Stock Market Prediction Using LSTM By Shantanu Garain

Importing Essential Libraries

In [2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Loading data to the notebook

In [3]:
```
data = pd.read_csv('aapl_raw_data.csv')
```

Looking for top 5 column of our dataset

In [4]:
```
data.head()
```

Out[4]:

	date	open	high	low	close	volume	adjusted_close	change_percent	avg_vol_20(
0	1984-11-05	24.7520	25.3792	24.5056	24.7520	470399	0.0856	NaN	NaN
1	1984-11-06	26.2528	26.3760	24.9984	26.2528	1005899	0.0908	6.07	NaN
2	1984-11-07	25.7488	26.3760	24.8752	25.7488	1033699	0.0891	-1.87	NaN
3	1984-11-08	24.7520	25.7488	24.6288	24.7520	393399	0.0856	-3.93	NaN
4	1984-11-09	23.2512	24.8752	23.0048	23.2512	1313099	0.0804	-6.07	NaN

Looking for bottom 5 column of our dataset

In [5]:
```
data.tail()
```

Out[5]:

	date	open	high	low	close	volume	adjusted_close	change_percent	avg_vol_2
9798	2023-09-22	174.67	177.080	174.05	174.79	56663000	174.79	0.49	66084972.
9799	2023-09-25	174.20	176.970	174.15	176.08	46172700	176.08	0.74	65821128..
9800	2023-09-26	174.82	175.200	171.66	171.96	64588900	171.96	-2.34	66859538..
9801	2023-09-27	172.62	173.040	169.05	170.43	66921800	170.43	-0.89	67555430.
9802	2023-09-28	169.34	172.026	167.62	170.69	56190062	170.69	0.15	67324239..

Size of our data

In [6]:

```
data.size
```

Out[6]:

```
88227
```

Shape of our data

In [7]:

```
data.shape
```

Out[7]:

```
(9803, 9)
```

Some important information about the columns of our data

In [8]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9803 entries, 0 to 9802
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   date            9803 non-null   object 
 1   open            9803 non-null   float64
 2   high            9803 non-null   float64
 3   low             9803 non-null   float64
 4   close           9803 non-null   float64
 5   volume          9803 non-null   int64  
 6   adjusted_close  9803 non-null   float64
 7   change_percent  9802 non-null   float64
 8   avg_vol_20d     9784 non-null   float64
dtypes: float64(7), int64(1), object(1)
memory usage: 689.4+ KB
```

Some statistical information about the data:

In [9]:
```
data.describe()
```

Out[9]:

	open	high	low	close	volume	adjusted_close	change_
count	9803.000000	9803.000000	9803.000000	9803.000000	9.803000e+03	9803.000000	9802.
mean	121.004966	122.456075	119.470947	121.002267	2.047305e+07	20.204214	0.
std	135.119023	136.267710	133.813208	135.076731	2.781417e+07	41.263009	2.
min	12.874400	13.190800	12.720400	12.936000	9.600000e+04	0.050200	-51.
25%	34.501600	35.000000	33.751200	34.501600	1.909849e+06	0.278900	-1.
50%	61.600000	62.748000	60.625600	61.751200	8.465999e+06	0.779200	0.
75%	154.675000	156.390000	153.070000	154.600000	2.843074e+07	18.431250	1.
max	702.410000	705.070000	699.570000	702.100000	3.326072e+08	196.185100	33.

In [ ]:

In [ ]:

DATA PREPROCESSING AND DATA CLEANING:

Null values in each column:

In [10]:
```
data.isnull().sum()
```

Out[10]:
```
date              0
open              0
high              0
low               0
close             0
volume            0
adjusted_close    0
change_percent    1
avg_vol_20d      19
dtype: int64
```

Check duplicate and remove them

In [11]:
```
data.duplicated().sum()
data.drop_duplicates(inplace=True)
```

Data Transformation

In [12]:
```
# check the data types of the solumns
data.dtypes
```

Out[12]:
```
date                 object
open                float64
high                float64
low                 float64
close               float64
volume                int64
adjusted_close      float64
change_percent      float64
avg_vol_20d         float64
dtype: object
```

In [13]:
```
# Change the 'date' format to datetime format
data['date'] = pd.to_datetime(data['date'])
data.dtypes
```

Out[13]:
```
date                datetime64[ns]
open                       float64
high                       float64
low                        float64
close                      float64
volume                       int64
adjusted_close             float64
change_percent             float64
avg_vol_20d                float64
dtype: object
```

Feature Engineering

In [14]:
```
def add_features(data):

