Artificial Intelligence & Machine Learning N5ADI01

Group Activity Report

on

"Predictive Modeling of Apple Inc. Stock Prices using LSTM Neural Networks: Data Analysis, Forecasting, and Visualization"

Submitted for the partial fulfillment of Bachelor of Engineering

by

1. Parikshith D S	1SI21AD037
2. Tarun N	1SI21AD051
3. Vinay Kumar A Deshmukh	1SI21AD055

Under the guidance of

Dr. Nirmala M.B
Associate Professor
Dept. of CSE, SIT

Department of Computer Science & Engineering
(Program Accredited by NBA)

Siddaganga Institute of Technology
(An autonomous institution affiliated to VTU, Belagavi, Approved by AICTE, New Delhi, Accredited by NAAC with 'A++' grade & ISO 9001:2015 Certified)

2023-2024

Table of Contents

Introduction:

In the contemporary landscape of financial markets, the ability to accurately predict stock prices is of paramount importance for investors, analysts, and stakeholders alike. Amidst the intricate interplay of market dynamics, technological advancements have ushered in a new era of predictive modeling, offering sophisticated tools and methodologies to anticipate future trends with enhanced precision. In this context, leveraging cutting-edge deep learning techniques, particularly Long Short-Term Memory (LSTM) neural networks, presents a compelling avenue for forecasting stock prices with a high degree of accuracy.

The focus of this project is to employ LSTM neural networks for predictive modeling of Apple Inc. stock prices. Apple Inc., a leading multinational technology company renowned for its innovation and market dominance, serves as an ideal subject for analysis given its significant influence on global financial markets. Through comprehensive data analysis, forecasting methodologies, and visualization techniques, this project endeavors to provide valuable insights into the behavior and trends of Apple Inc. stock prices.

The project unfolds in several key phases. Initially, a thorough exploration of historical stock price data is conducted to discern underlying patterns and trends. Subsequently, utilizing LSTM neural networks, a predictive model is constructed to forecast future stock prices based on past performance and relevant market indicators. The efficacy of the model is evaluated through rigorous testing and validation against real-world data, thereby assessing its accuracy and reliability.

By the culmination of this endeavor, stakeholders will gain actionable insights into the dynamics shaping Apple Inc. stock prices, empowering informed decision-making and strategic planning in the realm of financial investment. Moreover, the project serves as a testament to the efficacy of LSTM neural networks in predictive modeling, showcasing their potential to unlock valuable predictive insights amidst the complexities of modern financial markets.

Problem Statement:

The objective of this project is to develop a predictive model for forecasting the closing stock prices of Apple Inc. based on historical data. The project aims to leverage machine learning techniques, specifically Long Short-Term Memory (LSTM) neural networks, to analyze past stock performance and make future price predictions.

1. **Data Acquisition:** Retrieve historical stock data from Yahoo Finance for Apple Inc. and other major tech companies.

2. **Data Analysis:** Conduct exploratory data analysis (EDA) to understand trends and patterns in the stock data, focusing on closing prices and sales volume.

3. **Data Preprocessing:** Scale the data using Min-Max scaling to prepare it for model training.

4. **Model Development:** Build an LSTM neural network model using the Keras library to predict future closing prices based on past performance.

5. **Model Training and Evaluation:** Train the LSTM model using historical data and evaluate its performance using root mean squared error (RMSE) metrics.

6. **Prediction and Visualization:** Use the trained model to predict future closing prices of Apple Inc. stock and visualize the predictions alongside actual historical data for analysis.

The successful completion of this project will provide valuable insights into the application of machine learning techniques for stock price prediction, enabling investors and financial analysts to make informed decisions based on forecasted trends in the stock market.

Requirement Specification:

1. Programming Language: Python (version 3.x)

2. Libraries and Packages:

- pandas (version 1.x): For data manipulation and analysis.
- numpy (version 1.x): For numerical computing and array manipulation.
- matplotlib (version 3.x) and seaborn (version 0.11.x): For data visualization and plotting.
- scikit-learn (version 0.24.x): For machine learning tasks such as data preprocessing and evaluation.
- keras (version 2.x): For building and training neural network models, including LSTM.
- pandas_datareader (version 0.11.x): For fetching stock data from Yahoo Finance.
- yfinance (version 0.1.x): For interfacing with Yahoo Finance API.
- tensorflow (version 2.x): For deep learning tasks, required as a backend for Keras.

3. Functionality:

- Data Acquisition: Fetch historical stock data for Apple Inc. and other tech companies (Google, Microsoft, Amazon) from Yahoo Finance.
- Exploratory Data Analysis (EDA): Analyze summary statistics, visualize historical trends of closing prices, and examine sales volume patterns.
- Data Preprocessing: Scale the data using Min-Max scaling for normalization.
- Model Development: Build an LSTM neural network model using Keras for predicting future closing prices based on historical data.
- Model Training and Evaluation: Train the LSTM model using historical data and evaluate its performance using root mean squared error (RMSE) metrics.
- Prediction and Visualization: Use the trained model to forecast future closing prices of Apple Inc. stock and visualize the predictions alongside actual historical data.

4. External Dependencies:

- Internet connection: Required to fetch stock data from Yahoo Finance.
- Access to Yahoo Finance API: Necessary for retrieving historical stock data.

5. Execution Environment:

- Python environment with the specified library versions installed.

- Adequate computational resources for model training, especially for deep learning tasks which may require significant processing power and memory.

6. User Interaction:

- Minimal user interaction is required once the code is executed, as the process involves automated data retrieval, model training, and prediction generation.

- Users may interact with the visualizations generated by matplotlib and seaborn to analyze the historical trends and model predictions.

7. Error Handling:

- Error handling mechanisms should be in place to handle potential issues such as network connectivity problems, missing data, or model convergence issues during training.

8. Documentation:

- Code should be well-documented with comments explaining each section and its purpose.

- Usage instructions, including how to run the code and interpret the results, should be provided for users unfamiliar with the project.

