STOCK PRICE FORECASTING

MSDA 3999: Capstone Practicum

Field Project

Author:
Kruthik Reddy Theepireddy
Jaya Chandra Kadiveti
Girish Reddy Maligireddy

Advisor: Prof. Khalid Aboalayon

Masters in Data Analytics

Clark University
05/03/2024

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

DECLARATION

We hereby declare that this project entitled '**Stock Price Prediction**' is entirely our own work and it has never been submitted nor is it currently being submitted for any other degree.

Signed: Kruthik Reddy Theepireddy

Jaya Chandra Kadiveti

Girish Reddy Maligireddy

Date: 3rd May 2024

ACKNOWLEDGEMENTS

We are deeply grateful to **Prof. Khalid Aboalayon** for his crucial advice and assistance during the creation of our project, which is stock price prediction. His knowledge has been crucial in determining the project's course and raising the project's overall standard. In addition, we would like to thank Yahoo Finance for providing the main data source for our project and for its contributions. We also appreciate the constructive feedback from other learners like us, which considerably improved the quality of our study.

ABSTRACT

Forecasting stock prices accurately is a fundamental yet challenging task in financial markets. This report presents the development and implementation of a stock price forecasting model using machine learning techniques, aimed at providing investors and traders with reliable predictions to navigate market uncertainties effectively. Leveraging real-time stock data from Yahoo Finance, various machine learning algorithms including Random Forest, LSTM, ARIMA, and GRU were implemented and evaluated. The models were assessed based on metrics such as Root Mean Squared Error (RMSE) to determine their predictive accuracy. Additionally, exploratory data analysis techniques were employed to understand underlying patterns and relationships in the stock data. The study aims to forecast potential future price changes of NASDAQ stocks using machine learning techniques such as Gated Recurrent Units (GRU), Autoregressive Integrated Moving Average (ARIMA), among others. By training models with historical price change data, the project seeks to enhance the accuracy of stock price prediction beyond the current benchmark of 68% accuracy reported in existing literature. Key findings are expected to contribute to advancements in stock market prediction methodologies, offering valuable insights for investors and financial analysts. The report concludes with insights into lessons learned and suggestions for future enhancements, emphasizing the importance of incorporating macroeconomic indicators and risk management techniques for more robust forecasting models.

Keywords: NASDAQ; Machine Learning; GRU; LSTM; ARIMA; Stocks; Forecast.

CHAPTER ONE INTRODUCTION

In this introduction, we present an outline of the project's origins, objectives, and research issues, laying the groundwork for a thorough examination of stock market prediction methods. The project aims to contribute to the progress of predictive modelling approaches by conducting systematic study and evaluation, as well as providing practical insights for investors, analysts, and researchers navigating financial markets.

1.1 Background

The contemporary stock market, particularly the NASDAQ exchange, is renowned for its dynamic nature, characterized by swift fluctuations and uncertainties. Investors and financial analysts continuously seek robust methodologies to forecast future price changes accurately. This project emerges from this context, aiming to address the challenge of forecasting NASDAQ stock prices with enhanced precision.

Previous research in the field has delved into various approaches, including traditional econometric models. However, while these methodologies have exhibited promise, there remains a notable gap in achieving consistently high accuracy rates in stock price predictions. Improving the accuracy of such forecasts holds significant implications for investment decisions, risk management strategies, and overall market efficiency.

1.2 Statement of the Problem

The accurate estimation of future stock prices has long been a challenging endeavor within the realm of financial forecasting. Traditional methods employed for this purpose often fall

short in capturing the intricate dynamics of the market, resulting in unreliable forecasts. This discrepancy poses significant challenges for finance professionals who rely on accurate predictions for strategic decision-making.

- **Complex Market Dynamics**: The stock market is characterized by a myriad of factors and variables that influence price movements, making it inherently complex to model accurately.
- **Limitations of Traditional Methods**: Conventional forecasting methods, such as time series analysis and fundamental analysis, often fail to account for the dynamic and non-linear nature of market behavior, leading to inaccurate predictions.
- **Unreliable Forecasts**: The inability of traditional methods to adequately capture market dynamics results in forecasts that are often unreliable and may lead to suboptimal investment decisions.

1.3 Purpose and Significance

Purpose:

The primary objective of this undertaking is to address the exigent requirement within the financial domain for precision and reliability in forecasting stock price variations, particularly within the dynamic milieu of the NASDAQ exchange. By harnessing advanced machine learning methodologies and conducting comprehensive data analysis, the project endeavors to formulate a forecasting model adept at furnishing investors, financial analysts, and market stakeholders with timely and accurate forecasts of NASDAQ stock price fluctuations.

Significance:

The significance of this endeavor is underscored by its potential to revolutionize prevailing stock market forecasting paradigms and substantially augment decision-making processes within the financial sector. By bridging the chasm between theoretical discourse and pragmatic application, the outcomes of this project bear profound implications, including:

Augmented Investment Strategies: The development of a meticulously accurate forecasting model empowers investors to execute judicious investment decisions, thereby optimizing returns and curtailing risks associated with stock market volatility.

Elevated Risk Mitigation: By furnishing dependable forecasts of stock price shifts, the forecasting model facilitates the formulation of resilient risk management strategies, enabling market participants to mitigate potential losses and fortify their investment portfolios.

Enhanced Market Efficiency: The availability of precise and timely stock price forecasts fosters heightened market efficiency by expediting transactions, diminishing market inefficiencies, and fostering equitable pricing mechanisms.

Advancement of Financial Scholarship: Through its innovative methodologies and comprehensive analyses, the project contributes to the progression of knowledge within the realm of stock market forecasting, igniting further research endeavors and informing industry benchmarks.

In summary, the creation of a pioneering forecasting model tailored to the NASDAQ exchange holds promise in transforming the landscape of stock market prognostications, proffering invaluable insights and opportunities for stakeholders traversing the financial terrain.

CHAPTER TWO LITERATURE REVIEW

2.1 Stock Closing Price Prediction using Machine Learning Techniques

A study conducted by Mehar Vijh et al. in 2020 explored the application of machine learning techniques for predicting stock closing prices. The research focused on utilizing the Random Forest algorithm, a popular ensemble learning method known for its robustness and flexibility.

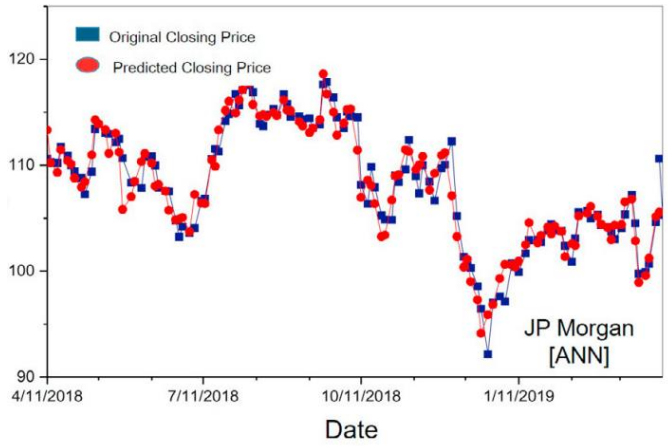


Fig. 1. Predicted v/s original (expected) closing stock price using RF

2.1.1 Methodology

- **Algorithm Selection:** The study opted for the Random Forest algorithm due to its ability to handle large datasets with high dimensionality and nonlinear relationships effectively.
- **Data Utilization:** Close values of stock prices were used both for training the model and for making predictions. By focusing on this specific aspect of stock data, the model aimed to capture the most relevant information for forecasting future closing prices.

- **Performance Evaluation:** The performance of the predictive model was assessed using the Root Mean Squared Error (RMSE) metric. The reported RMSE values ranged from 1.10 to 3.30, indicating the model's ability to accurately predict stock closing prices within a certain margin of error.
- **Data Splitting Technique:** The study employed a windowing method for splitting the data into training and testing sets. This technique involves dividing the dataset into sequential windows, with each window containing a subset of the data for training and validation purposes.

Company	RMSE
Nike	1.10
Goldman Sachs	3.30
JP Morgan and Co.	1.28
Johnson & Johnson	1.54

Fig. 2. RMSE values obtained by Random Forest

2.2 Stock Market Predictions with LSTM in Python

Avijeet Bishwal's work in 2024 delves into the utilization of Long Short-Term Memory (LSTM) networks for stock market predictions, showcasing advancements in machine learning techniques for financial forecasting.

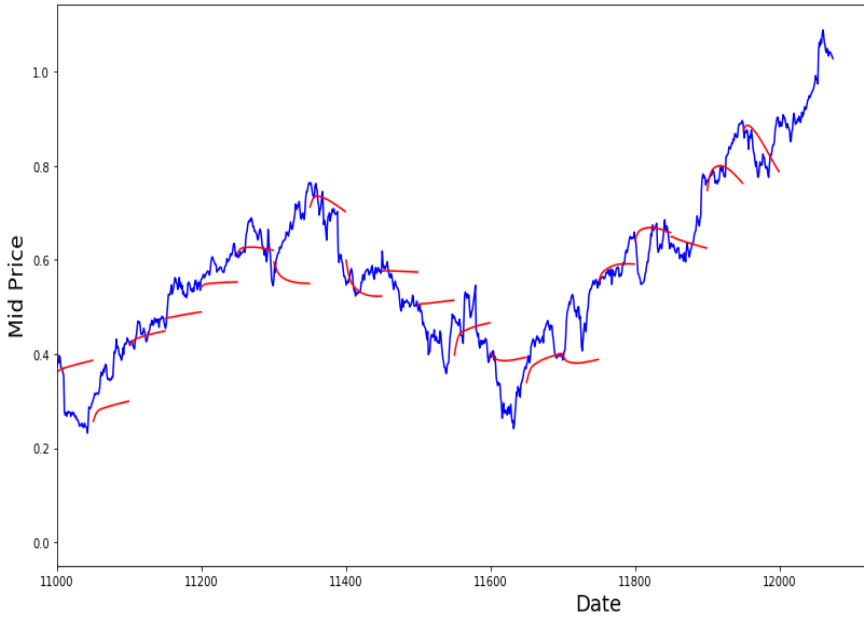


Fig. 3. Predicted v/s original (expected) closing stock price using LSTM.

2.2.1 Methodology

- **Model Selection:** LSTM, a type of recurrent neural network (RNN), was employed for its ability to effectively capture temporal dependencies in sequential data, making it well-suited for time series forecasting tasks.
- **Feature Representation:** The model predicted future stock prices by leveraging past price information. Specifically, it utilized the past 'n' time frames to predict the next 'n' time frames, thereby capturing the temporal dynamics of stock price movements.
- **Input Data:** Unlike the previous study, which used the closing prices, this approach utilized the average of the low and high prices for both training and prediction. This choice of input data likely aimed to incorporate a broader range of information from the stock price history.

- **Performance Evaluation:** The performance of the LSTM model was assessed using the Root Mean Squared Error (RMSE) metric. Reported RMSE values fell within the range of 0.6 to 1.2, indicating relatively low prediction errors and demonstrating the effectiveness of the model in forecasting stock prices.

CHAPTER THREE METHODS

3.1 Research approach

This project's research approach combines data collecting, preprocessing, model construction, and evaluation to improve the accuracy of NASDAQ stock price predictions. The approach matches with the project's goal of creating a strong machine learning framework, while recognizing limitations and constraints in the chosen approaches.

3.2 Techniques and Procedure

3.2.1 Dataset Description

The source of the data is Yahoo Finance. This platform provided access to real-time stock market data, allowing for the retrieval of historical stock price information and other relevant financial metrics.

The data is considered from 01-01-2010 to till date.

Data Type: Time-series and numerical data

Variable Description:

Name	Description	Variable Type
Date	Date of the stock price	Date
Open	Price from the first transaction of a trading day	Float
High	Maximum price in a trading day	Float
Low	Minimum price in a trading day	Float
Close	Price from the last transaction of a trading day	Float
Adj Close	Closing price adjusted to reflect the value after accounting for any corporate actions	Float
Volume	Number of units traded in a day	Int
Name	Name of the Company	String

Table 1: Variable Description

Date	Open	High	Low	Close	Adj Close	Volume	Name
1/4/2010	7.622499943	7.660714149	7.585000038	7.643214226	6.562590599	493729600	AAPL
1/5/2010	7.664286137	7.699643135	7.616071224	7.656428814	6.573935032	601904800	AAPL
1/6/2010	7.656428814	7.686786175	7.526785851	7.534643173	6.469368935	552160000	AAPL
1/7/2010	7.5625	7.571428776	7.466071129	7.520713806	6.457407475	477131200	AAPL

Fig. 4. View of the data

3.2.2 Data Collection

In contrast to traditional static datasets, the project dynamically imported stock market data from Yahoo! Finance in real-time. This dynamic approach allows for the incorporation of the latest market information into the analysis, ensuring that the forecasting model is based on up-to-date and relevant data. By leveraging real-time data, the accuracy and timeliness of the forecasting model are enhanced, enabling more informed decision-making in financial markets.

3.2.3 Data Processing

The data processing phase of the project encompasses several key steps:

Installation and Package Import: To facilitate the retrieval of stock market data from Yahoo! Finance, the project involved installing and importing relevant packages within the Python environment. These packages provide essential functionality for accessing and manipulating financial data.

