NETFLIX STOCK PREDICTION USING ML and LSTM

RISHAV KUMAR

rishavkumar9713@gmail.com

In [1]:
```
# importing all important libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
import os
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
import keras.backend as K
from keras.optimizers import Adam
from keras.models import load_model
from keras.layers import LSTM
np.random.seed(7)
```

Data exploration

In [2]:
```
# data is of netflix from date-(1-aug-2003)_to_(28-aug-2020) from yahoo finance
df = pd.read_csv(r'C:\Users\Lazy-dx\Documents\Python-Projects\Stock-predection/NFLX.csv', header=0)
df = df.sort_index(ascending=True, axis=0)
df.head()
```

Out[2]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2003-08-01	1.851429	1.857143	1.782143	1.782143	1.782143	6339200
1	2003-08-04	1.782143	1.797143	1.735714	1.782143	1.782143	3676400
2	2003-08-05	1.792857	1.817857	1.732857	1.736429	1.736429	3084200
3	2003-08-06	1.742857	1.750000	1.633571	1.668571	1.668571	8113000
4	2003-08-07	1.650714	1.667857	1.614286	1.649286	1.649286	7893200

In [3]:
```
df.shape
```

Out[3]:
```
(4299, 7)
```

In [4]:
```
df.describe()
```

Out[4]:

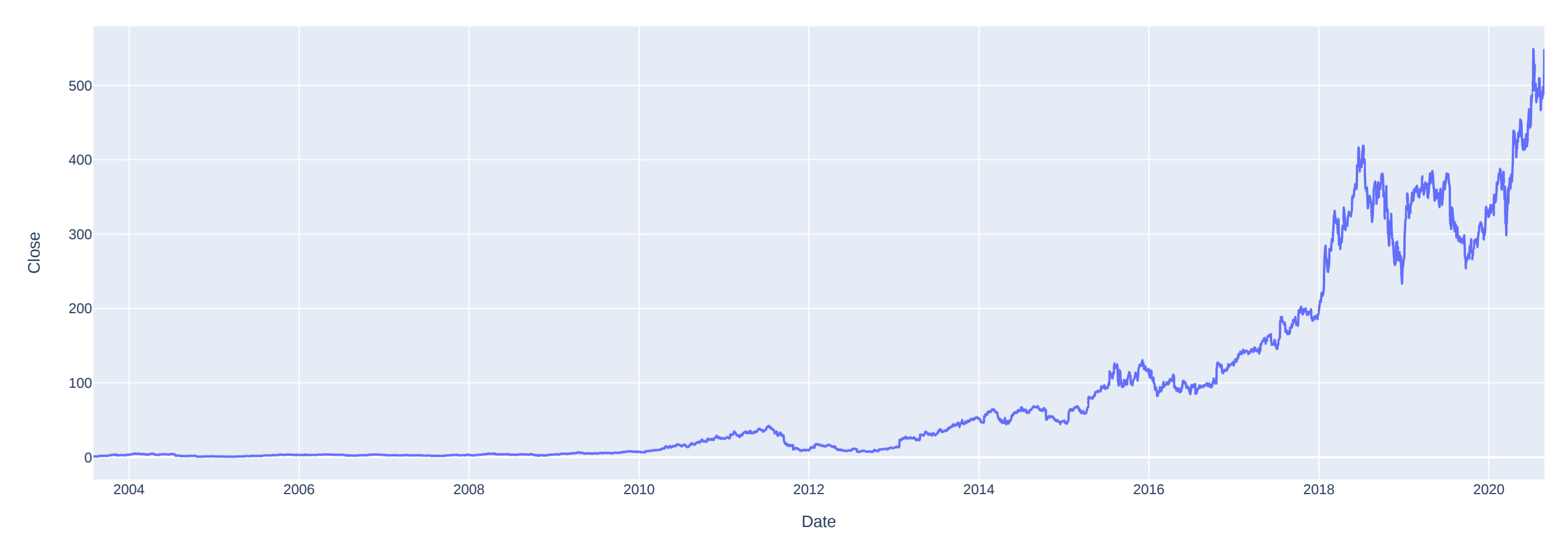
	Open	High	Low	Close	Adj Close	Volume
count	4299.000000	4299.000000	4299.000000	4299.000000	4299.000000	4.299000e+03
mean	85.137456	86.506299	83.731468	85.187151	85.187151	1.816332e+07
std	123.252283	125.243608	121.193015	123.337870	123.337870	2.003304e+07
min	1.300000	1.317143	1.272857	1.290000	1.290000	1.493800e+06
25%	4.143572	4.220000	4.067858	4.142857	4.142857	7.197400e+06
50%	22.384285	22.858572	21.722857	22.245714	22.245714	1.196440e+07
75%	105.120003	107.360000	102.685001	105.280002	105.280002	2.165660e+07
max	567.979980	575.369995	521.250000	548.729980	548.729980	3.234140e+08

