VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

FALL SEMESTER 24-25

Market Pulse:- Stay Ahead With Predictive Power

(Stock Prediction web Application Using Machine Learning)

Submitted To:

Dr B Saleena

Python Programming (PMCA602L)

Dr Jayasudha M

Data Structures and Algorithms (PMCA501L)

Submitted By:

Name: Sanskar Kumar
Register No: 24MCA1066

Name: Vishwas Kumar
Register No: 24MCA1001

Name: Shubham Kumar
Register No: 24MCA10

Abstract

This project develops a predictive model to forecast stock prices, utilizing historical data and machine learning. The model integrates with an interactive frontend built using Streamlit, enabling users to visualize trends and predictions effectively. Key features include moving averages, graphical comparisons of actual versus predicted values, and user-friendly interaction.

Introduction

Stock price prediction is a fundamental problem in financial markets, where accurate forecasting can significantly impact investment strategies, risk management, and decision-making processes. The stock market, known for its volatility and sensitivity to a variety of economic, political, and sectoral influences, presents a complex challenge for prediction models. As the amount of financial data grows exponentially, there is an increasing need for automated tools that can analyze vast amounts of historical data, identify trends, and generate predictions in real-time.

This project aims to address these challenges by developing a machine learning-based system that leverages historical stock price data to forecast future prices. By using Recurrent Neural Networks (RNNs), a type of deep learning model specifically suited for time-series forecasting, the system is designed to analyze patterns in stock price movements over time. The project integrates this model with a user-friendly visualization interface built using Streamlit, allowing users to interact with the system, visualize trends, and compare predictions against actual data.

The goal of this project is not only to provide a predictive tool but also to make stock market analysis accessible and intuitive for both novice and experienced users, helping them stay informed about market trends and make data-driven investment decisions.

Problem Statement

The ability to predict stock prices has long been a crucial objective for investors, analysts, and financial institutions, given the potential to make data-driven decisions that reduce risks and maximize returns. However, accurately forecasting stock prices remains an inherently challenging task. Stock markets are influenced by numerous factors such as economic indicators, geopolitical events, corporate earnings, and even social sentiment, making price movements complex and unpredictable. Moreover, historical data alone may not capture all the nuances of market behavior, adding further difficulty to prediction models.

Traditional methods, including statistical models such as ARIMA or simple moving averages, often struggle to provide accurate predictions in highly volatile markets. These methods rely on predefined assumptions that fail to capture the nonlinear and dynamic nature of stock price movements. As a result, many prediction systems lack the ability to adapt to sudden shifts in market conditions, making them less reliable in real-world applications.

To address these issues, there is a growing interest in employing machine learning models, particularly deep learning techniques, for stock price prediction. Recurrent Neural Networks (RNNs), especially Long Short-Term Memory (LSTM) networks, have shown promise in handling time-series data by learning from historical patterns and making predictions based on these learned patterns. However, even with advanced algorithms, challenges persist in ensuring that these models can handle noisy, incomplete, and often volatile financial data.

This project aims to develop an automated stock price prediction system that leverages machine learning, particularly RNNs, to forecast future prices based on historical data. The system will also integrate an interactive frontend that visualizes predictions, allowing users to compare actual and predicted stock prices. By doing so, it aims to provide a more accurate, reliable, and user-friendly tool for stock market analysis.

DataSet Used

For this project, the dataset used is historical stock price data retrieved from the **Yahoo Finance API**. This is prebuilt available in python we have to import this API as (yfinance) by writing the command as (pip install yfinance) and after that we are importing this module as

import yfinance as yf

The dataset includes the following attributes:

1. **Open**: The price at which the stock opened for trading on a given day.
2. **High**: The highest price reached by the stock during the trading day.
3. **Low**: The lowest price reached by the stock during the trading day.
4. **Close**: The price of the stock at the end of the trading day. This is typically the most important price for analysis.
5. **Volume**: The total number of shares traded during the trading day.

Data Retrieval

- The data spans a **20-year period**, but the exact range can be adjusted by the user based on the stock symbol they input.
- **Frequency**: The data is typically daily, meaning each record corresponds to a single trading day.

Data Processing

- **Rolling Averages**: The dataset is processed to compute moving averages (e.g., for 100, 200, and 250 days) to identify trends in stock prices over time.

- **Normalization**: The closing prices are scaled using **MinMaxScaler** to fit within a range of 0 to 1, which helps the machine learning model to perform better, especially when using deep learning techniques.

Methodology

The methodology of this stock price prediction system follows a structured approach that involves multiple stages, including data collection, preprocessing, model training, and visualization. Each step is designed to ensure the system provides reliable predictions and an interactive experience for the user.

1. Data Collection

The first stage in the system involves collecting historical stock price data. This is done dynamically using the **Yahoo Finance API**, which provides a vast dataset of stock prices from multiple companies across a significant period. The data includes daily stock prices, with key attributes such as the opening price, closing price, highest price, lowest price, and trading volume. These values are essential for understanding the trends and behavior of a stock over time.

The data collection process is automated, allowing the user to specify the stock symbol they are interested in. Once the symbol is entered, the system fetches the corresponding stock data from Yahoo Finance for the past 20 years. Users can adjust the time range, but the default setting retrieves two decades' worth of information.

2. Data Preprocessing

Once the data is collected, it undergoes preprocessing to prepare it for model training. The primary goal of data preprocessing is to ensure that the data is in a format that can be efficiently used by the machine learning model. Several important steps are carried out during this stage:

- **Normalization**: Since the stock data can vary greatly in magnitude, it is essential to scale the values to a common range. This is done using the **MinMaxScaler**, which normalizes the data to fall within a range of 0 to 1. This step helps prevent any one feature from dominating the model, especially when using deep learning techniques.

- **Moving Averages**: To identify trends and reduce noise in the data, moving averages for different periods (e.g., 100, 200, and 250 days) are calculated. These moving averages help smooth out fluctuations in stock prices and provide a clearer view of the stock’s overall trend. By including these features, the model can capture short- and long-term trends in the stock's performance.

- **Data Splitting**: The dataset is split into training and testing sets, typically with 70% of the data used for training and 30% used for testing. This split is essential for evaluating the model’s performance on unseen data and ensuring that it generalizes well.

3. Modeling

The core of the prediction system lies in the **Recurrent Neural Network (RNN)**. RNNs are a class of deep learning models that are particularly well-suited for sequence prediction tasks like stock price forecasting. Unlike traditional feed-forward neural networks, RNNs have connections that allow information to persist over time, making them ideal for processing sequential data such as time-series.

In this system, the RNN model is pre-trained using historical stock data, enabling it to learn patterns and dependencies in the price movements. The model processes the sequences of historical stock prices (up to 100 days) to predict the stock's future price for the next day. Once trained, the model can make predictions on the testing set, which contains data the model has not seen before.

The primary benefit of using an RNN over other models is its ability to capture long-term dependencies in the data. This is crucial for stock price prediction, as prices are influenced by both short-term and long-term factors. By training on a large dataset with temporal patterns, the model becomes capable of recognizing trends and making accurate predictions based on past price movements.

4. Visualization

To make the model's predictions accessible and understandable, the system integrates **Streamlit** for interactive visualization. Streamlit is a Python library that simplifies the process of building interactive web applications for data science and machine learning projects. With Streamlit, users can visualize both the actual and predicted stock prices, compare different moving averages, and explore the model's performance through a series of intuitive plots.