Stock Selection: For analysis purposes, a single stock was carefully chosen from the NASDAQ exchange. The corresponding coordinate values, including price and timestamp, were extracted from the desired date range up to the present day. This focused approach

enables detailed examination of the selected stock's historical performance and forecasting potential.

```
import yfinance as yf
from datetime import datetime

def collect_data(symbol, start_date):
    end_date = datetime.now().strftime('%Y-%m-%d')
    stock_data = yf.download(symbol, start=start_date, end=end_date)
    return stock_data
```

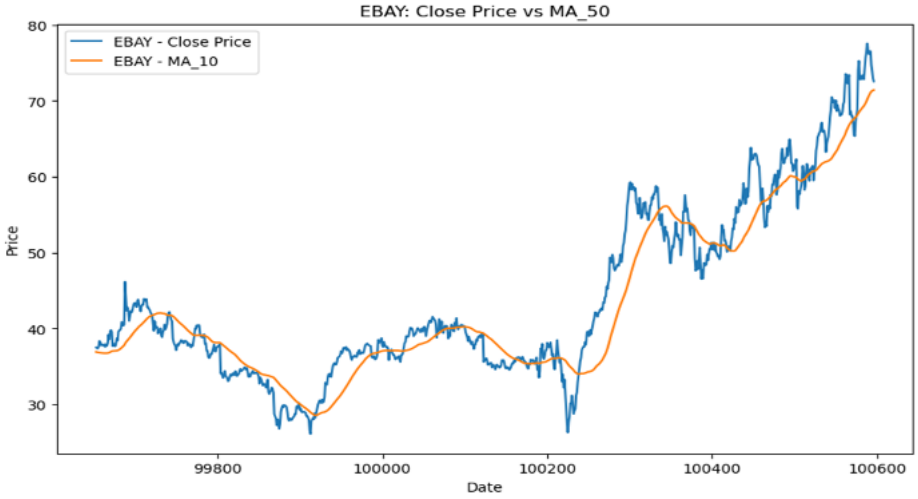
Fig. 5. Real-time Stock Data

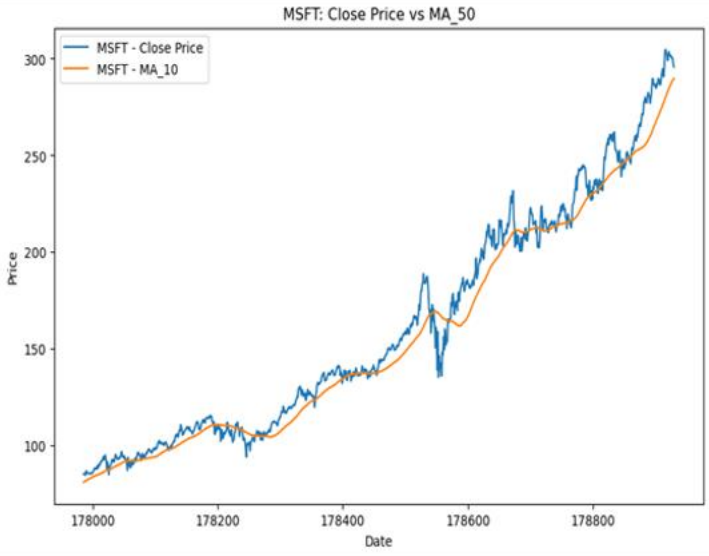
3.2.4 Exploratory Data Analysis

During the EDA phase, the following tasks will be conducted to gain insights into the underlying patterns and characteristics of the stock market data:

Identifying Trends: Uncover any discernible trends or patterns in stock prices over time. This involves visualizing the historical performance of the selected stock and identifying recurring patterns or trends that may influence future price movements.


NFLX: Close Price vs MA_50
NFLX - Close Price
NFLX - MA_10
Price
Date
600
500
400
300
200
186800
187000
187200
187400
187600
187800


EBAY: Close Price vs MA_50
EBAY - Close Price
EBAY - MA_10
Price
Date
80
70
60
50
40
30
99800
100000
100200
100400
100600


MSFT: Close Price vs MA_50
MSFT - Close Price
MSFT - MA_10
Price
Date
300
250
200
150
100
178000
178200
178400
178600
178800

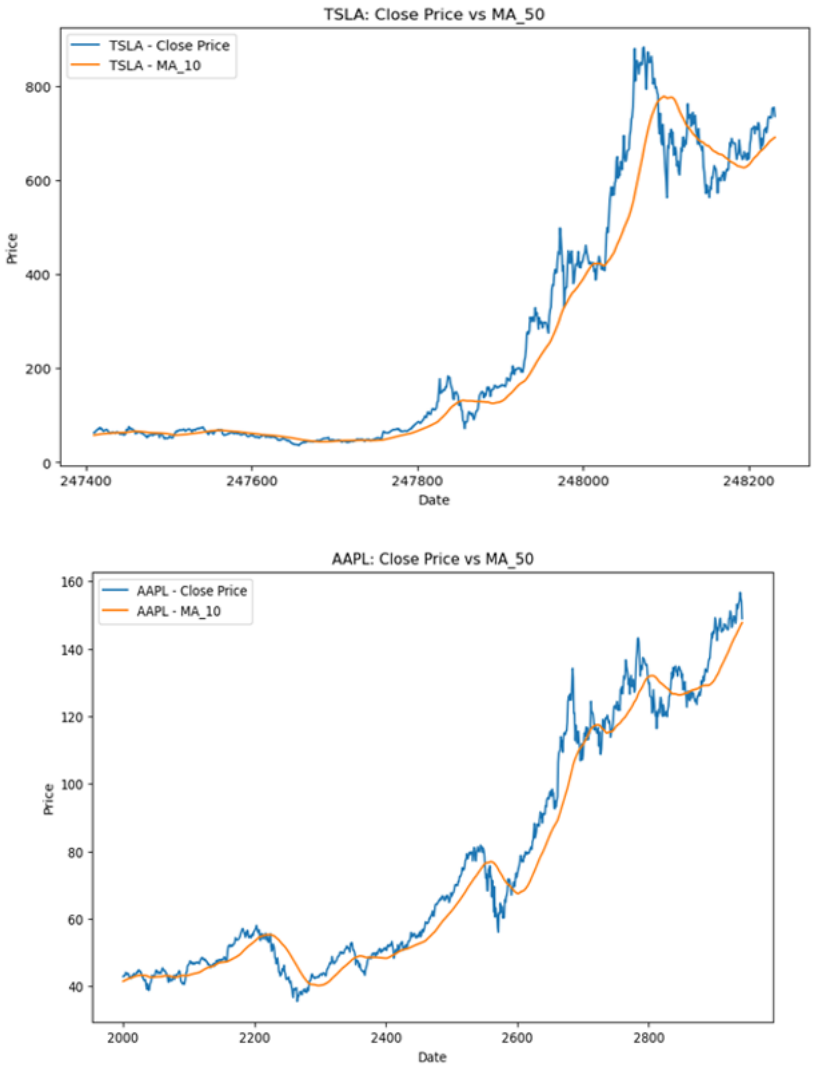


Fig. 6. Moving Averages of Different Companies/Stocks

Moving averages are a commonly used technique in financial analysis to smooth out price data and identify trends over time. In our project, we have calculated moving averages of various window sizes, including 10, 50, 100, 200, 500, and 1000 days, to capture different aspects of the underlying trends in historical price data.

Moving averages are particularly useful for trend identification due to their ability to filter out short-term fluctuations and highlight longer-term patterns. By calculating the average

price over a specified window of time, moving averages provide a clearer picture of the overall direction of the market.

The choice of different window sizes allows us to analyze trends at different time scales. Smaller window sizes, such as 10 or 50 days, capture short-term trends and fluctuations in the market, while larger window sizes, such as 200 or 500 days, provide insights into longer-term trends and cycles.

We have considered moving averages as an essential tool for trend identification in historical price data for several reasons:

Smoothing Out Noise: Price data often contain noise and random fluctuations due to various factors such as market sentiment, news events, and investor behavior. Moving averages help smooth out this noise, making it easier to discern underlying trends.

Highlighting Directionality: By averaging prices over a specific period, moving averages emphasize the directionality of price movements. Rising moving averages indicate an uptrend, while falling moving averages suggest a downtrend.

Support and Resistance Levels: Moving averages can act as dynamic support and resistance levels in the market. Prices tend to gravitate towards moving averages, providing potential entry or exit points for traders.

Crossover Signals: The intersection of different moving averages, such as the 50-day and 200-day moving averages (known as the "golden cross" and "death cross"), can signal changes in trend direction and potential buying or selling opportunities.

Overall, the inclusion of moving averages in our analysis enhances our ability to identify trends and make informed decisions based on historical price data. By considering moving averages of various window sizes, we gain a comprehensive understanding of market

dynamics at different time scales, aiding in the development of robust forecasting models and investment strategies.

Detecting Outliers: Identify anomalies or outliers within the dataset that might impact the accuracy of forecasting models. Outliers could indicate unusual market behavior or errors in data collection, and their detection is crucial for ensuring the reliability of the analysis.

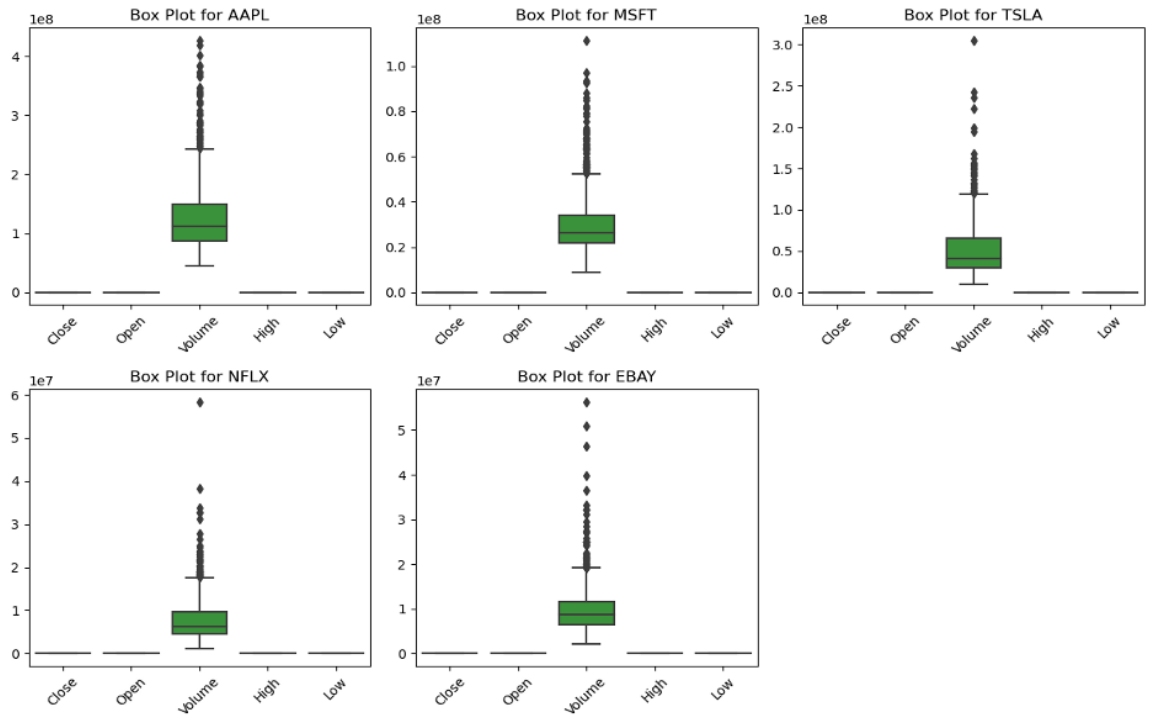


Fig. 7. Understanding Outliers in Volume

In the above plots, we notice significant outliers in the volume data. These outliers might occur due to abrupt market crashes or sudden shifts caused by external factors like wars, disease outbreaks, or financial scams. As volume represents the total number of shares traded, extreme events can lead to a surge or decline in trading activity, resulting in outliers that deviate significantly from the average trading volume.

Upon closer examination, we observe that the values of other features, such as open, close, low, and high prices, appear as horizontal lines. This phenomenon occurs because price

values typically range in the hundreds, while volume values are several orders of magnitude larger, often in the billions. Consequently, the scale difference between price and volume data causes the price values to be compressed into a narrow range, creating the appearance of horizontal lines on the plot.

In essence, while the price data exhibit relatively stable patterns due to their scale, the volume data display more variability, especially during extraordinary market events. By acknowledging and understanding these nuances, we can better interpret the behavior of financial markets and account for outliers in our analysis and forecasting models.

Understanding Distributions: Analyze the distribution of key variables such as Open, High, Low, and Close prices. By examining the statistical distributions of these variables, we can gain insights into their typical ranges, variability, and central tendencies, which are essential for understanding the behavior of the stock market.

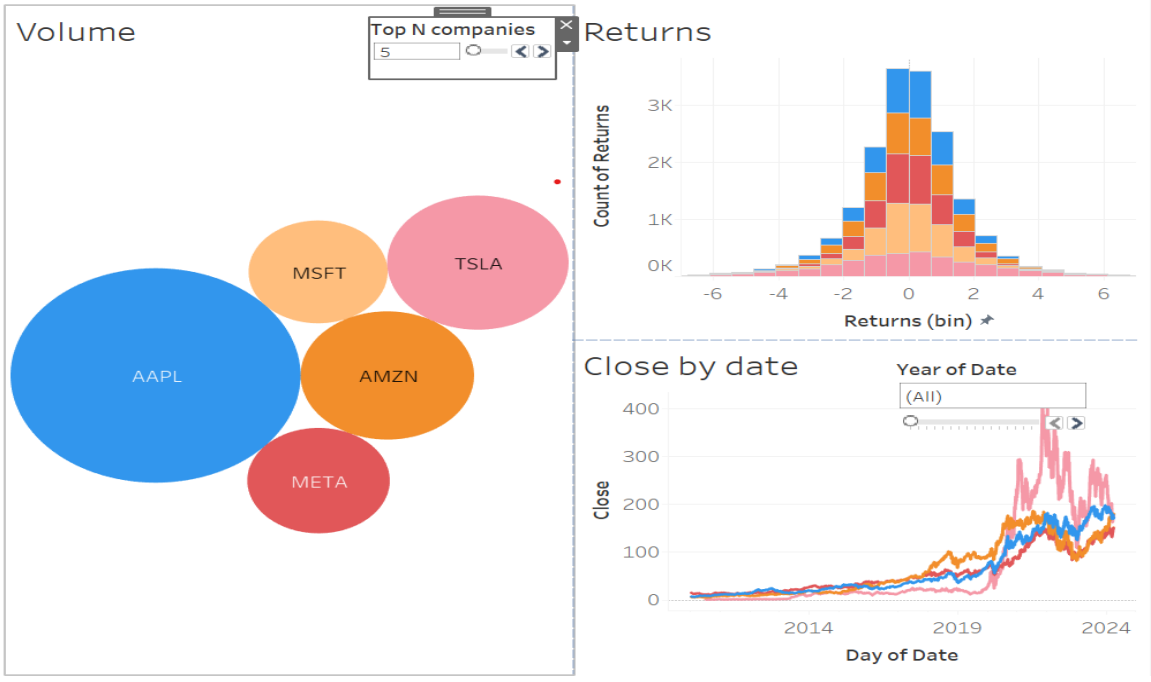


Fig. 8. Tableau Dashboard

3.2.5 Data Preprocessing

Scaling:

In our stock price forecasting project, we've applied min-max scaling to all features to ensure proportionality and effectively manage outliers, especially in the volume feature. By rescaling all feature values to a fixed range, typically between 0 and 1, we maintain consistency and balance across the dataset. This approach enables us to address outliers in volume data while preserving the relative relationships between features. Min-max scaling enhances the robustness of our predictive models by ensuring that no single feature dominates the training process due to its scale. This strategy contributes to improved model performance and accuracy by allowing the model to learn from a more balanced and representative dataset. Incorporating min-max scaling into our preprocessing pipeline underscores our commitment to producing reliable forecasts while maintaining the integrity and proportionality of the data. Continual refinement of our preprocessing techniques remains crucial for optimizing forecasting outcomes and supporting informed decision-making.