    # add day of the week feature
    data['day_of week'] = data['date'].dt.dayofweek

    # add month feature
    data['month'] = data['date'].dt.month

    # Add year feature
    data['year'] = data['date'].dt.year

    # Add week of the year feature
    data['week_of_year'] = data['date'].dt.isocalendar().week

    # Add day of the year feature
    data['day_of_year'] = data['date'].dt.dayofyear

    return data
```

In [15]:
```
df = add_features(data)
df.head()
```

Out[15]:

	date	open	high	low	close	volume	adjusted_close	change_percent	avg_vol_20c
0	1984-11-05	24.7520	25.3792	24.5056	24.7520	470399	0.0856	NaN	NaN
1	1984-11-06	26.2528	26.3760	24.9984	26.2528	1005899	0.0908	6.07	NaN
2	1984-11-07	25.7488	26.3760	24.8752	25.7488	1033699	0.0891	-1.87	NaN
3	1984-11-08	24.7520	25.7488	24.6288	24.7520	393399	0.0856	-3.93	NaN
4	1984-11-09	23.2512	24.8752	23.0048	23.2512	1313099	0.0804	-6.07	NaN

In [16]:
```
df.corr()
```

```
C:\Users\SHANTANU GARAIN\AppData\Local\Temp\ipykernel_7180\1134722465.py:1: Future
Warning: The default value of numeric_only in DataFrame.corr is deprecated. In a f
uture version, it will default to False. Select only valid columns or specify the
