation (26), the input gate acts as a filter, scaling $g^{(t)}$, with a Sigmoid function that is another activation function that has the formula as in [33] as follows:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{33}$$

It has a range of [0,1] and is used to determine how much of the signals go from the network to the memory cell. If the gate is closed, the activation is close to zero [51].

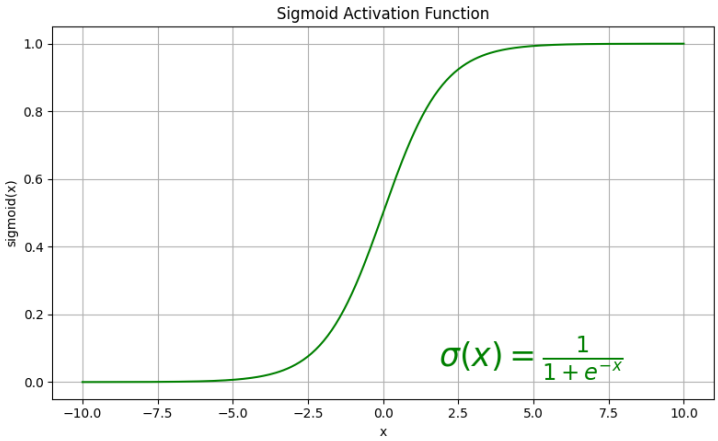


Figure 28: Sigmoid Activation Function

The activation function ensures that only part of the new information enters the short-term memory.

One can experiment with other activation functions, like ReLU, Swish, Hard Sigmoid, and others.

Forget gate in Equation (27) decides how much of the old memory $s^{(t-1)}$ should be retained. If $f^(t)$ is close to 0, it means most old memory will be forgotten, and if it is close to 1, most old memory will be retained.

The output gate in Equation (28) decides what information to pass to the next step using the sigmoid activation function. In other words, it controls how much of the long-term memory $s^{(t-1)}$ is exposed as short-term memory $h^{(t-1)}$.

In Equation (29), the cell state update, that is the long-term memory, we use element-wise multiplication to calculate two parts of the equation first how much of the new information will be added to long-term memory, and second, what past information should be retained in long-term memory. The cell state then updates the blend of new and old information in long-term memory.

The last Equation (30), a hidden state update, that is the short-term memory, is controlled by the output gate. Only part of the long-term memory will be saved in short-term memory. It is a filtered version of long-term memory $s^{(t-1)}$.

While mathematical Equations (25) - (30) provide a theoretical background for understanding the architecture of the LSTM memory cell, visual representation may further clarify how information is processed and updated over time. The following figures illustrate the workings of a memory cell by showing every gate and how they are used to update the cell memory.

Figure 29, inspired by [32], depicts the LSTM memory cell at time t, where cell state $s^{(t)}$, new updated memory, and hidden state, current output, $h^{(t)}$ are updated based on the previous time step $t-1$.

Figure 30, from [56], shows the relationship between the inputs ($x^{(t)}$, $s^{(t-1)}$, and $h^{(t-1)}$) and outputs ($s^{(t)}$ and $h^{(t)}$) in time step t, and how information are transferred from time step $t-1$ to t and from t to $t+1$.

The advantage of this algorithm is instead of storing everything, the LSTM multiplies each piece of information by a gate value to selectively allow only some data to

