FOREIGN TRADE UNIVERSITY – HCMC CAMPUS

FINAL ASSIGNMENT

COURSE: **AI IN THE ERA OF DIGITAL TRANSFORMATION**

Date: 02/01/2024 – Class Code: ML306, 704 – Course Code: **AIDE300**

No.	Full Name	ID	Peer Evaluation (0%-100%)
1	Ly Nguyen Thien Bao	2011156053	100%
2	Phan Nguyen Thien Trang	2011155631	100%

Grade (in numbers)	**Grade (in words)**
Examiner 1 **Dr. Lê Trung Thành**	**Examiner 2** **Dr. Nguyễn Thị Hoàng Anh**

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Stock Definition & Characteristics of Stock Market

1.1.1 Stock Definition

A stock, also known as equity, is a security that represents the ownership of a fraction of the issuing corporation. Units of stock are called "shares" which entitles the owner to a proportion of the corporation's assets and profits equal to how much stock they own.

Stocks are bought and sold predominantly on stock exchanges and are the foundation of many individual investors' portfolios. Stock trades have to conform to government regulations meant to protect investors from fraudulent practices (Teweles et al., 1998).

Stock price definition is a fundamental concept in finance, representing the market value at which shares of a company are bought and sold. Stock prices are influenced by a multitude of factors including company performance, market conditions, investor sentiment, and macroeconomic indicators (Kumar & Kumar, 2021; Chan, Lam, & Wong, 2021; Mishra & Nalini, 2022).

1.1.2 Characteristics of Stock Market

Understanding the characteristics of the stock market is essential for effective stock price prediction and investment decision-making. Market efficiency remains a key characteristic, with subsequent research exploring deviations from perfect efficiency, such as behavioral biases and market anomalies (Choijil et al.,2022). Market volatility is another critical characteristic, influenced by factors such as economic conditions, geopolitical events, and market sentiment. Volatility can be measured through indicators like the CBOE Volatility Index, which reflects expected market volatility (Liu, Min, et al., 2023). Liquidity is also a significant characteristic, impacting the ease of trading and price impact. High liquidity enhances market efficiency and reduces transaction costs. Additionally, the stock market exhibits long-term trends, with the historical outperformance of equities compared to other asset classes being an important consideration for investors.

1.1.3 Stock Market in Vietnam

The stock market in Vietnam has witnessed notable growth and development over the years, reflecting the country's evolving economic landscape. With the introduction of the Ho Chi Minh Stock Exchange (HOSE) and the Hanoi Stock Exchange (HNX), Vietnam has established a platform for companies to raise capital and investors to participate in the market (Tran, V., 2022). According to The Vietnamese stock market exhibits several unique characteristics that distinguish it from other markets. As a relatively young market, established in the early 2000s, it offers a dynamic and evolving environment. Retail investors play a significant role, contributing to market liquidity and volatility. Foreign investor participation has been increasing, although government-imposed ownership limits can impact their involvement. Government policies and economic reforms influence the market's development and investor sentiment. Additionally, the market is susceptible to the influence of geopolitical events and global economic conditions. These distinctive features, including its youthfulness, retail investor dominance, foreign investor limitations, policy impact, and vulnerability to external factors, contribute to the Vietnamese stock market's distinct character and make it an intriguing subject for research and analysis.

1.1.4 Stock Price Prediction

The evolution of stock price forecasting has followed a similar path. Historically, analysts have used fundamental analysis to identify undervalued or overvalued stocks. This approach involves examining a company's financial statements to determine its intrinsic value (Lee et al., 1987). However, in recent years, more sophisticated techniques have emerged that utilize big data analytics and machine learning algorithms to identify trends and predict future stock prices (Kelleher et al., 2018).

To create more accurate forecasting models, researchers are now combining traditional financial analysis with artificial intelligent techniques (Yoo et al., 2005). By doing so, they can identify patterns in the data that are not immediately apparent to human analysts.

In conclusion, the evolution of stock price forecasting has been driven by advances in technology and a desire for more accurate predictions. By combining traditional financial analysis with advanced techniques and incorporating new data sources, analysts are creating more sophisticated forecasting models that can identify patterns and predict future stock prices with greater accuracy.

1.2 Project Objectives and Subjects

1.2.1 Project Objectives

The accurate forecasting of future stock prices holds significant importance in the financial sector, specifically for investors, shareholders, and regulators. By leveraging data analysis, these stakeholders can gain valuable insights into the market, enabling them to make informed decisions in addition to relying on their domain expertise. Therefore, the primary objective of this project is to predict stock prices of FPT companies by comparing and evaluating between traditional econometric models and Artificial Intelligence ones.

1.2.2 Project subjects:

This project utilizes stock price data for FPT Corporation that was downloaded from Investing.com. The dataset covers approximately 17 years of daily closing prices for FPT Corporation stock, ranging from December 14, 2006 to December 26, 2023. This historical stock price data has been uploaded to GitHub to serve as the dataset for the project. Specifically, the previous closing prices for FPT Corporation are being used to generate forecasts for the closing stock price over the most recent 30-day period.

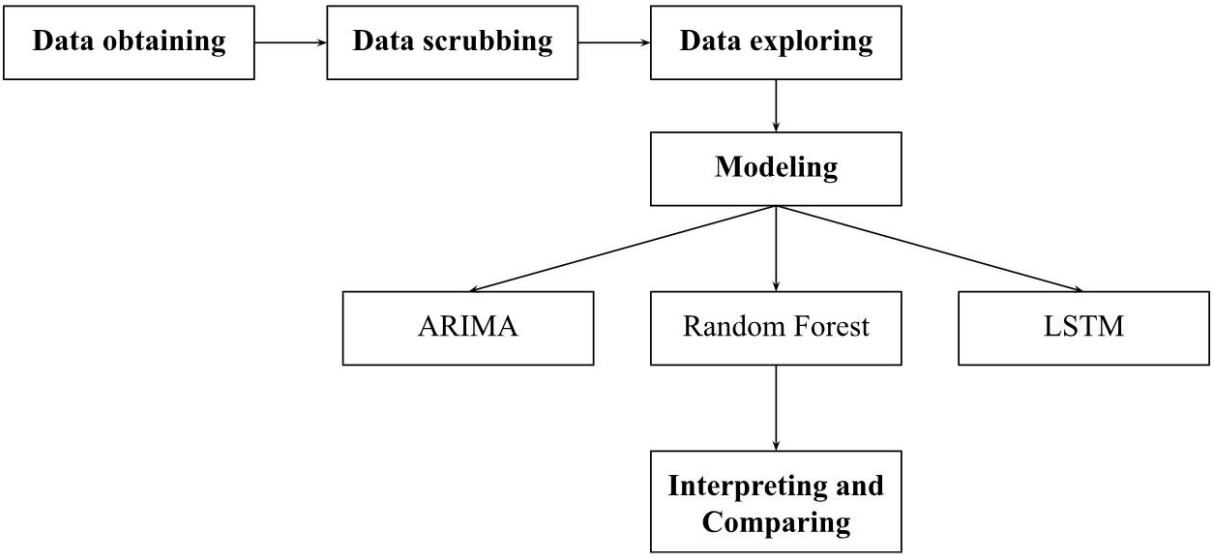
1.2.3 Project methodology:

The project follows a specific methodology. First, the data is subjected to several preprocessing steps to generate a clean dataset. Then, an analysis of the data is performed to uncover descriptive statistics, identify trends over time, examine the data distribution, and create a correlation plot. Subsequently, three different methods are utilized to forecast the stock price: ARIMA, Random Forest, and LSTM. The results obtained from these models are interpreted and compared using selected performance

metrics. Overall, the project involves data preprocessing, data exploration, model selection, prediction, result interpretation, and model comparison.

three machine learning methods often used for financial time series prediction including Random Forest, Recurrent Neural Network and Long Short-Term Memory will be tested.

Figure 1.1: Project Methodology



1.3 Related Concepts

1.3.1 ARIMA

Autoregressive integrated moving average (ARIMA), popularly known as the Box–Jenkins methodology (G. P. E. Box and G. M. Jenkins, 2015). The methods used in this study focus on analyzing the probabilistic or stochastic characteristics of economic time series independently, following the principle of allowing the data to speak for themselves. The ARIMA models enable Yt to be explained by its own past values and stochastic errors. As a result, these ARIMA models are sometimes referred to as theoretic models because they are not derived from any specific economic theory (Gujarati, D. N., & Porter, 2009). According to Gujarati, D. N., & Porter (2009).

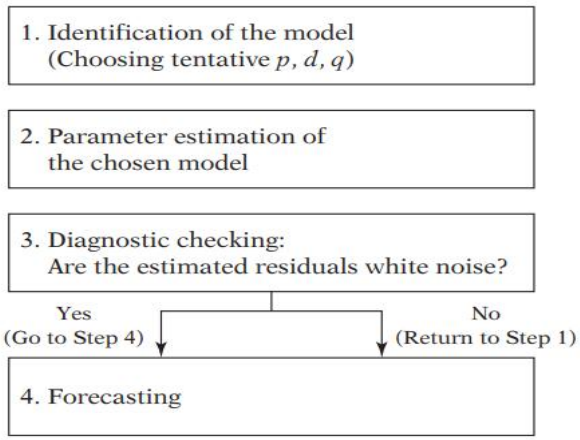
+ An autoregressive (AR) process: a stochastic model used to describe a time series data where the current value of the series is linearly dependent on its past values and a random error term. In an AR process, the value of the variable at

time t (denoted as Yt) is expressed as a linear combination of its lagged values, typically up to a certain order p, along with a random error term.

+ A Moving Average (MA) Process: is a stochastic model commonly used to describe a time series data where the current value of the series is linearly dependent on the past values of the random error terms. In an MA process, the value of the variable at time t (denoted as Yt) is expressed as a linear combination of the past error terms, typically up to a certain order q.
+ A considerable number of economic time series exhibit nonstationary behavior, meaning they lack a constant mean or exhibit trends over time. In order to make such a time series stationary, it is often necessary to differentiate it a certain number of times. Once the differencing is performed, an autoregressive moving average (ARMA) model can be applied to the transformed series. When referring to the original time series, it is then denoted as an autoregressive integrated moving average (ARIMA) time series, represented as ARIMA(p, d, q), where p denotes the number of autoregressive terms, d the number of times the series has to be differenced before it becomes stationary, and q the number of moving average terms.

An ARIMA (Autoregressive Integrated Moving Average) process is a time series model that combines autoregressive (AR), differencing (I), and moving average (MA) components. To determine the appropriate values of the parameters p, d, and q for an ARIMA process, the Box-Jenkins (BJ) methodology is widely used (Gujarati, D. N., & Porter, 2009). The BJ methodology involves four main steps, including identification, estimation, diagnostic checking and forecasting. In our research, the BJ methodology will be used for ARIMA.

Figure 1.2: Box-Jenkins Methodology



Source: Gujarati, D. N., & Porter, D. C. (2009)

1.3.2 Random Forest

Random Forest, introduced by Leo Breiman in 2001, is a powerful and popular algorithm in the field of machine learning and data science. It stands as an ensemble learning method that combines the predictions of multiple decision trees to make more accurate and robust predictions. This algorithm overcomes the limitations of individual decision trees by creating a diverse collection of trees.

The name "Random Forest" derives from its fundamental elements: randomness in data sampling and feature selection. Through bootstrapping, subsets of the original data are randomly selected to train each decision tree in the Random Forest. This random sampling reduces the model's sensitivity to specific instances in the training data, leading to improved generalization and lower variance (Panhalkar et al., 2022).

Random Forest also employs random feature selection during training. Instead of using all features for each tree, a random subset of features is chosen. This feature selection randomness enhances the diversity among the trees and mitigates the issue of feature correlation. Consequently, each decision tree focuses on different aspects of the data, contributing to a more comprehensive and robust prediction when their outputs are combined.

The strength of the Random Forest algorithm lies in its ability to effectively handle both classification and regression tasks. Its accuracy, versatility, and resistance

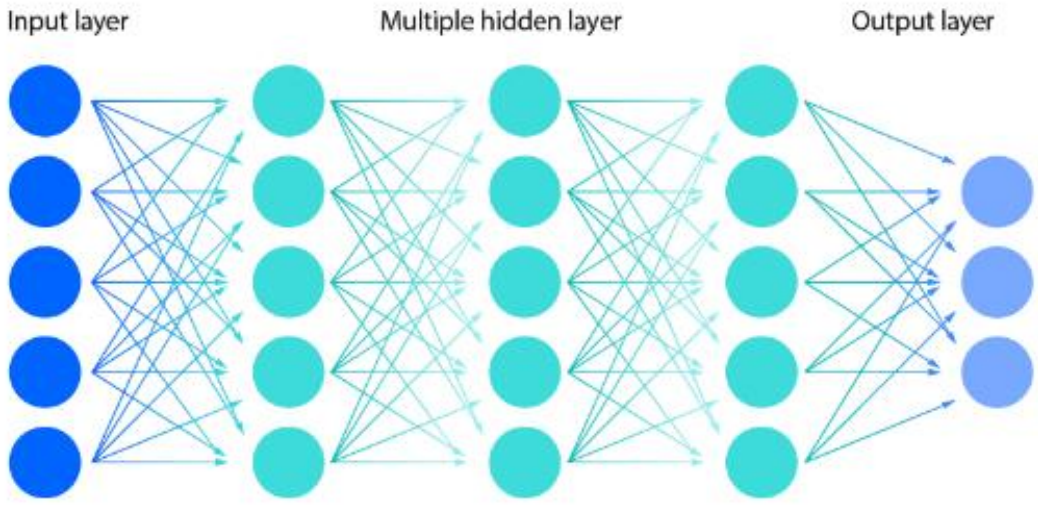
to overfitting have made it widely adopted in various domains such as finance, healthcare, and image recognition.

1.3.3 Neural Network

Neural Networks are the foundation of deep learning. They consist of interconnected nodes, called artificial neurons processing information, organized into layers. Each neuron receives input signals, performs a computation, and produces an output.

At the core of a neural network are interconnected layers of neurons that process information. The network consists of an input layer, an output layer, and hidden layers that handle the bulk of the computations. Each neuron serves as a processing unit, and the connections between neurons are assigned numerical values called weights (Thippanna et al., 2023).

Figure 1.3: The Net of Artificial Neural Network



Source: IBM, 2020

To train a neural network, a process called Gradient Descent is employed. This optimization algorithm iteratively adjusts the weights and biases of the network to minimize the error or loss. By calculating the gradient of the error function and updating the weights and biases in the direction of the steepest descent, the network gradually converges towards a configuration that yields more accurate predictions.

In the output layer of a neural network used for multi-class classification tasks, the Softmax function is often employed. This function normalizes a vector of inputs into a probability distribution over the classes. By exponentiating each input value and dividing it by the sum of the exponentiated values, the Softmax function ensures that the output values range between 0 and 1 and sum up to 1. This enables the network to provide probabilities for each class, indicating the likelihood of the input belonging to each class.

The activation functions used in the hidden layers play a crucial role in introducing nonlinearity to the neural network. One commonly used activation function is the Tanh (hyperbolic tangent) function. It maps the input values to a range between -1 and 1, enabling the network to learn complex relationships in the data. The Tanh function is symmetric around the origin, aiding in centering the data and avoiding biases towards positive or negative values.

Neural networks learn and improve their predictions through the process of forward propagation, where data is passed through the network, and backpropagation, where the error is propagated backward to adjust the weights. In the backpropagation process, the error signal computed at the output layer is propagated backward through the network, providing feedback to each layer and allowing them to adjust their weights. This propagation of the error signal is achieved by utilizing the chain rule of calculus, which facilitates the computation of the gradient of the error function with respect to the weights of each layer.

Overall, neural networks leverage Gradient Descent, the Softmax function, and activation functions like Tanh to recognize patterns, make accurate predictions, and tackle complex tasks. Through forward propagation and backpropagation, neural networks interactively learn from data and improve their predictive capabilities.

1.3.3.1 Recurrent Neural Network

Recurrent Neural Networks (RNNs) are a type of neural network that excel at modeling and processing sequential data by utilizing feedback loops. These feedback loops allow information to be transmitted from one step of the sequence to the next,

enabling RNNs to consider previous inputs and context when making predictions. In an RNN, each step of the sequence corresponds to a time step, and the input at each time step is processed by the RNN. The output is generated based on the current input and the information from previous time steps that is retained in the feedback loops. The weights and biases in an RNN are shared across all time steps, allowing the network to capture and learn the temporal dependencies in the data (Hu et al., 2021).

The advantage of using RNNs is their ability to incorporate contextual information during training, which makes them well-suited for tasks involving stocks and Forex (Bormpotsis et al., 2023). The fluctuations in these markets at a particular time often contain connections to previous trends, and RNNs can effectively capture these temporal dependencies.

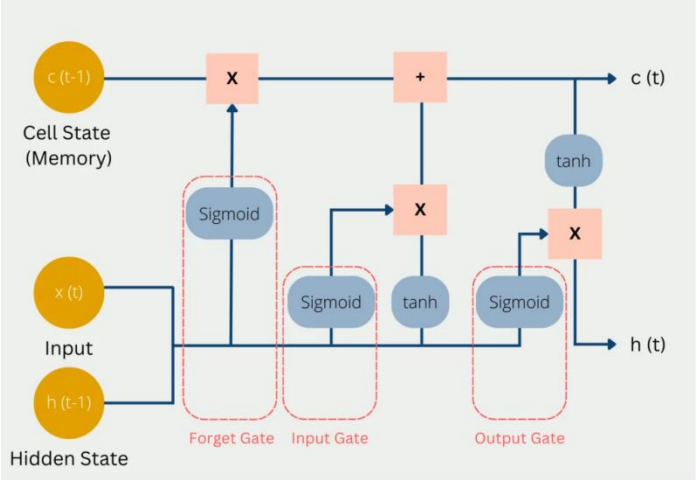
However, there is a main challenge with RNNs, which is exploding/vanishing gradients. Gradients can become extremely small or large as they are backpropagated over multiple time steps, leading to difficulties in learning long-term dependencies (Grosse, 2017).

1.3.3.2 Long Short Term Memory

LSTM (Long Short-Term Memory) is a type of model or structure designed for sequential data, addressing the vanishing/exploding gradient problem in recurrent neural networks (RNNs)(Lipton et al., 2015). LSTM enables long-term affiliation learning, which was not possible with traditional RNNs.

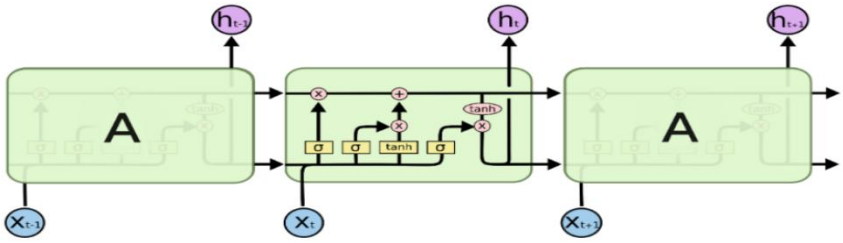
The key feature of LSTM is its ability to capture long-term dependencies in time series data. Unlike RNNs, which struggle with learning long-term affiliations, LSTMs excel at properly learning such relationships. They achieve this by replacing the mid-layer of the RNN with an LSTM block, which consists of recurrent subnetworks called memory blocks. Each memory block contains autoregressive memory cells and units for input, output, and forgetting, enabling continuous writing, reading, and regulation of cell functions (Seddik et al., 2023).

Figure 1.4: LSTM cell



Source: Data Base Camp (2022)

Figure 1.5: The Repeating Module in LSTM



Source: S. Saini and V. Sahula, 2018

The hidden state represents short term memories. The cell state in the given figure corresponds to the long-term memory, which remains unchanged by weights or biases. This characteristic enables the long-term memory to persist over multiple consecutive units without causing gradients to explode or vanish. During each computational iteration, the current input x(t), the previous state of the short-term memory c(t-1), and the previous state of the hidden state h(t-1) are all used. According to Bishop, C. M., and Bishop, H. (2023) in the book Deep Learning: Foundations and Concepts, they stated that there are primarily three stages through which the short-term memory and hidden state pass on their way to generating a new Cell State and Hidden State.

+ In the initial stage, the information undergoes a process known as the **Forget gate**. This step involves determining which current and previous information

should be retained and which should be discarded. It encompasses both the hidden state from the previous iteration and the current input. To make this decision, these values are input into a sigmoid function, which restricts the output to a range between 0 and 1. A value of 0 indicates that the previous information can be disregarded because there may be new and more significant information available. Conversely, a value of 1 signifies that the previous information should be preserved. The outcomes of this operation are then multiplied with the current cell state. Consequently, any knowledge that is deemed unnecessary and assigned a value of 0 is effectively forgotten or eliminated from further consideration.