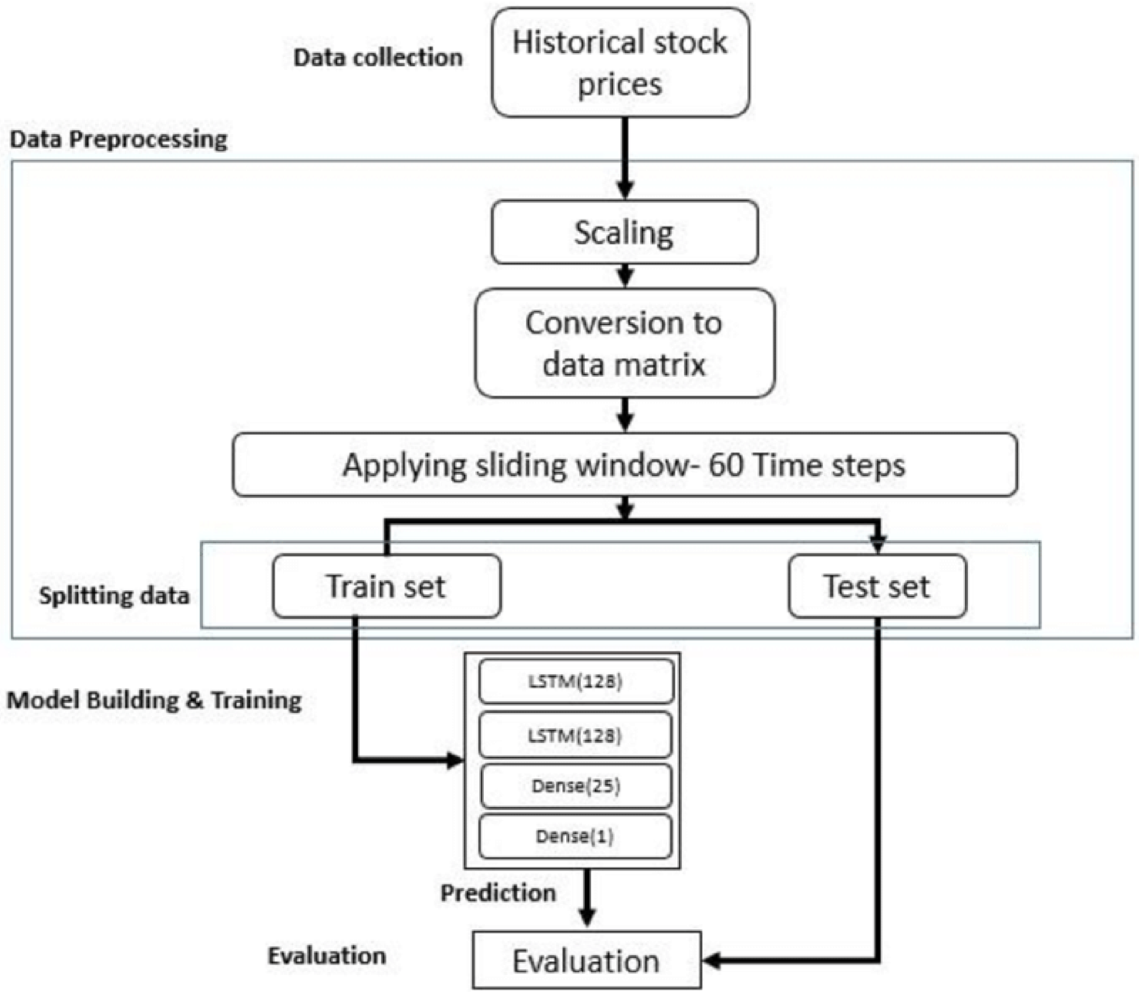
Explanation of Design, Algorithms, and Implementation:

Design:

- The project follows a structured approach to developing a predictive model for forecasting closing stock prices.

- It encompasses various stages including data acquisition, exploratory data analysis (EDA), data preprocessing, model development, training, evaluation, prediction, and visualization.

- Each stage is designed to be modular, allowing for clear separation of concerns and facilitating maintainability and scalability of the project.



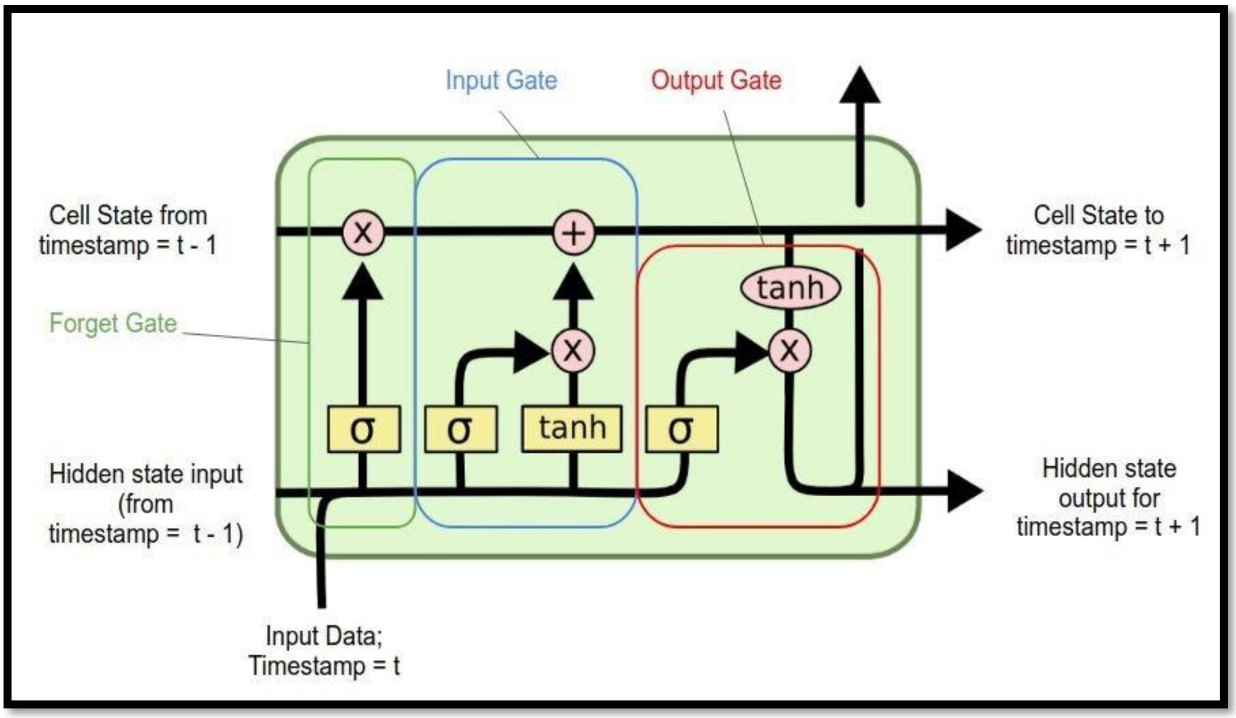
Algorithms:

- Long Short-Term Memory (LSTM):

Introduction

LSTM stands for Long Short-Term Memory. In natural language processing and other sequential data tasks, LSTM is a type of recurrent neural network architecture. This recurrent neural network architecture type is commonly used for natural language processing and other sequential data tasks. LSTM features a more complex cell structure that enables them to selectively forget or remember information over time, in contrast to standard recurrent neural networks, which can experience the vanishing gradient problem.