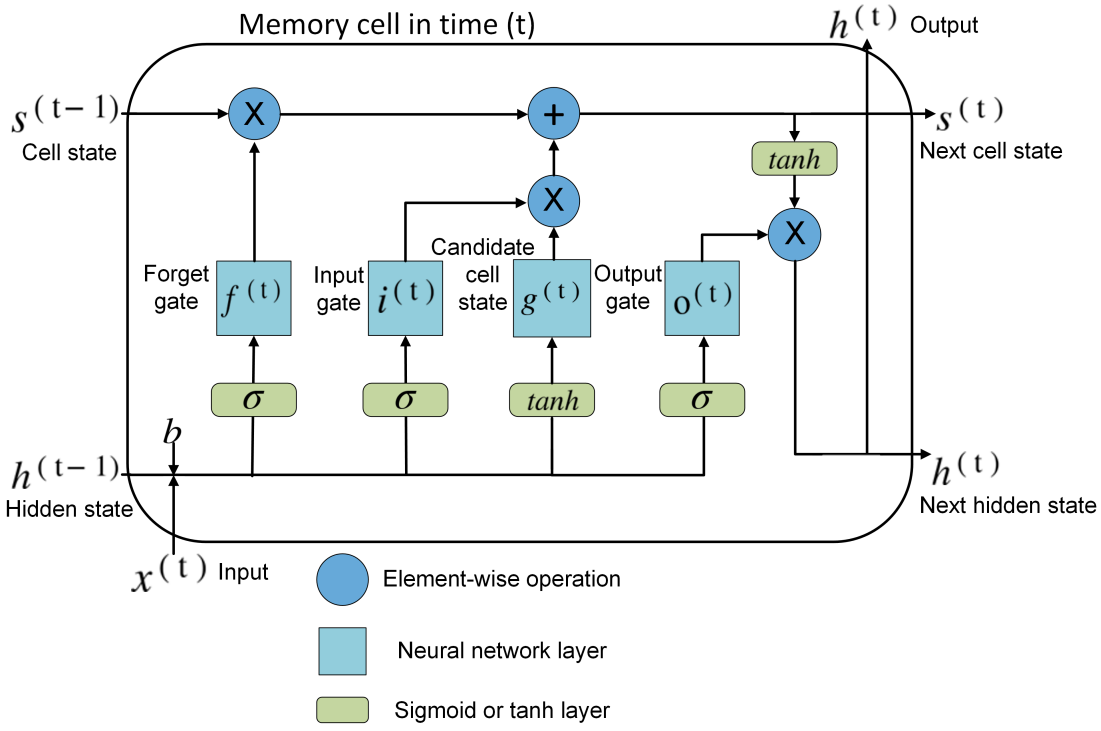


Figure 29: Structure of LSTM cell at time step t

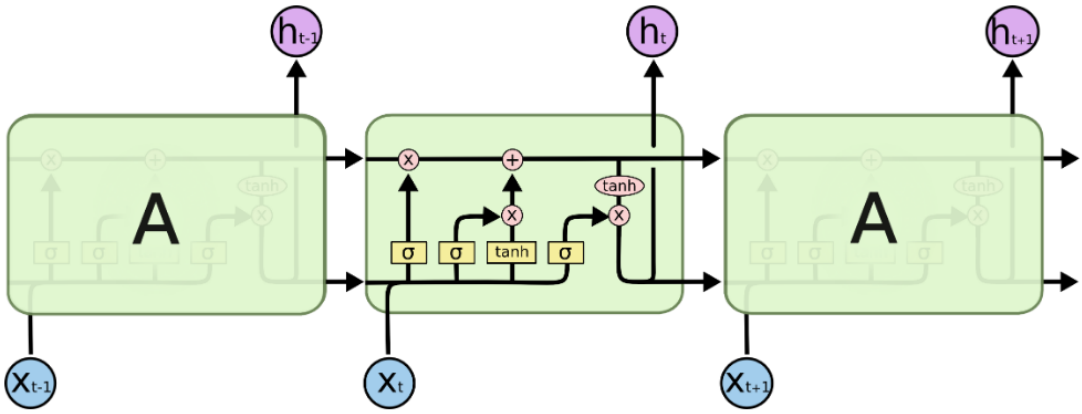


Figure 30: Structure os LSTM cell at time step t in relationship to time steps $t-1$ and $t+1$

enter the memory cell. This mechanism allows an LSTM model to remember long-term dependencies while avoiding excessive noise [2].

5.2 Data preparation

The LSTM does not assume stationarity, therefore, we do not have to transform the time series. However, we will predict the returns of the prices rather than the original prices. Returns are often more stable and easier to predict. This transformation will help improve the accuracy of the model forecast. We will also be using *StandardScaler()* to scale our target, the closing prices of SPY ETF. If we were to fit a network to unscaled data, there is a possibility that the large scale would affect the speed of learning and convergence, and may prevent the network from learning effectively [23].

In the training stage, we use the sliding window approach to enable the model to learn temporal dependencies in the time series. This approach transforms the data into a supervised learning format, where each input sample is a sequence with a fixed length of 350 consecutive trading days, and a corresponding output for each sample of 249 consecutive trading days, representing the prediction horizon. The model then performs a direct multi-step forecasting to learn patterns and improve the model's performance. In the rolling window approach, we can let x_t denote the closing price at time t, then each training sample is defined as $[x_{t-350}, x_{t-349}, ..., x_{t-1}]$ and the output sample is then defined as $[x_t, x_{t+1}, ..., x_{t+248}]$. All input sequences are shaped as (samples, timesteps, features), as the LSTM architecture requires.

