University of Hertfordshire UH

School of Physics, Engineering and Computer Science

MSc Data Science Project

7PAM2002-0509-2023

Department of Physics, Astronomy and Mathematics

Data Science FINAL PROJECT REPORT

Project Title:

Comparative Study of Stock Market Forecasting Using Time Series Models.

Student Name and SRN:

Raghavendhra Rao Devineni and 21072747

Supervisor: Vandana Das

Date Submitted: 29-08-2024

Word Count: 7858

DECLARATION STATEMENT

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Data Science at the University of Hertfordshire.

I have read the guidance to students on academic integrity, misconduct and plagiarism information at Assessment Offences and Academic Misconduct and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project module or course.

I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6). I have not used chatGPT, or any other generative AI tool, to write the report or code (other than where declared or referenced).

I did not use human participants or undertake a survey in my MSc Project.

I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

Student Name printed: Raghavendhra Rao Devineni

Student Name signature: 

Student SRN number: 21072747

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

Table of Contents

Contents

Table of Figures

Abstract:

Making predictions about the stock market is challenging since many factors influence how the market behaves. In this project, we will use the Istanbul Stock Exchange (ISE) using the data from the UCI machine learning repository. The goal of this project is to compare how well Autoregressive Integrated Moving Average (ARIMA), Long Short-Term Memory (LSTM), and Prophet models forecast the ISE index. Finding the model with the highest accuracy while utilizing the least amount of data input is the project's main aim. By creating the model architecture, assessing the model performance, and figuring out the best model for short-term stock prediction, this study will greatly enhance the understanding of the stock exchange trends and the effectiveness of models in financial forecasting.

Introduction:

Background

Forecasting the stock market's future using past data and other influencing factors is referred to as stock market prediction. There are two popular approaches to stock price forecasting. First and foremost is "fundamental analysis", which can be divided into top-down and bottom-up approaches. Bottom-up analysis uses "price/earnings (P/E)" ratios and other metrics to analyze a company's performance whereas top-down analysis starts from the state of the economy and predicts how the economy will affect the stock. Second is "technical analysis" that assumes there are patterns in charts that can predict the future movement of prices. A few investors utilize both approaches: technical analysis indicates the trend's entry point, while fundamental analysis offers them multiple points of view on a stock (Figure 1).

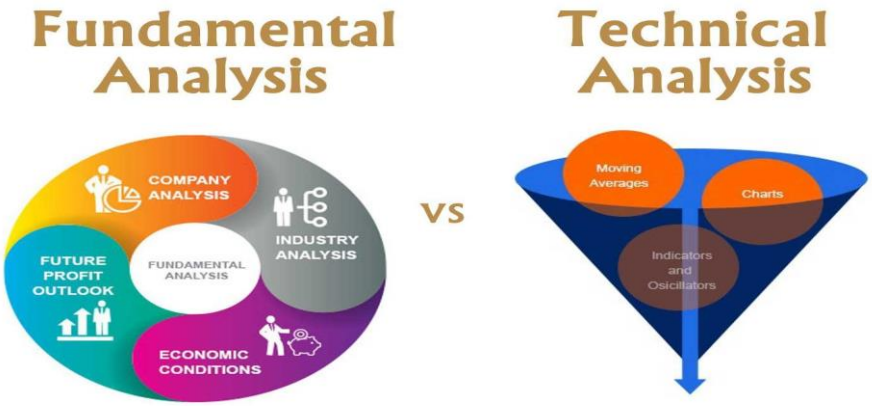


Figure 1: Fundamental Analysis and Technical Analysis Comparison (Source: Analytics Vidhya, 2021)

The ability to make better decisions and control risk is one of the main advantages that offers investors, financial analysts, and traders to reduce risk and make smarter judgments. Because of several factors that may impact market behavior in addition to the regular fluctuations in prices, predicting stock market values is a difficult and complex undertaking. One important financial market that represents Turkey's economic performance and investment opportunities is the Istanbul Stock Exchange (ISE).

Both domestic and foreign investors can benefit from understanding and forecasting the ISE attribute. In this study, I made use of the UCI machine learning repository's dataset "Istanbul Stock Exchange (ISE)". With so many variables involved in predicting stock values, it is difficult to make predictions of values with great accuracy, which is where machine learning comes in. The applications of machine learning and statistical models in financial forecasting have been developed to improve the accuracy and confidence of investing predictions.

Features include the value of the Istanbul Stock Exchange (ISE) index, the Standard & Poor's 500 index (SP), which tracks the performance of 500 large US companies, the German stock market index for 30 major companies (DAX), the London Stock Exchange's 100 largest companies (FTSE), the Tokyo Stock Exchange's 225 large companies (NIKKEI), and the Brazilian stock market index (BOVESPA). EU, an index that monitors stock performance among EU member nations. Emerging Markets Index (EM), which represents stocks in

developing nations. Collectively, these characteristics offer a thorough understanding of the regional (ISE) and worldwide market environments, assisting in the development of a strong predictive and full understanding of the connections between the targeted variable and other financial metrics.

Machine learning research is used to establish how much historical data the model must study to anticipate the stock market. Each pricing attribute is given a weight by the machine learning model. Recurrent neural networks (RNNs) and long short-term memory (LSTMs) are the common machine learning models used to predict time series data, including stock prices, weather forecasts, and housing prices. Assessing how important each factor is of recent versus historical data helps to identify the factors that have the greatest influence on present- or next-day prices.

Problem Statement

This study's main goal is to use historical financial data to anticipate the Istanbul Stock Exchange (ISE) with accuracy. Choosing the right model for prediction is important because of the complexity of financial markets. Even though they are widely used, traditional time series models like ARIMA might not be able to properly represent complex dependencies on time and non-linear relationships. On the other hand, advanced deep learning models such as the LSTM network have shown progress in processing sequential data; however, they require sufficient training data and appropriate tuning. The objective of this research is to evaluate and compare the ISE indices forecasting performance of the LSTM, ARIMA and Prophet models to identify which model requires less data to provide more accurate forecasts.

Justification of the study

This study is justified by the potential benefits of improving investment forecasting accuracy. Accurate models may enhance financial decision-making, investment strategies, and risk minimization. By evaluating the performance of ARIMA, LSTM, and Prophet models, and identifying the benefits and drawbacks of each model, this project contributes to the ongoing research in financial projections. The findings will be useful to investors, financial analysts, and researchers who are interested in applying machine learning and statistical models in the financial market. And in cases where data is very limited, understanding which model performs best with less data might be extremely beneficial.

Research Questions

The research aims to explain the following questions:

1. When predicting the stock market with limited amounts of data input, which model LSTM or ARIMA provides the highest accuracy?
2. By studying market movements, daily returns, moving averages, and correlations between stocks, how can we forecast future stock behavior?

Aims and Objectives

This research aims to find out whether a model like LSTM or ARIMA predicts the Istanbul Stock Exchange (ISE) index more accurately. The objective is to:

1. Explain how well the ARIMA and LSTM model predicts the ISE index.
2. Determine which model works best with minimum data input.
3. Describes some efficient financial forecasting techniques.

Literature Review

Qian et al. (2022), in their case study titled "Stock Predicting based on ARIMA, and LSTM" applied the LSTM and ARIMA models, where 95% of the total data for each stock was used as training data to train the model parameters, and the remaining 5% was used as test data. The data utilized included Google Stock Prices from April 2017 to April 2022, sourced from Yahoo Finance. The model's predicted outcomes on the test dataset were used to validate its benefits and drawbacks. To avoid overfitting, several dropout layers and six LSTM layers were incorporated into the LSTM model architecture. The output's dimensions were set to 64 units, and the return sequences parameter was used to represent stock. Using LSTM and ARIMA, predictions were made to determine whether to return the entire series or only the last output in the sequence, with input_shape serving as the shape of the training set. A dense layer with an output of one unit was added, and a drain layer with an output value of 0.2 was removed. The model operated with a batch size of 32 and was used for 100 iterations. However, the study did not train the RNN on the entire observation sequence; instead, it used a batch of small subsequences randomly selected from the training data. The LSTM model performs substantially better than the ARIMA model, which has much higher value MAE:154.5234, MSE:38,849.0341, RMSE:197.1015. The LSTM model has an MAE of 9.5921, MSE of 156.9601, and RMSE of 12.5283. This illustrates how much more accurate and dependable LSTM is at projecting future market prices, which makes it a more useful tool for investors.

Ma (2020) examined three models in their research, analyzing the underlying assumptions and forecast outcomes of each model. The study concluded that while the LSTM model is heavily influenced by data processing, it ultimately offers the best prediction performance. The ANN model outperformed the ARIMA model in terms of performance, and Ma (2020) suggested that the ANN might contribute significantly to the LSTM model's performance. Additionally, the ARIMA-GARCH model was found to improve the accuracy of ARIMA by strengthening the white noise sequence. The LSTM model, compared to the other two models, adds additional variables to better differentiate between abrupt changes and sudden market fluctuations. The dataset used in this study was DELL's stock price data from 2010.

Sunki et al. (2024) concluded that forecasting time series in the stock market is a challenging task requiring technical methods and careful research. While no forecasting technique can predict stock prices with complete accuracy, time series forecasting offers valuable insights and aids investors in making informed decisions. These models incorporate various factors, including trend, seasonality, and autocorrelation, to generate projections. The study concluded that the ARIMA model provided a better fit to the data compared to both LSTM and FBProphet models. Based on RMSE values, ARIMA had the lowest (7.8919253), indicating the highest predictive accuracy for the dataset, followed by the LSTM model (RMSE of 10.33765) and FBProphet (RMSE of 9.11863).

Kobiela et al. (2022) conducted a study comparing ARIMA and LSTM models to determine which performs better based on selected input data, parameters, and feature count. The study used Mean Absolute Percentage Error (MAPE) and Mean Square Error (MSE) as the relative metrics for comparison. The results showed that ARIMA generally performs better, especially over longer periods, as it is better at processing single-feature (price) data. LSTM struggled with learning and only performed better for one-day forecasts. The study suggests that future research could involve adding more features and exploring hybrid models to potentially improve LSTM performance. The research concluded that for single-feature stock price prediction on NASDAQ data, ARIMA is currently more successful.

Varshney and Srivastava (2024) compared and calculated stock price forecasts using ARIMA and ANN. The results showed that the ANN approach was more accurate than ARIMA modeling for stock price forecasting. The study used the NIFTY 50 index closing price from NSE India and employed the Levenberg–Marquardt algorithm to simulate and train the network. The forecast errors for both models were minimal, indicating near-equal forecast performance. However, ANN predictions were found to be more accurate, with predicted lines nearly overlapping actual values. In contrast, ARIMA predictions showed divergence, especially towards the end of the study period.

Talati et al. (2024) suggested that LSTM can manage the sequential and nonlinear structure of financial data, making it effective at predicting stock prices. The authors believed that feature selection and data preparation could further enhance the model's performance. The study trained the model on three different datasets: Infosys (1996 to 2022), Microsoft (1986 to 2022), and TCS (2002 to 2022).

Huang (2023) concluded that the LSTM model could learn from sequential stock data patterns and include long-term dependencies, making it highly predictive for stock prices. The study found that the proposed model outperformed previous models by Xu, Cohen & Ullah, and Qasim, with a 35.18% and 5.86% improvement, respectively. The model achieved high accuracy (86.77%) and significantly reduced errors, with lower MSE and RMSE values compared to the original models. The model consisted of four layers (100, 50, 100, and 30 units) with dropout used to prevent overfitting. These findings demonstrate the efficiency of the LSTM model and the significance of social media variables for accurate stock movement forecasting. The study used historical stock price data from Google stocks from 2014 to 2019.