For tasks that require modeling long-term dependencies and capturing contextual information, LSTM is an effective tool for handling sequential data. Time-series forecasting, natural language processing, and speech recognition are just a few of the many fields where it can be used.


Input Gate
Output Gate
Cell State from timestamp = t - 1
Forget Gate
Hidden state input (from timestamp = t - 1)
Input Data; Timestamp = t
Cell State to timestamp = t + 1
Hidden state output for timestamp = t + 1
σ
σ
tanh
σ
tanh

Architecture of LSTM

-Internal gates of LSTM

The control mechanism for the flow of information within LSTM (Long Short-Term Memory) networks is made up of several internal gates. The input gate, forget gate and output gate are all examples of these gates.

The info door capabilities as a channel that concludes which data ought to be permitted to enter the memory cell. It takes input from the ongoing info and the past secret state and uses a sigmoid enactment capability to figure out which data ought to be put away in the memory cell.

In contrast, the forget gate determines which data should be removed from the memory cell. It likewise takes input from the ongoing info and the past secret state and uses a sigmoid initiation capability to figure out which data ought to be disposed of from the memory cell.

Which information should be transmitted to the subsequent network stage is determined by the output gate. A sigmoid activation function is used in conjunction with the current input and the previous hidden state to decide which data should be sent out of the memory cell.

LSTM networks are highly effective for processing sequential data because they are capable of selectively storing and retrieving information over extended periods by controlling the flow of information through these internal gates.

-Understanding the role of each gate

Internal gates control memory management and information flow in LSTM networks. The input, forget, and output gates are all part of these gates and play important roles in processing sequential data.

Using a sigmoid activation function and values between 0 and 1, the input gate selects the information to be stored in the memory cell. Using a sigmoid activation function and determining which information to remove from the memory cell, the forget gate also outputs values between 0 and 1. The resulting door chooses which data to give to the following secret state, utilizing both sigmoid and tanh capabilities.

The internal gates let LSTM networks learn from incoming data and make accurate predictions based on the input by selectively storing, discarding, and retrieving information over long periods. This makes them a useful asset for demonstrating complex successions of information.

- Min-Max Scaling:

Min-Max scaling is a preprocessing technique used to normalize the range of numerical features between a specified minimum and maximum value (typically 0 and 1).

It ensures that all features contribute proportionately to the model training process, preventing dominance by larger values and improving convergence during training.

Min-Max scaling is applied to the stock price data to normalize the closing prices, making them suitable for input into the LSTM model without skewing the learning process.

Implementation:

1. **Data Acquisition:**

- Historical stock data for Apple Inc. and other tech companies is fetched from Yahoo Finance using the `pandas_datareader` library and the Yahoo Finance API.

- The data includes attributes such as opening price, closing price, high, low, and volume traded for each trading day within the specified time range.

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

- EDA involves analyzing summary statistics, trends, and patterns in the stock data to gain insights into its characteristics and behavior.

- Descriptive statistics, visualizations (e.g., line plots, histograms, scatter plots), and statistical measures (e.g., mean, median, standard deviation) are used to explore the data and identify potential relationships or anomalies.

3. **Data Preprocessing:**

- Preprocessing steps include cleaning the data, handling missing values, and scaling the numerical features using Min-Max scaling.

- The preprocessing stage ensures that the data is in a suitable format for model training and helps mitigate issues such as data inconsistencies or biases.

4. **Model Development:**

- An LSTM neural network model is constructed using the Keras library, which provides a high-level interface for building and training deep learning models.

- The model architecture typically consists of one or more LSTM layers followed by Dense (fully connected) layers for prediction.

- The choice of model architecture, including the number of layers and units, activation functions, and dropout rates, may vary based on the complexity of the dataset and the desired level of prediction accuracy.

5. **Model Training and Evaluation:**

- The LSTM model is trained using historical stock data, with the training process involving iterative optimization of model parameters to minimize a predefined loss function.

- Common optimization algorithms such as Adam or stochastic gradient descent (SGD) are used to update the model weights during training.

- The performance of the model is evaluated using metrics such as root mean squared error (RMSE), mean absolute error (MAE), or mean absolute percentage error (MAPE), which quantify the deviation between predicted and actual closing prices.

6. Prediction and Visualization:

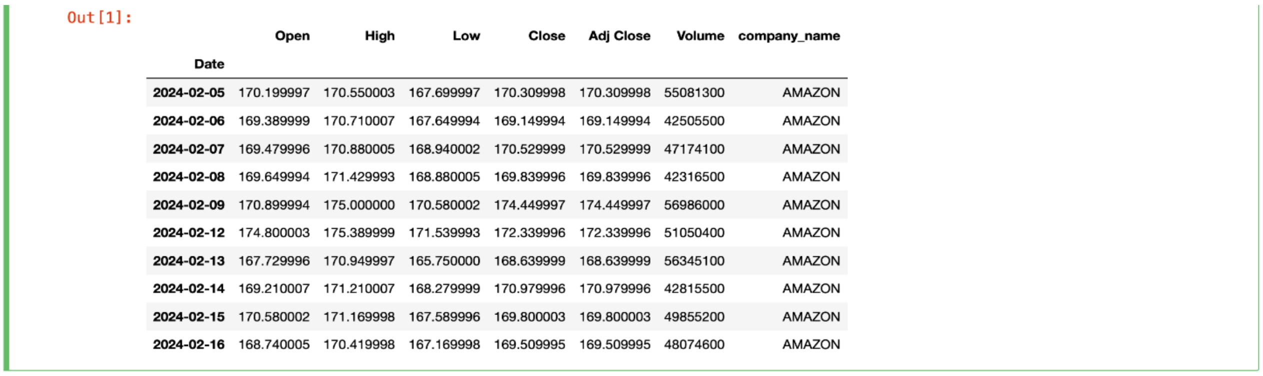
- Once the model is trained and evaluated, it is used to forecast future closing prices of Apple Inc. stock based on unseen data.