5.3 LSTM in Python

To create an LSTM model, we will use the Python libraries TensorFlow and Keras. To define the model architecture, we will use the function Sequential() from *tensorflow.keras.models* that builds a neural network by linearly stacking layers. The architecture begins with creating an empty Sequential model, to which layers are progressively added with the function model.add(). The first layer consists of LSTM units, where we define the number of neurons (or hidden units) and set the argument return_sequences=True, which is crucial because it enables the LSTM layer to output sequences rather than a single value [12]. Then, we need to define the input_shape argument, which specifies the shape of the input data.

After each LSTM layer, a Dropout layer is included. It is a regularization technique that randomly sets a portion of the input units to zero during training, effectively ex-

cluding them from activation and weight updates [13]. The rate is in the interval $(0, 1)$ and indicates the percentage of nodes dropped during each training iteration. This helps prevent overfitting by reducing reliance on specific nodes. The last LSTM layer is set to return_sequences=False, signaling the end of sequence processing.

The last layer is a Dense(n) layer with n units, corresponding to the number of days ahead we aim to predict. This layer generates the final output predictions [16].

Now that we have created the architecture of the model, we will use the function model.compile() that specifies the training configuration where the arguments are optimizer, loss, and metrics. The default argument values in the function, as in [31], are:

$$compile(optimizer =' rmsprop', loss = None, loss_weights = None,$$
$$metrics = None, weighted_metrics = None, run_eagerly = False,$$
$$steps_per_execution = 1, jit_compile =' auto', auto_scale_loss = True)$$

There are many optimizers to choose from that have different algorithms, they are responsible for updating the model's weights to minimize the loss function. We will use the optimizer *Adam* (Adaptive Moment Estimation), an extension to stochastic gradient descent for training deep learning models. The optimizer has multiple hyperparameters, the one we will tune being *learning_rate* (or step size) with a default value 0.001 [53]. According to [17], optimizer *Adam* is currently recommended as the default algorithm to use. The *learning_rate* is a float value that determines how much the model's weights are updated during training. High learning weight can speed up training, but if it's too high, it might overshoot and not find the minimum. On the other hand, if we set the learning rate too low, it can be computationally ineffective, but it ensures more precise convergence [17].

The next parameter is *loss*, or the loss function, which is calculated at each iteration to compute gradients for updating model weights through backpropagation, continuing until updates no longer improve the target evaluation metric. The first loss function we will implement is Mean Squared Error, which computes the mean squared error between the true values and predictions [29]. The second loss function we will use is *Huberloss*, which combines the best properties of Mean Squared Error and Mean Absolute Error; the hybrid nature makes it less sensitive to outliers, similar to MSE, but also penalizes small errors, similar to MAE [55].

And lastly, metrics are used to evaluate deep learning models, but they don't affect the learning. You can list the names of the metrics you want the model to monitor while training the model. Some of the more often used metrics for regression are: Mean absolute error, Mean squared error, Root mean squared error, Mean absolute percentage error [25].

After the model is compiled with a specified optimizer, loss function, and evaluation metric, it is then trained on the input data using the function model.fit(). The default argument values, as in [31], are:

$$
\begin{aligned}
&fit(x = None, y = None, batch_size = None, epochs = 1, verbose =' auto',\\
&callbacks = None, validation_split = 0.0, validation_data = None,\\
&shuffle = True, class_weight = None, sample_weight = None,\\
&initial_epoch = 0, steps_per_epoch = None, validation_steps = None,\\
&validation_batch_size = None, validation_freq = 1)
\end{aligned}
$$

Parameters x and y are input features and target values for the model to learn from. The shape of x should be (samples, timesteps, features).

One of the parameters we will tune is *batch_size*, which determines the number of training samples the model processes before updating its internal weights [24]. A smaller batch size can help achieve better generalization due to the added noise in gradient updates, whereas a larger batch size can lead to faster training. Still, it may reduce stability and increase the risk of overfitting. Typical values of the batch size are 32, 64, and 128 according to [46]. The other significant parameter is *epochs*, which represents the number of iterations over the entire dataset the model will go through. The larger the value, the better the opportunity to learn, but the greater the risk of overfitting [59].