In [5]:
```
df.isnull().any()
```

Out[5]:
```
Date         False
Open         False
High         False
Low          False
Close        False
Adj Close    False
Volume       False
dtype: bool
```

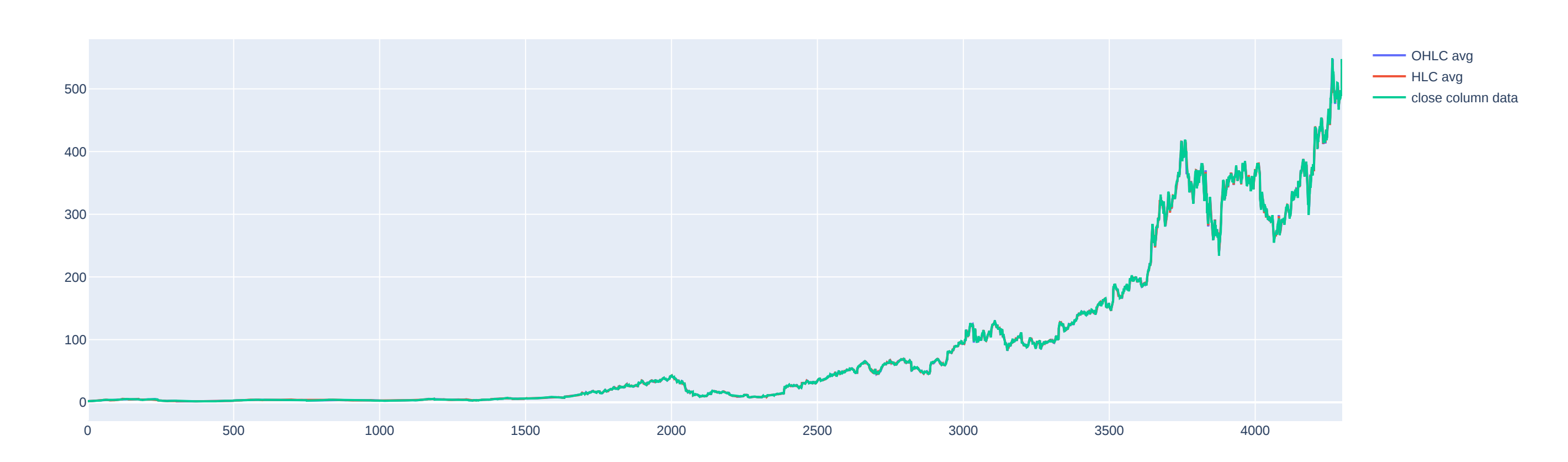
Time series graph of data

In [6]:
```
fig = px.line(df, x='Date', y='Close')
fig.show()
```



In [7]:
```
# Taking diff indicators for prediction
# ohlc_avg is the average of open, high, low, close values
# hlc_avgs is the average of high, low, close value
# we will take only ohlc_avg data only in whole nb
ohlc_data = df.iloc[:, 1:5]
ohlc_avg = ohlc_data.mean(axis=1)
hlc_avg = df[['High', 'Low', 'Close']].mean(axis=1)
close = df.Close
```