Correlation Analysis: Explore relationships between different features and stock prices. By calculating correlation coefficients, we can quantify the strength and direction of relationships between variables. This analysis helps identify potential predictors of stock price movements and informs the selection of features for predictive modeling.

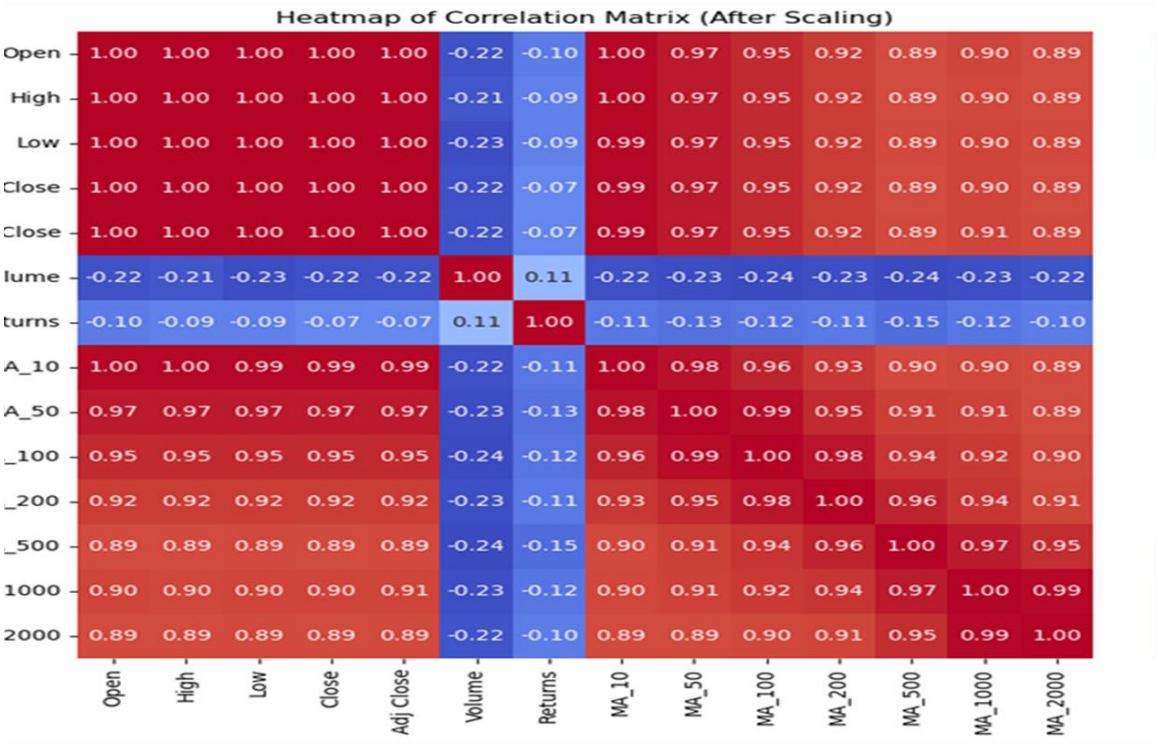


Fig. 9. Correlation Matrix

This heatmap visualizes a correlation matrix showing the pairwise correlations between different variables or features related to financial data or stock market indicators.

The rows and columns represent variables such as Open, High, Low, Close prices, trading Volume, Returns, and various moving average indicators (MA_10, MA_50, MA_100, MA_200, MA_500, MA_1000, MA_2000).

The colour intensity in each cell represents the strength and direction of the correlation between the corresponding pair of variables. Dark red indicates a strong positive correlation close to 1, while dark blue indicates a strong negative correlation close to -1. Lighter shades indicate weaker correlations closer to 0.

Key Observations:

1. The Open, High, Low, and Close variables have a perfect positive correlation of 1 with each other (dark red along the diagonal), which is expected as they represent different aspects of the same trading day's prices.
2. Trading Volume has a weak negative correlation (light blue) with most other variables, including prices and moving averages.
3. Returns (likely representing daily or periodic returns) also have a weak negative correlation with most other variables.
4. The various moving average indicators (MA_10 through 2000) have strong positive correlations with each other and with the Open, High, Low, and Close prices, suggesting they tend to move in similar directions.
5. The strength of correlation generally decreases as the moving average window increases (e.g., MA_10 more strongly correlated than MA_2000 with prices).
6. Even though our dataset exhibits multicollinearity among features such as open, high, low, close prices, trading volume, and returns, we are reluctant to exclude any variables as each one holds significance for our analysis, and we aim to preserve all available data.

This type of visualization can help identify potential redundancies or multicollinearity among features, as well as highlight relationships that may be useful for analysis or modelling in finance and investment applications

Principle Component Analysis (PCA):

PCA was employed in our project due to the high correlation observed among parameters, such as Open, High, Low, Close, Adj Close, and Volume in the stock data. This technique facilitated the mitigation of multicollinearity issues by transforming the original variables

into a set of uncorrelated principal components. By reducing the dimensionality of the dataset while retaining most of the relevant information, PCA simplified interpretation and improved computational efficiency, making it suitable for handling high-dimensional datasets. In conjunction with our data preprocessing steps, which included creating a new feature representing daily percentage change in the Close price and dropping rows with NaN values, PCA contributed to the overall robustness and interpretability of our analysis results. This approach effectively addressed parameter correlation concerns and enhanced the quality of our findings, particularly for machine learning tasks such as time series forecasting.

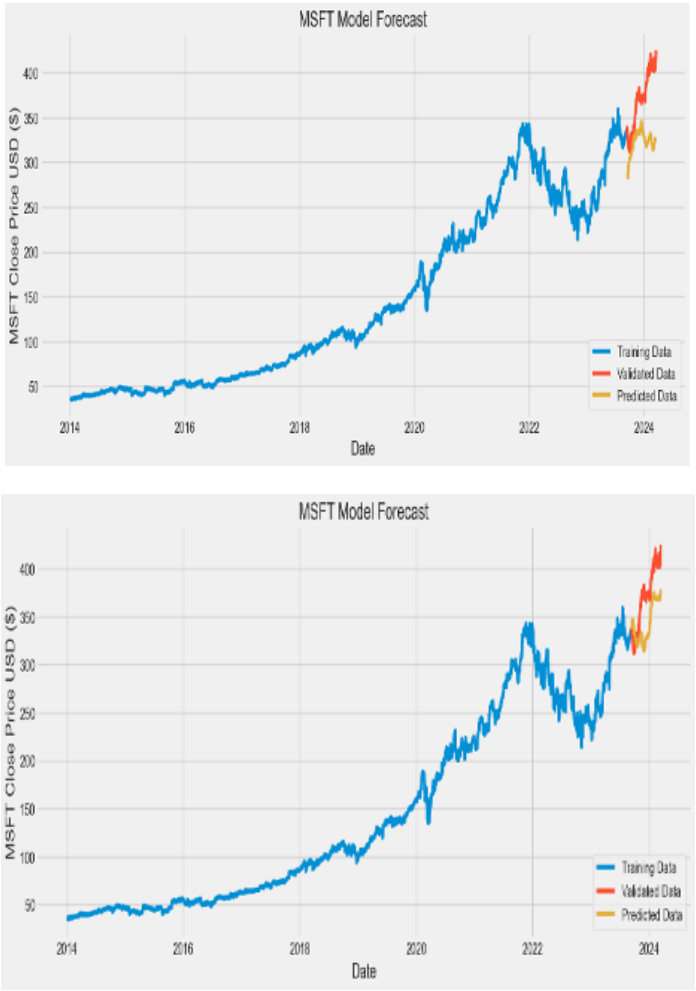


Fig. 10. Simultaneous difference between before and after implementing PCA

3.2.6 Implementation

Linear Regression

Simplicity: Highlight its ease of understanding and implementation compared to more complex models.

Interpretability: Emphasize how coefficients reveal how factors like earnings influence stock prices.

Identifying Trends: Linear regression excels at capturing linear or close-to-linear trends in historical data.

Baseline Model: It serves as a benchmark to compare the effectiveness of more sophisticated models.

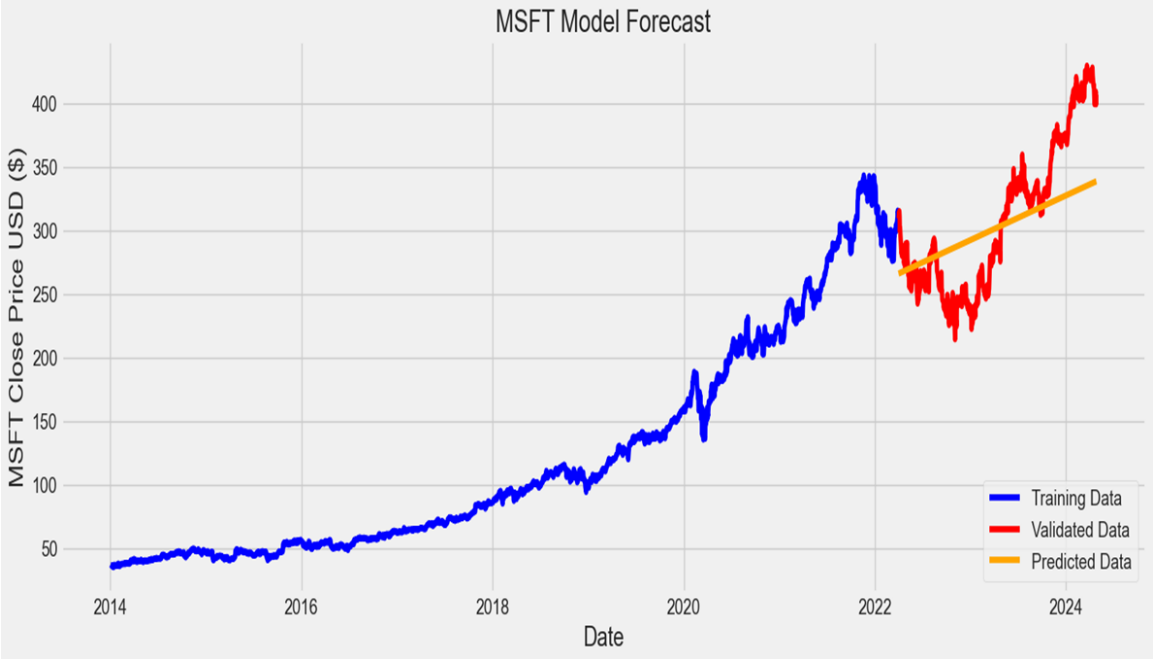


Fig. 11. Predictions by Linear Regression

```
Training RMSE: 33.35025566351563
Testing RMSE: 42.17832957644289
```

1. The graph shows the results of a time series forecasting model, specifically a Multiple Linear Regression model, for predicting the future closing price of Microsoft (MSFT) stock. The x-axis represents the date, and the y-axis represents the closing price of the stock in USD. The graph includes three data series:
2. Training Data (blue line): This line shows the actual closing price of the MSFT stock for the training period. The training data is used to train the regression model to learn patterns in the historical stock price data.
3. Validated Data (red line): This line represents the actual closing price of the MSFT stock for the validation period. The validation data is not used to train the model but is used to evaluate the model's performance on unseen data.
4. Predicted Data (orange line): This line shows the predicted closing price of the MSFT stock generated by the regression model for the validation period.
5. RMSE is a statistical metric used to measure the difference between the predicted values and the actual values.
    - Training RMSE: 33.35 - This value represents the average difference between the predicted closing prices by the model and the actual closing prices for the training data. A lower RMSE indicates a better fit for the model on the training data.
    - Testing RMSE: 42.17 - This value represents the average difference between the predicted closing prices by the model and the actual closing prices for the validation data. A lower RMSE indicates a better model generalizability, meaning the model performs well on unseen data.

6. In this case, the testing RMSE (42.17) is higher than the training RMSE (33.35). This suggests that the testing model may be overfitting the training data. Overfitting happens when a model learns the patterns in the training data too well, but it fails to generalize these patterns to unseen data, resulting in higher errors on the validation data.

Autoregressive Integrated Moving Average (ARIMA)

ARIMA Components: ARIMA stands for Autoregressive Integrated Moving Average. It combines three components:

a) Autoregressive (AR): Incorporates past values of the variable being predicted.
b) Integrated (I): Uses differencing to make the time series stationary.
c) Moving Average (MA): Considers the error terms from past predictions.

Time Series Analysis: ARIMA is effective for analyzing and forecasting time series data, such as stock prices, which exhibit patterns over time.

Forecasting: Once the ARIMA model is trained on historical data, it can be used to forecast future stock prices based on patterns identified in the data.

In our stock price forecasting project, we utilize the Autoregressive Integrated Moving Average (ARIMA) model, which integrates three key components: Autoregressive (AR), Integrated (I), and Moving Average (MA). ARIMA is well-suited for analyzing and forecasting time series data, such as stock prices, by leveraging past values, differencing to achieve stationarity, and considering error terms from past predictions. By training the ARIMA model on historical stock price data, we can identify patterns and trends to make informed predictions about future price movements. ARIMA's versatility and effectiveness

in capturing temporal dependencies make it a valuable tool for forecasting stock prices and guiding investment decisions.