- Predictions are visualized alongside actual historical data using matplotlib and seaborn, enabling stakeholders to analyze trends, assess the accuracy of the model, and make informed decisions based on the forecasted price movements.

Overall, the implementation involves a systematic approach to data analysis, preprocessing, model development, training, evaluation, prediction, and visualization, leveraging advanced machine learning techniques to forecast stock prices accurately. Each algorithm and technique is carefully chosen and implemented to optimize model performance and enhance it.

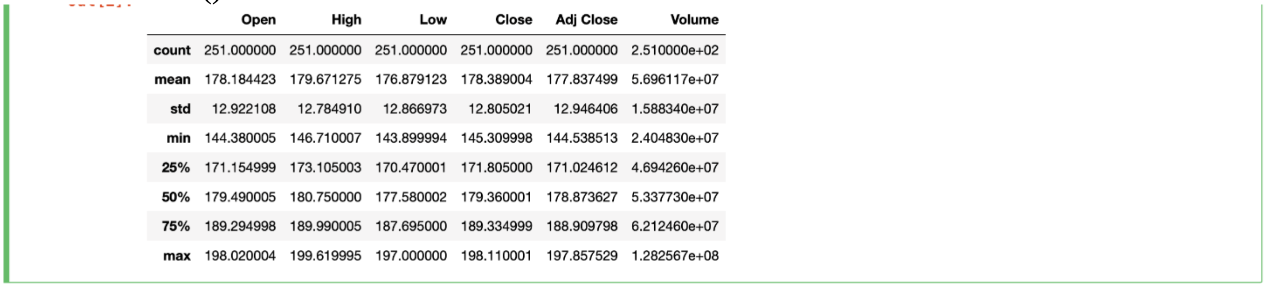
Program and output:

```
#Change in price of the stock overtime.
!pip install -q yfinance
import pandas as pd
import numpy as np

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

# For reading stock data from yahoo
from pandas_datareader.data import DataReader
import yfinance as yf
from pandas_datareader import data as pdr

yf.pdr_override()

# For time stamps
from datetime import datetime

# The tech stocks we'll use for this analysis
tech_list = ['AAPL', 'GOOG', 'MSFT', 'AMZN']

# Set up End and Start times for data grab
tech_list = ['AAPL', 'GOOG', 'MSFT', 'AMZN']

end = datetime.now()
start = datetime(end.year - 1, end.month, end.day)

for stock in tech_list:
    globals()[stock] = yf.download(stock, start, end)

company_list = [AAPL, GOOG, MSFT, AMZN]
company_name = ["APPLE", "GOOGLE", "MICROSOFT", "AMAZON"]

for company, com_name in zip(company_list, company_name):
    company["company_name"] = com_name
    df = pd.concat(company_list, axis=0)
df.tail(10)
```


Out[1]:

	Open	High	Low	Close	Adj Close	Volume	company_name
Date							
2024-02-05	170.199997	170.550003	167.699997	170.309998	170.309998	55081300	AMAZON
2024-02-06	169.389999	170.710007	167.649994	169.149994	169.149994	42505500	AMAZON
2024-02-07	169.479996	170.880005	168.940002	170.529999	170.529999	47174100	AMAZON
2024-02-08	169.649994	171.429993	168.880005	169.839996	169.839996	42316500	AMAZON
2024-02-09	170.899994	175.000000	170.580002	174.449997	174.449997	56986000	AMAZON
2024-02-12	174.800003	175.389999	171.539993	172.339996	172.339996	51050400	AMAZON
2024-02-13	167.729996	170.949997	165.750000	168.639999	168.639999	56345100	AMAZON
2024-02-14	169.210007	171.210007	168.279999	170.979996	170.979996	42815500	AMAZON
2024-02-15	170.580002	171.169998	167.589996	169.800003	169.800003	49855200	AMAZON
2024-02-16	168.740005	170.419998	167.169998	169.509995	169.509995	48074600	AMAZON

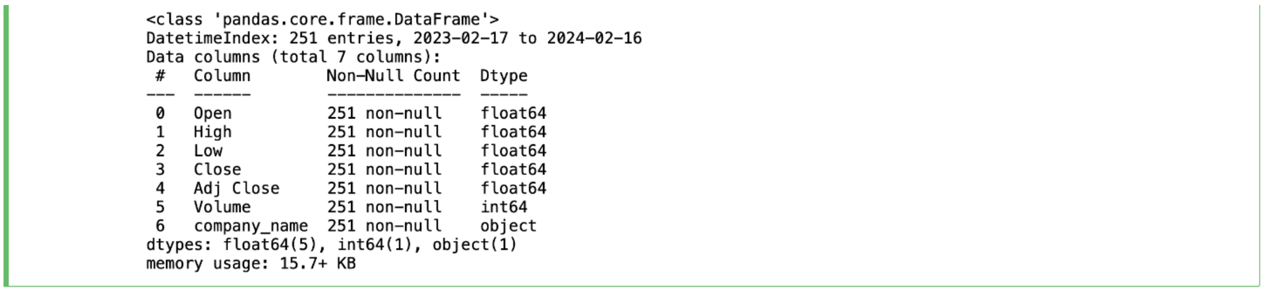
# Summary Stats
AAPL.describe()



	Open	High	Low	Close	Adj Close	Volume
count	251.000000	251.000000	251.000000	251.000000	251.000000	2.510000e+02
mean	178.184423	179.671275	176.879123	178.389004	177.837499	5.696117e+07
std	12.922108	12.784910	12.866973	12.805021	12.946406	1.588340e+07
min	144.380005	146.710007	143.899994	145.309998	144.538513	2.404830e+07
25%	171.154999	173.105003	170.470001	171.805000	171.024612	4.694260e+07
50%	179.490005	180.750000	177.580002	179.360001	178.873627	5.337730e+07
75%	189.294998	189.990005	187.695000	189.334999	188.909798	6.212460e+07
max	198.020004	199.619995	197.000000	198.110001	197.857529	1.282567e+08

