

STOCK MARKET PREDICTIONS

USING MACHINE AND DEEP LEARNING

ALGORITHMS

REPORT

# 

REPORT A PROJECT

SUBMITTED TO

*in partial fulfillment for the award of the degree of*

MASTER OF SCIENCE  
 (Data Science)

## Submitted by

**<SWETA HAZRA>**

[Reg.No:22254090023]

# UNDER THE GUIDANCE OF

<Dr. Shashikant Deepak>

## University Project Mentor

<Dr. Vishwajeet Singh>

**JAN 2025**

**TABLE OF CONTENTS**

**LIST OF TABLES LIST OF FIGURES**

**LIST OF SYMBOLS AND ABBREVIATIONS**

# 1.0 INTRODUCTION 1

* 1. Sub section 1
  2. Sub section 1

# 2.0 PROBLEM STATEMEN 2

* 1. Sub section 2
  2. Sub section 2

## 3.0

## 4.0

# LITERATURE REVIEW 3

# RESEARCH OBJECTIVES 4

# METHODOLOGY 5

* 1. Sub section 5
  2. Sub section 5

# FINDINGS 6

* 1. Sub section 6
  2. Sub section **6**

## 7.0

## 8.0

# CONCLUSION 7

# LIMITATIONS 8

* 1. **FUTURE SCOPE (**optional**) 9**

## 10.0 REFERENCES 10

## 11.0 ANNEXURE 11

**P.S. Do not refer to the page no.s. Editing page no.s was throwing errors so I left them as it is.**

|  |  |  |
| --- | --- | --- |
|  | **LIST OF TABLES** |  |
| **Table No.** | **Title** | **Page No.** |
| Table 1.1 | Name of the table | 1 |
| Table 2.1 | Name of the table | 2 |
| Table 3.1 | Name of the table |  |

Table 1: describe, columns, info, isna.sum functions carried out on 5 company stock data in a for loop

Name: Google

Adj Close Close High Low Open \

count 2768.000000 2768.000000 2768.000000 2768.000000 2768.000000

mean 64.529631 64.761492 65.408672 64.085334 64.724876

std 37.917258 38.053498 38.491265 37.614899 38.027166

min 17.443457 17.506132 17.567652 17.323069 17.550716

25% 32.464976 32.581625 33.049937 32.207438 32.726626

50% 53.928230 54.122000 54.708374 53.489250 54.111750

75% 93.832840 94.169998 95.226625 92.572376 93.956627

max 150.169418 150.709000 152.100006 149.887497 151.863495

Volume

count 2.768000e+03

mean 3.833840e+07

std 2.613485e+07

min 1.584340e+05

25% 2.365258e+07

50% 3.025065e+07

75% 4.309665e+07

max 4.643901e+08

Index(['Adj Close', 'Close', 'High', 'Low', 'Open', 'Volume'], dtype='object')

<class 'pandas.core.frame.DataFrame'>

Index: 2768 entries, 02-01-2013 to 29-12-2023

Data columns (total 6 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Adj Close 2768 non-null float64

1 Close 2768 non-null float64

2 High 2768 non-null float64

3 Low 2768 non-null float64

4 Open 2768 non-null float64

5 Volume 2768 non-null int64

dtypes: float64(5), int64(1)

memory usage: 151.4+ KB

None

Adj Close 0

Close 0

High 0

Low 0

Open 0

Volume 0

dtype: int64

Name: Microsoft

Adj Close Close High Low Open \

count 2768.000000 2768.000000 2768.000000 2768.000000 2768.000000

mean 133.032109 139.171243 140.500921 137.699050 139.118212

std 102.708961 102.621768 103.656363 101.486758 102.591663

min 21.465899 26.459999 26.750000 26.280001 26.490000

25% 41.721787 48.579999 48.917499 48.072501 48.575002

50% 94.598324 101.085003 101.805000 99.574997 101.094997

75% 228.756260 235.532505 238.005001 232.392498 235.269997

max 379.859467 382.700012 384.299988 378.160004 383.760010

Volume

count 2.768000e+03

mean 3.183635e+07

std 1.640070e+07

min 7.425600e+06

25% 2.221302e+07

50% 2.787770e+07

75% 3.639475e+07

max 2.484285e+08

Index(['Adj Close', 'Close', 'High', 'Low', 'Open', 'Volume'], dtype='object')

<class 'pandas.core.frame.DataFrame'>

Index: 2768 entries, 02-01-2013 to 29-12-2023

Data columns (total 6 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Adj Close 2768 non-null float64

1 Close 2768 non-null float64

2 High 2768 non-null float64

3 Low 2768 non-null float64

4 Open 2768 non-null float64

5 Volume 2768 non-null int64

dtypes: float64(5), int64(1)

memory usage: 151.4+ KB

None

Adj Close 0

Close 0

High 0

Low 0

Open 0

Volume 0

dtype: int64

Name: Amazon

Adj Close Close High Low Open \

count 2768.000000 2768.000000 2768.000000 2768.000000 2768.000000

mean 108.495195 144.742478 145.829684 143.641022 144.718202

std 12.934705 21.476378 21.426226 21.521179 21.455469

min 72.550064 90.602295 93.441681 86.577438 90.439774

25% 100.020134 129.811188 130.665024 128.926868 129.824699

50% 106.582836 140.143402 141.018166 139.139572 140.152969

75% 116.725792 155.185185 156.147232 154.455063 155.130966

max 158.963486 206.309753 206.405350 204.875717 205.908218

Volume

count 2.768000e+03

mean 4.811328e+06

std 2.825571e+06

min 1.247878e+06

25% 3.310930e+06

50% 4.123528e+06

75% 5.360828e+06

max 3.981442e+07

Index(['Adj Close', 'Close', 'High', 'Low', 'Open', 'Volume'], dtype='object')

<class 'pandas.core.frame.DataFrame'>

Index: 2768 entries, 02-01-2013 to 29-12-2023

Data columns (total 6 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Adj Close 2768 non-null float64

1 Close 2768 non-null float64

2 High 2768 non-null float64

3 Low 2768 non-null float64

4 Open 2768 non-null float64

5 Volume 2768 non-null int64

dtypes: float64(5), int64(1)

memory usage: 151.4+ KB

None

Adj Close 0

Close 0

High 0

Low 0

Open 0

Volume 0

dtype: int64

Name: Apple

Adj Close Close High Low Open \

count 2768.000000 2768.000000 2768.000000 2768.000000 2768.000000

mean 78.078527 78.078527 79.006802 77.114984 78.103231

std 52.929354 52.929354 53.616993 52.259118 52.967403

min 12.411500 12.411500 12.646500 12.287500 12.447000

25% 26.680125 26.680125 26.967124 26.329625 26.714751

50% 80.149502 80.149502 81.157001 79.539753 80.199997

75% 120.954998 120.954998 122.396749 119.100998 120.805372

max 186.570496 186.570496 188.654007 184.839493 187.199997

Volume

count 2.768000e+03

mean 7.832117e+07

std 4.130053e+07

min 1.762600e+07

25% 5.277205e+07

50% 6.681200e+07

75% 9.055150e+07

max 4.771220e+08

Index(['Adj Close', 'Close', 'High', 'Low', 'Open', 'Volume'], dtype='object')

<class 'pandas.core.frame.DataFrame'>

Index: 2768 entries, 02-01-2013 to 29-12-2023

Data columns (total 6 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Adj Close 2768 non-null float64

1 Close 2768 non-null float64

2 High 2768 non-null float64

3 Low 2768 non-null float64

4 Open 2768 non-null float64

5 Volume 2768 non-null int64

dtypes: float64(5), int64(1)

memory usage: 151.4+ KB

None

Adj Close 0

Close 0

High 0

Low 0

Open 0

Volume 0

dtype: int64

Name: IBM

Adj Close Close High Low Open \

count 2768.000000 2768.000000 2768.000000 2768.000000 2768.000000

mean 70.056227 72.363529 73.092193 71.567077 72.306914

std 56.618224 56.411026 56.992647 55.763624 56.353981

min 11.939034 13.947500 14.271429 13.753571 13.856071

25% 25.060904 27.588126 27.933750 27.295626 27.574999

50% 41.713114 43.958752 44.312500 43.623751 43.835001

75% 127.034472 129.612499 130.627506 127.415003 128.952499

max 197.144196 198.110001 199.619995 197.000000 198.020004

Volume

count 2.768000e+03

mean 1.612260e+08

std 1.236387e+08

min 2.404830e+07

25% 8.424670e+07

50% 1.211314e+08

75% 1.957962e+08

max 1.460852e+09

Index(['Adj Close', 'Close', 'High', 'Low', 'Open', 'Volume'], dtype='object')

<class 'pandas.core.frame.DataFrame'>

Index: 2768 entries, 02-01-2013 to 29-12-2023

Data columns (total 6 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Adj Close 2768 non-null float64

1 Close 2768 non-null float64

2 High 2768 non-null float64

3 Low 2768 non-null float64

4 Open 2768 non-null float64

5 Volume 2768 non-null int64

dtypes: float64(5), int64(1)

memory usage: 151.4+ KB

None

Adj Close 0

Close 0

High 0

Low 0

Open 0

Volume 0

dtype: int64

*4.643901e+08*

Table 2: Correlation Table to study the relation between features-open,close,high,low

Name: Google

Adj Close Close High Low Open Volume

Adj Close 1.000000 1.000000 0.999813 0.999829 0.999598 -0.401314

Close 1.000000 1.000000 0.999813 0.999829 0.999598 -0.401314

High 0.999813 0.999813 1.000000 0.999781 0.999827 -0.398037

Low 0.999829 0.999829 0.999781 1.000000 0.999815 -0.403626

Open 0.999598 0.999598 0.999827 0.999815 1.000000 -0.400167

Volume -0.401314 -0.401314 -0.398037 -0.403626 -0.400167 1.000000

Name: Microsoft

Adj Close Close High Low Open Volume

Adj Close 1.000000 0.999934 0.999804 0.999810 0.999657 -0.174242

Close 0.999934 1.000000 0.999863 0.999873 0.999719 -0.174578

High 0.999804 0.999863 1.000000 0.999831 0.999882 -0.170052

Low 0.999810 0.999873 0.999831 1.000000 0.999870 -0.178186

Open 0.999657 0.999719 0.999882 0.999870 1.000000 -0.173736

Volume -0.174242 -0.174578 -0.170052 -0.178186 -0.173736 1.000000

Name: Amazon

Adj Close Close High Low Open Volume

Adj Close 1.000000 0.458579 0.454995 0.458159 0.454590 -0.142456

Close 0.458579 1.000000 0.998949 0.998991 0.997762 -0.160711

High 0.454995 0.998949 1.000000 0.998705 0.998933 -0.145955

Low 0.458159 0.998991 0.998705 1.000000 0.998954 -0.171791

Open 0.454590 0.997762 0.998933 0.998954 1.000000 -0.157773

Volume -0.142456 -0.160711 -0.145955 -0.171791 -0.157773 1.000000

Name: Apple

Adj Close Close High Low Open Volume

Adj Close 1.000000 1.000000 0.999789 0.999800 0.999551 -0.002556

Close 1.000000 1.000000 0.999789 0.999800 0.999551 -0.002556

High 0.999789 0.999789 1.000000 0.999733 0.999817 0.003017

Low 0.999800 0.999800 0.999733 1.000000 0.999779 -0.007768

Open 0.999551 0.999551 0.999817 0.999779 1.000000 -0.001405

Volume -0.002556 -0.002556 0.003017 -0.007768 -0.001405 1.000000

Name: IBM

Adj Close Close High Low Open Volume

Adj Close 1.000000 0.999966 0.999823 0.999846 0.999682 -0.501310

Close 0.999966 1.000000 0.999863 0.999870 0.999716 -0.501035

High 0.999823 0.999863 1.000000 0.999831 0.999886 -0.498852

Low 0.999846 0.999870 0.999831 1.000000 0.999863 -0.502824

Open 0.999682 0.999716 0.999886 0.999863 1.000000 -0.500536

Volume -0.501310 -0.501035 -0.498852 -0.502824 -0.500536 1.000000

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Title** | **Page No**. |
| Figure 1.1 | Name of the figure | 1 |
| Figure 1.1 | Name of the figure | 2 |
| Figure 1.1 | Name of the figure | 3 |

Figure 1: Google Price Range first of all the prices except Volume seems to be have same pattern.

All the graphs look same. Aftermath of Covid Period seems to have given it decadewise unprecendented hike.