In [8]:
```
fig1 = go.Figure()

fig1.add_trace(go.Scatter(x = df.index, y = ohlc_avg,
                  name='OHLC avg'))
fig1.add_trace(go.Scatter(x = df.index, y = hlc_avg,
                  name='HLC avg'))
fig1.add_trace(go.Scatter(x = df.index, y = close,
                  name='close column data'))
fig1.show()
```



In [9]:
```
pip install -U kaleido
```

```
Requirement already satisfied: kaleido in c:\users\lazy-dx\anaconda3\lib\site-packages (0.2.1)
Note: you may need to restart the kernel to use updated packages.
```

In [10]:
```
if not os.path.exists("images"):
    os.mkdir("images")
```

In [35]:
```
fig1.write_image("/Users/Lazy-dx/Documents/Python-Projects/Stock-predection/images/previous_stock_graph.png")
```

In [11]:
```
# we will create a new df which has only 2 column which is useful to predict data

new_data = pd.DataFrame(index=range(0,len(df)), columns=['Date', 'ohlc_avg'])
for i in range(0, len(df)):
  new_data['Date'][i] = df['Date'][i]
  new_data['ohlc_avg'][i] = ohlc_avg[i]
```

In [12]:
```
new_data.head()
```

Out[12]:

	Date	ohlc_avg
0	2003-08-01	1.818214
1	2003-08-04	1.774286
2	2003-08-05	1.77
3	2003-08-06	1.69875
4	2003-08-07	1.645536

In [13]:
```
# setting index
new_data.index = new_data.Date
new_data.drop('Date', axis=1, inplace=True)
```

In [14]:
```
print(len(new_data))
```

```
4299
```

In [15]:
```
ds = new_data.values
```

In [16]:
```
# we will take 80% data in train and remaining in test
train = int(len(new_data)*0.8)
test = len(new_data) - train
train, test = new_data.iloc[0:train,:], new_data.iloc[train:len(new_data),:]
```

In [17]:
```
train.shape
```

Out[17]:
```
(3439, 1)
```

In [18]:
```
test.shape
```

Out[18]:
```
(860, 1)
```

Normalizing data

In [19]:
```
# we have normalize the data cuz data is like 149...., 488..something like that
# so we have to normalize betwwen 0 and 1
scalar = MinMaxScaler(feature_range=(0, 1))
scaled_data = scalar.fit_transform(ds)
```

splitting the data into x_train, y_train

In [20]:
```
# splitting the data to x_train, y_train
# we will first train upto 60 and then predict on 61 and then
# we will train from 61 to 120 then predict on 121 likewise we will go
x_train, y_train = [], []
for i in range(60, len(train)):
  x_train.append(scaled_data[i-60:i,0])
  y_train.append(scaled_data[i,0])

x_train, y_train = np.array(x_train), np.array(y_train)
```

In [21]:
```
# now we have reshape the array to 3-d to pass the data into lstm [number of samples, time steps/batch_size, features]
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```

In [22]:
```
x_train.shape
```

Out[22]:
```
(3379, 60, 1)
```

Modelling

In [23]:
```
# create and fit the lstm network
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1], 1)))
model.add(Dropout(0.25))
model.add(LSTM(units=50))
model.add(Dense(1))
model.add(Activation('linear'))
model.compile(loss='mean_squared_error', optimizer='adam')
```

In [24]:
```
model.fit(x_train, y_train, epochs=50, batch_size=32, verbose=1)
```