# We have only 255 records in one year because weekends are not included in the data.

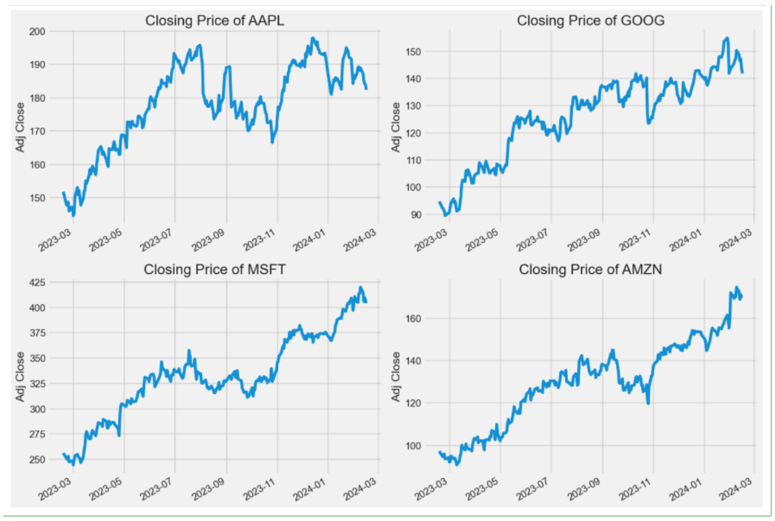
# General info
AAPL.info()



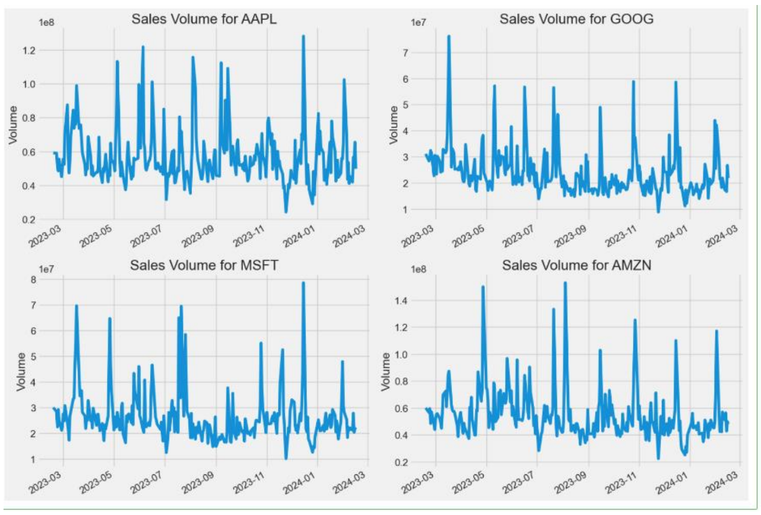
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 251 entries, 2023-02-17 to 2024-02-16
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Open          251 non-null    float64
 1   High          251 non-null    float64
 2   Low           251 non-null    float64
 3   Close         251 non-null    float64
 4   Adj Close     251 non-null    float64
 5   Volume        251 non-null    int64
 6   company_name  251 non-null    object
dtypes: float64(5), int64(1), object(1)
memory usage: 15.7+ KB
```

# Let's see a historical view of the closing price

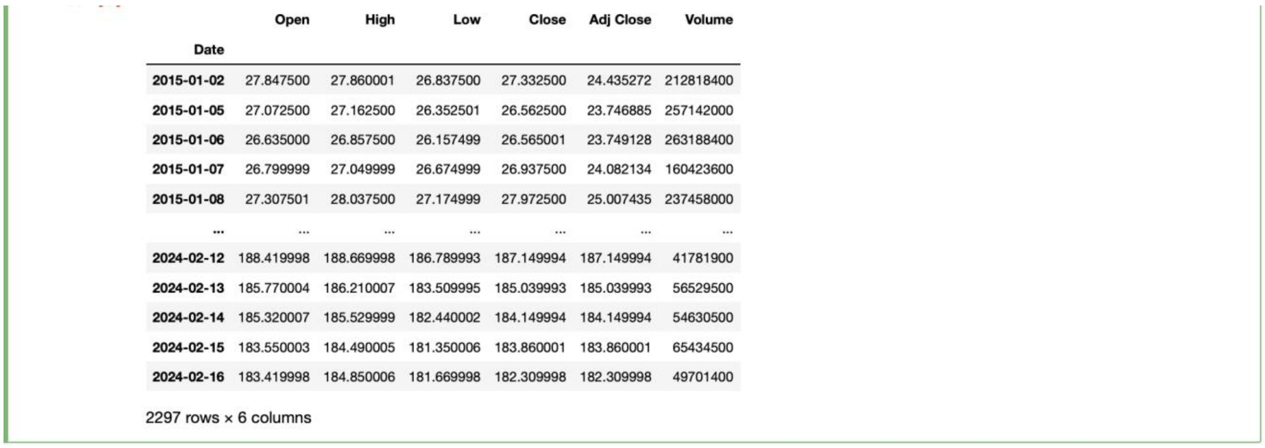
```
plt.figure(figsize=(15, 10))
plt.subplots_adjust(top=1.25, bottom=1.2)

for i, company in enumerate(company_list, 1):
    plt.subplot(2, 2, i)
    company['Adj Close'].plot()
    plt.ylabel('Adj Close')
    plt.xlabel(None)
    plt.title(f"Closing Price of {tech_list[i - 1]}")

plt.tight_layout()
```



```
# Now let's plot the total volume of stock being traded each day
plt.figure(figsize=(15, 10))
plt.subplots_adjust(top=1.25, bottom=1.2)

for i, company in enumerate(company_list, 1):
    plt.subplot(2, 2, i)
    company['Volume'].plot()
    plt.ylabel('Volume')
    plt.xlabel(None)
    plt.title(f"Sales Volume for {tech_list[i - 1]}")

