Predicting the Stock prices of Tesla using LSTM (neural network) model.

In [69]:
```python
import pandas as pd
import math
import pandas_datareader as dr
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```python
from sklearn.preprocessing import MinMaxScaler
```

In [70]:
```python
import keras
from keras.models import Sequential
from keras.layers import Dense, LSTM
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```python
# Getting the Tesla Stock Data from December 2019 to May 2020
df = dr.data.get_data_yahoo('TSLA', start='2017-12-01', end='2020-05-01')
df
```

Out[4]:

Date	High	Low	Open	Close	Volume	Adj Close
2017-12-01	310.320007	305.049988	305.440002	306.529999	4292900	306.529999
2017-12-04	308.269989	300.609985	306.500000	305.200012	5835100	305.200012
2017-12-05	308.000000	301.000000	302.000000	303.700012	4646500	303.700012
2017-12-06	313.390015	300.000000	300.100006	313.260010	7195300	313.260010
2017-12-07	318.630005	311.049988	312.000000	311.239990	4780600	311.239990
...	...	...	...	...	...	...
2020-04-27	799.489990	735.000000	737.609985	798.750000	20681400	798.750000
2020-04-28	805.000000	756.690002	795.640015	769.119995	15222000	769.119995
2020-04-29	803.200012	783.159973	790.169983	800.510010	16216000	800.510010
2020-04-30	869.820007	763.500000	855.190002	781.880005	28400100	781.880005
2020-05-01	772.770020	683.039978	755.000000	701.320007	32531800	701.320007

607 rows × 6 columns

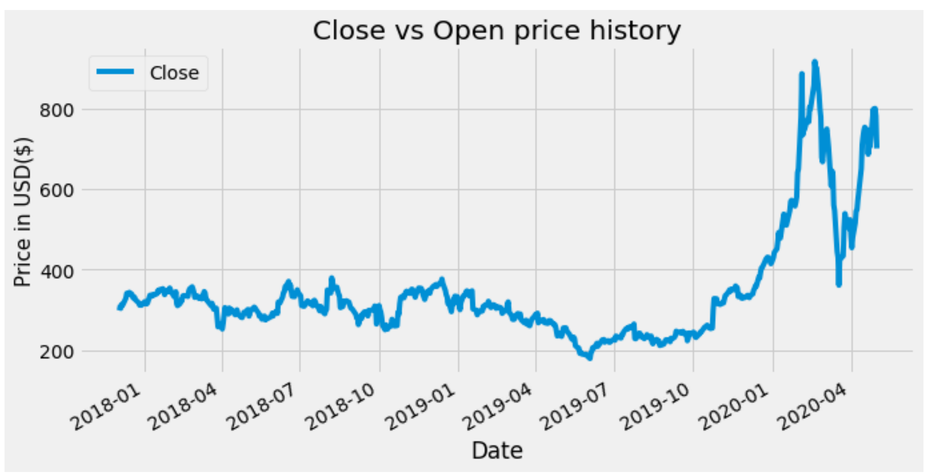
In [5]:
```python
# get the number of rows and columns in the data set
df.shape
```

Out[5]:
```
(607, 6)
```

In [42]:
```python
# Plot for closing and opening prices

df[['Close']].plot(figsize=(10,5));
plt.title('Close vs Open price history')
plt.ylabel('Price in USD($)', fontsize=15)
```

Out[42]:
```
Text(0, 0.5, 'Price in USD($)')
```



In [16]:
```python
# New data frame for just close column

data = df.filter(['Close'])

# Convert data to a numpy array

dataset = data.values

# Splitting the data into test and training data

training_data_len = math.ceil(len(dataset)*0.8)

training_data_len
```

Out[16]:
```
486
```

In [68]:
```python
# scale the data
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)

scaled_data[1:10]
```

Out[68]:
```
array([[0.17093915],
       [0.16890787],
       [0.1818539 ],
       [0.17911841],
       [0.18438622],
       [0.20304693],
       [0.21945968],
       [0.21675131],
       [0.21520755]])
```