Adebiyi et al. (2014) examined the ARIMA and ANN models for predicting Dell stock prices. The study found that ARIMA (1,0,0) was the optimal configuration based on minimal forecast errors and a close match to actual prices. The ANN model with a 10-17-1 architecture also demonstrated high accuracy and minimal mistakes. Although both models performed well, the ANN model generally produced more accurate forecasts. Statistical tests revealed little variance between the actual and predicted values for either model. The study suggested that future research could explore hybrid models using new market indexes and recent stock data to further improve forecast accuracy.

Paper	**Machine-Learning models**	**Datasets**	**Evaluation metrics used**	**Results**
Stock Predicting based on LSTM and ARIMA	LSTM	Google Stock Price	RMSE	12.5283
	ARIMA	Google Stock Price	RMSE	197.1015
Comparison of ARIMA, ANN, and LSTM for Stock Price Prediction	LSTM	DELL's stock price	NA	NA
	ARIMA			
	ANN			
Time series forecasting of the stock market using	LSTM	Netflix data	RMSE	10.33765269
	ARIMA	Netflix data	RMSE	7.891925316

ARIMA, LSTM, and FB prophet				
	FB prophet	Netflix data	RMSE	9.118639484
ARIMA vs LSTM on NASDAQ stock exchange data	LSTM	NASDAQ data	MAPE	5.52
	ARIMA	NASDAQ data	MAPE	1.64
A Comparative Study of Future Stock Price Prediction Through Artificial Neural Network and ARIMA Modelling	ANN	NSE India data	RMSE	0.1140
	ARIMA	NSE India data	RMSE	1.4656
Stock Market Prediction Using LSTM Technique	LSTM	Infosys	RMSE	39.21
		Microsoft	RMSE	19.76
		TCS	RMSE	126.88
Enhancing Stock Market Prediction Through LSTM Modeling and Analysis	LSTM	Google stocks	Accuracy	86.77 %
Comparison of ARIMA and Artificial Neural Networks Models for Stock Price Prediction	ARIMA	Dell stock prices	RMSE	0.989716
	ANN	Dell stock prices	RMSE	0.90847

Tabel 1: Literature Review of Evaluation Metrics

Methodology

Overview

The goal of this project is to use machine learning techniques to predict the Istanbul Stock Exchange. This will be executed in a few steps (Figure 2). Collecting and reading the data is the first step, and then comes data preprocessing, to handle the missing values and identify the features from the dataset. Then data is split into training and testing and trained using LSTM and ARIMA models. Subsequently, we will test the models, and in the end, we'll assess model performance by tuning the parameters to see which model forecasts the future stocks more accurately.

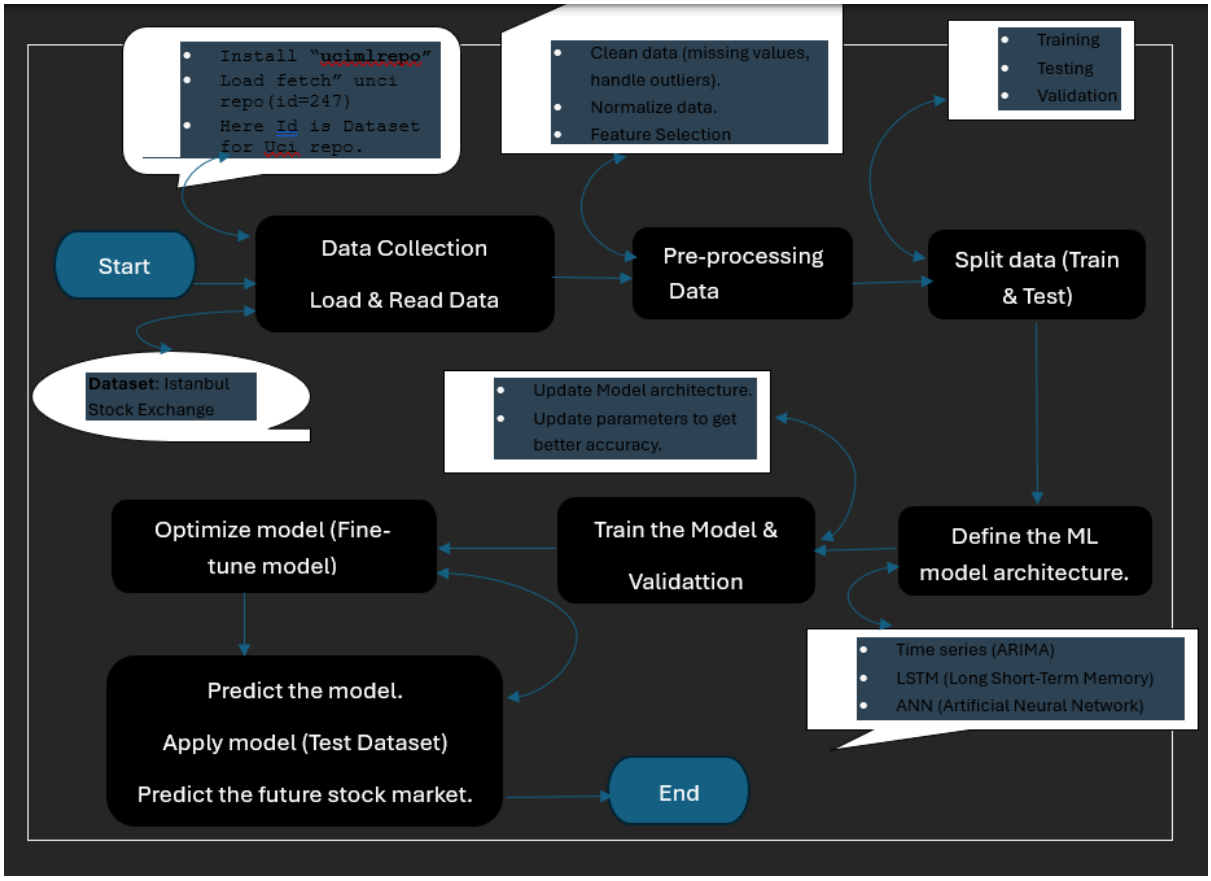


Figure 2: Data management plan flow chart

Long Short-Term Memory (LSTM):

LSTM are a type of Recurrent Neural Network (RNN) that process sequential data and collect long-term dependencies. "Hochreiter" and "Schmidhuber" created LSTM to address the issues identified by conventional RNN and machine learning methods. When handling time-series data, like stock prices, which are generally non-stationary and show trends and changing circumstances, LSTM proves to be particularly useful.

Architecture

By understanding the LSTM's architecture, we shall see in the following section how it solves this issue. At the high level, the LSTM works a lot like an RNN cell, and this is how the LSTM network works on the inside. The LSTM network is divided into three components, each of which performs a different function, as can been seen in the image below.

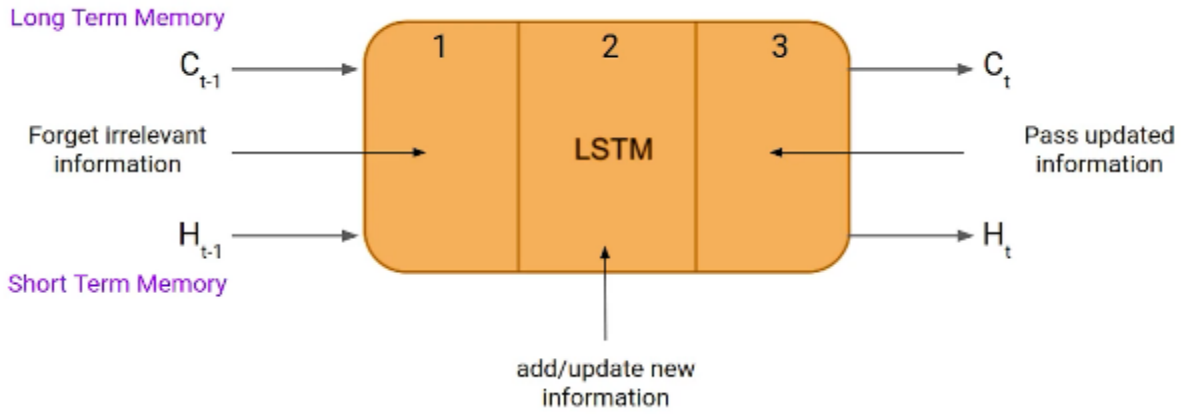


Figure 3: LSTM Architecture (Source: Analytics Vidhya, 2022)

As we can see from above figure [Figure 3], an LSTM unit consists of three gates, all of them responsible for regulating the information that enter and exit the memory cell (also called an LSTM cell). The first one is the "Forget gate", in which data from the preceding timestamp which, could either be kept or ignored depending on its importance. Secondly, the cell tries to learn new information from the current input coming to the cell itself to the second gate, called "input gate". The third and last gate, also known as the "output gate", is the one responsible for taking the modified data from the timestamp of the current cell to the next timestamp. This LSTM cycle is considered a single-time step.

An LSTM contains a hidden state, similar to a simple RNN, denoted below as 'H(t-1)' for the hidden state of the timestamp a moment ago and 'Ht' for the hidden state of the current timestamp, respectively. A long-term memory is a cell-state, denoted below as 'C(t-1)' for the past timestamp and C(t) for the present timestamp, respectively. Long-term memory is the cell state; short-term memory is the hidden state.

Let's use the equation to better understand how these gates function in the LSTM architecture [Figure 4].

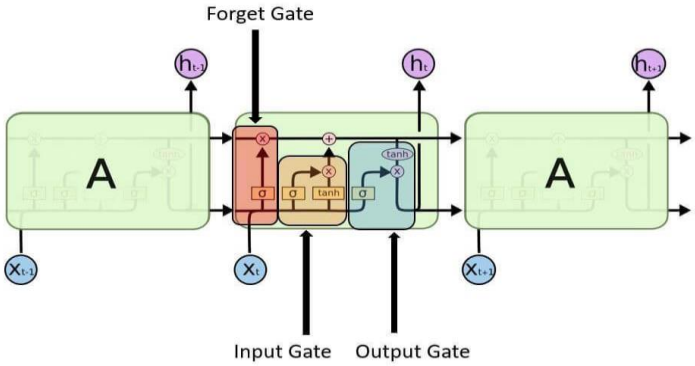


Figure 4: Long Short-Term Memory Network

Forget Gate:

In this gate, we must figure out whether to keep or remove the data from the previous time step. Below is the forget gate equation.

$$f_t = \sigma(X_t * W_{xf} + H_{t-1} * W_{hf} + b_f)$$

Here, Let's understand the working of the equation,

Xt: It's the timestamp's current input.

Uf: The input's associated weight.

Ht-1: The previous timestamp's hidden state

Wf: The weight matrix related to the hidden state.

Then this value is fed to a sigmoid function. This gives us f and t as a 0 or a 1. The cell state of the earlier timestamp is then complemented by this f and t.

The input and output gate equation is essentially the same as that of the forget gate. Likewise, because of the sigmoid function, its value will also range between 0 and 1. The equations below can be used to feed new data into the input gate and extract the output from the output gate.

Input-gate new information:

To handle new information, a sigmoid function is used to modify the existing data, and that's how the neural networks operate rather than classifying information to be important or not. As a result, the data has been updated completely. LSTM, on the other hand, uses a mechanism to transmit the data known as cell states while executing minimal addition and multiplication changes to the data.

Here is the equation to calculate the new information.

$$N_t = tanh\,(X_t * U_c + H_{t-1} * W_c)$$

The function of a hidden state at timestamp t-1 and input x at timestamp t now identifies the new information that has to be provided to the cell state. And tanh, is the activation function: the value of new information will be between -1 and 1 due to tanh: Here the value of Nt is added to the cell state at default timestamp if value of Nt is positive, and substracts the cell state at default timestamp if it is negative.Here, the cell state at the default timestamp is Ct-1 and the rest of values are the same what we computed.

Output-gate:

In this gate we will use Ot and tanh of the updated cell information to compute the current hidden state as shown below.

$$o_t = \sigma(X_t * U_o + H_{t-1} * W_o)$$

The hidden state, responsible for long-term memory (Ct), is determined using the output of the present step. To produce the output of the present timestamp, the activation function "SoftMax" is used on the hidden state Ht. It is with the highest predicted score in the output that the next prediction is made.