Key visualizations include:

- **Stock Price Trends**: Line charts showing the historical stock price and the predicted prices for future dates.
- **Moving Averages**: Charts displaying the calculated moving averages for various time periods, helping users identify long-term trends and short-term fluctuations.
- **Comparison Graphs**: These graphs show how well the model's predictions align with the actual stock price data, providing a clear view of the model's performance.

The interactive interface allows users to input different stock symbols, view real-time data updates, and make comparisons between multiple stocks or different prediction periods.

Algorithm

The **Recurrent Neural Network (RNN)** used in this project is designed to process time-series data and predict future values based on past observations. RNNs are unique because they maintain a "memory" of previous inputs, allowing them to capture patterns in sequences of data.

The algorithm works as follows:

1. **Normalization with MinMaxScaler**: Before feeding the data into the RNN, the stock prices are scaled using the MinMaxScaler. This normalization ensures that all the features are on a similar scale, which helps improve the training performance of the neural network.

2. **Data Splitting**: The historical stock data is split into training (70%) and testing (30%) sets. The training set is used to train the RNN, while the testing set is used to evaluate its performance.

3. **Sequence Preparation**: Since RNNs require sequential data as input, the data is converted into sequences of 100 consecutive days. These sequences represent the historical data that the model uses to predict the stock price for the next day.

4. **Prediction**: Once the RNN model is trained on the training data, it is used to generate predictions for the testing set. The predictions are then scaled back to the original price range using the inverse of the MinMaxScaler, making them interpretable for the user.

Implementation

The implementation of the stock price prediction system uses several powerful libraries to bring together data collection, machine learning, and visualization:

1. Input Module

The input module allows users to input the stock symbol (e.g., "GOOG" for Google) they want to analyze. This input is then used to fetch the corresponding historical stock data from Yahoo Finance. The user also has the option to adjust the time range for the stock data.

2. Data Module

This module handles the retrieval of stock data from the Yahoo Finance API. It also performs the necessary preprocessing steps, including normalization and the calculation of moving averages. The data is then formatted into sequences that the RNN can process for training and prediction.

3. Prediction Module

The prediction module houses the pre-trained RNN model. It uses the training data to generate predictions for future stock prices. This module also handles the inverse scaling of predicted values, ensuring that the output is in a format that users can easily interpret and use for decision-making.

4. Visualization Module

Using Streamlit and Matplotlib, the visualization module allows users to interact with the predictions and trends visually. It generates various plots such as the stock's actual closing price versus the predicted price, as well as moving averages over different periods.

Code Snippets for this whole Model :

Stock_Prediction.py

```
import streamlit as st
import pandas as pd
import numpy as np
from tensorflow.keras.models import load_model
import matplotlib.pyplot as plt
import yfinance as yf
from datetime import datetime
from sklearn.preprocessing import MinMaxScaler


st.title("Market Pulse:- Stay Ahead With Predictive Power")
stock = st.text_input("Enter the Stock ID", "GOOG")
end = datetime.now()
start = datetime(end.year-20, end.month, end.day)

data = yf.download(stock, start, end)

if isinstance(data.columns, pd.MultiIndex):
    data.columns = data.columns.droplevel('Ticker')

try:
    model = load_model("Latest_stock_price_model.keras")
except FileNotFoundError:
    st.error("Model file not found. Ensure 'Latest_stock_price_model.keras' is in
the correct path.")
    st.stop()

st.subheader("Stock Data")
st.write(data)
```

```
def plot_graph(figsize, values, full_data, extra_data=0, extra_dataset=None):
    fig = plt.figure(figsize=figsize)
      plt.plot(full_data.index,  full_data['Close'],  'b',  label="Original  Close
Price")
    plt.plot(full_data.index, values, 'orange', label="Moving Average")
    if extra_data and extra_dataset is not None:
         plt.plot(full_data.index,  extra_dataset,  'g',  label="Additional  Moving
Average")
    plt.legend()
    return fig

for ma in [250, 200, 100]:
    data[f'MA_for_{ma}_days'] = data['Close'].rolling(ma).mean()
    st.subheader(f'Original Close Price and MA for {ma} days')
    st.pyplot(plot_graph((15, 6), data[f'MA_for_{ma}_days'], data))

splitting_len = int(len(data) * 0.7)
x_test = pd.DataFrame(data['Close'][splitting_len:])

scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(x_test)

x_data, y_data = [], []
for i in range(100, len(scaled_data)):
    x_data.append(scaled_data[i - 100:i])
    y_data.append(scaled_data[i])

x_data, y_data = np.array(x_data), np.array(y_data)

predictions = model.predict(x_data)
inv_pre = scaler.inverse_transform(predictions)
inv_y_test = scaler.inverse_transform(y_data)

ploting_data = pd.DataFrame({
    'Original Test Data': inv_y_test.flatten(),
    'Predictions': inv_pre.flatten()
}, index=data.index[splitting_len + 100:])

st.subheader("Original values vs Predicted values")
st.write(ploting_data)

st.subheader('Original Close Price vs Predicted Close price')
fig = plt.figure(figsize=(15, 6))
```

```
plt.plot(data.index[:splitting_len + 100], data['Close'][:splitting_len + 100],
label="Data - not used")
plt.plot(ploting_data['Original Test Data'], label="Original Test Data")
plt.plot(ploting_data['Predictions'], label="Predicted Test Data")
plt.legend()
st.pyplot(fig)
```

Stock_Prediction.ipynb

```
import yfinance as yf
```

```
from datetime import datetime
end = datetime.now()
start = datetime(end.year-20,end.month,end.day)
```

```
stock = 'GOOG'
data = yf.download(stock,start,end)
```

```
data.tail(5)
```

Price	Adj Close	Close	High	Low	Open	Volume
Ticker	GOOG	GOOG	GOOG	GOOG	GOOG	GOOG
Date						
2024-10-29 00:00:00+00:00	171.139999	171.139999	171.860001	168.660004	169.384995	28916100
2024-10-30 00:00:00+00:00	176.139999	176.139999	183.789993	175.744995	182.410004	49698300
2024-10-31 00:00:00+00:00	172.690002	172.690002	178.419998	172.559998	174.720001	32801900
2024-11-01 00:00:00+00:00	172.649994	172.649994	173.820007	170.309998	171.539993	21752900
2024-11-04 00:00:00+00:00	170.679993	170.679993	171.919998	169.485001	171.240005	16172300

```
data.shape
```

```
data.describe()
```

Price	Adj Close	Close	High	Low	Open	Volume
Ticker	GOOG	GOOG	GOOG	GOOG	GOOG	GOOG
count	5033.000000	5033.000000	5033.000000	5033.000000	5033.000000	5.033000e+03
mean	47.232294	47.342557	47.821751	46.851248	47.324787	1.118498e+08
std	45.129748	45.225151	45.696721	44.756690	45.202831	1.449350e+08
min	4.102013	4.112087	4.221676	4.017691	4.096396	1.584340e+05
25%	13.185575	13.217956	13.372377	13.089936	13.250334	2.674390e+07
50%	27.624363	27.692204	27.868486	27.457115	27.746250	5.221800e+07
75%	62.029667	62.181999	62.737000	61.682049	62.150002	1.362287e+08
max	192.406723	192.660004	193.309998	190.619995	191.750000	1.650833e+09

```
data.isna().sum()
```

