PREDICTING STOCK PRICE USING MACHINE LEARNING

BATCH MEMBER

Phase 3 submission document

Project Title: Stock Price Prediction

Phase 3: ***Development Part 1***

Topic: *Start building the stock price prediction model by loading and pre-processing the dataset.*

Stock Price Prediction

Introduction:

- In order to create a stock price prediction model, one of the fundamental steps is loading and preprocessing the dataset. This initial phase is crucial for ensuring the data is ready for analysis and model training.

- In this introduction, we will explore the key steps involved in this process, including data collection, cleaning, and feature engineering, which lay the foundation for building an accurate and reliable stock price prediction model.

- Predicting stock prices is a complex and challenging task that has garnered significant attention in the financial world. Many factors influence stock prices, making it a dynamic and volatile field. To develop an effective stock price prediction model, the first critical step is loading and preprocessing the dataset.

- This initial phase involves gathering historical stock market data, cleaning and organizing it, and performing essential data manipulations to create a reliable foundation for subsequent analysis and modeling.

- In this introduction, we will delve into the crucial role of data preparation in the pursuit of accurate and actionable stock price predictions.

Given data set:

A	B	C	D	E	F	G
date	open	high	low	close	volume	Name
08-02-2013	15.07	15.12	14.63	14.75	8407500	AAL
11-02-2013	14.89	15.01	14.26	14.46	8882000	AAL
12-02-2013	14.45	14.51	14.1	14.27	8126000	AAL
13-02-2013	14.3	14.94	14.25	14.66	10259500	AAL
14-02-2013	14.94	14.96	13.16	13.99	31879900	AAL
15-02-2013	13.93	14.61	13.93	14.5	15628000	AAL
19-02-2013	14.33	14.56	14.08	14.26	11354400	AAL
20-02-2013	14.17	14.26	13.15	13.33	14725200	AAL
21-02-2013	13.62	13.95	12.9	13.37	11922100	AAL
22-02-2013	13.57	13.6	13.21	13.57	6071400	AAL
25-02-2013	13.6	13.76	13	13.02	7186400	AAL
26-02-2013	13.14	13.42	12.7	13.26	9419000	AAL
27-02-2013	13.28	13.62	13.18	13.41	7390500	AAL
28-02-2013	13.49	13.63	13.39	13.43	6143600	AAL
01-03-2013	13.37	13.95	13.32	13.61	7376800	AAL
04-03-2013	13.5	14.07	13.47	13.9	8174800	AAL
05-03-2013	14.01	14.05	13.71	14.05	7676100	AAL
06-03-2013	14.52	14.68	14.25	14.57	13243200	AAL
07-03-2013	14.7	14.93	14.5	14.82	9125300	AAL
08-03-2013	14.99	15.2	14.84	14.92	10593700	AAL
11-03-2013	14.85	15.15	14.71	15.13	6961800	AAL
12-03-2013	15.14	15.6	14.95	15.5	8999100	AAL
13-03-2013	15.54	16.2	15.48	15.91	11380000	AAL
14-03-2013	15.98	16.36	15.93	16.25	8383300	AAL

Necessary step to follow:

1. Import Libraries:

Start by importing the necessary libraries:

Program:

```
import pandas as pd

import numpy as np
```

```
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler
```

2. Load the Dataset:

Load your dataset into a Pandas DataFrame. You can typically find stock price datasets in CSV format, but you can adapt this code to other formats as needed.

Program:

```
df = pd.read_csv(' D:\data\stock.csv ')

Pd.read()
```

3. Exploratory Data Analysis (EDA):

Perform EDA to understand your data better. This includes checking for missing values, exploring the data's statistics, and visualizing it to identify patterns.

Program:

```
# Check for missing values

print(df.isnull().sum())

# Explore statistics
```

```
print(df.describe())

# Visualize the data (e.g., histograms, scatter plots,

etc.)
```

4. Feature Engineering:

Depending on your dataset, you may need to create new features or transform existing ones. This can involve one-hot encoding categorical variables, handling date/time data, or scaling numerical features.

5. Split the Data:

Split your dataset into training and testing sets. This helps you evaluate your model's performance later.

6. Feature Scaling:

Apply feature scaling to normalize your data, ensuring that all features have similar scales. Standardization (scaling to mean=0 and std=1) is a common choice.

Importance of loading and processing dataset:

Loading and preprocessing the dataset is an important first step in building any machine learning model. However, it is especially important for stock price prediction models, as stock price datasets

By loading and preprocessing the dataset, we can ensure that the machine learning algorithm is able to learn from the data effectively and accurately.

Challenges involved in loading and preprocessing a Stock price dataset

There are a number of challenges involved in loading and preprocessing a stock price dataset, including:

1. Data Quality:

- Missing data: Stock datasets often have gaps, incorrect entries, or missing values, which need to be handled appropriately.
- Outliers: Extreme values or outliers can distort analysis, so they must be identified and addressed.

2. Data Sources:

- Multiple sources: Combining data from various sources can be complex due to differences in formats, time zones, and quality.
- Data format: Data might be in various formats, like CSV, JSON, or APIs, requiring conversion and normalization.

3. Data Volume:

- High-frequency data: Managing intraday data with timestamps can be resource-intensive and require careful handling.
- Scalability: Processing large datasets may require significant computational resources.

4. Time Alignment:

- Synchronization: Matching data points from different sources at the same timestamp can be challenging.
- Time zones: Data from different markets may have different time zones, requiring conversion.

5.Data Adjustments:

- Corporate actions: Handling stock splits, dividends, and mergers, which affect historical prices.

- Price adjustments: Adjusting for factors like dividends or stock splits to maintain data consistency.