value of numeric_only to silence this warning.
  df.corr()
```

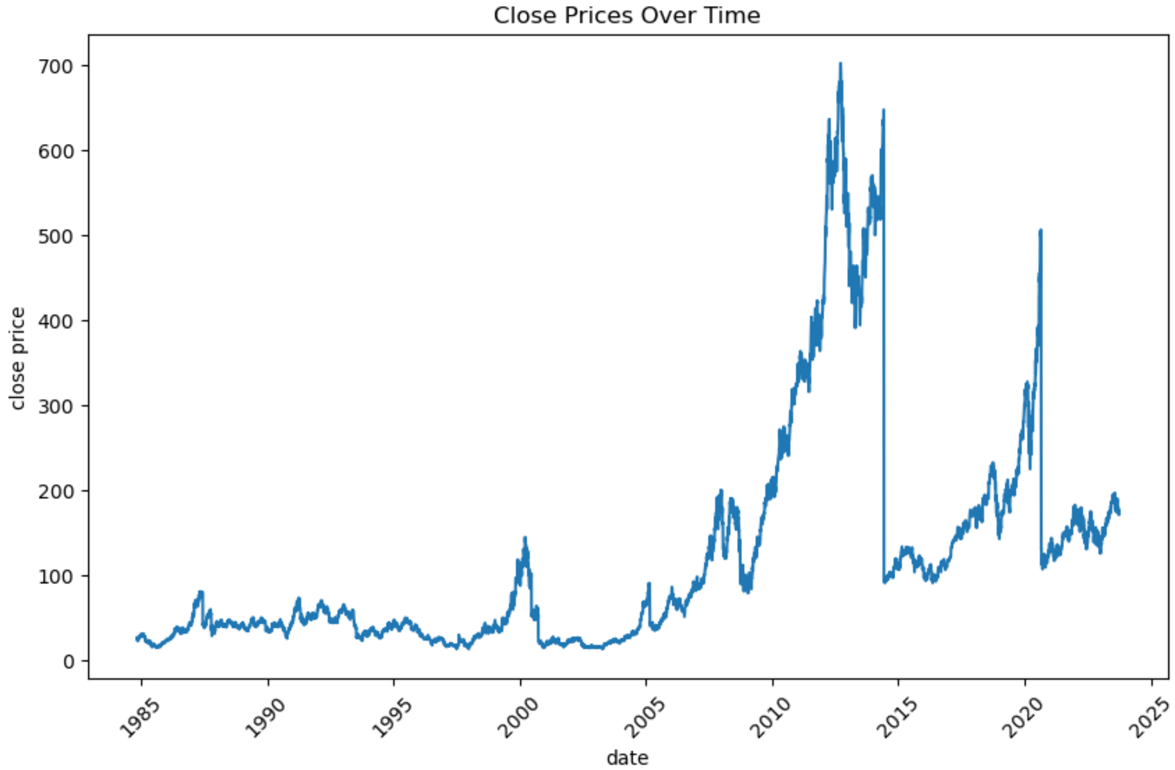
Out[16]:

	open	high	low	close	volume	adjusted_close	change_perce
open	1.000000	0.999914	0.999887	0.999802	0.220890	0.252259	0.0037
high	0.999914	1.000000	0.999860	0.999905	0.222368	0.252422	0.0071
low	0.999887	0.999860	1.000000	0.999899	0.219083	0.253012	0.0077
close	0.999802	0.999905	0.999899	1.000000	0.220718	0.252873	0.0128
volume	0.220890	0.222368	0.219083	0.220718	1.000000	0.748368	-0.0182
adjusted_close	0.252259	0.252422	0.253012	0.252873	0.748368	1.000000	0.0013
change_percent	0.003715	0.007143	0.007743	0.012879	-0.018281	0.001316	1.0000
avg_vol_20d	0.110114	0.111613	0.107308	0.109400	-0.007536	-0.343451	0.0032
day_of week	0.004424	0.003533	0.004137	0.003385	0.012004	0.002525	-0.0268
month	-0.012036	-0.012402	-0.011961	-0.012248	-0.002384	0.002997	-0.0082
year	0.564063	0.564115	0.564681	0.564475	0.704051	0.676509	0.0041
week_of_year	-0.012406	-0.012777	-0.012347	-0.012654	-0.000948	0.002119	-0.0077
day_of_year	-0.011722	-0.012076	-0.011628	-0.011927	-0.002202	0.002869	-0.0097

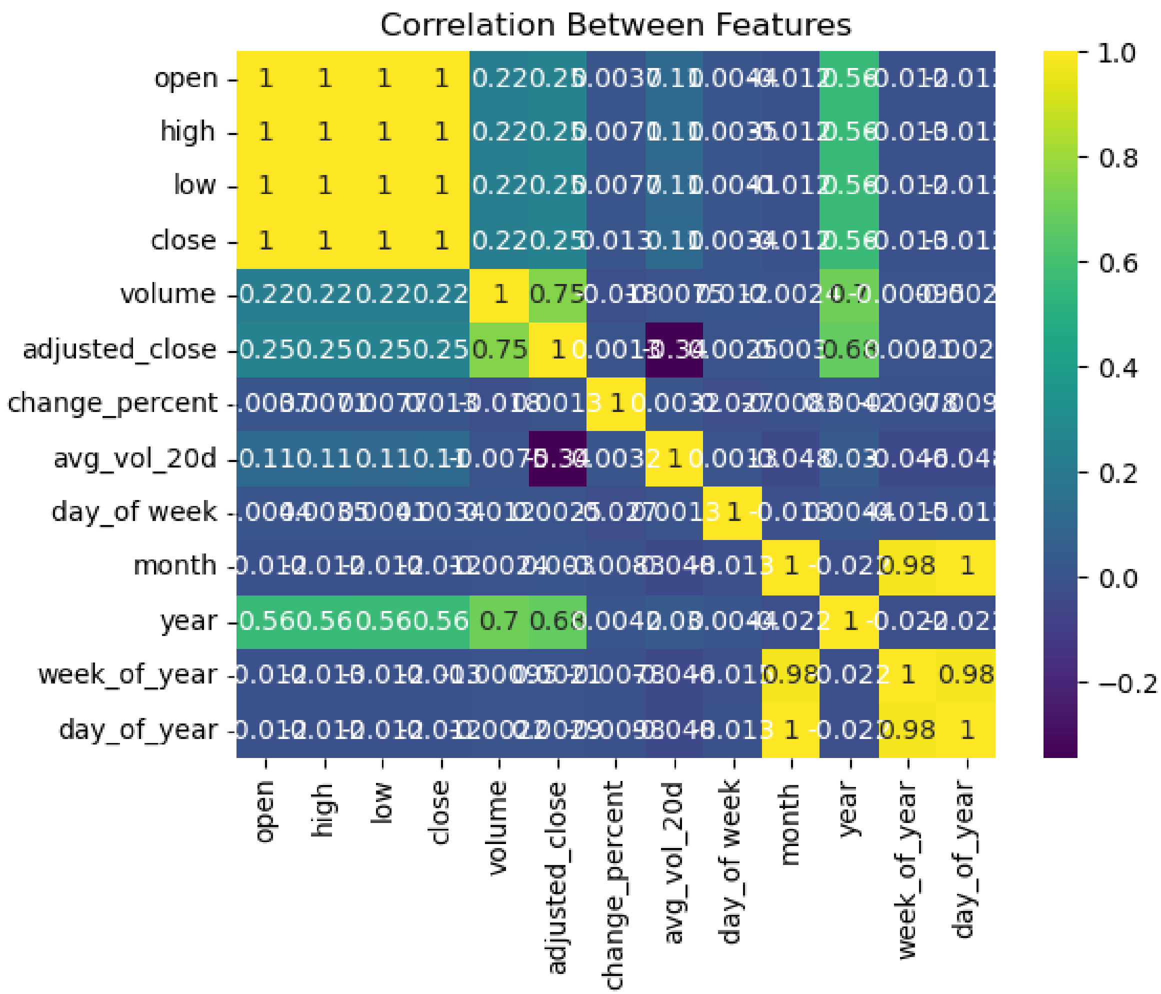
EXPLORATORY DATA ANALYSIS

In [17]:
```
plt.figure(figsize=(10, 6))