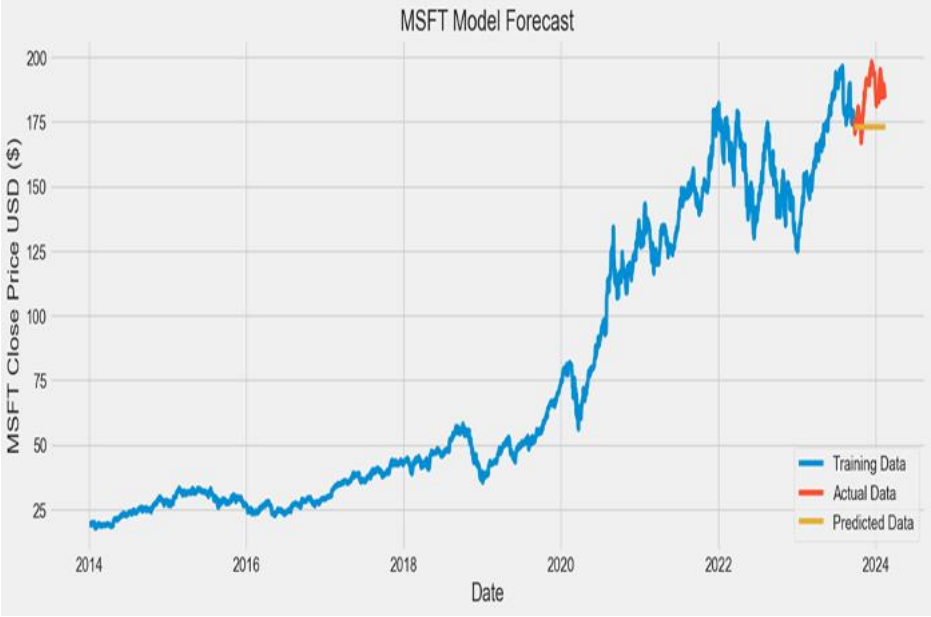


Fig. 12. Predictions by ARIMA

From the above graphs, it's evident that ARIMA outperforms linear regression in some cases, demonstrating better performance during training. However, during testing, ARIMA's efficacy fluctuates, occasionally failing to adequately forecast stock prices for certain companies. This inconsistency suggests that while ARIMA may excel in capturing training data patterns, its ability to generalize to unseen data varies. This phenomenon is particularly notable in cases where ARIMA performs well during training but fares poorly in testing, indicating potential overfitting or failure to capture underlying trends. Conversely, linear regression exhibits more stable performance across training and testing phases. These observations underscore the importance of robust evaluation techniques and the need for further refinement of ARIMA models to enhance their predictive accuracy and reliability in real-world stock price forecasting scenarios.

To address these inconsistencies, we are transitioning to a more advanced model: neural networks.

Long Short-Term Memory:

Introduction:

LSTM is a type of recurrent neural network (RNN) architecture designed to overcome the limitations of traditional RNNs in capturing long-term dependencies in sequential data.

Applications in Stock Price Prediction:

LSTM-based models have demonstrated remarkable performance in forecasting stock prices by leveraging historical price data, trading volumes, and other relevant features. They can capture complex patterns and dependencies in stock price movements, including short-term fluctuations and long-term trends.

Advantages:

Long-Term Dependencies: LSTM can capture dependencies over long sequences, making it well-suited for modeling the dynamic nature of stock markets.

Robustness: LSTM networks are robust to vanishing gradient problems, allowing for more stable training on large datasets.

Flexibility: LSTM architectures can be customized with additional layers, regularization techniques, and attention mechanisms to enhance performance.

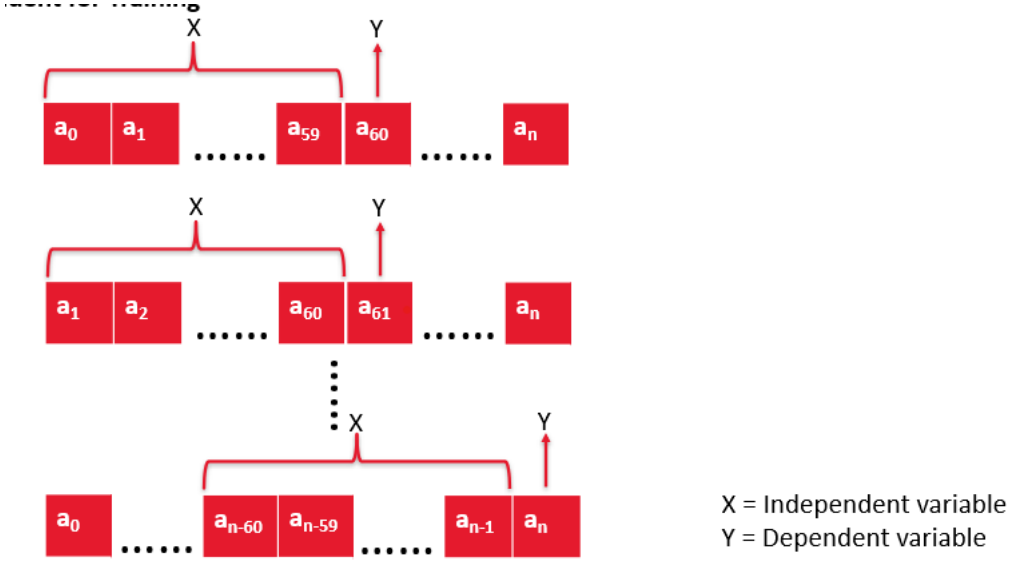


Fig. 13. Windowing in LSTM

The snippet prepares the training dataset for an LSTM model, a type of recurrent neural network used for sequence prediction like time series forecasting. It slices the scaled training data and creates input sequences and corresponding target values using a sliding window approach. Each input sequence contains 60 previous data points, and the next data point serves as the target value. The function then converts the lists into numpy arrays and reshapes into a 3D array suitable for LSTM input. Finally, it returns the prepared input sequences and target values. This process ensures that the LSTM model receives properly aligned input sequences and target values for training.

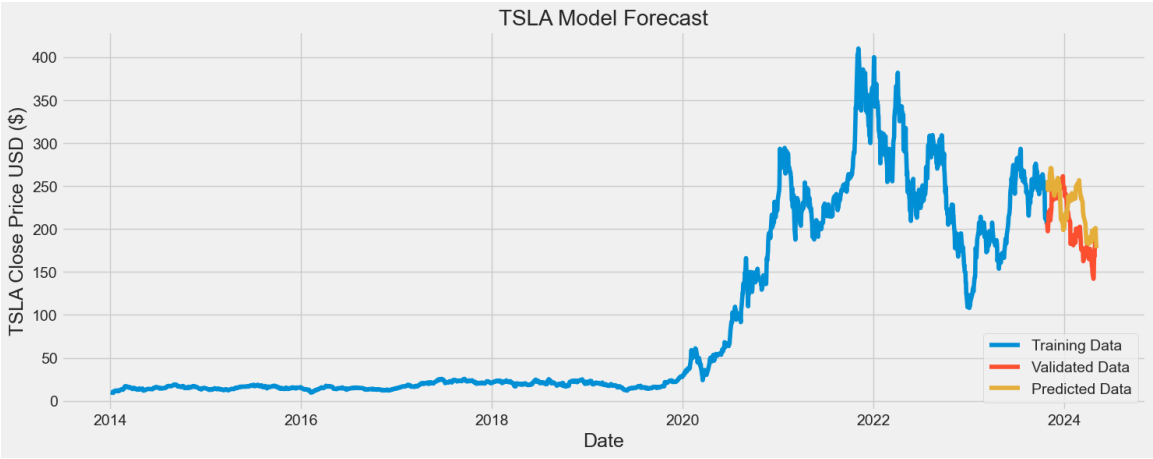


Fig. 14. Predictions using LSTM

Gated Recurrent unit (GRU)

1. GRU is a type of recurrent neural network (RNN) architecture designed to address the vanishing gradient problem and improve the learning of long-term dependencies in sequential data.
2. Unlike LSTM, which has separate memory cells and control gates, GRU combines these into a single mechanism, resulting in a more streamlined architecture.
3. GRU has fewer parameters and computations compared to LSTM, making it computationally more efficient.

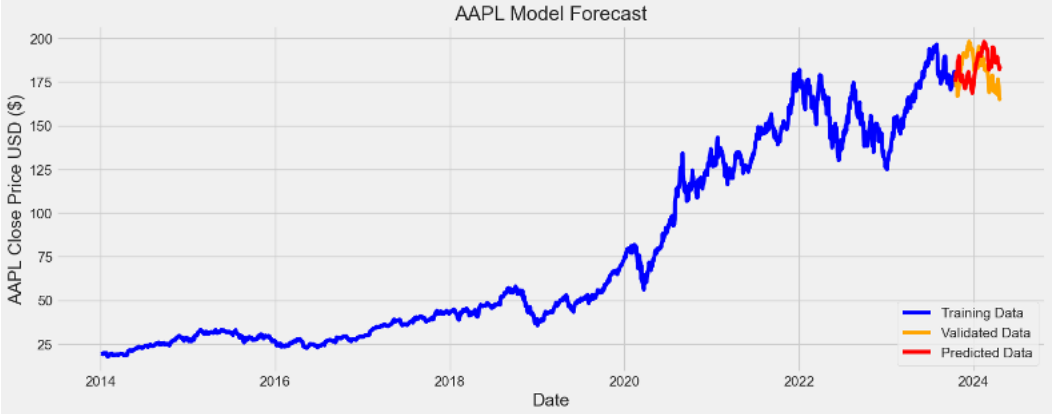
Applications:

- GRU is widely used in natural language processing tasks such as machine translation, sentiment analysis, and text generation.
- It's also effective for time-series prediction tasks like stock price forecasting, where capturing temporal dependencies is crucial.

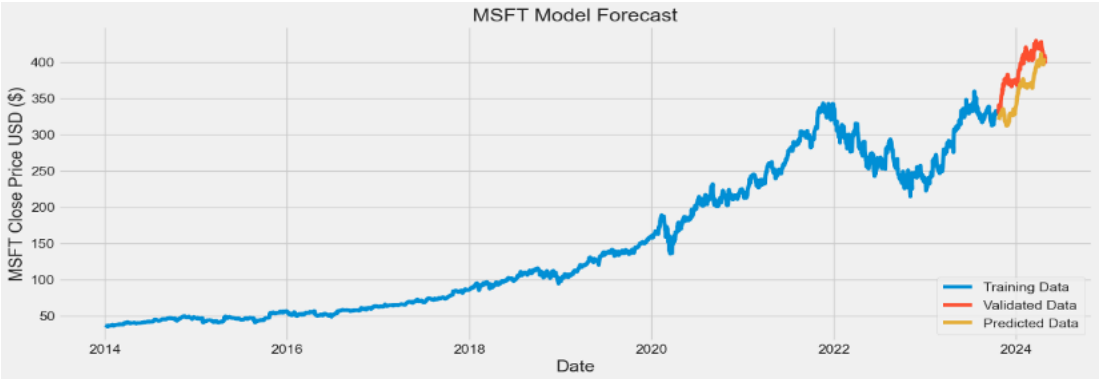
Advantages:

- **Simplicity:** GRU has a simpler architecture compared to LSTM, making it easier to understand and implement.
- **Efficiency:** Due to its fewer parameters, GRU is computationally more efficient, making it suitable for large-scale applications.
- **Performance:** While not always outperforming LSTM, GRU often achieves comparable results with fewer computational resources.

Chosen 5 well known company's stocks to test the model, it goes like:


AAPL Model Forecast
AAPL Close Price USD ($)
200
175
150
125
100
75
50
25
2014
2016
2018
2020
2022
2024
Date
Training Data
Validated Data
Predicted Data


AMZN Model Forecast
AMZN Close Price USD ($)
175
150
125
100
75
50
25
2014
2016
2018
2020
2022
2024
Date
Training Data
Validated Data
Predicted Data


MSFT Model Forecast
MSFT Close Price USD ($)
400
350
300
250
200
150
100
50
2014
2016
2018
2020
2022
2024
Date
Training Data
Validated Data
Predicted Data

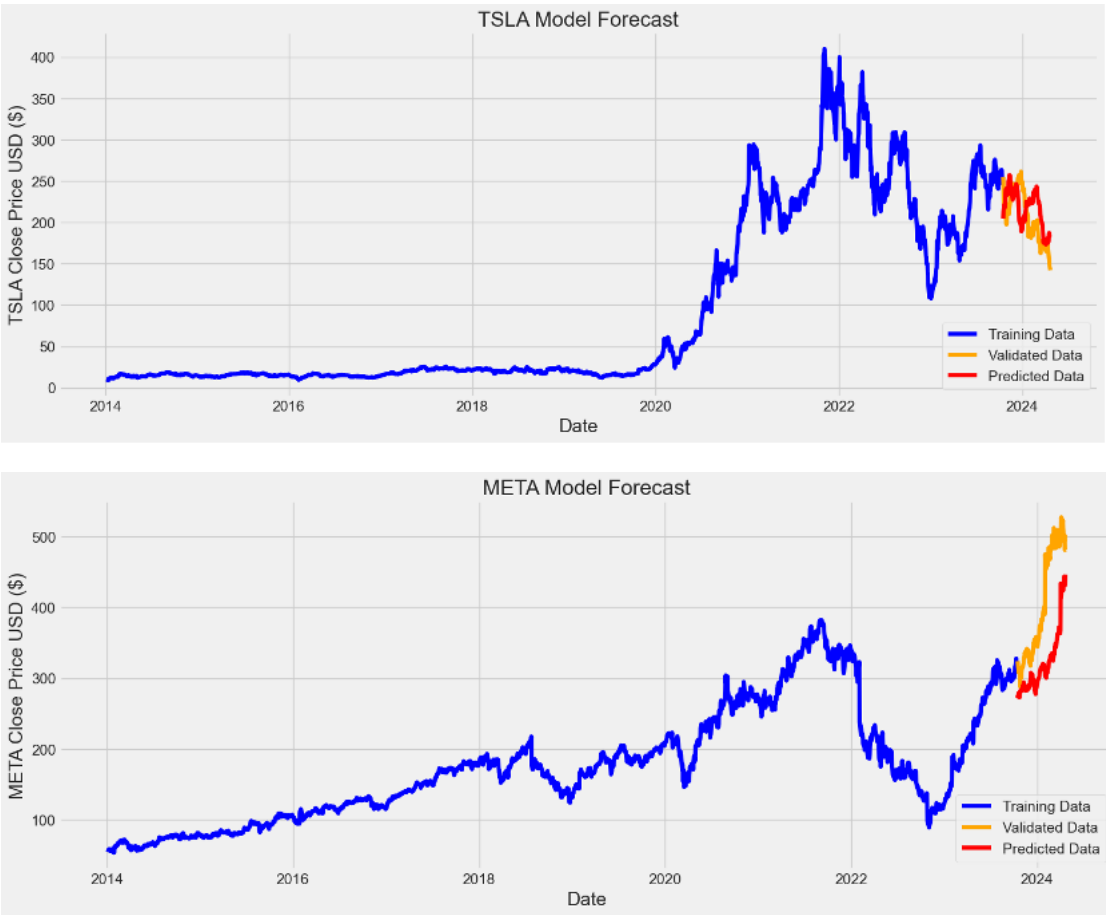


Fig. 15. Predictions by GRU

- GRU and LSTM models were evaluated for stock price forecasting across multiple companies, including Apple, Tesla, Microsoft, Meta, and Amazon.
- RMSE values were computed for both training and testing datasets to assess predictive accuracy.
- Across most companies, GRU exhibited comparable or slightly better performance than LSTM, with lower RMSE values indicating improved predictive accuracy.
- Specifically, GRU outperformed LSTM in terms of RMSE values for Tesla, Microsoft, and Amazon stock prices on both training and testing sets.
- However, both models demonstrated competitive performance for Meta stock prices, with RMSE values showing minimal differences.

