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='2019-12-17')
df
```

Out[ ]:

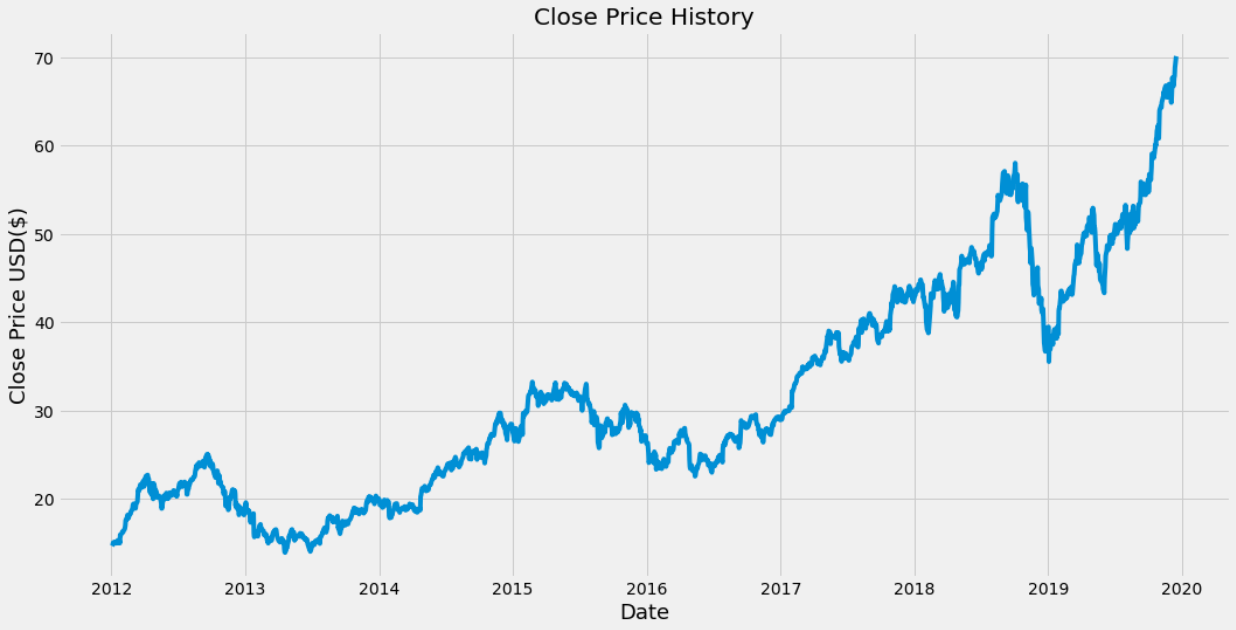
Date	High	Low	Open	Close	Volume	Adj Close
2012-01-03	14.732143	14.607143	14.621429	14.686786	302220800.0	12.557467
2012-01-04	14.810000	14.617143	14.642857	14.765714	260022000.0	12.624949
2012-01-05	14.948214	14.738214	14.819643	14.929643	271269600.0	12.765110
2012-01-06	15.098214	14.972143	14.991786	15.085714	318292800.0	12.898557
2012-01-09	15.276786	15.048214	15.196429	15.061786	394024400.0	12.878095
...	...	...	...	...	...	...
2019-12-11	67.775002	67.125000	67.202499	67.692497	78756800.0	66.519081
2019-12-12	68.139999	66.830002	66.945000	67.864998	137310400.0	66.688614
2019-12-13	68.824997	67.732498	67.864998	68.787498	133587600.0	67.595131
2019-12-16	70.197502	69.245003	69.250000	69.964996	128186000.0	68.752190
2019-12-17	70.442497	69.699997	69.892502	70.102501	114158400.0	68.887321

2003 rows × 6 columns

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

Out[ ]:
```
(2003, 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[ ]:
```
1603
```