# Line plot of close prices over time
sns.lineplot(data=df, x='date', y='close')
plt.title('Close Prices Over Time')
```

```
plt.xlabel('date')
plt.ylabel('close price')
plt.xticks(rotation=45)
plt.show()

# Heatmap of correlation between features
corr = df.corr()
#sns.heatmap(corr, cmap='coolwarm', annot=True)
sns.heatmap(corr, cmap='viridis', annot=True)
plt.title('Correlation Between Features')
plt.show()
```




```
C:\Users\SHANTANU GARAIN\AppData\Local\Temp\ipykernel_7180\1531136231.py:12: Futur
eWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a
future version, it will default to False. Select only valid columns or specify the
value of numeric_only to silence this warning.
  corr = df.corr()
```



LSTM Model

In [18]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense, LSTM
from keras.layers import Dropout

from sklearn.preprocessing import MinMaxScaler
```

In [19]:
```
# Create new dataframe with Close column
data = df.filter(['close'])
# Convert it with numpy array
dataset = data.values
# Get the number of rows to train the model on
len_train_data = int(np.ceil(len(dataset) * .95))

len_train_data
```

Out[19]: 9313

Normalizing Data

In [20]:
```
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(dataset)

trained_scaled_data = scaler.fit_transform(dataset)
```

```
trained_scaled_data
```

Out[20]:
```
array([[0.01714541],
       [0.01932312],
       [0.0185918 ],
       ...,
       [0.23074914],
       [0.22852906],
       [0.22890633]])
```

Creating the training data set

In [21]:
```
train_data = trained_scaled_data[0:int(len_train_data), :]
```

Now splitting the data into x_train, y_train

In [22]:
```
# Splitting the data into x_train, y_train

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 train data to numpy array
x_train, y_train = np.array(x_train), np.array(y_train)

# Reshapping our new data
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```

```
[array([0.01714541, 0.01932312, 0.0185918 , 0.01714541, 0.0149677 ,
       0.01623532, 0.01532523, 0.01569902, 0.01569902, 0.0149677 ,
       0.01296876, 0.01405761, 0.01478893, 0.01569902, 0.01605656,
       0.01696664, 0.01877057, 0.0180555 , 0.01714541, 0.01659286,
       0.01732418, 0.01914435, 0.02094828, 0.02076951, 0.02003819,
       0.01950189, 0.01823427, 0.0185918 , 0.01950189, 0.02041198,
       0.02276846, 0.02112705, 0.02094828, 0.02041198, 0.02112705,
       0.02132207, 0.02150083, 0.02294722, 0.02348352, 0.0216796 ,
       0.02239467, 0.02239467, 0.0222159 , 0.02185837, 0.02294722,
       0.0247674 , 0.02439361, 0.02566124, 0.0247674 , 0.02512493,
       0.02203713, 0.02276846, 0.02367854, 0.02494617, 0.02421485,
       0.02330476, 0.02421485, 0.02512493, 0.02457238, 0.02457238])]
[0.023304757648397192]