In [22]:
```python
# create the training data set

train_data= scaled_data[0:training_data_len,:]
#split 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()
```

```
[array([0.1727402 , 0.17093915, 0.16890787, 0.1818539 , 0.17911841,
       0.18438622, 0.20304693, 0.21945968, 0.21675131, 0.21520755,
       0.22273683, 0.21653463, 0.20601261, 0.20314173, 0.20677095,
       0.19802291, 0.18731127, 0.17966012, 0.18469766, 0.1792674 ,
       0.19169883, 0.1872571 , 0.18369557, 0.18634977, 0.21320334,
       0.20951995, 0.21102308, 0.2152888 , 0.21294604, 0.21814612,
       0.22776086, 0.22425352, 0.23163382, 0.23371928, 0.23538494,
       0.22604106, 0.21486901, 0.22192431, 0.23097028, 0.22594625,
       0.23744329, 0.23059111, 0.22314307, 0.20876161, 0.20989912,
       0.22483581, 0.18452165, 0.17800801, 0.18519874, 0.19593744,
       0.19410928, 0.21003455, 0.21195747, 0.21098245, 0.20899179,
       0.22642023, 0.23438282, 0.24165484, 0.23294738, 0.22220868])]
[0.20578237554751974]

[array([0.1727402 , 0.17093915, 0.16890787, 0.1818539 , 0.17911841,
       0.18438622, 0.20304693, 0.21945968, 0.21675131, 0.21520755,
       0.22273683, 0.21653463, 0.20601261, 0.20314173, 0.20677095,
       0.19802291, 0.18731127, 0.17966012, 0.18469766, 0.1792674 ,
       0.19169883, 0.1872571 , 0.18369557, 0.18634977, 0.21320334,
       0.20951995, 0.21102308, 0.2152888 , 0.21294604, 0.21814612,
       0.22776086, 0.22425352, 0.23163382, 0.23371928, 0.23538494,
       0.22604106, 0.21486901, 0.22192431, 0.23097028, 0.22594625,
       0.23744329, 0.23059111, 0.22314307, 0.20876161, 0.20989912,
       0.22483581, 0.18452165, 0.17800801, 0.18519874, 0.19593744,
       0.19410928, 0.21003455, 0.21195747, 0.21098245, 0.20899179,
       0.22642023, 0.23438282, 0.24165484, 0.23294738, 0.22220868]), array([0.17093915, 0.16890787, 0.1818539 , 0.179
11841, 0.18438622,
       0.20304693, 0.21945968, 0.21675131, 0.21520755, 0.22273683,
       0.21653463, 0.20601261, 0.20314173, 0.20677095, 0.19802291,
       0.18731127, 0.17966012, 0.18469766, 0.1792674 , 0.19169883,
       0.1872571 , 0.18369557, 0.18634977, 0.21320334, 0.20951995,
       0.21102308, 0.2152888 , 0.21294604, 0.21814612, 0.22776086,
       0.22425352, 0.23163382, 0.23371928, 0.23538494, 0.22604106,
       0.21486901, 0.22192431, 0.23097028, 0.22594625, 0.23744329,
       0.23059111, 0.22314307, 0.20876161, 0.20989912, 0.22483581,
       0.18452165, 0.17800801, 0.18519874, 0.19593744, 0.19410928,
       0.21003455, 0.21195747, 0.21098245, 0.20899179, 0.22642023,
       0.23438282, 0.24165484, 0.23294738, 0.22220868, 0.20578238])]
[0.20578237554751974, 0.21145642598465322]
```

In [23]:
```python
# convert the x_train and y_train to numpy arrays
x_train, y_train= np.array(x_train), np.array(y_train)
```

In [24]:
```python
# Reshaping th data
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1],1))
x_train.shape
```

Out[24]:
```
(426, 60, 1)
```

In [27]:
```python
# Building 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 [28]:
```python
# compliing the model

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

In [29]:
```python
# training the model

model.fit(x_train, y_train, batch_size=1, epochs=1)
```

```
Epoch 1/1
426/426 [==============================] - 29s 69ms/step - loss: 0.0010
```

Out[29]:
```
<keras.callbacks.callbacks.History at 0x7ff84f593950>
```

In [30]:
```python
# create the testing data set
# new array containing scaled values from index 1543 to 2003
test_data = scaled_data[training_data_len - 60 : , :]
# Create data set 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 [31]:
```python
# convert data to a numpy array
x_test = np.array(x_test)
```

In [33]:
```python
#reshape the data
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1],1))
```

In [34]:
```python
# get model predictions
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
```

In [36]:
```python
# get the root mean squared error (RMSE)
rmse = np.sqrt(np.mean(predictions-y_test)**2)
rmse
```

Out[36]:
```
18.356471038061724
```

In [71]:
```python
# plot the data

train=data[:training_data_len]
valid = data[training_data_len:]
valid['predictions']= predictions

# visualizing the data
plt.figure(figsize=(16,8))
plt.title('model')
plt.xlabel('Date',fontsize=16)
plt.ylabel('close prices ($USD)')
plt.plot(train['Close'])
plt.plot(valid[['Close', 'predictions']])
plt.legend(['Train', 'val', 'predections'], loc='lower right')
plt.show()
import warnings; warnings.simplefilter('ignore')
import warnings
warnings.filterwarnings('ignore')
```

In [45]:
```python
# show the valid vs the predicted prices
valid
```

Out[45]:

Date	Close	predictions
2019-11-07	335.540009	311.378845
2019-11-08	337.140015	314.954529
2019-11-11	345.089996	318.816101
2019-11-12	349.929993	323.349457
2019-11-13	346.109985	328.224701
...	...	...
2020-04-27	798.750000	713.849304
2020-04-28	769.119995	726.460449
2020-04-29	800.510010	737.155823
2020-04-30	781.880005	750.484741
2020-05-01	701.320007	760.048401

121 rows × 2 columns

In [54]:
```python
TSLA_quote = dr.data.get_data_yahoo('TSLA', start='2017-12-01', end='2020-05-01')
#create new data frame
new_df = TSLA_quote.filter(['Close'])
# Get the last 60 days price values and convert dataframe to array
last_60_days = new_df[-60:].values
# scale the data between 0 & 1
last_60_days_scaled = scaler.transform(last_60_days)
# create empty list
X_test = []
# append last 60 days
X_test.append(last_60_days_scaled)
# convert x into a numpy array
X_test= np.array(X_test)
# reshape the array
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1],1))
# getting the predicted scaled prices
pred_price = model.predict(X_test)
# Undo the scaling
pred_price = scaler.inverse_transform(pred_price)
print(pred_price)
```

```
[[754.6718]]
```

In [65]:
```python
TSLA_quote2 = dr.data.get_data_yahoo('TSLA', start='2020-05-04', end='2020-05-04')
print(TSLA_quote2['Close'])
```

```
Date
2020-05-04    761.190002
Name: Close, dtype: float64
```

From the following results we can see our prediction of 754 dollars is not too far from the actual price of 761 on May 4th, 2020. Therefore, we can say that the LSTM model does a good job in making our predictions.