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

scaled_data
```

Out[ ]:
```
array([[0.01316509],
       [0.01457063],
       [0.01748985],
       ...,
       [0.97658263],
       [0.99755134],
       [1.        ]])
```

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([[0.01316509],
       [0.01457063],
       [0.01748985],
       [0.02026915],
       [0.01984303],
       [0.02080338],
       [0.02036454],
       [0.01962679],
       [0.01862191],
       [0.02173194],
       [0.02453668],
       [0.02367172],
       [0.01893355],
       [0.02345548],
       [0.01900352],
       [0.03569838],
       [0.03440732],
       [0.0360927 ],
       [0.03973694],
       [0.04194384],
       [0.0417594 ],
       [0.0410789 ],
       [0.04397903],
       [0.04670744],
       [0.04979839],
       [0.05479095],
       [0.0652785 ],
       [0.06543749],
       [0.07127594],
       [0.07563885],
       [0.06814049],
       [0.07102789],
       [0.07097066],
       [0.07906688],
       [0.07791571],
       [0.08004628],
       [0.08387497],
       [0.08600558],
       [0.09214292],
       [0.09661394],
       [0.09790501],
       [0.09835659],
       [0.09071194],
       [0.08886753],
       [0.08914103],
       [0.09632778],
       [0.09835024],
       [0.10269409],
       [0.11293358],
       [0.12659476],
       [0.12403805],
       [0.1240444 ],
       [0.13392141],
       [0.13701237],
       [0.13481179],
       [0.13280207],
       [0.13070964],
       [0.13766105],
       [0.14243103],
       [0.14442805]])]
```

```
[0.13949272033425864]

[array([[0.01316509],
       [0.01457063],
       [0.01748985],
       [0.02026915],
       [0.01984303],
       [0.02080338],
       [0.02036454],
       [0.01962679],
       [0.01862191],
       [0.02173194],
       [0.02453668],
       [0.02367172],
       [0.01893355],
       [0.02345548],
       [0.01900352],
       [0.03569838],
       [0.03440732],
       [0.0360927 ],
       [0.03973694],
       [0.04194384],
       [0.0417594 ],
       [0.0410789 ],
       [0.04397903],
       [0.04670744],
       [0.04979839],
       [0.05479095],
       [0.0652785 ],
       [0.06543749],
       [0.07127594],
       [0.07563885],
       [0.06814049],
       [0.07102789],
       [0.07097066],
       [0.07906688],
       [0.07791571],
       [0.08004628],
       [0.08387497],
       [0.08600558],
       [0.09214292],
       [0.09661394],
       [0.09790501],
       [0.09835659],
       [0.09071194],
       [0.08886753],
       [0.08914103],
       [0.09632778],
       [0.09835024],
       [0.10269409],
       [0.11293358],
       [0.12659476],
       [0.12403805],
       [0.1240444 ],
       [0.13392141],
       [0.13701237],
       [0.13481179],
       [0.13280207],
       [0.13070964],
       [0.13766105],
```

```
[0.14243103],
[0.14442805]]), array([[0.01457063],
[0.01748985],
[0.02026915],
[0.01984303],
[0.02080338],
[0.02036454],
[0.01962679],
[0.01862191],
[0.02173194],
[0.02453668],
[0.02367172],
[0.01893355],
[0.02345548],
[0.01900352],
[0.03569838],
[0.03440732],
[0.0360927 ],
[0.03973694],
[0.04194384],
[0.0417594 ],
[0.0410789 ],
[0.04397903],
[0.04670744],
[0.04979839],
[0.05479095],
[0.0652785 ],
[0.06543749],
[0.07127594],
[0.07563885],
[0.06814049],
[0.07102789],
[0.07097066],
[0.07906688],
[0.07791571],
[0.08004628],
[0.08387497],
[0.08600558],
[0.09214292],
[0.09661394],
[0.09790501],
[0.09835659],
[0.09071194],
[0.08886753],
[0.08914103],
[0.09632778],
[0.09835024],
[0.10269409],
[0.11293358],
[0.12659476],
[0.12403805],
[0.1240444 ],
[0.13392141],
[0.13701237],
[0.13481179],
[0.13280207],
[0.13070964],
[0.13766105],
[0.14243103],
[0.14442805],
```

```
       [0.13949272]])]
[0.13949272033425864, 0.13293562570222134]
```