Essential elements of LSTM:

- **Memory Capability**: Because LSTMs can remember information over time, they are perfect for capturing long-term dependencies in sequential data.
- **Gated mechanism**: LSTMs use gates to manage information flow, enabling them to store important data and eliminate unnecessary information.

Why use LSTM for predicting the stock market?

Recent advances in data science have shown that for nearly all these sequence prediction problems, long short-term Memory networks have turned out to be the best solution (Talati et al.). Being able to predict what's going to happen next In traditional statistical methods (such as linear regression or Autoregressive Integrated Moving Average, or ARIMA), we assume the data is stationary and that things like the variance and the mean don't change over time.

Unlike standard neural networks that do not internally have feedback connections, LSTM can handle complete data sequences instead of just single points. It also maintains very high accuracy in identifying sequential patterns in data, such as speech, text, and time series, to name a few. Because of its deliberate scope designed specifically to extract critical insights from sequential data, LSTM has become a competitive tool in deep learning and artificial intelligence applications that have accelerated breakthroughs in a variety of industries.

Stock prices, on the other hand, show seasonality, patterns, and trends despite not being stationary. LSTMs are ideally suited for stock market prediction because they can manage this non-linear correlation within the data.

Autoregressive Integrated Moving Average (ARIMA)

An autoregressive integrated moving average (ARIMA) is a model of statistical analysis based on the use of time series data to predict future trends or understand more about the underlying data set by using what is already known about previous historical data. The ARIMA model is a type of regression analysis where the single dependent variable is tested against other changing (independent) variables. The model uses not the actual values but the differences in those values to predict future tendencies in the market in securities or the financial markets.

Architecture

An understanding of an ARIMA model can be gained by describing each of its components as follows:

1. Autoregressive (AR): A time-varying variable that depends on its own lagged (ie, old) values and is evaluated by itself is called an autoregression.
2. Integrated (I): To enable the time series to become stationary, the difference between the current and previous values is substituted for the data values, which are represented by integrated.
3. Moving average (MA): Considers a data point's connection with a residual error by applying a moving average model to observations that are lagged.

With a standard notation, every component in ARIMA operates as a parameter. Standard notations for ARIMA models are with p, d, and q, where the parameters are replaced by integer values to denote the type of ARIMA model that is being utilized. One way to define the parameters is as follows:

- AR (P): the model's autoregressive terms or the number of lag observations.

- I (d): the variation in the observation that is not seasonal, or the number of differencing cycles applied to the raw observations.
- MA (q): the moving average window size, can also referred as the moving average order.

Computer methods and machine learning approaches are used to compute the ARIMA model, which is complex and performs best on very large data sets. To make the data stationary in an ARIMA model, they are different. A model that demonstrates consistency proves the data stability across time. Since most of the market and economic data exhibits trends, the goal of differentiation is to eliminate any seasonal patterns or trends.

Seasonality (the tendency of the patterns in the time series to repeat over a period of 12 months) may skew the regression models. If a trend develops which is not self-evident, many calculations in the process cannot be completed, not getting the desired results.

To start with constructing an ARIMA model for a stock, we firstly download the maximum available stock price data. Then we determine the trend in data to find the lowest level of differencing (d). For a series we can differ for making it stationary if we have a 'lag-1' correlation between data points that is zero or negative or close to it. If the 'lag-1' is greater than zero, you might need to differenciate the series more. After that you compare the self-correlation and partial autocorrelation in order to find the number of regression (p) and the number of moving average (q).

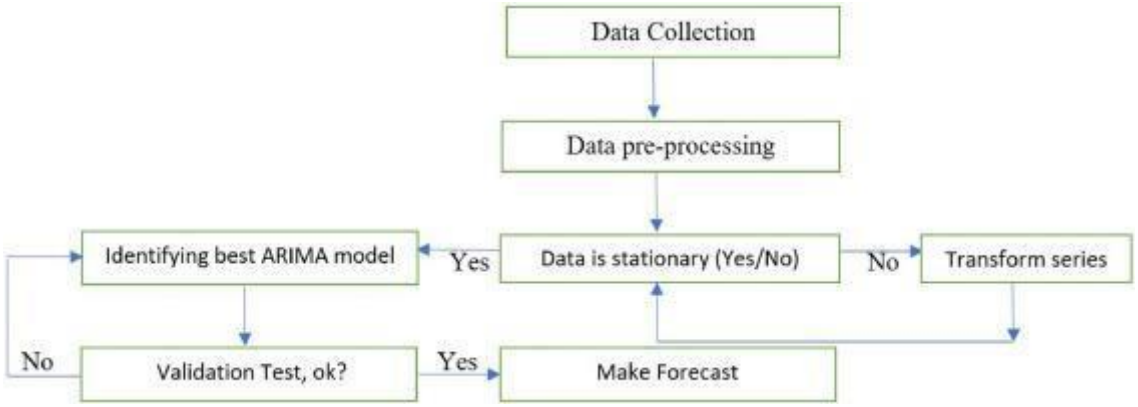


Figure 5: Methodologies for using the ARIMA model for forecasting

To understand the working of the ARIMA model in basic terms [Figure 5]:

- Collect data: Gather the data that will be studied or used to forecast trends.
- Review data: Ensure that there are no major long-term changes to the data. If so, you may need to make sure that data is stationary through small adjustments.
- Identify model: Examine the data to determine how much, if any, adjustment is necessary and how historical data influences current data. Python libraries can be used for the time series data.
- Check model accuracy: Compare the actual data with the predictions generated by your Python ARIMA model to see if the model accurately describes the data.
- Predict future: After the model is well developed, utilize it to project future events based on the prediction of your model.
- Improve model: If the predicted outcomes don't look great, tune the parameters of the model until the projections look great.
- And finally, run the model on the testing dataset, verify the predictions, and compare the predicted and actual values.

Why use ARIMA for predicting the stock market?

Since ARIMA captures patterns, seasonality, and correlation between past and future values, it is a useful time series data model for stock market forecasting. The model's adaptability to different market conditions can be achieved by fine-tuning it to certain data sets. ARIMA stabilizes non-stationary data by differencing the series, an important characteristic of stock prices. ARIMA's historical accuracy and statistical basis make it a useful tool in financial forecasting, however, it is best suited for short-term forecasts.

Data Preparation

Overview:

Using advanced machine learning techniques, namely "Long Short-Term Memory (LSTM) networks" and "ARIMA" model, the main goal of this research is to accurately forecast stock values for the "Istanbul Stock Exchange (ISE)". To create and assess prediction models, the approach consists of multiple phases, such as feature selection, data preprocessing, data-collecting, and detailed data analysis. This section describes how the dataset was prepared and analyzed for accurate stock market forecasting, along with the considerations and actions that were involved.

Data Collection

The UCI machine learning repository, which offers a large dataset for predicting stock market research, is the source of the data used in this work. Daily data from several global market indexes, including the ISE index, FTSE, DAX, NIKKEI, BOVESPA, and Standard & Poor (SP), EU, EM, is included in the dataset. Together, these indexes reflect a range of economic situations and offer a solid basis for simulating the movements of the Istanbul Stock Exchange.

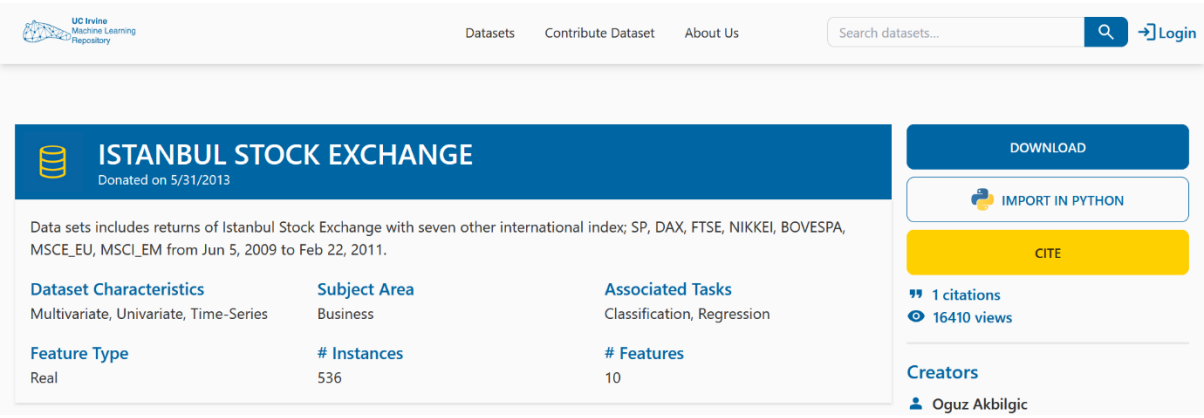


Figure 6: Dataset (Source: UC Irvine Machine Learning Repository)

Description of the dataset:

The primary target variable is the Istanbul Stock Exchange (ISE) index, which displays the value of an index of stocks and bonds traded there. Furthermore, the dataset contains other attributes that map to other important global indices, including:

- Standard & Poor's 500 index (SP): It is a key metric for both the US stock market and the world economy since it represents the economic growth of 500 large-cap US corporations.
- DAX: over 30 significant German firms that trade on the "Frankfurt Stock Exchange" are tracked by DAX, which offers market insights throughout Europe.

- FTSE: The FTSE, which tracks the top 100 firms listed on the “London Stock Exchange”, is an essential indicator of the state of the UK economy.
- NIKKEI: Considering the state of the Japanese economy, it consists of 225 business that are listed on the “Tokyo Stock Exchange”.
- BOVESPA: Brazil’s main stock market index, which shows how the country’s economy is doing.
- EU: A composite indicator that reflects the European Union’s economic activity.
- EM: The Emerging market index provides a perspective on the state of the developing countries’ economies by representing stocks from these markets.

Data Preprocessing

To ensure that the data is clear, consistent, and appropriate for analysis, preprocessing the data is an essential step in getting the dataset ready for modeling. The steps needed to preprocess the data are described in this section.

Handling missing values: Machine learning model performance can be greatly impacted by missing data. Therefore, handling null values, and missing data is the first step in preprocessing.

Scaling and Normalization:

All numerical features were and scaled to fall into a comparable range since any machine learning model, particularly LSTM, ARIMA, works better with normalized data. This enables greater convergence of the model and boosts the learning process.

MinMax Scaling: Features were scaled using the “MinMaxScaler” method. By modifying the values according to the lowest and highest values of each feature, this scaler converts the attributes to a specified range, usually [0,1]. To maintain the strength of each feature’s relationship, scaling was done to each feature separately.

Data Feature Selection

Feature engineering is the process of adding new features or changing pre-existing ones to improve the model’s ability to predict the future.

Feature selection: In addition to the ISE index, the main features used for the analysis are the DAX, SP, FTSE, NIKKEI, BOVESPA, EU, and EM. These characteristics were selected considering their potential impact on the target variable and its importance.

Below, finds the correlation matrix (heat map) for the data used for the model performance.