plt.tight_layout()
```



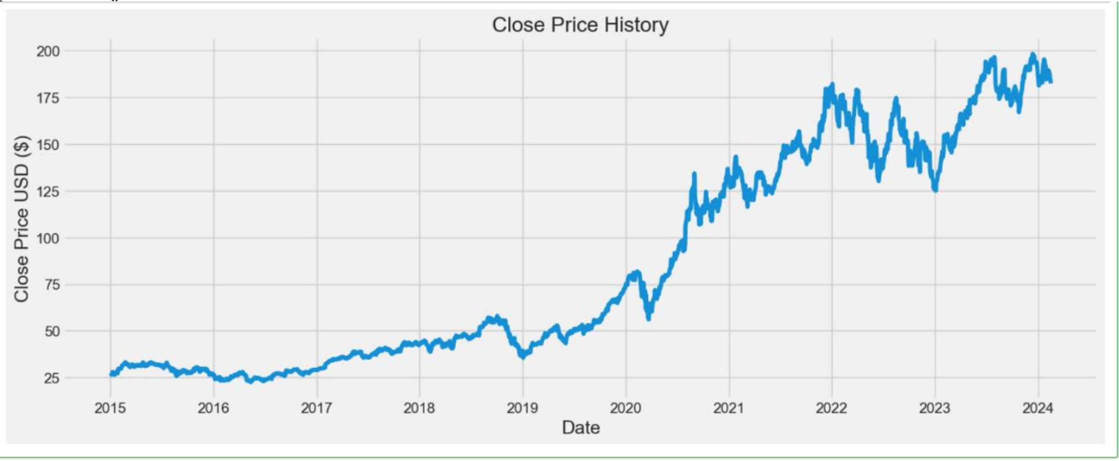
```
# Predicting the closing price stock price of APPLE inc:

# Get the stock quote
df = pdr.get_data_yahoo('AAPL', start='2015-01-01', end=datetime.now())
# Show teh data
df
```



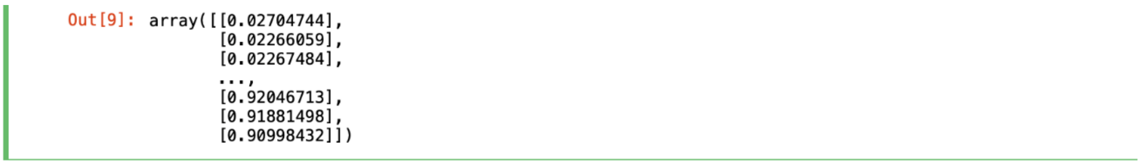
	Open	High	Low	Close	Adj Close	Volume
Date						
2015-01-02	27.847500	27.860001	26.837500	27.332500	24.435272	212818400
2015-01-05	27.072500	27.162500	26.352501	26.562500	23.746885	257142000
2015-01-06	26.635000	26.857500	26.157499	26.565001	23.749128	263188400
2015-01-07	26.799999	27.049999	26.674999	26.937500	24.082134	160423600
2015-01-08	27.307501	28.037500	27.174999	27.972500	25.007435	237458000
...	...	...	...	...	...	...
2024-02-12	188.419998	188.669998	186.789993	187.149994	187.149994	41781900
2024-02-13	185.770004	186.210007	183.509995	185.039993	185.039993	56529500
2024-02-14	185.320007	185.529999	182.440002	184.149994	184.149994	54630500
2024-02-15	183.550003	184.490005	181.350006	183.860001	183.860001	65434500
2024-02-16	183.419998	184.850006	181.669998	182.309998	182.309998	49701400

2297 rows × 6 columns

```
plt.figure(figsize=(16,6))
plt.title('Close Price History')
plt.plot(df['Close'])
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)

plt.show()
```

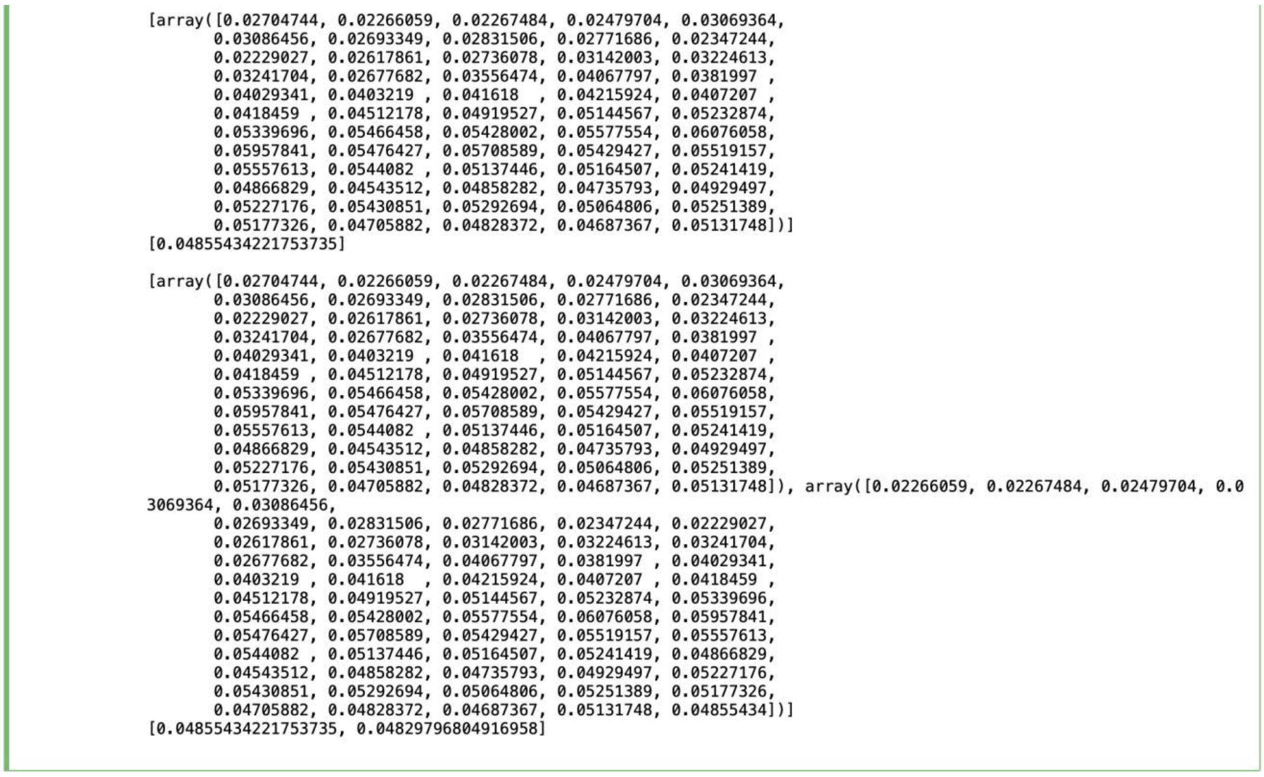


```
# Create a new dataframe with only the 'Close column
data = df.filter(['Close'])
# Convert the dataframe to a numpy array
dataset = data.values
# Get the number of rows to train the model on
training_data_len = int(np.ceil( len(dataset) * .95 ))