+ In the second stage, the information undergoes the **Input Gate** process where the importance of the current input in solving the task is determined. This involves multiplying the current input by the hidden state and the weight matrix from the previous iteration. The information deemed important by the Input Gate is then added to the cell state, resulting in the formation of the new cell state, denoted as c(t). This updated cell state now represents the current state of the long-term memory and will be utilized in the subsequent iteration.
+ In the last stage, **Output Gate**, the LSTM model calculates the output in the form of the Hidden State. This output can vary depending on the specific application, such as a word that adds meaning to a sentence. In this stage, the sigmoid function determines which information should pass through the output gate. Subsequently, the cell state is activated using the tanh function and multiplied by the output gate's activation result. This multiplication facilitates the generation of the final Hidden State output, which encapsulates the relevant information for the given task or application.

LSTM has emerged as a highly effective tool for modeling sequential data. Its ability to capture long-term dependencies within the data has significantly improved predictions in numerous domains, including natural language processing, speech recognition, and time series forecasting (Yu et al., 2019). Additionally, there are

various types of LSTM that have been developed, each with its own unique characteristics and applications. Some of these variations include stacked LSTMs, encoder-decoder LSTMs, bidirectional LSTMs, CNN LSTMs, generative LSTMs, etc. (Yadav, A., Jha, C. K., & Sharan, A., 2020). These diverse types of LSTMs provide flexibility in modeling sequential data, allowing for tailored approaches to different tasks and data structures.

##**CHAPTER 2: EXPERIMENTAL RESULTS**

2.1 FPT CORPORATION OVERVIEW

First, the 'vnstock' package will be upgraded to the most recent version through the execution of the command '!pip install -U vnstock'. Subsequently, all modules and functions will be imported from the 'vnstock' package using the command 'from vnstock import *'. This process facilitated the utilization of the package's advanced features and functionalities for the implementation of the report.

```
#Install "vnstock" package
!pip install -U vnstock

Requirement already satisfied: vnstock in
/usr/local/lib/python3.10/dist-packages (0.2.8.7)

#Import all modules and functions
from vnstock import *
```

Next, data pertaining to FPT Corporation's overview, shareholders, subsidiaries, governance, financial ratios, etc will be collected as follow. The retrieved data will also be analyzed to provide valuable insights.

###Company overview

```
#Retrieve an overview of FPT Coporation
company_overview('FPT')

  ticker exchange             industry companyType  noShareholders  \
0    FPT     HOSE  Công nghệ Thông tin          CT           48598

   foreignPercent  outstandingShare  issueShare establishedYear
noEmployees  \
0            0.49            1270.0      1270.0            2002
45903

   stockRating  deltaInWeek  deltaInMonth  deltaInYear shortName
industryEn  \
0          3.1       -0.009         0.023        0.362  FPT Corp
Technology

   industryID industryIDv2                 website
0         310         9537  https://www.fpt.com.vn
```

Since its establishment in 2002, FPT Corporation has diversified its operations across various sectors, including software development, IT services, telecommunications, and digital transformation solutions. With 48,598 shareholders, its shares are traded under the ticker symbol "FPT." FPT Corporation boasts a workforce of 45,903 employees.

###Company large shareholder

```
#Retrieve and display the information about 5 largest shareholders of
FPT Corporation
company_large_shareholders (symbol='FPT').head()
```

```
  ticker                                          shareHolder
shareOwnPercent
0    FPT                                    Trương Gia Bình
0.0699
1    FPT  Tổng Công ty Đầu Tư Và Kinh Doanh Vốn Nhà Nước...
0.0579
2    FPT                                     Bùi Quang Ngọc
0.0164
3    FPT                              Trương Thị Thanh Thanh
0.0150
4    FPT                                    Hoàng Minh Châu
0.0141
```

The company's five largest shareholders are Truong Gia Binh, the State Capital Investment Corporation, Bui Quang Ngoc, Truong Thi Thanh Thanh, and Hoang Minh Chau.

###Subsidiaries company

```
#Retrieve and display the information about 5 largest subsidiaries of
FPT Coporation
company_subsidiaries_listing (symbol='CTS', page_size=100,
page=0).head()
```

```
  ticker                                  subCompanyName
subOwnPercent
0    CTS                      Công ty Cổ phần Kiên Hùng
0.109
1    CTS                   Quỹ Đầu Tư Năng Động Việt Nam
0.062
2    CTS   Công ty Cổ phần Phát triển Nhà Bà Rịa Vũng Tàu
0.038
3    CTS           Công ty Cổ phần Than Hà Tu - Vinacomin
0.032
4    CTS  Công ty Cổ phần Chế biến Hàng Xuất khẩu Long An
0.018
```

Among its subsidiaries are Kien Hung Joint Stock Company, Vietnam Dynamic Investment Fund, Ba Ria Vung Tau Development Joint Stock Company, Ha Tu Coal Joint Stock Company - Vinacomin, and Long An Export Processing Joint Stock Company.

###Company governance

```
#Retrieve and display members of FPT Corporation's governance
company_officers (symbol='FPT', page_size=20, page=0)
```

	ticker	officerName	officerPosition	officerOwnPercent
0	FPT	Trương Gia Bình	Chủ tịch Hội đồng Quản trị	0.0699
1	FPT	Bùi Quang Ngọc	Phó Chủ tịch Hội đồng Quản trị	0.0164
2	FPT	Trương Thị Thanh Thanh	Phó Chủ tịch Hội đồng Quản trị	0.0150
5	FPT	Hoàng Minh Châu	Phó Tổng Giám đốc	0.0141
4	FPT	Hoàng Minh Châu	Phó Chủ tịch Hội đồng Quản trị	0.0141
6	FPT	Phan Ngô Tống Hưng	Phó Chủ tịch Hội đồng Quản trị	0.0134
3	FPT	Đỗ˜Cao Bảo	Thành viên Hội đồng Quản trị	0.0095
7	FPT	Nguyễn Thành Nam	None	0.0094
13	FPT	Lê Quang Tiến	Phó Chủ tịch Hội đồng Quản trị	0.0063
11	FPT	Dương Dũng Triều	None	0.0044
8	FPT	Nguyễn Thê´Phương	Phó Tổng Giám đốc	0.0033
9	FPT	Nguyễn Thê´Phương	Người phụ trách quản trị công ty	0.0033
10	FPT	Nguyễn Văn Khoa	None	0.0027
16	FPT	Nguyễn Minh Sơn	Thành viên Ban kiểm soát	0.0025
14	FPT	Nguyễn Khắc Thành	None	0.0024
12	FPT	Hoàng Việt Anh	None	0.0023
15	FPT	Nguyễn Khải Hoàn	Thành viên Ban kiểm soát	0.0009
17	FPT	Đỗ˜Thị Ngọc Mai	None	0.0007
18	FPT	Nguyễn Thị Dư	None	0.0007
19	FPT	Đỗ˜Bảo Dương	None	0.0006

As can be seen, the company's governance structure consists one Chairman of the Board of Directors (Mr. Truong Gia Binh) who hold 0.0669 percent of company shares, five vice chairman of the Board of Directors and others who have the critical position in the company.

###Company news

```
#Retrieve and display the news articles related to the company 'FPT',
showing the first few records
company_news (symbol='FPT', page_size=15, page=0).head()
```

```
  ticker  price  priceChange  priceChangeRatio  priceChangeRatio1W  \
0    FPT  94600         -500            -0.005              -0.017
1    FPT  94600         -500            -0.005              -0.017
2    FPT  94000        -2200            -0.023              -0.011
3    FPT  96200            0             0.000               0.021
4    FPT  96200            0             0.000               0.021

   priceChangeRatio1M        id  \
0               0.017  11028681
1               0.017  11028196
2               0.036  11022790
3               0.041  11022533
4               0.041  11021723

                                               title source  \
0      FPT cán mốc 1 tỷ USD từ thị trường nước ngoài   DTCK
1  Ông Hoàng Nam Tiến khoe FPT Software chính thứ...  CafeF
2   FPT ký kết hợp tác với hai doanh nghiệp Nhật Bản   DTCK
3  Báo cáo kết quả phân phối chứng quyền có bảo đ...   HOSE
4  FPT báo lãi sau thuế 11 tháng tăng trưởng 19%,...   NSTT

           publishDate
0  2023-12-22 18:00:00
1  2023-12-22 13:32:00
2  2023-12-18 07:10:00
3  2023-12-15 17:31:00
4  2023-12-15 09:41:00
```

###Dividend history

```
#Retrieve and display the dividend history of the company "FPT",
showing the first few records
dividend_history("FPT").head()
```

```
  exerciseDate  cashYear  cashDividendPercentage issueMethod
0     24/08/23      2023                    0.10        cash
1     05/07/23      2022                    0.10        cash
2     05/07/23      2023                    0.15       share
3     24/08/22      2022                    0.10        cash
4     13/06/22      2021                    0.10        cash
```

###Financial ratio

```
#Retrieve the quarterly financial ratios for the company "FPT",
including industry averages for comparison
financial_ratio("FPT", 'quarterly', True)
```

```
range                  Q3-2023 Q2-2023 Q1-2023 Q4-2022 Q3-2022 Q2-
2022  \
ticker                     FPT     FPT     FPT     FPT     FPT
FPT
quarter                     Q3      Q2      Q1      Q4      Q3
Q2
year                      2023    2023    2023    2022    2022
2022
priceToEarning            20.0    16.0    15.4    15.7    16.6
18.7
priceToBook                5.2     3.8     3.8     3.9     4.4
4.7
valueBeforeEbitda         13.6    14.1    12.4    12.3    13.8
13.0
roe                       0.28   0.266   0.266   0.272   0.284
0.271
roa                      0.104   0.099   0.104   0.101   0.101
0.095
daysReceivable              47      54      51      49      50
53
daysInventory               23      26      26      24      30
31
daysPayable                 32      32      34      41      34
40
ebitOnInterest             8.6     9.0    12.7    12.8    10.6
8.9
earningPerShare           4798    4573    4369    4169    4129
3871
bookValuePerShare        18609   19016   17786   16561   15665
15401
equityOnTotalAsset       0.381   0.399   0.445   0.407   0.361
0.347
equityOnLiability          0.8     0.9     1.1     1.0     0.8
0.7
currentPayment             1.2     1.2     1.3     1.3     1.2
1.2
quickPayment               1.1     1.2     1.2     1.2     1.2
1.1
epsChange                0.049   0.047   0.048    0.01   0.067
0.054
ebitdaOnStock           8610.0  8691.0  9111.0  8575.0  7998.0
7565.0
grossProfitMargin        0.399   0.371   0.391   0.402   0.391
0.381
operatingProfitMargin    0.166   0.154    0.16   0.153   0.158
0.157
```

```
postTaxMargin              0.126   0.121   0.128   0.104    0.13
0.124
debtOnEquity                 0.7     0.7     0.5     0.5     0.8
0.9
debtOnAsset                  0.3     0.3     0.2     0.2     0.3
0.4
debtOnEbitda                 1.8     2.2     1.9     1.9     2.3
2.6
shortOnLongDebt            110.1    90.0    38.5     7.4     7.8
9.2
assetOnEquity                2.6     2.5     2.2     2.5     2.8
2.9
capitalBalance              6482    7471    6481    6400    6572
6744
cashOnEquity               0.252   0.218   0.122   0.254   0.155
0.223
cashOnCapitalize           0.059   0.053   0.033   0.074   0.041
0.067
cashCirculation               38      48      43      31      46
43
revenueOnWorkCapital         7.8     6.8     7.1     7.5     7.4
6.9
capexOnFixedAsset         -0.348  -0.342  -0.295  -0.284  -0.286  -
0.284
revenueOnAsset               0.9     0.8     0.9     0.8     0.8
0.8
postTaxOnPreTax              0.7     0.7     0.7     0.7     0.7
0.7
ebitOnRevenue              0.166   0.154    0.16   0.153   0.158
0.157
preTaxOnEbit                 1.1     1.2     1.1     1.0     1.1
1.2
payableOnEquity              1.2     1.1     0.9     1.0     1.3
1.4
ebitdaOnStockChange       -0.009  -0.046   0.063   0.072   0.057  -
0.139
bookValuePerShareChange   -0.021   0.069   0.074   0.057   0.017
0.02

range                   Q1-2022 Q4-2021 Q3-2021 Q2-2021  ...  Q1-2010
Q4-2009  \
ticker                      FPT     FPT     FPT     FPT  ...      FPT
FPT
quarter                      Q1      Q4      Q3      Q2  ...       Q1
Q4
year                       2022    2021    2021    2021  ...     2010
2009
priceToEarning             20.2    18.9    24.0    23.6  ...      NaN
NaN
priceToBook                 4.9     4.6     5.7     5.4  ...      NaN
```

```
NaN
valueBeforeEbitda          15.5    16.5    13.9    15.7  ...       7.8
NaN
roe                        0.26   0.257   0.251   0.241  ...     0.385
0.387
roa                       0.093   0.091   0.093   0.092  ...     0.138
0.127
daysReceivable               55      44      47      49  ...        13
16
daysInventory                28      23      29      29  ...        18
15
daysPayable                  39      47      35      45  ...        11
13
ebitOnInterest              9.7    10.6    11.5    12.0  ...       6.1
8.0
earningPerShare            3673    3411    3160    3006  ...      1249
1171
bookValuePerShare         15105   14128   13373   13218  ...      2742
2538
equityOnTotalAsset        0.345   0.334   0.352   0.357  ...     0.239
0.216
equityOnLiability           0.7     0.7     0.7     0.7  ...       0.5
0.4
currentPayment              1.2     1.2     1.2     1.2  ...       1.7
1.7
quickPayment                1.2     1.1     1.1     1.1  ...       1.3
1.4
epsChange                 0.077   0.079   0.051   0.033  ...     0.066
-0.037
ebitdaOnStock            8792.0  8344.0  8020.0  7755.0  ...   13862.0
NaN
grossProfitMargin         0.406   0.383   0.373   0.398  ...     0.126
0.09
operatingProfitMargin     0.145   0.137   0.158   0.173  ...     0.056
0.035
postTaxMargin             0.127   0.122   0.129   0.116  ...     0.036
0.019
debtOnEquity                1.0     0.9     0.9     0.9  ...       1.2
1.3
debtOnAsset                 0.4     0.4     0.4     0.4  ...       0.4
0.4
debtOnEbitda                2.6     2.3     2.1     2.1  ...       1.2
NaN
shortOnLongDebt             9.2     7.8     8.8     8.5  ...       1.1
1.0
assetOnEquity               2.9     3.0     2.8     2.8  ...       4.2
4.6
capitalBalance             6480    5305    4764    4645  ...      3251
3284
cashOnEquity              0.21   0.253   0.268   0.248  ...     0.406
```

```
0.758
cashOnCapitalize          0.052   0.054   0.069   0.057  ...    0.104
0.206
cashCirculation              44      20      41      33  ...       20
17
revenueOnWorkCapital        6.6     8.4     7.7     7.5  ...     27.8
23.1
capexOnFixedAsset        -0.294  -0.311  -0.413  -0.397  ...    -0.02
NaN
revenueOnAsset               0.8     0.7     0.8     0.8  ...      4.9
4.7
postTaxOnPreTax              0.7     0.7     0.7     0.6  ...      1.6
1.5
ebitOnRevenue            0.145   0.137   0.158   0.173  ...    0.056
0.035
preTaxOnEbit                 1.3     1.2     1.2     1.0  ...      1.0
0.8
payableOnEquity              1.4     1.5     1.4     1.3  ...      1.9
2.2
ebitdaOnStockChange        0.054    0.04   0.034  -0.076  ...      NaN
NaN
bookValuePerShareChange    0.069   0.056   0.012   0.009  ...     0.08
0.04

range                   Q3-2009 Q2-2009 Q1-2009 Q4-2008 Q3-2008 Q2-
2008  \
ticker                      FPT     FPT     FPT     FPT     FPT
FPT
quarter                      Q3      Q2      Q1      Q4      Q3
Q2
year                       2009    2009    2009    2008    2008
2008
priceToEarning              NaN     NaN     NaN     NaN     NaN
NaN
priceToBook                 NaN     NaN     NaN     NaN     NaN
NaN
valueBeforeEbitda           NaN     NaN     NaN     NaN     NaN
NaN
roe                        0.42   0.369   0.368   0.379   0.369
0.36
roa                       0.162    0.14    0.14   0.146   0.156
0.145
daysReceivable               13      15      16      16      12
15
daysInventory                21      21      18      16      16
16
daysPayable                  15      13      13      12      11
9
ebitOnInterest             31.5    26.2    32.1    37.0    15.4
17.7
```

earningPerShare	1216	994	960	924	860	824
bookValuePerShare	2441	2211	2187	2111	1906	1928
equityOnTotalAsset	0.297	0.313	0.333	0.312	0.279	0.27
equityOnLiability	0.8	0.9	0.9	0.8	0.7	0.6
currentPayment	1.6	1.6	1.7	1.5	1.4	1.4
quickPayment	1.1	1.0	1.1	1.1	0.9	0.9
epsChange	0.224	0.035	0.04	0.074	0.044	0.369
ebitdaOnStock	NaN	NaN	NaN	NaN	NaN	NaN
grossProfitMargin	0.125	0.106	0.118	0.111	0.097	0.109
operatingProfitMargin	0.059	0.047	0.057	0.047	0.039	0.056
postTaxMargin	0.038	0.026	0.03	0.036	0.016	0.023
debtOnEquity	0.6	0.5	0.3	0.5	0.5	0.8
debtOnAsset	0.2	0.2	0.1	0.2	0.2	0.3
debtOnEbitda	NaN	NaN	NaN	NaN	NaN	NaN
shortOnLongDebt	10411.2	7049.7	4333.2	6108.7	14.1	23.8
assetOnEquity	3.4	3.2	3.0	3.2	3.6	3.7
capitalBalance	2113	1786	1789	1574	1491	1426
cashOnEquity	0.552	0.354	0.348	0.51	0.51	0.481
cashOnCapitalize	0.13	0.091	0.101	NaN	NaN	NaN
cashCirculation	19	23	20	20	17	22
revenueOnWorkCapital	28.3	24.0	22.7	22.2	30.3	25.0
capexOnFixedAsset	NaN	NaN	NaN	NaN	NaN	NaN
revenueOnAsset	4.9	5.1	5.3	5.9	6.8	6.3
postTaxOnPreTax	1.5	1.7	1.8	1.2	1.8	1.6
ebitOnRevenue	0.059	0.047	0.057	0.047	0.039	0.056

```
preTaxOnEbit                  1.0     1.0     0.9     0.9     0.7
0.7
payableOnEquity               1.2     1.2     1.1     1.3     1.5
1.7
ebitdaOnStockChange           NaN     NaN     NaN     NaN     NaN
NaN
bookValuePerShareChange     0.104   0.011   0.036   0.107  -0.011
0.013

range                    Q1-2008 Q4-2007
ticker                       FPT     FPT
quarter                       Q1      Q4
year                        2008    2007
priceToEarning               NaN     NaN
priceToBook                  NaN     NaN
valueBeforeEbitda            NaN     NaN
roe                        0.285   0.291
roa                        0.112   0.118
daysReceivable                21      26
daysInventory                 19      18
daysPayable                   12      15
ebitOnInterest              13.3    28.0
earningPerShare              602     567
bookValuePerShare           1903    1863
equityOnTotalAsset         0.268   0.316
equityOnLiability            0.6     0.6
currentPayment               1.4     1.5
quickPayment                 1.0     1.0
epsChange                   0.06   0.094
ebitdaOnStock                NaN     NaN
grossProfitMargin          0.082   0.096
operatingProfitMargin      0.035   0.037
postTaxMargin              0.022   0.029
debtOnEquity                 1.1     0.7
debtOnAsset                  0.4     0.2
debtOnEbitda                 NaN     NaN
shortOnLongDebt             34.2    22.9
assetOnEquity                3.7     3.2
capitalBalance              1408    1357
cashOnEquity               0.678   0.452
cashOnCapitalize             NaN     NaN
cashCirculation               28      30
revenueOnWorkCapital        17.5    14.1
capexOnFixedAsset            NaN     NaN
revenueOnAsset               4.9     4.9
postTaxOnPreTax              1.6     1.3
ebitOnRevenue              0.035   0.037
preTaxOnEbit                 1.0     1.1
payableOnEquity              1.8     1.6
```

```
ebitdaOnStockChange          NaN     NaN
bookValuePerShareChange    0.021   0.009

[41 rows x 64 columns]
```

###Financial statements (Income Statement, Balance sheet, Cashflow statement)

```
# Retrieve the quarterly income statement of FPT Corporation
financial_report (symbol='CTS', report_type='IncomeStatement',
frequency='Quarterly')
```

```
                                              CHỈ TIÊU       Q2 2021  \
0                                  Doanh thu hoạt động  2.548297e+11
1   Lãi từ các tài sản tài chính ghi nhận thông qu...  1.431925e+11
2                       Lãi bán các tài sản tài chính  2.760954e+10
3   Chêch lệch tăng đánh giá lại các tài sản tài c...  8.890308e+10
4   Cổ tức, tiền lãi phát sinh từ tài sản tài chín...  2.667987e+10
..                                                ...           ...
78          Thu nhập toàn diện phân bổ cho chủ sở hữu  1.268058e+11
79  Thu nhập toàn diện phân bổ cho cổ đông không k...  0.000000e+00
80              Thu nhập thuần trên cổ phiếu phổ thông  0.000000e+00
81                        Lãi cơ bản trên cổ phiếu   1.192000e+03
82                   Thu nhập pha loãng trên cổ phiếu  0.000000e+00

         Q3 2021       Q4 2021       Q1 2022       Q2 2022       Q3
2022  \
0   2.013442e+11  4.501579e+11  2.264412e+11  3.098045e+11
1.439564e+11
1   9.740064e+10  3.173530e+11  9.795139e+10  2.029242e+11
4.727474e+10
2   4.622704e+09  1.484420e+11  6.288634e+10  1.590521e+10
6.279546e+09
3   8.092503e+10  1.525241e+11  5.985517e+09  1.603170e+11
1.366435e+10
4   1.185291e+10  1.638683e+10  2.907953e+10  2.670194e+10
2.733085e+10
..           ...           ...           ...           ...           .
..
78  3.001149e+10  1.897897e+11  5.760451e+10 -3.989289e+09
1.161657e+10
79  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
0.000000e+00
80  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
0.000000e+00
81  2.820000e+02  1.784000e+03  5.410000e+02 -9.000000e+00
1.150000e+02
82  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
0.000000e+00
```

```
         Q4 2022       Q1 2023       Q2 2023       Q3 2023
0   1.919530e+11  1.990163e+11  3.787698e+11  2.557544e+11
1   9.548191e+10  8.764977e+10  2.257481e+11  8.436731e+10
2   3.068641e+09  4.145925e+10  1.520655e+11  1.452073e+10
3   5.693829e+10  3.860060e+10  6.323989e+10  5.649124e+10
4   3.547498e+10  7.589912e+09  1.044273e+10  1.335534e+10
..           ...           ...           ...           ...
78 -5.311273e+09  1.497126e+10  8.342798e+10  6.768349e+10
79  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
80  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
81 -2.200000e+01  7.200000e+01  5.150000e+02  4.550000e+02
82  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00

[83 rows x 11 columns]
```

```
# Retrieve the quarterly balance sheet of FPT Corporation
financial_report (symbol='CTS', report_type='BalanceSheet',
frequency='Quarterly')
```

```
                                       CHỈ TIÊU       Q2 2021        Q3
2021  \
0                              TỔNG CỘNG TÀI SẢN  5.145502e+12
5.517816e+12
1                               TÀI SẢN NGẮN HẠN  5.065915e+12
5.440394e+12
2                     Tài sản tài chính ngắn hạn  4.704200e+12
5.010904e+12
3                      Tiền và tương đương tiền   1.432634e+11
4.720617e+10
4                                          Tiền  1.432634e+11
4.720617e+10
..                                           ...           ...
...
155                     Vốn ngân sách nhà nước    0.000000e+00
0.000000e+00
156          Nguồn kinh phí đã hình thành TSCĐ  0.000000e+00
0.000000e+00
157  LỢI ÍCH CỦA CỔ ĐÔNG THIỂU SỐ (trước 2015)  0.000000e+00
0.000000e+00
158                          TỔNG CỘNG NGUỒN VỐN  5.145502e+12
5.517816e+12
159      LỢI NHUẬN ĐÃ PHÂN PHỐI CHO NHÀ ĐẦU TƯ  0.000000e+00
0.000000e+00

          Q4 2021       Q1 2022       Q2 2022       Q3 2022        Q4
2022  \
0    6.685686e+12  7.736635e+12  6.236224e+12  5.673231e+12
5.036970e+12
1    6.610379e+12  7.663157e+12  6.164772e+12  5.595968e+12
4.961836e+12
```

```
2    6.025552e+12  6.340984e+12  5.369165e+12  5.138226e+12
4.950488e+12
3    3.365850e+11  2.657481e+11  1.386745e+11  9.583052e+10
1.263470e+11
4    3.365850e+11  2.657481e+11  1.386745e+11  9.583052e+10
1.263470e+11
..            ...           ...           ...           ...
...
155  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
0.000000e+00
156  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
0.000000e+00
157  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
0.000000e+00
158  6.685686e+12  7.736635e+12  6.236224e+12  5.673231e+12
5.036970e+12
159  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
0.000000e+00

          Q1 2023       Q2 2023       Q3 2023
0    6.406208e+12  7.021398e+12  8.194782e+12
1    6.333871e+12  6.943960e+12  8.104398e+12
2    6.325921e+12  6.686428e+12  7.594308e+12
3    6.950490e+11  3.662940e+11  3.437648e+11
4    6.950490e+11  3.662940e+11  3.437648e+11
..            ...           ...           ...
155  0.000000e+00  0.000000e+00  0.000000e+00
156  0.000000e+00  0.000000e+00  0.000000e+00
157  0.000000e+00  0.000000e+00  0.000000e+00
158  6.406208e+12  7.021398e+12  8.194782e+12
159  0.000000e+00  0.000000e+00  0.000000e+00

[160 rows x 11 columns]
```

```
# Retrieve the quarterly Cashflow statement of FPT Corporation
financial_report (symbol='CTS', report_type='CashFlow',
frequency='Quarterly')
```

```
                                              CHỈ TIÊU       Q2 2021  \
0   Lưu chuyển thuần từ hoạt động kinh doanh chứng...  2.449262e+11
1   Lợi nhuận từ hoạt động kinh doanh trước thay đ...  8.025353e+10
2                                Lợi nhuận trước thuế´ 1.585317e+11
3                           Điều chỉnh cho các khoản -1.418760e+10
4                           Khấu hao tài sản cố´định  2.436895e+09
..                                                ...           ...
93          Tiền và các khoản tương đương tiền đầu kỳ  6.369694e+10
94                Tiền mặt, tiền gửi ngân hàng đầu kỳ  6.369694e+10
95                         Các khoản tương đương tiền  0.000000e+00
96  Ảnh hưởng của thay đổi tỷ giá hối đoán quy đổi...  0.000000e+00
97        Tiền và các khoảng tương đương tiền cuối kỳ  1.432634e+11
```

```
         Q3 2021       Q4 2021       Q1 2022       Q2 2022       Q3
2022  \
0  -5.735797e+11 -5.018765e+11 -3.499498e+11  7.752107e+11
1.911542e+11
1  -3.777142e+10  2.548259e+11  8.700873e+10  3.653995e+10
8.930853e+09
2   3.665286e+10  2.372519e+11  6.857547e+10  3.900688e+09
1.347039e+10
3   6.500752e+09  1.439315e+10 -3.149248e+10 -1.589058e+10 -
2.528020e+10
4   2.144722e+09  2.089398e+09  1.978763e+09  1.927028e+09
2.057271e+09
..           ...           ...           ...           ...           .
..
93  1.432634e+11  4.720617e+10  3.365850e+11  2.657481e+11
1.386745e+11
94  1.432634e+11  4.720617e+10  3.365850e+11  2.657481e+11
1.386745e+11
95  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
0.000000e+00
96  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
0.000000e+00
97  4.720617e+10  3.365850e+11  2.657481e+11  1.386745e+11
9.583052e+10

         Q4 2022       Q1 2023       Q2 2023       Q3 2023
0   2.293434e+11 -8.332713e+11 -5.564554e+11 -9.251566e+11
1  -4.973245e+09  3.069722e+10  1.416858e+11  9.673803e+09
2  -7.152379e+09  1.348757e+10  9.452949e+10  6.474025e+10
3  -9.758044e+09 -1.103872e+10 -2.841097e+10 -1.874096e+10
4   2.338789e+09  2.376210e+09  1.267457e+09  1.058315e+09
..           ...           ...           ...           ...
93  9.583052e+10  1.583470e+11  6.950490e+11  3.712301e+11
94  9.583052e+10  1.263470e+11  6.950490e+11  3.662940e+11
95  0.000000e+00  3.200000e+10  0.000000e+00  0.000000e+00
96  0.000000e+00  0.000000e+00  0.000000e+00  4.936056e+09
97  1.263470e+11  6.950490e+11  3.712301e+11  3.437648e+11

[98 rows x 11 columns]
```

###Stock evaluation

```
#Retrieve and Preview Evaluated Stock Data for FPT Corporation
stock_evaluation (symbol='FPT', period=1, time_window='D').head()
```

```
  ticker   fromDate     toDate    PE   PB  industryPE  vnindexPE
industryPB  \
0    FPT 2023-01-03 2023-01-03  15.6  3.8        16.3       10.9
```

```
2.7
1    FPT 2023-01-04 2023-01-04  15.6  3.8        16.3        11.0
2.7
2    FPT 2023-01-05 2023-01-05  15.7  3.9        16.4        11.1
2.7
3    FPT 2023-01-06 2023-01-06  15.6  3.8        16.3        11.0
2.7
4    FPT 2023-01-09 2023-01-09  15.6  3.8        16.2        11.1
2.6

   vnindexPB
0        1.6
1        1.6
2        1.6
3        1.6
4        1.6
```

The evaluated stock data provides insights into the stock's recent performance. This allows us to quickly assess the initial performance or summary statistics of FPT Corporation's stock.

###Historical price

```
#Retrieve and preview the historical stock data for FPT Corporation.
stock_historical_data(symbol="FPT", start_date="2005-01-01",
end_date="2023-12-01", resolution="1D", type="stock", beautify=True,
decor=False, source='DNSE').head()

         time  open  high   low  close  volume ticker
0  2012-03-20  6950  6950  6900   6900   94310    FPT
1  2012-03-21  6900  6950  6840   6900  145260    FPT
2  2012-03-22  6900  6900  6780   6780   90050    FPT
3  2012-03-23  6840  6840  6780   6840   71320    FPT
4  2012-03-26  6840  6840  6780   6840  112360    FPT
```

By specifying the desired symbol and date range, it allows us to analyze the stock's performance over a specific time frame. Historical stock data of FPT Coporation were retrieved from January 1, 2005, to December 1, 2023, with a daily resolution.

> On the whole, since its establishment in 2002, FPT Corporation has diversified its operations across various sectors, including software development, IT services, telecommunications, and digital transformation solutions. With 48,598 shareholders, its shares are traded under the ticker symbol "FPT." It was listed on HOSE since December 20006. The company's five largest shareholders are Truong Gia Binh, the State Capital Investment Corporation, Bui Quang Ngoc, Truong Thi Thanh Thanh, and Hoang Minh Chau. FPT Corporation boasts a workforce of 45,903 employees. Among its subsidiaries are Kien Hung Joint Stock Company, Vietnam Dynamic Investment Fund, Ba Ria Vung Tau Development Joint Stock Company, Ha Tu Coal Joint Stock Company - Vinacomin, and Long An Export Processing Joint Stock Company. The company's governance structure consists of one chairman of BOD, 5 vice chairman and others people who have important position in the company operation.

Furthermore, the dataset provides comprehensive information on company news, financial ratios, dividend history, financial statements, stock evaluation, and historical prices.

##NOTE: OSEMN Pipeline ****

In adherence to the commonly followed data science pipeline known as "OSEMN," the authors will be proceeding with the outlined steps throughout this analysis.The purpose of each step can be explained as follows:

1. **O**btaining the data plays a pivotal importance as the initial step in addressing the project problem.
2. **S**crubbing or cleaning the data is the next step. This steps consists of crucial tasks such as data imputation to handle missing or invalid data and refining column names to enhance clarity and consistency, ensuring the data is in a suitable format for further analysis and interpretation.
3. **E**xploring the data will follow right after and gain deeper insights into the dataset. This involves examining the presence of outliers or unusual data points and comprehending the relationships between explanatory variables and the response variable.
4. **M**odeling the data will construct predictive models to forecast future stock prices.
5. I**N**terpreting the data is last. Upon conducting thorough analysis and evaluating the results, the authors will compare the performance of the tested models and provide insights on the suitable models for stock price predictions.

##**2.2 Obtaining the Data** ** The necessary libraries and modules required will be imported for data analysis, visualization, machine learning, and working with dates and times.*

The following libraries will be imported:

- pandas: A library for data manipulation and analysis, providing data structures and functions to effectively work with structured data.
- numpy: A library for numerical computing in Python, providing support for large, multi-dimensional arrays and matrices.
- re: A module for regular expression operations, which allows for pattern matching and manipulation of strings.
- matplotlib.pyplot: A plotting library that provides a MATLAB-like interface for creating visualizations.
- seaborn: A data visualization library built on top of matplotlib, providing additional functionality and aesthetic enhancements.
- bokeh.plotting, bokeh.models, bokeh.layouts: Libraries for interactive and web-based plotting and visualization.

- math: A module that provides mathematical functions and operations.
- scipy.stats: A module that contains a large number of probability distributions and statistical functions.
- tensorflow: A popular open-source library for machine learning and deep learning, providing tools for building and training neural networks.
- sklearn.preprocessing: A module from the scikit-learn library that provides various data preprocessing techniques.
- sklearn.linear_model: A module that provides linear regression models for machine learning tasks.
- sklearn.ensemble: A module that provides ensemble-based machine learning methods, such as random forests.
- tqdm: A library for creating progress bars and monitoring the progress of iterative tasks.

```
# Pandas, Bokeh, Seaborn, Mathplot library
import pandas as pd
import numpy as np
import re
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from bokeh.plotting import figure, show, output_file, output_notebook
from bokeh.models import ColumnDataSource, RangeTool,
DatetimeTickFormatter

# ARIMA
import statsmodels
import statsmodels.api as sm
from statsmodels.tsa.stattools import coint, adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.api import adfuller, kpss, seasonal_decompose
from statsmodels.tsa.arima.model import ARIMA
import itertools
import warnings


# Machine learning and deep learning library
import math
from scipy import stats
import tensorflow as tf
from tensorflow.keras import layers
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.linear_model import LinearRegression
```

```
from sklearn.ensemble import RandomForestRegressor
from tqdm import tqdm
import tensorflow.compat.v1 as tf
from tqdm import tqdm
from sklearn.model_selection import GridSearchCV
tf.compat.v1.random.set_random_seed(123)

# Datetime library
import datetime
from datetime import datetime
from datetime import timedelta
```

Based on the given information, it appears that the authors of a financial analysis project were looking for data on the stock FPT. They found that the vnstock library is a useful tool for financial analysts but discovered that the data for FPT in the library is only available starting from 20/03/2012, which is considered too limited for analysis.

To overcome this limitation, the authors decided to collect the data from Investing.com instead. However, they encountered another obstacle as Investing.com does not provide an API for directly importing data into Python.Therefore, the authors downloaded the data from Investing.com and then uploaded it to GitHub, presumably to access and utilize the data for analysis.

```
url = 'https://raw.githubusercontent.com/thienbao8112/AI_EDT_Final-
project_FPT-stock-price-prediction/main/FPT.csv'
df = pd.read_csv(url)
df = df.drop(columns=['% Thay đổi'])
```

Subsequently, the authors imported the pandas library and assigned the URL of the CSV file to the variable url from github.

> In this part, the necessary libraries and modules for data analysis, visualization, machine learning, and working with dates and times were imported. The stock price data from the provided CSV file was read into a pandas DataFrame for further analysis.

2.3 Scrubbing the Data

Fisrt of all, using df.info() alllow us to see general information of the data including:

- The number of rows and columns in the DataFrame.
- The name and data type of each column.
- The count of non-null values in each column.
- The memory usage of the DataFrame.

```
# Get general information about the data
df.info()
print("-" * 50)
print(pd.value_counts(df.dtypes))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4245 entries, 0 to 4244
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Ngày      4245 non-null   object
 1   Lần cuối  4245 non-null   object
 2   Mở        4245 non-null   object
 3   Cao       4245 non-null   object
 4   Thấp      4245 non-null   object
 5   KL        4245 non-null   object
dtypes: object(6)
memory usage: 199.1+ KB
-------------------------------------------------
object    6
dtype: int64

#Print the DataFrame
df
```

	Ngày	Lần cuối	Mở	Cao	Thấp	KL
0	26/12/2023	97,200.0	96,500.0	98,600.0	96,400.0	3.36M
1	25/12/2023	96,000.0	94,900.0	96,200.0	94,800.0	2.02M
2	22/12/2023	94,600.0	95,000.0	95,100.0	94,400.0	1.28M
3	21/12/2023	95,100.0	94,600.0	95,100.0	94,000.0	1.76M
4	20/12/2023	94,700.0	94,300.0	95,000.0	94,100.0	1.35M
...	...	...	...	...	...	...
4240	20/12/2006	20,668.4	20,668.4	20,668.4	20,668.4	3.15M
4241	19/12/2006	19,695.8	19,695.8	19,695.8	19,695.8	1.97M
4242	18/12/2006	18,763.7	18,763.7	18,763.7	18,763.7	3.09M
4243	15/12/2006	17,872.1	17,872.1	17,872.1	17,872.1	3.80M
4244	14/12/2006	17,021.1	17,021.1	17,021.1	17,021.1	4.02M

```
[4245 rows x 6 columns]
```

Certain columns of the DataFrame will be renamed to improve understanding, using rename funtion in pandas library:

- 'Ngày' is renamed to 'Date'
- 'Lần cuối' is renamed to 'Close'
- 'Mở' is renamed to 'Open'
- 'Cao' is renamed to 'High'
- 'Thấp' is renamed to 'Low'
- 'KL' is renamed to 'Volume'

```
# Renaming certain columns for better understanding
df = df.rename(columns = {
    'Ngày': 'Date',
    'Lần cuối': 'Close',
```

```
    'Mở': 'Open',
    'Cao': 'High',
    'Thấp': 'Low',
    'KL': 'Volume'
    })
```

Data in the DataFrame will be cleaned and transformed to a more suitable format for analysis, including converting the 'Date' column to datetime format, setting it as the index, being sorted in ascending order based on the date and converting numerical columns to float type.

```
# Change type of Date to datetime format and change Date to index
df['Date'] = pd.to_datetime(df['Date'], format='%d/%m/%Y')
df['Date'] = df['Date'].dt.strftime('%Y/%m/%d')
df.set_index('Date', inplace=True)
df.index = pd.to_datetime(df.index)
df.sort_index(ascending=True, inplace=True)

# Change type of Close, Open, High, Low to float
columns_to_convert = ['Close', 'Open', 'High', 'Low']

for column in columns_to_convert:
    df[column] = df[column].str.replace(',', '').astype(float)

# Change type of Volumn to float
def convert_volume(value):
    value = value.replace(',', '')
    if value.endswith('M'):
        return float(value[:-1]) * 1000000
    elif value.endswith('K'):
        return float(value[:-1]) * 1000
    else:
        return float(value)

df['Volume'] = df['Volume'].map(convert_volume)
```

Next, rows in the DataFrame df that contain missing values will be removed and the number of missing values eliminated from the dataset is provided.

The result indicates that the initial shape of the DataFrame df is (4245, 5), indicating that it has 4245 rows and 5 columns. After dropping any rows with missing values using df.dropna(), the shape of the DataFrame remains the same, with (4245, 5). This suggests that there were no missing values in the original DataFrame.

```
# Check to see if there are any missing values and eliminate from data
set
i = df.shape[0]
print("Initial shape of data:", df.shape)
df = df.dropna()
print("Shape of data after drop:", df.shape)
```

```
print("-"*50)
print("Missing_values_eliminated:",  i - df.shape[0])
```

```
Initial shape of data: (4245, 5)
Shape of data after drop: (4245, 5)
--------------------------------------------------
Missing_values_eliminated: 0
```

Duplicated values in the DataFrame df will be checked and eliminated subsequently. As seen, the initial shape of the DataFrame df is (4245, 5), indicating that it has 4245 rows and 5 columns. After dropping the duplicate rows using df.drop_duplicates(), the shape of the DataFrame becomes (4244, 5), indicating that one duplicate row was removed.

```
#Check to see if there are any duplicated values and eliminate from
data set
i = df.shape[0]
print("Initial shape of the dataset : ", df.shape)
df = df.drop_duplicates(keep="first", ignore_index=False)
print("Shape of the dataset after dropping the duplicate rows : ",
df.shape)
print("-"*50)
print("Duplicated_values_eliminated:",  i - df.shape[0])
```

```
Initial shape of the dataset :  (4245, 5)
Shape of the dataset after dropping the duplicate rows :  (4244, 5)
--------------------------------------------------
Duplicated_values_eliminated: 1
```

```
df.head()
```

Date	Close	Open	High	Low	Volume
2006-12-14	17021.1	17021.1	17021.1	17021.1	4020000.0
2006-12-15	17872.1	17872.1	17872.1	17872.1	3800000.0
2006-12-18	18763.7	18763.7	18763.7	18763.7	3090000.0
2006-12-19	19695.8	19695.8	19695.8	19695.8	1970000.0
2006-12-20	20668.4	20668.4	20668.4	20668.4	3150000.0

After being scrubbed, the initial data exploration and cleaning steps were performed on the DataFrame. The general information of the data, including column names, data types, and non-null value counts, was displayed. Certain columns were renamed for better understanding. The data was cleaned by converting the 'Date' column to datetime format, setting it as the index, sorting the DataFrame based on the date, and converting numerical columns to float type. Missing values were checked and eliminated from the dataset, and duplicated rows were also checked and removed. The resulting DataFrame is now ready for further analysis.

##2.4 Exploring the data ***

We also evaluate the descriptive statistics of the data of FPT stock. As show in the below cell, the describe function in Python is used to generate descriptive statistics, including count, mean, standard deviation, minimum value, quartiles (25%, 50%, 75%), and maximum value of each column.

As we can see from the below result, there are 4244 records (ranging from 14/12/2006 to 26/12/2023), which is quite a large amount of time, approximately 17 years.

For the close column, the average Close price of FPT stock was 26 010 VND, which is quite low in comparision with the current value. This can be explained that in the initial stage of FPT, the company may grow slowly and have to face a lot of problems as well as economic events (such as financial crisis 2008, Brexit, Covid 19, etc.). The minimum close price of FPT stock was 2985.5 VND. However, in recent years back to now, with the advancement of technology, the price have increase a lot. For the maximum close price, it reached to 99 000 VND, which is much higher than the average value. This reflects the improvements in the company operations and the market valuation for the company.

For the volumn, we can see that the company stock is actively traded, with the average volumn traded is approximately 2 085 575 volumn per day. The maximum volumn trading is 18 970 000 volumn per day, and the minimum volumn trading is 17 370 volumn per day. This reflect the market demand of the comany stock price, and it can be concluded to some extent that, the company price voliality is depend on market demand and supply.

```
df.describe()
```

```
              Close          Open          High           Low
Volume
count   4244.000000   4244.000000   4244.000000   4244.000000
4.244000e+03
mean   26010.223162  26009.166612  26289.742554  25740.676697
2.085575e+06
std    25736.684927  25729.357837  26026.116165  25440.485841
1.725397e+06
min     2986.500000   2899.600000   3054.800000   2899.600000
1.737000e+04
25%     8378.700000   8388.000000   8467.200000   8288.025000
9.079450e+05
50%    14523.400000  14525.850000  14643.200000  14433.600000
1.610000e+06
75%    30437.275000  30380.775000  30958.200000  30020.475000
2.690000e+06
max    99000.000000  99000.000000  100800.000000  98500.000000
1.897000e+07
```

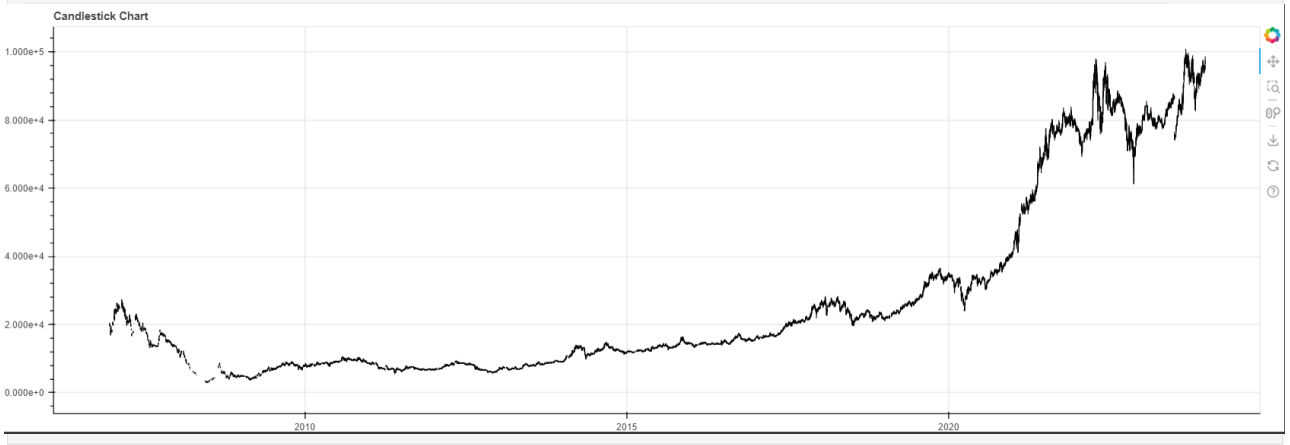
Now, we want to see the price of FPT stock overtime through a graph. So, we create a candle stick time series chart to illustrate. The idea is that we have to create columns, which illustrate the status (whether the stock increase, decrease or not change in that day). We create funtion inc_dec to classify whether in that day, the stock price increase, decrease or no change. If close price > opening price will be increase, close price < opening price will be decrease and others are equal. Secondly, the height to illustrate the difference between the close and open price by

absolute value. Finally, we plotting the increasing candle stick, choose rows where the Status is Increase and ploting them, fill the candle stick with green color. We do likewise with the decrease and equal one. Then we have graph as the result below.

According to the data depicted from 14/12/2006 to 26/12/2023,the overall trend of FPT stock indicates a positive trajectory over the analyzed period. In the period 2021 - 2022, we saw a significant increase in FPT stock price. However, there are three significant vents where the stock price of FPT experienced a decline: (1) in 2008, during the global financial crisis: The stock price of FPT witnessed a reduction as a result of the widespread economic turmoil during this period. (2)in late 2020, due to the Covid-19 pandemic: The stock price of FPT faced a decline as the pandemic caused disruptions to the global economy and financial markets, (3) in late 2022, during an economic recession: FPT stock price experienced a decrease amid an economic recession, which negatively affected investor sentiment and market conditions.

```
# We crete a df_1 to visualize as we donnot want to add anymore column
to the original data frame
df_1 = df.copy()
# Visualize stock graph
# Create funtion to know whether in that day, the stock price
increase, decrease or not change.
def inc_dec(c, o):
    if c > o:
        value="Increase"
    elif c < o:
        value="Decrease"
    else:
        value="Equal"
    return value

# The status column to illustrate the price change of stock, middle is
the price between the Open and Close price, Height is the height of
candle for that day.
df_1["Status"]=[inc_dec(c,o) for c, o in zip(df_1.Close, df_1.Open)]
df_1["Middle"] = (df_1.Open + df_1.Close)/2
df_1["Height"] = abs(df_1.Close - df_1.Open)

hours_24=24*60*60*1000 #24 represents the number of hours, 60
represents the number of minutes in an hour, 60 represents the number
of seconds in a minute,
                                        # and 1000 represents the
number of milliseconds in a second. By multiplying these values
together, we get the number of milliseconds in 24 hours, which is used
to visualize x_axis value

# Create a ColumnDataSource
source = ColumnDataSource(df)

# Create the figure
p = figure(x_axis_type='datetime', width=1500, height=500)
p.title.text = "Candlestick Chart"
```

```
p.segment(df_1.index, df_1.High, df_1.index, df_1.Low, color="Black")

# Plot the increasing candlesticks, choose column Status, Middle,
Height where
p.rect(df_1.index[df_1.Status=="Increase"],
df_1.Middle[df_1.Status=="Increase"],
       width=hours_24, height=df_1.Height[df_1.Status=="Increase"],
       fill_color="#e74c3c", line_color="black")

# Plot the decreasing candlesticks
p.rect(df_1.index[df_1.Status=="Decrease"],
df_1.Middle[df_1.Status=="Decrease"],
       width=hours_24, height=df_1.Height[df_1.Status=="Decrease"],
       fill_color="#2ecc71", line_color="black")

# Plot the equal candlesticks
p.rect(df_1.index[df_1.Status=="Equal"],
df_1.Middle[df_1.Status=="Equal"],
       width=hours_24, height=df_1.Height[df_1.Status=="Equal"],
       fill_color='r', line_color="black")

# Output the chart to an HTML file
output_file("chart.html")

# Show the chart
output_notebook()
show(p)

" "
```



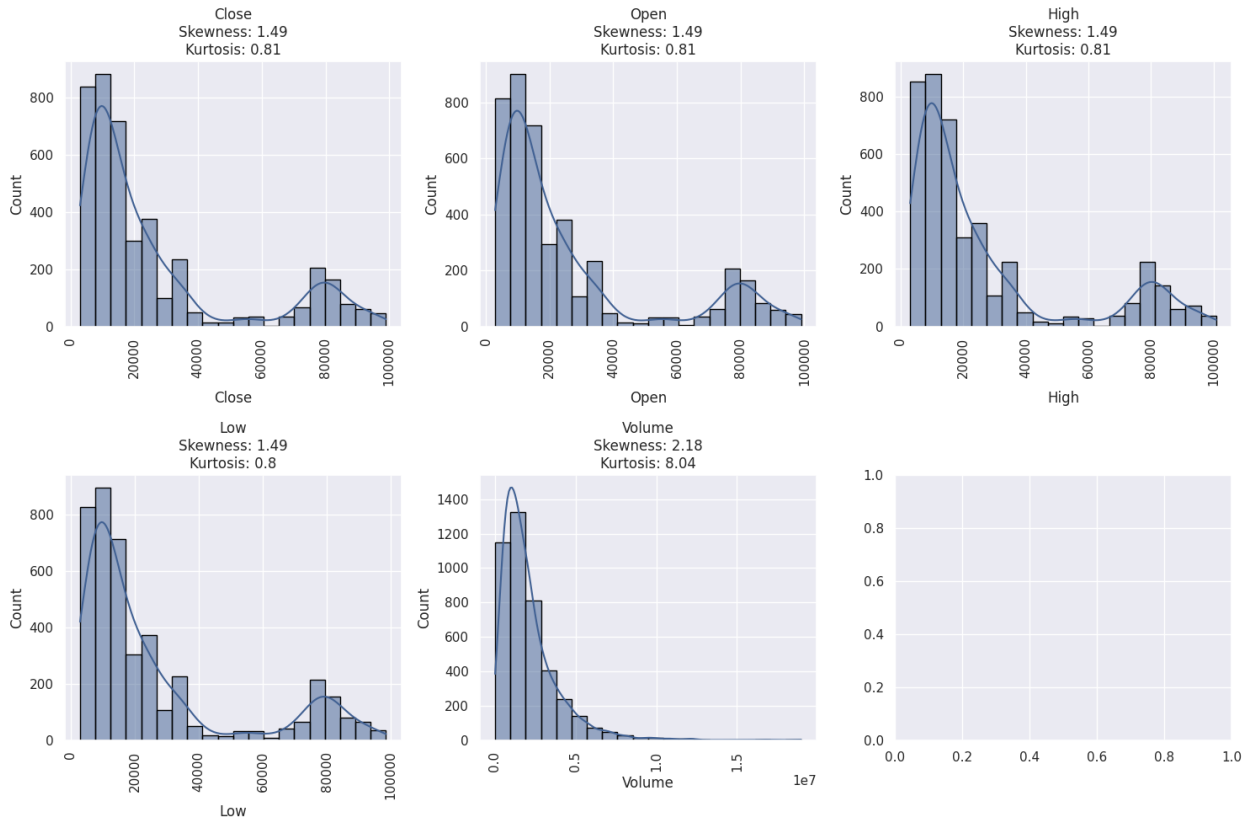
Besides, we also try to figure out the distribution, the skewness and kutoris of the data of FPT provided. First of all, we create 'fields' variable to store the column name of the data, excluding the Date, as we do not need to see the distribution of Date. Next, we create figuresize = (15, 10) which sets the size of the figure to be created as 15 inches in width and 10 inches in height. After having done the figure size or the huge figure, we create subplot by' cols' and 'rows' which are variables used to determine the number of columns and rows in the subplot grid as you can see, there are 3 subplot in a row (as indicated for cols = 3) and rows equal to the number of columns

we want to display divided by cols = 3, and we have 2 rows. After that, we set up a loop that iterates over the fields list by the code "for i, field in enumerate(fields):" to display each column that we want to plot. Then we use seaborn histplot "sns.histplot(df[field], color='#426294', edgecolor='black', kde=True, bins=20, ax=axes[i])" to set the column color, edgecolor, column width. Then we also display the Skewness and Kutoris (round to 2 decimal) and display as the result below.

Based on the histogram plot, it is observed that the Close, Open, High, and Low columns of the dataset exhibit the same skewness and kurtosis values of 1.49 and 0.81, respectively. A kurtosis value of 0.81 indicates a relatively low kurtosis, suggesting that the data points are relatively concentrated around the mean. However, the skewness value of 1.49 indicates a moderate skewness, implying the presence of outliers and a larger number of higher values in the dataset. Since this analysis pertains to time series forecasting, where the trend can either increase or decrease over time, it is not appropriate to remove outliers. Instead, a scaling method can be employed to process the data, which will be discussed in the following part.

For the volumn, the data exhibits a skewness of 2.18 and a kurtosis of 8.04, indicating a significant presence of outliers and extremely large values.

```
fields = df.columns[df.columns != "Date"]
figuresize = (15, 10)
cols = 3
rows = math.ceil(len(fields) / cols)

fig, axes = plt.subplots(rows, cols, figsize=figuresize)
axes = axes.flatten()

for i, field in enumerate(fields):
    sns.histplot(df[field], color='#426294', edgecolor='black',
kde=True, bins=20, ax=axes[i])
    axes[i].set_title(field + "\nSkewness: " +
str(round(df[field].skew(), 2)) + "\nKurtosis: " +
str(round(df[field].kurt(), 2)))
    axes[i].tick_params(axis='x', rotation=90)

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



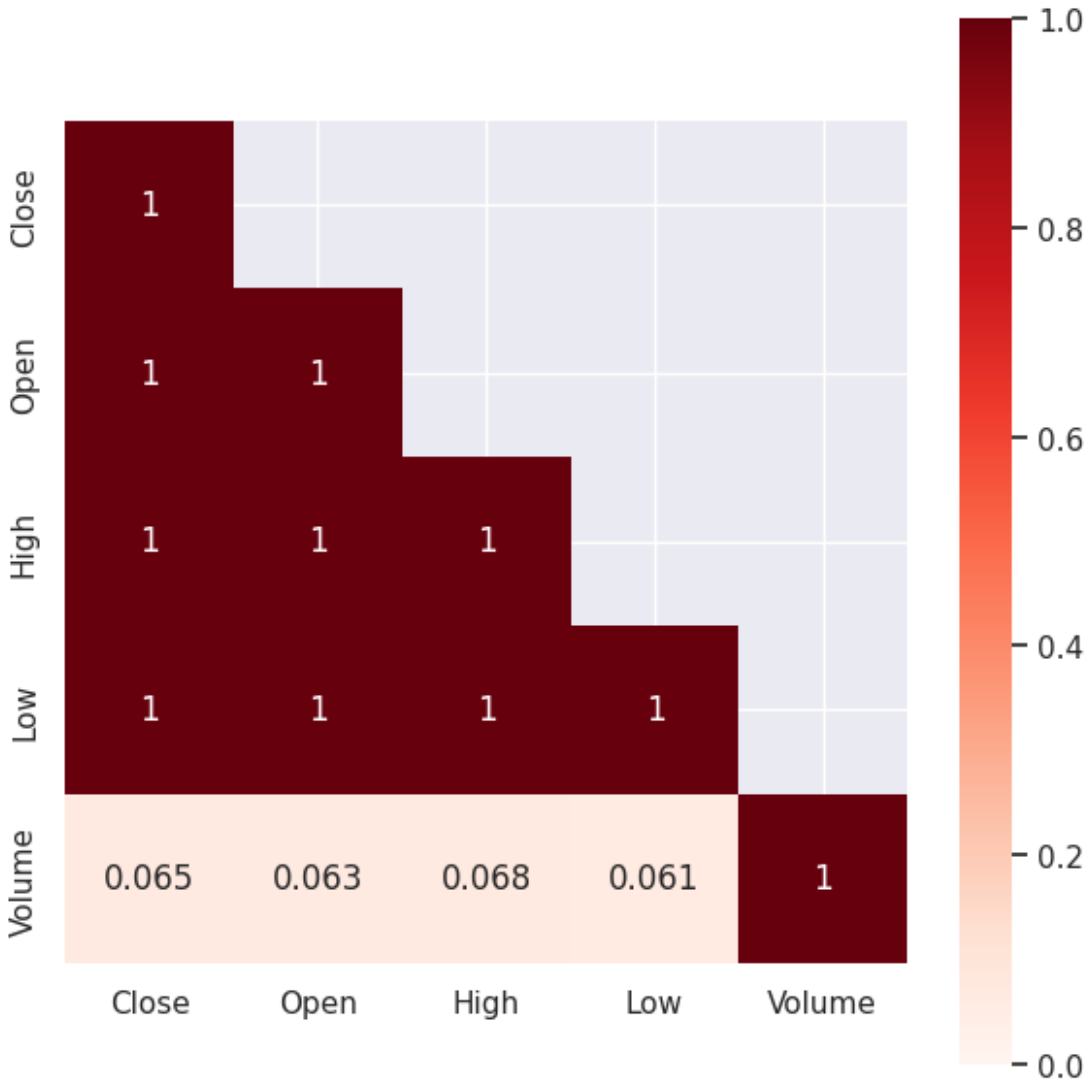
Then, we also try to figure out the relationship beween columns to avoid the multicollinearity which occurs when there is a strong linear association between independent variables. We first use df.corr() to calculate correlation between columns and store in matrix variable. Secondly, we converts the correlation matrix, into a NumPy array named heat. Thirdly, we created a figure and axes objects using plt.subplots() and assigns them to variables fig and ax, respectively, and customizing the figure size(7:7) (7 inch height and width) and font scale for the visualization for the plot. Finally, we generated a heatmap plot using seaborn's heatmap() function.

The strong correlation observed between the "Close," "Open," "Low," and "High" columns suggests a relationship between these variables in the dataset. This correlation can be explained by the inherent nature of financial market data, particularly in the context of stock prices of VN. According to central banks of VietNam, the stock market price of each days cannot change more than 7%. As a result, the difference between the opening and closing prices, as well as the low and high prices, is limited within this 7% range. This restriction leads to a strong correlation between these variables in the dataset, as they are bound by the same regulatory constraint.

```
# Correlation matrix
matrix = df.corr()

heat = np.array(matrix)
heat[np.tril_indices_from(heat)] = False
fig,ax= plt.subplots()
fig.set_size_inches(7,7)
sns.set(font_scale=1.0)
# vmax=1.0 and vmin=0.0 set the upper and lower limits of the color
```

```
range, respectively, for the heatmap.
# square=True ensures that the heatmap cells are square-shaped.
# annot=True displays the correlation values in each cell of the
heatmap.
# cmap="Reds" sets the color map used for the heatmap to "Reds", which
represents higher values with warmer colors.
sns.heatmap(matrix, mask=heat,vmax=1.0, vmin=0.0,
square=True,annot=True, cmap="Reds")
```

```
<Axes: >
```



In line with our objectives, we have chosen to solely consider the "Close" price of FPT stock data. Although the volume information could be valuable as well, it does not align with our specific goals. Hence, we will concentrate exclusively on analyzing and utilizing the "Close" price as the main input in our analysis.

So we drop the column Open, High,Low, Volume column using drop of pandas library

```
df = df.drop(['Open', 'High', 'Low', 'Volume'], axis=1)
df.head()
```

```
              Close
Date
2006-12-14  17021.1
2006-12-15  17872.1
2006-12-18  18763.7
2006-12-19  19695.8
2006-12-20  20668.4
```

Conclusion: in the explore the data part, we have presented descriptive statistics, specifically focusing on the price volatility of FPT stock over time. We have also assessed the data distribution and examined the correlation between different variables.Based on our objectives and the analysis above, we have decided to select only the Close price as the input for our model in the next part.

##**2.5 Modeling** ***

##Overview metrics

Root Mean Squared Error (RMSE)

RMSE (Root Mean Square Error) is a widely utilized metric for assessing the accuracy and performance of regression models in various academic and research disciplines. It serves as a measure of the average discrepancy between predicted values and actual values. RMSE values can range from zero to positive infinity and use the same units as the dependent (outcome) variable. The more lower the RMSE, the better the model. The formula of RMSE is as follow:

RMSE = $\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\acute{y} - y_i\right)^2}$

Percentage Accuracy

The accuracy, as measured by the mean squared percentage error (MSPE). the accuracy of predictions is quantified by comparing the average predicted value to each individual predicted value. The squared differences of these values are averaged, and the square root is taken. Subtracting this value from 1 provides a measure of accuracy, which is then multiplied by 100 to express it as a percentage. The accuracy percentage typically ranges from 0 to 100, with higher values indicating a higher accuracy percentage that the model's predictions align more closely with the actual values. The formula is as follow:

Percentage Accuracy =¿) * 100

$\acute{y}$: real value

y_i: predicted value

n: number of data

We also wrote a function to calculate RMSE and accuracy base on the formula above. These metrics will be used to evaluate our model performance in the following part.

```
  # Calculate RMSE (Root Mean Squared Error)
def calculate_rmse(real, predict):
    n = len(real)
    squared_error = 0
    for i in range(n):
        squared_error += (real[i] - predict[i])**2
    mean_squared_error = squared_error / n
    rmse = math.sqrt(mean_squared_error)
    return rmse

# CalculatePercentage accuracy
def calculate_accuracy(real, predict):
    real = np.array(real) + 1
    predict = np.array(predict) + 1
    percentage = 1 - np.sqrt(np.mean(np.square((real - predict) /
real)))
    return percentage * 100
```

##2.5.1 ARIMA

First of all, before coming to the ARIMA model, we splitted the data into train and test. We will use the last 30 days in the data to evaluated between the predicted and actual one.

First of all, we state the test_size variable equal 30, which indicate 30 days prediction. Secondly, we select all rows of the df DataFrame except for the last 30 rows which represents the training data. Then, we select the last 30 rows of the df DataFrame, representing the testing data. They all used negative indexing. Finally, we print displays the dimensions (number of rows and columns) of the original df DataFrame, df_train, and df_test, respectively.

According to the results, out of the total 4244 days of data, 4214 days will be used for training the model and the other 30 days will be use to evaluate the prediction of the ARIMA model.

```
test_size=30 #predict the next 30 days
train = df["Close"][:-test_size]
test = df["Close"][-test_size:]
train.shape, test.shape
```

```
((4214,), (30,))
```

The first step in analyzing time series data is to assess its stationarity. Stationarity refers to a property of a time series where its statistical properties remain constant over time. If a time series is stationary, it is easier to model and predict the better value.

We wrote a funtion named "check_stationarity" to check the stationary of the data with dataframe as an input. We used Augmented Dickey-Fuller (ADF) test, a statistical test commonly used for checking stationarity. First of all, the ADF test returns two values: the test statistic and the p-value. The function extracts these values from the ADF test results using adf_results = adfuller(df)[1:3]. Secondly, we determine whether the time series is stationary or non-stationary based on the ADF test results. If the p-value is less than 0.05 it indicates stationarity; otherwise, if the p-value is greater than or equal to 0.05, it indicated non-stationarity, and adf_s is assigned

the string 'Non-Stationary'. Finally, we creates a DataFrame named stationary_df to store the test results and terurn the results.

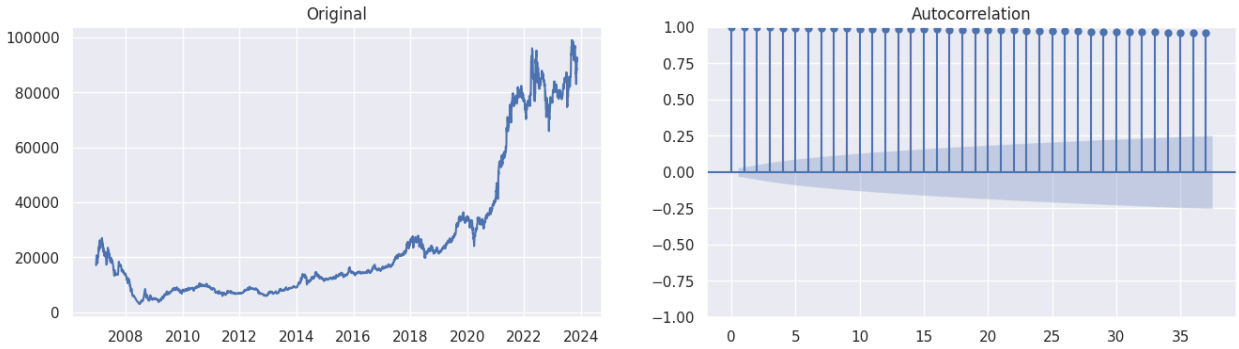
```
def check_stationarity(df):
    adf_results = adfuller(df)[1:3]
    adf_s = 'Non-Stationary'
    if adf_results[0] < 0.05:
        adf_s = 'Stationary'

    tests = ['ADF']
    pvalues = [adf_results[0]]
    is_stationary = [adf_s]
    stationary_df = pd.DataFrame({'tests': tests, 'p-value': pvalues,
'is_stationary': is_stationary})

    return stationary_df

check_stationarity(train)
```

```
  tests   p-value   is_stationary
0   ADF  0.998452  Non-Stationary
```

Besides, we also showed the original train data overtime, as well as the autocorrelation plot to check the stationary of the test data. First of all, we created a figure with two subplots using the subplots() function. The figure will have 1 row and 2 columns, and the size of the figure will be 16 inch in width and 4 inch in height. Then, we ploted the original time series data (train) on the first subplot (ax1) using the plot() function from ax1.plot(train); sets the title of the first subplot to "Original". After that, we calculated and plots the autocorrelation function (ACF) of the train data on the second subplot (ax2) using the plot_acf() function. Finally, we display two plots as a figure below.

If a time series is stationary, it means that its statistical properties do not change over time (The mean, variance (or standard deviation) remains constant over time and autocorrelation does not rely on the specific time points). However, based on the observed results, it can be concluded that the mean value of the time series changes over the years, and the autocorrelation exhibits statistical significance. Therefore, we can concluded that the series is not stationary once again.

```
# Create the original data trend and autocorrelation
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 4))
ax1.plot(train)
ax1.set_title("Original")
plot_acf(train, ax=ax2)

plt.show()
```

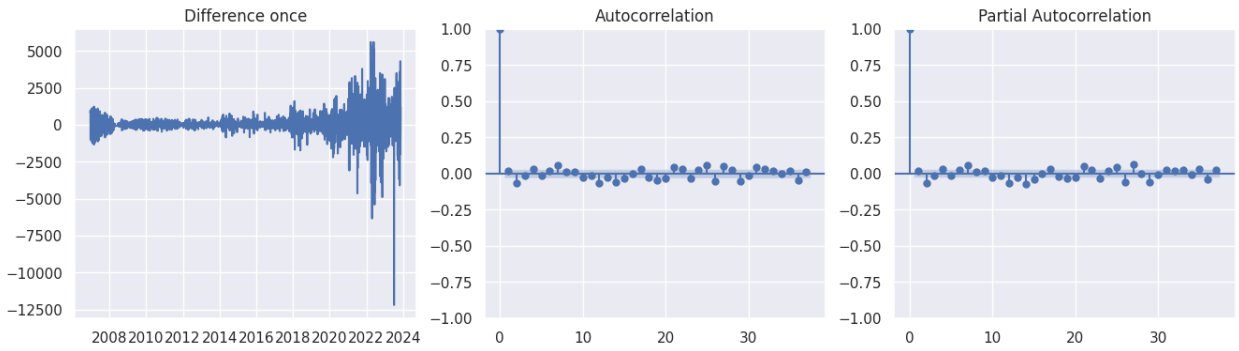


As a result our data is non stationary which means they lack a constant mean or exhibit trends over time. In order to make such a time series stationary, it is often necessary to differentiate it a certain number of times (Gujarati, D. N., & Porter, 2009). So we differentiate the data once, then show the data value over time, auto correlation and partial correlation after having differentiate.

First of all, we calculated the first-order difference of the test data using the diff() function and removes any resulting NaN values using dropna(). The differenced data is assigned to the variable diff. Secondly, we created a figure with three subplots using the subplots() function. The figure will have 1 row and 3 columns, and the size of the figure will be 16 inch width and 4 inch height. After that, we plots the differenced time series data, calculates and plots the autocorrelation function (ACF) and partial autocorrelation (PACF) of the differenced data. Finally, we display the plots on the screen.

Based on the results obtained from the differenced data, it appears that the series has become stationary. The values of the time series now fluctuate around a relatively constant mean value over the years. Additionally, the autocorrelation structure of the differenced data shows a satisfactory pattern. Therefore, from three graph below, we can conclude that the data after differenced have been stationary.

```
# 1st order difference of train data
diff = train.diff().dropna()

# Plot the 1st difference of train data overtime, PACF and PACF
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(16, 4))
ax1.plot(diff)
ax1.set_title("Difference once")
plot_acf(diff, ax=ax2)
plot_pacf(diff, ax=ax3)

# Show the plot
plt.show()
```



After having check the stationary of the diffenciated data using plot, we want to further confirm this by utilizing the function we created above to conduct the Augmented Dickey-Fuller (ADF) test. As a result the data now have been stationary.

```
# Chekc stationary of 1st differientation of train data
check_stationarity(diff)
```

```
  tests       p-value is_stationary
0   ADF  2.666436e-24    Stationary
```

According to Gujarati, D. N., & Porter (2009), after the differencing is performed, an autoregressive moving average (ARMA) model can be applied to the transformed series. When referring to the original time series, it is then denoted as an autoregressive integrated moving average (ARIMA) time series, represented as ARIMA(p, d, q), where p denotes the number of autoregressive terms, d the number of times the series has to be differenced before it becomes stationary, and q the number of moving average terms.

To determine the appropriate value for p (the number of autoregressive terms), we examined the partial autocorrelation plot of the differenced series. Based on this plot, it suggests that the p value could potentially be 2, there are many more values that may be used, but we just take the minimum value which equal 2.

After applying first-order differencing to the data, we obtained a differenced series with a d-value of 1.

Regarding the q value (the number of moving average terms), we analyzed the autocorrelation plot. From this plot, it appears that the minimum acceptable q value is 2.

In summary, based on the analysis of the partial autocorrelation and autocorrelation plots, it is suggested that the ARIMA model for the data could have the combination of ARIMA (2;1;2).

Now we will create an instance of the ARIMA model with the order parameter set to (2,1,2), which represents the chosen order of the AR, I, and MA terms. Finally, we fit the model to our test dataset using the fit() method, and the result of the fitting process is stored in the result variable.

```
# ARIMA Model, chosse order = (2,1,2)
model = ARIMA(train, order=(2, 1, 2))
result = model.fit()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/
tsa_model.py:473: ValueWarning: A date index has been provided, but it
has no associated frequency information and so will be ignored when
e.g. forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
```

Then we display a comprehensive summary of the fitted ARIMA model, including information about the model coefficients, standard errors, t-statistics, p-values, and various statistical metrics such as AIC, BIC, and HQIC.

The results indicate that the model is quite good, with AIC approximately 488, and P>|z| is 0.000 < 0.05. The Ljung-Box Q-statistic with p-values equal 0.2 > 0.05 which indicate that the residuals are not significantly autocorrelated, indicating that ARIMA (2,1,2) model is appropriate.

```
result.summary()
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results

==========================================================================
========
Dep. Variable:                   Close   No. Observations:
4214
Model:                 ARIMA(2, 1, 2)   Log Likelihood               -
33288.646
Date:                Mon, 01 Jan 2024   AIC
66587.291
Time:                        13:44:58   BIC
66619.021
Sample:                             0   HQIC
66598.509
                                - 4214

Covariance Type:                  opg

==========================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
```

```
------------------------------------------------------------------------
--------
ar.L1         -0.5041      0.012    -43.081      0.000      -0.527
-0.481
ar.L2         -0.9396      0.011    -82.485      0.000      -0.962
-0.917
ma.L1          0.5183      0.014     37.256      0.000       0.491
0.546
ma.L2          0.9181      0.013     68.143      0.000       0.892
0.944
sigma2      4.274e+05   1922.389    222.327      0.000    4.24e+05
4.31e+05
==========================================================================
==============
Ljung-Box (L1) (Q):                   0.20   Jarque-Bera (JB):
374317.24
Prob(Q):                              0.66   Prob(JB):
0.00
Heteroskedasticity (H):              13.80   Skew:
-1.50
Prob(H) (two-sided):                  0.00   Kurtosis:
49.08
==========================================================================
==============

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

After obtaining the best fit for the ARIMA model, we proceed to make predictions for the next 30 days. Firstly, we call the get_forecast() method on the result object with the parameter test_size for the predicted size. Then, access the predicted_mean attribute of the resulting forecast object and assign the values to the variable fc, which means forecast values. Secondly, we call the get_forecast() method on the result object with and use the conf_int() method to obtain the confidence intervals, and assign the resulting DataFrame to the variable pred_ci.

```
 # Forecasted values
pred = result.get_forecast(test_size).predicted_mean

# Confidence intervals
pred_ci = result.get_forecast(test_size).conf_int()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/
tsa_model.py:836: ValueWarning: No supported index is available.
Prediction results will be given with an integer index beginning at
`start`.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
```

```
.py:836: FutureWarning: No supported index is available. In the next
version, calling this method in a model without a supported index will
result in an exception.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:836: ValueWarning: No supported index is available. Prediction
results will be given with an integer index beginning at `start`.
  return get_prediction_index(
```

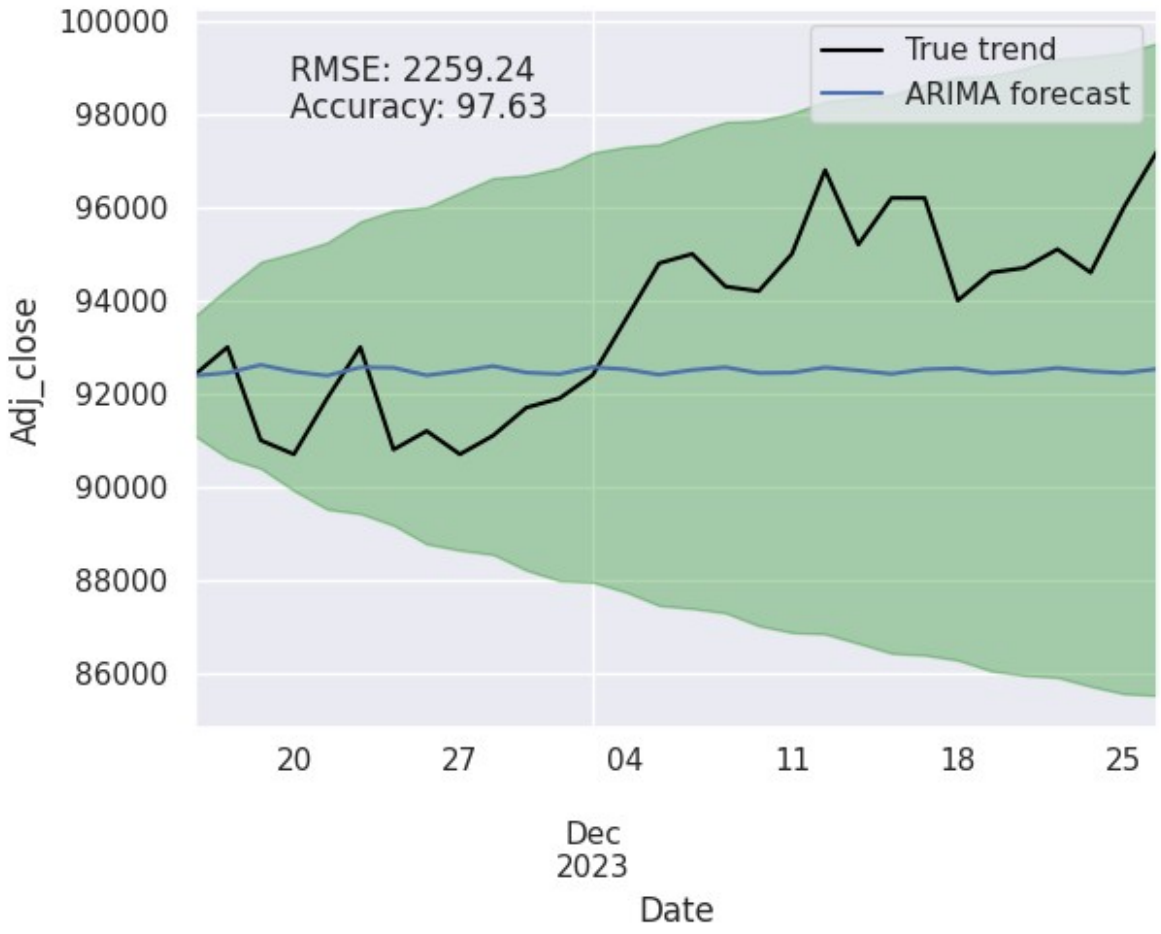
We set the new index for pred_ci and pred as we want those index is similar to the index of the test data in order to make it easier to visualization and comparision.

```
# Set new index for pred_ci
pred_ci.index = df.index[-30:]

 # Set new index for fc
pred.index = df.index[-30:]
```

The below cell firstly calculate the performance metrics between predicted and test value. Then we, plot the truedata line, label "True trend" and set the color line is black. After that, we plotting the arima forecast value. After that, we utilized the fill_between() function of the ax object, which represents the axes of the plot. The function fills the area between the lower (pred_ci.iloc[:, 0]) and upper (pred_ci.iloc[:, 1]) confidence intervals with the specified color ('green'). The pred_ci.index specifies the x-values or index positions where the filling should occur. The alpha parameter sets the transparency level of the filled area. Furthurmore, we display the RMSE and accuracy in the plot as well. Finally, we plotting the data and have the below graph.

```
# Calculating metrics
accuracy = calculate_accuracy(test, pred)
rmse = calculate_rmse(test, pred)

# Plotting the true data
ax = test.plot(label='True trend', color = 'black')

# Plotting the ARIMA forecast
ax = pred.plot(label='ARIMA forecast')

# Filling the area between the confidence intervals
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='green', alpha=.3)

# Displaying RMSE and accuracy as text on the plot
ax.text(0.1, 0.9, f'RMSE: {rmse:.2f}', transform=ax.transAxes)
ax.text(0.1, 0.85, f'Accuracy: {accuracy:.2f}',
transform=ax.transAxes)

# Setting the labels and legend
```

```
ax.set_ylabel('Adj_close')
plt.legend()

# Displaying the plot
plt.show()
```




```
# This code is used to create a Dataframe, which store our performance
metrics to compare between the models, in the later part
comparison = pd.DataFrame(columns=['Model', 'RMSE', 'Accuracy'])
arima = pd.DataFrame({'Model': ['ARIMA'], 'RMSE': [rmse], 'Accuracy':
[accuracy]})
```

Conclusion:

We evaluate the model through 2 keys performace metrics, namely accuracy and RMSE that we have created the function above.

The obtained results indicate a high level of accuracy, with an accuracy rate of approximately 97.63%, and a relatively low root mean squared error (RMSE) of approximately 2259.24. These findings suggest that there is generally a small discrepancy between the predicted values generated by the ARIMA model and the actual values.

However, upon closer examination of the plot displaying the true values and the predicted values generated by the ARIMA model, it becomes evident that the forecasted trend of the ARIMA model does not closely align with the actual trend of the data. This observation suggests that the ARIMA model may not accurately capture the underlying trend or pattern of the actual values. These findings indicate that while the ARIMA model may exhibit satisfactory overall performance in terms of accuracy and error metrics, it may struggle to capture the complex dynamics or long-term trends present in the data.

##2.5.2 Random Forest

In order to use the random forest algorithm for stock price prediction, it is necessary to include a column called "Close t-1" as a feature in our model. This column represents the closing price of the stock from the previous day. By including this feature, we are able to make predictions based on the historical stock price trends.

Firstly, we created a new DataFrame df_rf by selecting only the first column of df using iloc[:,0:1], which indicate the column Close price. Secondly, we added a new column called "Close t-1" to df_rf by shifting the values of the original column by one position using the shift(1) function. In other words, the value in row n of the "Close t-1" column is the value from row n+1 of the "Close" column. Finally, rows that contain missing values (NaN) in df_rf are removed using the dropna() function, ensuring that the dataset is clean and suitable for further analysis. The Dataframe df_rf is shown below.

```
# create dataframe for ranfdom fores data
df_rf=df.iloc[:,0:1]
df_rf["Close t-1"] = df_rf.shift(1)
df_rf = df_rf.dropna()

# Show df_rf
df_rf.head()
```

```
<ipython-input-230-116ac3daf2e4>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df_rf["Close t-1"] = df_rf.shift(1)
```

Date	Close	Close t-1
2006-12-15	17872.1	17021.1
2006-12-18	18763.7	17872.1
2006-12-19	19695.8	18763.7
2006-12-20	20668.4	19695.8
2006-12-21	19655.3	20668.4

In this step, we splitted the data into train and test. We will use the last 30 days in the data to evaluated between the predicted and actual one.

According to the results, out of the total 4243 days of data, 4213 days will be used for training the model and the other 30 days will be use to evaluate the prediction of the Random Forest model.

```
# Specific the test size
test_size = 30
# Split train and test
df_rf_train = df_rf[:-test_size]
df_rf_test = df_rf[-test_size:]
df_rf.shape, df_rf_train.shape, df_rf_test.shape
```

```
((4243, 2), (4213, 2), (30, 2))
```

Once the data has been divided into training and testing sets, the next step is to create the features (X) and target variable (y) for Random Forest Regression. In this case, the target variable is assigned to y_train and y_test by selecting only the "Close" column, as it represents the variable we want to predict with the Random Forest Regression model. The process including:

(1): Creating X_train and X_test features by removing the "Close" column using the drop() function.

(2): Creating y_train and y_test by selecting the "Close" column only.

```
# Crreate target and feature for test and train data
X_train =df_rf_train.drop("Close", axis=1)
y_train = df_rf_train["Close"]
X_test = df_rf_test.drop("Close", axis=1)
y_test = df_rf_test["Close"]
# show the shape of train and test
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

```
((4213, 1), (4213,), (30, 1), (30,))
```

Now after having processing the data, we will create and train a Random Forest Regression model using scikit-learn. The first line created an instance of the Random Forest Regression model. The RandomForestRegressor class is imported from the scikit-learn library. The criterion parameter is set to 'absolute_error', which specifies the criterion used to measure the quality of a split during the tree construction.

In Random Forest, each time we run the model, the bootstraping samples are initialized randomly, which can result in different trees being built during different training runs. So the "random_state parameter = 42" is to ensure reproducibility of results. This means that each time we train the model with the same random_state value, the same set of bootstrapping will be made during the tree construction process.

We train the Random Forest Regression model. This last line of code fits the model to the training data, allowing it to learn the patterns and relationships between the features (X_train) and the target variable (y_train) to make prediction.

```
# Create model, with Default Bootstrapping = True
model = RandomForestRegressor(criterion='absolute_error', random_state
= 42)

# Train model
model.fit(X_train,y_train)

RandomForestRegressor(criterion='absolute_error', random_state=42)
```

After having trained the model, we will use it to make prediction, with the X_test data, and preds variable will return the predicted_y with each X_test data.

```
# Create predictions
preds = model.predict(X_test)
```

Now we evaluate the model through 2 keys performace metrics, namely accuracy and RMSE that we have created the function above.

The results demonstrate a high accuracy of approximately 97.94% and a very low root mean squared error (RMSE) of approximately 1930.09. These findings indicate a minimal difference between the predicted values and the actual values. Despite these favorable outcomes, there is still room for improvement through parameter tuning to identify optimal model settings. By refining the parameters, we can aim to enhance the model's performance and achieve even better predictive accuracy and reduced errors.

```
# Calculate performance metrics
accuracy_rf = calculate_accuracy(y_test,preds)
rmse_rf = calculate_rmse(y_test.ravel(),preds)

# Print the performance metrics value
print("Accuracy:", accuracy_rf)
print("RMSE:", rmse_rf)

Accuracy: 97.94235387847581
RMSE: 1930.0946363453709

# This code is used to store our performance metrics to compare
between the models, in the later part
RANDOMFOREST = pd.DataFrame({'Model': ['Random Forest'], 'RMSE':
[rmse_rf], 'Accuracy': [accuracy_rf]})
```

###Random Forest parameter tuning

First of all, we define the parameter grid for the grid search: The param_grid dictionary specifies the different values to be tested for each hyperparameter of the Random Forest regressor including the n_estimators (The number of trees in the forest), max_depth (The maximum depth of the tree), min_samples_split (The minimum number of samples required to split an internal node), and min_samples_leaf (The minimum number of samples required to be at a leaf node) parameters are being tuned.

Secondly, we create a Random Forest regressor with specific settings. The max_features = 1 (max_features=n_features), indicating that all features should be considered at each split. The criterion is set to 'absolute_error', indicating that the mean absolute error (MAE) is be used as the criterion for splitting.

Thirdly, we performed a grid search using cross-validation to find the best combination of hyperparameters for a Random Forest regression model. The GridSearchCV class is utilized for this purpose. The Random Forest regressor, represented by the rf object, serves as the estimator. The parameter grid, defined by the param_grid dictionary, outlines the different values to be tested for each hyperparameter. To evaluate the model's performance, 5-fold cross-validation is employed, indicated by the cv parameter being set to 5. The evaluation metric used is the negative mean absolute error (neg_mean_absolute_error).

After that, we fit the grid search model to the data which trains and evaluates the Random Forest regressor with different combinations of parameters from the parameter grid.

Finally, we print the best parameters and best score to display.

```
# Define the parameter grid for the grid search
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create a Random Forest regressor
rf = RandomForestRegressor(max_features=1.0,
criterion='absolute_error', random_state = 42)

# Perform grid search with cross-validation
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5,
scoring='neg_mean_absolute_error')

# Fit the grid search model to the data
grid_search.fit(X_train, y_train)

# Print the best parameters and best score
print("Best parameters: ", grid_search.best_params_)
```

```
Best parameters:  {'max_depth': 10, 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 50}
```

```
# Model with best parameters
best_params = {
    'max_depth': 10,
    'min_samples_leaf': 2,
    'min_samples_split': 2,
    'n_estimators': 50
    }
```

```
# Create model
model = RandomForestRegressor(max_features=1.0,
criterion='absolute_error', **best_params, random_state=42)

# Train model
model.fit(X_train,y_train)

RandomForestRegressor(criterion='absolute_error', max_depth=10,
                      min_samples_leaf=2, n_estimators=50,
random_state=42)

# Create predictions
preds = model.predict(X_test)
```

The obtained results demonstrate improved performance after applying parameter tuning to the Random Forest regressor model. The resulting RMSE (Root Mean Squared Error) value of 1479.27 indicates that, on average, the predicted stock close values deviate by approximately 1479.27 units from the actual values. Additionally, the achieved accuracy of 98.4255% suggests that the model accurately predicts the direction or trend of the stock close values in a large majority of cases.

```
# Calculate performance metrics
accuracy_rf = calculate_accuracy(y_test,preds)
rmse_rf = calculate_rmse(y_test.ravel(),preds)

# Print the performance metrics value
print("Accuracy:", accuracy_rf)
print("RMSE:", rmse_rf)

Accuracy: 98.42557922748908
RMSE: 1479.2729725510433

# This code is used to store our performance metrics to compare
between the models, in the later part
RANDOMFOREST_TUNING = pd.DataFrame({'Model': ['Random Forest Tuning'],
'RMSE': [rmse_rf], 'Accuracy': [accuracy_rf]})
```

In this step, we will visualize the actual stock close line and the predicted values after having tuned by plotting them on a graph. Additionally, we will include the RMSE and accuracy metrics on the plot to provide a comprehensive overview of the model's performance.

The graph illustrates that the predicted line closely tracks the actual values of the FPT stock's close price. This indicates that the model is capable of capturing the underlying patterns and trends in the data, resulting in accurate predictions in the short term. However, it is observed that as the prediction extends further into the future, the predicted values start to deviate more from the actual values. This is a common characteristic in financial forecasting, as the inherent uncertainty and complexity of the market make it is elusive in longer time.

For building the graph below, we have coded as following. First we create the figure with 15 inch width and 7 inch heigt to store the graph. Secondly, we plotting the actual line, identify the

x_axis values from 0 to the length of the preds array, y_test represents the actual values of the stock's close price, ploting on the y_axis, choose the label for the line is "Actual" and set line color to black. We do like wise with predicted values. After that, we assigns the RMSE and accuracy values to text variables and then position the text strings on the plot using the plt.text() function. The (0.02, 0.15) and (0.02, 0.1) coordinates specify the x and y locations, respectively, as a fraction of the axes' size. The transform=plt.gca().transAxes parameter ensures that the coordinates are relative to the axes, and the fontsize=12 sets the font size of the text. The verticalalignment='top' aligns the text to the top of the specified coordinates. Finally, we set x and y axis label adn show the plot.

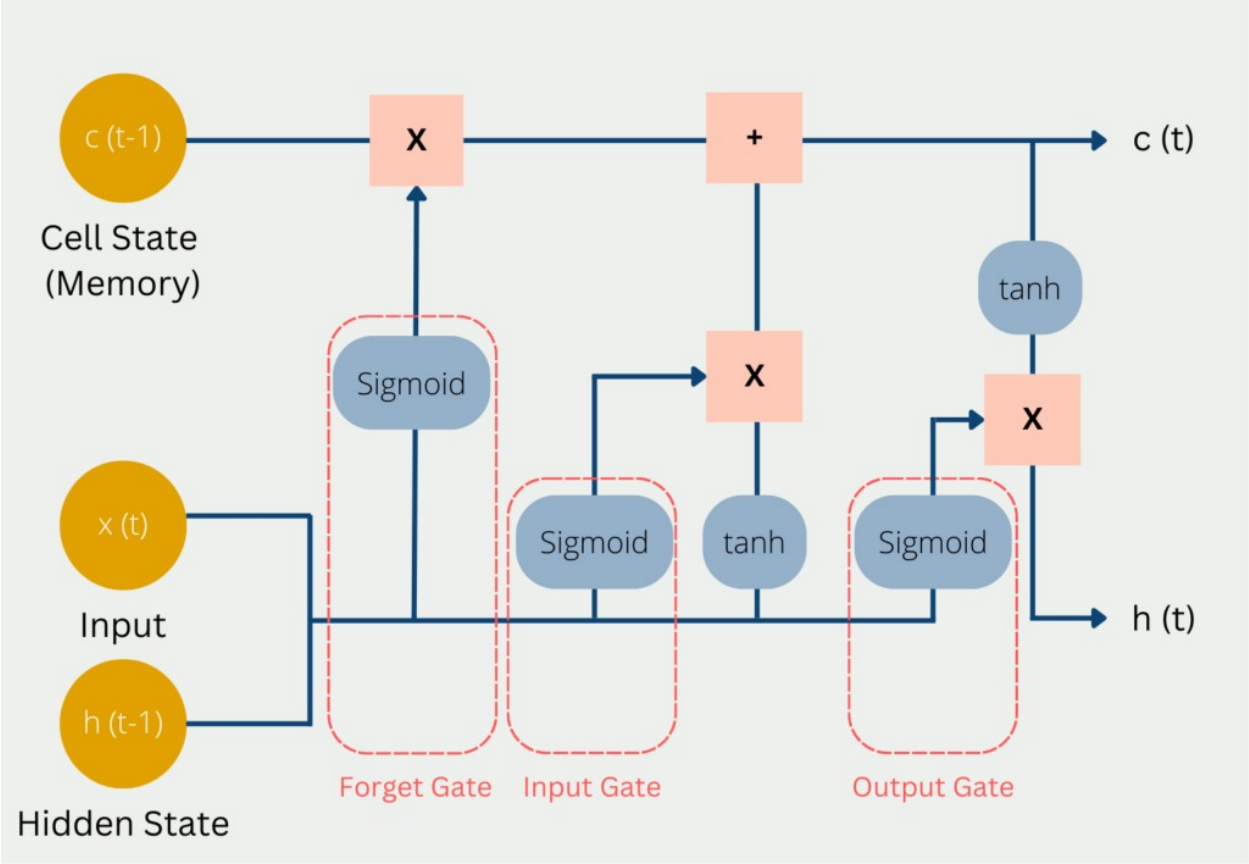
```
# Create figure
plt.figure(figsize=(15, 7))
sns.set_style("whitegrid")

# Plotting the actual, predicted and title
plt.plot(range(len(preds)), y_test, label='Actual', color="black")
plt.plot(range(len(preds)), preds, label='Predicted')
plt.title("FPT Close price - Random Forest", color='black',
fontsize=20)

# Adding the rmse and accuracy text to be shown in the plot.
rmse_text = f"RMSE: {rmse_rf:.4f}"
accuracy_text = f"Accuracy: {np.mean(accuracy_rf):.4f}"

plt.text(0.02, 0.15, rmse_text, transform=plt.gca().transAxes,
fontsize=12, verticalalignment='top')
plt.text(0.02, 0.1, accuracy_text, transform=plt.gca().transAxes,
fontsize=12, verticalalignment='top')

# Set the x axis and y axis label, then show the plot
plt.xlabel('Index')
plt.ylabel('Close price', color='black', fontsize=15)
plt.legend()
plt.show()
```

Conclusion:

After we used the Random Forest regressor to forecast stock prices over a span of 30 days, the subsequent outcomes were obtained::

1. The root mean squared error (RMSE) and accuracy before doing parameter tuning are approximately 1930.09 and 97.94% correspondingly.
2. After the tuning parameters process using Grid Search, the optimal parameters obtained are: {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 50}.
3. The performance metrics of the tuned model demonstrated significant improvement compared to the previous version. The root mean squared error (RMSE) decreased to 1479.27, while the accuracy increased to 98.4255%.
4. In both the Random Forest model before and after tuning, the predicted values closely align with the actual values. However, as the forecast horizon extends further into the future, the predicted values gradually exhibit greater divergence from the actual values.

##2.5.3 Long-Short Term Memory (LSTM)



To examine the data provided to the model, we can use the code snippet df.iloc[:, 0:1] to extract all rows (:) and the first column (0:1) from the dataset. By doing this, we can observe that the Date column, which serves as the index column, has been arranged in descending order, starting from the furthest day (14/12/2006) and progressing towards the most recent day (26/12/2023).

```
df.iloc[:, 0:1]

              Close
Date
2006-12-14  17021.1
2006-12-15  17872.1
2006-12-18  18763.7
2006-12-19  19695.8
2006-12-20  20668.4
...             ...
2023-12-20  94700.0
2023-12-21  95100.0
2023-12-22  94600.0
2023-12-25  96000.0
2023-12-26  97200.0

[4244 rows x 1 columns]
```

Min-Max scaling is a prevalent data preprocessing technique used to normalize numerical features, while remains the distribution of oroginal data. The min max scaling is to rescale the data within a specific range, typically between 0 and 1.

The purpose of the min max scaling is that in LSTM model, it use activation functions namely sigmoid and tanh. These functions are sensitive to the input range and can saturate or exhibit limited sensitivity when dealing with large input values. Scaling the input to a smaller range improves the activation function's effectiveness and can lead to better model performance. Furthurmore, LSTM models utilize the backpropagation algorithm to compute gradients and update the model's weights during training. Large input values can result in large gradients, and when large gradients are present, weight updates can be significant, the learning rate is significnat which making it difficult to find the optimal value where loss function is minimum.

First of all, we created an instance of the MinMaxScaler and fits it to the data in the first column (iloc[:, 0:1]) of the DataFrame df and make sure that the values in the selected column are treated as floats by astype('float32'). Secondly, we applies the scaling transformation to the data in the first column of df using the transform() method of the minmax scaler object. The transformed data is stored in the df_log variable. Finally, we converted the transformed data stored in df_log into a new pandas DataFrame, assigning it back to the df_log variable and show the first few rows of the df_log DataFrame.

```
# Sacling
minmax = MinMaxScaler().fit(df.iloc[:, 0:1].astype('float32')) # Close
index
df_log = minmax.transform(df.iloc[:, 0:1].astype('float32')) # Close
index
df_log = pd.DataFrame(df_log)
df_log.head()
```

```
          0
0  0.146173
1  0.155037
2  0.164323
3  0.174031
4  0.184161
```

Splitting training and testing size is the important part of evaluating the model because we can evaluate how well the model generalizes to unseen data. The training set is used to train the model, while the testing set serves as a proxy for new, unseen data. The model's performance on the testing set provides an estimate of how well it will perform on real-world data.

In time series forecasting, particularly in predicting stock prices, the accuracy of predictions tends to decrease as the forecast time increase. Therefore, we choose to forecast the close price for a short period of 30 days, and using the remaining data for training the model.

First of all, we state the test_size variable equal 30, which indicate 30 days prediction. Secondly, we select all rows of the df_log DataFrame except for the last 30 rows which represents the training data. Then, we select the last 30 rows of the df_log DataFrame, representing the testing data. They all used negative indexing. Finally, we print displays the dimensions (number of rows and columns) of the original df DataFrame, df_train, and df_test, respectively. According to the

results, out of the total 4244 days of data, 4214 days will be used for training the model and the other 30 days will be use to evaluate the prediction of the LSTM model.

```
# Choose test size is 30 days
test_size = 30

# Split train and test
df_train = df_log.iloc[:-test_size]
df_test = df_log.iloc[-test_size:]
df.shape, df_train.shape, df_test.shape
```

```
((4244, 1), (4214, 1), (30, 1))
```

In this step, we defined a TensorFlow model class called Model:

In our below code, the model class represents a recurrent neural network (RNN) model with LSTM (Long Short-Term Memory) cells. We created the model with initialize parameters inclduing learning rate, number of layers, input size (size), hidden layer size (size_layer), output size, and forget bias. Then in the model class, defined an LSTM cell with the specified size_layer and returns the LSTM cell object.

Secondly, to construct RNN, LSTM cells are used, and multiple layers are stacked together to create a multi-layer RNN. The number of layers is determined by the parameter "num_layers", and each layer employs an LSTM cell created by the "lstm_cell" function.

Thirdly, we defined TensorFlow placeholders for the input data (self.X) and the target output (self.Y). Placeholders are used to feed actual data into the TensorFlow computational graph during training or inference.

Fourthly, we applies dropout regularization to the LSTM cells. The DropoutWrapper is a wrapper class that adds dropout to the provided RNN cell (rnn_cells). The output_keep_prob parameter specifies the probability of keeping a unit output. which equal to drop_out variable we will define later.

Fifthly, A placeholder is defined for the initial state of the LSTM cells, with a shape of (None, num_layers * 2 * size_layer). The None dimension represents the batch size, which can vary and will be determined during the execution phase when actual data is fed into the placeholder. Bacth size is the number of input sequences that are processed simultaneously in parallel.

After that, Dynamic unrolling of the RNN is performed over the input sequence (self.X). This step utilizes the dropout-wrapped RNN cells, input data, and initial state. The outputs of the RNN (self.outputs) and the last state of the RNN (self.last_state) are returned.

Then, we fully connected dense layer to the last output of the RNN (self.outputs[:, -1]) to obtain the logits, which are the unnormalized predictions of the model. The dense layer maps the RNN output to the desired output size (output_size).

Next, we calculated the mean squared error (MSE) between the target output (self.Y) and the logits (self.logits). The cost represents the discrepancy between the model's predictions and the actual target values.

Finally. we use Adam optimizer with the specified learning rate (learning_rate) which minimize method is then called with the cost function (self.cost) to create a training operation that minimizes the cost. Because Adam is like stochastic gradient descent, but not require as stochastic, therefore Adam tends to find the optimal values faster, so we decided to use it.

The code below is inspired by Husein Zolkepli, in his repository named Stock-Prediction-Models (Source: https://github.com/huseinzol05/Stock-Prediction-Models)

```
tf.disable_eager_execution()
# Create class representing RNN LSTM
class Model:
    def __init__(
        self,
        learning_rate,
        num_layers,
        size,
        size_layer,
        output_size,
        forget_bias=0.1,
    ):
        # Create lstm cell, with hidden size layer
        def lstm_cell(size_layer):
            return tf.compat.v1.nn.rnn_cell.LSTMCell(size_layer,
state_is_tuple=False)

        # Create RNN with LSTM cell
        rnn_cells = tf.compat.v1.nn.rnn_cell.MultiRNNCell(
            [lstm_cell(size_layer) for _ in range(num_layers)],
            state_is_tuple=False,
        )

        # Define place holder for input and output data
        self.X = tf.compat.v1.placeholder(tf.float32, (None, None,
size))
        self.Y = tf.compat.v1.placeholder(tf.float32, (None,
output_size))
        drop = tf.compat.v1.nn.rnn_cell.DropoutWrapper(
            rnn_cells, output_keep_prob=forget_bias
        )
        # Define initial state of LSTM
        self.hidden_layer = tf.compat.v1.placeholder(
            tf.float32, (None, num_layers * 2 * size_layer)
        )
        # Unrolling
        self.outputs, self.last_state = tf.compat.v1.nn.dynamic_rnn(
            drop, self.X, initial_state=self.hidden_layer,
dtype=tf.float32
        )
        # Connected dense layer to the last output of the RNN
        self.logits = tf.compat.v1.layers.dense(self.outputs[:, -1],
```

```
output_size)

        # Calculate MSE, with Adam optimizer
        self.cost = tf.reduce_mean(tf.square(self.Y - self.logits))
        self.optimizer =
tf.compat.v1.train.AdamOptimizer(learning_rate).minimize(
            self.cost
        )
```

We also defined anchor function to performs exponential smoothing on a given signal and returns a buffer containing the smoothed values. The purpose of smoothing the input signal could be to reduce noise or highlight underlying patterns in the data.Because Stock prices are known to exhibit various forms of noise, including random price fluctuations, outliers, and market volatility. Smoothing the input signal can help reduce the impact of this noise, resulting in a better representation of the underlying trends.

First, the anchor function takes a signal and a weight parameter as inputs. Then, we created an empty list called buffer to hold the smoothed values. After that, we set initial value of last to the first value in the signal list. The funtion now iterates through each value i in the signal list. Inside the loop, we calculated the smoothed value by combining the previous smoothed value (last) with the current value (i) using a weighted average. Then, we calculated smoothed value is added to the buffer list, while current smoothed value is stored in last for the next iteration. Finally, we loop until all values in the signal list are processed and return buffer list containing the smoothed values

```
def anchor(signal, weight):
    buffer = []
    last = signal[0]
    for i in signal:
        smoothed_val = last * weight + (1 - weight) * i
        buffer.append(smoothed_val)
        last = smoothed_val
    return buffer
```

Now, we define the ininitalization for our LSTM model. Simulation_size = 10

- Simulation_size = 10, means that we will make 10 different LSTM forecast
- num_layer = 1, the number of layer in our LSTM model
- size_layer = 128, means 128 hidden layer size
- timestamp = 5, which is the length or size of the input sequence used for training the LSTM model. It determines how many time steps or days of historical data are considered as input to make predictions. In this case, we use the time steps of 5 days.
- drop_out rate =0.85, regularization technique that helps prevent overfitting and improve the generalization ability of the model. Because the stock close price day by day are highly correlated, so we will randomly drop out large amount of neurons in order to preventing over-reliance on specific neurons.
- epoch = 300, which means we will do backpropagation for every Weight and Bias using the input data for 300 times. We can increase this number for better training.

- future_day = test_size = 30, which we will predict for 30 days
- learning_rate = 0.01, this determines the step size at each iteration that the optimizer takes to update the model's parameters. A higher learning rate can result in faster convergence but may also cause overshooting. However, if the learning rate is too small, it may affect the problems of slower trainning and the risk of stucking in local minimum. However, in the project, we still use the learning rate = 0.01.

```
# Set the number of simulation
simulation_size = 10

# Set the parameters for LSTM
num_layers = 1
size_layer = 128
timestamp = 5
epoch = 300
dropout_rate = 0.85
future_day = test_size
learning_rate = 0.01
```

Now we make forecast with LSTM. But first, we need to write the function to perform. The idea of function forecast is as followed:

1. We created an interactive TensorFlow session and initializes the session's variables as mentioned above
2. The date information from the dataset is converted into a datetime format.
3. A progress bar is created to monitor the training process of the LSTM model.
4. The training loop starts, iterating for a specified number of epochs. Within each epoch, the training data is processed in batches (determined by the timestamp variable above) to feed into the model. The model's logits (output), last state, optimizer, and loss are computed using the batch data and session execution.
5. The generated predictions are stored in the output_predict array, which is initialized with the first row of the training data. The loop iterates over the completed batches of the training data, predicting the next batch of data based on the previous batch. The last state of each batch is saved for the next iteration.
6. In case there are remaining data points that do not fit into a complete batch, they are processed separately.
7. If there is a need to generate predictions for future days (future_day variable), the loop continues to predict the next data points based on the last available data and the previously saved last state.
8. The predicted values are transformed back (reverse transform) to their original scale using an inverse transformation method (possibly a min-max scaling inverse transformation) and returned the predicted values for the future days.

The code below is inspired by Husein Zolkepli, in his repository named Stock-Prediction-Models (Source: https://github.com/huseinzol05/Stock-Prediction-Models)

```
def forecast():
    tf.compat.v1.reset_default_graph()
    # Set the model as the model class created before, with sepcific
initialization
    modelnn = Model(
        learning_rate, num_layers, df_log.shape[1], size_layer,
df_log.shape[1], dropout_rate
    )
    # creates an interactive TensorFlow session
    sess = tf.InteractiveSession()
    sess.run(tf.global_variables_initializer())
    # Convert the index Date to datetime format
    date_ori = pd.to_datetime(df.iloc[:, 0]).tolist()

    # Create progress bar, to check the training process of LSTM model
    pbar = tqdm(range(epoch), desc = 'train loop')
    # Training loop
    for i in pbar:
        init_value = np.zeros((1, num_layers * 2 * size_layer))
        total_loss, total_acc = [], []
        for k in range(0, df_train.shape[0] - 1, timestamp):
            index = min(k + timestamp, df_train.shape[0] - 1)
            batch_x = np.expand_dims(
                df_train.iloc[k : index, :].values, axis = 0
            )
            batch_y = df_train.iloc[k + 1 : index + 1, :].values
            logits, last_state, _, loss = sess.run(
                [modelnn.logits, modelnn.last_state,
modelnn.optimizer, modelnn.cost],
                feed_dict = {
                    modelnn.X: batch_x,
                    modelnn.Y: batch_y,
                    modelnn.hidden_layer: init_value,
                },
            )
            init_value = last_state
            total_loss.append(loss)
            total_acc.append(calculate_accuracy(batch_y[:, 0],
logits[:, 0]))
        pbar.set_postfix(cost = np.mean(total_loss), acc =
np.mean(total_acc))

    # Generate Future Predictions
    output_predict = np.zeros((df_train.shape[0] + future_day,
df_train.shape[1]))
    output_predict[0] = df_train.iloc[0]
    upper_b = (df_train.shape[0] // timestamp) * timestamp
```

```
    init_value = np.zeros((1, num_layers * 2 * size_layer))

    # Predict data in completed batches
    for k in range(0, (df_train.shape[0] // timestamp) * timestamp,
timestamp):
        out_logits, last_state = sess.run(
            [modelnn.logits, modelnn.last_state],
            feed_dict = {
                modelnn.X: np.expand_dims(
                    df_train.iloc[k : k + timestamp], axis = 0
                ),
                modelnn.hidden_layer: init_value,
            },
        )
        init_value = last_state
        output_predict[k + 1 : k + timestamp + 1] = out_logits
    # Handle Remaining Data
    if upper_b != df_train.shape[0]:
        out_logits, last_state = sess.run(
            [modelnn.logits, modelnn.last_state],
            feed_dict = {
                modelnn.X: np.expand_dims(df_train.iloc[upper_b:],
axis = 0),
                modelnn.hidden_layer: init_value,
            },
        )
        output_predict[upper_b + 1 : df_train.shape[0] + 1] =
out_logits
        future_day -= 1
        date_ori.append(date_ori[-1] + timedelta(days = 1))
    # Update Initial State
    init_value = last_state
    # Generate Further Future Predictions
    for i in range(future_day):
        o = output_predict[-future_day - timestamp + i:-future_day +
i]
        out_logits, last_state = sess.run(
            [modelnn.logits, modelnn.last_state],
            feed_dict = {
                modelnn.X: np.expand_dims(o, axis = 0),
                modelnn.hidden_layer: init_value,
            },
        )
        init_value = last_state
        output_predict[-future_day + i] = out_logits[-1]
        date_ori.append(date_ori[-1] + timedelta(days = 1))

    # Inverse Transformation:
    output_predict = minmax.inverse_transform(output_predict)
```

```
    deep_future = anchor(output_predict[:, 0], 0.3)

    return deep_future[-test_size:]
```

As the result below, there are differences accuracy and cost for each simulation, the reason is that when we calculate the loss function in Adam optimizers. The Adam optimizer uses random initialization to determine the starting point for optimization, which can lead to different optimization paths and results. Therefore, the results will be different. To address this problem, we performed multiple simulations, typically 10, with different initializations. By executing the model multiple times, we can explore a range of potential outcomes and select the most optimal forecast (base on the highest acc) among those simulations.

It is clearly stated in the results that the simulation 3 have the best results, which accacc=98.8 and cost=0.000531. Therefore, we will use simulation 3.

```
results = []
for i in range(simulation_size):
    print('-'*50)
    print('simulation %d' % (i + 1))
    results.append(forecast())
    print('-'*50)
```

```
<ipython-input-55-3740c144c911>:14: UserWarning:
`tf.nn.rnn_cell.LSTMCell` is deprecated and will be removed in a
future version. This class is equivalent as
`tf.keras.layers.LSTMCell`, and will be replaced by that in Tensorflow
2.0.
  return tf.compat.v1.nn.rnn_cell.LSTMCell(size_layer,
state_is_tuple=False)
WARNING:tensorflow:<keras.src.layers.rnn.legacy_cells.LSTMCell object
at 0x7ec067741a80>: Using a concatenated state is slower and will soon
be deprecated.  Use state_is_tuple=True.
WARNING:tensorflow:`tf.nn.rnn_cell.MultiRNNCell` is deprecated. This
class is equivalent as `tf.keras.layers.StackedRNNCells`, and will be
replaced by that in Tensorflow 2.0.
WARNING:tensorflow:From <ipython-input-55-3740c144c911>:28:
dynamic_rnn (from tensorflow.python.ops.rnn) is deprecated and will be
removed in a future version.
Instructions for updating:
Please use `keras.layers.RNN(cell)`, which is equivalent to this API

--------------------------------------------------
simulation 1

WARNING:tensorflow:From
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/legacy_ce
lls.py:1043: calling Zeros.__init__ (from
tensorflow.python.ops.init_ops) with dtype is deprecated and will be
removed in a future version.
Instructions for updating:
```

```
Call initializer instance with the dtype argument instead of passing
it to the constructor
<ipython-input-55-3740c144c911>:31: UserWarning: `tf.layers.dense` is
deprecated and will be removed in a future version. Please use
`tf.keras.layers.Dense` instead.
  self.logits = tf.compat.v1.layers.dense(self.outputs[:, -1],
output_size)
train loop: 100%|██████████| 300/300 [26:04<00:00,  5.21s/it,
acc=98.7, cost=0.000642]
<ipython-input-55-3740c144c911>:14: UserWarning:
`tf.nn.rnn_cell.LSTMCell` is deprecated and will be removed in a
future version. This class is equivalent as
`tf.keras.layers.LSTMCell`, and will be replaced by that in Tensorflow
2.0.
  return tf.compat.v1.nn.rnn_cell.LSTMCell(size_layer,
state_is_tuple=False)
WARNING:tensorflow:<keras.src.layers.rnn.legacy_cells.LSTMCell object
at 0x7ec064adead0>: Using a concatenated state is slower and will soon
be deprecated.  Use state_is_tuple=True.
WARNING:tensorflow:`tf.nn.rnn_cell.MultiRNNCell` is deprecated. This
class is equivalent as `tf.keras.layers.StackedRNNCells`, and will be
replaced by that in Tensorflow 2.0.

--------------------------------------------------
--------------------------------------------------
simulation 2

<ipython-input-55-3740c144c911>:31: UserWarning: `tf.layers.dense` is
deprecated and will be removed in a future version. Please use
`tf.keras.layers.Dense` instead.
  self.logits = tf.compat.v1.layers.dense(self.outputs[:, -1],
output_size)
/usr/local/lib/python3.10/dist-packages/tensorflow/python/client/sessi
on.py:1793: UserWarning: An interactive session is already active.
This can cause out-of-memory errors in some cases. You must explicitly
call `InteractiveSession.close()` to release resources held by the
other session(s).
  warnings.warn('An interactive session is already active. This can '
train loop: 100%|██████████| 300/300 [23:44<00:00,  4.75s/it,
acc=98.6, cost=0.000747]
<ipython-input-55-3740c144c911>:14: UserWarning:
`tf.nn.rnn_cell.LSTMCell` is deprecated and will be removed in a
future version. This class is equivalent as
`tf.keras.layers.LSTMCell`, and will be replaced by that in Tensorflow
2.0.
  return tf.compat.v1.nn.rnn_cell.LSTMCell(size_layer,
state_is_tuple=False)
WARNING:tensorflow:<keras.src.layers.rnn.legacy_cells.LSTMCell object
at 0x7ec064b097e0>: Using a concatenated state is slower and will soon
be deprecated.  Use state_is_tuple=True.
```

```
WARNING:tensorflow:`tf.nn.rnn_cell.MultiRNNCell` is deprecated. This
class is equivalent as `tf.keras.layers.StackedRNNCells`, and will be
replaced by that in Tensorflow 2.0.

--------------------------------------------------
--------------------------------------------------
simulation 3

<ipython-input-55-3740c144c911>:31: UserWarning: `tf.layers.dense` is
deprecated and will be removed in a future version. Please use
`tf.keras.layers.Dense` instead.
  self.logits = tf.compat.v1.layers.dense(self.outputs[:, -1],
output_size)
/usr/local/lib/python3.10/dist-packages/tensorflow/python/client/sessi
on.py:1793: UserWarning: An interactive session is already active.
This can cause out-of-memory errors in some cases. You must explicitly
call `InteractiveSession.close()` to release resources held by the
other session(s).
  warnings.warn('An interactive session is already active. This can '
train loop: 100%|██████████| 300/300 [22:56<00:00,  4.59s/it,
acc=98.8, cost=0.000531]
<ipython-input-55-3740c144c911>:14: UserWarning:
`tf.nn.rnn_cell.LSTMCell` is deprecated and will be removed in a
future version. This class is equivalent as
`tf.keras.layers.LSTMCell`, and will be replaced by that in Tensorflow
2.0.
  return tf.compat.v1.nn.rnn_cell.LSTMCell(size_layer,
state_is_tuple=False)
WARNING:tensorflow:<keras.src.layers.rnn.legacy_cells.LSTMCell object
at 0x7ec05e73f2b0>: Using a concatenated state is slower and will soon
be deprecated.  Use state_is_tuple=True.
WARNING:tensorflow:`tf.nn.rnn_cell.MultiRNNCell` is deprecated. This
class is equivalent as `tf.keras.layers.StackedRNNCells`, and will be
replaced by that in Tensorflow 2.0.

--------------------------------------------------
--------------------------------------------------
simulation 4

<ipython-input-55-3740c144c911>:31: UserWarning: `tf.layers.dense` is
deprecated and will be removed in a future version. Please use
`tf.keras.layers.Dense` instead.
  self.logits = tf.compat.v1.layers.dense(self.outputs[:, -1],
output_size)
/usr/local/lib/python3.10/dist-packages/tensorflow/python/client/sessi
on.py:1793: UserWarning: An interactive session is already active.
This can cause out-of-memory errors in some cases. You must explicitly
call `InteractiveSession.close()` to release resources held by the
other session(s).
  warnings.warn('An interactive session is already active. This can '
```

```
train loop: 100%|██████████| 300/300 [25:11<00:00,  5.04s/it,
acc=98.8, cost=0.000588]
<ipython-input-55-3740c144c911>:14: UserWarning:
`tf.nn.rnn_cell.LSTMCell` is deprecated and will be removed in a
future version. This class is equivalent as
`tf.keras.layers.LSTMCell`, and will be replaced by that in Tensorflow
2.0.
  return tf.compat.v1.nn.rnn_cell.LSTMCell(size_layer,
state_is_tuple=False)
WARNING:tensorflow:<keras.src.layers.rnn.legacy_cells.LSTMCell object
at 0x7ec0684b0790>: Using a concatenated state is slower and will soon
be deprecated.  Use state_is_tuple=True.
WARNING:tensorflow:`tf.nn.rnn_cell.MultiRNNCell` is deprecated. This
class is equivalent as `tf.keras.layers.StackedRNNCells`, and will be
replaced by that in Tensorflow 2.0.

--------------------------------------------------
--------------------------------------------------
simulation 5

<ipython-input-55-3740c144c911>:31: UserWarning: `tf.layers.dense` is
deprecated and will be removed in a future version. Please use
`tf.keras.layers.Dense` instead.
  self.logits = tf.compat.v1.layers.dense(self.outputs[:, -1],
output_size)
/usr/local/lib/python3.10/dist-packages/tensorflow/python/client/sessi
on.py:1793: UserWarning: An interactive session is already active.
This can cause out-of-memory errors in some cases. You must explicitly
call `InteractiveSession.close()` to release resources held by the
other session(s).
  warnings.warn('An interactive session is already active. This can '
train loop: 100%|██████████| 300/300 [23:42<00:00,  4.74s/it,
acc=98.7, cost=0.000656]
<ipython-input-55-3740c144c911>:14: UserWarning:
`tf.nn.rnn_cell.LSTMCell` is deprecated and will be removed in a
future version. This class is equivalent as
`tf.keras.layers.LSTMCell`, and will be replaced by that in Tensorflow
2.0.
  return tf.compat.v1.nn.rnn_cell.LSTMCell(size_layer,
state_is_tuple=False)
WARNING:tensorflow:<keras.src.layers.rnn.legacy_cells.LSTMCell object
at 0x7ec0640e1a80>: Using a concatenated state is slower and will soon
be deprecated.  Use state_is_tuple=True.
WARNING:tensorflow:`tf.nn.rnn_cell.MultiRNNCell` is deprecated. This
class is equivalent as `tf.keras.layers.StackedRNNCells`, and will be
replaced by that in Tensorflow 2.0.

--------------------------------------------------
--------------------------------------------------
simulation 6
```

```
<ipython-input-55-3740c144c911>:31: UserWarning: `tf.layers.dense` is
deprecated and will be removed in a future version. Please use
`tf.keras.layers.Dense` instead.
  self.logits = tf.compat.v1.layers.dense(self.outputs[:, -1],
output_size)
/usr/local/lib/python3.10/dist-packages/tensorflow/python/client/sessi
on.py:1793: UserWarning: An interactive session is already active.
This can cause out-of-memory errors in some cases. You must explicitly
call `InteractiveSession.close()` to release resources held by the
other session(s).
  warnings.warn('An interactive session is already active. This can '
train loop: 100%|██████████| 300/300 [23:42<00:00,  4.74s/it,
acc=98.7, cost=0.000608]
<ipython-input-55-3740c144c911>:14: UserWarning:
`tf.nn.rnn_cell.LSTMCell` is deprecated and will be removed in a
future version. This class is equivalent as
`tf.keras.layers.LSTMCell`, and will be replaced by that in Tensorflow
2.0.
  return tf.compat.v1.nn.rnn_cell.LSTMCell(size_layer,
state_is_tuple=False)
WARNING:tensorflow:<keras.src.layers.rnn.legacy_cells.LSTMCell object
at 0x7ec067ff84f0>: Using a concatenated state is slower and will soon
be deprecated.  Use state_is_tuple=True.
WARNING:tensorflow:`tf.nn.rnn_cell.MultiRNNCell` is deprecated. This
class is equivalent as `tf.keras.layers.StackedRNNCells`, and will be
replaced by that in Tensorflow 2.0.

--------------------------------------------------
--------------------------------------------------
simulation 7

<ipython-input-55-3740c144c911>:31: UserWarning: `tf.layers.dense` is
deprecated and will be removed in a future version. Please use
`tf.keras.layers.Dense` instead.
  self.logits = tf.compat.v1.layers.dense(self.outputs[:, -1],
output_size)
/usr/local/lib/python3.10/dist-packages/tensorflow/python/client/sessi
on.py:1793: UserWarning: An interactive session is already active.
This can cause out-of-memory errors in some cases. You must explicitly
call `InteractiveSession.close()` to release resources held by the
other session(s).
  warnings.warn('An interactive session is already active. This can '
train loop: 100%|██████████| 300/300 [24:34<00:00,  4.92s/it,
acc=98.8, cost=0.000538]
<ipython-input-55-3740c144c911>:14: UserWarning:
`tf.nn.rnn_cell.LSTMCell` is deprecated and will be removed in a
future version. This class is equivalent as
`tf.keras.layers.LSTMCell`, and will be replaced by that in Tensorflow
2.0.
  return tf.compat.v1.nn.rnn_cell.LSTMCell(size_layer,
```

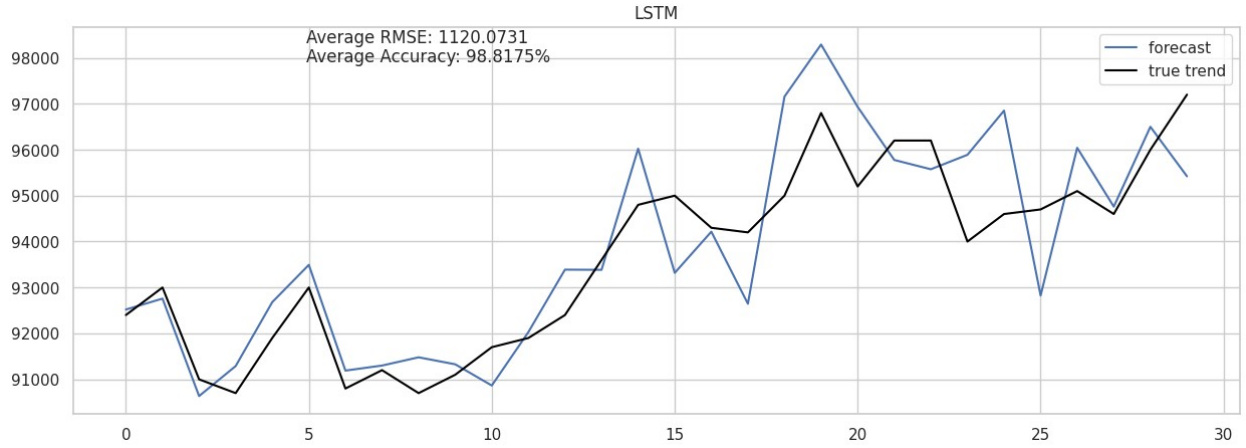
```
state_is_tuple=False)
WARNING:tensorflow:<keras.src.layers.rnn.legacy_cells.LSTMCell object
at 0x7ec0681c7e50>: Using a concatenated state is slower and will soon
be deprecated.  Use state_is_tuple=True.
WARNING:tensorflow:`tf.nn.rnn_cell.MultiRNNCell` is deprecated. This
class is equivalent as `tf.keras.layers.StackedRNNCells`, and will be
replaced by that in Tensorflow 2.0.