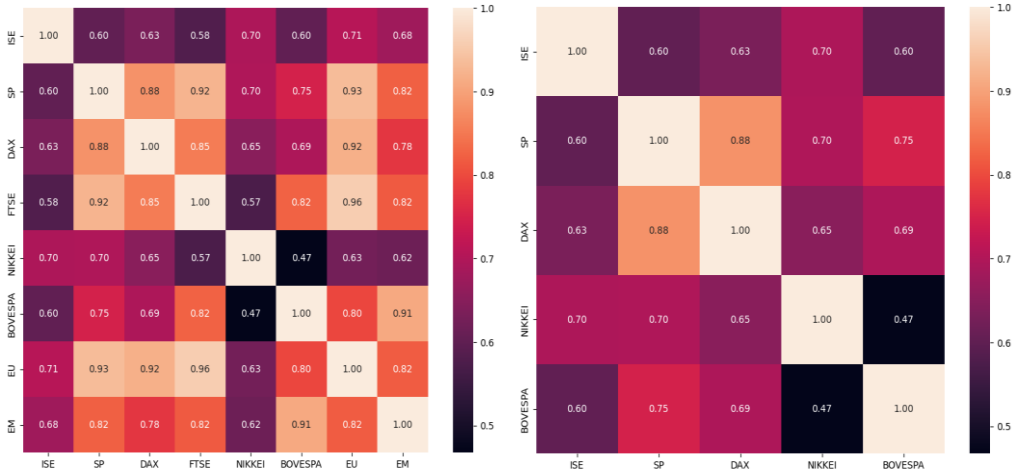


Figure 7 [a]: Matrix before feature extraction *Figure 7[b]: Matrix after feature extraction*

The first heatmap shows the ISE, SP, DAX, FTSE, NIKKEI, BOVESPA, EU, and EM among its seven indices. With values like 0.87 between DAX and FTSE and 0.95 between EU and FTSE, the heatmap demonstrates the strong correlation between European indices like the FTSE, Eu, and DAX. They have a great deal in common, indicating a strong regional connection. The EU (0.72) and BOVESPA (0.69) have moderate correlations with the Emerging Markets (EM) index, suggesting that there is some global trend alignment. Conversely, the NIKKEI exhibits poor correlations with the majority of indices, particularly with the SP (0.13) and DAX (0.26), indicating that it may be useful for feature extraction diversification since it gathers distinct market data.

ISE, SP, DAX, NIKKEI, and BOVESPA are among the five indices in the second heatmap, which focuses on a smaller number of more specific indexes. Certain correlations hold in this instance, such as the robust correlation of 0.72 between SP and BOVESPA and the high correlation of 0.69 between DAX and SP. There are fewer market perspectives available without FTSE, EU, and EM.

Transforming data:

The data must be transformed into a 3D array format, including the features, time steps, and samples, to be ready for the LSTM model. Whereas, for ARIMA models data must be stationary, which means that statistical parameters like mean, and variance shouldn't vary with time. Trends and seasonality are eliminated using transformations like differencing, which subtracts old data points from current ones, to achieve consistency.

Splitting the dataset:

The dataset was divided into training and testing data sets to assess how well the predictive models performed. Generally, a higher percentage of the data about 70-80% is used for training to feed the model with sufficient knowledge to learn, and the remaining data is used for testing.

Model Training and Analysis:

LSTM:

```
lstm_model.summary() # model summary
```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 1, 128)	69,632
dropout (Dropout)	(None, 1, 128)	0
lstm_1 (LSTM)	(None, 1, 64)	49,408
dropout_1 (Dropout)	(None, 1, 64)	0
lstm_2 (LSTM)	(None, 32)	12,416
dense (Dense)	(None, 1)	33

Total params: 131,489 (513.63 KB)
Trainable params: 131,489 (513.63 KB)
Non-trainable params: 0 (0.00 B)

Figure 8[a]: Model with all features

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do
  super().__init__(**kwargs)
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
lstm_3 (LSTM)	(None, 1, 128)	68,096
dropout_2 (Dropout)	(None, 1, 128)	0
lstm_4 (LSTM)	(None, 1, 64)	49,408
dropout_3 (Dropout)	(None, 1, 64)	0
lstm_5 (LSTM)	(None, 32)	12,416
dense_1 (Dense)	(None, 1)	33

Total params: 129,953 (507.63 KB)
Trainable params: 129,953 (507.63 KB)
Non-trainable params: 0 (0.00 B)

Figure 8[b]: Model after feature extraction

The figures above describe the creation of the sequential model using keras for sequence-based applications such as time series prediction. The first layer is an LSTM layer with 128 units that processes the input sequence and outputs the entire sequence. After that, a dropout layer randomly removes 30% of the units during training to prevent overfitting. A second LSTM layer with 64 units returns the entire sequence, followed by another dropout layer. When the output of the model is a single value, the third LSTM layer, which has 32 units is different such that it only returns the last output of the sequence. Since the last layer is dense and has only one unit, the model will only produce one continuous value. The model is compiled using the "Adam optimizer" with a learning rate of 0.0003, and the "Mean Squared Error (MSE) as the loss function, followed by "Mean Absolute Error (MAE) as the performance metric.

Regression activities involving sequential data, which means time series forecasting, frequently use this structure.

Training curve:

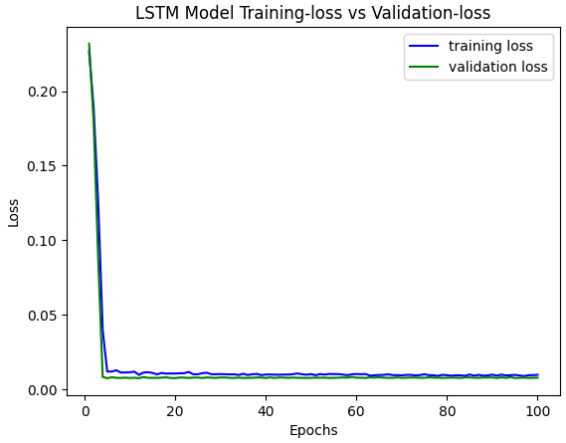


Figure 9[a]: Training curve with all features

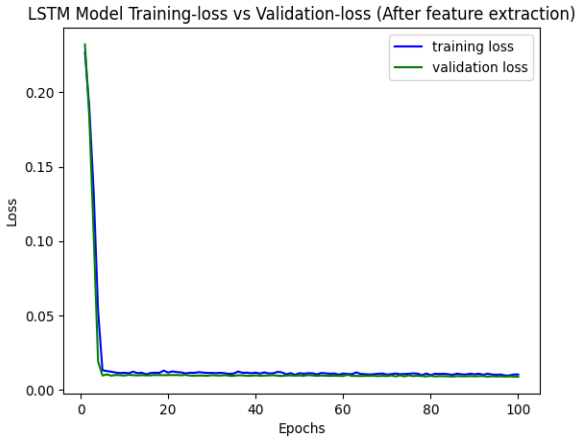


Figure 9[b]: Training curve after feature extraction

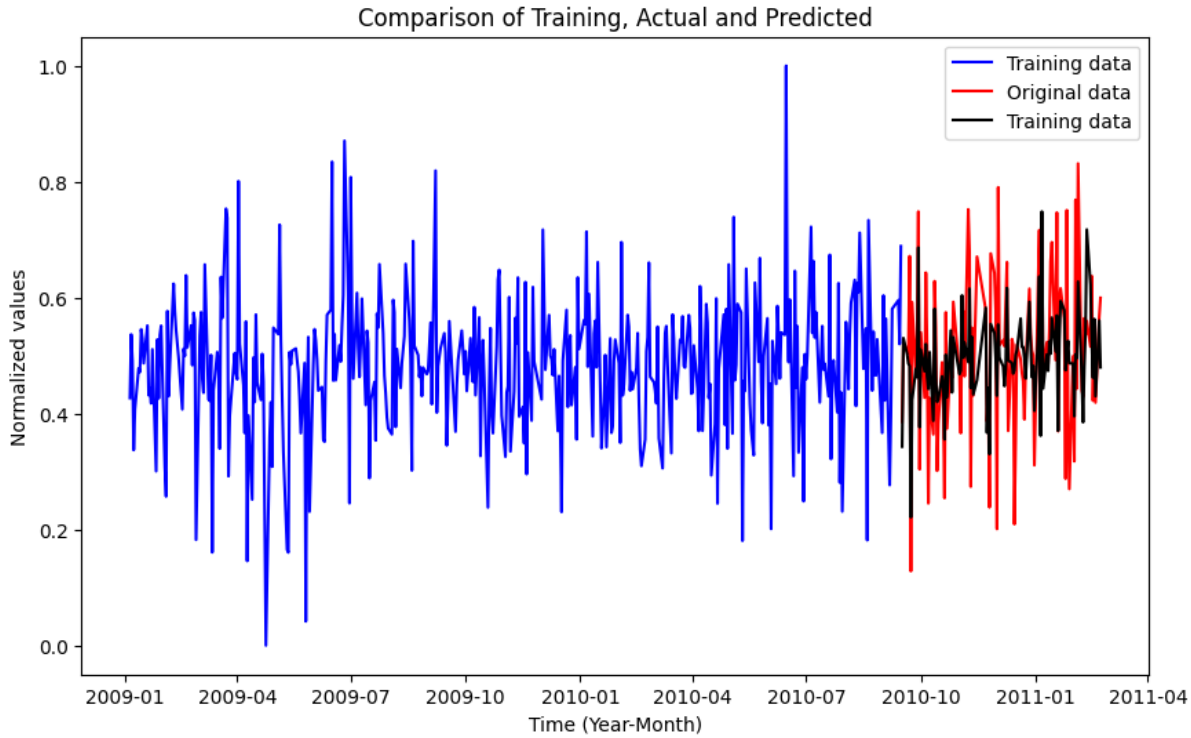


Figure 10: Training, Actual vs Forecasted Values

The model is trained on all features that are available in the above image. The blue line, which represents the training data, has a higher degree of volatility, suggesting that the model fits the data including the noise in the dataset quite closely. The model obtains a lower training loss of 0.100 despite this overfitting, indicating that it is well-tuned to the training set, maybe at the cost of its capacity to generalize to new data. While the black line shows the model's prediction, closely matching the training data, the red line

reflects the original target data. This could result in a minimal training loss but a worse performance on unknown data.

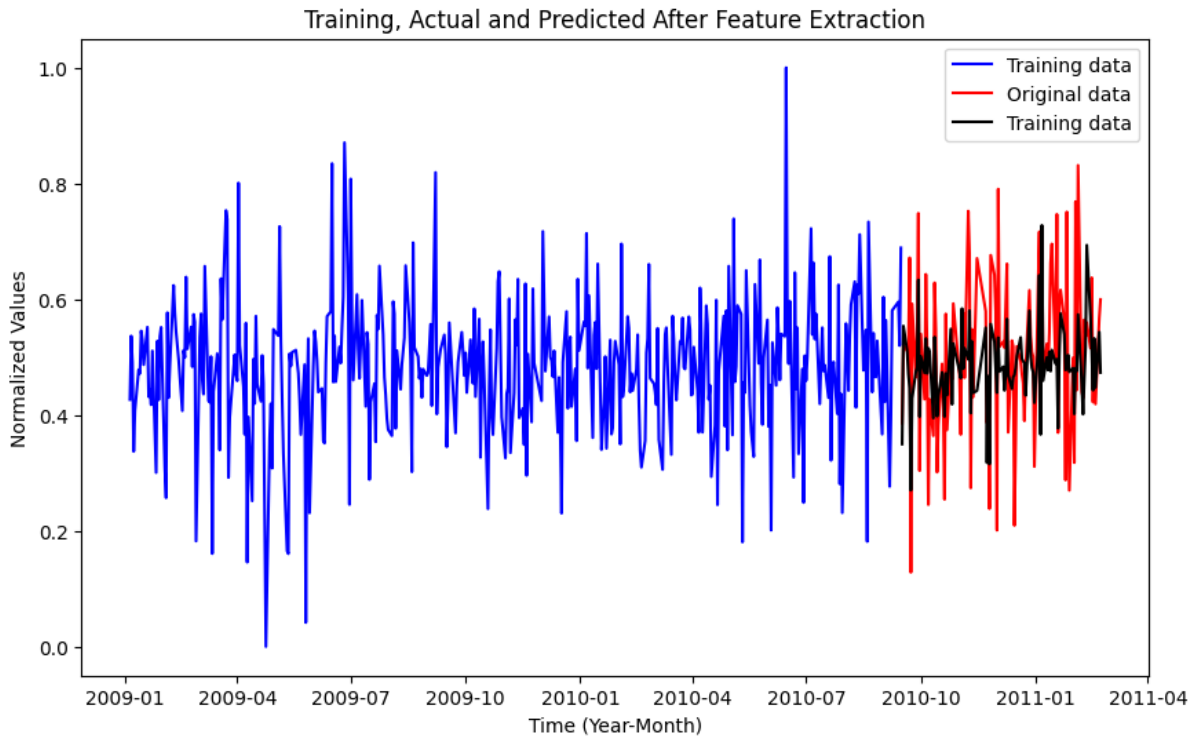
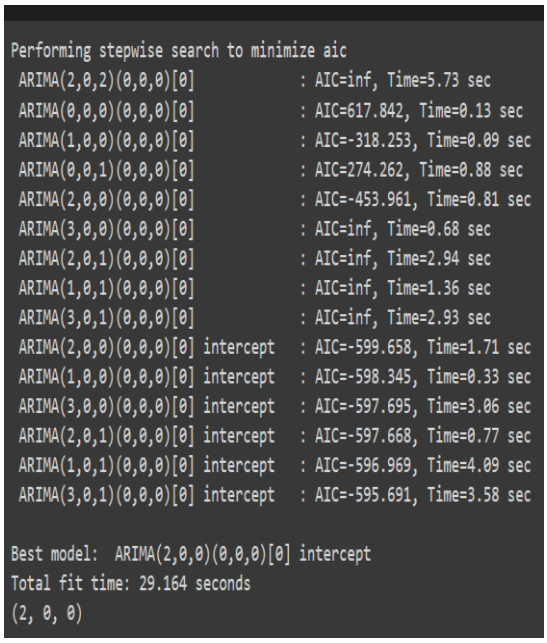