		0
Price	Ticker	
Adj Close	GOOG	0
Close	GOOG	0
High	GOOG	0
Low	GOOG	0
Open	GOOG	0
Volume	GOOG	0

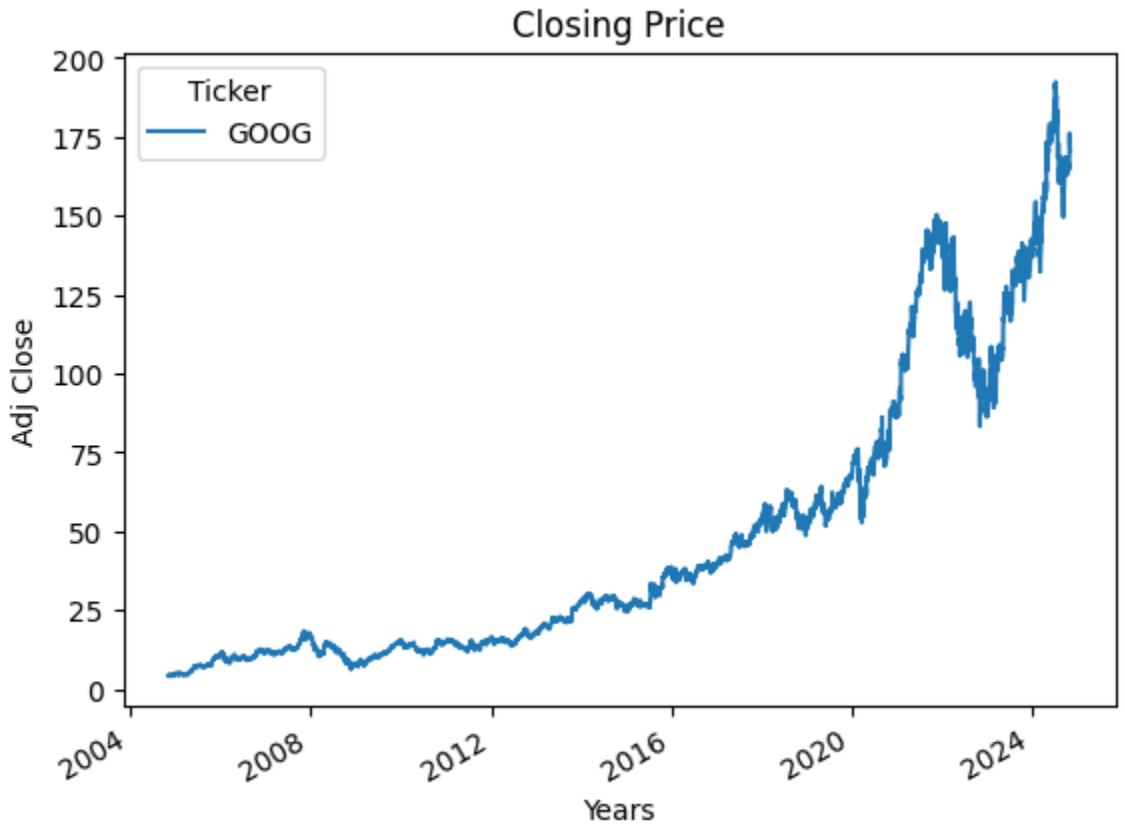
dtype: int64

```
data.columns
```

```
MultiIndex([('Adj Close', 'GOOG'),
            (    'Close', 'GOOG'),
            (     'High', 'GOOG'),
            (      'Low', 'GOOG'),
            (     'Open', 'GOOG'),
            (   'Volume', 'GOOG')],
           names=['Price', 'Ticker'])
```

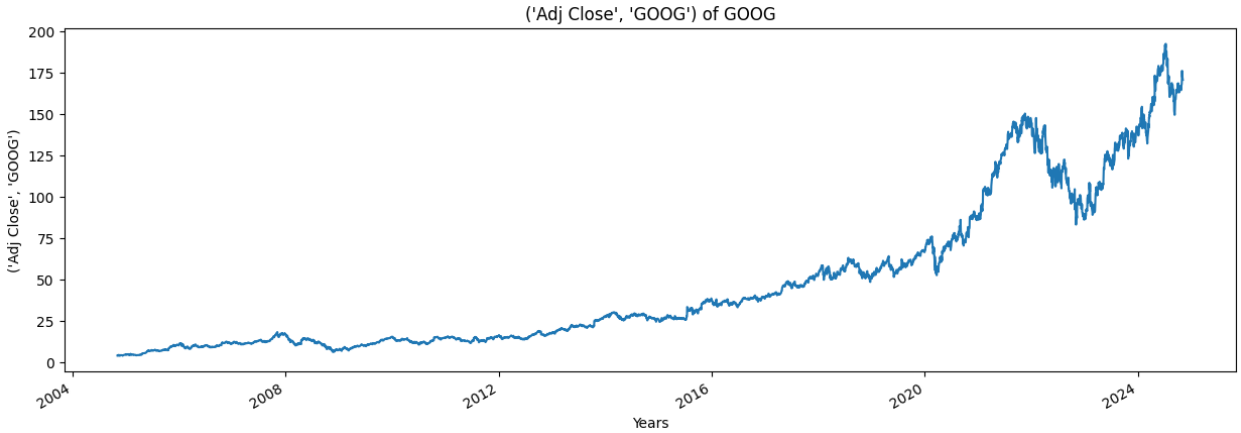
```
import matplotlib.pyplot as plt
%matplotlib inline
```

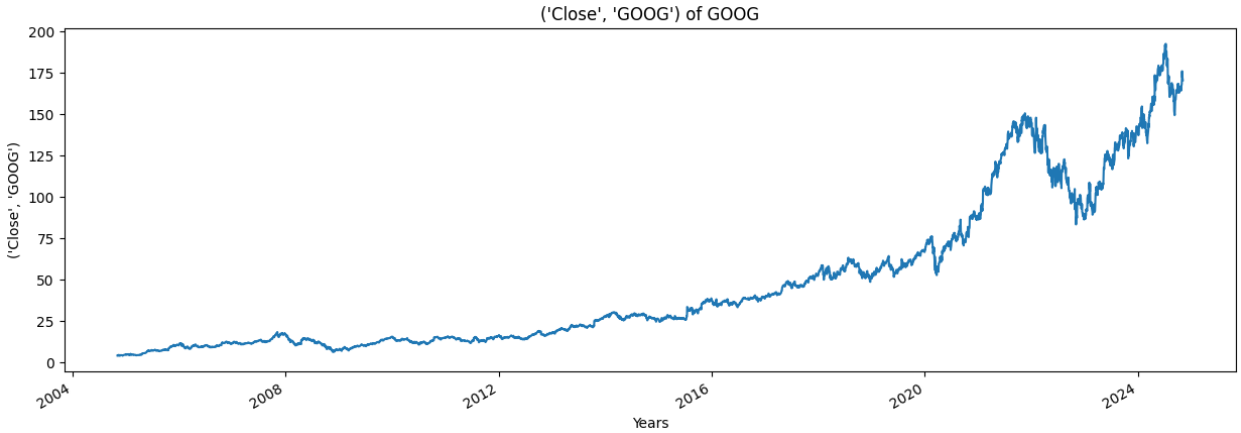
```
data['Adj Close'].plot()
plt.xlabel("Years")
plt.ylabel("Adj Close")
plt.title("Closing Price")
plt.show()
```