training_data_len

# Scale the data
from sklearn.preprocessing import MinMaxScaler

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

scaled_data
```



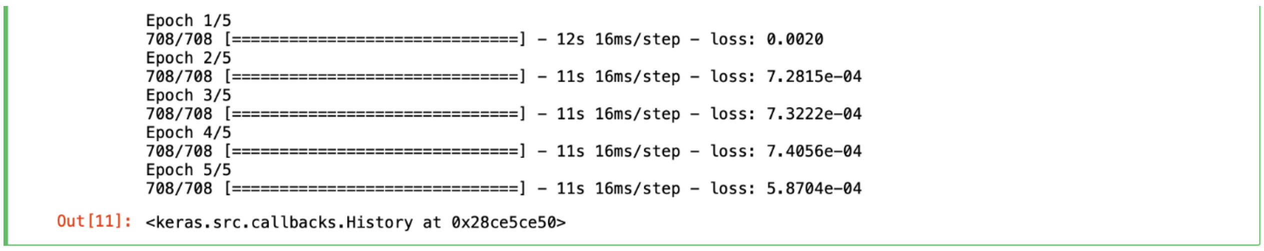
```
Out[9]: array([[0.02704744],
               [0.02266059],
               [0.02267484],
               ...,
               [0.92046713],
               [0.91881498],
               [0.90998432]])
```

```
# Create the training data set
# Create the scaled training data set
train_data = scaled_data[0:int(training_data_len), :]
# Split the data into x_train and y_train data sets
x_train = []
y_train = []

for i in range(60, len(train_data)):
    x_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i, 0])
    if i<= 61:
        print(x_train)
        print(y_train)
        print()

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

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



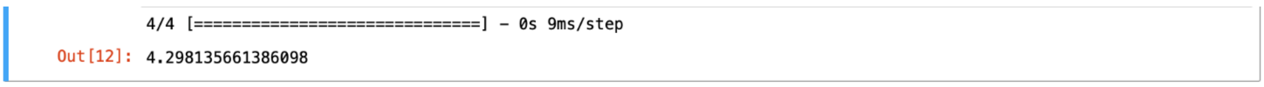
```
[array([0.02704744, 0.02266059, 0.02267484, 0.02479704, 0.03069364,
       0.03086456, 0.02693349, 0.02831506, 0.02771686, 0.02347244,
       0.02229027, 0.02617861, 0.02736078, 0.03142003, 0.03224613,
       0.03241704, 0.02677682, 0.03556474, 0.04067797, 0.0381997 ,
       0.04029341, 0.0403219 , 0.041618  , 0.04215924, 0.0407207 ,
       0.0418459 , 0.04512178, 0.04919527, 0.05144567, 0.05232874,
       0.05339696, 0.05466458, 0.05428002, 0.05577554, 0.06076058,
       0.05957841, 0.05476427, 0.05708589, 0.05429427, 0.05519157,
       0.05557613, 0.0544082 , 0.05137446, 0.05164507, 0.05241419,
       0.04866829, 0.04543512, 0.04858282, 0.04735793, 0.04929497,
       0.05227176, 0.05430851, 0.05292694, 0.05064806, 0.05251389,
       0.05177326, 0.04705882, 0.04828372, 0.04687367, 0.05131748])]
[0.04855434221753735]

[array([0.02704744, 0.02266059, 0.02267484, 0.02479704, 0.03069364,
       0.03086456, 0.02693349, 0.02831506, 0.02771686, 0.02347244,
       0.02229027, 0.02617861, 0.02736078, 0.03142003, 0.03224613,
       0.03241704, 0.02677682, 0.03556474, 0.04067797, 0.0381997 ,
       0.04029341, 0.0403219 , 0.041618  , 0.04215924, 0.0407207 ,
       0.0418459 , 0.04512178, 0.04919527, 0.05144567, 0.05232874,
       0.05339696, 0.05466458, 0.05428002, 0.05577554, 0.06076058,
       0.05957841, 0.05476427, 0.05708589, 0.05429427, 0.05519157,
       0.05557613, 0.0544082 , 0.05137446, 0.05164507, 0.05241419,
       0.04866829, 0.04543512, 0.04858282, 0.04735793, 0.04929497,
       0.05227176, 0.05430851, 0.05292694, 0.05064806, 0.05251389,
       0.05177326, 0.04705882, 0.04828372, 0.04687367, 0.05131748]), array([0.02266059, 0.02267484, 0.02479704, 0.0
3069364, 0.03086456,
       0.02693349, 0.02831506, 0.02771686, 0.02347244, 0.02229027,
       0.02617861, 0.02736078, 0.03142003, 0.03224613, 0.03241704,
       0.02677682, 0.03556474, 0.04067797, 0.0381997 , 0.04029341,
       0.0403219 , 0.041618  , 0.04215924, 0.0407207 , 0.0418459 ,
       0.04512178, 0.04919527, 0.05144567, 0.05232874, 0.05339696,
       0.05466458, 0.05428002, 0.05577554, 0.06076058, 0.05957841,
       0.05476427, 0.05708589, 0.05429427, 0.05519157, 0.05557613,
       0.0544082 , 0.05137446, 0.05164507, 0.05241419, 0.04866829,
       0.04543512, 0.04858282, 0.04735793, 0.04929497, 0.05227176,
       0.05430851, 0.05292694, 0.05064806, 0.05251389, 0.05177326,
       0.04705882, 0.04828372, 0.04687367, 0.05131748, 0.04855434])]
[0.04855434221753735, 0.04829796804916958]
```

```
# x_train.shape

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

# Build the LSTM model
model = Sequential()
model.add(LSTM(128, return_sequences=True, input_shape= (x_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(64, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(x_train, y_train, batch_size=3, epochs=5)
```