Figure 11: Training, Actual vs Predicted Values after Feature Extraction

Fewer features were used to train the model in this image. The training data after feature extraction is represented by the brown line, which exhibits less fluctuation than the first image and may indicate that noise has less of an impact on the model. The model's predictions, which closely match the actual data, are displayed on the yellow line, while the blue line depicts the original target data. Because it is not overfitting to noise, the model with fewer features is probably better at generalizing to new data, even though its training loss is 0.0127 larger than in the image [Figure 11].

ARIMA:

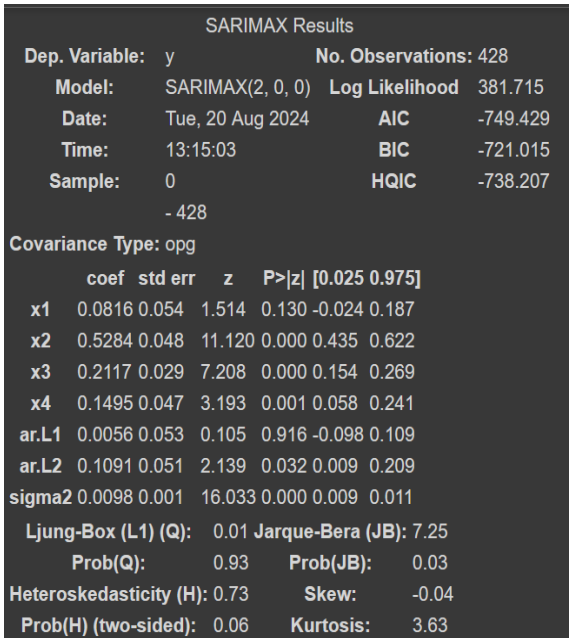
A code is designed to develop a better time series forecasting model by adding exogenous (independent) variables to the ARIMA technique. A function included in the "pmdarima" package, "auto_arima" iterates over various combinations of the ARIMA components (p, q, d) to automatically determine the optimal ARIMA model parameters (FIGURE). Then, a model is fitted using the "SARIMAX" function from the statsmodels.



```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0]             : AIC=inf, Time=5.73 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=617.842, Time=0.13 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=-318.253, Time=0.09 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=274.262, Time=0.88 sec
 ARIMA(2,0,0)(0,0,0)[0]             : AIC=-453.961, Time=0.81 sec
 ARIMA(3,0,0)(0,0,0)[0]             : AIC=inf, Time=0.68 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=inf, Time=2.94 sec
 ARIMA(1,0,1)(0,0,0)[0]             : AIC=inf, Time=1.36 sec
 ARIMA(3,0,1)(0,0,0)[0]             : AIC=inf, Time=2.93 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=-599.658, Time=1.71 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=-598.345, Time=0.33 sec
 ARIMA(3,0,0)(0,0,0)[0] intercept   : AIC=-597.695, Time=3.06 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=-597.668, Time=0.77 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=-596.969, Time=4.09 sec
 ARIMA(3,0,1)(0,0,0)[0] intercept   : AIC=-595.691, Time=3.58 sec

Best model:  ARIMA(2,0,0)(0,0,0)[0] intercept
Total fit time: 29.164 seconds
(2, 0, 0)
```

Figure 12 [a]: Finding optimal order using Auto-ARIMA



SARIMAX Results

Dep. Variable:	y	**No. Observations:**	428
Model:	SARIMAX(2, 0, 0)	**Log Likelihood**	381.715
Date:	Tue, 20 Aug 2024	**AIC**	-749.429
Time:	13:15:03	**BIC**	-721.015
Sample:	0	**HQIC**	-738.207
	- 428		

Covariance Type: opg

	coef	std err	z	P>|z|	[0.025	0.975]
x1	0.0816	0.054	1.514	0.130	-0.024	0.187
x2	0.5284	0.048	11.120	0.000	0.435	0.622
x3	0.2117	0.029	7.208	0.000	0.154	0.269
x4	0.1495	0.047	3.193	0.001	0.058	0.241
ar.L1	0.0056	0.053	0.105	0.916	-0.098	0.109
ar.L2	0.1091	0.051	2.139	0.032	0.009	0.209
sigma2	0.0098	0.001	16.033	0.000	0.009	0.011

Ljung-Box (L1) (Q):	0.01	**Jarque-Bera (JB):**	7.25
Prob(Q):	0.93	**Prob(JB):**	0.03
Heteroskedasticity (H):	0.73	**Skew:**	-0.04
Prob(H) (two-sided):	0.06	**Kurtosis:**	3.63

Figure 12[b]: SariMax Model Summary

Model summary description:

The auto_arima function defines SARIMAX as the model with an order of (2,0,0) that is utilized to forecast the target variable ‘y’. There were 428 data points used in the construction of this model. The model’s log-likelihood is 381.715, which is a good fit because higher values suggest a better match. Several criteria are used to compare this model’s quality to others: the Bayesian Information Criterion (BIC), which penalizes more complex models more heavily than the Akaike Information Criterion (AIC), which is -749.429; the Hannan-Quinn Information Criterion (HQIC), which balances model complexity and fit, is -738.207; and finally, AIC has the lower value indicates a better model.

A well-fitted and statistically reliable model is produced by the SARIMAX (2, 0, 0) model with four exogenous variables. While the AR component, particularly the second lag, reflects some of the auto-regressive behavior in the time series, the significant exogenous variable implies that external factors are essential in forecasting the target variable. The model assessment is typically good despite some variance from normality, which makes it a trustworthy forecasting tool for the provided data.

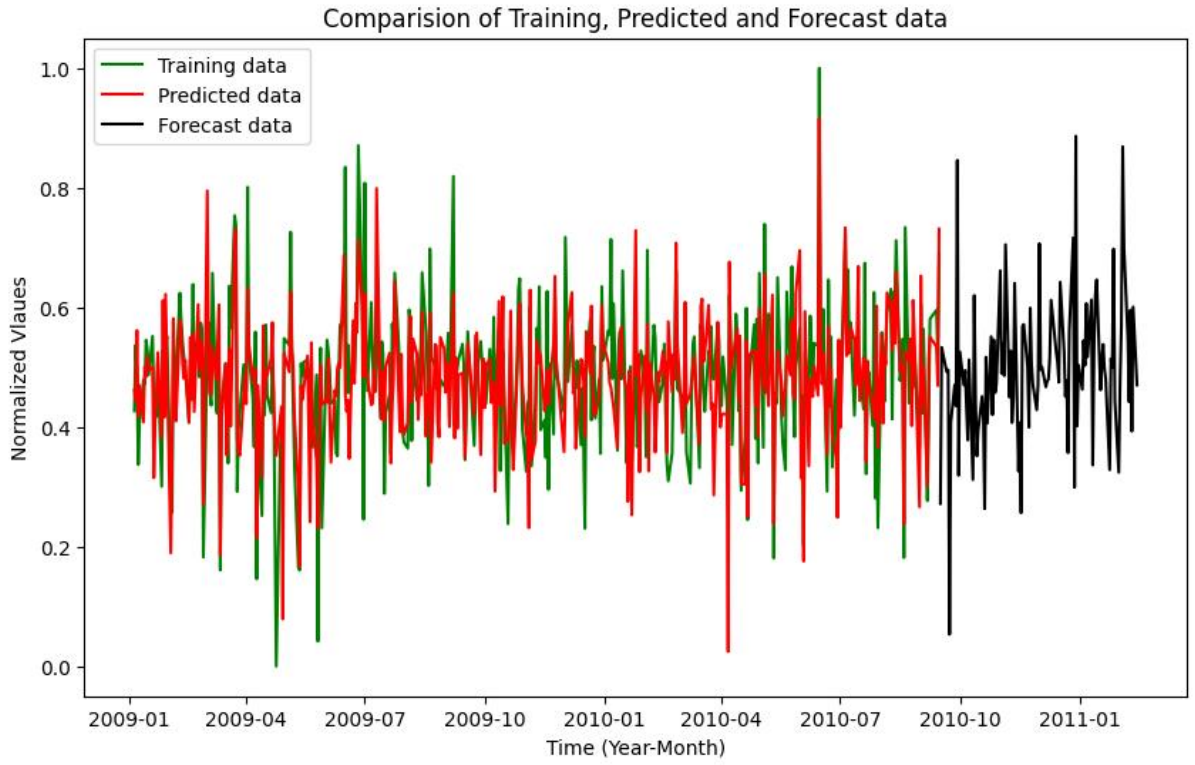


Figure 13 : ARIMA - Train, Actual vs Predicted Values

The predicted data in the above graph [13], which displays the ARIMA model trained with all features, closely resembles the training data, suggesting a possibility of overfitting. As seen by the variable projected data, the model performs well on training data but produces less trustworthy adaptation for future predictions because it captures even changes in the data.