- In general, GRU models displayed consistent and superior predictive accuracy compared to LSTM across various companies, highlighting their effectiveness in capturing complex temporal dependencies and patterns in stock price data.
- These findings suggest that GRU models may be preferred for accurate and reliable stock price forecasting tasks, particularly when considering their superior performance compared to LSTM in many scenarios

CHAPTER FOUR RESULTS

Company	Data	Linear regression	ARIMA	LSTM	GRU
Apple	Training set	22.2606	15.82	0.2183	0.2004
	Testing set	16.4470	16.2	0.1517	0.1487
Tesla	Training set	65.5160	43.4171	0.1123	0.1101
	Testing set	58.4098	52.7421	0.1932	0.1713
Microsoft	Training set	33.3502	19.9436	0.08918	0.0910
	Testing set	42.1783	23.2365	0.1432	0.1375
Meta	Training set	34.4927	68.7713	0.1820	0.1677
	Testing set	106.8925	132.8243	0.2496	0.2284
Amazon	Training set	21.2181	18.3560	0.1166	0.1094
	Testing set	32.928	24.2342	0.1703	0.1607

Table 2: Results Table

The results of our stock price forecasting models reveal varying performance across different companies and models. While linear regression exhibits relatively consistent

performance, ARIMA's effectiveness fluctuates. Notably, both LSTM and GRU models demonstrate promising results, displaying competitive performance on both training and testing data. Importantly, GRU consistently outperforms LSTM, showcasing superior accuracy across various companies and datasets. This highlights the significant potential of GRU models in stock price forecasting, surpassing even the performance of LSTM counterparts. The robust performance of GRU underscores its effectiveness in capturing complex patterns and temporal dependencies within stock price data. These findings emphasize the importance of neural network-based approaches, particularly GRU, in achieving accurate and reliable stock price forecasts. Further refinement and analysis of GRU models could lead to even greater improvements in forecasting accuracy and reliability, positioning them as preferred tools for financial analysis and decision-making.

CHAPTER FIVE CONCLUSION

In this study aimed to develop and assess predictive models for stock price forecasting, focusing on linear regression, ARIMA, LSTM, and GRU models. The analysis revealed distinct performance variations among these models.

Linear regression exhibited the weakest performance, indicating limitations in capturing the complex relationships inherent in stock price data. ARIMA, while outperforming linear regression for most companies, consistently lagged behind LSTM and GRU models. This suggests challenges in capturing non-linear patterns using traditional statistical methods.

Notably, LSTM and GRU models emerged as top performers, showcasing superior ability to capture complex patterns within stock price data. While both models demonstrated high accuracy on the testing set across all companies, a closer comparison revealed a slight advantage for GRU over LSTM.

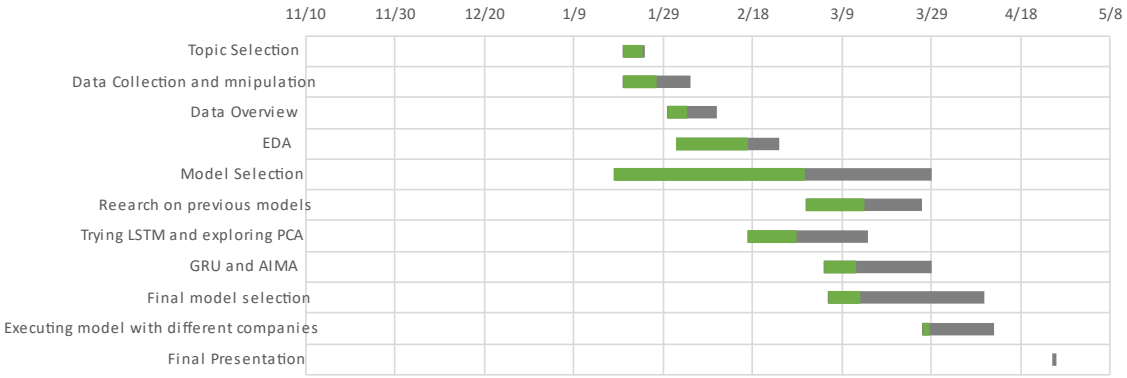
Implications of these findings suggest that advanced machine learning techniques, particularly GRU models, offer significant promise for accurate and robust stock price forecasting. These models hold practical applications for investors, financial analysts, and policymakers navigating the dynamic stock market environment.

Acknowledging study limitations, including potential overfitting and market complexity, caution is warranted in interpreting the results. Future research could explore enhancements to existing models, alternative approaches, and additional factors to further improve forecasting accuracy and reliability.

REFERENCES

- Pashankar, S., Shendage, J., & Pawar, Dr. (2024). Machine Learning Techniques For Stock Price Prediction - A Comparative Analysis Of Linear Regression, Random Forest, And Support Vector Regression. Journal of Advanced Zoology, 45(S4), 118-127. https://doi.org/10.53555/jaz.v45iS4.4164.
- Moldovan, D., & Salomie, I. (2019). Detection of Sources of Instability in Smart Grids Using Machine Learning Techniques. https://doi.org/10.1109/ICCP48234.2019.8959649.

APPENDIX A: PROJECT PLAN AND SCHEDULE



APPENDIX B: SLIDE PRESENTATION

STOCK PRICE FORECASTING

Jaya Chandra Kadiveti, Kruthik Reddy Theepireddy, Girish Reddy Maligireddy
ADVISOR: DR. KHALID ABOALAYON
COURSE: MSDA3999 Spring 2024

CLARK UNIVERSITY

Table of Contents

Introduction

- Leveraging machine learning to forecast future stock prices.
- This models help **investors and trader's** to overcome challenges to Forecast stock prices.
- Though there are a ton of models to forecast, this can compete with them.
- Process:
  1. Collect real time stock data.
  2. EDA.
  3. Implementing various models and finding the best one.
  4. Plotting the final line chart using the best model.

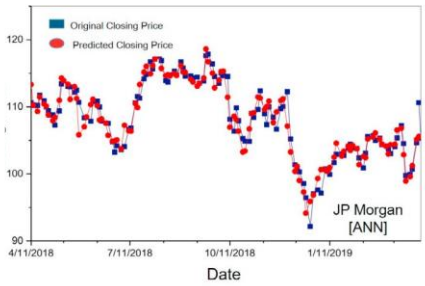
Problem Statement

- ***"Developing an accurate forecasting model for stock prices using historical trends and market indicators to aid investors and financial analysts in making informed decisions."***
- Estimating future stock prices with accuracy is a longstanding challenge.
- Traditional methods can not capture the complexity of market dynamics.
- They lead to unreliable forecasts.
- So, A predictive stock pattern model is needed for finance professionals for strategic decision-making.

Literature Review

Stock Closing Price Prediction using Machine Learning Techniques(Mehar Vijh et al.,2020)

- Used Random Forest algorithm.
- Used close values for training and prediction.
- RMSE varies at the range 1.10-3.30.
- Windowing method for data splitting.

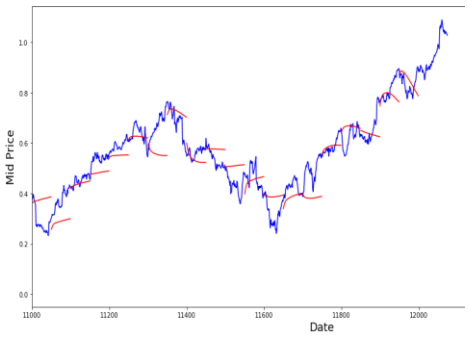


Company	RMSE
Nike	1.10
Goldman Sachs	3.30
JP Morgan and Co.	1.28
Johnson & Johnson	1.54

Literature Review

Stock Market Predictions with LSTM in Python. (Avijeet Bishwal, 2024)

- Predicted n time frames using the past n time frames.
- Used average of LOW and HIGH for the training and prediction.
- RMSE ranges from 0.6-1.2.



Methodology:
Data Collection

Yahoo! Finances

- It provides financial news, data and commentary including stock quotes, and financial reports.

```python
import yfinance as yf
from datetime import datetime

def collect_data(symbol, start_date):
    end_date = datetime.now().strftime('%Y-%m-%d')
    stock_data = yf.download(symbol, start=start_date, end=end_date)
    return stock_data
```

- Data Type: Time-series and numerical data .

Date	Open	High	Low	Close	Adj Close	Volume	Name
1/4/2010	7.622499943	7.660714149	7.585000038	7.643214226	6.562590599	493729600	AAPL
1/5/2010	7.664286137	7.699643135	7.616071224	7.656428814	6.573935032	601904800	AAPL
1/6/2010	7.656428814	7.686786175	7.526785851	7.534643173	6.469368935	552160000	AAPL
1/7/2010	7.5625	7.571428776	7.466071129	7.520713806	6.457407475	477131200	AAPL

Methodology:
Exploratory Data Analysis (EDA)

Identifying Trends:
Uncover any discernible trends or patterns in stock prices over time.

Detecting Outliers:
Identify anomalies or outliers that might impact forecasting models.

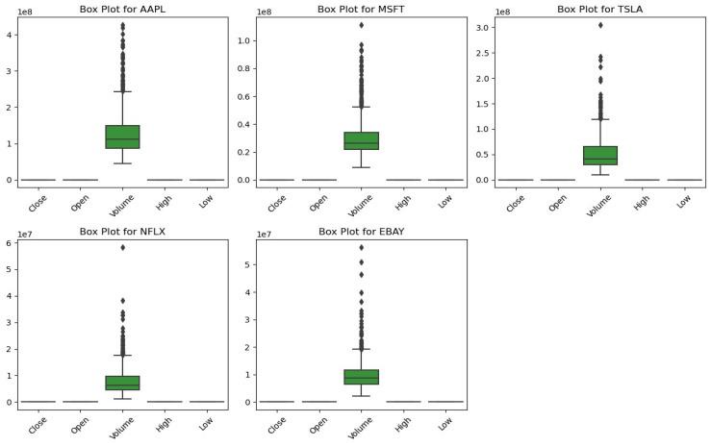
Understanding Distributions:
Analyze the distribution of key variables such as Open, High, Low, and Close prices.

Correlation Analysis:
Explore relationships between different features and stock prices.