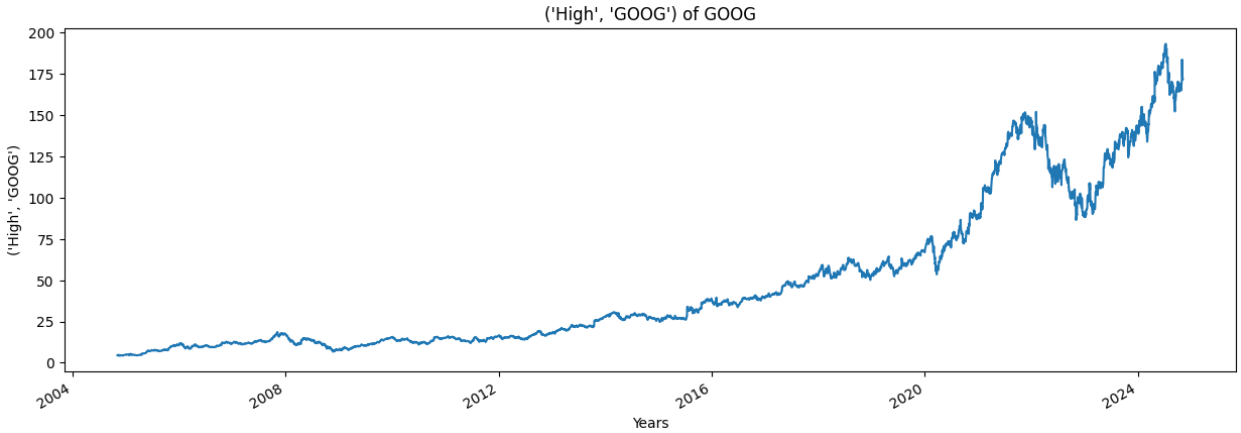


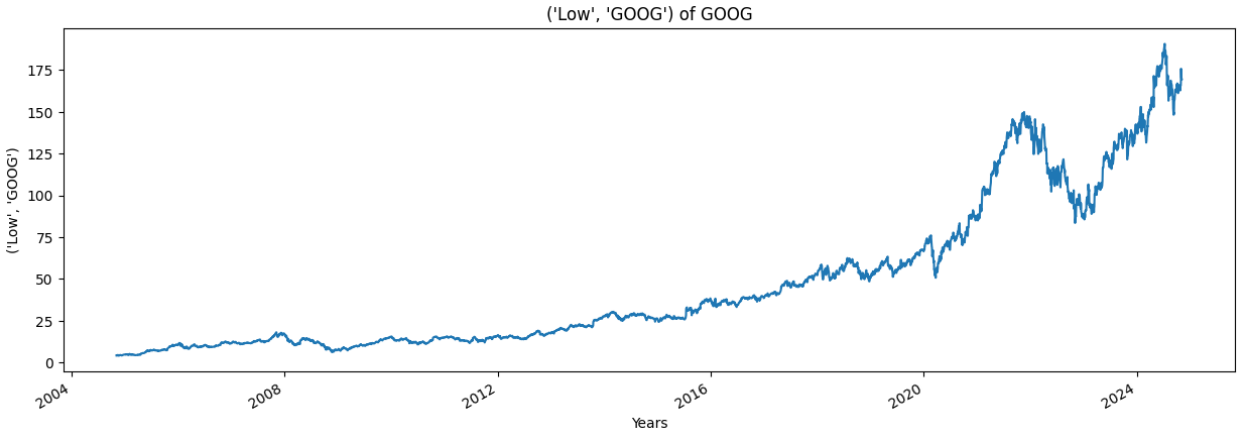

```
def plot_graph(figSize,values,column_name):
  plt.figure()
  values.plot(figsize =figSize)
  plt.xlabel("Years")
  plt.ylabel(column_name)
  plt.title(f"{column_name} of {stock}")
  plt.show()
```

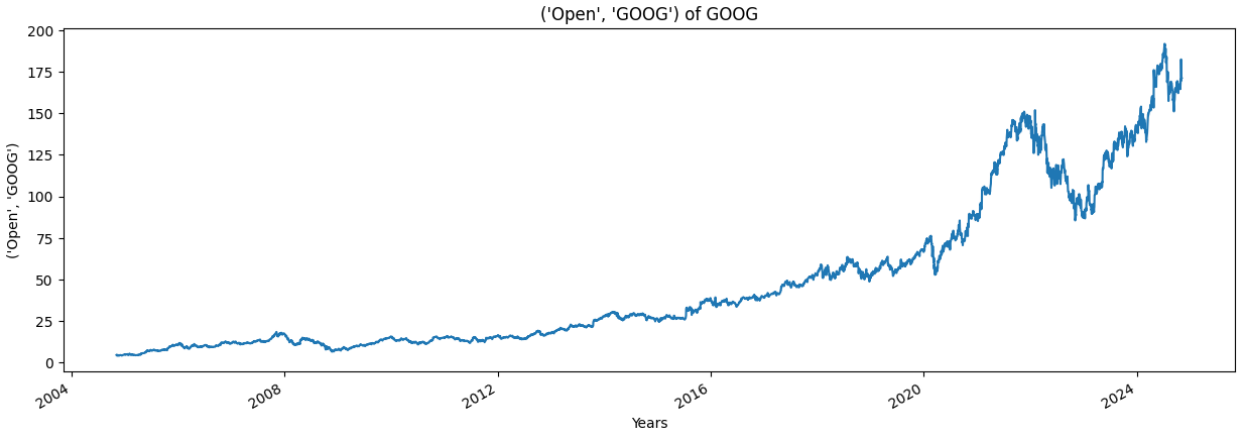
```
for column in data.columns:
  plot_graph((15,5),data[column],column)
```

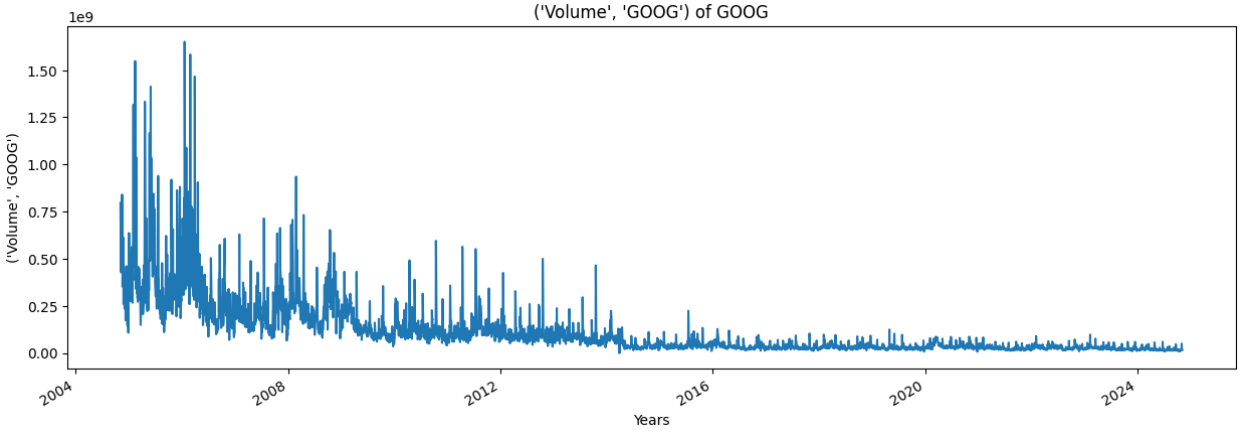

('Adj Close', 'GOOG') of GOOG
('Adj Close', 'GOOG')
Years
0
25
50
75
100
125
150
175
200
2004
2008
2012
2016
2020
2024


('Close', 'GOOG') of GOOG
('Close', 'GOOG')
Years
0
25
50
75
100
125
150
175
200
2004
2008
2012
2016
2020
2024


('High', 'GOOG') of GOOG
('High', 'GOOG')
Years
0
25
50
75
100
125
150
175
200
2004
2008
2012
2016
2020
2024








```
import pandas as pd
df = pd.DataFrame([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])
df.head()
```