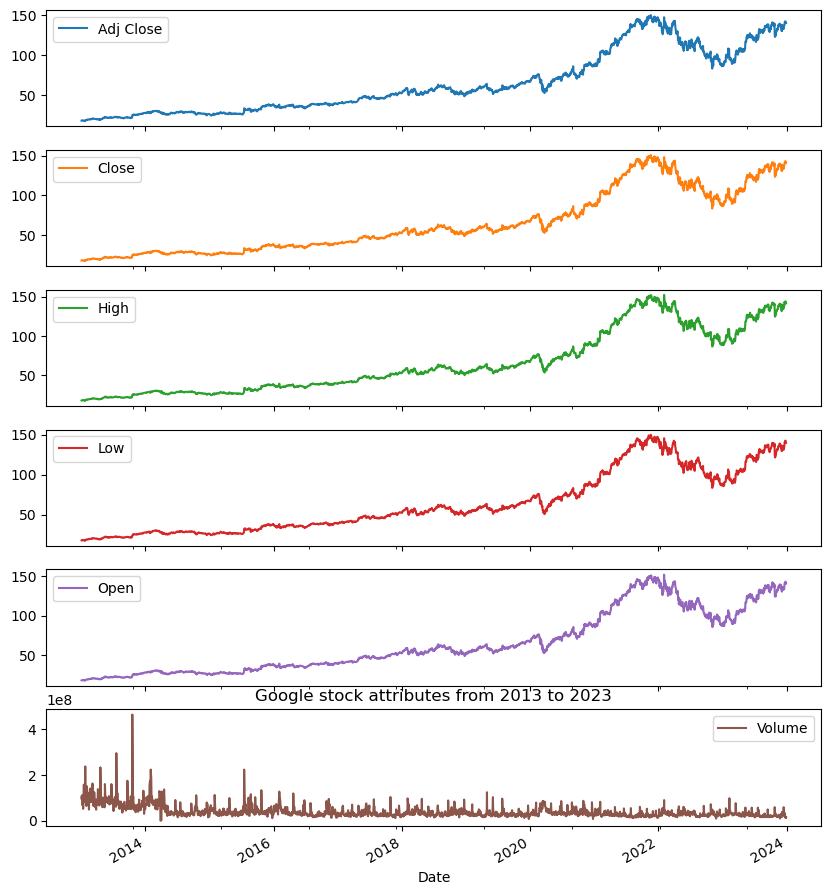


Figure 2: As seen from the Graph, there has been a steady hike in stock price of google from 2013-2019. From 2019 there accelerated increase in price till 2021(Covid Period) and then slight low till 2022 and then another high till 2023.

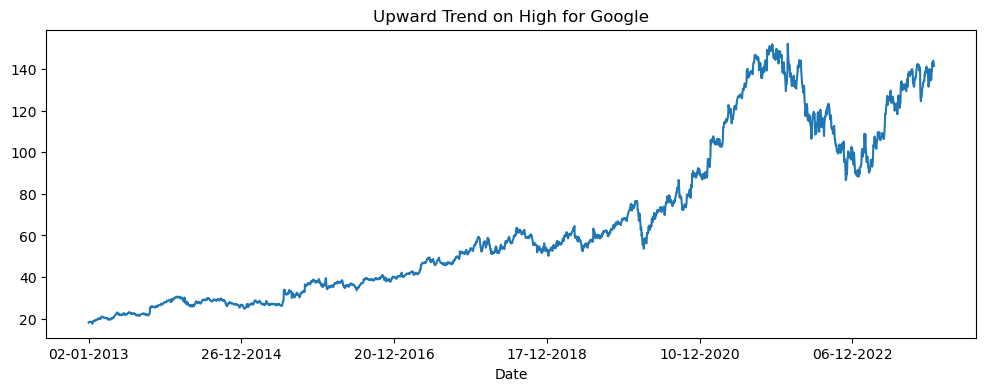
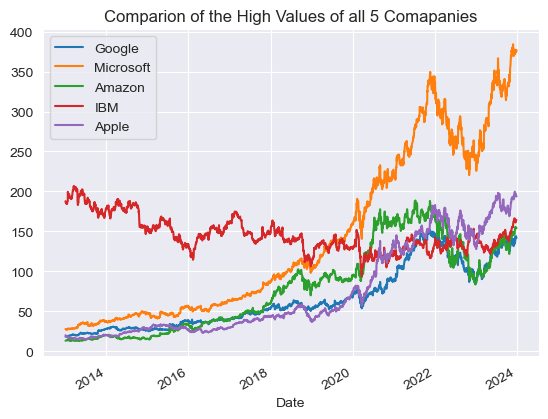


Figure 3: All Companies seem to show a huge increase in the Stock prices in this period though IBM’s case it seems less compared to others.



Comparison of High Values after normalization

Figure 4: Normalization shows a clear Trend in the graphs except Amazon.

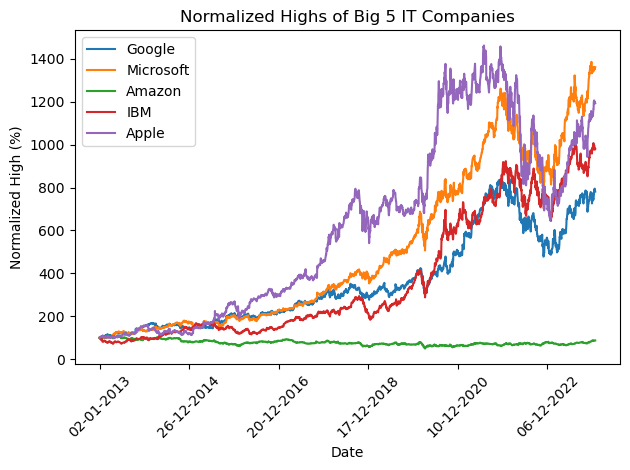


Figure 5: Range of Highs seem to avgerage in the beginning i.e. 2013 but then shows drastic hike

Till end 2020 (Covid Period) and then another rise from beginning of 2023.

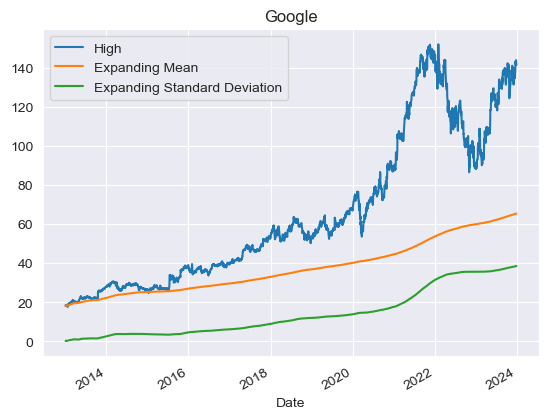
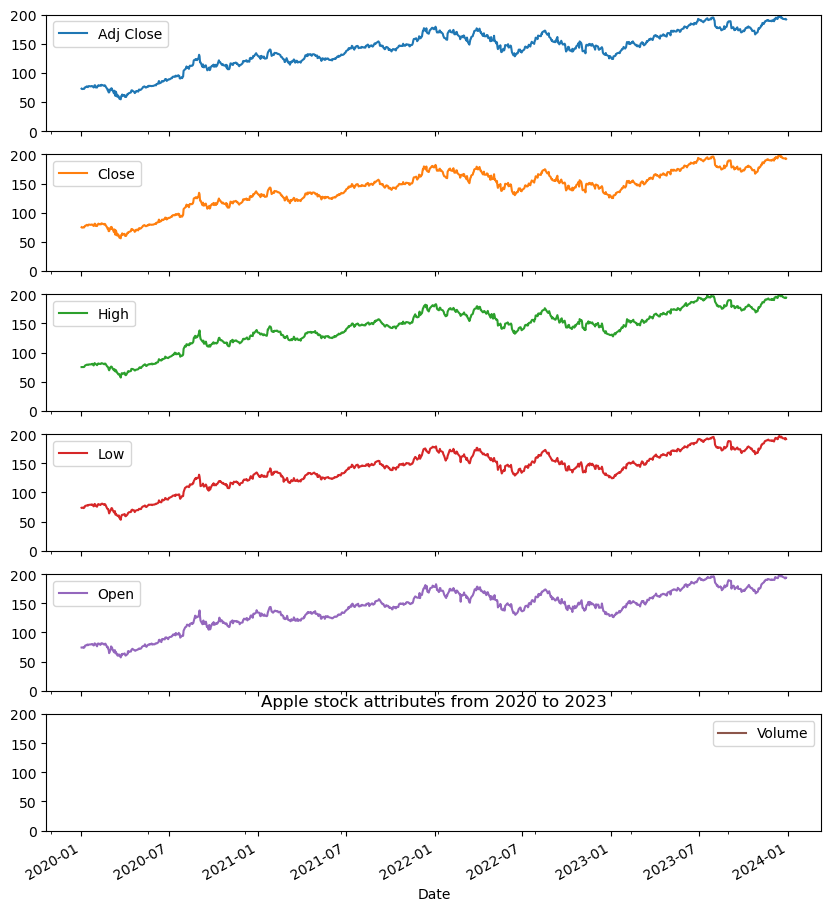


Figure 6: Apple shows overall a steady performance throughout. No drastic highs or lows.

Meaning the drastic effect of Covid seen on previous graphs is missing here. Although the

Highest Highs seems to have been achieved in 2023. So may be there is a positive effect on

Apple’s share price on the aftermath of Covid.



**LIST OF SYMBOLS AND ABBREVIATIONS**

## Symbol Explanation

Here's a list of abbreviations and symbols used in my code, along with their meanings:

**General Abbreviations**

1. **yf**: yfinance - A Python library to fetch financial data.
2. **np**: numpy - A Python library for numerical computations.
3. **pd**: pandas - A Python library for data manipulation and analysis.
4. **plt**: matplotlib.pyplot - A library for plotting graphs.
5. **sns**: seaborn - A statistical data visualization library.
6. **px**: plotly.express - A library for interactive visualizations.
7. **sm**: statsmodels.api - A library for statistical models and tests.

**Symbols and Functions**

1. **#**: Comment symbol used for annotating or disabling code.
2. **"""**: Triple quotes used for multi-line docstrings or comments.
3. **import**: Keyword to include external libraries in the code.
4. **def**: Defines a function.
5. **:**: Indicates the start of a code block (like in functions or loops).
6. **[]**: Denotes a list or is used for indexing.
7. **{}**: Denotes a dictionary.
8. **()**: Parentheses used for function calls or grouping operations.
9. **=**: Assignment operator.
10. **==**: Equality operator.
11. **.**: Attribute or method access operator.
12. **...**: Ellipsis placeholder for incomplete code.

**Data Processing Terms**

1. **companies**: Dictionary storing company names and their Google Drive file IDs.
2. **data\_frames**: Dictionary containing pandas DataFrames for each company’s stock data.
3. **df**: Common abbreviation for a pandas DataFrame.
4. **file\_id**: Unique identifier for files on Google Drive.
5. **url**: Web link to access data files.
6. **file\_name**: Local filename for downloaded stock data.
7. **parse\_dates**: Parameter to convert date strings into datetime objects.
8. **index\_col**: Specifies which column to use as the DataFrame index.

**Plotting Parameters**

1. **fig**: Figure object for visualizations.
2. **title**: Plot title.
3. **label**: Label for the legend.
4. **marginal**: Adds additional plots (e.g., box plot) to histograms.
5. **nbins**: Number of bins for histograms.
6. **opacity**: Transparency level of a plot.
7. **rotation**: Angle of axis labels for readability.

**Data Normalization & Scaling**

1. **MinMaxScaler**: Scales data to a range, typically between -1 and 1.
2. **div**: Divides values in a column by a scalar or another column.
3. **mul**: Multiplies values by a scalar.

**Statistical Terms**

1. **corr**: Correlation matrix showing relationships between columns.
2. **expanding**: Calculates expanding window statistics like mean or standard deviation.
3. **mean**: Arithmetic average.
4. **std**: Standard deviation.

**Loops and Logic**

1. **for**: Looping construct.
2. **if**: Conditional construct.
3. **try**: Attempts code execution, handles exceptions with except.
4. **break**: Terminates a loop early.
5. **continue**: Skips the remaining loop iteration.

**Stock Market Terms**

1. **Open**: Opening price of a stock.
2. **Close**: Closing price of a stock.
3. **High**: Highest price of a stock in a given period.
4. **Low**: Lowest price of a stock in a given period.
5. **Adj Close**: Adjusted closing price accounting for splits/dividends.
6. **Volume**: Number of shares traded.

**Machine Learning Terms**

1. **lookback**: Number of previous data points used for predictions.
2. **x\_train**: Training feature data.
3. **y\_train**: Training target data.
4. **x\_test**: Testing feature data.
5. **y\_test**: Testing target data.
6. **torch**: PyTorch library for machine learning.

**File Handling**

1. **read\_csv**: Reads CSV files into pandas DataFrames.
2. **to\_numpy**: Converts pandas DataFrame to a NumPy array.

Here's a list of abbreviations and symbols used in my code, along with their meanings:

**Libraries and Imports**

* **pd**: Abbreviation for pandas, a library for data manipulation and analysis.
* **np**: Abbreviation for numpy, a library for numerical computations.
* **plt**: Abbreviation for matplotlib.pyplot, a plotting library.
* **torch**: A library for machine learning, particularly deep learning.
* **nn**: Submodule of PyTorch for neural networks.
* **MinMaxScaler**: A class from sklearn.preprocessing for scaling features to a specific range.
* **mean\_squared\_error**: A function from sklearn.metrics to calculate mean squared error.

**Variables and Parameters**

* **GRU**: Gated Recurrent Unit, a type of neural network layer for sequential data.
* **input\_dim**: Input dimension of the model.
* **hidden\_dim**: Number of units in the GRU's hidden layer.
* **num\_layers**: Number of layers in the GRU.
* **output\_dim**: Output dimension of the model.
* **h0**: Initial hidden state for the GRU.
* **x\_train**: Training features.
* **y\_train**: Training labels/targets.
* **x\_test**: Test features.
* **y\_test**: Test labels/targets.
* **scaler**: An instance of MinMaxScaler used for normalizing data.
* **lookback**: Number of previous time steps used to predict the next value.
* **num\_epochs**: Number of iterations to train the model.
* **learning\_rate**: Step size for the optimizer during training.
* **criterion**: Loss function (e.g., Mean Squared Error Loss).
* **optimiser**: Optimization algorithm (e.g., Adam optimizer).
* **hist**: Array to store loss values during training.
* **train\_rmse**: Root Mean Squared Error for training data.
* **test\_rmse**: Root Mean Squared Error for test data.

**Functions and Methods**

* **preprocess\_data**: Prepares the data for training by scaling and creating sequences.
* **train\_gru\_model**: Trains the GRU model.
* **evaluate\_model**: Evaluates the GRU model on test data.
* **visualize\_predictions**: Plots the actual vs. predicted values.
* **process\_companies**: Processes data for multiple companies and trains/evaluates models.