```
Epoch 1/50
106/106 [==============================] - 13s 30ms/step - loss: 2.4677e-04
Epoch 2/50
106/106 [==============================] - 3s 29ms/step - loss: 4.8687e-05
Epoch 3/50
106/106 [==============================] - 3s 29ms/step - loss: 4.0926e-05
Epoch 4/50
106/106 [==============================] - 3s 29ms/step - loss: 3.8484e-05
Epoch 5/50
106/106 [==============================] - 3s 29ms/step - loss: 3.4244e-05
Epoch 6/50
106/106 [==============================] - 3s 30ms/step - loss: 3.9540e-05
Epoch 7/50
106/106 [==============================] - 3s 29ms/step - loss: 3.4566e-05
Epoch 8/50
106/106 [==============================] - 3s 29ms/step - loss: 3.2510e-05
Epoch 9/50
106/106 [==============================] - 3s 29ms/step - loss: 2.8029e-05
Epoch 10/50
106/106 [==============================] - 3s 29ms/step - loss: 3.1069e-05
Epoch 11/50
106/106 [==============================] - 3s 29ms/step - loss: 2.9622e-05
Epoch 12/50
106/106 [==============================] - 3s 29ms/step - loss: 3.1025e-05
Epoch 13/50
106/106 [==============================] - 3s 30ms/step - loss: 3.0431e-05
Epoch 14/50
106/106 [==============================] - 3s 30ms/step - loss: 2.8284e-05
Epoch 15/50
106/106 [==============================] - 3s 29ms/step - loss: 2.4488e-05
Epoch 16/50
106/106 [==============================] - 3s 29ms/step - loss: 2.7021e-05
Epoch 17/50
106/106 [==============================] - 3s 29ms/step - loss: 2.9045e-05
Epoch 18/50
106/106 [==============================] - 3s 29ms/step - loss: 2.6615e-05
Epoch 19/50
106/106 [==============================] - 3s 29ms/step - loss: 2.6352e-05
Epoch 20/50
106/106 [==============================] - 3s 29ms/step - loss: 2.3865e-05
Epoch 21/50
106/106 [==============================] - 3s 29ms/step - loss: 2.4812e-05
Epoch 22/50
106/106 [==============================] - 3s 29ms/step - loss: 2.4211e-05
Epoch 23/50
106/106 [==============================] - 3s 29ms/step - loss: 2.5494e-05
Epoch 24/50
106/106 [==============================] - 3s 30ms/step - loss: 2.6021e-05
Epoch 25/50
106/106 [==============================] - 3s 29ms/step - loss: 2.2670e-05
Epoch 26/50
106/106 [==============================] - 3s 29ms/step - loss: 2.1758e-05
Epoch 27/50
106/106 [==============================] - 3s 29ms/step - loss: 2.3548e-05
Epoch 28/50
106/106 [==============================] - 3s 29ms/step - loss: 2.1014e-05
Epoch 29/50
106/106 [==============================] - 3s 29ms/step - loss: 2.5895e-05
Epoch 30/50
106/106 [==============================] - 3s 29ms/step - loss: 2.1331e-05
Epoch 31/50
106/106 [==============================] - 3s 29ms/step - loss: 2.2895e-05
Epoch 32/50
106/106 [==============================] - 3s 29ms/step - loss: 2.0816e-05
Epoch 33/50
106/106 [==============================] - 3s 29ms/step - loss: 2.0259e-05
Epoch 34/50
106/106 [==============================] - 3s 29ms/step - loss: 1.9540e-05
Epoch 35/50
106/106 [==============================] - 3s 30ms/step - loss: 1.8987e-05
Epoch 36/50
106/106 [==============================] - 3s 29ms/step - loss: 1.9991e-05
Epoch 37/50
106/106 [==============================] - 3s 29ms/step - loss: 1.8618e-05
Epoch 38/50
106/106 [==============================] - 3s 29ms/step - loss: 1.9385e-05
Epoch 39/50
106/106 [==============================] - 3s 29ms/step - loss: 1.9008e-05
Epoch 40/50
106/106 [==============================] - 3s 29ms/step - loss: 1.8726e-05
Epoch 41/50
106/106 [==============================] - 3s 29ms/step - loss: 2.0957e-05
Epoch 42/50
106/106 [==============================] - 3s 29ms/step - loss: 1.8326e-05
Epoch 43/50
106/106 [==============================] - 3s 30ms/step - loss: 1.9417e-05
Epoch 44/50
106/106 [==============================] - 3s 29ms/step - loss: 1.7418e-05
Epoch 45/50
106/106 [==============================] - 3s 30ms/step - loss: 1.8561e-05
Epoch 46/50
106/106 [==============================] - 3s 30ms/step - loss: 1.6316e-05
Epoch 47/50
106/106 [==============================] - 3s 29ms/step - loss: 2.1855e-05
Epoch 48/50
106/106 [==============================] - 3s 29ms/step - loss: 1.9422e-05
Epoch 49/50
106/106 [==============================] - 3s 29ms/step - loss: 1.7320e-05
Epoch 50/50
106/106 [==============================] - 3s 29ms/step - loss: 1.6905e-05
```

Out[24]:
```
<keras.callbacks.History at 0x1724179a2e0>
```

Prediction

In [26]:
```
# predicting 920 values, using past 60 from the train data
inputs = new_data[len(new_data)-len(test) - 60:].values
inputs = inputs.reshape(-1,1)
inputs = scalar.transform(inputs)
```

In [27]:
```
inputs.shape
```

Out[27]:
```
(920, 1)
```

In [28]:
```
x_test = []
for i in range(60,inputs.shape[0]):
    x_test.append(inputs[i-60:i,0])
x_test = np.array(x_test)
```

In [29]:
```
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
```

In [30]:
```
predicted_price = model.predict(x_test)
# inverse transform for getting back all normal values from scaled values
predicted_price = scalar.inverse_transform(predicted_price)
```