	0
0	10
1	20
2	30
3	40
4	50

```
df['MA'] = df.rolling(5).mean()
df
```

	0	MA
0	10	NaN
1	20	NaN
2	30	NaN
3	40	NaN
4	50	30.0
5	60	40.0
6	70	50.0
7	80	60.0
8	90	70.0
9	100	80.0

```
for i in range(2004,2025):
    print(i,list(data.index.year).count(i))
```

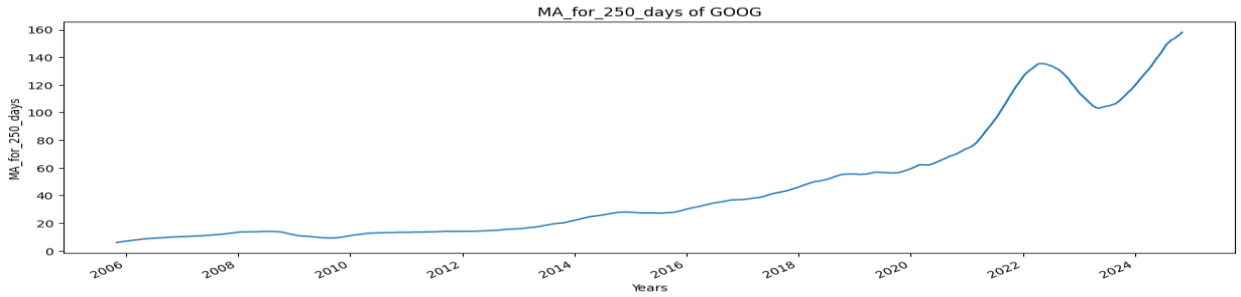
```
2004 39
2005 252
2006 251
2007 251
2008 253
2009 252
2010 252
2011 252
2012 250
2013 252
2014 252
2015 252
2016 252
2017 251
2018 251
2019 252
2020 253
2021 252
2022 251
2023 250
2024 213
```

```
data['MA_for_250_days'] = data['Adj Close'].rolling(250).mean()
data['MA_for_250_days'][0:250].tail()
```

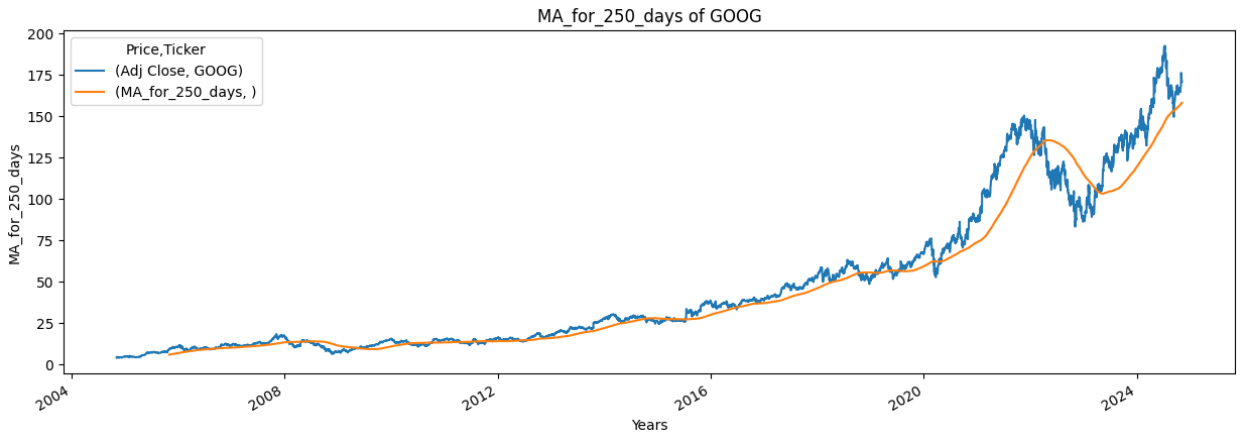
Date	MA_for_250_days
2005-10-26 00:00:00+00:00	NaN
2005-10-27 00:00:00+00:00	NaN
2005-10-28 00:00:00+00:00	NaN
2005-10-31 00:00:00+00:00	NaN
2005-11-01 00:00:00+00:00	5.978247

dtype: float64

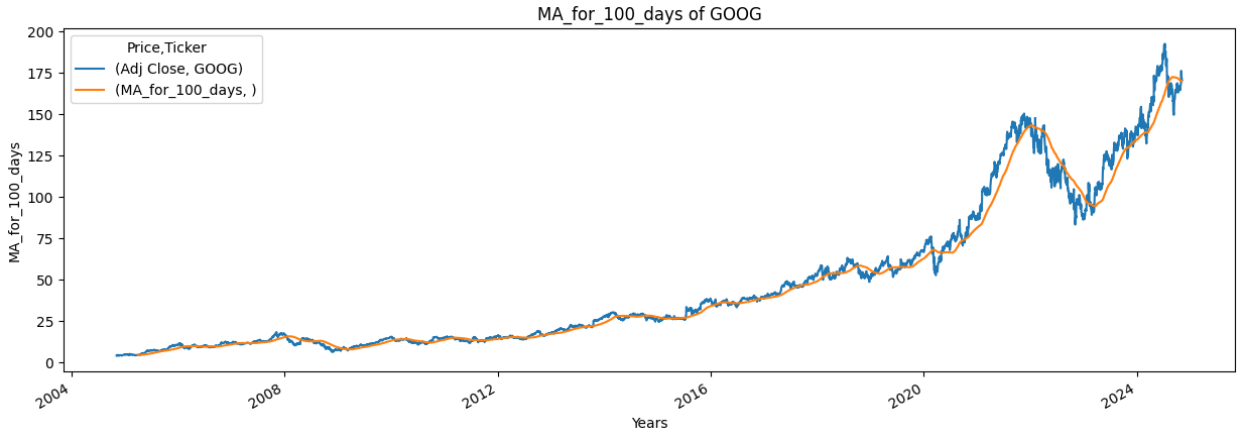
```
plot_graph((15,5), data['MA_for_250_days'], 'MA_for_250_days')
```



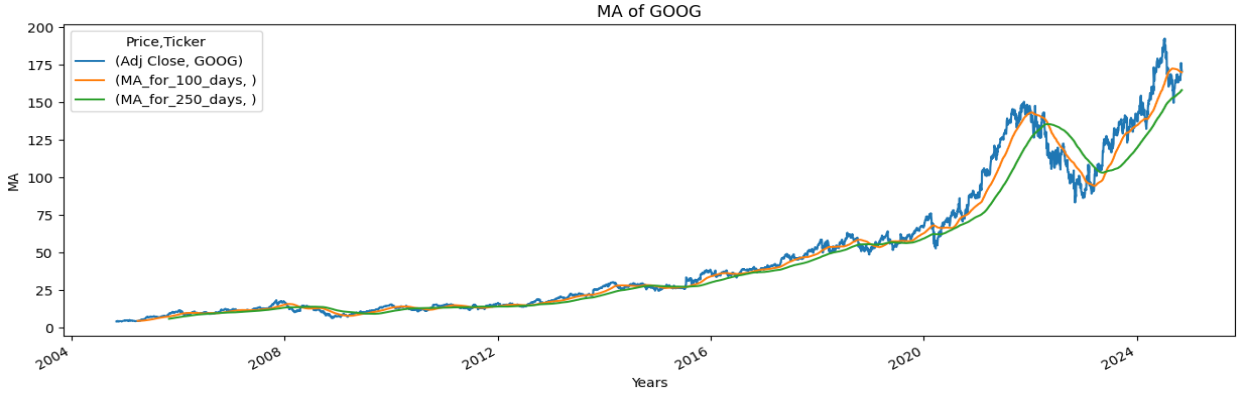
```
plot_graph((15,5), data[['Adj Close','MA_for_250_days']], 'MA_for_250_days')
```




```
data['MA_for_100_days'] = data['Adj Close'].rolling(100).mean()
plot_graph((15,5), data[['Adj Close','MA_for_100_days']], 'MA_for_100_days')
```