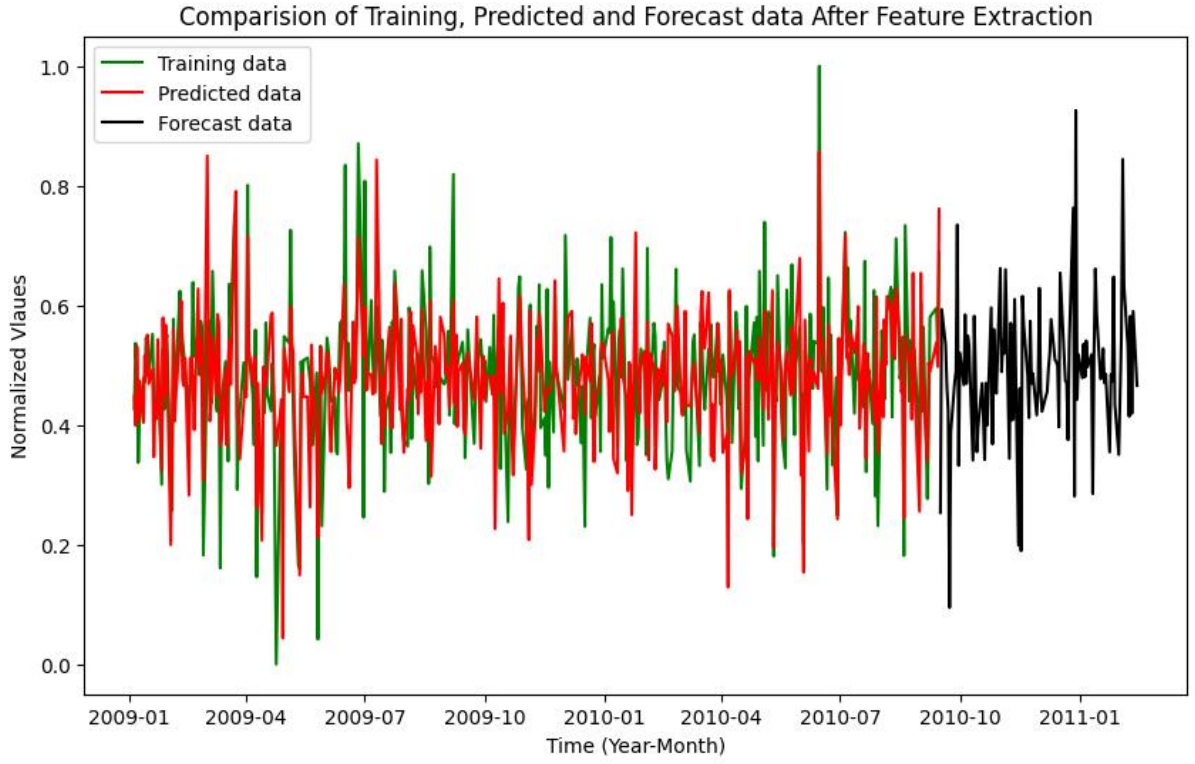


Figure 14 : ARIMA - Train, Actual vs Predicted Values with feature extraction

Following feature extraction, the model in this graph [14] becomes less sensitive to noise and wider based on key patterns. The model provides more stable, consistent, and better-generalized forecasts with a better balance between accuracy and robustness, even though the prediction lines are not as closely matched with the training data.

PROPHET MODEL:

Facebook created prophet, an open-source forecasting tool, to handle timeseries data with significant seasonal trends and the possibility of missing data or variations. It is easy to use, requires little adjustments, and works especially well for tracking seasonality on a daily, weekly, and annual basis. Prophet automatically manages holidays, specially used to forecast activities like traffic, trading, and sales. With only few manual modifications, the model is reliable and applicable to both basic and complicated datasets.

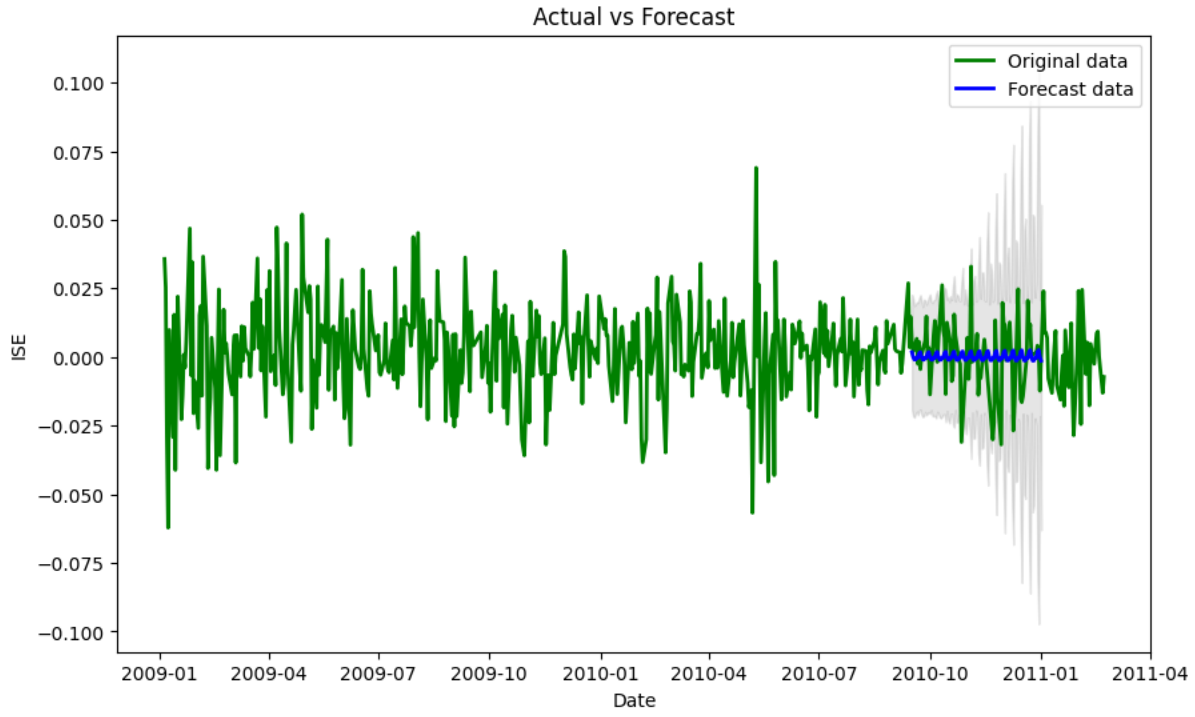


Figure15: Prophet - Actual vs Forecast Values

The graph shows a comparison of the prophet model predicted values (in blue) and the ISE index's actual historical values (in green). When compared to the projected data, the actual data exhibits substantial volatility, with frequent variations and a missing trend. The gap implies that the prophet model is having trouble explaining the ISE index's underlying volatility and complexity. As time passes, there is an increase amount of uncertainty in the model's predictions, as indicated by the confidence interval (shown by the gray-shaded area) surrounding the forecast. But even within this range, the forecast is unable to match the significant fluctuations observed in the real data. The result suggests that prophet may not be the best choice for high-risk financial data, such as the ISE index, even though it is an effective method for time series with distinct trends and seasonality.

Results And Analysis:

Several metrics, including (MAE), (MAPE), (MSE), (RMSE), are frequently used to assess the performance of statistical or machine learning models. The model's prediction reliability and accuracy are revealed by these indicators. The LSTM model exhibits comparatively low error rates before feature extraction. The model's average variation from the actual values is 0.0785 units, as indicated by the MAE of 0.0785. The average percentage variation from the actual data is estimated to be approximately 18% based on the MAPE of 17.97%. The model is producing predictions that are reasonably near to the actual values, with fewer squared errors, according to the MSE of 0.01078 and RMSE of 0.1038. It makes sense that RMSE would be somewhat higher than MAE because it assigns greater weight to larger errors.

The performance of the LSTM model marginally declined after feature extraction. The mean absolute error (MAE) went up to 0.0838, suggesting a modest increase. Additionally, the MAPE rose to 19.51%, indicating that the average percentage error is now greater than it was before feature extraction at roughly 19.51%. But both the MSE and the RMSE increased to 0.01231 and 0.1109, accordingly, suggesting that the model's predictions are now less accurate when the extracted features are used. The LSTM model's performance did not increase with feature extraction. Rather, all error metrics experienced a minor rise, suggesting that the feature extraction process may not have been successful or may have added noise or redundant features that negatively impacted model performance.

Lower error rates were obtained when the ARIMA model was first trained will all the features, resulting in a mean Absolute Error (MAE) of 0.080 and RMSE of 0.106. These reduced error values show how well the model matches the training set, picking up on even the smallest differences and trends. This, however, may indicate overfitting, a condition in which the model gets overly adapted to the training set, hence impairing its capacity to generalize effectively to fresh, untested data. Next, to feature extraction, which involved removing unnecessary or less significant features, the model's error rates marginally jumped, with an RMSE of 0.114 and an MAE of 0.085. These larger error values point to a more generalized model that is less noise-sensitive and more suited for real-world forecasting, even though they also reflect a less accurate fit on the training set. By striking a balance between model complexity and robustness, the feature-extracted model trades off some accuracy for better generalization and stability.

The error measurements of the prophet model show that the RMSE is roughly 0.013 and the MAE is about 0.01. The RMSE is marginally higher than the MAE, indicating that although the model performs rather well on average, its capacity to capture significant deviations is limited. This suggests that forecasting the more erratic components of the data may be difficult.

Numerous tests were carried out by adjusting the hyperparameters to determine the ideal model structure after an LSTM model was defined and developed to estimate a single constant value. The number of timesteps, the number of units in LSTM layers, the dropout rate, and the learning rate were the main hyperparameters that were adjusted during these studies. The model was trained for 50 epochs in the first experiment using a single timestep, 100 units, a 0.2 dropout rate, and a learning rate of 0.001. The validation MAE of 0.0998 showed a decent result for short-term prediction. Later tests found that while the training length got higher, performance was somewhat improved, with a validation MAE of 0.1039, when the timestep and units were increased to 8 and 120, respectively, while the learning rate was lowered to 0.0001. On the other hand, trails with a timestep of 10 and 20 demonstrated that more errors appeared by a combination of a reduced learning rate and additional increases in timestep and units, indicating problems with overfitting and challenging convergence.

The model performed poorly in subsequent trials with smaller unit counts and extremely low learning rates, as demonstrated by larger MAE values and slower training, suggesting that the model had difficulty learning under these circumstances. When the learning rate was slightly increased to 0.00001 and 0.0001, the model performed better, especially when a timestep of 5 and 100 units was used. However, the model did not stabilize and produce consistent results until a learning rate of 0.0003. The model performed exceptionally well, which involved configuring the model with a timestep of 1, a batch size of 16, 128 LSTM units, a dropout rate of 0.3, and a learning rate of 0.0003. The model was trained for 100 epochs, and the results showed that the training process produced a low loss of 0.0100 and an MAE of 0.0784 on the training data. The validation results, on the other hand, were extremely promising, showing a validation loss of 0.0081 and a validation MAE of 0.0726. These figures suggest that the models were well-tuned and capable of producing precise predictions, effectively striking a balance between underfitting and overfitting. Out of all the configurations examined, this one proved to be the most efficient, offering the best performance.

Model	**Dataset**	**Train Size**	**Test Size**	**RMSE**	**RMSE (after feature extraction)**
LSTM	Istanbul Stock Exchange	428	108	0.10513	0.11152
ARIMA	Istanbul Stock Exchange	428	108	0.10575	0.11407
Prophet	Istanbul Stock Exchange	428`	108	0.01312	N/A

Tabel 2: Stock market forecasting results

Conclusion

The results show that, although the LSTM model performed well overall, its performance declined after feature extraction, as indicated by increases in the MAE, MAPE, MSE, and RMSE. This implies that noise or redundant features may have been included during the feature extraction process, which would have reduced the predicted accuracy of the model. On the other hand, the ARIMA model showed promise at first, showing lower error metrics that would point to overfitting. However, after feature extraction, the model became more generalized at the cost of marginally higher error rates.

Though the RMSE values seen better in prophet model, it's growing uncertainty at the conclusion of the timeframe suggests that it underestimated the original data's volatility. This implies that even when the forecast accurately depicts the overall trend, it still can capture the more variations from the real data.

The LSTM model's hyperparameter tuning showed that precise changes, especially in learning rates and timestep counts, have a big impact on the model's performance. Low training loss and MAE were attained by the ideal configuration, suggesting that the model was properly calibrated to balance between underfitting and overfitting. Though typically accurate, the prophet model had trouble capturing large variations, as seen by the fact that its RMSE was marginally greater than the MAE. This implies that forecasting more unpredictable data components may have constraints.

In conclusion, the LSTM model with improved hyperparameters performed the best overall even though other models showed the ability to generate correct predictions. To prevent overfitting and ensure the model's stability in various forecasting circumstances, special attention must be paid to feature selection and hyperparameters adjustment.