```
27/27 [==============================] - 2s 16ms/step
```

In [31]:
```
rms=np.sqrt(np.mean(np.power((test-predicted_price),2)))
rms
```

```
C:\Users\Lazy-dx\anaconda3\lib\site-packages\numpy\core\fromnumeric.py:3438: FutureWarning:

In a future version, DataFrame.mean(axis=None) will return a scalar mean over the entire DataFrame. To retain the old behavior, use 'frame.mean(axis=0)' or just 'frame.mean()'
```

Out[31]:
```
ohlc_avg    20.592266
dtype: float64
```

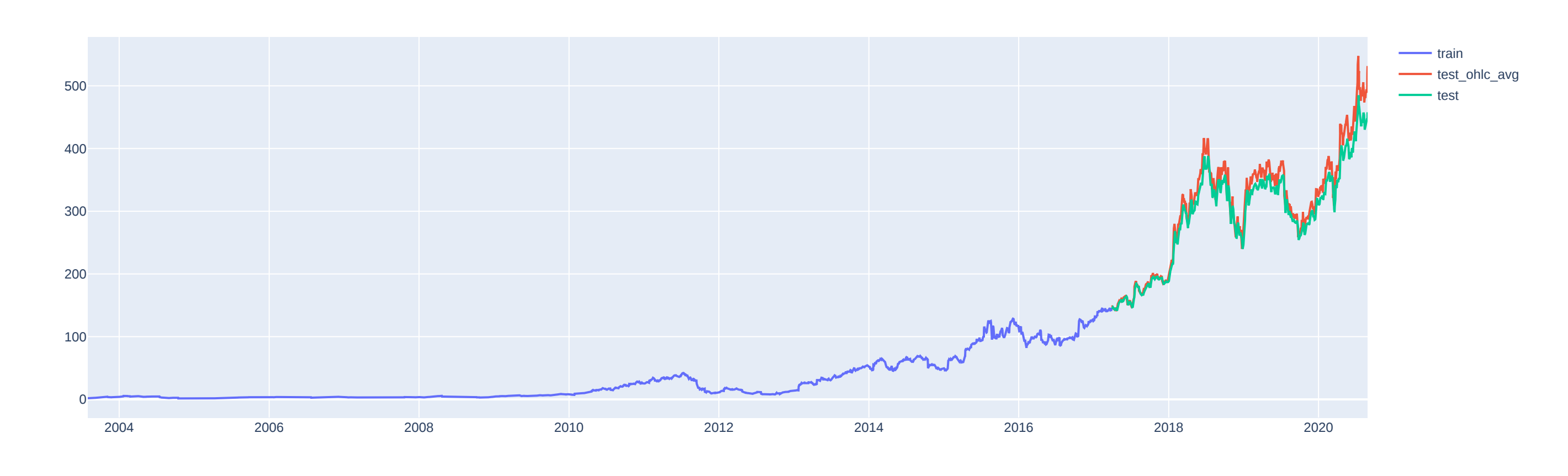
In [32]:
```
# create a new column of predicted values
test['Prediction'] = predicted_price
test.head()
```