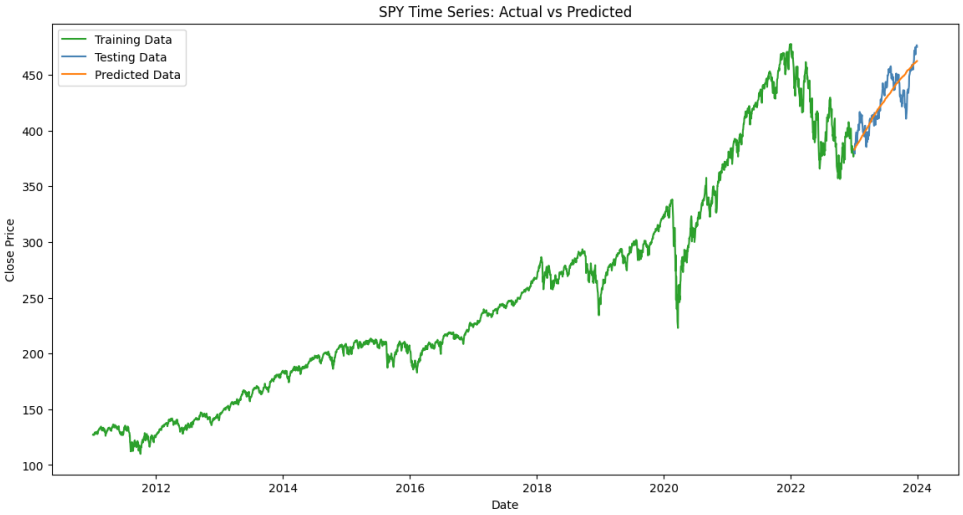
5.4 Models

We tested several hyperparameter combinations and summarized the most relevant results in a table to compare results and identify the optimal model.

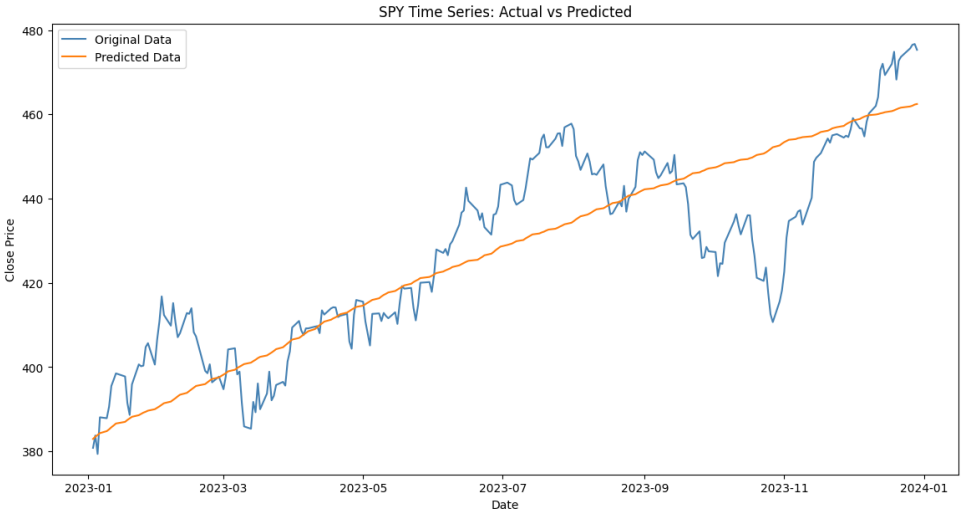
#Layers	Units	Dropout	Optimizer	Loss	Learning rate	Epochs	Batch	MAE	RMSE	MAPE	R^2
4	100	0.2	adam	mse	0.0001	30	32	12.9305	16.2868	3.08%	0.5002
4	100	0.2	adam	mse	0.00005	30	32	10.6588	12.8428	2.50%	0.6892
4	100	0.2	adam	mse	0.00001	50	32	13.2082	16.6751	3.13%	0.4761
4	100	0.2	adam	mse	0.00005	40	32	14.3456	17.7514	3.39%	0.4062
4	100	0.2	adam	mse	0.00005	30	64	10.9332	13.6484	2.58%	0.6490
4	100	0.1	adam	mse	0.0001	30	32	15.8802	19.5744	3.76%	0.2780
4	100	0.15	adam	mse	0.0001	30	32	11.0254	13.7114	2.60%	0.6457
4	100	0.15	adam	mse	0.00005	30	32	10.6441	12.8441	2.49%	0.6891
4	100	0.15	adam	mse	0.00005	40	32	15.1591	18.7955	3.59%	0.3343
4	100	0.2	adam	huber(δ=1.0)	0.0005	30	32	10.9024	13.7402	2.57%	0.6443
4	100	0.2	adam	huber(δ=1.0)	0.0005	40	32	10.0479	12.5489	2.33%	0.7033
4	100	0.2	adam	huber(δ=1.0)	0.0001	30	32	11.2753	14.6801	2.66%	0.5939
4	100	0.15	adam	huber(δ=1.0)	0.0005	30	32	27.7666	31.4188	6.49%	-0.8601
4	100	0.15	adam	huber(δ=1.0)	0.0001	30	32	31.0813	34.8014	7.25%	-1.2821
4	100	0.15	adam	huber(δ=1.0)	0.001	30	32	9.9714	12.9319	2.33%	0.6849
4	100	0.1	adam	huber(δ=1.0)	0.0005	30	32	11.3093	13.9737	2.63%	0.6321
4	100	0.1	adam	huber(δ=1.0)	0.001	30	32	11.4965	13.9211	2.65%	0.6348
4	100	0.1	adam	huber(δ=1.0)	0.0001	30	32	9.8561	12.3620	2.32%	0.7120