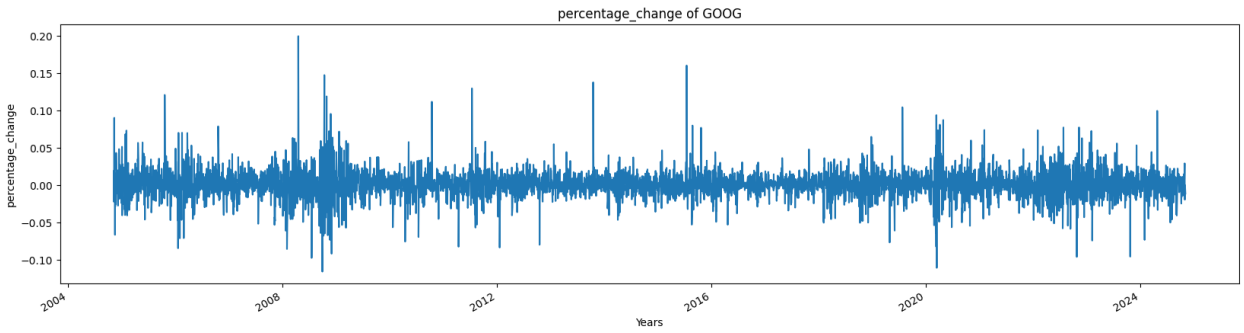

```
plot_graph((15,5), data[['Adj Close','MA_for_100_days', 'MA_for_250_days']], 'MA')
```



```
data['percentage_change_cp'] = data['Adj Close'].pct_change()[f'{stock}']
data[['Adj Close','percentage_change_cp']].head()
```

Price	Adj Close	percentage_change_cp
Ticker	GOOG	
Date		
2004-11-05 00:00:00+00:00	4.207607	NaN
2004-11-08 00:00:00+00:00	4.287112	0.018896
2004-11-09 00:00:00+00:00	4.191458	-0.022312
2004-11-10 00:00:00+00:00	4.170587	-0.004979
2004-11-11 00:00:00+00:00	4.547246	0.090313

```
plot_graph((20,5), data['percentage_change_cp'], 'percentage_change')
```




```
Adj_close_price = data[['Adj Close']]
max(Adj_close_price.values),min(Adj_close_price.values)
```

(array([192.40672302]), array([4.10201311]))

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(Adj_close_price)
scaled_data
```

```
array([[5.60759640e-04],
       [9.82976607e-04],
       [4.74999469e-04],
       ...,
       [8.95293535e-01],
       [8.95081068e-01],
       [8.84619294e-01]])
```

```
len(scaled_data)
```

5033

```
x_data = []
y_data = []

for i in range(100, len(scaled_data)):
    x_data.append(scaled_data[i-100:i])
    y_data.append(scaled_data[i])

import numpy as np
x_data, y_data = np.array(x_data), np.array(y_data)
```

```
x_data[0],y_data[0]
```

```
(array([[0.00056076],
       [0.00098298],
       [0.000475  ],
       [0.00036416],
       [0.00236443],
       [0.00222985],
       [0.00260853],
       [0.00098166],
       [0.00097638],
       [0.00032194],
       [0.00056736],
       [0.        ],
       [0.0003193 ],
       [0.00127458],
       [0.00188548],
       [0.0021045 ],
       [0.00222721],
       [0.00196069],
       [0.0018868 ],
       [0.00201874],
       [0.00147645],
       [0.0008352 ],
       [0.00064388],
       [0.00109909],
```

```
int(len(x_data)*0.7)
```

3453

```
4908-100-int(len(x_data)*0.7)
```

1355

```
splitting_len = int(len(x_data)*0.7)
x_train = x_data[:splitting_len]
y_train = y_data[:splitting_len]

x_test = x_data[splitting_len:]
y_test = y_data[splitting_len:]
```

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(3453, 100, 1)
(3453, 1)
(1480, 100, 1)
(1480, 1)
```

```
from keras.models import Sequential
from keras.layers import Dense, LSTM
```

```
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))
```

```
model.compile(optimizer='adam', loss='mean_squared_error')
```

```
model.fit(x_train, y_train, batch_size=1, epochs = 2)
```

```
Epoch 1/2
3453/3453 ━━━━━━━━━━━━━━━━━━━━ 310s 88ms/step - loss: 3.0477e-04
Epoch 2/2
3453/3453 ━━━━━━━━━━━━━━━━━━━━ 339s 93ms/step - loss: 6.8524e-05

<keras.src.callbacks.history.History at 0x7f6a048c3d60>
```

```
model.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
lstm_6 (LSTM)	(None, 100, 128)	66,560
lstm_7 (LSTM)	(None, 64)	49,408
dense_6 (Dense)	(None, 25)	1,625
dense_7 (Dense)	(None, 1)	26

Total params: 352,859 (1.35 MB)

Trainable params: 117,619 (459.45 KB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 235,240 (918.91 KB)

```
predictions = model.predict(x_test)
predictions
```

```
47/47 ━━━━━━━━━━━━━━━━━━━━ 3s 64ms/step

array([[0.27238572],
       [0.27008662],
       [0.26872337],
       ...,
       [0.9336694 ],
       [0.93606496],
       [0.93305296]], dtype=float32)
```

```
inv_predictions = scaler.inverse_transform(predictions)
inv_predictions
```

```
array([[ 55.393528],
       [ 54.960598],
       [ 54.70389 ],
       ...,
       [179.91635 ],
       [180.36745 ],
       [179.80028 ]], dtype=float32)
```

```
inv_y_test = scaler.inverse_transform(y_test)
inv_y_test
```

```
array([[ 51.30949402],
       [ 51.02519226],
       [ 50.34685898],
       ...,
       [172.69000244],
       [172.6499939 ],
       [170.67999268]])
```

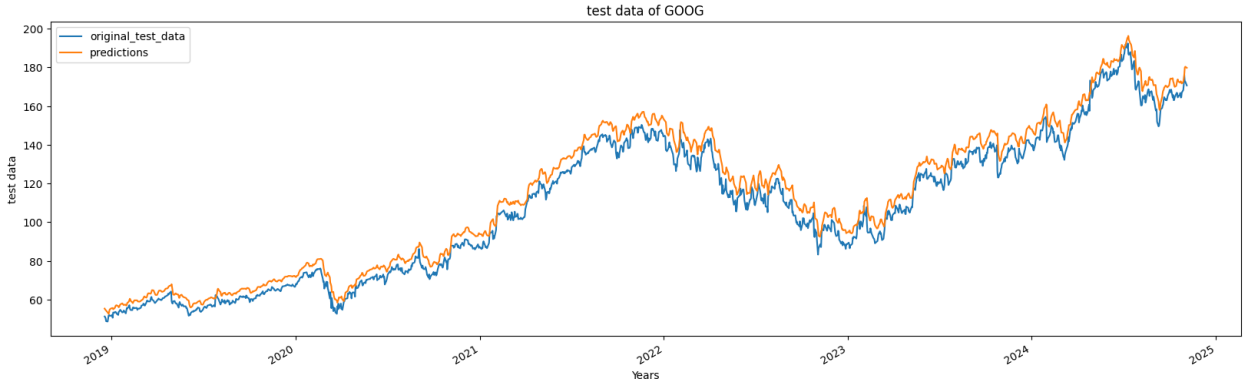
```
rmse = np.sqrt(np.mean( (inv_predictions - inv_y_test)**2))
rmse
```