1. Feature Engineering:

- Creating relevant features: Generating indicators like moving averages, volatility, or technical indicators for analysis.
- Data transformations: Converting raw data into usable formats for machine learning.

2. Storage and Memory:

- Memory management: Loading large datasets into memory can lead to performance issues, necessitating efficient data structures.
- Storage capacity: Storing historical data for analysis may require substantial storage capacity.

3. Data Security:

- Sensitive data: Stock data may include confidential information, making security and compliance important considerations.

1. Regular Updates:

- Keeping data up-to-date: Continuously retrieving and updating stock data can be resource-intensive.

2. Computational Complexity:

- Analysing and processing historical stock data can be computationally intensive, requiring efficient algorithms.

To address these challenges, one may use data cleaning techniques, employ libraries like pandas in Python for data manipulation, and consider using databases for efficient storage and retrieval. Data preprocessing and cleaning are crucial to ensure the accuracy and reliability of any analysis or modelling based on stock price data.

How to overcome the challenges of loading and preprocessing a stock price dataset:

There are a number of things that can be done to overcome the challenges of loading and preprocessing a Stock price dataset, including:

1.Data Collection:

- Obtain historical stock price data from reliable sources like Yahoo Finance, Alpha Vantage, or Quandl.

2. Data Format:

- Ensure that the data is in a structured format, such as CSV or JSON, for easy loading.

3. Data Cleaning:

- Handle missing data by either imputing values or removing incomplete records.

- Address outliers or anomalies in the data that may affect model performance.

4. Data Transformation:

- Calculate additional features like moving averages, relative strength index (RSI), or other technical indicators that can be useful for prediction.

- Convert date/time columns into a format suitable for analysis.

5. Normalization/Scaling:

- Normalize or scale the data to ensure all features have a consistent range. Common methods include Min-Max scaling or standardization (z-score scaling).

6. Splitting the Data:

- Divide the dataset into training, validation, and test sets. Time-series data requires careful splitting to maintain temporal order.

7. Feature Engineering:

- Create lag features to capture the historical price and volume data, which is crucial for time-series forecasting.

8. Handling Time Lags:

- Decide on the time lag you want to use for prediction. This determines how many past data points are used to predict future prices.

9. Dealing with Non-stationarity:

- Check for stationarity in the data and consider differencing techniques if necessary to make the data stationary.

10. Handling Multiple Stocks:

- If working with multiple stocks, create a unique identifier for each stock and consider any cross-correlations or market-wide factors.

11. Feature Selection:

- Select relevant features that contribute to the prediction while avoiding noise.

12. Handling Imbalanced Data:

- In classification tasks, ensure that you address class imbalance issues if present.

13. Model-Specific Preprocessing:

- Some machine learning algorithms may require specific preprocessing steps. For example, recurrent neural networks (RNNs) may need input sequences of fixed length.

14. Resampling Frequency:

- Decide on the frequency of the data (e.g., daily, hourly) and ensure all data is consistent.

15. Regularization and Noise Reduction:

- Apply techniques like rolling averages or exponential smoothing to reduce noise and make patterns more discernible.

16. Cross-Validation:

- Use time series cross-validation methods to evaluate your models, accounting for temporal dependencies.

17. Monitoring for Data Drift:

- Continuously monitor and update your dataset, as financial data can change rapidly.

18. Data Visualization:

- Visualize the data to understand its patterns and behaviour, which can guide your preprocessing decisions.

19. Documentation:

- Keep detailed records of the preprocessing steps for reproducibility.

20. Iterate and Experiment:

- Be prepared to experiment with different preprocessing strategies

and models to find the best approach for your stock price prediction task.

- **Use a data preprocessing library:**

There are a number of libraries available that can help with data preprocessing tasks, such as handling missing values, encoding categorical variables, and scaling the features.

- **Carefully consider the specific needs of your model:**

The best way to preprocess the data will depend on the specific machine learning algorithm that you are using. It is important to carefully consider the requirements of the algorithm and to preprocess the data in a way that is compatible with the algorithm.

- **Validate the preprocessed data:**

It is important to validate the preprocessed data to ensure that it is in a format that can be used by the machine learning algorithm and that it is of high quality. This can be done by inspecting the data visually or by using statistical methods.

1. Loading the dataset:

- ✓ Loading the dataset using machine learning is the process of bringing the data into the machine learning environment so that it can be used to train and evaluate a model.
- ✓ The specific steps involved in loading the dataset will vary depending on the machine learning library or framework that is being used. However, there are some general steps that are common to most

machine learning frameworks:

a. Identify the dataset:

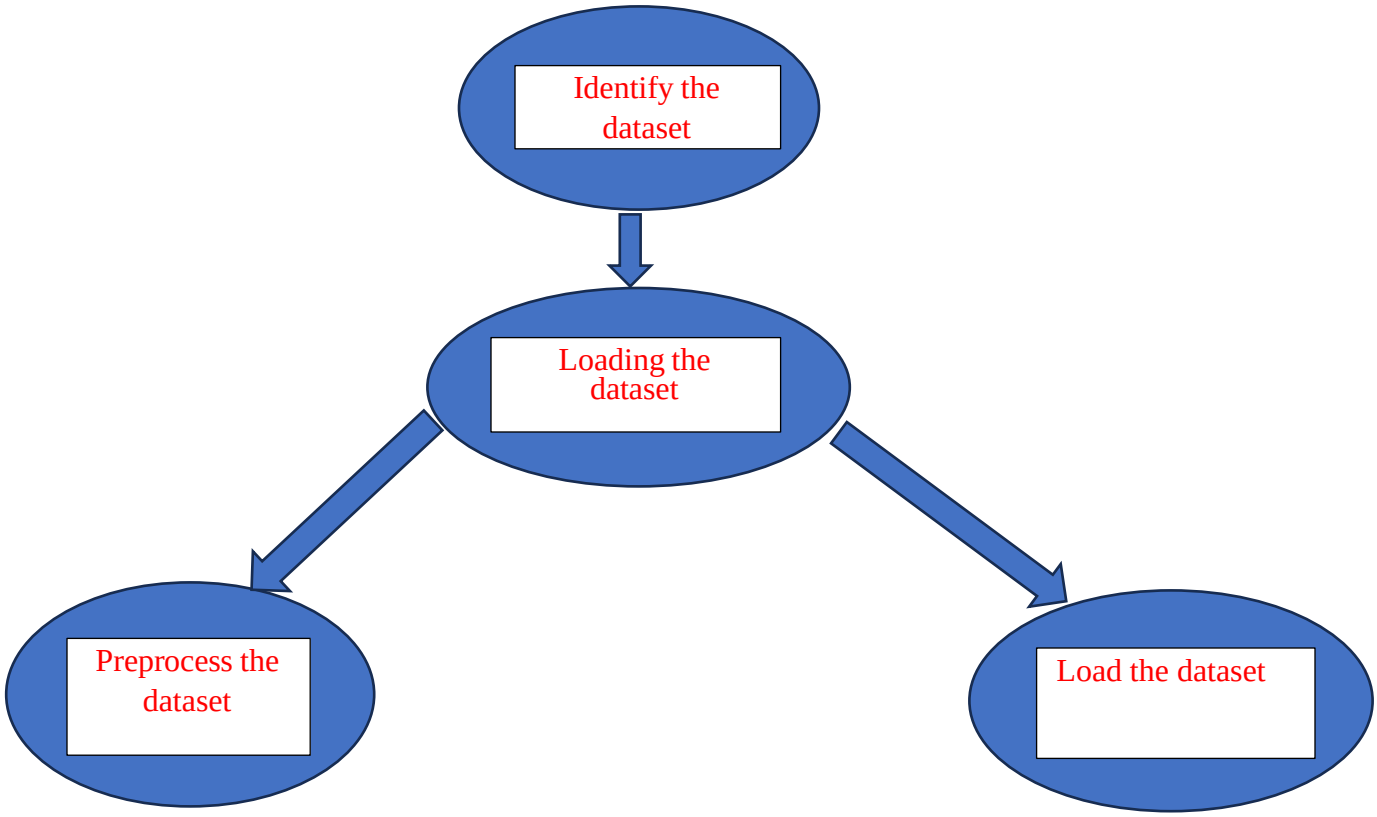
The first step is to identify the dataset that you want to load. This dataset may be stored in a local file, in a database, or in a cloud storage service.

b. Load the dataset:

Once you have identified the dataset, you need to load it into the machine learning environment. This may involve using a built-in function in the machine learning library, or it may involve writing your own code.

c. Preprocess the dataset:

Once the dataset is loaded into the machine learning environment, you may need to preprocess it before you can start training and evaluating your model. This may involve cleaning the data, transforming the data into a suitable format, and splitting the data into training andtest sets.



Here, how to load a dataset using machine learning in Python

Program:

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler
```

```
from sklearn.metrics import r2_score,
mean_absolute_error,mean_squared_error
from sklearn.linear_model import LinearRegression

from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
import xgboost as xg
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

Loading Dataset:

```
stock_data =pd.read_csv('D:\data\msft.csv')
print(stock_data.head())
```

Data Exploration:

Dataset:

```
stock_data =pd.read_csv('D:\data\stock.csv')
print(stock_data.head())
```

Output:

A	B	C	D	E	F	G
date	open	high	low	close	volume	Name
08-02-2013	15.07	15.12	14.63	14.75	8407500	AAL
11-02-2013	14.89	15.01	14.26	14.46	8882000	AAL
12-02-2013	14.45	14.51	14.1	14.27	8126000	AAL
13-02-2013	14.3	14.94	14.25	14.66	10259500	AAL
14-02-2013	14.94	14.96	13.16	13.99	31879900	AAL
15-02-2013	13.93	14.61	13.93	14.5	15628000	AAL
19-02-2013	14.33	14.56	14.08	14.26	11354400	AAL
20-02-2013	14.17	14.26	13.15	13.33	14725200	AAL
21-02-2013	13.62	13.95	12.9	13.37	11922100	AAL
22-02-2013	13.57	13.6	13.21	13.57	6071400	AAL
25-02-2013	13.6	13.76	13	13.02	7186400	AAL
26-02-2013	13.14	13.42	12.7	13.26	9419000	AAL
27-02-2013	13.28	13.62	13.18	13.41	7390500	AAL
28-02-2013	13.49	13.63	13.39	13.43	6143600	AAL
01-03-2013	13.37	13.95	13.32	13.61	7376800	AAL
04-03-2013	13.5	14.07	13.47	13.9	8174800	AAL
05-03-2013	14.01	14.05	13.71	14.05	7676100	AAL
06-03-2013	14.52	14.68	14.25	14.57	13243200	AAL
07-03-2013	14.7	14.93	14.5	14.82	9125300	AAL
08-03-2013	14.99	15.2	14.84	14.92	10593700	AAL
11-03-2013	14.85	15.15	14.71	15.13	6961800	AAL
12-03-2013	15.14	15.6	14.95	15.5	8999100	AAL
13-03-2013	15.54	16.2	15.48	15.91	11380000	AAL
14-03-2013	15.98	16.36	15.93	16.25	8383300	AAL

2. Preprocessing the dataset:

- Data preprocessing is the process of cleaning, transforming, and integrating data in order to make it ready for analysis.
- This may involve removing errors and inconsistencies, handling missing values, transforming the data into a consistent format, and scaling the data to a suitable range.

