In [ ]:
```
#Import Libraries
import math
import pandas_datareader as web
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.layers import Dense, LSTM
from keras.models import Sequential
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
```

In [ ]:
```
#Fetch the stock data
df = web.DataReader('AAPL', data_source='yahoo', start='2012-01-01', end='2021-12-17')
df
```

Out[ ]:

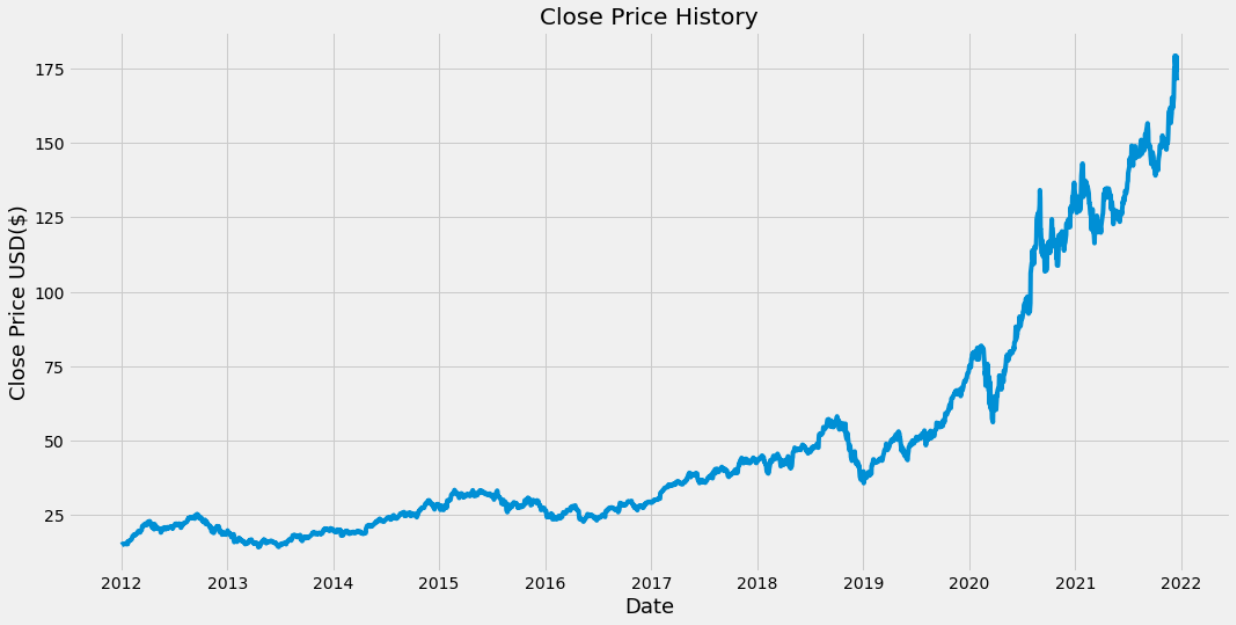
Date	High	Low	Open	Close	Volume	Adj Close
2012-01-03	14.732143	14.607143	14.621429	14.686786	302220800.0	12.557464
2012-01-04	14.810000	14.617143	14.642857	14.765714	260022000.0	12.624949
2012-01-05	14.948214	14.738214	14.819643	14.929643	271269600.0	12.765114
2012-01-06	15.098214	14.972143	14.991786	15.085714	318292800.0	12.898554
2012-01-09	15.276786	15.048214	15.196429	15.061786	394024400.0	12.878098
...	...	...	...	...	...	...
2021-12-13	182.130005	175.529999	181.119995	175.740005	153237000.0	175.258881
2021-12-14	177.740005	172.210007	175.250000	174.330002	139380400.0	173.852753
2021-12-15	179.500000	172.309998	175.110001	179.300003	131063300.0	178.809143
2021-12-16	181.139999	170.750000	179.279999	172.259995	150185800.0	171.788406
2021-12-17	173.470001	169.690002	169.929993	171.139999	195432700.0	170.671478

2508 rows × 6 columns

In [ ]:
```
#Get the number of rows an colums in the data set
df.shape
```

Out[ ]:
```
(2508, 6)
```

In [ ]:
```
#Display price history
plt.figure(figsize=(16,8))
plt.title('Close Price History')
plt.plot(df['Close'])
plt.xlabel('Date', fontsize = 18)
plt.ylabel('Close Price USD($)', fontsize=18)
plt.show()
```



In [ ]:
```
#Create a new dataset with only "Close" collum
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 = math.ceil(len(dataset) * .8)

training_data_len
```

Out[ ]:
```
2007
```

In [ ]:
```
#Scale the data
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)

scaled_data
```

Out[ ]:
```
array([[0.00446691],
       [0.00494381],
       [0.00593431],
       ...,
       [0.99909371],
       [0.95655653],
       [0.94978929]])
```

In [ ]:
```
#Create the training dataset
#Create the scaled training dataset
train_data = scaled_data[:training_data_len, :]
#Split the data into x_train and y_train data sets
x_train = []
y_train = []

#It takes every 60 days(0-60, 1-61, 2-62)
#Trains the data to estimate the next day with the last 60 days
for i in range(60, len(train_data)):
    x_train.append(train_data[i-60:i])
    y_train.append(train_data[i, 0])
    if i<= 61:
        print(x_train)
```

```
        print(y_train)
        print()
```

```
[array([[0.00446691],
       [0.00494381],
       [0.00593431],
       [0.00687732],
       [0.00673274],
       [0.00705859],
       [0.00690969],
       [0.00665937],
       [0.00631841],
       [0.00737365],
       [0.0083253 ],
       [0.00803182],
       [0.00642415],
       [0.00795844],
       [0.00644789],
       [0.01211246],
       [0.01167441],
       [0.01224626],
       [0.01348275],
       [0.01423154],
       [0.01416897],
       [0.01393807],
       [0.01492209],
       [0.01584783],
       [0.0168966 ],
       [0.01859057],
       [0.022149  ],
       [0.02220294],
       [0.02418393],
       [0.02566426],
       [0.02312007],
       [0.02409976],
       [0.02408035],
       [0.02682739],
       [0.0264368 ],
       [0.0271597 ],
       [0.02845878],
       [0.02918169],
       [0.0312641 ],
       [0.03278111],
       [0.03321917],
       [0.03337239],
       [0.03077856],
       [0.03015276],
       [0.03024555],
       [0.03268402],
       [0.03337024],
       [0.03484411],
       [0.03831837],
       [0.0429536 ],
       [0.04208611],
       [0.04208827],
       [0.04543954],
       [0.0464883 ],
       [0.04574164],
       [0.04505975],
       [0.04434979],
       [0.0467084 ],
       [0.04832685],
       [0.04900444]])]
```

```
[0.04732988300131198]

[array([[0.00446691],
       [0.00494381],
       [0.00593431],
       [0.00687732],
       [0.00673274],
       [0.00705859],
       [0.00690969],
       [0.00665937],
       [0.00631841],
       [0.00737365],
       [0.0083253 ],
       [0.00803182],
       [0.00642415],
       [0.00795844],
       [0.00644789],
       [0.01211246],
       [0.01167441],
       [0.01224626],
       [0.01348275],
       [0.01423154],
       [0.01416897],
       [0.01393807],
       [0.01492209],
       [0.01584783],
       [0.0168966 ],
       [0.01859057],
       [0.022149  ],
       [0.02220294],
       [0.02418393],
       [0.02566426],
       [0.02312007],
       [0.02409976],
       [0.02408035],
       [0.02682739],
       [0.0264368 ],
       [0.0271597 ],
       [0.02845878],
       [0.02918169],
       [0.0312641 ],
       [0.03278111],
       [0.03321917],
       [0.03337239],
       [0.03077856],
       [0.03015276],
       [0.03024555],
       [0.03268402],
       [0.03337024],
       [0.03484411],
       [0.03831837],
       [0.0429536 ],
       [0.04208611],
       [0.04208827],
       [0.04543954],
       [0.0464883 ],
       [0.04574164],
       [0.04505975],
       [0.04434979],
       [0.0467084 ],
```

```
[0.04832685],
[0.04900444]]), array([[0.00494381],
[0.00593431],
[0.00687732],
[0.00673274],
[0.00705859],
[0.00690969],
[0.00665937],
[0.00631841],
[0.00737365],
[0.0083253 ],
[0.00803182],
[0.00642415],
[0.00795844],
[0.00644789],
[0.01211246],
[0.01167441],
[0.01224626],
[0.01348275],
[0.01423154],
[0.01416897],
[0.01393807],
[0.01492209],
[0.01584783],
[0.0168966 ],
[0.01859057],
[0.022149  ],
[0.02220294],
[0.02418393],
[0.02566426],
[0.02312007],
[0.02409976],
[0.02408035],
[0.02682739],
[0.0264368 ],
[0.0271597 ],
[0.02845878],
[0.02918169],
[0.0312641 ],
[0.03278111],
[0.03321917],
[0.03337239],
[0.03077856],
[0.03015276],
[0.03024555],
[0.03268402],
[0.03337024],
[0.03484411],
[0.03831837],
[0.0429536 ],
[0.04208611],
[0.04208827],
[0.04543954],
[0.0464883 ],
[0.04574164],
[0.04505975],
[0.04434979],
[0.0467084 ],
[0.04832685],
[0.04900444],
```

```
       [0.04732988]])]
[0.04732988300131198, 0.045105060652022425]
```

In [ ]:
```
#Convert the x_train and y_train to numpy arrays
x_train, y_train = np.array(x_train), np.array(y_train)
```

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

Out[ ]:
```
(1947, 60, 1)
```

In [ ]:
```
#Build the LSTM model
model = Sequential()
model.add(LSTM(100, return_sequences=True, input_shape=(x_train.shape[1], 1)))
model.add(LSTM(100, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))
```

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

In [ ]:
```
#Train the model
model.fit(x_train, y_train, batch_size=1, epochs=1)
```

Out[ ]:
```
1947/1947 [==============================] - 25s 12ms/step - loss: 1.8612e-04
<keras.callbacks.History at 0x21dc3442a10>
```

In [ ]:
```
#Create the testing data set
#Create a new array containing scaled values from index 1543 to 2003
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])
```

In [ ]:
```
#Convert the data to numpy
x_test = np.array(x_test)
```

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

In [ ]:
```
#Get the models predicted price values
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
```

```
16/16 [==============================] - 0s 18ms/step
```

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

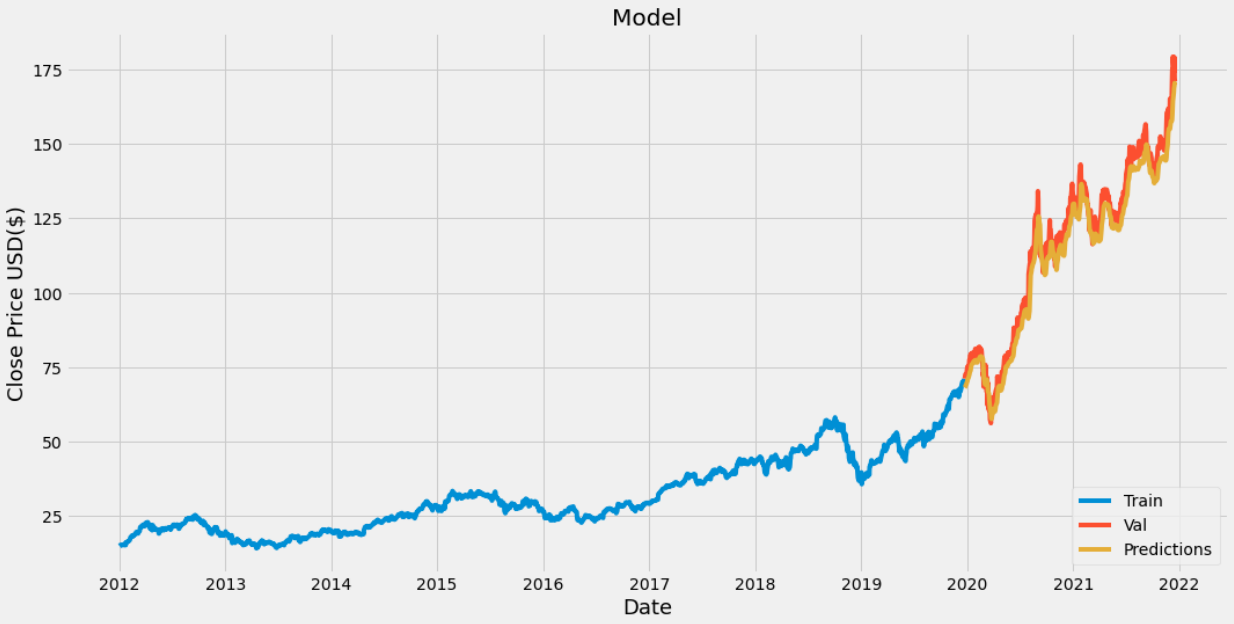
Out[ ]:
```
4.13589909262286
```

In [ ]:
```
#Plot the data
train = data[:training_data_len]
valid = data[training_data_len:]
valid['Predictions'] = predictions

#Visualize the model
plt.figure(figsize=(16,8))
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()
```

```
C:\Users\MONSTER\AppData\Local\Temp\ipykernel_2956\2668772349.py:4: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  valid['Predictions'] = predictions
```



In [ ]:
```
#Show predicted prices
valid
```

Out[ ]:

Date	Close	Predictions
2019-12-24	71.067497	73.969574
2019-12-26	72.477501	74.321930
2019-12-27	72.449997	74.803780
2019-12-30	72.879997	75.297401
2019-12-31	73.412498	75.796692
...	...	...
2021-12-13	175.740005	187.112091
2021-12-14	174.330002	189.152939
2021-12-15	179.300003	190.457504
2021-12-16	172.259995	191.920700
2021-12-17	171.139999	192.262436

501 rows × 2 columns

In [ ]:
```
#Get the quote
apple_quote = web.DataReader('AAPL', data_source='yahoo', start='2012-01-01', end='202
#Create a new dataframe
new_df = apple_quote.filter(['Close'])
#Get the last 60 day closing price values and convert the dataframe to an array
last_60_days = new_df[-60:].values
#Scale the data to be values between 0 and 1
last_60_days_scaled = scaler.transform(last_60_days)
#Create an empty list
x_test = []
#Append the past 60 days
x_test.append(last_60_days_scaled)
#Convert the x_test to numpy
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 predicted scale price
pred_price = model.predict(x_test)
#undo the scaling
pred_price = scaler.inverse_transform(pred_price)
print(pred_price)
```

```
1/1 [==============================] - 0s 14ms/step
[[191.88269]]
```

In [ ]:
```
apple_quote2 = web.DataReader('AAPL', data_source='yahoo', start='2021-12-17', end='20
print(apple_quote2['Close'])
```

```
Date
2021-12-17    171.139999
Name: Close, dtype: float64
```