Stock Market Prediction And Forecasting Using Stacked LSTM

```
In [ ]: ### Keras and Tensorflow >2.0
```

```
In [403]: ### Data Collection
          import pandas_datareader as pdr
          key=""
```

```
In [404]: df = pdr.get_data_tiingo('AAPL', api_key=key)
```

```
In [283]: df.to_csv('AAPL.csv')
```

```
In [405]: import pandas as pd
```

```
In [406]: df=pd.read_csv('AAPL.csv')
```

```
In [407]: df.head()
```

Out[407]:

	Unnamed: 0	symbol	date	close	high	low	open	volume	adjClose	adjHigh	adjLow	adjOpen	adjV
0	0	AAPL	2015-05-27 00:00:00+00:00	132.045	132.260	130.05	130.34	45833246	121.682558	121.880685	119.844118	120.111360	458
1	1	AAPL	2015-05-28 00:00:00+00:00	131.780	131.950	131.10	131.86	30733309	121.438354	121.595013	120.811718	121.512076	307
2	2	AAPL	2015-05-29 00:00:00+00:00	130.280	131.450	129.90	131.23	50884452	120.056069	121.134251	119.705890	120.931516	508
3	3	AAPL	2015-06-01 00:00:00+00:00	130.535	131.390	130.05	131.20	32112797	120.291057	121.078960	119.844118	120.903870	321
4	4	AAPL	2015-06-02 00:00:00+00:00	129.960	130.655	129.32	129.86	33667627	119.761181	120.401640	119.171406	119.669029	336

In [409]:
```
df.tail()
```

Out[409]:

	Unnamed: 0	symbol	date	close	high	low	open	volume	adjClose	adjHigh	adjLow	adjOpen	adjVolume
1253	1253	AAPL	2020-05-18 00:00:00+00:00	314.96	316.50	310.3241	313.17	33843125	314.96	316.50	310.3241	313.17	33843125
1254	1254	AAPL	2020-05-19 00:00:00+00:00	313.14	318.52	313.0100	315.03	25432385	313.14	318.52	313.0100	315.03	25432385
1255	1255	AAPL	2020-05-20 00:00:00+00:00	319.23	319.52	316.2000	316.68	27876215	319.23	319.52	316.2000	316.68	27876215
1256	1256	AAPL	2020-05-21 00:00:00+00:00	316.85	320.89	315.8700	318.66	25672211	316.85	320.89	315.8700	318.66	25672211
1257	1257	AAPL	2020-05-22 00:00:00+00:00	318.89	319.23	315.3500	315.77	20450754	318.89	319.23	315.3500	315.77	20450754

In [410]:
```
df1=df.reset_index()['close']
```

In [412]:
```
df1
```

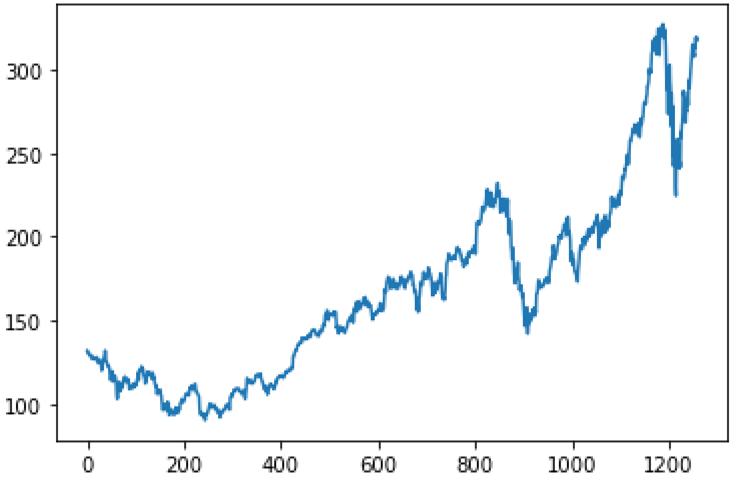
Out[412]:
```
0       132.045
1       131.780
2       130.280
3       130.535
4       129.960
         ...
1253    314.960
1254    313.140
1255    319.230
1256    316.850
1257    318.890
Name: close, Length: 1258, dtype: float64
```

In [413]:

```
import matplotlib.pyplot as plt
plt.plot(df1)
```

Out[413]:

```
[<matplotlib.lines.Line2D at 0x2d1a92724e0>]
```



In [291]:

```
### LSTM are sensitive to the scale of the data. so we apply MinMax scaler
```

In [292]:

```
import numpy as np
```

In [414]:

```
df1
```

Out[414]:

```
0       132.045
1       131.780
2       130.280
3       130.535
4       129.960
          ...
1253    314.960
1254    313.140
1255    319.230
1256    316.850
1257    318.890
Name: close, Length: 1258, dtype: float64
```

In [415]:

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
df1=scaler.fit_transform(np.array(df1).reshape(-1,1))
```

In [417]:

```
print(df1)
```

```
[[0.17607447]
 [0.17495567]
 [0.16862282]
 ...
 [0.96635143]
 [0.9563033 ]
 [0.96491598]]
```

In [418]:

```
##splitting dataset into train and test split
training_size=int(len(df1)*0.65)
test_size=len(df1)-training_size
train_data,test_data=df1[0:training_size,:],df1[training_size:len(df1),:1]
```

In [419]:

```
training_size,test_size
```

Out[419]:

```
(817, 441)
```

```
In [422]: train_data
```

```
Out[422]: array([[0.17607447],
                 [0.17495567],
                 [0.16862282],
                 [0.1696994 ],
                 [0.16727181],
                 [0.16794731],
                 [0.16473866],
                 [0.16174111],
                 [0.1581525 ],
                 [0.15654817],
                 [0.16271215],
                 [0.1614878 ],
                 [0.1554927 ],
                 [0.15443722],
                 [0.15730811],
                 [0.15604154],
                 [0.15849025],
                 [0.15308621],
                 [0.15735033],
```

```
In [423]: import numpy
          # convert an array of values into a dataset matrix
          def create_dataset(dataset, time_step=1):
              dataX, dataY = [], []
              for i in range(len(dataset)-time_step-1):
                  a = dataset[i:(i+time_step), 0]   ###i=0, 0,1,2,3-----99   100
                  dataX.append(a)
                  dataY.append(dataset[i + time_step, 0])
              return numpy.array(dataX), numpy.array(dataY)
```

```
In [424]: # reshape into X=t,t+1,t+2,t+3 and Y=t+4
          time_step = 100
          X_train, y_train = create_dataset(train_data, time_step)
          X_test, ytest = create_dataset(test_data, time_step)
```

In [426]:

```
print(X_train.shape), print(y_train.shape)
```

```
(716, 100)
(716,)
```

Out[426]:

```
(None, None)
```

In [299]:

```
print(X_test.shape), print(ytest.shape)
```

```
(340, 100)
(340,)
```

Out[299]:

```
(None, None)
```

In [427]:

```
# reshape input to be [samples, time steps, features] which is required for LSTM
X_train =X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
X_test = X_test.reshape(X_test.shape[0],X_test.shape[1] , 1)
```

In [428]:

```
### Create the Stacked LSTM model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
```

In [429]:

```
model=Sequential()
model.add(LSTM(50,return_sequences=True,input_shape=(100,1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
```

In [430]:

```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_7 (LSTM)                (None, 100, 50)           10400
_________________________________________________________________
lstm_8 (LSTM)                (None, 100, 50)           20200
_________________________________________________________________
lstm_9 (LSTM)                (None, 50)                20200
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 51
=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
```

In [306]:

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 100, 50)           10400
_________________________________________________________________
lstm_5 (LSTM)                (None, 100, 50)           20200
_________________________________________________________________
lstm_6 (LSTM)                (None, 50)                20200
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 51
=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```

```

In [431]:

```
model.fit(X_train,y_train,validation_data=(X_test,ytest),epochs=100,batch_size=64,verbose=1)
```

```
Epoch 1/100
12/12 [==============================] - 6s 487ms/step - loss: 0.0206 - val_loss: 0.0505
Epoch 2/100
12/12 [==============================] - 4s 309ms/step - loss: 0.0035 - val_loss: 0.0046
Epoch 3/100
12/12 [==============================] - 4s 300ms/step - loss: 0.0014 - val_loss: 0.0040
Epoch 4/100
12/12 [==============================] - 3s 287ms/step - loss: 8.1361e-04 - val_loss: 0.0073
Epoch 5/100
12/12 [==============================] - 3s 290ms/step - loss: 6.6860e-04 - val_loss: 0.0062
Epoch 6/100
12/12 [==============================] - 3s 255ms/step - loss: 6.4653e-04 - val_loss: 0.0062
Epoch 7/100
12/12 [==============================] - 3s 291ms/step - loss: 6.6186e-04 - val_loss: 0.0062
Epoch 8/100
12/12 [==============================] - 4s 300ms/step - loss: 6.2498e-04 - val_loss: 0.0049
Epoch 9/100
12/12 [==============================] - 4s 297ms/step - loss: 6.2745e-04 - val_loss: 0.0042
Epoch 10/100
```

In [37]:

```
import tensorflow as tf
```

In [39]:

```
tf.__version__
```

Out[39]:

```
'2.1.0'
```

In [432]:

```
### Lets Do the prediction and check performance metrics
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
```

In [433]:

```
##Transformback to original form
train_predict=scaler.inverse_transform(train_predict)
test_predict=scaler.inverse_transform(test_predict)
```

In [434]:

```
### Calculate RMSE performance metrics
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(y_train,train_predict))
```

Out[434]:

```
140.9909210035748
```

In [435]:

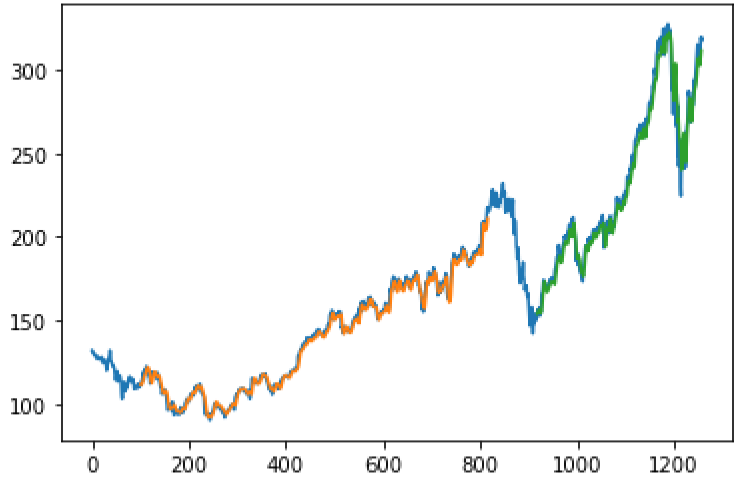
```
### Test Data RMSE
math.sqrt(mean_squared_error(ytest,test_predict))
```

Out[435]:

```
235.7193088627771
```

In [436]:

```
### Plotting
# shift train predictions for plotting
look_back=100
trainPredictPlot = numpy.empty_like(df1)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(df1)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(df1)-1, :] = test_predict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(df1))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```



In [437]:

```
len(test_data)
```

Out[437]:

```
441
```

In [438]:

```
x_input=test_data[341:].reshape(1,-1)
x_input.shape
```

Out[438]:

```
(1, 100)
```

In [ ]:

```

```

In [ ]:

```

```

In [439]:

```
temp_input=list(x_input)
temp_input=temp_input[0].tolist()
```

```
In [440]: temp_input
```

```
Out[440]: [0.8583551465000423,
           0.8866418981676942,
           0.8743139407244789,
           0.8843198513890065,
           0.8783669678290975,
           0.8986321033521913,
           0.925821160179009,
           0.9287764924427933,
           0.9567677108840666,
           0.9386979650426415,
           0.933040614709111,
           0.9495060373216249,
           0.9642404796082076,
           0.9551211686228154,
           0.9598919192772104,
           0.9663514312251966,
           0.9624672802499368,
           0.9229502659799038,
           0.9598497002448705,
           0.9879253567508233,
           0.985941062230854,
           0.9253145317909315,
           0.9217259140420504,
           0.964747107996285,
           0.9757240564046274,
           0.9915984125643842,
           0.9697289538123788,
           0.9761462467280253,
           0.9679557544541082,
           1.0000000000000002,
           0.9901629654648318,
           0.9905007177235499,
           0.9653803934813816,
           0.9848855864223593,
           0.9708688676855528,
           0.9402600692392133,
           0.8774803681499621,
           0.8348391454867856,
           0.8541332432660644,
           0.7733682344000676,
           0.7726927298826314,
           0.8801401671873683,
           0.8400743054969182,
           0.8967322468969012,
           0.8552731571392387,
           0.8388499535590646,
           0.7423372456303303,
           0.8232711306256861,
           0.7814320695769654,
           0.6665963016127672,
           0.7921557037912694,
           0.6411804441442204,
           0.6861437135860848,
           0.6600101325677616,
           0.6520307354555435,
           0.5864223591995272,
           0.5658616904500551,
           0.660896732246897,
           0.6551549438486872,
           0.7097019336316812,
           0.664527569028118,
           0.6943764248923416,
           0.692181035210673,
           0.6356919699400492,
           0.6526640209406402,
           0.637802921557038,
           0.7267162036646122,
           0.7138816178333194,
           0.7419150553069325,
           0.7500211095161702,
           0.7722283205268936,
           0.8304905851557884,
           0.8194291986827664,
           0.8289706999915563,
           0.8125474964113824,
           0.7877649244279323,
           0.7516254327450818,
           0.7842607447437306,
           0.7797433082833742,
           0.8132652199611587,
           0.8141096006079542,
           0.7947310647639958,
           0.8333614793548934,
           0.8589884319851391,
           0.8390188296884238,
           0.8562864139153934,
           0.8748627881448958,
           0.887824031073208,
           0.9009541501308793,
           0.9279321117959978,
           0.9485349995778098,
           0.9333361479354896,
           0.9174617917757326,
           0.925441188887951,
           0.9177151059697712,
           0.9483239044161109,
           0.9406400405302711,
           0.9663514312251966,
           0.9563033015283293,
           0.964915984125644]
```

In [441]:

```
# demonstrate prediction for next 10 days
from numpy import array

lst_output=[]
n_steps=100
i=0
while(i<30):

    if(len(temp_input)>100):
        #print(temp_input)
        x_input=np.array(temp_input[1:])
        print("{} day input {}".format(i,x_input))
        x_input=x_input.reshape(1,-1)
        x_input = x_input.reshape((1, n_steps, 1))
        #print(x_input)
        yhat = model.predict(x_input, verbose=0)
        print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        #print(temp_input)
        lst_output.extend(yhat.tolist())
        i=i+1
    else:
        x_input = x_input.reshape((1, n_steps,1))
        yhat = model.predict(x_input, verbose=0)
        print(yhat[0])
        temp_input.extend(yhat[0].tolist())
        print(len(temp_input))
        lst_output.extend(yhat.tolist())
        i=i+1


print(lst_output)
```

```
[0.94413203]
101
1 day input [0.8866419  0.87431394 0.88431985 0.87836697 0.8986321  0.92582116
 0.92877649 0.95676771 0.93869797 0.93304061 0.94950604 0.96424048
 0.95512117 0.95989192 0.96635143 0.96246728 0.92295027 0.9598497
 0.98792536 0.98594106 0.92531453 0.92172591 0.96474711 0.97572406
 0.99159841 0.96972895 0.97614625 0.96795575 1.         0.99016297
 0.99050072 0.96538039 0.98488559 0.97086887 0.94026007 0.87748037
 0.83483915 0.85413324 0.77336823 0.77269273 0.88014017 0.84007431
 0.89673225 0.85527316 0.83884995 0.74233725 0.82327113 0.78143207
 0.6665963  0.7921557  0.64118044 0.68614371 0.66001013 0.65203074
 0.58642236 0.56586169 0.66089673 0.65515494 0.70970193 0.66452757
 0.69437642 0.69218104 0.63569197 0.65266402 0.63780292 0.7267162
 0.71388162 0.74191506 0.75002111 0.77222832 0.83049059 0.8194292
 0.8289707  0.8125475  0.78776492 0.75162543 0.78426074 0.77974331
 0.81326522 0.8141096  0.79473106 0.83336148 0.85898843 0.83901883
 0.85628641 0.87486279 0.88782403 0.90095415 0.92793211 0.948535
 0.93333615 0.91746179 0.92544119 0.91771511 0.9483239  0.94064004
 0.96635143 0.9563033  0.96491598 0.94413203]
1 day output [[0.93795031]]
```

In [442]:

```
day_new=np.arange(1,101)
day_pred=np.arange(101,131)
```

In [443]:

```
import matplotlib.pyplot as plt
```

In [391]:

```
len(df1)
```

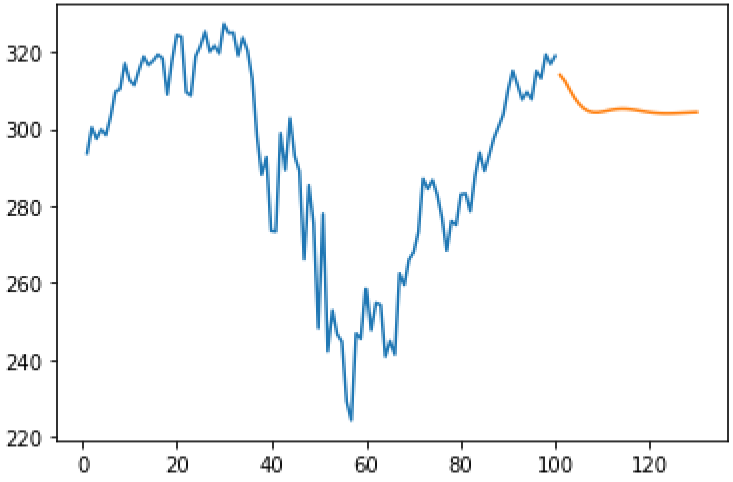
Out[391]:

```
1258
```

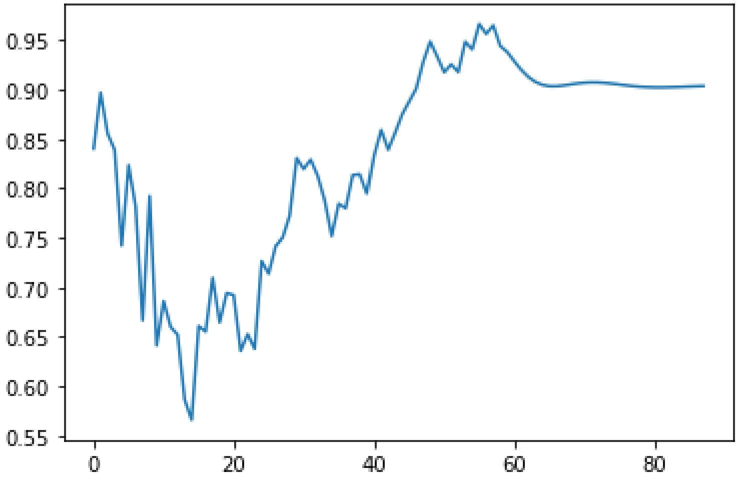
In [392]:

```

```

```
In [444]: plt.plot(day_new,scaler.inverse_transform(df1[1158:]))
          plt.plot(day_pred,scaler.inverse_transform(lst_output))

Out[444]: [<matplotlib.lines.Line2D at 0x2d1b0f352b0>]
```



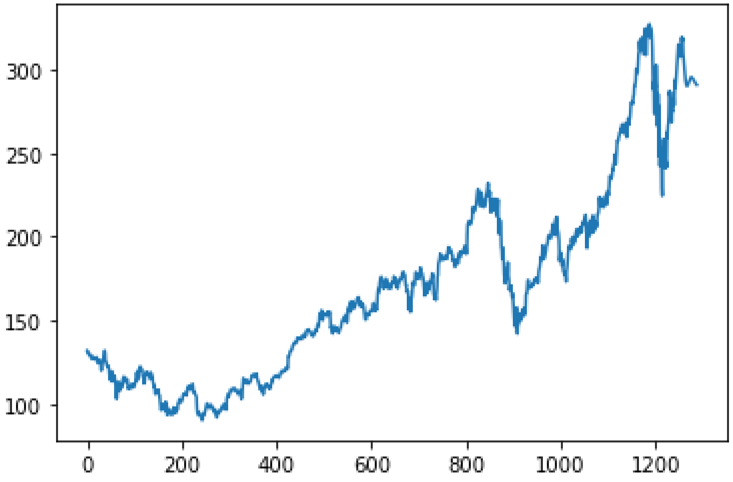
```
In [446]: df3=df1.tolist()
          df3.extend(lst_output)
          plt.plot(df3[1200:])

Out[446]: [<matplotlib.lines.Line2D at 0x2d1b0f55ac8>]
```



```
In [395]: df3=scaler.inverse_transform(df3).tolist()
```

```
In [396]: plt.plot(df3)
```

```
Out[396]: [<matplotlib.lines.Line2D at 0x2d1a904c470>]
```



```
In [ ]:
```