```
6.783782504920198
```

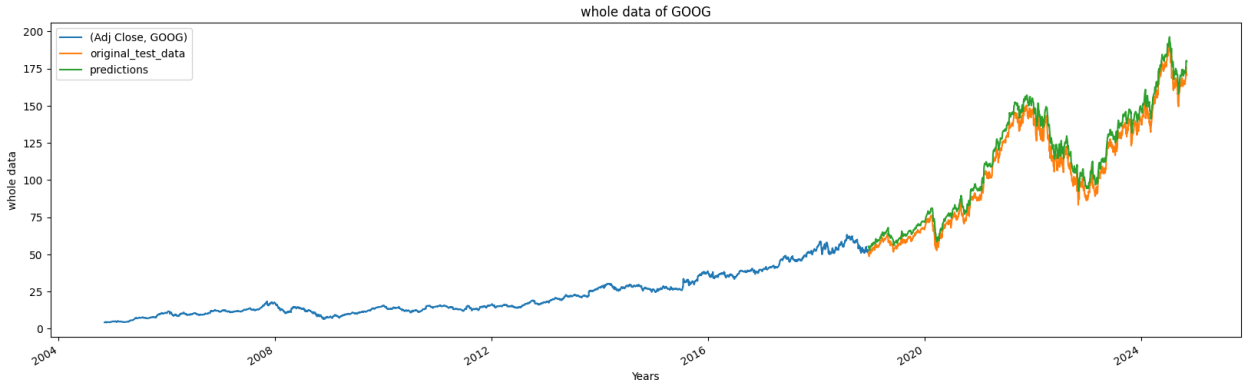
```
ploting_data = pd.DataFrame(
 {
  'original_test_data': inv_y_test.reshape(-1),
    'predictions': inv_predictions.reshape(-1)
 } ,
    index = data.index[splitting_len+100:]
)
ploting_data.head()
```

Date	original_test_data	predictions
2018-12-18 00:00:00+00:00	51.309494	55.393528
2018-12-19 00:00:00+00:00	51.025192	54.960598
2018-12-20 00:00:00+00:00	50.346859	54.703892
2018-12-21 00:00:00+00:00	48.857018	54.317795
2018-12-24 00:00:00+00:00	48.691425	53.409996

```
plot_graph((20,6), ploting_data, 'test data')
```

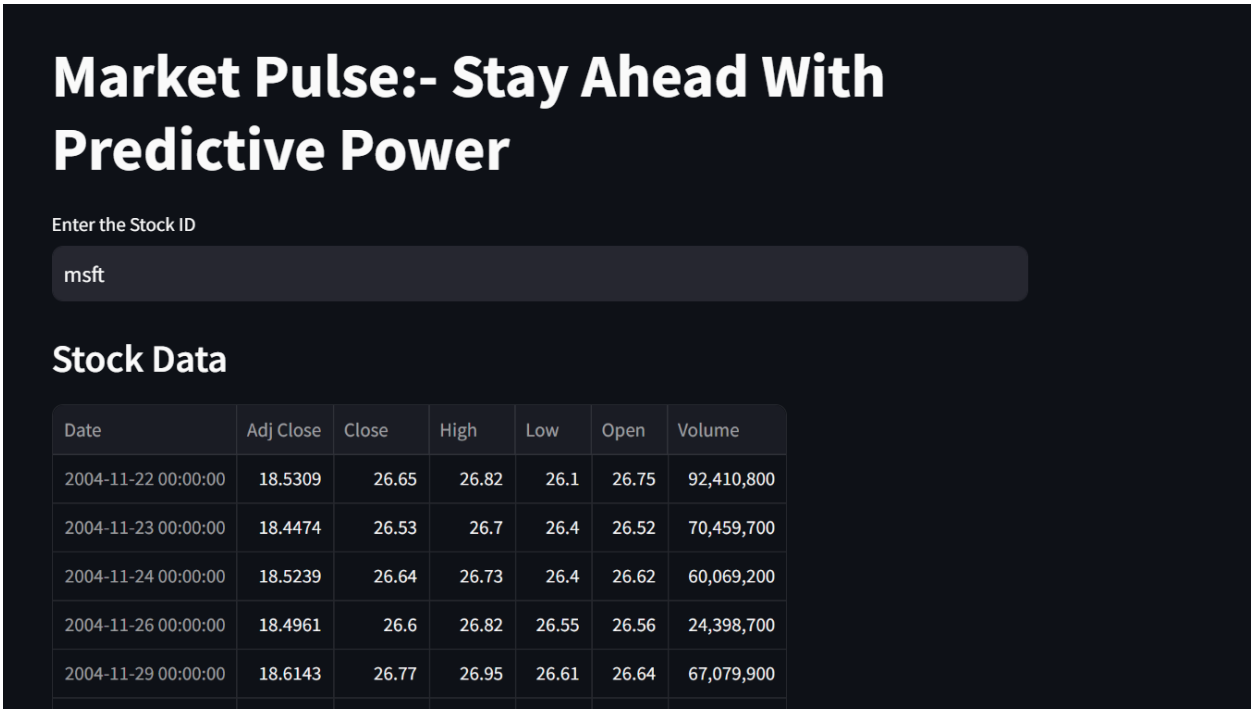



```
plot_graph((20,6),  pd.concat([Adj_close_price[:splitting_len+100],ploting_data],
axis=0), 'whole data')
```

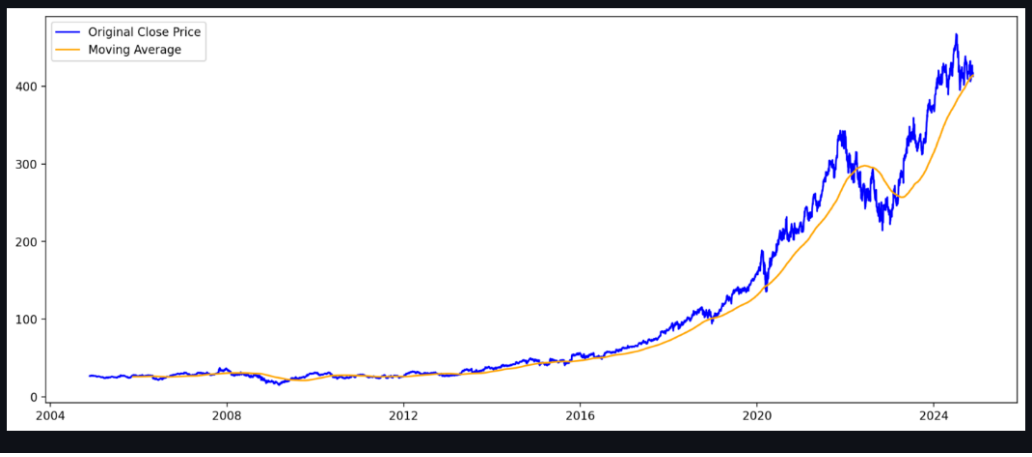