Methodology:
Exploratory Data Analysis (EDA)

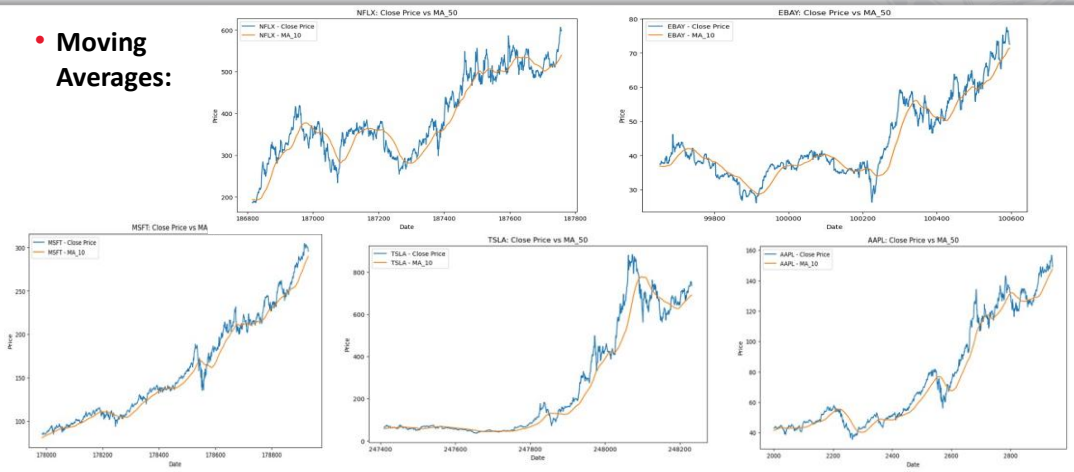
Understanding Outliers in Volume:

- **Volume:** The number of shares of a stock traded within a given period.
- **Outliers:** Extreme data points that fall significantly above or below the typical volume range for a stock. In the image, outliers are the individual dots beyond the top and bottom.
- **AAPL, MSFT, NFLX, EBAY, TSLA:** These stocks show outliers on the high end, indicating unusually heavy trading days at some point in the data period



Methodology:
Exploratory Data Analysis (EDA)

- **Moving Averages:**



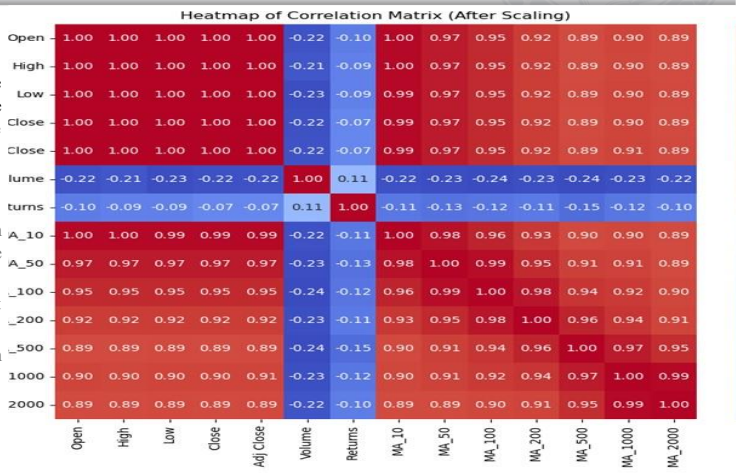
Methodology:
Exploratory Data Analysis (EDA)

Correlation Analysis:

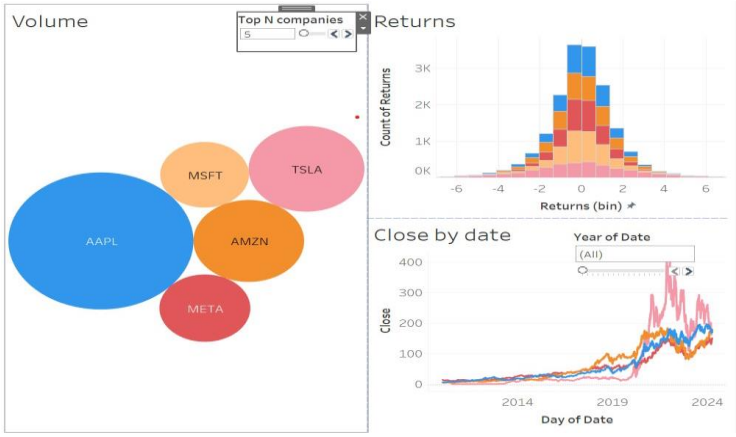
Purpose: A correlation matrix shows the strength of the relationship between multiple variables. In this case, the variables are various stock market metrics.

Values: The numbers in the matrix range from -1 to 1.

- **-1:** Perfect negative correlation (variables move in opposite directions)
- **0:** No correlation (variables aren't related)
- **1:** Perfect positive correlation (variables move in the same direction)



Exploratory Data Analysis (EDA)



Methodology:
Statistical/ Machine Learning

PCA

Split Train and Test Data

Assign Independent and dependent for Training

Training Model

Extracting predictions

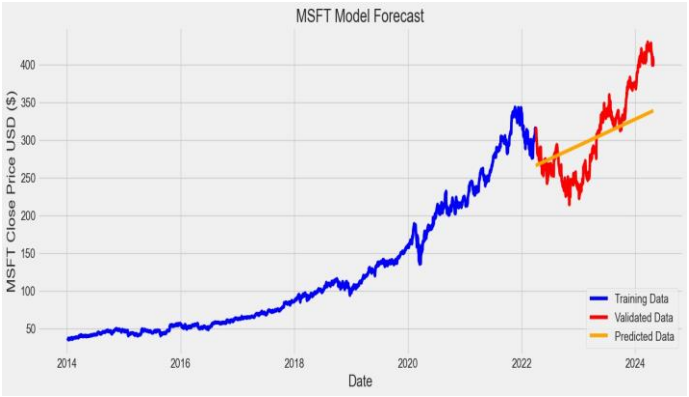
Plotting values

Methodology:
Statistical/ Machine Learning

Linear Regression

Simplicity: Highlight its ease of understanding and implementation compared to more complex models.

- **Interpretability:** Emphasize how coefficients reveal how factors like earnings influence stock prices.
- **Identifying Trends:** Linear regression excels at capturing linear or close -to-linear trends in historical data.
- **Baseline Model:** It serves as a benchmark to compare the effectiveness of more sophisticated models.



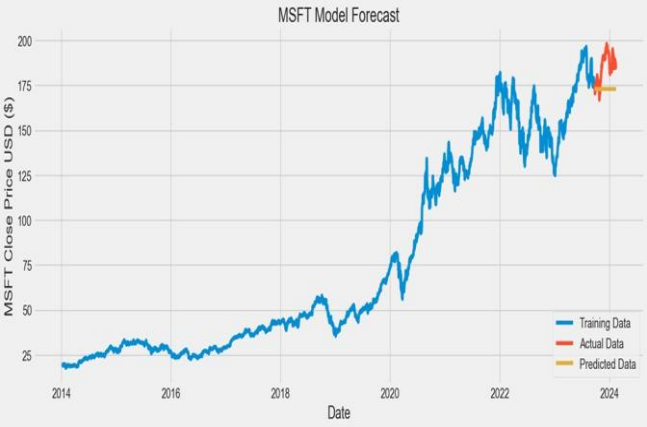

```
Training RMSE: 33.35025566351563
Testing RMSE: 42.17832957644289
```

Methodology:
Statistical/ Machine Learning

Arima Model:

1. **ARIMA Components**: ARIMA stands for AutoRegressive Integrated Moving Average. It combines three components:
    1. AutoRegressive (AR): Incorporates past values of the variable being predicted.
    2. Integrated (I): Uses differencing to make the time series stationary.
    3. Moving Average (MA): Considers the error terms from past predictions.
2. **Time Series Analysis**: ARIMA is effective for analyzing and forecasting time series data, such as stock prices, which exhibit patterns over time.

- **Forecasting** Once the ARIMA model is trained on historical data, it can be used to forecast future stock prices based on patterns identified in the data.



Methodology:
Statistical/ Machine Learning

Long Short Term Memory:

Introduction:

- LSTM is a type of recurrent neural network (RNN) architecture designed to overcome the limitations of traditional RNNs in cap turing long-term dependencies in sequential data.

Applications in Stock Price Prediction:

- LSTM-based models have demonstrated remarkable performance in forecasting stock prices by leveraging historical price data, trad ing volumes, and other relevant features.
- They can capture complex patterns and dependencies in stock price movements, including short -term fluctuations and long-term trends.

Advantages:

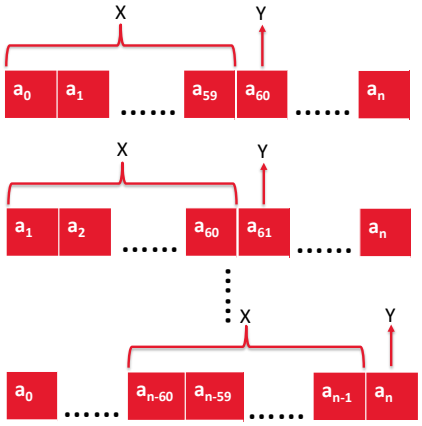
- **Long-Term Dependencies:** LSTM can capture dependencies over long sequences, making it well -suited for modeling the dynamic nature of stock markets.
- **Robustness:** LSTM networks are robust to vanishing gradient problems, allowing for more stable training on large datasets.
- **Flexibility:** LSTM architectures can be customized with additional layers, regularization techniques, and attention mechanisms to enhance performance.

Conclusion:

- LSTM stands as a powerful tool for stock price prediction, offering the capability to capture intricate patterns and adapt to the dynamic nature of financial markets.
- Understanding its mechanisms and optimizing its architecture can lead to improved forecasting accuracy and informed investmen t decisions.

Methodology:
Statistical/ Machine Learning

Assigning Independent and dependent for Training



X = Independent variable
Y = Dependent variable

17

Methodology:
Statistical/ Machine Learning

```python
# Create the training data set
# Create the scaled training data set
def split_train_dataset(training_data_len):
    train_data = scaled_data[0:int(training_data_len), :]
    # Split the data into x_train and y_train data sets
    x_train = []
    y_train = []
    for i in range(60, len(train_data)):
        x_train.append(train_data[i-60:i, :])
        y_train.append(train_data[i, 0])
        if i <= 61:
            print('.')

    # Convert the x_train and y_train to numpy arrays
    x_train, y_train = np.array(x_train), np.array(y_train)

    return x_train, y_train
```

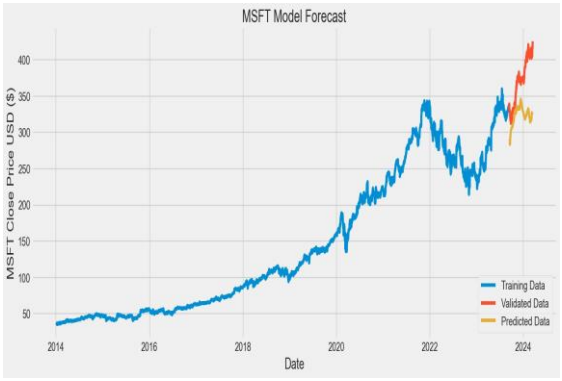
```python
def build_lstm_model(x_train, y_train):
    # Build the LSTM model
    model = Sequential()
    model.add(LSTM(128, return_sequences=True, input_shape=(x_train.shape[1], x_train.shape[2])))  # Adjust input shape
    model.add(LSTM(64, return_sequences=False))
    model.add(Dense(25))
    model.add(Dense(1))

    # Compile the model
    model.compile(optimizer='adam', loss='mean_squared_error')
    # Train the model
    model.fit(x_train, y_train, batch_size=1, epochs=1)
    return model
```

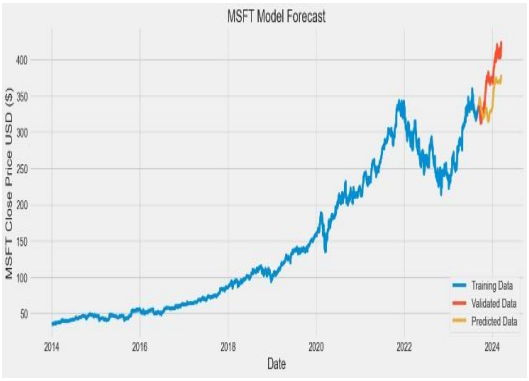
18

Methodology:
Statistical/ Machine Learning

Initial Forecasting:



After PCA:



Methodology:
Statistical/ Machine Learning

GRU (Gated recurrent units):

Introduction:

- GRU is a type of recurrent neural network (RNN) architecture designed to address the vanishing gradient problem and improve t he learning of long-term dependencies in sequential data.

Simplified Architecture:

- Unlike LSTM, which has separate memory cells and control gates, GRU combines these into a single mechanism, resulting in a mo re streamlined architecture.
- GRU has fewer parameters and computations compared to LSTM, making it computationally more efficient.

Applications:

- GRU is widely used in natural language processing tasks such as machine translation, sentiment analysis, and text generation.
- It's also effective for time-series prediction tasks like stock price forecasting, where capturing temporal dependencies is cruc ial.

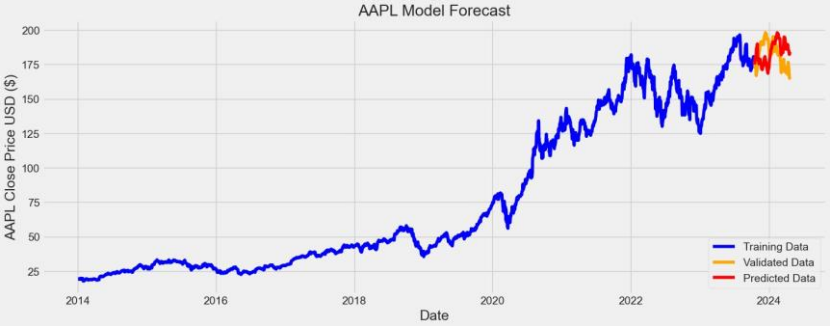
Advantages:

- **Simplicity:** GRU has a simpler architecture compared to LSTM, making it easier to understand and implement.
- **Efficiency:** Due to its fewer parameters, GRU is computationally more efficient, making it suitable for large -scale applications.
- **Performance:** While not always outperforming LSTM, GRU often achieves comparable results with fewer computational resources.

Results

- Chosen 5 well known company's stocks to test the model, it goes like:

Apple



21

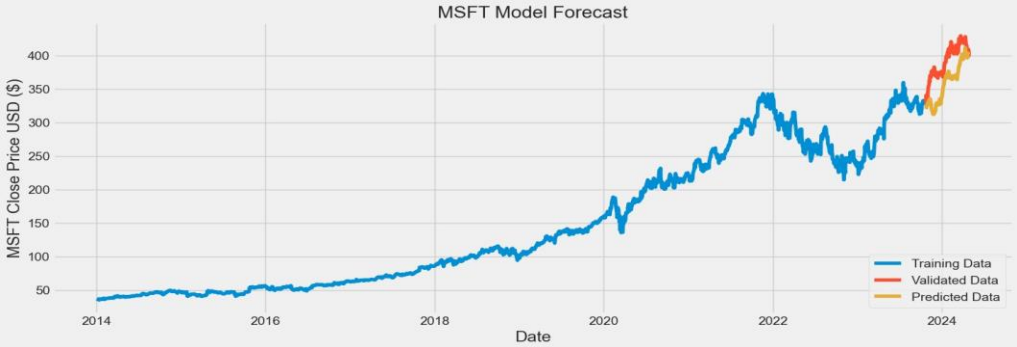
Results

Amazon

22

Results

Microsoft

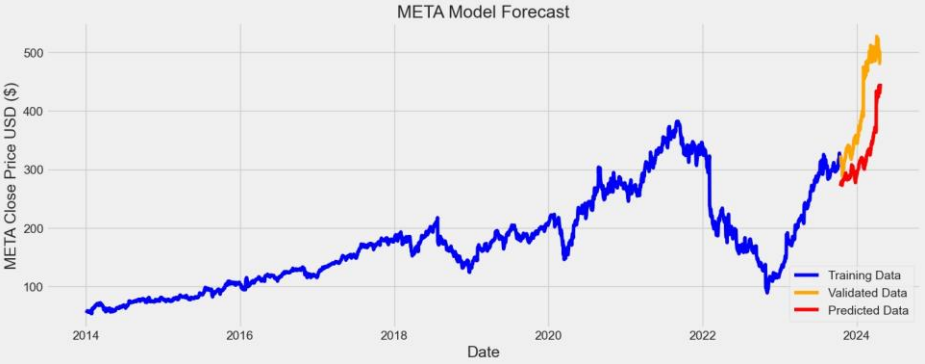


Results

Tesla

Results

Meta



25

Results

Company	Data	Linear regression	ARIMA	LSTM	GRU
Apple	Training set	22.2606	15.82	0.2183	0.2004
	Testing set	16.4470	16.2	0.1517	0.1487
Tesla	Training set	65.5160	43.4171	0.1123	0.1101
	Testing set	58.4098	52.7421	0.1932	0.1713
Microsoft	Training set	33.3502	19.9436	0.08918	0.0910
	Testing set	42.1783	23.2365	0.1432	0.1375
Meta	Training set	34.4927	68.7713	0.1820	0.1677
	Testing set	106.8925	132.8243	0.2496	0.2284
Amazon	Training set	21.2181	18.3560	0.1166	0.1094
	Testing set	32.928	24.2342	0.1703	0.1607

26

Challenges

FEATURES ARE HIGHLY CORRELATED.