Reference:

1. Qian, H., 2022. Stock predicting based on LSTM and ARIMA. In: *Proceedings of the International Conference on Economic Development and Business Culture (ICEDBC 2022).* Atlantis Press.
2. Ma, Q., 2020. Comparison of ARIMA, ANN, and LSTM for stock price prediction.
3. Sunki, A., Satyakumar, C., Surya Narayana, G., Koppera, V. and Hakeem, M., 2024. Time series forecasting of the stock market using ARIMA, LSTM, and FB Prophet. *MATEC Web of Conferences*.
4. Kobiela, D., Krefta, D., Król, W. and Weichbroth, P., 2022. ARIMA vs LSTM on NASDAQ stock exchange data. *Procedia Computer Science.*
5. Varshney, S. and Srivastava, P., 2024. A comparative study of future stock price prediction through artificial neural network and ARIMA modelling. *Journal of the Institution of Engineers (India): Series C.*
6. Talati, D., Patel, M. and Patel, B., 2024. Stock market prediction using LSTM technique. *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*.
7. Huang, W., 2023. Enhancing stock market prediction through LSTM modeling and analysis. *School of Software, South China Normal University, Guangzhou, China*.
8. Adebiyi, A., Aderemi, A. and Ayo, C.K., 2014. Comparison of ARIMA and artificial neural networks models for stock price prediction. *Journal of Data Science.*
9. Ouattara, A. (2023). Understanding the difference between technical and fundamental analysis in investment. *LinkedIn*.
10. Analytics Vidhya. (2021). Introduction to Long Short-Term Memory (LSTM).
11. Analytics Vidhya. (2022). An overview on Long Short-Term Memory (LSTM).
12. Khan, F.M. and Gupta, R., 2020. ARIMA and NAR based prediction model for time series analysis of COVID-19 cases in India. *Journal of Safety Science and Resilience*.
13. Istanbul Stock Exchange Data Set (2013). *UC Irvine Machine Learning Repository*.

Appendix:

```
# -*- coding: utf-8 -*-
"""
Created on Mon Ju1 5 22:24:49 2024

@author: Raghavendhra Devineni

Original file is located at
    https://colab.research.google.com/drive/13czNhF8Tq52Yguwa3KBaxwFkGODUmqqG
"""



# Import libraries for data preprocessing and model training
from ucimlrepo import fetch_ucirepo
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import numpy as np
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras.optimizers import Adam
from sklearn.metrics import mean_absolute_error, mean_squared_error,
mean_absolute_percentage_error
import seaborn as sns
```

```
TF_ENABLE_ONEDNN_OPTS=0

# Fetch dataset istanbul stock exchange dataset id
ise_data = fetch_ucirepo(id=247)

print("dataset type: ", type(ise_data), "\n")

print("keys: ", ise_data.keys(), "\n")

data_info = ise_data.data['features']
print("data type: ", type(data_info), "\n")

print("print first 5 rows..!")
print(data_info.head(), "\n")

stock_data = data_info

print(stock_data.describe(),"\n \n")

print(stock_data.info(),"\n \n")

print(ise_data.variables,"\n \n")

print(stock_data['date'].head(),"\n \n")
```

```
stock_data['date'] = pd.to_datetime(stock_data['date'], format='%d-%b-%y')

print(stock_data['date'].head(),"\n", "\n")

stock_data.set_index('date', inplace=True)

print("Check null values...!")
data_null_values = stock_data.isnull().sum()
print("null values found in dataset: ", data_null_values,"\n \n")

print("Calculating the mean for each month...!")
stock_mean_data = stock_data.resample('M').mean()
print(stock_mean_data.head(),"\n \n")

print("Removing the duplicate columns...!")
stock_mean_data = stock_mean_data.loc[:, ~stock_mean_data.columns.duplicated()]

print(stock_mean_data.head(),"\n \n")

fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(15, 10), constrained_layout=True)
data_columns = stock_mean_data.columns
for col, ax in enumerate(axes.flat):
    if col < len(data_columns):
        stock_mean_data[data_columns[col]].plot(ax=ax)
```

```
        ax.set_title(data_columns[col])
        ax.set_xlabel('year')
        ax.set_ylabel('price in mean')

plt.show()

print("Removing the duplicate columns...!")
stock_data = stock_data.loc[:, ~stock_data.columns.duplicated()]
print(stock_data.head(),"\n \n")

X_stock_key_features = ['SP', 'DAX', 'FTSE', 'NIKKEI', 'BOVESPA', 'EU', 'EM']

Y_stock_target = ['ISE']

def train_test_split_data(stock_data, X_stock_features, Y_stock_target):

  x_train, x_test, y_train, y_test = train_test_split(stock_data[X_stock_key_features], +
stock_data[Y_stock_target],
                                                      test_size=0.2, random_state=42)

  return x_train, x_test, y_train, y_test

x_train, x_test, y_train, y_test = train_test_split_data(stock_data, X_stock_key_features,
Y_stock_target)

print(len(x_train))
```

```
print(y_train.shape)

print(y_test.shape)


"""# **Normalization**"""


def normalize_data(x_train, x_test, y_train, y_test):


  scaler_data =MinMaxScaler()


  norm_x_train = scaler_data.fit_transform(x_train) # features variable

  norm_x_test = scaler_data.transform(x_test)

  norm_y_train = scaler_data.fit_transform(y_train) # targeted variable

  norm_y_test = scaler_data.transform(y_test)


  return norm_x_train, norm_x_test, norm_y_train, norm_y_test


norm_x_train, norm_x_test, norm_y_train, norm_y_test = normalize_data(x_train, x_test,
y_train, y_test)


print(norm_x_train.shape)

print(norm_y_train.shape)

print(norm_x_test.shape)

print(norm_y_test.shape)
```

```
norm_x_train = norm_x_train.reshape((norm_x_train.shape[0], 1, norm_x_train.shape[1]))

norm_x_test = norm_x_test.reshape((norm_x_test.shape[0], 1, norm_x_test.shape[1]))


print(norm_x_train.shape)

print(norm_y_train.shape)

print(norm_x_test.shape)

print(norm_y_test.shape)



def lstm_model(norm_x_train):

  model = Sequential()

  model.add(LSTM(units=128, return_sequences=True, input_shape=(norm_x_train.shape[1], norm_x_train.shape[2])))

  model.add(Dropout(0.3))


  model.add(LSTM(units=64, return_sequences=True))

  model.add(Dropout(0.3))


  model.add(LSTM(units=32))

  model.add(Dense(1)) # predicting only single values index(ISE) (predict single continuous value)


  model.compile(loss='mean_squared_error',

                optimizer=Adam(learning_rate=0.0003),

                metrics=['mean_absolute_error'])
```

```
  return model

lstm_model = lstm_model(norm_x_train)

lstm_model.summary()

history = lstm_model.fit(norm_x_train, norm_y_train, epochs=100, batch_size=16,
                         validation_split=0.2, verbose=1)

y_pred = lstm_model.predict(norm_x_test)
y_pred.shape

y_pred_train = lstm_model.predict(norm_x_train)
y_pred_train.shape

#Access the training data  to check the training loss and Mean Absolute Error (MAE)
training_hist = history.history

train_loss, train_mean_absolute_error = lstm_model.evaluate(norm_x_test, norm_y_test)
print("\n")
print("training loss: ", train_loss)
```

```
# visualize the training loss and validation loss from training hstory
training_loss = training_hist['loss']
training_val_loss = training_hist['val_loss']
training_epochs = range(1, len(training_loss)+1)
plt.plot(training_epochs, training_loss, 'b', label='training loss')
plt.plot(training_epochs, training_val_loss, 'g', label='validation loss')
# Set the title and labels for plot
plt.title("LSTM Model Training-loss vs Validation-loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
# Add legend to differentiate between the lines on graph
plt.legend()
# show the plot
plt.show()


plt.figure(figsize=(10,6))


plt.plot(range(len(norm_y_train)), norm_y_train, label='Training data', color='blue')


plt.plot(range(len(norm_y_train), 536), norm_y_test, label='Original data', color='red')


plt.plot(range(len(norm_y_train), 536), y_pred, label='Training data', color='black')


plt.title("Comparison of Training, Actual and Predicted")
plt.xlabel('Time Steps')
```

```
plt.ylabel('Normalized values')
plt.legend()

plt.show()

def calculate_error_rate(norm_y_test, y_pred):
  y_pred = np.array(y_pred)
  norm_y_test = np.array(norm_y_test)

  lstm_mae = mean_absolute_error(norm_y_test, y_pred)

  lstm_mape = mean_absolute_percentage_error(norm_y_test, y_pred)

  lstm_mse = mean_squared_error(norm_y_test, y_pred)

  lstm_rmse = np.sqrt(lstm_mse)

  return lstm_mae, lstm_mape, lstm_mse, lstm_rmse

lstm_mae, lstm_mape, lstm_mse, lstm_rmse = calculate_error_rate(norm_y_test, y_pred)

print(f"Mean absolute error: {lstm_mae} ")
print(f"Mean absolute percentage Error: {lstm_mape} ")
print(f"Mean squared Error: {lstm_mse} ")
print(f"Root mean square error: {lstm_rmse} ")
```

"""# **LSTM model with feature extraction**"""

stock_data.head()

stock_corr_matrix_data = stock_data.corr()

Display the correlation matrics

stock_corr_matrix_data

plt.figure(figsize=(9,8))

sns.heatmap(stock_corr_matrix_data, annot=True, fmt=".2f")

plt.show()

stock_feature_extracted_data = stock_data.drop(columns = ['FTSE','EU','EM'])

print(stock_feature_extracted_data.head())

fe_stock_corr_matrix_data = stock_feature_extracted_data.corr()

print("\n",fe_stock_corr_matrix_data.head())