```
C:\Users\Lazy-dx\AppData\Local\Temp\ipykernel_11288\2551594717.py:2: SettingWithCopyWarning:


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/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Out[32]:

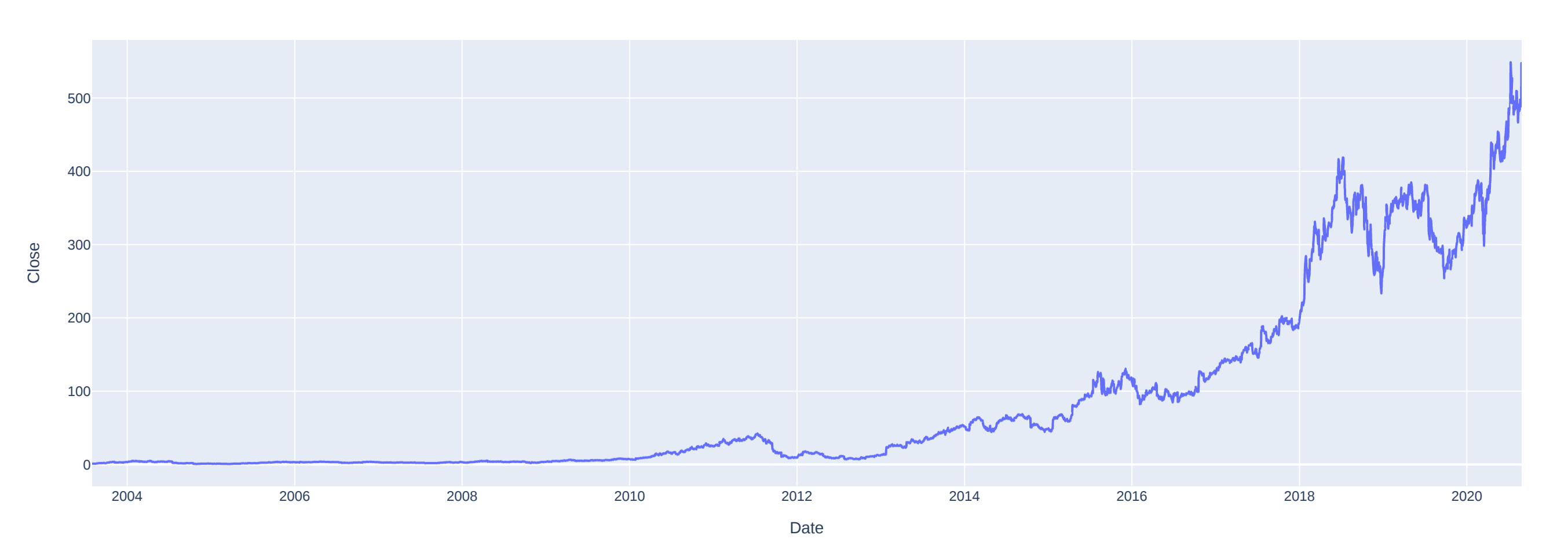
Date	ohlc_avg	Prediction
2017-03-30	147.422497	144.529953
2017-03-31	147.842499	146.021042
2017-04-03	146.842499	147.063400
2017-04-04	146.079998	147.140686
2017-04-05	144.829998	146.596817

In [33]:
```
# Graph for comparing the results of model predicted and original value
fig2 = go.Figure()

fig2.add_trace(go.Scatter(x = train.index, y = train.ohlc_avg,
                  name='train'))
fig2.add_trace(go.Scatter(x = test.index, y = test.ohlc_avg,
                  name='test_ohlc_avg'))
fig2.add_trace(go.Scatter(x = test.index, y = test.Prediction,
                  name='test'))
fig2.show()
```



In [36]:
```
fig2.write_image("/Users/Lazy-dx/Documents/Python-Projects/Stock-predection/images/after_stock_predection-graph.png")
```

In [34]:
```
fig3 = px.line(df, x='Date', y='Close')
fig3.show()
```



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