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[ ]:
```
(1543, 60, 1)
```

In [ ]:
```
#Build the LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(x_train.shape[1], 1)))
model.add(LSTM(50, 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[ ]:
```
1543/1543 [==============================] - 18s 11ms/step - loss: 7.0952e-04
<keras.callbacks.History at 0x21d98feef20>
```

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)
```

```
13/13 [==============================] - 1s 12ms/step
```

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

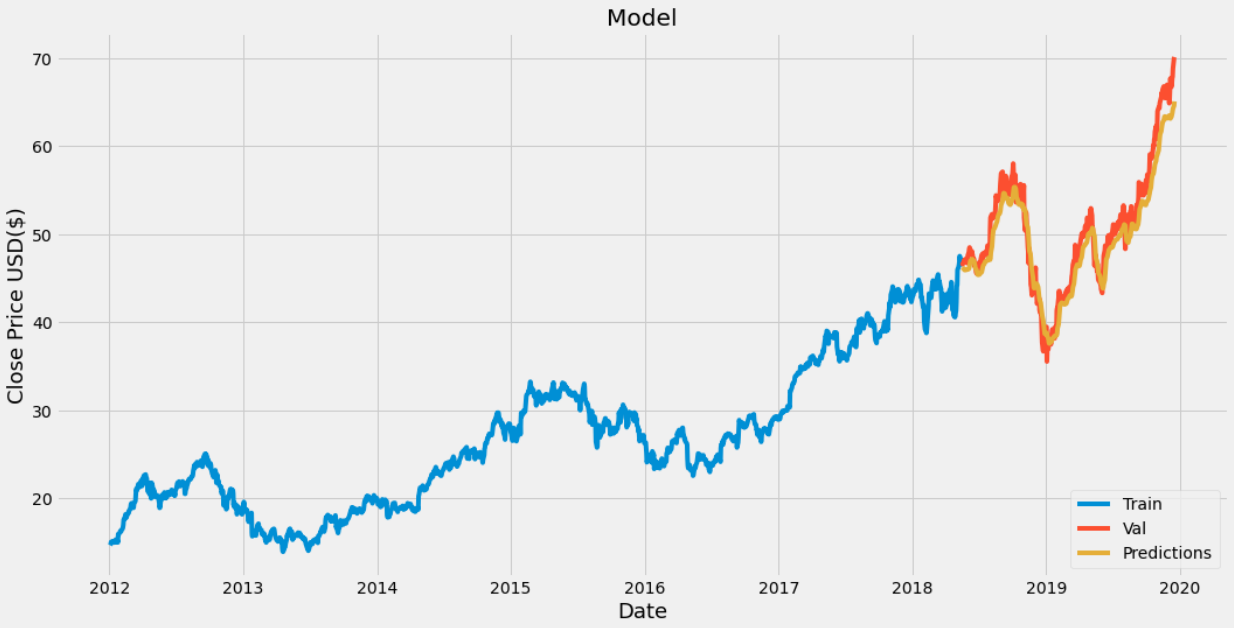
Out[ ]:
```
1.330406436920166
```

In [ ]:
```
#Plot the data
train = data[:training_data_len]
valid = data[trainin_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\1399011820.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
2018-05-17	46.747501	46.229820
2018-05-18	46.577499	46.188049
2018-05-21	46.907501	46.087700
2018-05-22	46.790001	46.009842
2018-05-23	47.090000	45.939613
...	...	...
2019-12-11	67.692497	63.725300
2019-12-12	67.864998	63.976810
2019-12-13	68.787498	64.239098
2019-12-16	69.964996	64.586746
2019-12-17	70.102501	65.074348

400 rows × 2 columns

In [ ]:
```
#Get the quote
apple_quote = web.DataReader('AAPL', data_source='yahoo', start='2012-01-01', end='201
#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 16ms/step
[[65.576645]]
```

In [ ]:
```
apple_quote2 = web.DataReader('AAPL', data_source='yahoo', start='2019-12-18', end='20
print(apple_quote2['Close'])
```

```
Date
2019-12-18    69.934998
Name: Close, dtype: float64
```