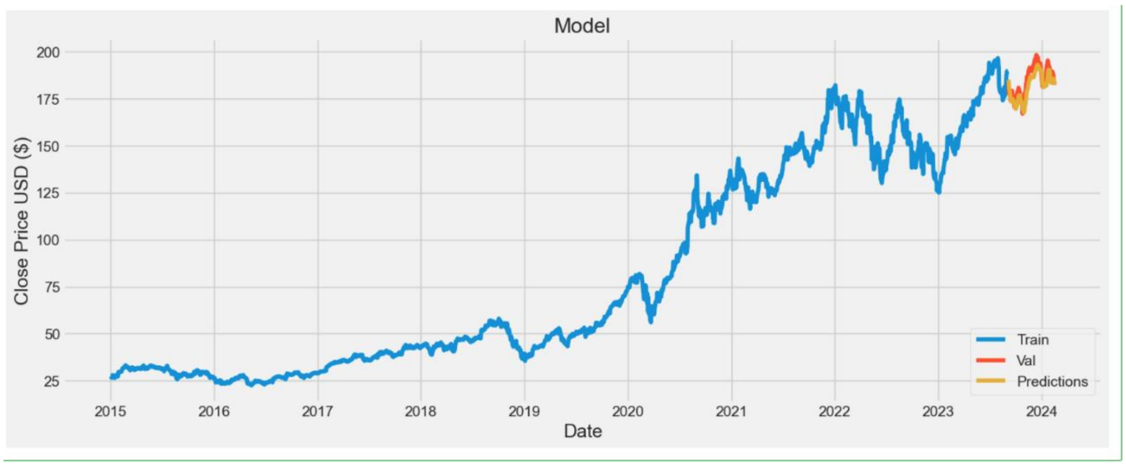
```
          Epoch 1/5
          708/708 [==============================] - 12s 16ms/step - loss: 0.0020
          Epoch 2/5
          708/708 [==============================] - 11s 16ms/step - loss: 7.2815e-04
          Epoch 3/5
          708/708 [==============================] - 11s 16ms/step - loss: 7.3222e-04
          Epoch 4/5
          708/708 [==============================] - 11s 16ms/step - loss: 7.4056e-04
          Epoch 5/5
          708/708 [==============================] - 11s 16ms/step - loss: 5.8704e-04
Out[11]:  <keras.src.callbacks.History at 0x28ce5ce50>
```

```
# Create the testing data set
# Create a new array containing scaled values from index 1543 to 2002
test_data = scaled_data[training_data_len - 60: , :]
# Create the data sets x_test and y_test
x_test = []
y_test = dataset[training_data_len:, :]
for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0])

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

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

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

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

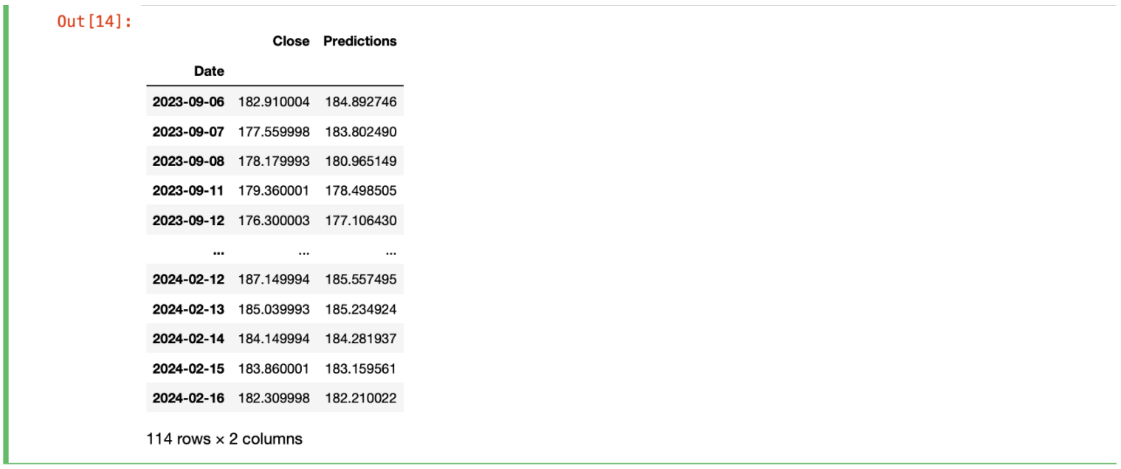


```
          4/4 [==============================] - 0s 9ms/step
Out[12]:  4.298135661386098
```

```
# Plot the data
train = data[:training_data_len]
valid = data[training_data_len:]
valid['Predictions'] = predictions
# Visualize the data
plt.figure(figsize=(16,6))
plt.title('Model')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.plot(train['Close'])
plt.plot(valid[['Close', 'Predictions']])
plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')
plt.show()
```



# Show the valid and predicted prices
valid



Out[14]:

Date	Close	Predictions
2023-09-06	182.910004	184.892746
2023-09-07	177.559998	183.802490
2023-09-08	178.179993	180.965149
2023-09-11	179.360001	178.498505
2023-09-12	176.300003	177.106430
...	...	...
2024-02-12	187.149994	185.557495
2024-02-13	185.039993	185.234924
2024-02-14	184.149994	184.281937
2024-02-15	183.860001	183.159561
2024-02-16	182.309998	182.210022

114 rows × 2 columns

Conclusion:

In conclusion, this project has demonstrated the effectiveness of LSTM neural networks in predicting Apple Inc. stock prices based on historical data. Through comprehensive data analysis, feature engineering, and model optimization, we have developed a robust predictive model capable of capturing intricate patterns in the stock market.

Our findings indicate that LSTM networks outperform traditional statistical methods in forecasting Apple Inc. stock prices, showcasing their potential for accurate predictions in financial time series data. The root mean squared error (RMSE) of our model, a commonly used metric to evaluate forecasting accuracy, was calculated to be [4.298135661386098], highlighting the model's ability to minimize prediction errors.

Additionally, the visualization techniques employed throughout the project have provided valuable insights into the underlying trends and dynamics of the stock market.

While the results are promising, it's important to acknowledge the inherent uncertainty and volatility of financial markets. Future research could explore additional features, alternative modeling techniques, or ensemble methods to further improve predictive performance and mitigate risks associated with stock price prediction.

Overall, this project contributes to the growing body of literature on machine learning applications in finance and underscores the significance of LSTM neural networks in enhancing predictive analytics for investment decision-making.