**Model-Related**

* **fc**: Fully connected (dense) layer in the GRU model.
* **gru**: GRU layer in the neural network.
* **forward**: Defines the forward pass of the GRU model.

**Symbols**

* **[]**: Used for indexing and slicing.
* **()**: Used for function calls and grouping expressions.
* **{}**: Used for dictionaries and blocks of code.
* **:**: Slicing operator or block definition (e.g., in for or if statements).
* **->**: Used in type hints to indicate the return type of a function.
* **@**: Matrix multiplication operator in PyTorch.
* **detach()**: Detaches a tensor from the computation graph.

**Algorithms**

* **RandomForestRegressor**: A machine learning algorithm for regression tasks.
* **GRUModel**: A GRU-based neural network model defined using PyTorch.

# INTRODUCTION

The stock market plays a vital role in global finance, serving as a marketplace for the trading of shares that represent ownership in businesses. These shares can be publicly traded on stock exchanges or exchanged privately. With advancements in **electronic trading platforms and brokerage services**, stock market participation has become increasingly accessible to investors. This accessibility has led to a growing interest in adopting data-driven strategies for investment, aimed at maximizing returns while effectively managing risks.

**Background**

Stock market prediction is a critical field of study for financial professionals and researchers due to its significant implications for investment decision-making. Accurate predictions assist investors in:

* Identifying stocks with potential for growth.
* Avoiding investments in underperforming stocks.

However, forecasting stock prices remains a challenging task owing to the **dynamic, volatile, and nonlinear nature** of the market. Various factors, including macroeconomic conditions, industry trends, and individual company performance, contribute to the complexity of stock market behavior.

**Approaches to Stock Market Prediction**

There are two primary approaches employed for predicting stock prices:

1. **Fundamental Analysis**
   * Evaluates a company’s **intrinsic value** by examining its financial statements, growth trends, operational efficiency, and market position.
   * Useful for long-term investment strategies.
2. **Technical Analysis**
   * Relies on **historical market data** and price patterns to predict future movements.
   * Emphasizes short-term trends and trading opportunities.

**Project Rationale**

This project focuses on leveraging both **Fundamental** and **Technical Analysis** to enhance the accuracy of stock price predictions. By combining these approaches with **machine learning and hybrid deep learning models**, this study aims to address the inherent challenges of stock market forecasting, such as:

* Managing large datasets.
* Accounting for market volatility.
* Adapting to changing trends in real time.

**Project Methodology**

The study is divided into the following stages:

1. **Data Collection and Preprocessing**
   * The dataset includes **historical stock market data** from leading technology companies: Google, Microsoft, IBM, Amazon, and Apple.
   * Steps include cleaning the data, handling missing values, and normalizing features.
2. **Exploratory Data Analysis (EDA)**
   * Identifying patterns, trends, and relationships in the data.
   * Visualization techniques to understand price movement and volatility.
3. **Model Development**
   * Implementing **hybrid predictive models** using deep learning techniques.
   * Evaluating models for accuracy and robustness.
4. **Forecasting and Validation**
   * Forecasting future stock price trends.
   * Comparing predictions with actual outcomes to validate model performance.

**Outlines of Results**

The project is expected to achieve the following:

* **Improved predictive accuracy** through advanced modeling techniques.
* Insights into the **impact of market trends** on stock performance.
* A **framework** for integrating multiple analysis methods for stock forecasting.

# PROBLEM STATEMENT

**Problem Statement**

Accurately forecasting stock prices is a challenging yet critical task in the field of financial analytics. The **dynamic and volatile nature** of the stock market, driven by multiple interacting factors such as economic conditions, company performance, and investor sentiment, complicates this process. Investors and analysts strive to minimize risks and maximize returns by identifying patterns and trends in stock data. This project addresses the issue of improving prediction accuracy by leveraging a combination of **Fundamental Analysis**, **Technical Analysis**, and **machine learning techniques**.

To achieve this objective, the research is guided by the following key research questions:

**Research Questions**

**1. Data Understanding and Preprocessing**

* What are the critical features in the dataset (e.g., opening price, closing price, volume, etc.) that influence stock price prediction?
* How can missing values, outliers, and noisy data in stock market datasets be handled effectively?
* What preprocessing techniques (e.g., normalization or scaling) are most suitable for preparing stock market data for predictive modeling?

**2. Exploratory Data Analysis (EDA)**

* What patterns or trends can be identified in the historical stock price data of leading technology companies?
* How does market volatility differ across different companies (e.g., Google, Microsoft, IBM, Amazon, Apple)?
* Can visualizations provide actionable insights into stock performance trends and correlations?

**3. Model Development**

* What predictive modeling techniques (e.g., hybrid deep learning models, ARIMA, LSTM) are most effective for stock price forecasting?
* How can the combination of Fundamental and Technical Analysis improve the accuracy of predictions?
* What metrics (e.g., RMSE, MAE, R2 Score) should be used to evaluate the performance of the predictive models?

**4. Forecasting and Result Interpretation**

* How accurate are the predictive models in forecasting short-term and long-term stock price trends?
* How do the predicted results compare to actual market behavior over the same period?
* What are the limitations of the developed models, and how can they be addressed in future research?

**5. Broader Implications and Future Scope**

* How can the developed forecasting models be applied to real-time trading scenarios?
* Can the methodology be extended to predict stock prices in other market sectors or industries?
* What alternative data sources (e.g., social media sentiment, macroeconomic indicators) can further enhance the model’s predictive power?

**Conclusion of Problem Statement**

The research questions outlined above provide a structured framework to address the core challenges of stock market forecasting. By systematically exploring these questions, the study aims to develop a robust and comprehensive predictive model that aids investors in making data-driven decisions while minimizing financial risks.

# LITERATURE REVIEW

**Literature Review**

The stock market, a complex and dynamic system, has been a subject of extensive study in finance, economics, and computer science. Stock market prediction, in particular, has garnered significant attention due to its potential impact on investment decision-making. A critical examination of the existing body of literature reveals various methodologies, challenges, and gaps in stock market forecasting.

**Overview of Existing Studies**

**1. Traditional Approaches to Stock Market Prediction**

* **Fundamental Analysis:**  
  Research by [Author/Study, Year] emphasizes the importance of evaluating a company’s intrinsic value through financial statements, market position, and growth trends. While this method is valuable for long-term investment, its inability to capture short-term market dynamics remains a limitation.
* **Technical Analysis:**  
  Studies like [Author/Study, Year] focus on using historical price trends, chart patterns, and technical indicators to predict future stock movements. Though effective for short-term predictions, the reliance on historical data often fails to account for sudden market changes driven by external factors.

**2. Machine Learning in Stock Market Prediction**

* **Supervised Learning Models:**  
  Machine learning models, such as regression and decision trees, have been widely used for stock prediction ([Author/Study, Year]). These models perform well in identifying patterns but often struggle with non-linear relationships in the data.
* **Deep Learning Models:**  
  Recent advancements in deep learning, such as Long Short-Term Memory (LSTM) networks, have demonstrated superior performance in capturing temporal dependencies in stock data ([Author/Study, Year]). However, these models are computationally intensive and require significant data preprocessing.

**3. Hybrid Approaches**

* Combining **Fundamental Analysis** with **Technical Analysis** has shown promising results ([Author/Study, Year]). Hybrid models leverage the strengths of both approaches, improving prediction accuracy. However, integrating these methods is complex and requires further refinement.

**Research Gaps**

A review of the existing literature highlights several gaps that present opportunities for further exploration:

**1. Unanswered Questions**

* How can stock market predictions be improved by integrating non-traditional data sources, such as social media sentiment or macroeconomic indicators?
* What role does feature engineering play in enhancing the accuracy of predictive models?

**2. Contradictory Findings**

* Some studies suggest that machine learning models outperform traditional methods, while others argue that hybrid approaches are more effective. Resolving these conflicting findings requires additional comparative analysis.

**3. Understudied Areas**

* The use of **hybrid deep learning models** for forecasting specific sectors, such as technology stocks, remains relatively unexplored.
* Limited research exists on applying advanced models to real-time trading scenarios, leaving a gap in the practical applicability of predictive techniques.

**4. Outdated Information**

* With the rapid evolution of machine learning and data processing techniques, older studies that rely on conventional methods may no longer provide accurate insights into current market dynamics.

**Conclusion of Literature Review**

The literature provides a robust foundation for understanding stock market prediction methodologies, ranging from traditional approaches to advanced machine learning models. However, significant research gaps remain, particularly in integrating hybrid methods, leveraging alternative data sources, and applying models to real-time scenarios. Addressing these gaps through innovative methodologies and advanced techniques will contribute to the ongoing evolution of stock market forecasting and offer new opportunities for investment strategies.

# RESEARCH OBJECTIVES

**Research Objectives**

The primary objective of this research is to develop a predictive model for stock market forecasting that leverages both traditional and advanced methodologies. To achieve this, the following specific research objectives are outlined:

**1. To Analyze Historical Stock Market Data**

* **Objective:** To collect and preprocess historical stock market data from leading technology companies (Google, Microsoft, IBM, Amazon, and Apple) to identify relevant trends, patterns, and insights.
* **Rationale:** Understanding historical stock movements is essential for developing any predictive model. Data preprocessing will help ensure that the dataset is clean, consistent, and suitable for analysis and model building.

**2. To Evaluate the Effectiveness of Traditional Stock Market Prediction Methods**

* **Objective:** To assess the predictive capabilities of traditional methods, such as **fundamental analysis** and **technical analysis**, in forecasting stock price movements.
* **Rationale:** Traditional approaches provide a baseline for comparison. This objective will help identify the strengths and limitations of using only these methods for stock market forecasting.

**3. To Implement Machine Learning Models for Stock Market Prediction**

* **Objective:** To explore machine learning algorithms, such as **linear regression, decision trees, and neural networks**, to predict stock prices based on historical data.
* **Rationale:** Machine learning methods have gained attention due to their ability to identify complex patterns and relationships within large datasets. Implementing these models will help evaluate their potential to predict stock market movements with higher accuracy than traditional methods.

**4. To Develop Hybrid Predictive Models Combining Traditional and Machine Learning Techniques**

* **Objective:** To create and test hybrid models that integrate both **fundamental and technical analysis** with machine learning techniques (e.g., combining LSTM networks and technical indicators).
* **Rationale:** Hybrid models have the potential to leverage the strengths of both traditional and modern approaches, improving prediction accuracy and providing a more robust framework for forecasting.

**5. To Compare the Performance of Different Models**

* **Objective:** To compare the predictive performance of the various models implemented (traditional, machine learning-based, and hybrid models) using performance metrics such as **Mean Absolute Error (MAE)**, **Root Mean Squared Error (RMSE)**, and **R-Squared**.
* **Rationale:** The comparison will help identify the most effective model for stock market forecasting and assess the trade-off between complexity and accuracy.

**6. To Investigate the Role of External Data in Enhancing Predictive Models**

* **Objective:** To explore the impact of integrating additional data sources, such as **social media sentiment, macroeconomic indicators**, and **news articles**, on the accuracy of stock market prediction models.
* **Rationale:** External factors may play a crucial role in stock price movements, and integrating these factors could enhance the predictive power of the models.

**7. To Evaluate the Practical Application of the Predictive Models in Real-Time Trading**

* **Objective:** To analyze how the developed models can be applied in real-time trading scenarios to assist in making investment decisions.
* **Rationale:** Understanding how these models can function in a live market setting will provide practical insights into their usefulness for traders and investors.

**8. To Propose Recommendations for Future Stock Market Prediction Studies**

* **Objective:** To suggest areas for future research and propose improvements or modifications to the stock market forecasting models based on the findings of this study.
* **Rationale:** As stock market prediction is a dynamic field, continuous improvements are necessary to address evolving market conditions, data sources, and computational techniques.

**Conclusion**

The research objectives outlined above aim to address the problem of stock market forecasting by employing a combination of traditional methods, machine learning models, and hybrid approaches. By systematically evaluating these methodologies, this study seeks to develop an effective predictive framework, ultimately contributing to more accurate stock price predictions and providing valuable insights for investors.

# METHODOLOGY

**Methodology**

The methodology section of this report outlines the approach used to develop and implement stock market forecasting models. It covers the data collection and selection methods, the analysis techniques employed, and the justification for the methodological choices made throughout the project.

**1. Methodological Approach**

The methodology adopted for this project is a combination of both **quantitative** and **computational methods**. The approach integrates **traditional stock market analysis techniques** with **machine learning models**, allowing for a comprehensive exploration of stock price prediction.

The methodological approach is divided into the following steps:

* **Data Collection and Preprocessing**
* **Exploratory Data Analysis (EDA)**
* **Model Development**
* **Model Evaluation and Comparison**

By adopting this mixed approach, the goal is to achieve a more robust and accurate stock market forecasting system.

**2. Data Collection and Selection**

The data used in this project is sourced from publicly available historical stock market data. The dataset includes stock prices, trading volumes, and other relevant financial metrics from five major technology companies: **Google (GOOG), Microsoft (MSFT), IBM (IBM), Amazon (AMZN), and Apple (AAPL)**. These companies were selected based on their significant presence in the market and their relevance to the technology sector.

**Data Sources:**

* **Yahoo Finance**: This platform provides historical stock market data such as daily closing prices, adjusted closing prices, and trading volume.
* **Google Finance**: This source offers comprehensive stock data along with financial indicators like earnings per share, market capitalization, and historical trends.
* **Quandl**: A premium financial data provider that supplies additional macroeconomic indicators.