Visualisation and Pre-Processing of Data:

In[1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
url = 'D:\data\msft.csv'
dataset_train = pd.read_csv('D:\data\msft.csv')
training_set = dataset_train.iloc[:, 1:2].values
print(dataset_train.head())
```

Out[1]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	3/13/1986	0.088542	0.101563	0.088542	0.097222	0.062549	1031788800
1	3/14/1986	0.097222	0.102431	0.097222	0.100694	0.064783	308160000
2	3/17/1986	0.100694	0.103299	0.100694	0.102431	0.065899	133171200
3	3/18/1986	0.102431	0.103299	0.098958	0.099826	0.064224	67766400
4	3/19/1986	0.099826	0.100694	0.097222	0.098090	0.063107	47894400

In [2]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tensorflow import keras
import seaborn as sns
```

```
import os
from datetime import datetime
import warnings
warnings.filterwarnings("ignore")
data = pd.read_csv('D:\data\msft.csv')
print(data.shape)
print(data.sample(7))
```

Out[2]:

```
(8525, 7)
            Date       Open       High  ...      Close  Adj Close    Volume
5340    5/15/2007  30.900000  31.090000  ...  30.900000  23.175358  75013900
1971   12/28/1993   2.515625   2.593750  ...   2.585938   1.663682  27776000
8041   02-05-2018  90.559998  93.239998  ...  88.000000  85.242134  51031500
163    11-03-1986   0.135417   0.138021  ...   0.137153   0.088238  42192000
6390    7/14/2011  26.620001  27.010000  ...  26.469999  21.534384  46382300
3936   10/15/2001  27.950001  29.250000  ...  29.030001  18.676666  68437000
1172   10/30/1990   0.868056   0.892361  ...   0.887153   0.570756  52563200

[7 rows x 7 columns]
```

In [3]:

```
print(data.info())
```

Out[3]:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8525 entries, 0 to 8524
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       8525 non-null   object
 1   Open       8525 non-null   float64
 2   High       8525 non-null   float64
 3   Low        8525 non-null   float64
 4   Close      8525 non-null   float64
 5   Adj Close  8525 non-null   float64
 6   Volume     8525 non-null   int64
dtypes: float64(5), int64(1), object(1)
memory usage: 466.3+ KB
None
```

In[4]:

```
df["Date"]=pd.to_datetime(df.Date,format="%d-%m-%Y")
df.index=df['Date']
plt.figure(figsize=(16,8))
plt.plot(df["Date"],label='stock Price history')
```

Out[4]:

In[5]:

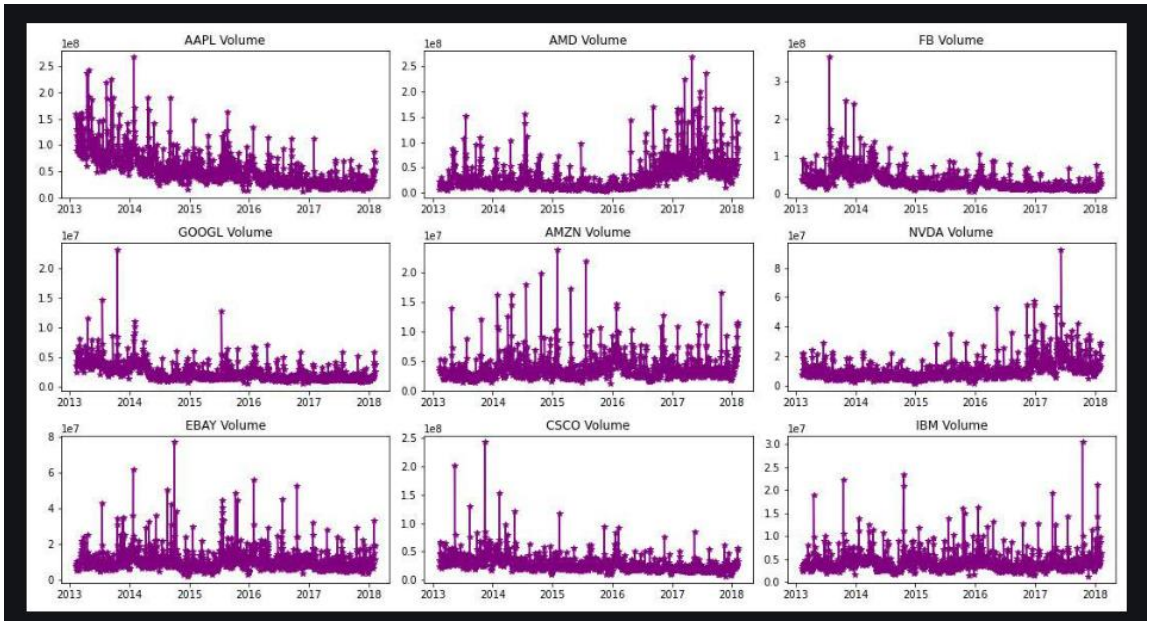
```
data['date'] = pd.to_datetime(data['date'])
# date vs open
# date vs close
plt.figure(figsize=(15, 8))
for index, company in enumerate(companies, 1):
        plt.subplot(3, 3, index)
        c = data[data['Name'] == company]
        plt.plot(c['date'], c['close'], c="r", label="close", marker="+")
        plt.plot(c['date'], c['open'], c="g", label="open", marker="^")
        plt.title(company)
        plt.legend()
        plt.tight_layout()
```

Out[5]:

In [6]:

```
plt.figure(figsize=(15, 8))
for index, company in enumerate(companies, 1):
    plt.subplot(3, 3, index)
    c = data[data['Name'] == company]
    plt.plot(c['date'], c['volume'], c='purple', marker='*')
    plt.title(f"{company} Volume")
    plt.tight_layout()
```

Out[6]:



In[7]:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(dataset)
train_data = scaled_data[0:int(training), :]
# prepare feature and labels
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])

x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))

model = keras.models.Sequential()
model.add(keras.layers.LSTM(units=64,
                            return_sequences=True,
                            input_shape=(x_train.shape[1], 1)))
model.add(keras.layers.LSTM(units=64))
model.add(keras.layers.Dense(32))
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Dense(1))
model.summary
```

Out[7]:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 60, 64)            16896

 lstm_1 (LSTM)               (None, 64)                33024

 dense (Dense)               (None, 32)                2080

 dropout (Dropout)           (None, 32)                0

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 52,033
Trainable params: 52,033
Non-trainable params: 0
_________________________________________________________________
```

In[8]:

```
model.compile(optimizer='adam', loss='mean_squared_error')
history = model.fit(x_train,  y_train,  epochs=10)
```

Out[8]:

```
Epoch 1/10
36/36 [==============================] - 4s 37ms/step - loss: 0.0334
Epoch 2/10
36/36 [==============================] - 1s 34ms/step - loss: 0.0095
Epoch 3/10
36/36 [==============================] - 1s 36ms/step - loss: 0.0092
Epoch 4/10
36/36 [==============================] - 1s 35ms/step - loss: 0.0081
Epoch 5/10
36/36 [==============================] - 1s 36ms/step - loss: 0.0073
Epoch 6/10
36/36 [==============================] - 1s 37ms/step - loss: 0.0073
Epoch 7/10
36/36 [==============================] - 1s 37ms/step - loss: 0.0071
Epoch 8/10
36/36 [==============================] - 1s 36ms/step - loss: 0.0071
Epoch 9/10
36/36 [==============================] - 1s 36ms/step - loss: 0.0069
Epoch 10/10
36/36 [==============================] - 1s 36ms/step - loss: 0.0065
```

In [9]:

```
test_data = scaled_data[training - 60:, :]
x_test = []
y_test = dataset[training:, :]
for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0]
x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
```

```
# predict the testing data
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)

# evaluation metrics
mse = np.mean(((predictions - y_test) ** 2))
print("MSE", mse)
print("RMSE", np.sqrt(mse))
```

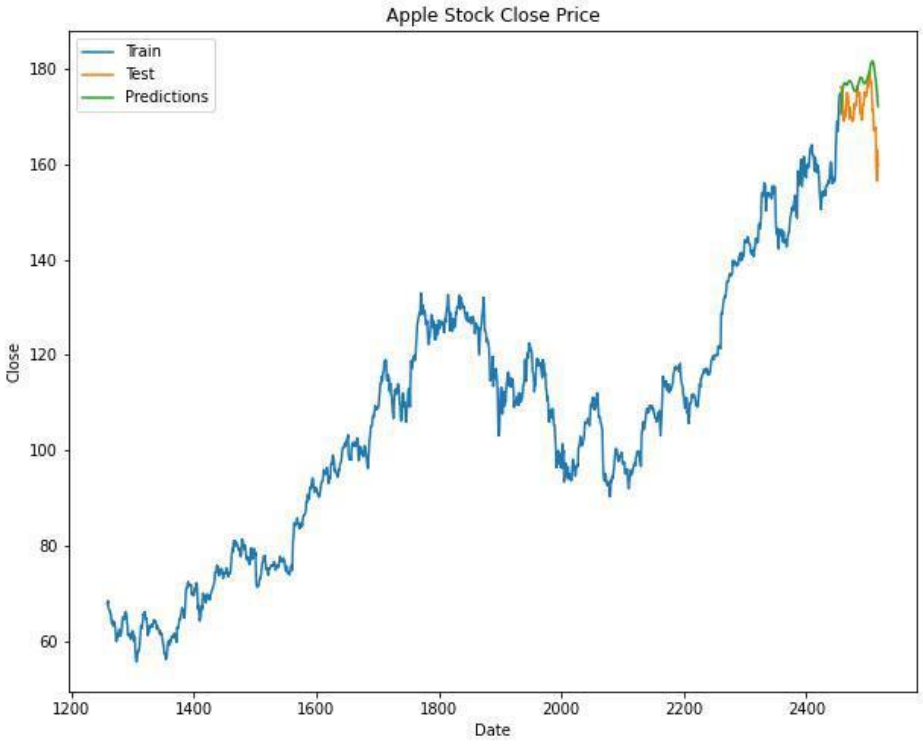
Out[9]:

```
2/2 [==============================] - 1s 13ms/step
MSE 46.06080444818086
RMSE 6.786811066191607
```

In [10]:

```
train = apple[:training]
test = apple[training:]
test['Predictions'] = predictions

plt.figure(figsize=(10, 8))
plt.plot(train['Date'], train['Close'])
plt.plot(test['Date'], test[['Close', 'Predictions']])
plt.title('Apple Stock Close Price')
plt.xlabel('Date')
plt.ylabel("Close")
plt.legend(['Train', 'Test', 'Predictions'])
```

Out[10]:



In[11]:

```
all_stock_tick_names = data['Name'].unique()
print(all_stock_tick_names)
```

Out[11]:

```
['AAL' 'AAPL' 'AAP' 'ABBV' 'ABC' 'ABT' 'ACN' 'ADBE' 'ADI' 'ADM' 'ADP'
 'ADSK' 'ADS' 'AEE' 'AEP' 'AES' 'AET' 'AFL' 'AGN' 'AIG' 'AIV' 'AIZ' 'AJG'
 'AKAM' 'ALB' 'ALGN' 'ALK' 'ALLE' 'ALL' 'ALXN' 'AMAT' 'AMD' 'AME' 'AMGN'
 'AMG' 'AMP' 'AMT' 'AMZN' 'ANDV' 'ANSS' 'ANTM' 'AON' 'AOS' 'APA' 'APC'
 'APD' 'APH' 'APTV' 'ARE' 'ARNC' 'ATVI' 'AVB' 'AVGO' 'AVY' 'AWK' 'AXP'
 'AYI' 'AZO' 'A' 'BAC' 'BAX' 'BA' 'BBT' 'BBY' 'BDX' 'BEN' 'BF.B' 'BHF'
 'BHGE' 'BIIB' 'BK' 'BLK' 'BLL' 'BMY' 'BRK.B' 'BSX' 'BWA' 'BXP' 'CAG'
 'CAH' 'CAT' 'CA' 'CBG' 'CBOE' 'CBS' 'CB' 'CCI' 'CCL' 'CDNS' 'CELG' 'CERN'
 'CFG' 'CF' 'CHD' 'CHK' 'CHRW' 'CHTR' 'CINF' 'CI' 'CLX' 'CL' 'CMA' 'CMCSA'
 'CME' 'CMG' 'CMI' 'CMS' 'CNC' 'CNP' 'COF' 'COG' 'COL' 'COO' 'COP' 'COST'
 'COTY' 'CPB' 'CRM' 'CSCO' 'CSRA' 'CSX' 'CTAS' 'CTL' 'CTSH' 'CTXS' 'CVS'
 'CVX' 'CXO' 'C' 'DAL' 'DE' 'DFS' 'DGX' 'DG' 'DHI' 'DHR' 'DISCA' 'DISCK'
 'DISH' 'DIS' 'DLR' 'DLTR' 'DOV' 'DPS' 'DRE' 'DRI' 'DTE' 'DUK' 'DVA' 'DVN'
 'DWDP' 'DXC' 'D' 'EA' 'EBAY' 'ECL' 'ED' 'EFX' 'EIX' 'EL' 'EMN' 'EMR'
 'EOG' 'EQIX' 'EQR' 'EQT' 'ESRX' 'ESS' 'ES' 'ETFC' 'ETN' 'ETR' 'EVHC' 'EW'
 'EXC' 'EXPD' 'EXPE' 'EXR' 'FAST' 'FBHS' 'FB' 'FCX' 'FDX' 'FE' 'FFIV'
 'FISV' 'FIS' 'FITB' 'FLIR' 'FLR' 'FLS' 'FL' 'FMC' 'FOXA' 'FOX' 'FRT'
 'FTI' 'FTV' 'F' 'GD' 'GE' 'GGP' 'GILD' 'GIS' 'GLW' 'GM' 'GOOGL' 'GOOG'
 'GPC' 'GPN' 'GPS' 'GRMN' 'GS' 'GT' 'GWW' 'HAL' 'HAS' 'HBAN' 'HBI' 'HCA'
 'HCN' 'HCP' 'HD' 'HES' 'HIG' 'HII' 'HLT' 'HOG' 'HOLX' 'HON' 'HPE' 'HPQ'
```

In[12]:

```
stock_name = input("Enter a Stock Price Name: ")