Table 12: Evaluation metrics for different LSTM configurations

Out of all these models we evaluated with evaluation metrics, the model with a dropout layer $Dropout(0.1)$, loss function $huber(\delta = 1.0)$, optimizer $Adam$ with learning rate 0.0001, 30 epochs, and with a batch size 32 performed the best.



a) SPY ETF actual prices and predicted prices



b) Actual and predicted prices from 1.1.2023 to 1.1.2024

Figure 33: Prediction of SPY ETF prices using LSTM model

Figure 33 presents the LSTM model's performance in forecasting the SPY ETF closing prices. The time series plot is divided into three parts: training set (in green), testing set (in blue), and the prediction (in orange). It is evident from the plot that the model captures the overall upward trend accurately. However, the short-term fluctuations were

highly underestimated. The model accurately depicts general price direction and could be used for planning long-term investments.

Several strategies could enhance the performance of the LSTM model. One approach is to explore a wider range of hyperparameter combinations, as numerous aspects of the model's architecture and training process can be optimized. Additionally, experimenting with alternative model structures, such as Bidirectional LSTMs, may improve the model's ability to capture temporal dependencies. Incorporating additional informative features, similar to the strategy applied in the KNN model, could also enrich the input space. Lastly, alternative sequence modeling methods, including expanding window or encoder-decoder architectures, may be more effective than the traditional sliding window approach.

6 Comparison of models

While all three models (ARIMA, KNN, and LSTM) were evaluated on the same prediction task, it is essential to recognize that the KNN model had access to more information through a broader feature set, including lagged values, calendar-based patterns, and technical indicators. In contrast, the ARIMA and LSTM models were trained on a univariate time series, using only past closing prices of the SPY ETF. This asymmetry complicates a direct comparison of predictive performance.

Model	**MAE**	**RMSE**	**MAPE**	**R^2**
ARIMA	25.7241	29.7269	5.87%	-0.6651
KNN	3.2648	4.1849	0.75%	0.9670
LSTM	9.8561	12.3620	2.32%	0.7120

Table 13: Comparison of the performance of all three models

Table 13 presents the evaluation metrics for all three forecasting models (ARIMA, KNN, and LSTM) based on their performance on the SPY ETF test set. The KNN model demonstrates the best overall performance across all metrics, achieving the lowest MAE (3.26), RMSE (4.18), and MAPE (0.75%), as well as the highest R^2 score (0.97). These results indicate a strong fit between predicted and actual values, with minimal deviation and high explanatory power.

The LSTM model performs moderately well, with an MAE of 9.8561 and an R^2 of 0.7120. This performance is reasonable, considering no additional features were used, only historical prices. It is also important to note that LSTM models were significantly more computationally demanding. In contrast, the ARIMA model returned the weakest results, with the highest MAE (25.72), RMSE (29.73), and MAPE (5.87%), as well as an R^2 of -0.6651, indicating a poor fit and the model's inability to explain the variability in the test data.

Although the KNN model outperformed the other two, it is crucial to note that this superiority may be attributed to its richer feature set, rather than its algorithmic characteristics alone. To ensure a fairer comparison, the ARIMA model could be extended to an ARIMAX framework, where we can include additional predictors, and the same feature

engineering applied to KNN could be incorporated into the LSTM model. Alternatively, the KNN model could be constrained to use only the target variable's historical values.