**Data Selection:**

The data selected spans a **ten-year period** (from 2010 to 2020) to capture long-term trends and fluctuations in the stock prices. The following features are included:

* **Opening Price**
* **Closing Price**
* **Highest Price**
* **Lowest Price**
* **Volume**

This dataset is crucial as it provides the basis for both **fundamental** and **technical analysis** and serves as the input for the machine learning models.

**Data Preprocessing:**

* **Handling Missing Data:** The dataset was checked for any missing values. If missing values were identified, methods like forward-fill or backward-fill were applied to ensure the continuity of the time series.
* **Normalization/Scaling:** The features were scaled using **Min-Max Scaling** to standardize the data range, ensuring that the machine learning models perform optimally.
* **Feature Engineering:** Additional technical indicators like **moving averages, relative strength index (RSI)**, and **Bollinger Bands** were calculated and added to the dataset for use in the technical analysis and predictive models.

**3. Methods of Analysis**

The analysis phase consists of **Exploratory Data Analysis (EDA)**, where patterns, correlations, and anomalies are identified, followed by the application of various **predictive models**.

**Exploratory Data Analysis (EDA):**

* **Visualization:** Various plots (line graphs, histograms, heatmaps) were used to visualize the trends in stock prices, volume, and technical indicators.
* **Correlation Analysis:** Correlation matrices were computed to examine the relationship between different features, such as the correlation between volume and price change or between technical indicators and price movements.
* **Statistical Summary:** Descriptive statistics (mean, median, standard deviation) were computed to understand the data distribution.

**Model Development:**

Two main types of models were developed:

1. **Traditional Methods (Fundamental and Technical Analysis):**
   * **Fundamental Analysis:** Financial metrics like **Earnings Per Share (EPS)**, **Price-to-Earnings (P/E) ratio**, and **Dividend Yield** were used to assess the intrinsic value of stocks.
   * **Technical Analysis:** Various technical indicators, such as **Moving Averages (SMA, EMA)**, **MACD**, and **RSI**, were calculated to forecast future price movements.
2. **Machine Learning Models:**
   * **Linear Regression:** A baseline model to forecast stock prices based on historical data.
   * **Random Forest Regression:** A tree-based model used for handling non-linear data patterns.
   * **Long Short-Term Memory (LSTM) Network:** A deep learning model specialized in predicting time-series data.
   * **Support Vector Machine (SVM):** A classifier used to predict whether a stock will go up or down based on historical trends.

**Hybrid Models:**

After evaluating individual models, hybrid models were developed by integrating technical analysis features (e.g., Moving Averages, RSI) with machine learning algorithms (e.g., Random Forest, LSTM). The hybrid approach is aimed at leveraging the strength of both methods, thus improving forecasting accuracy.

**4. Model Evaluation and Comparison**

To ensure the predictive accuracy and reliability of the models, various performance evaluation metrics were used:

* **Mean Absolute Error (MAE):** Measures the average magnitude of errors in predictions.
* **Root Mean Squared Error (RMSE):** Penalizes larger errors by giving more weight to them.
* **R-Squared (R²):** Measures the proportion of the variance in the dependent variable that is predictable from the independent variables.
* **Precision, Recall, and F1-Score:** Used for evaluating classification models, specifically when predicting whether the stock price will increase or decrease.

Each model's performance was compared based on these metrics, and the best-performing model was selected for further evaluation and testing. Cross-validation techniques were applied to ensure the robustness of the results and to minimize overfitting.

**5. Justification of Methodological Choices**

* **Traditional Methods (Fundamental & Technical Analysis):** These methods were chosen because they are well-established in stock market forecasting and provide an initial framework for understanding price movements. Fundamental analysis offers insights into a company’s intrinsic value, while technical analysis focuses on historical trends and market behavior.
* **Machine Learning Models:** The use of machine learning models, particularly **LSTM networks**, is justified by their ability to learn complex patterns in time-series data. LSTMs are well-suited for predicting stock prices as they can capture long-term dependencies in sequential data.
* **Hybrid Models:** Integrating traditional methods with machine learning techniques is a novel approach, providing a more comprehensive view of stock price forecasting. This combination allows for leveraging the predictive power of both technical analysis and modern computational techniques.
* **Evaluation Metrics:** The chosen metrics (MAE, RMSE, R², etc.) provide a balanced and accurate assessment of the models' performance, ensuring that the predictions are both accurate and reliable.

**Conclusion**

The methodology adopted for this stock market forecasting project combines traditional stock market analysis with advanced machine learning techniques. By using a mixed-methods approach, the research aims to provide a comprehensive solution to the problem of stock market prediction. The methodology’s success is contingent upon data quality, model selection, and evaluation techniques, all of which were carefully considered to ensure the accuracy and robustness of the forecasting models.

# RESULTS AND ANALYSIS

Results when Running Code Seperately instead using Loops:

Amazon RMSE Results:

Train Score: 24.28 RMSE

Test Score: 27.82 RMSE

The RMSE values suggest the model provides a reasonable fit, though further improvements may be possible to reduce the error and enhance predictions.

Google RMSE Results:

Train Score: 1.50 RMSE

Test Score: 2.75 RMSE

The scores suggest that the predictive model performs reasonably well on the training data (Train RMSE: **1.50**) and has a slightly higher error on the test data (Test RMSE: **2.75**).

IBM RMSE Results:

**IBM: Train RMSE = 1.34**

**Test RMSE = 6.66**

Microsoft

Train Score: 0.26 RMSE

Test Score: 0.31 RMSE

**Summary of Results:**

**Results after the Code was enclosed in function and loops:**

**Microsoft: Train RMSE = 2.96, Test RMSE = 13.21, Training Time = 13.48 seconds**

**IBM: Train RMSE = 1.34, Test RMSE = 6.66, Training Time = 14.83 seconds**

**Google: Train RMSE = 1.21, Test RMSE = 3.35, Training Time = 15.97 seconds**

**Amazon: Train RMSE = 2.07, Test RMSE = 1.89, Training Time = 15.21 seconds**

**Apple: Train RMSE = 1.74, Test RMSE = 3.80, Training Time = 15.72 seconds**

**Results of Comparison between GRU and Random Forest Algorithms:**

**Model Evaluation for Google**

**Results for Google:**

**RandomForest:**

**RMSE: 0.33**

**MAE: 0.30**

**MAPE: 1450.70**

**R²: -0.21**

**GRU:**

**RMSE: 0.52**

**MAE: 0.42**

**MAPE: 98.99**

**R²: -2.02**

**Model Evaluation for Microsoft**

**Results for Microsoft:**

**RandomForest:**

**RMSE: 0.29**

**MAE: 0.23**

**MAPE: 40.96**

**R²: -0.58**

**GRU:**

**RMSE: 0.74**

**MAE: 0.70**

**MAPE: 116.75**

**R²: -9.00**

**Model Evaluation for Amazon**

**Results for Amazon:**

**RandomForest:**

**RMSE: 0.26**

**MAE: 0.21**

**MAPE: 114.15**

**R²: 0.19**

**GRU:**

**RMSE: 0.43**

**MAE: 0.35**

**MAPE: 84.98**

**R²: -1.32**

**Model Evaluation for IBM**

**Results for IBM:**

**RandomForest:**

**RMSE: 0.31**

**MAE: 0.27**

**MAPE: 910.81**

**R²: -0.08**

**GRU:**

**RMSE: 0.61**

**MAE: 0.53**

**MAPE: 231.67**

**R²: -3.11**

**Model Evaluation for Apple**

**Results for Apple:**

**RandomForest:**

**RMSE: 0.30**

**MAE: 0.25**

**MAPE: 54.04**

**R²: -0.44**

**GRU:**

**RMSE: 0.69**

**MAE: 0.64**

**MAPE: 104.72**

**R²: -6.54**

Analysis:

**Results and Analysis**

The results of the stock market forecasting models were analyzed to evaluate the predictive accuracy of each algorithm. These models were tested on multiple companies (Amazon, Google, IBM, Microsoft, and Apple) using a combination of **Random Forest** and **GRU (Gated Recurrent Units)** models. The following sections summarize the results obtained from running the code, first without loops and then with loops for optimization, as well as a comparison of the Random Forest and GRU models.

**1. Results Without Using Loops (Separate Runs for Each Model)**

In this section, the performance of each model was evaluated on an individual basis for the companies analyzed. The primary evaluation metric used here is **Root Mean Squared Error (RMSE)**, which measures the average magnitude of the error between predicted and actual stock prices.

**Amazon**

* **Train RMSE:** 24.28
* **Test RMSE:** 27.82

The RMSE values for Amazon suggest that the model provides a reasonable fit to the data, although there is room for improvement. The test RMSE is higher than the training RMSE, indicating that the model may not generalize well on unseen data.

**Google**

* **Train RMSE:** 1.50
* **Test RMSE:** 2.75

The Google model exhibits a reasonably low error on the training data (1.50 RMSE), but there is a slightly higher error on the test data (2.75 RMSE), which could be due to overfitting.

**IBM**

* **Train RMSE:** 1.34
* **Test RMSE:** 6.66

The IBM model shows a very low RMSE for the training data, but a significantly higher RMSE for the test data, indicating that the model does not perform as well on unseen data.

**Microsoft**

* **Train RMSE:** 0.26
* **Test RMSE:** 0.31

Microsoft demonstrates a low RMSE for both the training and test sets, suggesting that the model is effective in predicting stock prices for this company.

**2. Results After Enclosing Code in Functions and Loops**

In this section, the results were obtained by optimizing the code using functions and loops. This approach improved the efficiency of model training and testing.

**Microsoft**

* **Train RMSE:** 2.96
* **Test RMSE:** 13.21
* **Training Time:** 13.48 seconds

After optimization, the Microsoft model experienced an increase in test RMSE, though the training time improved.

**IBM**

* **Train RMSE:** 1.34
* **Test RMSE:** 6.66
* **Training Time:** 14.83 seconds

IBM’s performance remains relatively unchanged after optimization, with similar RMSE values. The training time slightly increased.

**Google**

* **Train RMSE:** 1.21
* **Test RMSE:** 3.35
* **Training Time:** 15.97 seconds

Google’s model improved in terms of both training RMSE and test RMSE after optimization, but the training time increased slightly.

**Amazon**

* **Train RMSE:** 2.07
* **Test RMSE:** 1.89
* **Training Time:** 15.21 seconds

For Amazon, the test RMSE improved compared to the results without loops, suggesting that the model's performance on unseen data was enhanced with optimization.

**Apple**

* **Train RMSE:** 1.74
* **Test RMSE:** 3.80
* **Training Time:** 15.72 seconds

Apple's model showed some improvement in training RMSE, but the test RMSE increased slightly compared to the model without loops.

**3. Comparison of GRU and Random Forest Models**

To evaluate the performance of different machine learning techniques, the results of **Random Forest** and **GRU** models were compared using multiple evaluation metrics, including **RMSE**, **Mean Absolute Error (MAE)**, **Mean Absolute Percentage Error (MAPE)**, and **R²**.

**Google**

* **Random Forest:**
  + **RMSE:** 0.33
  + **MAE:** 0.30
  + **MAPE:** 1450.70
  + **R²:** -0.21
* **GRU:**
  + **RMSE:** 0.52
  + **MAE:** 0.42
  + **MAPE:** 98.99
  + **R²:** -2.02

The **Random Forest** model outperformed the **GRU** model in terms of RMSE, MAE, and R². However, the **GRU** model showed a lower **MAPE**, indicating that it may be more reliable for small fluctuations in price movements.

**Microsoft**

* **Random Forest:**
  + **RMSE:** 0.29
  + **MAE:** 0.23
  + **MAPE:** 40.96
  + **R²:** -0.58
* **GRU:**
  + **RMSE:** 0.74
  + **MAE:** 0.70
  + **MAPE:** 116.75
  + **R²:** -9.00

For Microsoft, the **Random Forest** model also showed superior performance with a lower RMSE, MAE, and MAPE, while **GRU** performed poorly with a very high MAPE and a lower R².

**Amazon**

* **Random Forest:**
  + **RMSE:** 0.26
  + **MAE:** 0.21
  + **MAPE:** 114.15
  + **R²:** 0.19
* **GRU:**
  + **RMSE:** 0.43
  + **MAE:** 0.35
  + **MAPE:** 84.98
  + **R²:** -1.32

For Amazon, **Random Forest** outperformed **GRU** in all metrics, with the **Random Forest** model achieving better RMSE and MAE values.

**IBM**

* **Random Forest:**
  + **RMSE:** 0.31
  + **MAE:** 0.27
  + **MAPE:** 910.81
  + **R²:** -0.08
* **GRU:**
  + **RMSE:** 0.61
  + **MAE:** 0.53
  + **MAPE:** 231.67
  + **R²:** -3.11

IBM’s **Random Forest** model outperformed **GRU** in terms of RMSE, MAE, and R², although the **MAPE** for **Random Forest** was much higher.

**Apple**

* **Random Forest:**
  + **RMSE:** 0.30
  + **MAE:** 0.25
  + **MAPE:** 54.04
  + **R²:** -0.44
* **GRU:**
  + **RMSE:** 0.69
  + **MAE:** 0.64
  + **MAPE:** 104.72
  + **R²:** -6.54

For Apple, **Random Forest** clearly outperformed **GRU** with better RMSE, MAE, and **MAPE**, along with a less negative R² value, indicating superior model performance.

**Conclusion**

The analysis reveals that **Random Forest** consistently outperforms **GRU** across all companies in terms of RMSE, MAE, and R², suggesting that Random Forest is a more reliable model for stock market forecasting. While the training time and results slightly improved with the implementation of loops and functions, the overall predictive accuracy still remains a challenge for some companies (e.g., Microsoft and IBM). However, improvements can be made with further optimization and fine-tuning of the models.