plt.figure(figsize=(9,8))

```
sns.heatmap(fe_stock_corr_matrix_data, annot=True, fmt=".2f")


plt.show()


X_stock_key_features = ['SP', 'DAX', 'NIKKEI', 'BOVESPA']


Y_stock_target = ['ISE']


x_train, x_test, y_train, y_test = train_test_split_data(stock_feature_extracted_data,
X_stock_key_features, Y_stock_target)


print(len(x_train))


print(y_train.shape)

print(y_test.shape)


fe_norm_x_train, fe_norm_x_test, fe_norm_y_train, fe_norm_y_test =
normalize_data(x_train, x_test, y_train, y_test)


print(fe_norm_x_train.shape)

print(fe_norm_y_train.shape)

print(fe_norm_x_test.shape)

print(fe_norm_y_test.shape)


fe_norm_x_train = fe_norm_x_train.reshape((fe_norm_x_train.shape[0], 1,
fe_norm_x_train.shape[1]))
```

```
fe_norm_x_test = fe_norm_x_test.reshape((fe_norm_x_test.shape[0], 1,
fe_norm_x_test.shape[1]))



print(fe_norm_x_train.shape)

print(fe_norm_y_train.shape)

print(fe_norm_x_test.shape)

print(fe_norm_y_test.shape)



def lstm_model(norm_x_train):

  # Build the LSTM model from scratch

  model = Sequential()

  model.add(LSTM(units=128, return_sequences=True, input_shape=(norm_x_train.shape[1],
norm_x_train.shape[2]))) #layer 1 with 50 units

  model.add(Dropout(0.3)) # prevent overfitting given 10% loss for every epoch

  model.add(LSTM(units=64, return_sequences=True)) # layer 2 with 64 units will only
return the last output seq

  model.add(Dropout(0.3))

  model.add(LSTM(units=32))

  model.add(Dense(1)) # predicting only single values index(ISE) (predict single continuous
value)




  model.compile(loss='mean_squared_error', #calculate the error (pred & act)

                optimizer=Adam(learning_rate=0.0003), #minimize the loss function.

                metrics=['mean_absolute_error']) # compiling the model
```

```
  return model



fe_lstm_model = lstm_model(fe_norm_x_train)



fe_lstm_model.summary()



# train the model

history1 = fe_lstm_model.fit(fe_norm_x_train, fe_norm_y_train, epochs=100,
batch_size=16,

                validation_split=0.2, verbose=1)



training_hist = history1.history



# evaluate model

train_loss, train_mean_absolute_error = fe_lstm_model.evaluate(fe_norm_x_test,
fe_norm_y_test)

print("\n")

print("training loss: ", train_loss)







training_loss = training_hist['loss']

training_val_loss = training_hist['val_loss']

training_epochs = range(1, len(training_loss)+1)

plt.plot(training_epochs, training_loss, 'b', label='training loss')
```

```
plt.plot(training_epochs, training_val_loss, 'g', label='validation loss')

# Set the title and labels for plot

plt.title("LSTM Model Training-loss vs Validation-loss (After feature extraction)")

plt.xlabel("Epochs")

plt.ylabel("Loss")

# Add legend to differentiate between the lines on graph

plt.legend()

# show the plot

plt.show()


# predict the test data with lstm model

y_pred = fe_lstm_model.predict(fe_norm_x_test)

y_pred.shape


plt.figure(figsize=(10,6))


plt.plot(range(len(fe_norm_y_train)), fe_norm_y_train, label='Training data',
color='#7D1007')


plt.plot(range(len(fe_norm_y_train), 536), fe_norm_y_test, label='Original data',
color='#06C')


plt.plot(range(len(fe_norm_y_train), 536), y_pred, label='Training data', color='#F6D173')


plt.title("Training, Actual vs Predicted")

plt.xlabel('Time Steps')
```

```
plt.ylabel('Normalized Values')

plt.legend()

plt.show()




lstm_mae, lstm_mape, lstm_mse, lstm_rmse = calculate_error_rate(fe_norm_y_test, y_pred)


print(f"Mean absolute error: {lstm_mae} ")

print(f"Mean absolute percentage Error: {lstm_mape} ")

print(f"Mean squared Error: {lstm_mse} ")

print(f"Root mean square error: {lstm_rmse} ")


"""## Implementing ARIMA model"""



from pmdarima import auto_arima

from statsmodels.tsa.statespace.sarimax import SARIMAX


X_stock_key_features = ['SP', 'DAX', 'FTSE', 'NIKKEI', 'BOVESPA', 'EU', 'EM']

# Target featuer used for forecasting the future

Y_stock_target = ['ISE']


# split the extracted feature data into training and testing

x_train, x_test, y_train, y_test = train_test_split_data(stock_data, X_stock_key_features,
Y_stock_target)
```

```
print(len(x_train))


print(y_train.shape)

print(y_test.shape)


# Function used to normalize the features and target variables

norm_x_train, norm_x_test, norm_y_train, norm_y_test = normalize_data(x_train, x_test,
y_train, y_test)


print(norm_x_train.shape)

print(norm_y_train.shape)

print(norm_x_test.shape)

print(norm_y_test.shape)


# convert the norm_y_train and norm_y_test to 1D array through flatten

norm_y_train_flat = norm_y_train.flatten()

norm_y_test_flat = norm_y_test.flatten()


print(norm_y_train_flat.shape)

print(norm_y_test_flat.shape)


# Build ARIMA model,

# Find the best order for AR(P), I(q), M(d) parameters using auto_arima,

# set the seasonality = True, to get the best sasonality order.
```

```
arima_auto_model = auto_arima(norm_y_train_flat, exogenous=norm_x_train,
seasonal=False,
                              trace=True, error_action='ignore',
                              suppress_warnings=True, stepwise=True)



# Identify the best order through auto_arima
arima_best_order = arima_auto_model.order
arima_best_order


# Use the SARIMAX model using the optimal order to train
# SARIMAX use both features and target for training the data unlike ARIMA it accepts only
target data
arima_model = SARIMAX(norm_y_train_flat, exog=norm_x_train, order=arima_best_order)
arima_model = arima_model.fit(disp=False)


# summarize the arima_model
arima_model.summary()


# use the model to predict on the training data
armia_predict = arima_model.predict(start=0, end=len(norm_y_train_flat)-1,
                                    exog=norm_x_train)
# print the shape of the model
armia_predict.shape


# Forecast the model on the test data (target)
```

```
arima_forecast = arima_model.forecast(steps=len(norm_y_test_flat), exog=norm_x_test)
# print the shape of the model
arima_forecast.shape


# visualize the data
# Set the figure size
plt.figure(figsize=(10,6))
# plot the training data and predicted data
plt.plot(norm_y_train_flat, label='Training data', color='g')
plt.plot(armia_predict, label='Predicted data', color='red')


# Add the title and labels for the graphs
# legend to differentiate between the lines and graph
plt.title('Comparision of Training vs Predicted data')
plt.xlabel('Time')
plt.ylabel('Normalized Values')
plt.legend()
# show the plot
plt.show()


# visualize the data
# Set the figure size
plt.figure(figsize=(10,6))
# plot the training data, predicted data, and forecast data
plt.plot(norm_y_train_flat, label='Training data', color='g')
```

```
plt.plot(armia_predict, label='Predicted data', color='red')

fore_start_index = len(norm_y_train_flat) + np.arange(len(arima_forecast))

plt.plot(fore_start_index, arima_forecast, label='Forecast data', color='black')

# Add the title and labels for the graphs

# legend to differentiate between the lines and graph

plt.title('Comparision of Training, Predicted and Forecast data')

plt.xlabel('Time')

plt.ylabel('Normalized Vlaues')

plt.legend()

# show the plot

plt.show()


# Function used to calculate the error metrics

lstm_mae, lstm_mape, lstm_mse, lstm_rmse = calculate_error_rate(norm_y_test,
arima_forecast)


# print the error metricsvalues

print(f"Mean absolute error: {lstm_mae} ")

print(f"Mean absolute percentage Error: {lstm_mape} ")

print(f"Mean squared Error: {lstm_mse} ")

print(f"Root mean square error: {lstm_rmse} ")


"""# **ARIMA Feature extraction**"""


# Identify the key features and targeted feature

# These features are used to train the model
```

```
X_stock_key_features = ['SP', 'DAX', 'NIKKEI', 'BOVESPA']

# Target feature used for forecasting the future

Y_stock_target = ['ISE']



# split the extracted feature data into training and testing

x_train, x_test, y_train, y_test = train_test_split_data(stock_feature_extracted_data,
X_stock_key_features, Y_stock_target)



print(len(x_train))



print(y_train.shape)

print(y_test.shape)



# Function used to normalize the features and target variables

norm_x_train, norm_x_test, norm_y_train, norm_y_test = normalize_data(x_train, x_test,
y_train, y_test)



print(norm_x_train.shape)

print(norm_y_train.shape)

print(norm_x_test.shape)

print(norm_y_test.shape)



# convert the norm_y_train and norm_y_test to 1D array through flatten

norm_y_train_flat = norm_y_train.flatten()

norm_y_test_flat = norm_y_test.flatten()
```

```
# Build ARIMA model,

# Find the best order for AR(P), I(q), M(d) parameters using auto_arima,

# set the seasonality = True, to get the best sasonality order.


arima_auto_model = auto_arima(norm_y_train_flat, exogenous=norm_x_train,
seasonal=False,

                        trace=True, error_action='ignore',

                        suppress_warnings=True, stepwise=True)



# Identify the best order through auto_arima

arima_best_order = arima_auto_model.order

arima_best_order


# Use the SARIMAX model using the optimal order to train

# SARIMAX use both features and target for training the data unlike ARIMA it accepts only
target data

arima_model = SARIMAX(norm_y_train_flat, exog=norm_x_train, order=arima_best_order)

arima_model = arima_model.fit(disp=False)


# summarize the arima_model

arima_model.summary()


# use the model to predict on the training data

armia_predict = arima_model.predict(start=0, end=len(norm_y_train_flat)-1,

                            exog=norm_x_train)
```

```
armia_predict.shape


# Forecast the model on the test data (target)
arima_forecast = arima_model.forecast(steps=len(norm_y_test_flat), exog=norm_x_test)
arima_forecast.shape


# visualize the data
# set the figure size
plt.figure(figsize=(10,6))
# plot the training data, predicted and Forecast data
plt.plot(norm_y_train_flat, label='Training data', color='g')
plt.plot(armia_predict, label='Predicted data', color='red')
fore_start_index = len(norm_y_train_flat) + np.arange(len(arima_forecast))
plt.plot(fore_start_index, arima_forecast, label='Forecast data', color='black')
# Add the title and labels for the graphs
# legend to differentiate between the lines and graph
plt.title('Comparision of Training, Predicted and Forecast data (After Feature Extraction)')
plt.xlabel('Time Step')
plt.ylabel('Normalized Vlaues')
plt.legend()
# show the plot
plt.show()


# Function used to calculate the error metrics
lstm_mae, lstm_mape, lstm_mse, lstm_rmse = calculate_error_rate(norm_y_test,
arima_forecast)
```

```
# print theerror metrics values
print(f"Mean absolute error: {lstm_mae} ")
print(f"Mean absolute percentage Error: {lstm_mape} ")
print(f"Mean squared Error: {lstm_mse} ")
print(f"Root mean square error: {lstm_rmse} ")

"""# **Implementing Prophet model**"""

# Install the prophet library created by Facebook
# !pip install prophet

# Import the necessary library
from prophet import Prophet

# reset the index to the  stock dataset
fbp_data = stock_data.reset_index()
fbp_data.head()

# check the date column is in datetime format
# rename the columns as date as 'ds' and ISE as 'y' (target column)
fbp_data['ds'] = pd.to_datetime(fbp_data['date'])
fbp_data = fbp_data.rename(columns={'ISE':'y'})
fbp_data.head()
```

```
# split the data into training and testing for prophet model

# assign the 80% for testing and 20% for testing

training_size = int(len(fbp_data)*0.8)

fbp_train = fbp_data[:training_size]

fbp_test = fbp_data[training_size:]

print(fbp_train.shape)

print(fbp_test.shape)


# create prophet model

prophet_model = Prophet(

    seasonality_mode='multiplicative',  # Ensure that multiplicative seasonality is applied

    changepoint_prior_scale=2.0,  # Maximum flexibility to capture trend changes

    seasonality_prior_scale=20.0  # Allow more variability in seasonality

)


# fit the model

prophet_model.fit(fbp_train)


# get the future values in the dataframe

fbp_model_future = prophet_model.make_future_dataframe(periods=len(fbp_test), freq='D')

# Add the regressor columns to the model_future dataframe

fbp_model_future = pd.concat([fbp_model_future, fbp_data.drop(['ds', 'y'], axis=1)],
axis=1).iloc[-len(fbp_model_future):]

# predict the vales

model_predict = prophet_model.predict(fbp_model_future)
```

```
print(fbp_test.shape)

print(fbp_model_future.shape)

print(model_predict.shape)


# check the forecast values for the test data set

forecast_test_data = model_predict[model_predict['ds'] >= fbp_test['ds'].min()]

forecast_test_data.shape


# visualize the data

plt.figure(figsize=(10,6))


# plot the actual data

plt.plot(fbp_data['ds'], fbp_data['y'], label='Original data', color='green', linewidth=2)


# plot the forecast data for test dataset

plt.plot(forecast_test_data['ds'], forecast_test_data['yhat'], label='Forecast data', color='blue',
linewidth=2)


# Fill the area between the forecast bounds

plt.fill_between(forecast_test_data['ds'], forecast_test_data['yhat_lower'],
forecast_test_data['yhat_upper'], color='gray', alpha=0.2)


# customize the plot

plt.legend()

plt.xlabel('Date')

plt.ylabel('ISE')
```

```
plt.title('Actual vs Forecast')


# show the plot
plt.show()


# Calculate the error matrics


# Merge the forecasted values with the actual data
predicted_data = fbp_test.copy()
predicted_data = predicted_data.merge(forecast_test_data[['ds', 'yhat']], on='ds')


# calculate the MAE
fbp_mae = mean_absolute_error(predicted_data['y'], predicted_data['yhat'])


# claculate the RMSE
fbp_rmse = np.sqrt(mean_squared_error(predicted_data['y'], predicted_data['yhat']))


print(f"Mean Absolute Error (MAE):{fbp_mae}")
print(f"Root Mean sqaure Error (RMSE):{fbp_rmse}")
```