Conclusion

In this thesis, we analyzed, transformed, and predicted the values of a time series, specifically the prices of the SPY ETF. We first examined the dataset and provided the theoretical background for the evaluation metrics used to compare the selected models. The thesis aimed to identify the models with the highest prediction accuracy, supported by a theoretical foundation for each of the three implemented algorithms.

The first model was the Autoregressive Integrated Moving Average (ARIMA), which required transforming the time series to achieve stationarity. We adjusted the hyperparameters in the $auto_arima()$ function to develop four different models, which were then compared using all the evaluation metrics discussed in Section 2. The second model was K-Nearest Neighbors (KNN), which involved creating new features, addressing non-stationarity, and scaling all numerical variables. We applied sequential feature selection and developed several models for comparison. The final model was the Long Short-Term Memory (LSTM) network, a deep learning model. We scaled the data and employed the Sliding Window Method for Sequence-to-Many forecasting. Several models were developed using different hyperparameter values, including dropout rates, loss functions, learning rates, number of epochs, and batch sizes. These models were then evaluated, and the best-performing one was selected.

In the concluding section, we compared all three models to determine which achieved the highest predictive accuracy based on the chosen evaluation metrics. The best-performing model was KNN, with the lowest values for MAE (3.26), RMSE (4.18), and MAPE (0.75%), and the highest R^2 score (0.97), demonstrating its suitability for forecasting SPY ETF prices over the specified horizon. Despite these promising results, each model has limitations. ARIMA assumes linearity and stationarity, which may not hold in volatile financial markets. KNN depends heavily on the feature set and may struggle to capture long-term dependencies. LSTM requires substantial data and computational resources and is highly sensitive to hyperparameter tuning.

As mentioned earlier, we aimed to evaluate three different modeling approaches for time series forecasting, ARIMA, KNN, and LSTM, and identify the most accurate predictor of SPY ETF prices. Based on the analysis, this objective was achieved. However, as noted in Section 6, comparing these models is not entirely fair, as two of them (ARIMA

and LSTM) used only the target variable, closing prices, while KNN relied on a richer feature set that incorporated more information about market dynamics. For a more balanced comparison, one would need to either use KNN for univariate time series prediction or extend LSTM and ARIMA with the same features used in KNN, for instance by adopting the ARIMAX variant instead of classical ARIMA.

These findings are particularly relevant for retail investors and financial institutions seeking to enhance their short- to mid-term price forecasts using classical or machine learning models. Choosing the appropriate model is crucial for achieving accurate and interpretable forecasts. Although the results are specific to SPY ETF prices, the modeling approaches and evaluation framework presented here can be applied to similar financial time series, subject to appropriate feature engineering and domain-specific adjustments.

Future research could explore additional variations of the three models, alternative data preprocessing techniques, or hybrid modeling approaches. Incorporating new features and further tuning hyperparameters could also contribute to improved predictive performance.

In conclusion, this comparison underscores the importance of model architecture and input features in time series forecasting. The results highlight the need to carefully balance model complexity, interpretability, and data richness in financial prediction tasks. As data availability and computational power continue to grow, selecting the most suitable forecasting model will remain a technical and strategic challenge in financial analytics.

References

[1] Akaike Information Criterion, available on the Internet (12.1.2025): https://www.sciencedirect.com/topics/pharmacology-toxicology-and-pharmaceutical-science/akaike-information-criterion

[2] Al-Selwi, S. M., Hassan, M. F., Abdulkadir, S. J., et al.: RNN-LSTM: From applications to modeling techniques and beyond—Systematic review, Journal of King Saud University - Computer and Information Sciences 36 (2024), article number: 102068

[3] Applying k-nearest neighbors to time series forecasting: two new approaches, available on the Internet (12.1.2025): https://www.sciencedirect.com/science/article/pii/S0169207019301872

[4] Best Times of the Day, Week, and Month to Trade Stocks, available on the Internet (12.1.2025): https://www.investopedia.com/day-trading/best-time-day-week-month-trade-stocks/

[5] Box, G. E. P., Jenkins, G. M., Reinsel, G. C., Ljung, G. M.: Time Series Analysis: Forecasting and Control, Wiley, Nemecko, 2015