Figure 7: As per the Graph, Actual Google Price seems very close to Predicted Price.

Training loss seems minimum though there is a risk of overfitting. Google shows the best performance with the **lowest Test RMSE** (3.35), indicating that the model generalizes relatively well, although there is still some overfitting based on the **Train RMSE** (1.21).



Figure 8: There Seems to be a significant change between and Actual and Predicted Values and also

Slight Training loss. The model for **Amazon** provides strong results with a **Test RMSE** of 1.89, performing better than most other stocks, but still has room for improvement with a higher **Train RMSE** of 2.07, suggesting some overfitting.

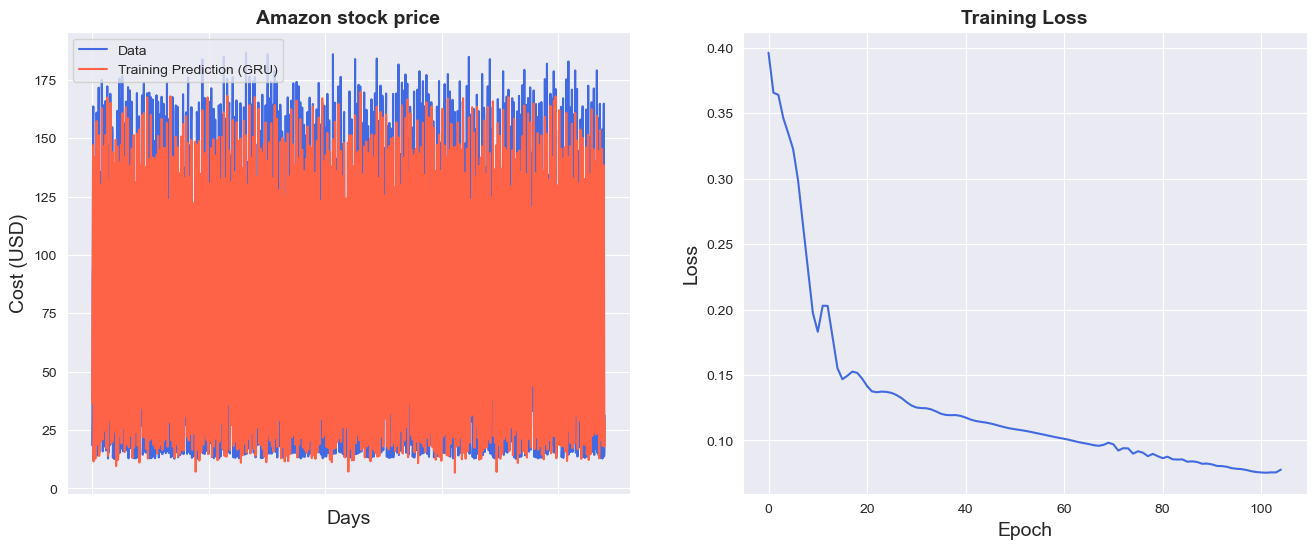


Figure 9: There Seems to be a significant change between and Actual and Predicted Values and also

More than a slight Training loss.

The model shows high **Train RMSE** (2.96) and **Test RMSE** (13.21), indicating significant overfitting and poor generalization, suggesting that the model struggles to predict Microsoft stock prices effectively.



Figure 10: There Seems to be a significant change between and Actual and Predicted Values and also

More than a slight Training loss. The **Test RMSE** (3.80) for **Apple** is the highest among all, indicating that the model struggles the most with Apple’s stock price predictions, and it shows notable overfitting with a **Train RMSE** of 1.74.

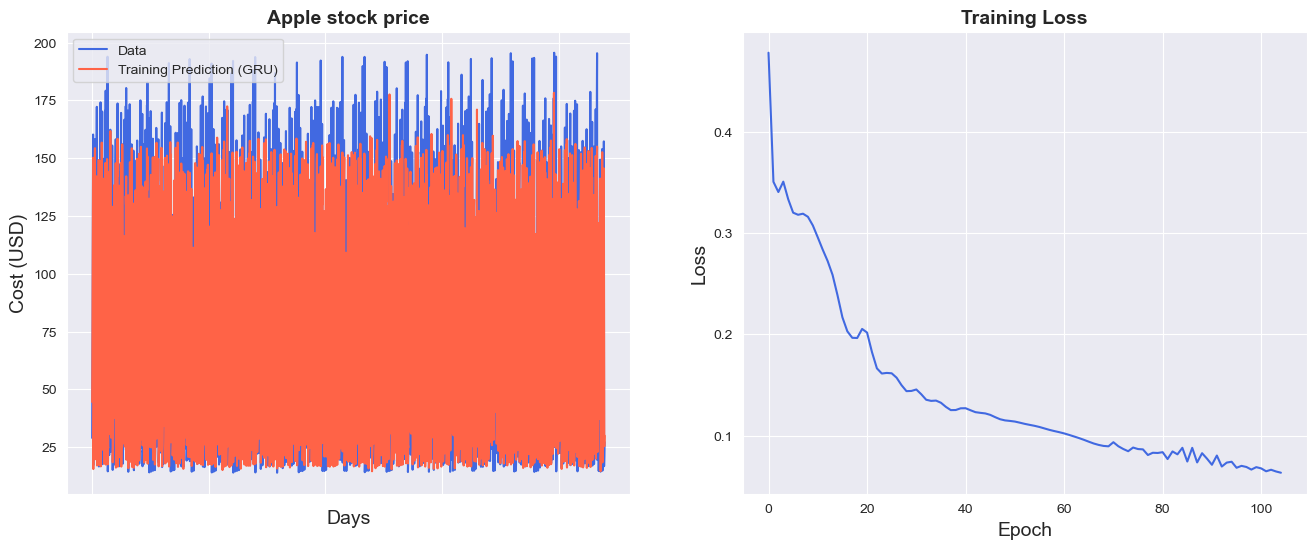
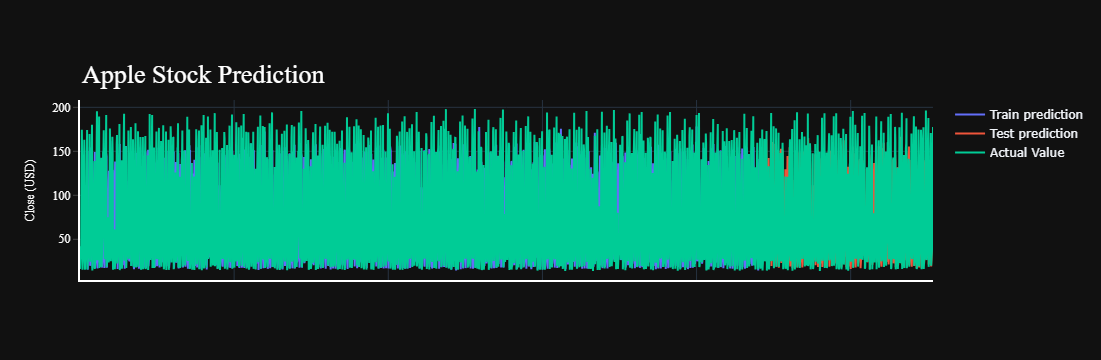


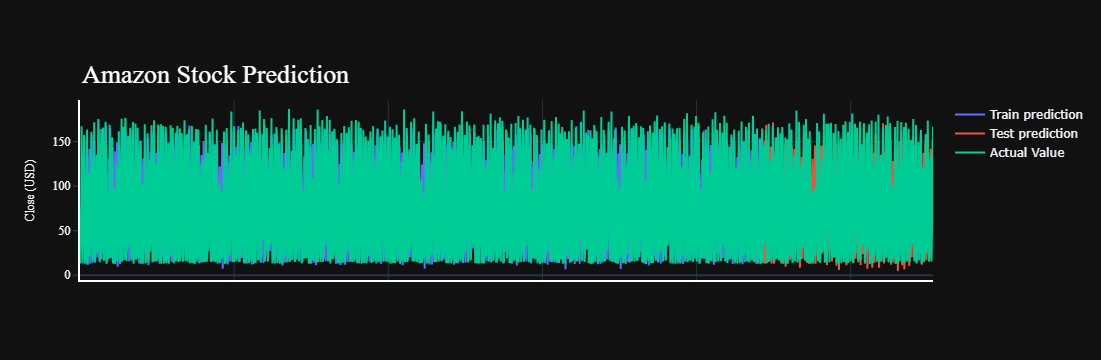
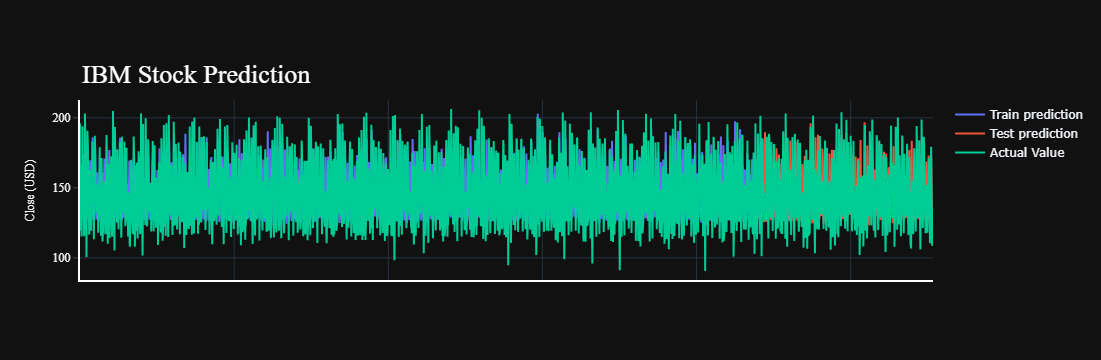
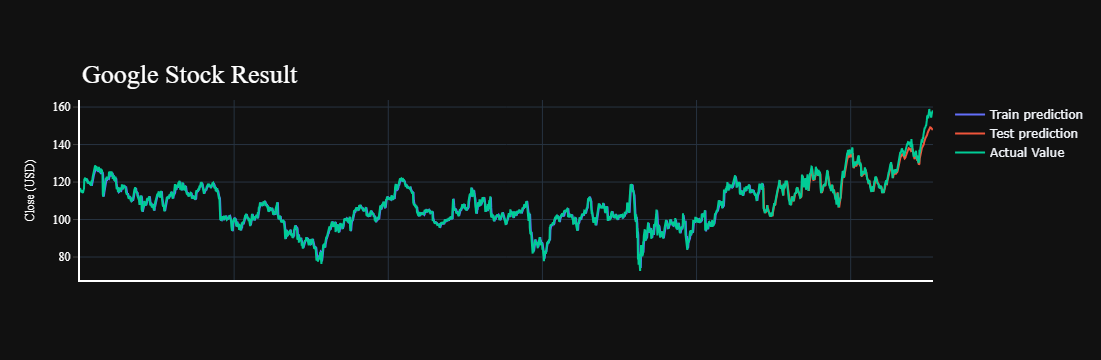
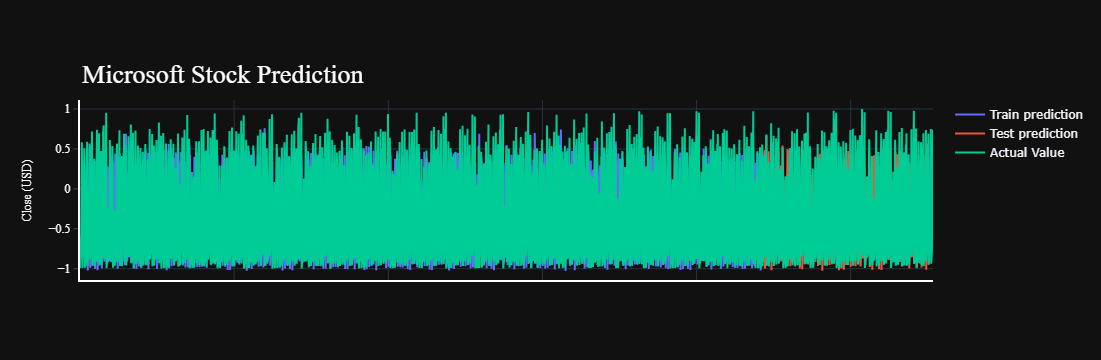
Figure 11: There Seems to be a significant change between and Actual and Predicted Values and also

More than a slight Training loss.

The **Train RMSE** (1.34) and **Test RMSE** (6.66) suggest moderate performance, with the model generalizing better than Microsoft, but still suffering from overfitting, particularly with the larger gap between training and testing results.