--------------------------------------------------
--------------------------------------------------
simulation 8

<ipython-input-55-3740c144c911>:31: UserWarning: `tf.layers.dense` is
deprecated and will be removed in a future version. Please use
`tf.keras.layers.Dense` instead.
  self.logits = tf.compat.v1.layers.dense(self.outputs[:, -1],
output_size)
/usr/local/lib/python3.10/dist-packages/tensorflow/python/client/sessi
on.py:1793: UserWarning: An interactive session is already active.
This can cause out-of-memory errors in some cases. You must explicitly
call `InteractiveSession.close()` to release resources held by the
other session(s).
  warnings.warn('An interactive session is already active. This can '
train loop: 100%|██████████| 300/300 [21:54<00:00,  4.38s/it,
acc=98.7, cost=0.000558]
<ipython-input-55-3740c144c911>:14: UserWarning:
`tf.nn.rnn_cell.LSTMCell` is deprecated and will be removed in a
future version. This class is equivalent as
`tf.keras.layers.LSTMCell`, and will be replaced by that in Tensorflow
2.0.
  return tf.compat.v1.nn.rnn_cell.LSTMCell(size_layer,
state_is_tuple=False)
WARNING:tensorflow:<keras.src.layers.rnn.legacy_cells.LSTMCell object
at 0x7ec05e3eaa10>: Using a concatenated state is slower and will soon
be deprecated.  Use state_is_tuple=True.
WARNING:tensorflow:`tf.nn.rnn_cell.MultiRNNCell` is deprecated. This
class is equivalent as `tf.keras.layers.StackedRNNCells`, and will be
replaced by that in Tensorflow 2.0.

--------------------------------------------------
--------------------------------------------------
simulation 9

<ipython-input-55-3740c144c911>:31: UserWarning: `tf.layers.dense` is
deprecated and will be removed in a future version. Please use
`tf.keras.layers.Dense` instead.
  self.logits = tf.compat.v1.layers.dense(self.outputs[:, -1],
output_size)
/usr/local/lib/python3.10/dist-packages/tensorflow/python/client/sessi
on.py:1793: UserWarning: An interactive session is already active.
```

```
This can cause out-of-memory errors in some cases. You must explicitly
call `InteractiveSession.close()` to release resources held by the
other session(s).
  warnings.warn('An interactive session is already active. This can '
train loop: 100%|██████████| 300/300 [20:55<00:00,  4.19s/it, acc=-
110, cost=10.4]
<ipython-input-55-3740c144c911>:14: UserWarning:
`tf.nn.rnn_cell.LSTMCell` is deprecated and will be removed in a
future version. This class is equivalent as
`tf.keras.layers.LSTMCell`, and will be replaced by that in Tensorflow
2.0.
  return tf.compat.v1.nn.rnn_cell.LSTMCell(size_layer,
state_is_tuple=False)
WARNING:tensorflow:<keras.src.layers.rnn.legacy_cells.LSTMCell object
at 0x7ec05e090dc0>: Using a concatenated state is slower and will soon
be deprecated.  Use state_is_tuple=True.
WARNING:tensorflow:`tf.nn.rnn_cell.MultiRNNCell` is deprecated. This
class is equivalent as `tf.keras.layers.StackedRNNCells`, and will be
replaced by that in Tensorflow 2.0.

--------------------------------------------------
--------------------------------------------------
simulation 10

<ipython-input-55-3740c144c911>:31: UserWarning: `tf.layers.dense` is
deprecated and will be removed in a future version. Please use
`tf.keras.layers.Dense` instead.
  self.logits = tf.compat.v1.layers.dense(self.outputs[:, -1],
output_size)
/usr/local/lib/python3.10/dist-packages/tensorflow/python/client/sessi
on.py:1793: UserWarning: An interactive session is already active.
This can cause out-of-memory errors in some cases. You must explicitly
call `InteractiveSession.close()` to release resources held by the
other session(s).
  warnings.warn('An interactive session is already active. This can '
train loop: 100%|██████████| 300/300 [21:31<00:00,  4.31s/it,
acc=96.8, cost=0.00423]

--------------------------------------------------
```

Then we calculated the performance metrics of the best simulation, including accuracy and RMSE. The idea of the code below is the same as the idea for calculating performance metrics and plotting the data in the Random Forest model that had been explained above.

The results indicated that the accuracy and RMSE are 98.8175 and 1120.0731 respetively. The graph illustrates that the predicted line closely tracks the actual values of the FPT stock's close price. This indicates that the model is capable of capturing the underlying patterns and trends in the data, resulting in accurate predictions in the short term. However, it is observed that as the prediction extends further into the future, the predicted values start to deviate more from the actual values.

```
# Calculate performance metrics
accuracy = calculate_accuracy(df['Close'].iloc[-test_size:].values,
results[2])
rmse = calculate_rmse(df['Close'].iloc[-test_size:].values,
results[2])

# Plotting the results
plt.figure(figsize = (15, 5))
# for no, r in enumerate(results):
plt.plot(results[2], label = 'forecast ')
plt.plot(df['Close'].iloc[-test_size:].values, label = 'true trend', c
= 'black')

rmse_text = f"Average RMSE: {np.mean(rmse):.4f}"
accuracy_text = f"Average Accuracy: {np.mean(accuracy):.4f}%"

plt.text(0.2, 0.95, rmse_text, transform=plt.gca().transAxes,
fontsize=12, verticalalignment='bottom')
plt.text(0.2, 0.9, accuracy_text, transform=plt.gca().transAxes,
fontsize=12, verticalalignment='bottom')

plt.legend()
plt.title("LSTM")
plt.show()
```




```
# This code is used to store our performance metrics to compare
between the models, in the later part
LSTM = pd.DataFrame({'Model': ['LSTM'], 'RMSE': [rmse], 'Accuracy':
[accuracy]})
```

Conclusion:

1. In LSTM model, it is common to apply MinMax scaling to the input data as a preprocessing step. This scaling is performed because the backpropagation algorithm

used in LSTM models is most effective when working with input values within a specific range.

2. The initial parameters for LSTM are num_layers = 1, size_layer = 128 timestamp = 5, epoch = 300, dropout_rate = 0.85, future_day = test_size,learning_rate = 0.01, of which definition have already been explained above.
3. The Adam optimizer uses random initialization to determine the starting point for optimization, which can lead to different optimization paths and results. So we ran 10 simnulations and choose the best one among them.
4. The predicted values closely align with the actual values, this is align with the value of RMSE and accuracy of 98.8175 and 1120.0731 respectively. However, as the forecast horizon extends further into the future, the predicted values gradually exhibit greater divergence from the actual values.

#2.6: Interpreting and Comparing the results

```
comparison = pd.concat([comparison, arima, RANDOMFOREST,
RANDOMFOREST_TUNING, LSTM], ignore_index=True)
comparison.head()
```

	Model	RMSE	Accuracy
0	ARIMA	2259.242801	97.627811
1	Random Forest	1930.094636	97.942354
2	Random Forest Tuning	1479.272973	98.425579
3	LSTM	1120.073131	98.817518

Interpreting the resultt

In terms of ARIMA model, The obtained results indicate a high level of accuracy, with an accuracy rate of approximately 97.63%, and a relatively low root mean squared error (RMSE) of approximately 2259.24. These findings suggest that there is generally a small discrepancy between the predicted values generated by the ARIMA model and the actual values. However, upon closer examination of the plot displaying the true values and the predicted values generated by the ARIMA model, it becomes evident that the forecasted trend of the ARIMA model does not closely align with the actual trend of the data. This observation suggests that the ARIMA model may not accurately capture the underlying trend or pattern of the actual values. These findings indicate that while the ARIMA model may exhibit satisfactory overall performance in terms of accuracy and error metrics, it may struggle to capture the complex dynamics or long-term trends present in the data.

Secondly, with Random Forest Regressor, the predicted line closely tracks the actual values of the FPT stock's close price. This indicates that the model is capable of capturing the underlying patterns and trends in the data, resulting in accurate predictions. The original model evaluation yielded a root mean squared error (RMSE) and accuracy score was approximated at 1930.09 and 97.94%. And after hyperparameter tuning procedures were applied to refine model configuration, RMSE saw a sizeable decrease to 1479.27, representing a notable improvement in error reduction. However, it is observed that as the prediction extends further into the future, the predicted values start to deviate more from the actual values.

Finally, for LSTM regressors. the predicted values closely align with the actual values, this is align with the value of RMSE and accuracy of approximately 1120 and 98.81 respectively.

However, similarly to Random forest, the predictions errors increase when the time we make forecast increase. This has been explained by Box and Geogre (2015) in the book " Time series analysis: forecasting and control", they stated that is because the stock market have phenomenon characteristics called volatility clustering, that is, periods in which they exhibit wide swings for an extended time period followed by a period of comparative tranquility .

Comparision

The results clearly indicate that the Deep Learning and Machine Learning models, specifically LSTM and Random Forest, outperform the traditional econometric approach ARIMA in terms of accuracy and RMSE. Among these models, LSTM demonstrates the highest performance, achieving an RMSE of X and an accuracy of Y. On the other hand, ARIMA exhibits the poorest performance, with an accuracy of A and an RMSE of B. Additionally, it is worth noting that LSTM RNN serves as the fundamental LSTM model, yet it produces the best results. By exploring different variations of LSTM models and conducting parameter tuning, we can potentially achieve even better outcomes.

Deep learning model have several advantages over traditional econometric approaches like ARIMA. The main explaination is that deep learning models can capture complex non-linear relationships in the data by activation function, which is often the case in real-world scenarios. ARIMA, on the other hand, assumes linearity and may struggle to capture non-linear patterns effectively. Furthurmore, LSTM, are designed to capture long-term dependencies in sequential data. They have a memory mechanism that allows them to retain information from earlier time steps and utilize it when making predictions.

CHAPTER 3: CONCLUSION

3.1 Conclusion

In terms of ARIMA model, The obtained results indicate a high level of accuracy, with an accuracy rate of approximately 97.63%, and a relatively low root mean squared error (RMSE) of approximately 2259.24. These findings suggest that there is generally a small discrepancy between the predicted values generated by the ARIMA model and the actual values. However, upon closer examination of the plot displaying the true values and the predicted values generated by the ARIMA model, it becomes evident that the forecasted trend of the ARIMA model does not closely align with the actual trend of the data. This observation suggests that the ARIMA model may not accurately capture the underlying trend or pattern of the actual values. These findings indicate that while the ARIMA model may exhibit satisfactory overall performance in terms of accuracy and error metrics, it may struggle to capture the complex dynamics or long-term trends present in the data.

Secondly, with Random Forest Regressor, the predicted line closely tracks the actual values of the FPT stock's close price. This indicates that the model is capable of capturing the underlying patterns and trends in the data, resulting in accurate predictions. The original model evaluation yielded a root mean squared error (RMSE) and accuracy score was approximated at 1930.09 and 97.94%. And after hyper-parameter tuning procedures were applied to refine model configuration, RMSE saw a sizeable decrease to 1479.27, representing a notable improvement in error reduction. However, it is observed that as the prediction extends further into the future, the predicted values start to deviate more from the actual values.

Finally, for LSTM regressors. the predicted values closely align with the actual values, this is align with the value of RMSE and accuracy of approximately 1120 and 98.81 respectively. However, similarly to Random forest, the predictions errors increase when the time we make forecast increase. This has been explained by Box and George (2015) in the book " Time series analysis: forecasting and control", they stated that is because the stock market have phenomenon characteristics called volatility

clustering, that is, periods in which they exhibit wide swings for an extended time period followed by a period of comparative tranquility.

The results clearly indicate that the Deep Learning and Machine Learning models, specifically LSTM and Random Forest, outperform the traditional econometric approach ARIMA in terms of accuracy and RMSE. Among these models, LSTM demonstrates the highest performance, achieving an RMSE of X and an accuracy of Y. On the other hand, ARIMA exhibits the poorest performance, with an accuracy of A and an RMSE of B. Additionally, it is worth noting that LSTM RNN serves as the fundamental LSTM model, yet it produces the best results. By exploring different variations of LSTM models and conducting parameter tuning, we can potentially achieve even better outcomes.

Deep learning model have several advantages over traditional econometric approaches like ARIMA. The main explanation is that deep learning models can capture complex non-linear relationships in the data by activation function, which is often the case in real-world scenarios. ARIMA, on the other hand, assumes linearity and may struggle to capture non-linear patterns effectively. Furthermore, LSTM, are designed to capture long-term dependencies in sequential data. They have a memory mechanism that allows them to retain information from earlier time steps and utilize it when making predictions.

Stock market prediction is so difficult due to its nonlinear, dynamic and complicated nature; yet, it is so important. Successful prediction has some interesting benefits that usually affect the decision of a financial trader on the purchase or sale of a financial instrument.

3.2 Limitations & Further perspective

The authors have also proposed some limitations and points of view for future project:

Firstly, the primary focus of the project was solely on utilizing time series data for stock price prediction. Therefore, the impact of other influential factors (such as

international market indexes, bulk commodity prices, breaking financial news, and investors behaviors) on forecasting stock prices was not explored. Future research endeavors should consider incorporating additional factors, such as public sentiment analysis of social networks, which can potentially influence stock market behavior. Integrating these factors into neural network models can enhance the accuracy and the meaning of stock price predictions.

Secondly, it is recommended to explore various types of LSTM models. These include stacked LSTMs, GRU LSTMs, bidirectional LSTMs, CNN LSTMs, stc. Investigating these different models can provide valuable information and suitability for stock price prediction tasks. It is highly noted that there is no universally applicable neural network model that can accurately predict the prices of all stocks in the market. By comparing and analyzing the results obtained from those diverse LSTM models, researchers can evaluate and find out the most effective models for predicting stock prices of a specific corporation.

Finally, the stock market, particularly in emerging markets like Vietnam, where regulatory measures may not be as stringent, there is space wherein large investors can make significant influence over the stock market. Therefore, it is challenging to rely solely on artificial intelligence for accurate predictions. To achieve greater accuracy, researchers should consider selecting stocks that are influenced by supply and demand, or alternatively, combine fundamental analysis with domain knowledge. By incorporating these approaches, researchers can leverage both quantitative and qualitative factors to enhance the accuracy of stock price predictions.

REFERENCES

1. Bollen, J., Mao, H., & Zeng, X. (2011). Twitter mood predicts the stock market. *Journal of Computational Science, 2*(1), 1-8.
2. Bormpotsis, C., Sedky, M., & Patel, A. (2023). Predicting Forex Currency Fluctuations Using a Novel Bio-Inspired Modular Neural Network. *Big Data and Cognitive Computing, 7*(3), 152.
3. Box, G. E., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time series analysis: forecasting and control.* John Wiley & Sons.
4. Chan, H. L., Lam, K. H., & Wong, H. Y. (2021). Stock price prediction using a hybrid model based on ARIMA, GARCH, and support vector regression. *Journal of Forecasting, 40*(5), 681-698.
5. Choijil, E., Méndez, C. E., Wong, W. K., Vieito, J. P., & Batmunkh, M. U. (2022). Thirty years of herd behavior in financial markets: A bibliometric analysis. *Research in International Business and Finance, 59*, 101506.
6. Damodaran, A. (2020). *Investment valuation: Tools and techniques for determining the value of any asset* (4th ed.). Wiley.
7. Grosse, R. (2017). Lecture 15: Exploding and vanishing gradients. *University of Toronto Computer Science.*
8. Gujarati, D. N., & Porter, D. C. (2009). *Basic econometrics*. McGraw-hill.
9. Hu, Z., Zhao, Y., & Khushi, M. (2021). A Survey of Forex and Stock Price Prediction Using Deep Learning. *Applied System Innovation, 4*(1), 9.
10. IBM, What are neural networks?, Available at: www.ibm.com/topics/neural-networks. Accessed 31 December, 2023.
11. Kelleher, J. D., & Tierney, B. (2018). *Data science*. MIT Press.
12. Krishna, S. H., Oyebode, O. J., Joshi, A., Reddy, G. S., Karthikeyan, P., & Ganesh, K. V. B. (2023, May). The technical Role of Regression Analysis in prediction and decision making. In *2023 3rd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)* (pp. 183-187). IEEE.

13. Kumar, A., & Kumar, A. (2021). Stock price prediction using LSTM and GRU based deep learning models. *International Journal of Information Management, 57*, 102297.
14. Lee, C. J. (1987). Fundamental analysis and the stock market. *Journal of Business Finance & Accounting, 14*(1), 131-141.
15. Lipton, Z. C., Berkowitz, J., & Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning.
16. Liu, M., Choo, W. C., Lee, C. C., & Lee, C. C. (2023). Trading volume and realized volatility forecasting: Evidence from the China stock market. *Journal of Forecasting, 42*(1), 76-100.
17. Long short-term memory networks (LSTM)- simply explained!. Data Basecamp. (2023, May 13). https://databasecamp.de/en/ml/lstms
18. Omar, A., & El-Hafeez, A. (2023). Optimizing epileptic seizure recognition performance with feature scaling and dropout layers. *Neural Computing and Applications*, 1-18.
19. Panhalkar, A. R., & Doye, D. D. (2022). A novel approach to build accurate and diverse decision tree forest. *Evolutionary Intelligence, 15*(1), 439-453.
20. Prion, S. K., & Haerling, K. A. (2020). Making Sense of Methods and Measurements: Simple Linear Regression. *Clinical Simulation in Nursing, 48*, 94-95.
21. S. Saini and V. Sahula, "Neural Machine Translation for English to Hindi," 2018 *Fourth International Conference on Information Retrieval and Knowledge Management (CAMP)*, Kota Kinabalu, Malaysia, 2018, pp. 1-6, doi: 10.1109/INFRKM.2018.8464781.
22. Schumaker, R. P., & Chen, H. (2009). Textual analysis of stock market prediction using breaking financial news: The AZFin text system. *ACM Transactions on Information Systems (TOIS), 27*(2), 12.

23. Seddik, S., Routaib, H., & Elhaddadi, A. (2023). Multi-variable time series decoding with Long Short-Term Memory and mixture attention. *Acadlore Transactions on AI and Machine Learning*, 2(3), 154-169.
24. Sharma, V. (2022). A Study on Data Scaling Methods for Machine Learning. *International Journal for Global Academic & Scientific Research, 1*(1), 23-33.
25. Teweles, R. J., & Bradley, E. S. (1998). *The stock market* (Vol. 64). John Wiley & Sons.
26. Tran, V. (2022). Stock market efficiency responses to strategic stock exchange regulatory changes: evidence from Vietnam (Doctoral dissertation, University of Sunderland).
27. Yadav, A., Jha, C. K., & Sharan, A. (2020). Optimizing LSTM for time series prediction in Indian stock market. *Procedia Computer Science*, 167, 2091-2100.
28. Yoo, P. D., Kim, M. H., & Jan, T. (2005, November). Machine learning techniques and use of event information for stock market prediction: A survey and evaluation. In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*(Vol. 2, pp. 835-841). IEEE.
29. Yu, Y., Si, X., Hu, C., & Zhang, J. (2019). A review of recurrent neural networks: LSTM cells and network architectures. *Neural computation*, 31(7), 1235-1270.