[6] Brockwell, P. J., Davis, R. A.: Introduction to Time Series and Forecasting, Springer, New York, 2002

[7] Brownlee, J.: Deep Learning for Time Series Forecasting: Predict the Future with MLPs, CNNs and LSTMs in Python, Machine Learning Mastery, 2018

[8] Chung, M. K.: Brain Network Analysis, Cambridge University Press, 2019

[9] Compare the effect of different scalers on data with outliers, available on the Internet (20.2.2025): https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html#plot-all-scaling-robust-scaler-section

[10] Cyclic Encoding: Sine and Cosine Transformations for Periodic Features, available on the Internet (25.2.2025): https://www.kaggle.com/discussions/general/491296

[11] Data Science: Concepts and Practice, available on the Internet (12.1.2025): https://www.elsevier.com/books/data-science/kotu/978-0-12-814761-0

[12] Difference Between Return Sequences and Return States for LSTMs in Keras, available on the Internet (23.4.2025): https://machinelearningmastery.com/return-sequences-and-return-states-for-lstms-in-keras/

[13] Dropout with LSTM Networks for Time Series Forecasting, available on the Internet (23.4.2025): https://machinelearningmastery.com/use-dropout-lstm-networks-time-series-forecasting/

[14] Explaining negative R-squared, available on the Internet (1.3.2025): https://towardsdatascience.com/explaining-negative-r-squared-17894ca26321/

[15] Fischer, T., Krauss, C.: Deep learning with long short-term memory networks for financial market predictions, European Journal of Operational Research 270 (2018), 654-669

[16] The 5 Step Life-Cycle for Long Short-Term Memory Models in Keras, available on the Internet (23.4.2025): https://machinelearningmastery.com/5-step-life-cycle-long-short-term-memory-models-keras/

[17] Gentle Introduction to the Adam Optimization Algorithm for Deep Learning, available on the Internet (23.4.2025): https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[18] Géron, A.: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, O'Reilly Media, 2019

[19] A Guide to Bayesian Information Criterion for Model Selection, available on the Internet (19.4.2025): https://www.numberanalytics.com/blog/bayesian-information-criterion-guide

[20] Handling Nominal Categorical Data in Machine Learning: A Comprehensive Guide, available on the Internet (23.4.2025): https://medium.com/%40lomashbhuva/handling-nominal-categorical-data-in-machine-learning-a-comprehensive-guide-6a01b8ed4875

[21] Hilpisch, Y. J.: Python for Finance: Mastering Data-driven Finance, O'Reilly Media, 2019

[22] Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory, Neural Computation 9 (1997), 1735-1780

[23] How to Scale Data for Long Short-Term Memory Networks in Python, available on the Internet (12.4.2025): https://machinelearningmastery.com/how-to-scale

-data-for-long-short-term-memory-networks-in-python/

[24] How to use Different Batch Sizes when Training and Predicting with LSTMs, available on the Internet (23.4.2025): https://machinelearningmastery.com/use-different-batch-sizes-training-predicting-python-keras/

[25] How to Use Metrics for Deep Learning with Keras in Python, available on the Internet (24.4.2025): https://machinelearningmastery.com/custom-metrics-deep-learning-keras-python/

[26] Jansen, S.: Machine Learning for Algorithmic Trading: Design and implement investment strategies based on smart algorithms that learn from data using Python, Packt Publishing, New York, 2018

[27] Jansen, S.: Machine Learning for Algorithmic Trading: Predictive models to extract signals from market and alternative data for systematic trading strategies with Python, Packt Publishing, New York, 2020

[28] Jozefowicz, R., Zaremba, W., Sutskever, I.: An Empirical Exploration of Recurrent Network Architectures, Proceedings of the 32nd International Conference on Machine Learning 37 (2015), 2342-2350

[29] Keras Loss Functions: Everything You Need to Know, available on the Internet (17.4.2025): https://neptune.ai/blog/keras-loss-functions

[30] Korstanje, J.: Advanced Forecasting with Python: With State-of-the-Art-Models Including LSTMs, Facebook's Prophet, and Amazon's DeepAR, Apress, 2021

[31] Li, F., Krishna, R., Xu, D.: Lecture 10: Recurrent Neural Networks, lecture slides, Stanford University, California, 2021, available on the Internet (22.4.2025): https://cs231n.stanford.edu/slides/2021/lecture_10.pdf