```
model.save("Latest_stock_price_model.keras")
```

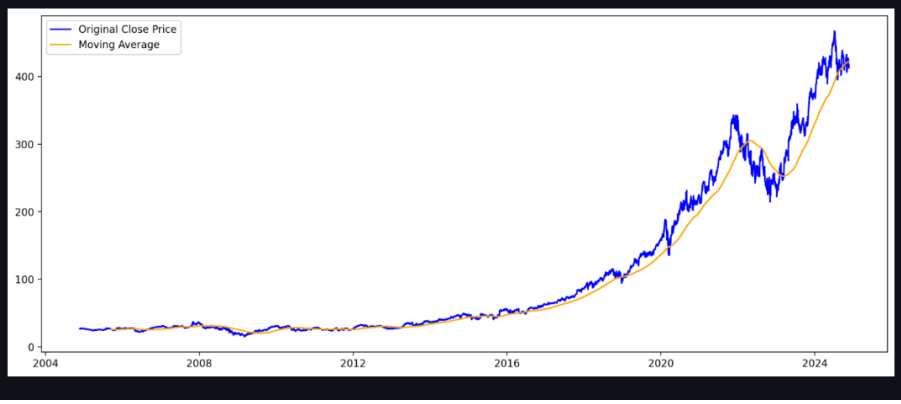
output :



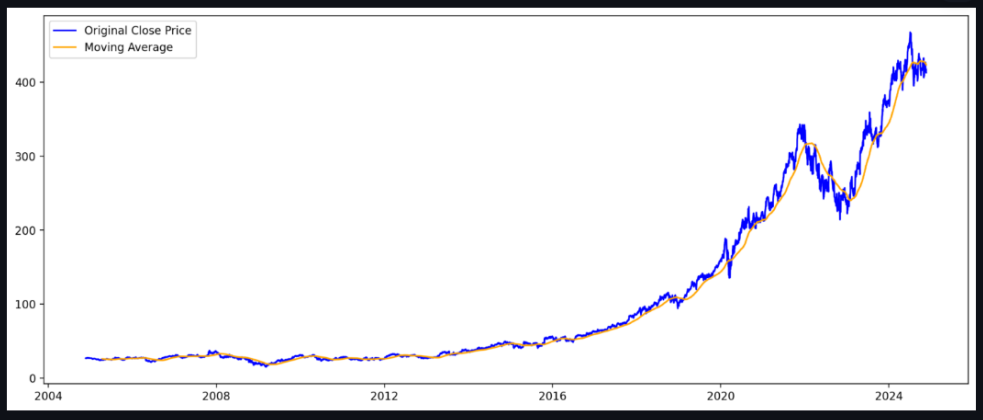
Original Close Price and MA for 250 days



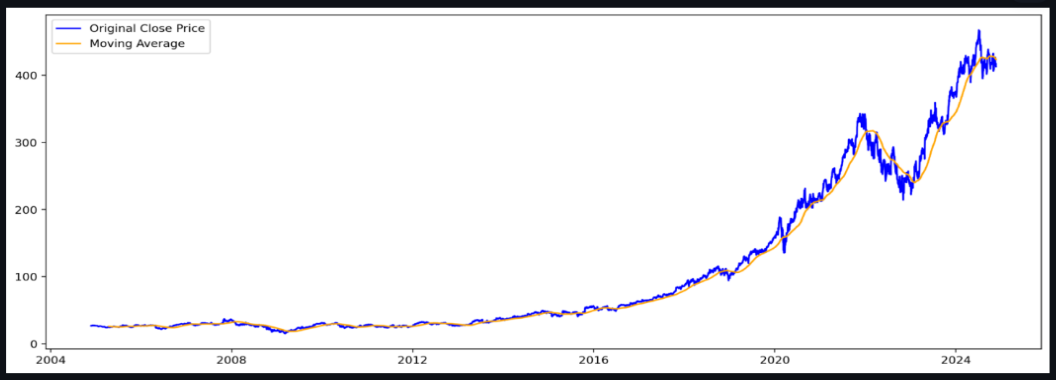
Original Close Price and MA for 200 days



Original Close Price and MA for 100 days



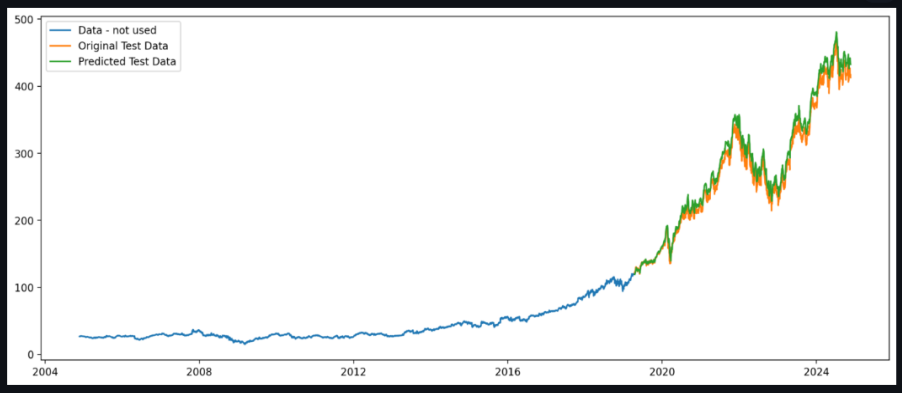
Original Close Price and MA for 100 days



Original values vs Predicted values

Date	Original Test Data	Predictions
2019-04-17 00:00:00	121.77	119.3741
2019-04-18 00:00:00	123.37	119.8093
2019-04-22 00:00:00	123.76	120.8691
2019-04-23 00:00:00	125.44	121.7882
2019-04-24 00:00:00	125.01	123.0637
2019-04-25 00:00:00	129.15	123.6753
2019-04-26 00:00:00	129.89	125.7956
2019-04-29 00:00:00	129.77	127.6564
2019-04-30 00:00:00	130.6	128.6308
2019-05-01 00:00:00	127.88	129.4027

Original Close Price vs Predicted Close price



Results

The model produces predictions that closely align with the actual stock prices. Moving averages

provide a smoothed view of the stock price trends, helping users better understand the overall direction of the stock. Visualizations show that the model effectively captures short-term fluctuations and long-term trends in stock prices.

The **predictions** can be compared to actual test data, which demonstrates how well the model performs in forecasting future stock prices. The model's ability to predict future prices is validated through graphical comparisons, showing that the predicted values closely follow the general pattern of the actual stock prices.

Evaluation

The performance of the model is evaluated using two common regression metrics:

- **Mean Squared Error (MSE)**: This metric measures the average squared difference between predicted and actual values. A lower MSE indicates a better model fit.
- **Root Mean Squared Error (RMSE)**: RMSE is the square root of the MSE, providing a more interpretable metric in terms of the original scale of the data. Again, lower values indicate better model accuracy.

These metrics help assess the model's ability to generalize and make accurate predictions on new, unseen data.

Challenges and Limitations

1. Model Challenges

While RNNs are powerful, they are also highly sensitive to the quality of the data. The model relies heavily on historical data, which may not always account for sudden market shocks, such as economic crises, political events, or natural disasters. Additionally, the model is not able to incorporate external variables like sentiment analysis or social media trends, which can also impact stock prices.

2. System Challenges

The system requires regular updates to stay relevant in dynamic market conditions. Financial data changes constantly, and the model must be retrained periodically with the latest data to ensure its predictions remain accurate. Without continuous updates, the model may become outdated and fail to predict market trends accurately.

Conclusion and Future Work

This project demonstrates the potential of machine learning in stock price prediction, integrating analytics with user-friendly tools. Future enhancements could include:
- Incorporating sentiment analysis for greater accuracy.
- Enabling real-time predictions with live streaming data.

References

1.TensorFlow documentation.
2.Yahoo Finance API documentation.
3. Streamlit and Matplotlib user guides.