[array([0.01714541, 0.01932312, 0.0185918 , 0.01714541, 0.0149677 ,
       0.01623532, 0.01532523, 0.01569902, 0.01569902, 0.0149677 ,
       0.01296876, 0.01405761, 0.01478893, 0.01569902, 0.01605656,
       0.01696664, 0.01877057, 0.0180555 , 0.01714541, 0.01659286,
       0.01732418, 0.01914435, 0.02094828, 0.02076951, 0.02003819,
       0.01950189, 0.01823427, 0.0185918 , 0.01950189, 0.02041198,
       0.02276846, 0.02112705, 0.02094828, 0.02041198, 0.02112705,
       0.02132207, 0.02150083, 0.02294722, 0.02348352, 0.0216796 ,
       0.02239467, 0.02239467, 0.0222159 , 0.02185837, 0.02294722,
       0.0247674 , 0.02439361, 0.02566124, 0.0247674 , 0.02512493,
       0.02203713, 0.02276846, 0.02367854, 0.02494617, 0.02421485,
       0.02330476, 0.02421485, 0.02512493, 0.02457238, 0.02457238]), array([0.0193
2312, 0.0185918 , 0.01714541, 0.0149677 , 0.01623532,
       0.01532523, 0.01569902, 0.01569902, 0.0149677 , 0.01296876,
       0.01405761, 0.01478893, 0.01569902, 0.01605656, 0.01696664,
       0.01877057, 0.0180555 , 0.01714541, 0.01659286, 0.01732418,
       0.01914435, 0.02094828, 0.02076951, 0.02003819, 0.01950189,
       0.01823427, 0.0185918 , 0.01950189, 0.02041198, 0.02276846,
       0.02112705, 0.02094828, 0.02041198, 0.02112705, 0.02132207,
       0.02150083, 0.02294722, 0.02348352, 0.0216796 , 0.02239467,
       0.02239467, 0.0222159 , 0.02185837, 0.02294722, 0.0247674 ,
       0.02439361, 0.02566124, 0.0247674 , 0.02512493, 0.02203713,
       0.02276846, 0.02367854, 0.02494617, 0.02421485, 0.02330476,
       0.02421485, 0.02512493, 0.02457238, 0.02457238, 0.02330476])]
[0.023304757648397192, 0.022768455694145366]
```

In [23]:
```
x_train.shape
```

Out[23]:
```
(9253, 60, 1)
```

Modelling

In [24]:
```
from keras.layers import Activation
from keras.models import Sequential
from keras.layers import LSTM, Dense, Activation
```

In [25]:
```
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.add(Activation('linear'))

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

```
# Train the model
model.fit(x_train, y_train, batch_size=32, epochs=14)
```

```
Epoch 1/14
290/290 [==============================] - 35s 91ms/step - loss: 0.0011
Epoch 2/14
290/290 [==============================] - 25s 87ms/step - loss: 4.2230e-04
Epoch 3/14
290/290 [==============================] - 27s 92ms/step - loss: 3.1383e-04
Epoch 4/14
290/290 [==============================] - 30s 102ms/step - loss: 2.8379e-04
Epoch 5/14
290/290 [==============================] - 30s 103ms/step - loss: 2.1232e-04
Epoch 6/14
290/290 [==============================] - 32s 109ms/step - loss: 2.5876e-04
Epoch 7/14
290/290 [==============================] - 29s 101ms/step - loss: 1.9490e-04
Epoch 8/14
290/290 [==============================] - 26s 91ms/step - loss: 1.8520e-04
Epoch 9/14
290/290 [==============================] - 27s 94ms/step - loss: 1.9211e-04
Epoch 10/14
290/290 [==============================] - 25s 87ms/step - loss: 1.8016e-04
Epoch 11/14
290/290 [==============================] - 26s 90ms/step - loss: 1.8897e-04
Epoch 12/14
290/290 [==============================] - 23s 80ms/step - loss: 1.9500e-04
Epoch 13/14
290/290 [==============================] - 23s 80ms/step - loss: 1.8863e-04
Epoch 14/14
290/290 [==============================] - 24s 82ms/step - loss: 1.6803e-04
```

Out[25]:

```
<keras.src.callbacks.History at 0x26dad5d2ce0>
```

Prediction

In [26]:

```
# Creating Test Data
test_data = trained_scaled_data[len_train_data - 60: , :]

# Create the datasets x_test and y_test
x_test = []
y_test = dataset[len_train_data:, :]
for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0])

# Convert data into numpy array
x_test = np.array(x_test)
```

In [27]:

```
x_test
```

Out[27]:
```
array([[0.19679496, 0.1974189 , 0.19419761, ..., 0.18569455, 0.18983    ,
        0.19139711],
       [0.1974189 , 0.19419761, 0.19160026, ..., 0.18983    , 0.19139711,
        0.19387838],
       [0.19419761, 0.19160026, 0.19255794, ..., 0.19139711, 0.19387838,
        0.19708516],
       ...,
       [0.26268639, 0.26049532, 0.25885566, ..., 0.23360768, 0.23485556,
        0.2367274 ],
       [0.26049532, 0.25885566, 0.25955215, ..., 0.23485556, 0.2367274 ,
        0.23074914],
       [0.25885566, 0.25955215, 0.25791249, ..., 0.2367274 , 0.23074914,
        0.22852906]])
```