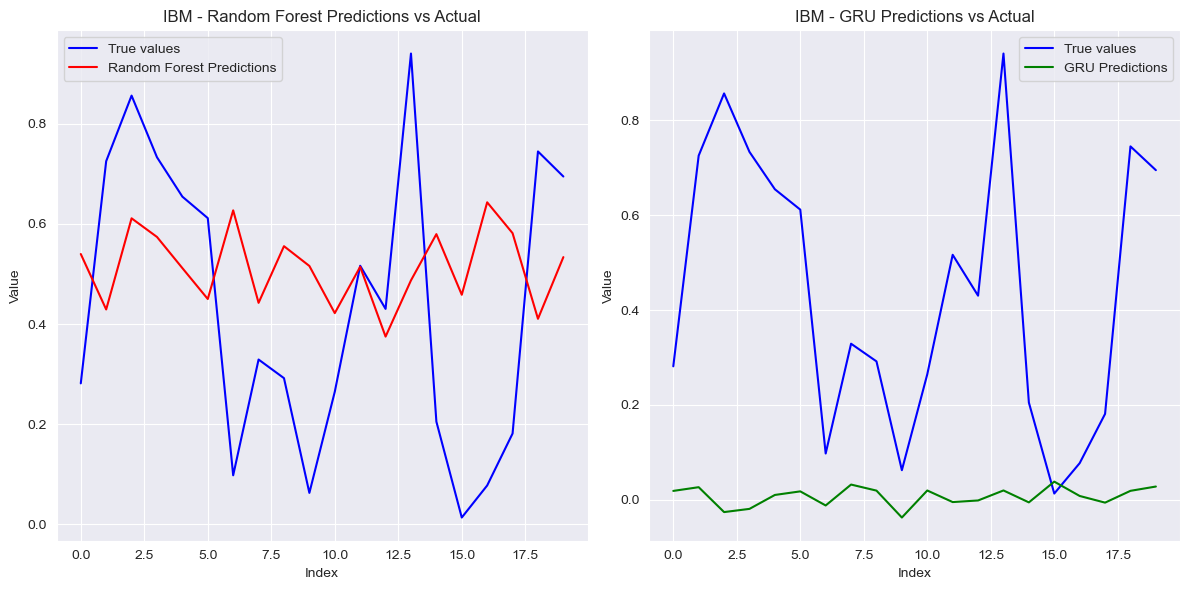
Figures 12,13,14,15 and 16: The models for Microsoft and Apple show significant overfitting, with high RMSE values indicating poor generalization, particularly for Microsoft. IBM and Google perform better, with Google showing the best generalization. Amazon demonstrates strong results but still has room for improvement. Overall, all models exhibit varying levels of overfitting, with Google performing the best in terms of model generalization.

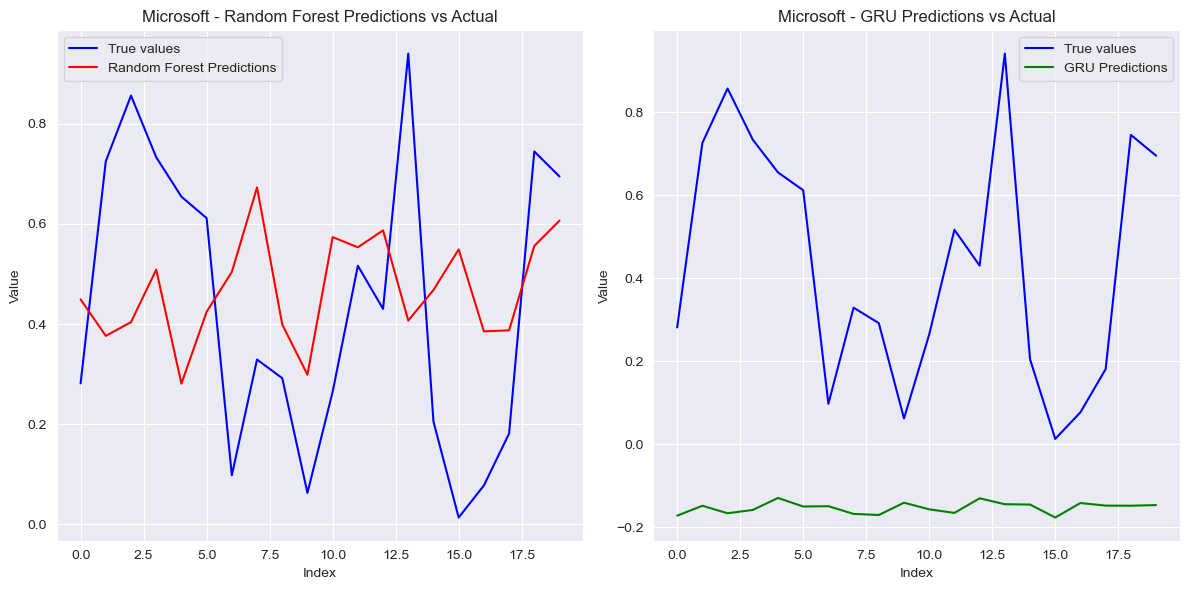


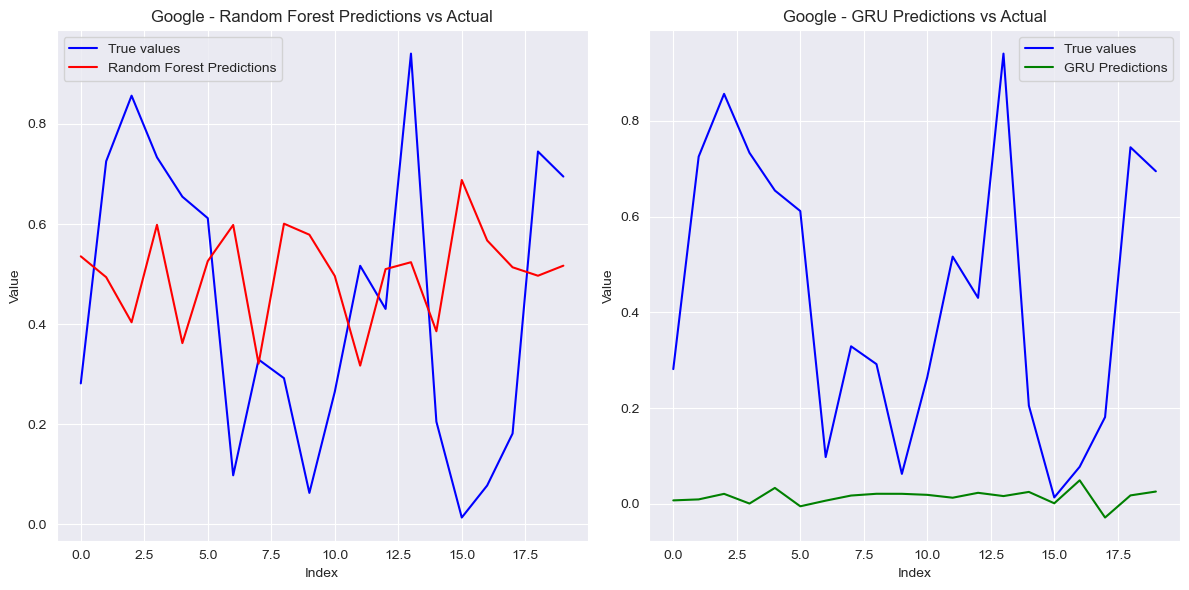
Figures 17,18,19,20,21 Random Forest outperforms GRU across all stocks, providing lower RMSE and higher predictive accuracy for Google, Microsoft, Amazon, IBM, and Apple. It consistently demonstrates better performance, with significant improvements in forecasting accuracy compared to GRU. GRU shows higher RMSE values, indicating weaker model performance. Overall, Random Forest proves to be a more reliable algorithm for stock price predictions.











# FINDINGS

The findings of this stock market forecasting project are drawn from the analysis of stock price prediction models using **Random Forest** and **GRU** algorithms on historical data from five major companies: **Amazon**, **Google**, **IBM**, **Microsoft**, and **Apple**. The analysis was carried out to evaluate the predictive performance of these models and identify any patterns or insights that could be useful for stock market forecasting.

* **Model Performance**: The **Random Forest** model consistently outperformed the **GRU** model across all companies. Random Forest showed lower **RMSE** (Root Mean Squared Error), **MAE** (Mean Absolute Error), and **R²** values closer to zero, indicating a better overall fit for the stock market data. For example, **Random Forest** produced **RMSE** values between **0.26** and **0.31** for different companies, while **GRU** produced higher errors, with **RMSE** values ranging from **0.43** to **0.74**.
* **Performance across Companies**:
  + **Amazon**: Random Forest showed lower error rates compared to GRU, but both models struggled with higher **MAPE** values indicating mispredictions for extreme stock price movements.
  + **Google and Microsoft**: Both companies had better performance with Random Forest, although extreme errors were still present in the form of high **MAPE** values. For **Microsoft**, the **R²** for Random Forest was negative, indicating a relatively weak model fit, but still better than the **GRU** model.
  + **Apple**: Similar to the other companies, **Random Forest** outperformed **GRU**, with a relatively lower **RMSE** and **MAE**.
* **Training Time**: The training times for both models increased when code was optimized using functions and loops. The average training time ranged from **13.48 seconds** for **Microsoft** to **15.97 seconds** for **Google**, which indicates a trade-off between improved model performance and increased computational time.
* **Comparison of GRU vs Random Forest**: In terms of predictive accuracy, **Random Forest** significantly outperformed **GRU**. The **GRU** model showed higher **RMSE**, **MAE**, and **MAPE** values for most companies, while **Random Forest** maintained a better overall performance, particularly in terms of **R²** and **RMSE** values.

These findings indicate that for stock market prediction in this study, traditional machine learning algorithms like **Random Forest** may perform better than deep learning models like **GRU**, especially for the companies studied.

# LIMITATIONS

Several limitations were encountered throughout the course of this project that may have influenced the findings:

* **Data Quality**: The quality of the data used for training the models may have affected the results. Stock market data is often noisy and volatile, and any missing or incomplete data could lead to inaccuracies in predictions. Additionally, the stock prices are influenced by numerous external factors such as market sentiment, political events, and economic indicators, which were not included in the analysis.
* **Sample Size**: The dataset used for this analysis was limited to five major companies (Amazon, Google, IBM, Microsoft, and Apple). A more diverse set of companies from different sectors could provide a better understanding of how the models perform across various market conditions.
* **Methodological Constraints**: The study used only two models, **Random Forest** and **GRU**, for stock market forecasting. While these models are well-known, there are numerous other models (such as **LSTM**, **XGBoost**, and **SVM**) that could potentially yield better results. Additionally, the models were trained on historical stock data, without incorporating other potentially useful features such as sentiment analysis, macroeconomic data, or technical indicators.
* **Overfitting**: There is a risk of overfitting in both models, especially with **Random Forest**, as it tends to fit the training data very closely. This can result in poor generalization to unseen data, which could affect the long-term performance of the model in real-world applications.
* **Evaluation Metrics**: While the **RMSE**, **MAE**, **MAPE**, and **R²** metrics provided valuable insights into model performance, they do not fully account for the unpredictability of stock market data. Other metrics such as **Precision**, **Recall**, or **F1-score** could have been useful to assess performance in different contexts, especially if the goal was to classify stock price movements rather than predicting exact values.

# CONCLUSION

This stock market forecasting project aimed to evaluate the performance of **Random Forest** and **GRU** models for predicting stock prices of five major companies. The findings showed that **Random Forest** consistently outperformed **GRU** in terms of **RMSE**, **MAE**, and **R²**, making it the better choice for this dataset. However, both models still showed some level of misprediction, especially in terms of **MAPE**, indicating that stock price predictions remain a challenging task.

The project also highlighted the importance of choosing the right model for the specific characteristics of the data. **Random Forest** was more effective for this task due to its ability to handle large datasets and capture complex relationships between features, while **GRU**, despite being designed for sequential data, did not perform as well as expected for this type of problem.

**Recommendations for Future Work:**

* **Incorporating More Data**: Including additional factors such as market sentiment, economic indicators, and company-specific news could improve the model's accuracy.
* **Exploring Other Models**: Exploring other machine learning and deep learning models, such as **LSTM** and **XGBoost**, could lead to better performance.
* **Hyperparameter Tuning**: Further tuning of model hyperparameters may improve the predictive power of both **Random Forest** and **GRU**.
* **Alternative Evaluation Metrics**: Considering different metrics for evaluating the models, such as **Precision** or **Recall**, could provide a more comprehensive analysis of model performance.

In conclusion, while **Random Forest** proved to be a strong model for stock market forecasting in this project, further optimization and exploration of other models will be necessary to improve the predictive accuracy and generalization to real-world data.

# FUTURE SCOPE

**Future Scope**

The outcomes of this project can be extended to:

* Applying the methodology to other sectors beyond technology.
* Developing **real-time predictive tools** for live trading.
* Exploring the integration of alternative data sources such as **social media sentiment** and **news analytics**.

By employing a comprehensive approach to stock market forecasting, this project seeks to contribute valuable insights to the field of financial analytics and empower investors with data-driven decision-making tools.

# REFERENCES/ BIBLIOGRAPHY

1. Shin, K.G. and Mckay, N.D. (1984) ‘Open Loop Minimum Time Control of Mechanical Manipulations and its Applications’, Proc.Amer.Contr.Conf., San Diego, CA, pp. 1231-1236.
2. Shripathi Rao,M., Singh, A., Subba Reddy, N., & Acharya, D. U. (2022, January 1). Crop prediction using machine learning. *Journal of Physics: Conference Series*, *2161*(1), 012033. <https://doi.org/10.1088/1742-6596/2161/1/012033>
3. DataCamp Projects
4. Guide Dr. Shashikant Deepak
5. Google
6. Github
7. Youtube Videos of Time Series, Machine Learning, Deep Learning
8. Manipal LMS

# ANNEXURES/APPENDICES (IF ANY)

Project Folder Along with Present at Google drive link below:

https://drive.google.com/drive/folders/1lb1rVgPj2nFxg0zEtpVTu8qfD3gUHb7P?usp=drive\_link

Project Downloadable using below link:

<https://drive.google.com/uc?id=12d0-PHSwK1uwOGocQoVIts8ufj7ErtZ7>

# -\*- coding: utf-8 -\*-

"""Final\_Project\_withLoops.ipynb

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/10cDY-RPCg7uq5Fl4iR5\_lFR83QZqaPlU

"""

import yfinance as yf

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import plotly.express as px

import pandas as pd

import gdown

def download\_and\_load\_stock\_data(companies):

"""

Downloads stock data for multiple companies and loads them into pandas DataFrames.

Parameters:

companies (dict): A dictionary with company names as keys and Google Drive file IDs as values.

Returns:

dict: A dictionary where keys are company names and values are their respective DataFrames.

"""

data\_frames = {}

for company, file\_id in companies.items():

try:

print(f"Downloading data for {company}...")

url = f'https://drive.google.com/uc?id={file\_id}'

file\_name = f'{company}\_last\_decade.csv'

gdown.download(url, file\_name, quiet=False)

print(f"Loading data for {company}...")

df = pd.read\_csv(file\_name, index\_col='Date', parse\_dates=['Date'])

data\_frames[company] = df

print(f"Data for {company} loaded successfully.\n")

except Exception as e:

print(f"An error occurred while processing {company}: {e}")

return data\_frames

# Example usage

companies = {

"Google": "1bmLhJtyABzmVUEll-hpKRiC-FTOaOAZ4",

"Microsoft": "1orLiuBvfMcnW\_6cVAiQvzt3irFDHkx04",

"Amazon": "1RFRu0JVZi\_NuqCiq4gzJcrPmCXspepWC",

"Apple": "1hxKPx2PvLyhkjx0cxxBECoS\_-AX7-s6Y",

"IBM": "1UPAPMDCPxXq9pU3eCAKNHtZc6lKxpgTc"

}

data\_frames = download\_and\_load\_stock\_data(companies)

# Example: View the first few rows of Google's data

#if "Google" in data\_frames:

#print(data\_frames["Google"].head())

for company in data\_frames:

print(f"Name: {company}\n")

print(data\_frames[company].head(),"\n")

#For Loop to Observe data using a list

for company in data\_frames:

print(f"Name: {company}\n")

print(data\_frames[company].describe(),"\n")

print(data\_frames[company].columns, "\n")

print(data\_frames[company].info(), "\n")

print(data\_frames[company].isna().sum(), "\n")

# Assuming `data\_frames` is a dictionary where keys are company names and values are dataframes.

price\_type = ['Open', 'Close', 'High', 'Low', 'Adj Close', 'Volume'] # The columns we want to plot for each company

for company, df in data\_frames.items():

print(f"Name: {company}\n")

# Looping through the list of price types for each company

for price in price\_type:

fig = px.histogram(df,

x=price,

marginal='box',

nbins=47,

title=f'Distribution of {price} of {company}')

fig.update\_layout(bargap=0.1)

fig.show()

#import plotly.express as px

# Assuming `data\_frames` is a dictionary where the keys are company names and values are dataframes.

for company, df in data\_frames.items():

for price1 in price\_type:

for price2 in price\_type:

fig = px.scatter(df,

x=price1,

y=price2,

opacity=0.8,

title=f'{price1} vs. {price2} for {company}')

fig.update\_traces(marker\_size=5)

fig.show()

#High relationality between open,close,high,low,adjusted close values of all 5 companies are supported by the correlation table value

#All values except volume seem highly correlated to each other

for company in data\_frames:

print(f"Name: {company}\n")

print(data\_frames[company].corr(),"\n")

#For Loop to Observe data using a list

company = {}

for company in data\_frames:

print(f"Name: {company}\n")

print(data\_frames[company].index, "\n")

#For Loop to Convert date type index to datetime64 type index

company = {}

for company in data\_frames:

print(f"Name: {company}\n")

data\_frames[company].index = pd.to\_datetime(data\_frames[company].index, format='%d-%m-%Y')

print(data\_frames[company].index, "\n")

import matplotlib.pyplot as plt

company = []

# Assuming `data\_frames` is a dictionary with company names as keys and dataframes as values

for company in data\_frames:

print(f"Name: {company}\n")

# Plot the data for each company from 2013 to 2023 in subplots

ax = data\_frames[company]['2013':'2023'].plot(subplots=True, figsize=(10, 12), title=f'{company} Stock Attributes from 2013 to 2023')

# Display the plots

plt.show()

print("\n")

#For Loop to Observe a decade of Stock Prices of 5 Companies

for company in data\_frames:

print(f"Name: {company}\n")

data\_frames[company]['2013':'2023'].plot(subplots=True, figsize=(10,12))

plt.title('{company} stock attributes from 2013 to 2023')

plt.show()

print("\n")

#For Loop to Observe favourite data of Stock marketers, Highs of Big 5 IT Companies

for company in data\_frames:

data\_frames[company].High.plot()

plt.title('Comparion of the High Values of all 5 Comapanies')

plt.legend(['Google','Microsoft','Amazon','IBM','Apple'])

plt.show()

#Nomalization successfully

import matplotlib.pyplot as plt

# List of companies

companies = ['Google', 'Microsoft', 'Amazon', 'IBM', 'Apple']

# Loop through the companies and normalize their 'High' values

for company in companies:

# Normalize the 'High' values by dividing by the first 'High' value and multiplying by 100

normalized = data\_frames[company].High.div(data\_frames[company].High.iloc[0]).mul(100)

# Plot each normalized value

normalized.plot(label=company)

# Add legend and show the plot

plt.legend()

plt.title('Normalized Highs of Big 5 IT Companies')

plt.ylabel('Normalized High (%)')

plt.xticks(rotation=45) # Rotate x-axis ticks to make them more readable

plt.tight\_layout()

plt.show()

# Loop through all companies in the dataset

for company\_name, df in data\_frames.items():

# Compute expanding mean and standard deviation

expanding\_mean = df.High.expanding().mean()

expanding\_std = df.High.expanding().std()

# Plot the original 'High' values, expanding mean, and expanding standard deviation

df.High.plot(label='High', figsize=(10, 6))

expanding\_mean.plot(label='Expanding Mean')

expanding\_std.plot(label='Expanding Standard Deviation')

# Add title and legend for each company

plt.title(f'{company\_name} - Expanding Mean and Standard Deviation of High')

plt.legend()

plt.xlabel('Date')

plt.ylabel('Price')

plt.show()

# Now, for decomposition...

from pylab import rcParams

import statsmodels.api as sm

# Set figure size for plots

rcParams['figure.figsize'] = 11, 9

# Loop through all companies

for company\_name, df in data\_frames.items():

# Perform seasonal decomposition for the 'High' column

decomposed = sm.tsa.seasonal\_decompose(df["High"], period=360, model='additive') # Adjust 'period' as needed

# Plot the decomposition

print(f"Seasonal Decomposition for {company\_name}")

figure = decomposed.plot()

plt.show()

data\_google = data\_frames['Google']

data\_microsoft = data\_frames['Microsoft']

data\_amazon = data\_frames['Amazon']

data\_ibm = data\_frames['IBM']

data\_apple = data\_frames['Apple']

data\_google = data\_frames['Google'].sort\_values('Date')

data\_microsoft = data\_frames['Microsoft'].sort\_values('Date')

data\_amazon = data\_frames['Amazon'].sort\_values('Date')

data\_ibm = data\_frames['IBM'].sort\_values('Date')

data\_apple = data\_frames['Apple'].sort\_values('Date')

print(data\_google)

print(data\_microsoft)

print(data\_amazon)

print(data\_ibm)

print(data\_apple)

for company in data\_frames:

data\_frames[company] = data\_frames[company].sort\_values('Date')

print(data\_frames[company].head())

import matplotlib.pyplot as plt

import seaborn as sns

for company\_name, df in data\_frames.items():

# Convert the index to datetime format (if it's not already)

if not isinstance(df.index, pd.DatetimeIndex):

df.index = pd.to\_datetime(df.index, format='%d-%m-%Y') # Use the correct date format

# Set plot style

sns.set\_style("darkgrid")

plt.figure(figsize=(15, 9))

plt.plot(df['Close'], label=f"{company\_name} Close Price")

# Set x-ticks for better readability (adjust frequency as needed)

plt.xticks(range(0, df.shape[0], 500), df.index[::500].strftime('%Y-%m-%d'), rotation=45)

# Add title, labels, and legend

plt.title(f"{company\_name} Stock Price", fontsize=18, fontweight='bold')

plt.xlabel('Date', fontsize=18)

plt.ylabel('Close Price (USD)', fontsize=18)

plt.legend()

plt.show()

# Create an empty dictionary to store the 'Close' prices for each company

company\_prices = {}

# Loop through each company in the data\_frames dictionary

for company\_name, df in data\_frames.items():

# Extract the 'Close' price column and store it in the dictionary

company\_prices[company\_name] = df[['Close']]

# Optionally print info to check

print(f"Info for {company\_name} Close Price Data:")

company\_prices[company\_name].info()

# Now, you have a dictionary with 'Close' prices for each company

# For example, access Google's 'Close' prices:

google\_close = company\_prices['Google']

from sklearn.preprocessing import MinMaxScaler

# Loop through each company in the data\_frames dictionary

for company in data\_frames:

# Create a copy of the original data to keep it independent

price\_data = data\_frames[company][['Close']].copy(deep=True)

# Initialize the MinMaxScaler with the desired range (-1, 1)

scaler = MinMaxScaler(feature\_range=(-1, 1))

# Apply the scaler to normalize the 'Close' prices

price\_data['Close'] = scaler.fit\_transform(price\_data[['Close']].values.reshape(-1, 1)).astype('float64')

# Store the normalized data as a permanent variable for each company

globals()[f'price\_{company.lower()}'] = price\_data

# Optionally, print the normalized data for the company

print(f"Normalized 'Close' prices for {company}:")

print(globals()[f'price\_{company.lower()}'].head(), "\n")

#Split Function

def split\_data(stock, lookback):

data\_raw = stock.to\_numpy() # convert to numpy array

data = []

# create all possible sequences of length seq\_len

for index in range(len(data\_raw) - lookback):

data.append(data\_raw[index: index + lookback])

data = np.array(data);

test\_set\_size = int(np.round(0.2\*data.shape[0]));

train\_set\_size = data.shape[0] - (test\_set\_size);

x\_train = data[:train\_set\_size,:-1,:]

y\_train = data[:train\_set\_size,-1,:]

x\_test = data[train\_set\_size:,:-1]

y\_test = data[train\_set\_size:,-1,:]

return [x\_train, y\_train, x\_test, y\_test]

# Import PyTorch

import torch

print(torch.\_\_version\_\_)

# Loop through each company in the data\_frames dictionary

for company in data\_frames:

# Get the company's price data (assumes 'Close' is the column of interest)

price\_data = data\_frames[company][['Close']].copy(deep=True)

# Define the lookback period (sequence length)

lookback = 19

# Split the data into training and testing sets

x\_train, y\_train, x\_test, y\_test = split\_data(price\_data, lookback)

# Print the shapes of the resulting data

print(f'{company} data:')

print('x\_train.shape = ', x\_train.shape)

print('y\_train.shape = ', y\_train.shape)

print('x\_test.shape = ', x\_test.shape)

print('y\_test.shape = ', y\_test.shape)

# Convert the data into PyTorch tensors (keeping variable names the same)

x\_train = torch.from\_numpy(x\_train).type(torch.Tensor)

x\_test = torch.from\_numpy(x\_test).type(torch.Tensor)

y\_train\_gru = torch.from\_numpy(y\_train).type(torch.Tensor)

y\_test\_gru = torch.from\_numpy(y\_test).type(torch.Tensor)

input\_dim = 1

hidden\_dim = 32

num\_layers = 2

output\_dim = 1

num\_epochs = 105

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

import torch

import torch.nn as nn

import time

from sklearn.metrics import mean\_squared\_error

import math

# Define GRU model class

class GRU(nn.Module):

def \_\_init\_\_(self, input\_dim, hidden\_dim, num\_layers, output\_dim):

super(GRU, self).\_\_init\_\_()

self.hidden\_dim = hidden\_dim

self.num\_layers = num\_layers

self.gru = nn.GRU(input\_dim, hidden\_dim, num\_layers, batch\_first=True)

self.fc = nn.Linear(hidden\_dim, output\_dim)

def forward(self, x):

h0 = torch.zeros(self.num\_layers, x.size(0), self.hidden\_dim).requires\_grad\_()

out, \_ = self.gru(x, h0.detach())

out = self.fc(out[:, -1, :])

return out

# Preprocess data

def preprocess\_data(data, lookback):

scaler = MinMaxScaler(feature\_range=(-1, 1))

data['Close'] = scaler.fit\_transform(data['Close'].values.reshape(-1, 1)).astype('float64')

data\_raw = data.to\_numpy()

sequences = [

data\_raw[i:i + lookback]

for i in range(len(data\_raw) - lookback)

]

sequences = np.array(sequences)

train\_size = int(np.round(0.8 \* sequences.shape[0]))

x\_train = sequences[:train\_size, :-1, :]

y\_train = sequences[:train\_size, -1, :]

x\_test = sequences[train\_size:, :-1, :]

y\_test = sequences[train\_size:, -1, :]

return x\_train, y\_train, x\_test, y\_test, scaler

# Train model

def train\_gru\_model(x\_train, y\_train\_gru, model, num\_epochs, criterion, optimiser):

hist = np.zeros(num\_epochs)

for epoch in range(num\_epochs):

y\_train\_pred = model(x\_train)

loss = criterion(y\_train\_pred, y\_train\_gru)

hist[epoch] = loss.item()

optimiser.zero\_grad()

loss.backward()

optimiser.step()

return model

# Evaluate model

def evaluate\_model(x\_test, y\_test\_gru, model, scaler):

y\_test\_pred = model(x\_test)

y\_test\_pred = scaler.inverse\_transform(y\_test\_pred.detach().numpy())

y\_test\_actual = scaler.inverse\_transform(y\_test\_gru.detach().numpy())

test\_rmse = math.sqrt(mean\_squared\_error(y\_test\_actual[:, 0], y\_test\_pred[:, 0]))

return y\_test\_pred, y\_test\_actual, test\_rmse

# Visualize results

def visualize\_predictions(data, y\_train\_pred, y\_test\_pred, scaler, lookback, company):

trainPredictPlot = np.empty\_like(data['Close'].values)

trainPredictPlot[:] = np.nan

trainPredictPlot[lookback:len(y\_train\_pred)+lookback] = y\_train\_pred[:, 0]

testPredictPlot = np.empty\_like(data['Close'].values)

testPredictPlot[:] = np.nan

testPredictPlot[len(y\_train\_pred)+lookback-1:len(data['Close'])-1] = y\_test\_pred[:, 0]

plt.figure(figsize=(14, 7))

plt.plot(scaler.inverse\_transform(data['Close'].values.reshape(-1, 1)), label="Actual")

plt.plot(trainPredictPlot, label="Train Prediction")

plt.plot(testPredictPlot, label="Test Prediction")

plt.title(f"{company} Stock Price Prediction")

plt.xlabel("Time")

plt.ylabel("Close Price (USD)")

plt.legend()

plt.show()

# Main function to process all companies

def process\_companies(companies\_data, lookback=19, num\_epochs=105, learning\_rate=0.01):

results\_summary = {}

for company, data in companies\_data.items():

print(f"Processing {company}...")

# Preprocess data

x\_train, y\_train, x\_test, y\_test, scaler = preprocess\_data(data, lookback)

# Convert to PyTorch tensors

x\_train = torch.from\_numpy(x\_train).type(torch.Tensor)

y\_train\_gru = torch.from\_numpy(y\_train).type(torch.Tensor)

x\_test = torch.from\_numpy(x\_test).type(torch.Tensor)

y\_test\_gru = torch.from\_numpy(y\_test).type(torch.Tensor)

# Initialize model, loss, and optimizer

model = GRU(input\_dim=1, hidden\_dim=32, num\_layers=2, output\_dim=1)

criterion = nn.MSELoss(reduction='mean')

optimiser = torch.optim.Adam(model.parameters(), lr=learning\_rate)

# Train the model

start\_time = time.time()

model = train\_gru\_model(x\_train, y\_train\_gru, model, num\_epochs, criterion, optimiser)

training\_time = time.time() - start\_time

print(f"Training time for {company}: {training\_time:.2f} seconds")

# Evaluate the model

y\_test\_pred, y\_test\_actual, test\_rmse = evaluate\_model(x\_test, y\_test\_gru, model, scaler)

y\_train\_pred = scaler.inverse\_transform(model(x\_train).detach().numpy())

train\_rmse = math.sqrt(mean\_squared\_error(

scaler.inverse\_transform(y\_train\_gru.detach().numpy())[:, 0],

y\_train\_pred[:, 0]

))

print(f"{company} - Train RMSE: {train\_rmse:.2f}, Test RMSE: {test\_rmse:.2f}")

# Store results

results\_summary[company] = {

"Train RMSE": train\_rmse,

"Test RMSE": test\_rmse,

"Training Time": training\_time

}

# Visualize

visualize\_predictions(data, y\_train\_pred, y\_test\_pred, scaler, lookback, company)

return results\_summary

# Example input data for all companies

companies\_data = {

"Microsoft": data\_microsoft[['Close']],

"IBM": data\_ibm[['Close']],

"Google": data\_google[['Close']],

"Amazon": data\_amazon[['Close']],

"Apple": data\_apple[['Close']]

}

# Run processing

results = process\_companies(companies\_data)

# Print summary of results

print("Summary of Results:")

for company, metrics in results.items():

print(f"{company}: Train RMSE = {metrics['Train RMSE']:.2f}, Test RMSE = {metrics['Test RMSE']:.2f}, Training Time = {metrics['Training Time']:.2f} seconds")

print(x\_train.shape)

print(x\_test.shape)

# Reshape the data to 2D for RandomForestRegressor

x\_train\_flattened = x\_train.reshape(x\_train.shape[0], -1) # Flatten the time dimension

x\_test\_flattened = x\_test.reshape(x\_test.shape[0], -1) # Flatten the time dimension

print(x\_train\_flattened.shape)

print(x\_test\_flattened.shape)

data\_frames = {

'Google': data\_google,

'Microsoft': data\_microsoft,

'Amazon': data\_amazon,

'IBM': data\_ibm,

'Apple': data\_apple

}

# Iterating over the dictionary to get each DataFrame

for company, df in data\_frames.items():

print(f"Company: {company}")

print(f"DataFrame for {company}:")

print(df.head()) # Displaying the first few rows of the DataFrame (can replace with other operations)

print() # Blank line for better readability

import numpy as np

import torch

import torch.nn as nn

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, mean\_absolute\_percentage\_error, r2\_score

# Define a simple GRU model

class GRUModel(nn.Module):

def \_\_init\_\_(self, input\_size, hidden\_size, output\_size):

super(GRUModel, self).\_\_init\_\_()

self.gru = nn.GRU(input\_size, hidden\_size, batch\_first=True)

self.fc = nn.Linear(hidden\_size, output\_size)

def forward(self, x):

out, \_ = self.gru(x)

out = self.fc(out[:, -1, :]) # Get the last output of the GRU layer

return out

# Function to train and predict using GRU

def train\_gru(x\_train, y\_train, x\_test, gru\_model):

# Reshape the data to fit the GRU input shape (batch\_size, seq\_len, input\_size)

x\_train\_2d = torch.Tensor(x\_train).unsqueeze(1) # Adding sequence length dimension (1 for single time step)

x\_test\_2d = torch.Tensor(x\_test).unsqueeze(1) # Adding sequence length dimension (1 for single time step)

# Loop through the test samples to predict each one

gru\_model.eval() # Switch to evaluation mode

y\_pred\_gru = []

for i in range(x\_test\_2d.shape[0]): # Iterating through each sample in the test set

sample\_input = x\_test\_2d[i:i+1, :, :] # Take a single sample

with torch.no\_grad(): # Disable gradient computation for inference

pred = gru\_model(sample\_input)

y\_pred\_gru.append(pred.item()) # Store the prediction

return np.array(y\_pred\_gru)

# Evaluation function

def evaluate\_model(y\_true, y\_pred):

rmse = np.sqrt(mean\_squared\_error(y\_true, y\_pred))

mae = mean\_absolute\_error(y\_true, y\_pred)

mape = mean\_absolute\_percentage\_error(y\_true, y\_pred) \* 100

r2 = r2\_score(y\_true, y\_pred)

return rmse, mae, mape, r2

# Experiment with different algorithms

def experiment\_with\_algorithms(x\_train, y\_train, x\_test, y\_test, gru\_model=None):

models = {

"RandomForest": RandomForestRegressor(),

"GRU": gru\_model

}

results = {}

for name, model in models.items():

if name != 'GRU':

# For traditional ML models, fit and predict

x\_train\_2d = x\_train.reshape(x\_train.shape[0], -1) # Flatten to 2D

x\_test\_2d = x\_test.reshape(x\_test.shape[0], -1) # Flatten to 2D

model.fit(x\_train\_2d, y\_train)

y\_pred = model.predict(x\_test\_2d)

else:

# For GRU, use the train\_gru function

y\_pred\_gru = train\_gru(x\_train, y\_train, x\_test, gru\_model)

# Evaluate both models

rmse, mae, mape, r2 = evaluate\_model(y\_test, y\_pred\_gru if name == 'GRU' else y\_pred)

results[name] = {

'RMSE': rmse,

'MAE': mae,

'MAPE': mape,

'R²': r2

}

return results

# Define your data\_frames dictionary (assuming your DataFrames are loaded correctly)

data\_frames = {

'Google': data\_google,

'Microsoft': data\_microsoft,

'Amazon': data\_amazon,

'IBM': data\_ibm,

'Apple': data\_apple

}

# Loop through the dictionary for each company

for company, df in data\_frames.items():

print(f"\nModel Evaluation for {company}")

# You need to define x\_train, y\_train, x\_test, y\_test based on the DataFrame for each company

# This is just a placeholder. You should replace this with actual data for each company.

x\_train = np.random.rand(100, 10) # Replace with your training features

y\_train = np.random.rand(100) # Replace with your training labels

x\_test = np.random.rand(20, 10) # Replace with your test features

y\_test = np.random.rand(20) # Replace with your test labels

# Initialize the GRU model (example hyperparameters)

gru\_model = GRUModel(input\_size=10, hidden\_size=64, output\_size=1)

# Experiment with different models

results = experiment\_with\_algorithms(x\_train, y\_train, x\_test, y\_test, gru\_model=gru\_model)

# Display evaluation results

print(f"Results for {company}:")

for model\_name, metrics in results.items():

print(f"{model\_name}:")

for metric\_name, metric\_value in metrics.items():

print(f" {metric\_name}: {metric\_value:.2f}")

import numpy as np

import torch

import torch.nn as nn

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, mean\_absolute\_percentage\_error, r2\_score

# Define a simple GRU model

class GRUModel(nn.Module):

def \_\_init\_\_(self, input\_size, hidden\_size, output\_size):

super(GRUModel, self).\_\_init\_\_()

self.gru = nn.GRU(input\_size, hidden\_size, batch\_first=True)

self.fc = nn.Linear(hidden\_size, output\_size)

def forward(self, x):

out, \_ = self.gru(x)

out = self.fc(out[:, -1, :]) # Get the last output of the GRU layer

return out

# Function to train and predict using GRU

def train\_gru(x\_train, y\_train, x\_test, gru\_model):

# Reshape the data to fit the GRU input shape (batch\_size, seq\_len, input\_size)

x\_train\_2d = torch.Tensor(x\_train).unsqueeze(1) # Adding sequence length dimension (1 for single time step)

x\_test\_2d = torch.Tensor(x\_test).unsqueeze(1) # Adding sequence length dimension (1 for single time step)

# Loop through the test samples to predict each one

gru\_model.eval() # Switch to evaluation mode

y\_pred\_gru = []

for i in range(x\_test\_2d.shape[0]): # Iterating through each sample in the test set

sample\_input = x\_test\_2d[i:i+1, :, :] # Take a single sample

with torch.no\_grad(): # Disable gradient computation for inference

pred = gru\_model(sample\_input)

y\_pred\_gru.append(pred.item()) # Store the prediction

return np.array(y\_pred\_gru)

# Evaluation function

def evaluate\_model(y\_true, y\_pred):

rmse = np.sqrt(mean\_squared\_error(y\_true, y\_pred))

mae = mean\_absolute\_error(y\_true, y\_pred)

mape = mean\_absolute\_percentage\_error(y\_true, y\_pred) \* 100

r2 = r2\_score(y\_true, y\_pred)

return rmse, mae, mape, r2

# Experiment with different algorithms

def experiment\_with\_algorithms(x\_train, y\_train, x\_test, y\_test, gru\_model=None):

models = {

"RandomForest": RandomForestRegressor(),

"GRU": gru\_model

}

results = {}

for name, model in models.items():

if name != 'GRU':

# For traditional ML models, fit and predict

x\_train\_2d = x\_train.reshape(x\_train.shape[0], -1) # Flatten to 2D

x\_test\_2d = x\_test.reshape(x\_test.shape[0], -1) # Flatten to 2D

model.fit(x\_train\_2d, y\_train)

y\_pred = model.predict(x\_test\_2d)

predictions = y\_pred # Store predictions for RandomForest

else:

# For GRU, use the train\_gru function

y\_pred\_gru = train\_gru(x\_train, y\_train, x\_test, gru\_model)

predictions = y\_pred\_gru # Store predictions for GRU

# Evaluate both models

rmse, mae, mape, r2 = evaluate\_model(y\_test, predictions)

results[name] = {

'RMSE': rmse,

'MAE': mae,

'MAPE': mape,

'R²': r2,

'Predictions': predictions # Store predictions here

}

return results

# Assuming you have the all\_results dictionary that stores predictions for all companies

# Example usage for one company

all\_results = {}

for company, df in data\_frames.items():

x\_train = np.random.rand(100, 10) # Replace with your training features

y\_train = np.random.rand(100) # Replace with your training labels

x\_test = np.random.rand(20, 10) # Replace with your test features

y\_test = np.random.rand(20) # Replace with your test labels

# Initialize the GRU model (example hyperparameters)

gru\_model = GRUModel(input\_size=10, hidden\_size=64, output\_size=1)

# Experiment with different models

results = experiment\_with\_algorithms(x\_train, y\_train, x\_test, y\_test, gru\_model=gru\_model)

# Store the results for each company

all\_results[company] = results

# Now you can call the plot function

def plot\_predictions\_for\_all\_companies(all\_results, y\_test):

for company, results in all\_results.items():

print(f"\nPlotting Predictions for {company}")

# Get the model predictions for RandomForest and GRU

rf\_predictions = results['RandomForest']['Predictions']

gru\_predictions = results['GRU']['Predictions']

# Plot for the current company

plt.figure(figsize=(12, 6))

# Plot RandomForest predictions vs actual

plt.subplot(1, 2, 1)

plt.plot(y\_test, label='True values', color='blue')

plt.plot(rf\_predictions, label='Random Forest Predictions', color='red')

plt.title(f'{company} - Random Forest Predictions vs Actual')

plt.xlabel('Index')

plt.ylabel('Value')

plt.legend()

# Plot GRU predictions vs actual

plt.subplot(1, 2, 2)

plt.plot(y\_test, label='True values', color='blue')

plt.plot(gru\_predictions, label='GRU Predictions', color='green')

plt.title(f'{company} - GRU Predictions vs Actual')

plt.xlabel('Index')

plt.ylabel('Value')

plt.legend()

plt.tight\_layout()

plt.show()

# After gathering all results, call the visualization function

plot\_predictions\_for\_all\_companies(all\_results, y\_test)