CAME UP WITH PCA.

Lessons Learned & Future Work

Lessons Learned:

- RNN,s are the better models to Forecast the stock patterns.
- Stock Forecasting Accuracy increases proportionally to the number of factors considered.

Future Work:

- Data Enhancement: Using macroeconomic indicators like GDP, Inflationetc
- Risk Management: Integrate risk management techniques into the forecasting framework, such as Value at Risk (VaR) analysis or stress testing.

Conclusion

Though any model cannot guarantee you with the exact predictions.

It can help investors and traders amplify investment strategies and better risk management practices in the financial markets.

29

30

References

- Stock Closing Price Prediction using Machine Learning Techniques(Mehar Vijh et al.,2020)
- Stock Market Predictions with LSTM in Python. (Avijeet Bishwal, 2024)

Acknowledgments

- We have the utmost gratitude for Prof. Khald's leadership and TA, knowledge, and support during the course of this endeavor. His astute counsel and unwavering assistance profoundly influenced the project's course and results.


THANK YOU

APPENDIX C: SOURCE CODE

```
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Set the seaborn style and matplotlib inline
sns.set_style('whitegrid')
plt.style.use("fivethirtyeight")
%matplotlib inline

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from datetime import datetime

def collect_data(symbol, start_date):
    end_date = datetime.now().strftime('%Y-%m-%d')
    stock_data = yf.download(symbol, start=start_date, end=end_date)
    return stock_data

from sklearn.metrics import mean_squared_error
import math

def plot_predictions(stock, data, training_data_len):
    # Prepare the data
    data['Date'] = data.index
    data.reset_index(drop=True, inplace=True)
    data['Date'] = pd.to_datetime(data['Date'])
    data['Date_Ordinal'] = data['Date'].apply(lambda date: date.toordinal())

    # Split the data into training and testing sets
    X = data[['Date_Ordinal']].values.reshape(-1, 1)
    y = data['Close'].values
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
      X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2,
random_state=42)

    # Train the linear regression model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Make predictions
    data['Predictions'] = model.predict(X)

    # Split the data for visualization
    train = data[:training_data_len]
    valid = data[training_data_len:]

    # Visualize the data
    plt.figure(figsize=(16,6))
```

```
    title = stock + ' Model Forecast'
    ylabel = stock + ' Close Price USD ($)'
    plt.title(title)
    plt.xlabel('Date', fontsize=18)
    plt.ylabel(ylabel, fontsize=18)
    plt.plot(train['Date'], train['Close'], label='Training Data', color='blue')
    plt.plot(valid['Date'], valid['Close'], label='Validated Data', color='red')
    plt.plot(valid['Date'], valid['Predictions'], label='Predicted Data', color='orange')
    plt.legend(loc='lower right')
    plt.show()

    return train, valid  # Return training and validation data
def calculate_rmse(y_true, y_pred):
    return math.sqrt(mean_squared_error(y_true, y_pred))
# Test the function
data = collect_data('INTC', '2014-01-01')
training_data_len = int(len(data) * 0.8)  # 80% of data for training
train, valid = plot_predictions('INTC', data, training_data_len)

# Calculate RMSE for training data
train_rmse = calculate_rmse(train['Close'], train['Predictions'])
print("Training RMSE:", train_rmse)

# Calculate RMSE for testing data
test_rmse = calculate_rmse(valid['Close'], valid['Predictions'])
print("Testing RMSE:", test_rmse)

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the
input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
plt.style.use("fivethirtyeight")
%matplotlib inline

# For reading stock data from yahoo
from pandas_datareader.data import DataReader

# For time stamps
from datetime import datetime
```

```
from math import sqrt
from math import sqrt
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler

#ignore the warnings
import warnings
warnings.filterwarnings('ignore')

try:
   from keras.models import Sequential
   from keras.layers import Dense, LSTM
   import tensorflow as tf
except:
   !pip install --upgrade tensorflow
   !pip install keras
   !pip install --ignore-installed --upgrade tensorflow-gpu
   !pip install wrapt==1.11.1
   from keras.models import Sequential
   from keras.layers import Dense, LSTM
   import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, LSTM
import tensorflow as tf
pip install yfinance

import yfinance as yf
from datetime import datetime

def collect_data(symbol, start_date):
   end_date = datetime.now().strftime('%Y-%m-%d')
   stock_data = yf.download(symbol, start=start_date, end=end_date)
   return stock_data

# Test the function
df = collect_data('AMD', '2014-01-01')
print(type(df))

def plot_close_val(data_frame, column, stock):
   plt.figure(figsize=(16,6))
   plt.title(column + ' Price History for ' + stock )
   plt.plot(data_frame[column])
   plt.xlabel('Date', fontsize=18)
   plt.ylabel(column + ' Price USD ($) for ' + stock, fontsize=18)
   plt.show()

#Test the function
plot_close_val(df, 'Close', 'AMD')
plot_close_val(df, 'Open',  'AMD')

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

```
df['Returns'] = (df['Close']/df['Close'].shift(1))-1

# Drop the NaN values created by the percentage change
df = df.dropna()

# Display the updated dataset with the 'Returns' column
print(df.head())

features = df[['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume', 'Returns']]

scalar=StandardScaler()
scalar.fit(features)

scaled_data=scalar.transform(features)
pca=PCA(n_components=1)
pca.fit(scaled_data)
x_pca=pca.transform(scaled_data)
x_pca.shape
x_pca
pca_columns = ['Principal_Component']  # Customize column names if needed
x_pca_df = pd.DataFrame(x_pca, columns=pca_columns)
print(x_pca_df.head())
def build_training_dataset(input_ds):
   # Create a new dataframe with only the 'Close column
   input_ds.reset_index()
   data = input_ds.filter(items=['Close'])
   # Convert the dataframe to a numpy array
   dataset = data.values
   # Get the number of rows to train the model on
   training_data_len = int(np.ceil( len(dataset) * .95 ))
   return data, dataset, training_data_len

#Test the function
training_data_df, training_dataset_np, training_data_len = build_training_dataset(df)
dataset=training_dataset_np
data=training_data_df
# Scale the data
from sklearn.preprocessing import MinMaxScaler
def scale_the_data(dataset):
   scaler = MinMaxScaler(feature_range=(0,1))
   scaled_data = scaler.fit_transform(dataset)
   return scaler, scaled_data

#Test the function
scaler, scaled_data = scale_the_data(training_dataset_np)
def build_training_dataset_pca(input_ds):
   # Create a new dataframe with only the 'Close column
   input_ds.reset_index()
   data_pca = input_ds.filter(items=['Principal_Component'])
   # Convert the dataframe to a numpy array
   dataset_pca = data.values
   # Get the number of rows to train the model on
   training_pca_data_len = int(np.ceil( len(dataset) * .95 ))
```

```
    return dataset_pca, training_pca_data_len

#Test the function
training_pca_dataset_np, training_pca_data_len = build_training_dataset_pca(x_pca_df)

# Scale the data
from sklearn.preprocessing import MinMaxScaler
def scale_the_data(dataset):
    scaler = MinMaxScaler(feature_range=(0,1))
    scaled_data = scaler.fit_transform(dataset)
    return scaler, scaled_data

#Test the function
scaler_pca, scaled_data_pca = scale_the_data(training_pca_dataset_np)
# Create the training data set
# Create the scaled training data set
def split_train_dataset(training_data_len):
    train_data = scaled_data_pca[0:int(training_data_len), :]
    # Split the data into x_train and y_train data sets
    x_train = []
    y_train = []
    for i in range(60, len(train_data)):
        x_train.append(train_data[i-60:i, 0])
        y_train.append(train_data[i, 0])
        if i<= 61:
            #print(x_train)
            #print(y_train)
            print('.')

    # Convert the x_train and y_train to numpy arrays
    x_train, y_train = np.array(x_train), np.array(y_train)

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

#Test the function
x_train,y_train = split_train_dataset(training_pca_data_len)
def build_lstm_model(x_train,y_train):
    # Build the LSTM model
    model = Sequential()
    model.add(LSTM(128, return_sequences=True, input_shape= (x_train.shape[1], 1)))
    model.add(LSTM(64, return_sequences=False))
    model.add(Dense(25))
    model.add(Dense(1))

    # Compile the model
    model.compile(optimizer='adam', loss='mean_squared_error')
    # Train the model
    model.fit(x_train, y_train, batch_size=1, epochs=1)
    return model
```

```
#Test the function
lstm_model = build_lstm_model(x_train,y_train)
def create_testing_data_set(model, scaler, training_data_len, test_data_len):
    # Create the testing data set
    # Assuming test_data is a 2D array
    test_data = scaled_data[training_data_len - test_data_len:, :]
    # Create the data sets x_test and y_test
    x_test = []
    y_test = dataset[training_data_len:, :]
    for i in range(test_data_len, len(test_data)):
        x_test.append(test_data[i - test_data_len:i, 0])

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

    # Reshape the data to match model's expected input shape
    x_test = x_test[:, :60]  # Select the first 60 timesteps

    # Get the models predicted price values
    predictions = model.predict(x_test)
    predictions = scaler.inverse_transform(predictions)

    # Get the root mean squared error (RMSE)
    rmse = np.sqrt(np.mean(((predictions - y_test) ** 2)))
    return x_test, y_test, predictions, rmse

# Test the function
TEST_DATA_LENGTH = 100
x_test, y_test, predictions, rmse = create_testing_data_set(lstm_model, scaler,
training_data_len, TEST_DATA_LENGTH)
def plot_predictions(stock, data,training_data_len):
    #Plot the data
    train = data[:training_data_len]
    valid = data[training_data_len:]
    valid['Predictions'] = predictions
    # Visualize the data
    plt.figure(figsize=(16,6))
    title = stock + ' Model Forecast'
    ylabel = stock + ' Close Price USD ($)'
    plt.title(title)
    plt.xlabel('Date', fontsize=18)
    plt.ylabel(ylabel, fontsize=18)
    plt.plot(train['Close'])
    plt.plot(valid[['Close', 'Predictions']])
    plt.legend(['Training Data', 'Validated Data', 'Predicted Data'], loc='lower right')
    plt.show()
    return valid

#Test the function
valid = plot_predictions('AMD',data,training_data_len)