In [28]:
```
# Reshape the data
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
```

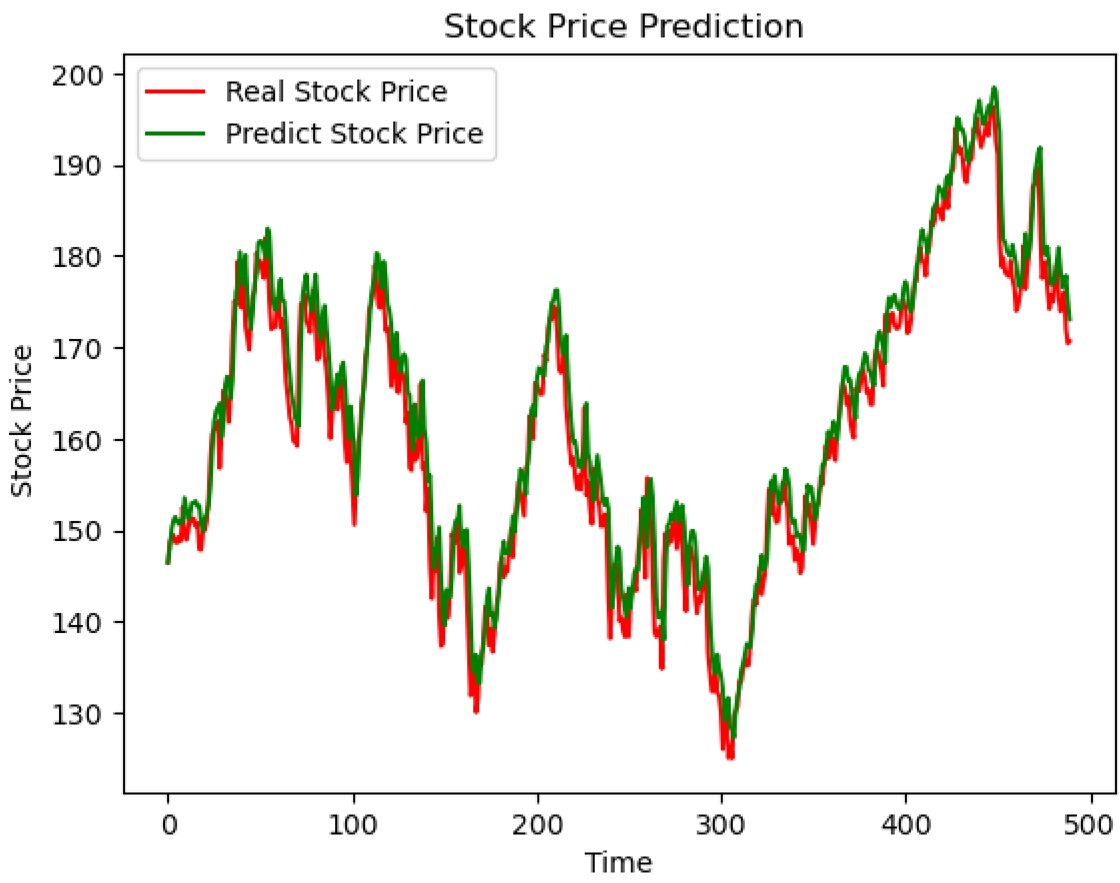
In [29]:
```
# Predict price
predict_price = model.predict(x_test)
# inverse transform for getting back all normal values from scaled values
predict_price = scaler.inverse_transform(predict_price)
```

```
16/16 [==============================] - 2s 34ms/step
```

In [30]:
```
rmse = np.sqrt(np.mean(((predict_price - y_test) ** 2)))
rmse
```

Out[30]:
```
3.6405515173721317
```

In [31]:
```
plt.plot(y_test, color = 'red', label = 'Real Stock Price')
plt.plot(predict_price, color = 'green', label = 'Predict Stock Price')
plt.title(' Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel(' Stock Price')
plt.legend()
plt.show()
```



As we can see from the above graph that there is very less loss between predicted and original data

Conclusion

In this project, we analyzed and predicted the stock price on the apple dataset. We startde with step by step process with preprocessing, data cleaning, feature engineering to extract usefull Information from our raw dataset. After that we perform exploratory data analysis to understand the relationships between the features and target variable.

Our model is based on LSTM model.

In [ ]: