Stock Price Prediction

About Dataset

Context

The art of forecasting stock prices has been a difficult task for many of the researchers and analysts. In fact, investors are highly interested in the research area of stock price prediction. For a good and successful investment, many investors are keen on knowing the future situation of the stock market. Good and effective prediction systems for the stock market help traders, investors, and analyst by providing supportive information like the future direction of the stock market. In this work, we present a recurrent neural network (RNN) and Long Short-Term Memory (LSTM) approach to predict stock market indices.

Acknowledgements

I want to find relationship between volume and price.

Inspiration

Here is couple of things one could try out with this data:

One day ahead prediction: Rolling Linear Regression, ARIMA, Neural Networks, LSTM Momentum/Mean-Reversion Strategies Security clustering, portfolio construction/hedging

Code:

```
import numpy as np
import pandas as pd

import os
import matplotlib.pyplot as plt
import pandas_datareader as web
import datetime as dt

from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM
from tensorflow.keras.callbacks import ModelCheckpoint,
EarlyStopping
```

```
import matplotlib.dates as dates
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

In [2]:

```
data=pd.read_csv('../input/netflix-stock-price/NFLX.csv')
data[0:3]
```

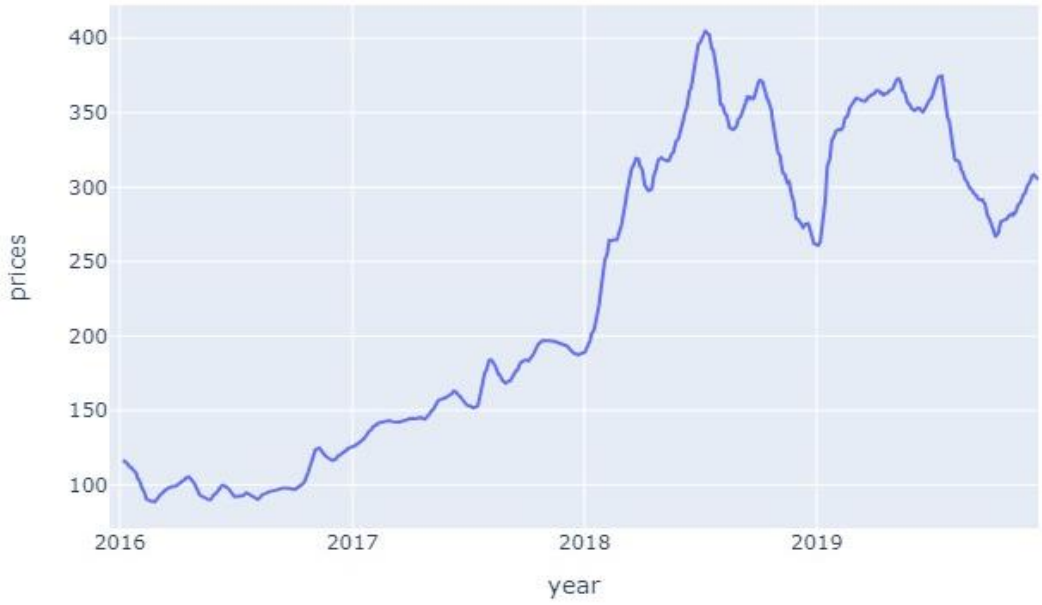
Out[2]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2015-12-16	119.800003	123.000000	118.089996	122.639999	122.639999	13181000
1	2015-12-17	123.970001	126.349998	122.419998	122.510002	122.510002	17284900
2	2015-12-18	120.849998	122.190002	117.919998	118.019997	118.019997	17948100

In [3]:

```
fig=make_subplots(specs=[[{"secondary_y":False}]])
fig.add_trace(go.Scatter(x=data['Date'],y=data['Open'].rolling(windo
w=14).mean(),name="netflix"),secondary_y=False,)
fig.update_layout(autosize=False,width=700,height=500,title_text="NE
TFLIX")
fig.update_xaxes(title_text="year")
fig.update_yaxes(title_text="prices",secondary_y=False)
fig.show()
```

2016201720182019100150200250300350400
NETFLIXyearprices



In [4]:

```
n=len(data)
train_data=data[0:(n//10)*9]
test_data=data[(n//10)*9:]
```

In [5]:

```
test_data[0:3]
```

Out[5]:

	Date	Open	High	Low	Close	Adj Close	Volume
900	2019-07-17	366.250000	366.500000	361.750000	362.440002	362.440002	13639500
901	2019-07-18	323.760010	329.850006	320.299988	325.209991	325.209991	31305900
902	2019-07-19	323.399994	325.850006	314.230011	315.100006	315.100006	16302500

In [6]:

```
print(len(train_data))
print(len(test_data))
```

```
900
107
```

In [7]:

```
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data =
scaler.fit_transform(train_data['Open'].values.reshape(-1,1))
```

In [8]:

```
prediction_days = 30

x_train = []
y_train = []

for x in range(prediction_days, len(scaled_data)-10):      ######
    x_train.append(scaled_data[x-prediction_days:x, 0])
    y_train.append(scaled_data[x+10, 0])      ###### predict 10 days
after

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

```
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1],
1))
```

In [9]:

```
print(x_train.shape)
print(y_train.shape)
```

```
(860, 30, 1)
(860,)
```

In [10]:

```
def LSTM_model():

    model = Sequential()
    model.add(LSTM(units = 50, return_sequences = True, input_shape
= (x_train.shape[1],1)))
    model.add(Dropout(0.2))
    model.add(LSTM(units = 50, return_sequences = True))
    model.add(Dropout(0.2))
    model.add(LSTM(units = 50))
    model.add(Dropout(0.2))
    model.add(Dense(units=1))

    return model
```

In [11]:

```
model = LSTM_model()
model.summary()
model.compile(optimizer='adam', loss='mean_squared_error', metrics =
['accuracy'])   ###
```

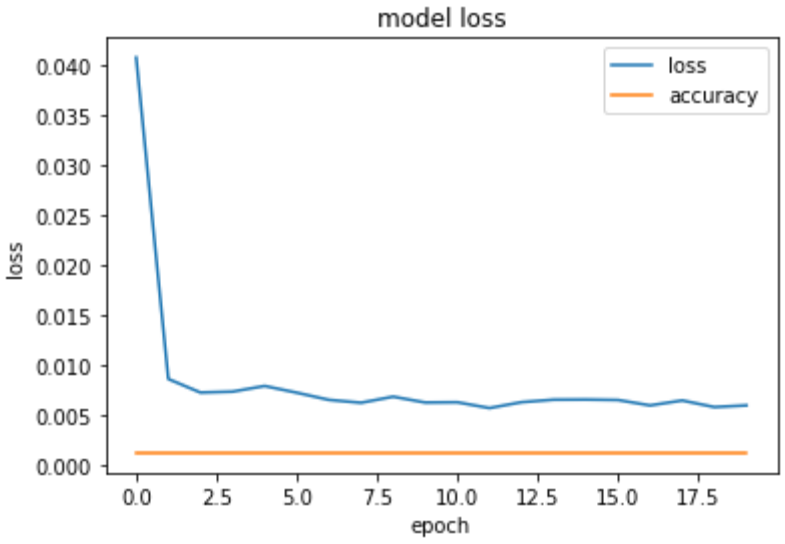
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 30, 50)            10400
_________________________________________________________________
dropout (Dropout)            (None, 30, 50)            0
_________________________________________________________________
lstm_1 (LSTM)                (None, 30, 50)            20200
_________________________________________________________________
dropout_1 (Dropout)          (None, 30, 50)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 50)                20200
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 51
```

```
=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
In [12]:

checkpointer = ModelCheckpoint(filepath = 'weights_best.hdf5',
verbose = 1, save_best_only = True)
his=model.fit(x_train,y_train,epochs=20,batch_size=32,callbacks=[che
ckpointer])
Epoch 1/20
27/27 [==============================] - 6s 40ms/step - loss: 0.0851
- accuracy: 1.7063e-04
Epoch 2/20
27/27 [==============================] - 1s 38ms/step - loss: 0.0075
- accuracy: 8.3561e-04
Epoch 3/20
27/27 [==============================] - 1s 38ms/step - loss: 0.0072
- accuracy: 8.3561e-04
Epoch 4/20
27/27 [==============================] - 1s 39ms/step - loss: 0.0068
- accuracy: 0.0021
Epoch 5/20
27/27 [==============================] - 1s 38ms/step - loss: 0.0081
- accuracy: 0.0044
Epoch 6/20
27/27 [==============================] - 1s 38ms/step - loss: 0.0073
- accuracy: 9.2146e-04
Epoch 7/20
27/27 [==============================] - 1s 38ms/step - loss: 0.0068
- accuracy: 4.2533e-04
Epoch 8/20
27/27 [==============================] - 1s 40ms/step - loss: 0.0064
- accuracy: 0.0044
Epoch 9/20
27/27 [==============================] - 1s 38ms/step - loss: 0.0072
- accuracy: 9.2146e-04
Epoch 10/20
27/27 [==============================] - 1s 38ms/step - loss: 0.0069
- accuracy: 0.0044
Epoch 11/20
27/27 [==============================] - 1s 38ms/step - loss: 0.0060
- accuracy: 6.8148e-04
Epoch 12/20
27/27 [==============================] - 1s 38ms/step - loss: 0.0062
- accuracy: 6.1173e-04
```

```
Epoch 13/20
27/27 [==============================] - 1s 38ms/step - loss: 0.0060
- accuracy: 4.8407e-04
Epoch 14/20
27/27 [==============================] - 1s 38ms/step - loss: 0.0066
- accuracy: 6.8148e-04
Epoch 15/20
27/27 [==============================] - 1s 38ms/step - loss: 0.0070
- accuracy: 0.0027
Epoch 16/20
27/27 [==============================] - 1s 45ms/step - loss: 0.0067
- accuracy: 0.0033
Epoch 17/20
27/27 [==============================] - 1s 38ms/step - loss: 0.0072
- accuracy: 1.7063e-04
Epoch 18/20
27/27 [==============================] - 1s 38ms/step - loss: 0.0067
- accuracy: 2.6565e-04
Epoch 19/20
27/27 [==============================] - 1s 39ms/step - loss: 0.0058
- accuracy: 8.3056e-05
Epoch 20/20
27/27 [==============================] - 1s 38ms/step - loss: 0.0060
- accuracy: 4.8407e-04
In [13]:
```

```
plt.plot(his.history['loss'])
plt.plot(his.history['accuracy'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['loss','accuracy'], loc='upper right')
plt.show()
```



In [14]:

```
actual_prices = test_data['Open'].values
total_dataset = pd.concat((train_data['Open'], test_data['Open']),
axis=0)

model_inputs = total_dataset[len(total_dataset)-len(test_data)-
prediction_days:].values
model_inputs = model_inputs.reshape(-1,1)
model_inputs = scaler.transform(model_inputs)
```

In [15]:

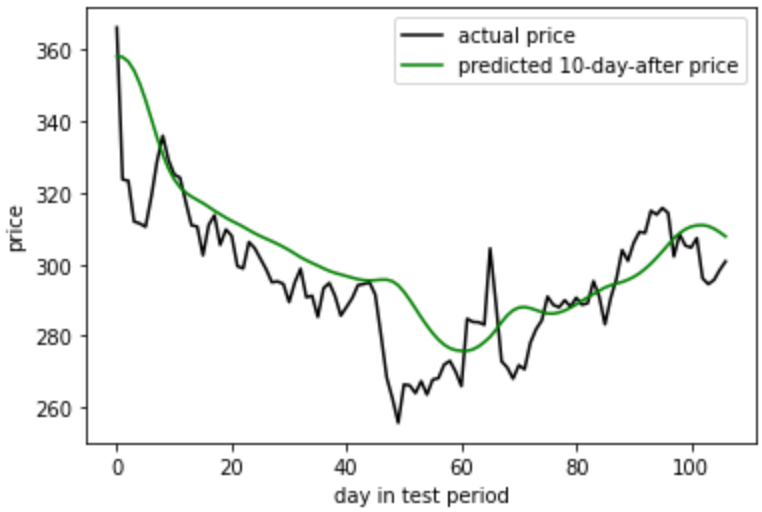
```
x_test = []
for x in range(prediction_days,len(model_inputs)):
    x_test.append(model_inputs[x-prediction_days:x,0])

x_test = np.array(x_test)
x_test = np.reshape(x_test,(x_test.shape[0],x_test.shape[1],1))

predicted_prices = model.predict(x_test)
predicted_prices = scaler.inverse_transform(predicted_prices)
```

In [16]:

```
plt.plot(actual_prices, color='black', label=f"actual price")
plt.plot(predicted_prices, color= 'green', label=f"predicted 10-day-
after price")
plt.title(f"Stock")
plt.xlabel("day in test period")
plt.ylabel(f"price")
plt.legend()
plt.show()
```



In [17]:

```
real_data = [model_inputs[len(model_inputs)+1-
prediction_days:len(model_inputs+1),0]]
real_data = np.array(real_data)
real_data =
np.reshape(real_data,(real_data.shape[0],real_data.shape[1],1))
print(real_data.shape)
```

```
(1, 29, 1)
```

In [18]:

linkcode

```
prediction = model.predict(real_data)
prediction = scaler.inverse_transform(prediction)
print(f"prediction: {prediction[0][0]}")
```

```
prediction: 306.7970886230469
```