[32] Li, S., Chen, J., Liu, B.: Protein remote homology detection based on bidirectional long short-term memory, BMC Bioinformatics 18 (2017)

[33] Lipton, Z. C., Berkowitz, J., Elkan, C.: A Critical Review of Recurrent Neural Networks for Sequence Learning, arXiv, 2015

[34] Lopez de Prado, M.: Advances in financial machine learning, John Wiley Sons, New York, 2024

[35] Lopez de Prado, M.: Machine Learning for Asset Managers, Cambridge University Press, New York, 2020

[36] Martínez, F., Frías, M. P., Pérez, M. D., Rivera, A. J.: A methodology for applying k-nearest neighbor to time series forecasting, Artificial Intelligence Review 52 (2019), 2019-2037

[37] Mastering Tanh: A Deep Dive into Balanced Activation for Machine Learning, available on the Internet (28.4.2025): https://medium.com/ai-enthusiast/mastering-tanh-a-deep-dive-into-balanced-activation-for-machine-learning-4734ec147dd9

[38] Mastering the Augmented Dickey-Fuller Test for Time Series Analysis, available on the Internet (24.4.2025): https://www.numberanalytics.com/blog/augmented-dickey-fuller-test-time-series

[39] Müller, A. C., Guido, S.: Introduction to Machine Learning with Python: A Guide for Data Scientists, O'Reilly Media, 2018

[40] OneHotEncoder, available on the Internet (25.2.2025): https://feature-engine.trainindata.com/en/1.8.x/user_guide/encoding/OneHotEncoder.html#

[41] pmdarima.arima.auto_arima, available on the Internet (24.4.2025): https://alkaline-ml.com/pmdarima/modules/generated/pmdarima.arima.auto_arima.html

[42] Probabilistic Model Selection with AIC, BIC, and MDL, available on the Internet (12.1.2025): https://machinelearningmastery.com/probabilistic-model-selection-measures/

[43] Provost, F., Fawcett, T.: Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking, O'Reilly Media, 2013

[44] RobustScaler, available on the Internet (20.2.2025): https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html

[45] r2_score, available on the Internet (26.4.2025): https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html

[46] Selecting Optimal LSTM Batch Size, available on the Internet (23.4.2025): https://medium.com/@canerkilinc/selecting-optimal-lstm-batch-size-63066d88b96b

[47] SequentialFeatureSelector: The popular forward and backward feature selection approaches (including floating variants), available on the Internet (31.1.2025):

https://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/

[48] Setting up the data and the model, available on the Internet (22.4.2025): https://cs231n.github.io/neural-networks-2/

[49] Sørensen, B. E., Short Introduction to Time Series, notes, Department of Economics University of Houston, Huston TX, 2022, available on the Internet (18.1.2025): https://uh.edu/~bsorense/arma-intro2022.pdf

[50] The SPY ETF: Everything You Need to Know, available on the Internet (12.1.2025): https://www.etf.com/sections/etf-basics/spy-etf-everything-you-need-know?social_auth=googleAuth

[51] Staudemeyer, R., Morris, E.: Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks, arXiv, 2019

[52] tf.keras.Model, available on the Internet (22.4.2025): https://www.tensorflow.org/api_docs/python/tf/keras/Model

[53] tf.keras.optimizers.Adam, available on the Internet (16.4.2025): https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam

[54] Tips to using auto_arima, available on the Internet (24.4.2025): https://alkaline-ml.com/pmdarima/tips_and_tricks.html

[55] Understanding Huber Loss function: Insights from Applications, available on the Internet (17.4.2025): https://medium.com/@devcharlie2698619/understandin

g-huber-loss-function-insights-from-applications-5c1c5145d2c4

[56] Understanding LSTM Networks, available on the Internet (22.4.2025): https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[57] Van Houdt, G., Mosquera, C., Nápoles, G.: A review on the long short-term memory model, Artificial Intelligence Review 53 (2020), 5929-5955

[58] Weight Initialization for Deep Learning Neural Networks, available on the Internet (22.4.2025): https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/

[59] What is Epoch in Machine Learning?, available on the Internet (23.4.2025): https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-is-epoch-in-machine-learning

[60] What Is the GARCH Process? How It's Used in Different Forms, available on the Internet (1.5.2025): https://www.investopedia.com/terms/g/generalalizedautogregressiveconditionalheteroskedasticity.asp

[61] Yan, X., Su, X.: Linear Regression Analysis: Theory and Computing, Singapur: World Scientific, 2009