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

```
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
plt.style.use("fivethirtyeight")
%matplotlib inline

# For reading stock data from yahoo
from pandas_datareader.data import DataReader

# For time stamps
from datetime import datetime
from math import sqrt
from math import sqrt
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler

#ignore the warnings
import warnings
warnings.filterwarnings('ignore')

try:
    from keras.models import Sequential
    from keras.layers import Dense, LSTM
    import tensorflow as tf
except:
    !pip install --upgrade tensorflow
    !pip install keras
    !pip install --ignore-installed --upgrade tensorflow-gpu
    !pip install wrapt==1.11.1
    from keras.models import Sequential
    from keras.layers import Dense, LSTM
    import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense
from statsmodels.tsa.arima.model import ARIMA
import tensorflow as tf

pip install yfinance

import yfinance as yf
```

```
from datetime import datetime

def collect_data(symbol, start_date):
    end_date = datetime.now().strftime('%Y-%m-%d')
    stock_data = yf.download(symbol, start=start_date, end=end_date)
    return stock_data

# Test the function
df = collect_data('AMD', '2014-01-01')
print(type(df))

def plot_close_val(data_frame, column, stock):
    plt.figure(figsize=(16,6))
    plt.title(column + ' Price History for ' + stock )
    plt.plot(data_frame[column])
    plt.xlabel('Date', fontsize=18)
    plt.ylabel(column + ' Price USD ($) for ' + stock, fontsize=18)
    plt.show()

#Test the function
plot_close_val(df, 'Close', 'AMD')
plot_close_val(df, 'Open',  'AMD')

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

df['Returns'] = (df['Close']/df['Close'].shift(1))-1

# Drop the NaN values created by the percentage change
df = df.dropna()

# Display the updated dataset with the 'Returns' column
print(df.head())
features = df[['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume', 'Returns']]
scalar=StandardScaler()
scalar.fit(features)
scaled_data=scalar.transform(features)
pca=PCA(n_components=1)
pca.fit(scaled_data)
x_pca=pca.transform(scaled_data)
pca=PCA(n_components=1)
pca.fit(scaled_data)
x_pca=pca.transform(scaled_data)
x_pca.shape
x_pca
pca_columns = ['Principal_Component']  # Customize column names if needed
x_pca_df = pd.DataFrame(x_pca, columns=pca_columns)
print(x_pca_df.head())
def build_training_dataset(input_ds):
    # Create a new dataframe with only the 'Close column
    input_ds.reset_index()
    data = input_ds.filter(items=['Close'])
    # Convert the dataframe to a numpy array
```

```
    dataset = data.values
    # Get the number of rows to train the model on
    training_data_len = int(np.ceil( len(dataset) * .95 ))
    return data, dataset, training_data_len

#Test the function
training_data_df, training_dataset_np, training_data_len = build_training_dataset(df)
dataset=training_dataset_np
data=training_data_df
# Scale the data
from sklearn.preprocessing import MinMaxScaler
def scale_the_data(dataset):
    scaler = MinMaxScaler(feature_range=(0,1))
    scaled_data = scaler.fit_transform(dataset)
    return scaler, scaled_data

#Test the function
scaler, scaled_data = scale_the_data(training_dataset_np)
def build_training_dataset_pca(input_ds):
    # Create a new dataframe with only the 'Close column
    input_ds.reset_index()
    data_pca = input_ds.filter(items=['Principal_Component'])
    # Convert the dataframe to a numpy array
    dataset_pca = data.values
    # Get the number of rows to train the model on
    training_pca_data_len = int(np.ceil( len(dataset) * .95 ))
    return dataset_pca, training_pca_data_len

#Test the function
training_pca_dataset_np,training_pca_data_len = build_training_dataset_pca(x_pca_df)
# Scale the data
from sklearn.preprocessing import MinMaxScaler
def scale_the_data(dataset):
    scaler = MinMaxScaler(feature_range=(0,1))
    scaled_data = scaler.fit_transform(dataset)
    return scaler, scaled_data

#Test the function
scaler_pca, scaled_data_pca = scale_the_data(training_pca_dataset_np)
# Import ARIMA model
from statsmodels.tsa.arima.model import ARIMA

# Function to build ARIMA training dataset
def build_arima_training_dataset(dataset):
    return dataset.flatten()

# Test the function
arima_training_dataset = build_arima_training_dataset(training_pca_dataset_np)

# Function to scale the ARIMA training dataset
def scale_arima_data(dataset):
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(dataset.reshape(-1, 1))
```

```
    return scaler, scaled_data.flatten()

# Test the function
scaler_arima, scaled_arima_data = scale_arima_data(arima_training_dataset)
from statsmodels.tsa.arima.model import ARIMA

# Function to build and train ARIMA model
def build_arima_model(dataset):
    # Fit ARIMA model
    arima_model = ARIMA(dataset, order=(5,1,0)) # Example order, you might need to tune it
    arima_model_fit = arima_model.fit()
    return arima_model_fit

# Test the function
arima_model_fit = build_arima_model(scaled_arima_data)
# Function to create testing dataset and make predictions using ARIMA model
def create_arima_predictions(arima_model_fit, scaler, scaled_data, training_data_len,
test_data_len):
    # Get the scaled test data
    test_data = scaled_data[-test_data_len:].reshape(-1, 1)

    # Make predictions
    predictions = arima_model_fit.forecast(steps=test_data_len)

    # Inverse scale the predictions
    predictions = scaler.inverse_transform(predictions.reshape(-1, 1)).flatten()

    # Get the root mean squared error (RMSE)
    y_test = scaler.inverse_transform(test_data).flatten()

    rmse = np.sqrt(np.mean((predictions - y_test) ** 2))

    return predictions, y_test, rmse

# Test the function
TEST_DATA_LENGTH = 100
predictions_arima, y_test_arima, rmse_arima = create_arima_predictions(arima_model_fit,
scaler_arima, scaled_arima_data, training_data_len, TEST_DATA_LENGTH)
import matplotlib.pyplot as plt

def plot_predictions_arima(stock, data, predictions, y_test, training_data_len):
    # Plot the data
    train = data[:training_data_len]
    valid = data[training_data_len:training_data_len+len(y_test)]
    valid['Predictions'] = predictions

    # Visualize the data
    plt.figure(figsize=(16, 6))
    plt.title(stock + ' Model Forecast')
    plt.xlabel('Date', fontsize=18)
    plt.ylabel(stock + ' Close Price USD ($)', fontsize=18)
    plt.plot(train['Close'])
    plt.plot(valid[['Close', 'Predictions']])
```

```
    plt.legend(['Training Data', 'Actual Data', 'Predicted Data'], loc='lower right')
    plt.show()
    return valid

# Test the function
plot_predictions_arima('AMD', data, predictions_arima, y_test_arima, training_data_len)
# Calculate RMSE for training and test sets
def calculate_rmse(predictions, actual):
    if len(actual) == 0:
        return None
    return np.sqrt(mean_squared_error(predictions, actual))
train_predictions = predictions_arima[:training_data_len]
train_actual = y_test_arima[:training_data_len]
test_predictions = predictions_arima[training_data_len:]
test_actual = y_test_arima[training_data_len:]
# Calculate RMSE for training set
train_rmse = calculate_rmse(train_predictions, train_actual)

if train_rmse is not None:
    print("RMSE for training set:", train_rmse)
else:
    print("Not enough training data available to calculate RMSE.")

# Calculate RMSE for test set
test_rmse = calculate_rmse(test_predictions, test_actual)
if test_rmse is not None:
    print("RMSE for test set:", test_rmse)
else:
    print("Not enough test data available to calculate RMSE.")
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the
input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
plt.style.use("fivethirtyeight")
%matplotlib inline

# For reading stock data from yahoo
from pandas_datareader.data import DataReader

# For time stamps
```

```
from datetime import datetime
from math import sqrt
from math import sqrt
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler

#ignore the warnings
import warnings
warnings.filterwarnings('ignore')
try:
   from keras.models import Sequential
   from keras.layers import Dense, LSTM
   import tensorflow as tf
except:
   !pip install --upgrade tensorflow
   !pip install keras
   !pip install --ignore-installed --upgrade tensorflow-gpu
   !pip install wrapt==1.11.1
   from keras.models import Sequential
   from keras.layers import Dense, LSTM
   import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, LSTM
from keras.layers import GRU, Dense
import tensorflow as tf
pip install yfinance
import yfinance as yf
from datetime import datetime

def collect_data(symbol, start_date):
   end_date = datetime.now().strftime('%Y-%m-%d')
   stock_data = yf.download(symbol, start=start_date, end=end_date)
   return stock_data

# Test the function
df = collect_data('TSLA', '2014-01-01')
print(type(df))

def plot_close_val(data_frame, column, stock):
   plt.figure(figsize=(16,6))
   plt.title(column + ' Price History for ' + stock )
   plt.plot(data_frame[column])
   plt.xlabel('Date', fontsize=18)
   plt.ylabel(column + ' Price USD ($) for ' + stock, fontsize=18)
   plt.show()

#Test the function
plot_close_val(df, 'Close', 'TSLA')
plot_close_val(df, 'Open',  'TSLA')
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
df['Returns'] = (df['Close']/df['Close'].shift(1))-1
```

```
# Drop the NaN values created by the percentage change
df = df.dropna()

# Display the updated dataset with the 'Returns' column
print(df.head())
features = df[['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume', 'Returns']]
scalar=StandardScaler()
scalar.fit(features)

scaled_data=scalar.transform(features)
pca=PCA(n_components=1)
pca.fit(scaled_data)
x_pca=pca.transform(scaled_data)
x_pca.shape
x_pca
pca_columns = ['Principal_Component']  # Customize column names if needed
x_pca_df = pd.DataFrame(x_pca, columns=pca_columns)
print(x_pca_df.head())
def build_training_dataset(input_ds):
   # Create a new dataframe with only the 'Close column
   input_ds.reset_index()
   data = input_ds.filter(items=['Close'])
   # Convert the dataframe to a numpy array
   dataset = data.values
   # Get the number of rows to train the model on
   training_data_len = int(np.ceil( len(dataset) * .95 ))
   return data, dataset, training_data_len

#Test the function
training_data_df, training_dataset_np, training_data_len = build_training_dataset(df)
dataset=training_dataset_np
data=training_data_df
# Scale the data
from sklearn.preprocessing import MinMaxScaler
def scale_the_data(dataset):
   scaler = MinMaxScaler(feature_range=(0,1))
   scaled_data = scaler.fit_transform(dataset)
   return scaler, scaled_data

#Test the function
scaler, scaled_data = scale_the_data(training_dataset_np)
def build_training_dataset_pca(input_ds):
   # Create a new dataframe with only the 'Close column
   input_ds.reset_index()
   data_pca = input_ds.filter(items=['Principal_Component'])
   # Convert the dataframe to a numpy array
   dataset_pca = data.values
   # Get the number of rows to train the model on
   training_pca_data_len = int(np.ceil( len(dataset) * .95 ))
   return dataset_pca, training_pca_data_len

#Test the function
training_pca_dataset_np, training_pca_data_len = build_training_dataset_pca(x_pca_df)
```

```
# Scale the data
from sklearn.preprocessing import MinMaxScaler
def scale_the_data(dataset):
    scaler = MinMaxScaler(feature_range=(0,1))
    scaled_data = scaler.fit_transform(dataset)
    return scaler, scaled_data

#Test the function
scaler_pca, scaled_data_pca = scale_the_data(training_pca_dataset_np)
# Create the training data set
# Create the scaled training data set
def split_train_dataset(training_data_len):
    train_data = scaled_data_pca[0:int(training_data_len), :]
    # Split the data into x_train and y_train data sets
    x_train = []
    y_train = []
    for i in range(60, len(train_data)):
        x_train.append(train_data[i-60:i, 0])
        y_train.append(train_data[i, 0])
        if i<= 61:
            #print(x_train)
            #print(y_train)
            print('.')

    # Convert the x_train and y_train to numpy arrays
    x_train, y_train = np.array(x_train), np.array(y_train)

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

#Test the function
x_train,y_train = split_train_dataset(training_pca_data_len)
def build_gru_model(x_train, y_train):
    # Build the GRU model
    model = Sequential()
    model.add(GRU(128, return_sequences=True, input_shape=(x_train.shape[1], 1)))
    model.add(GRU(64, return_sequences=False))
    model.add(Dense(25))
    model.add(Dense(1))

    # Compile the model
    model.compile(optimizer='adam', loss='mean_squared_error')
    # Train the model
    model.fit(x_train, y_train, batch_size=1, epochs=1)
    return model

# Test the function
gru_model = build_gru_model(x_train, y_train)
def create_testing_data_set(model, scaler, training_data_len, test_data_len):
    # Create the testing data set
    # Assuming test_data is a 2D array
```

```
    test_data = scaled_data[training_data_len - test_data_len:, :]
    # Create the data sets x_test and y_test
    x_test = []
    y_test = dataset[training_data_len:, :]
    for i in range(test_data_len, len(test_data)):
        x_test.append(test_data[i - test_data_len:i, 0])

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

    # Reshape the data to match model's expected input shape
    x_test = x_test[:, :60]  # Select the first 60 timesteps

    # Get the models predicted price values
    predictions = model.predict(x_test)
    predictions = scaler.inverse_transform(predictions)

    # Get the root mean squared error (RMSE)
    rmse = np.sqrt(np.mean(((predictions - y_test) ** 2)))
    return x_test, y_test, predictions, rmse

# Test the function
TEST_DATA_LENGTH = 100
x_test, y_test, predictions, rmse = create_testing_data_set(gru_model, scaler, training_data_len,
TEST_DATA_LENGTH)
def plot_predictions(stock, data,training_data_len):
    #Plot the data
    train = data[:training_data_len]
    valid = data[training_data_len:]
    valid['Predictions'] = predictions
    # Visualize the data
    plt.figure(figsize=(16,6))
    title = stock + ' Model Forecast'
    ylabel = stock + ' Close Price USD ($)'
    plt.title(title)
    plt.xlabel('Date', fontsize=18)
    plt.ylabel(ylabel, fontsize=18)
    plt.plot(train['Close'])
    plt.plot(valid[['Close', 'Predictions']])
    plt.legend(['Training Data', 'Validated Data', 'Predicted Data'], loc='lower right')
    plt.show()
    return valid

#Test the function
valid = plot_predictions('TSLA',data,training_data_len)
data = x_pca_df

# Split the data into train and test sets (for simplicity, let's assume a 80-20 split)
train_size = int(len(data) * 0.8)
train, test = data.iloc[:train_size], data.iloc[train_size:]

# Normalize the data using Min-Max scaling
scaler = MinMaxScaler()
```

```
train_scaled = scaler.fit_transform(train)
test_scaled = scaler.transform(test)

# Define a function for Time Series Cross-Validation
def time_series_cv(model_type, X_train, y_train, X_test, y_test):
    model = Sequential()
    if model_type == 'LSTM':
        model.add(LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2])))
    elif model_type == 'GRU':
        model.add(GRU(50, input_shape=(X_train.shape[1], X_train.shape[2])))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mean_squared_error')

    # Fit the model
    model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)

    # Make predictions
    train_predict = model.predict(X_train)
    test_predict = model.predict(X_test)

    # Invert predictions to original scale
    train_predict = scaler.inverse_transform(train_predict.reshape(-1, 1))
    y_train = scaler.inverse_transform(y_train.reshape(-1, 1))
    test_predict = scaler.inverse_transform(test_predict.reshape(-1, 1))
    y_test = scaler.inverse_transform(y_test.reshape(-1, 1))

    # Calculate RMSE
    train_score = np.sqrt(mean_squared_error(y_train, train_predict))
    test_score = np.sqrt(mean_squared_error(y_test, test_predict))

    return train_score, test_score

# Prepare the data for LSTM and GRU models
def prepare_data(data, n_steps):
    X, y = [], []
    for i in range(len(data) - n_steps):
        end_ix = i + n_steps
        X.append(data[i:end_ix])
        y.append(data[end_ix])
    return np.array(X), np.array(y)

# Choose the number of time steps (look-back period)
n_steps = 3

# Prepare the data
X_train, y_train = prepare_data(train_scaled, n_steps)
X_test, y_test = prepare_data(test_scaled, n_steps)

# Perform Time Series Cross-Validation for LSTM
lstm_train_score, lstm_test_score = time_series_cv('LSTM', X_train, y_train, X_test, y_test)
print("LSTM Train RMSE:", lstm_train_score)
print("LSTM Test RMSE:", lstm_test_score)
```

```
# Perform Time Series Cross-Validation for GRU
gru_train_score, gru_test_score = time_series_cv('GRU', X_train, y_train, X_test, y_test)
print("GRU Train RMSE:", gru_train_score)
print("GRU Test RMSE:", gru_test_score)
```