# 2. Extrating all the data having the name same as the stock name entered
all_data = data['Name'] == stock_name

# 3. Putting all the rows of specific stock in a variable
final_data = data[all_data]

# 4. Printing first 5 rows of the stock data of a specific stock name
final_data.head()
```

Out[12]:

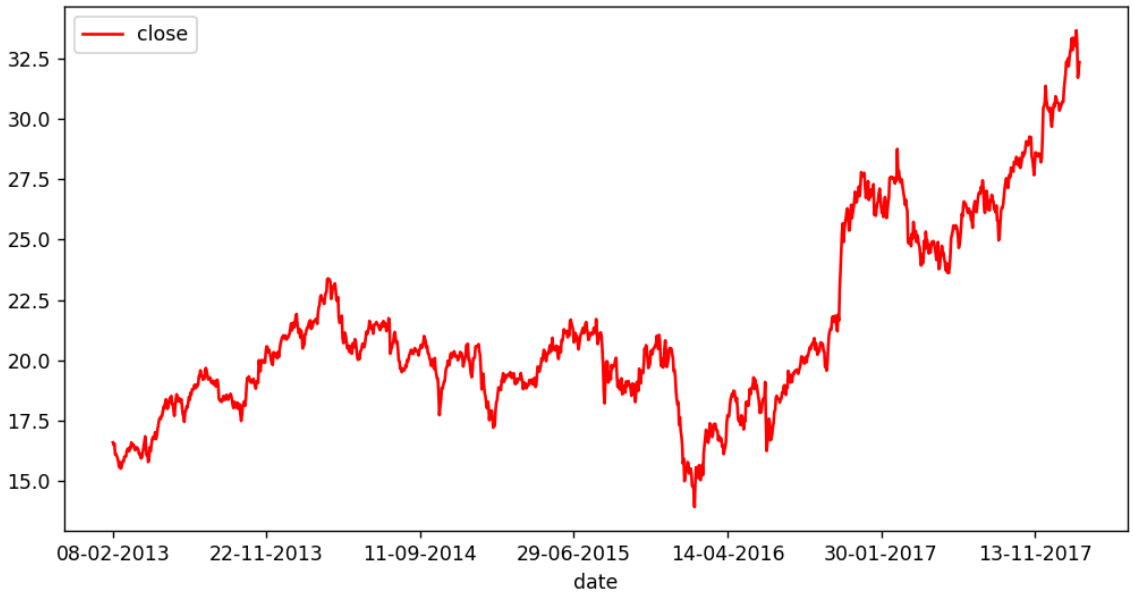
Enter a Stock Price Name: FITB

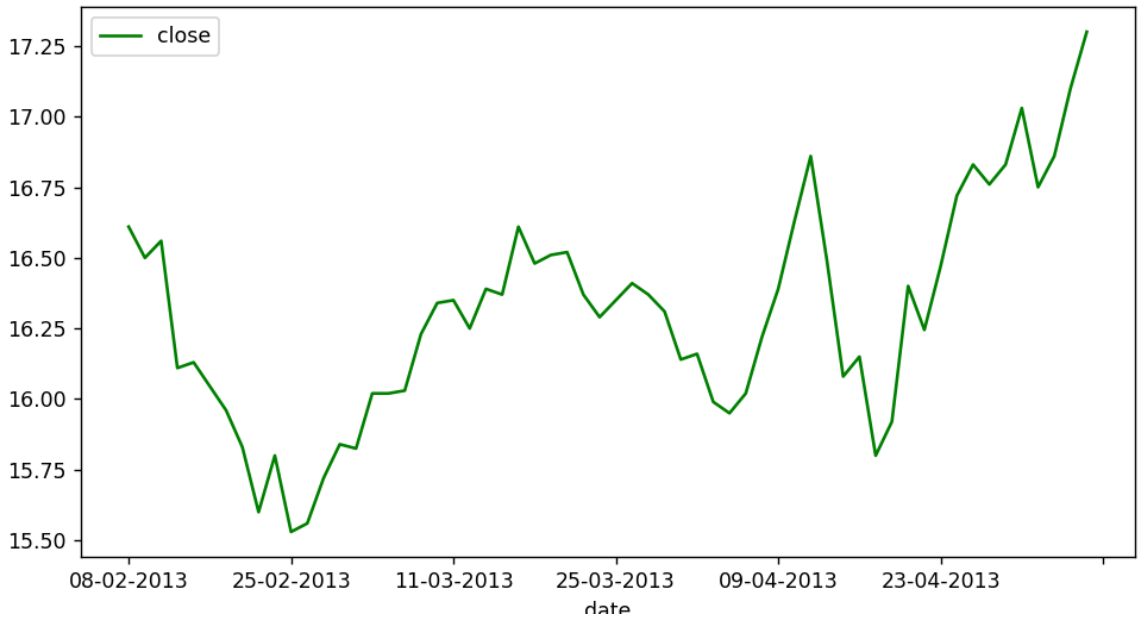
	date	open	high	low	close	volume	Name
173481	08-02-13	16.55	16.61	16.40	16.61	9814015	FITB
173482	11-02-13	16.44	16.52	16.38	16.50	12187679	FITB
173483	12-02-13	16.54	16.63	16.45	16.56	8284724	FITB
173484	13-02-13	16.53	16.55	16.05	16.11	16635670	FITB
173485	14-02-13	16.09	16.19	16.04	16.13	11091204	FITB

In[13]:

```
#  Plotting date vs the close market stock price
final_data.plot('date', 'close', color="red")

# Extract only top 60 rows to make the plot a little clearer
new_data = final_data.head(60)

#  Plotting date vs the close  market stock price
new_data.plot('date', 'close', color="green")

plt.show()
```

Out[13]:





In[14]:

```
# 1. Filter out the closing market price data
close_data = final_data.filter(['close'])
```

```
# 2. Convert the data into array for easy evaluation
dataset = close_data.values
# 3. Scale/Normalize the data to make all values between 0 and 1
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(dataset)

# 4. Creating training data size : 70% of the data
training_data_len = math.ceil(len(dataset) * .7)
train_data = scaled_data[0:training_data_len, :]

# 5. Separating the data into x and y data
x_train_data = []
y_train_data = []
for i in range(60, len(train_data)):
    x_train_data = list(x_train_data)
    y_train_data = list(y_train_data)
    x_train_data.append(train_data[i - 60:i, 0])
    y_train_data.append(train_data[i, 0])

    # 6. Converting the training x and y values to numpy arrays
    x_train_data1, y_train_data1 = np.array(x_train_data), np.array(y_train_data)

    # 7. Reshaping training s and y data to make the calculations easier
    x_train_data2 = np.reshape(x_train_data1, (x_train_data1.shape[0],
x_train_data1.shape[1], 1))  model = Sequential()
model.add(LSTM(units=50,
return_sequences=True,input_shape=(x_train_data2.shape[1],1)))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dense(units=25))
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train_data2, y_train_data1, batch_size=1, epochs=1)
```

OUT[14]:

```
822/822 [==============================] - 16s 16ms/step - loss: 0.0016
```

In[15]:

```
# 1. Creating a dataset for testing
test_data = scaled_data[training_data_len - 60:, :]
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])

# 2.  Convert the values into arrays for easier computation
x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))


# 3. Making predictions on the testing data
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)


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

OUT[15]:

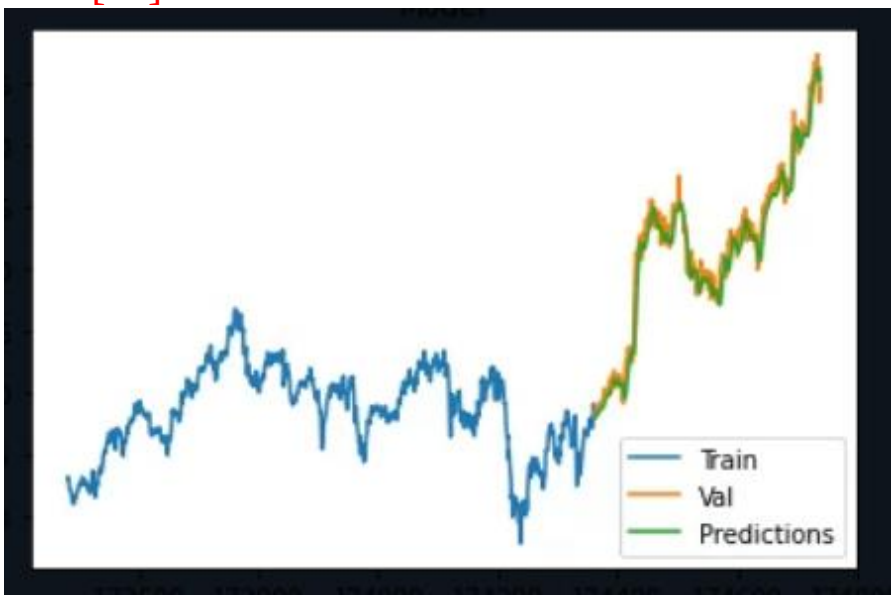
```
0.8354904240670301
```

In[16]:

```
train = data[:training_data_len]
valid = data[training_data_len:]
```

```
valid['Predictions'] = predictions

plt.title('Model')
plt.xlabel('Date')
plt.ylabel('Close')

plt.plot(train['close'])
plt.plot(valid[['close', 'Predictions']])

plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')

plt.show()
```

OUT[16]:



Some common data preprocessing tasks include:

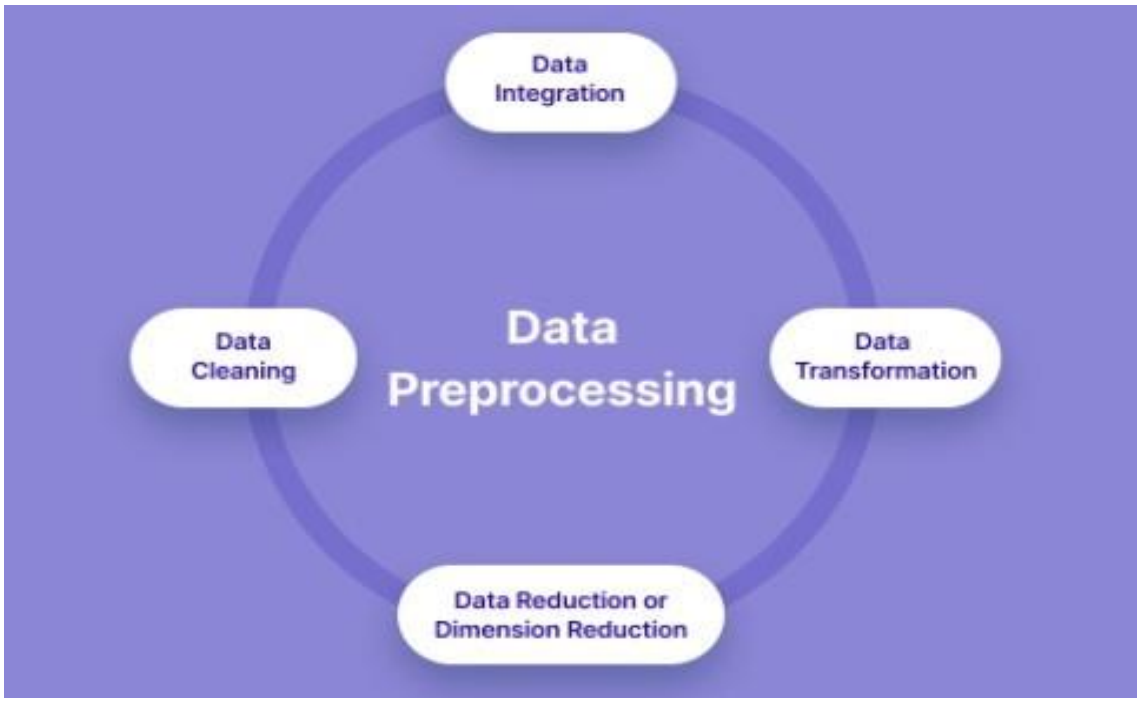
- **Data cleaning:** This involves identifying and correcting errors and inconsistencies in the data. For example, this may involve removing duplicate records, correcting typos, and filling in missing values.

- **Data transformation:** This involves converting the data into a format that is suitable for the analysis task. For example, this may involve converting categorical data to numerical data, or scaling the data to a suitable range.

- **Feature engineering:** This involves creating new features from the existing data. For example, this may involve creating features that represent interactions between variables, or features that represent summary statistics of the data.

- **Data integration:** This involves combining data from multiple sources into a single dataset. This may involve resolving inconsistencies in the data, such as different data formats or different variable names.

Data preprocessing is an essential step in many data science projects. By carefully preprocessing the data, data scientists can improve the accuracy and reliability of their results.



Conclusion:

- In the quest to build a stock price prediction model, we have embarked on a critical journey that begins with loading and preprocessing the dataset.We have traversed through essential steps, starting with importing the necessary libraries to facilitate data manipulation and analysis.
- Understanding the data's structure, characteristics, and any potential issues through exploratory data analysis (EDA) is essential for informed decision-making.
- Data preprocessing emerged as a pivotal aspect of this process. It involves cleaning, transforming, and refining the dataset to ensure that it aligns with the requirements of machine learning algorithms.
- With these foundational steps completed, our dataset is now primed for the subsequent stages of building and training a house price prediction model.