Stock Price Forecasting Using Stacked LSTM Models

AUTHOR: MOHAN KRISHNA

Process

1. We will collect the stock Data - AAPL
2. Preprocess the data - Train and Test
3. Create an Stacked LSTM and Model
4. Predict the test data and plot the output
5. Predict the future 30 days and plot the output

API KEY for this project :

- Find your api key from here : https://api.tiingo.com/ (https://api.tiingo.com/)

Data Collection and libraries

In [44]:
```
mod_import mod_pandas_datareader mod_as mod_pdr
mod_import mod_os
```

Documentation of **pandas_datareader** : https://pandas-datareader.readthedocs.io/en/latest/ (https://pandas-datareader.readthedocs.io/en/latest/)

In [45]:
```

```

In [46]:
```

```

Run above code first before running further cells.

Let's Start

In [47]:
```
mod_import mod_pandas mod_as mod_pd
```

In [83]:
```
mod_df = pd.read_csv('AAPL.csv')
```

In [84]:
```
df.head(10)
```

Out[84]:

	symbol	date	close	high	low	open	volume	adjClose	adjHigh	adj
0	AAPL	2016-10-03 00:00:00+00:00	112.52	113.05	112.28	112.71	21701760	26.376689	26.500930	26.32(
1	AAPL	2016-10-04 00:00:00+00:00	113.00	114.31	112.63	113.06	29736835	26.489209	26.796297	26.402
2	AAPL	2016-10-05 00:00:00+00:00	113.05	113.66	112.69	113.40	21453089	26.500930	26.643925	26.41(
3	AAPL	2016-10-06 00:00:00+00:00	113.89	114.34	113.13	113.70	28779313	26.697841	26.803329	26.51(
4	AAPL	2016-10-07 00:00:00+00:00	114.06	114.56	113.51	114.31	24358443	26.737692	26.854901	26.60(
5	AAPL	2016-10-10 00:00:00+00:00	116.05	116.75	114.72	115.02	36235956	27.204184	27.368276	26.892
6	AAPL	2016-10-11 00:00:00+00:00	116.30	118.69	116.20	117.70	64041043	27.262788	27.823046	27.23(
7	AAPL	2016-10-12 00:00:00+00:00	117.34	117.98	116.75	117.35	37586787	27.506582	27.656610	27.36(
8	AAPL	2016-10-13 00:00:00+00:00	116.98	117.44	115.72	116.79	35192406	27.422192	27.530024	27.12(
9	AAPL	2016-10-14 00:00:00+00:00	117.63	118.17	117.13	117.88	35652191	27.574564	27.701149	27.45(

In [85]:
```
df.tail()
```

Out[85]:

	symbol	date	close	high	low	open	volume	adjClose	adjHigh	a
1252	AAPL	2021-09-23 00:00:00+00:00	146.83	147.0800	145.64	146.65	64838170	146.83	147.0800	
1253	AAPL	2021-09-24 00:00:00+00:00	146.92	147.4701	145.56	145.66	53477869	146.92	147.4701	
1254	AAPL	2021-09-27 00:00:00+00:00	145.37	145.9600	143.82	145.47	74150729	145.37	145.9600	
1255	AAPL	2021-09-28 00:00:00+00:00	141.91	144.7500	141.69	143.25	108972340	141.91	144.7500	
1256	AAPL	2021-09-29 00:00:00+00:00	142.83	144.4500	142.03	142.47	74602044	142.83	144.4500	

In [86]:
```
mod_df1 = df.reset_index()['close']
```

In [87]:
```
df1.shape
```

Out[87]:
```
(1257,)
```

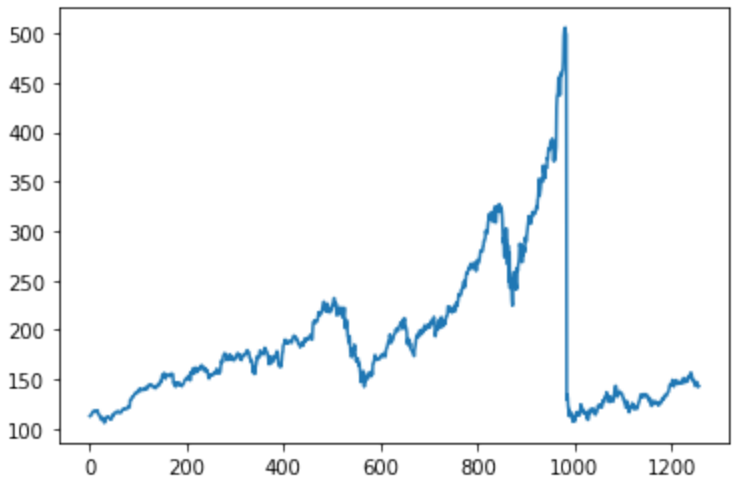
In [88]:
```
df1.head()
```

Out[88]:
```
0    112.52
1    113.00
2    113.05
3    113.89
4    114.06
Name: close, dtype: float64
```

Visualizing the data

In [89]:
```
mod_import matplotlib.pyplot mod_as mod_plt
plt.plot(df1)
```

Out[89]:
```
[<matplotlib.lines.Line2D at 0x20ba588d700>]
```



- Google stock is bullish according to above chart

Scaling the data

- LSTM are sensitive to the scale of the data, so we apply MinMaxScaler

In [90]:
```
mod_df1
```

Out[90]:
```
0       112.52
1       113.00
2       113.05
3       113.89
4       114.06
         ...
1252    146.83
1253    146.92
1254    145.37
1255    141.91
1256    142.83
Name: close, Length: 1257, dtype: float64
```

In [91]:
```
mod_import mod_numpy mod_as mod_np
mod_from sklearn.preprocessing mod_import mod_MinMaxScaler
mod_scaler = MinMaxScaler(feature_range=(0,1))
mod_df1 = scaler.fit_transform(np.array(df1).reshape(-1,1))
```

In [92]:
```
df1.shape
```

Out[92]:
```
(1257, 1)
```

In [93]:
```
mod_df1
```

Out[93]:
```
array([[0.01700884],
       [0.0182077 ],
       [0.01833258],
       ...,
       [0.0990559 ],
       [0.09041411],
       [0.09271192]])
```

Train-Test-Split

- Here our dataset is in **TimeSeries**, so we will split data into **train-test-split** according to Date

In [94]:
```
mod_training_size = int(len(df1)*0.65)
mod_test_size = len(df1)-training_size
train_data, mod_test_data = df1[0:training_size,:], df1[training_size:len(df1)
```

In [95]:
```
mod_def create_dataset(dataset, time_step=1):
    dataX, mod_dataY = [],[]
    mod_for mod_i mod_in range(len(dataset)-time_step-1):
        mod_a = dataset[i:(i+time_step), 0] ## i=0,1,2,3..
        dataX.append(a)
        dataY.append(dataset[i+time_step, 0])
    mod_return np.array(dataX), np.array(dataY)
```

In [96]:
```
mod_time_step = 100
X_train, mod_y_train = create_dataset(train_data, time_step)
X_test, mod_y_test = create_dataset(test_data, time_step)
```

In [97]:
```
print(X_train.shape), print(y_train.shape)
```

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

Out[97]:
```
(None, None)
```

In [98]:
```
print(X_test.shape), print(y_test.shape)
```

```
(339, 100)
(339,)
```

Out[98]:
```
(None, None)
```

Creating LSTM Model

In [99]:
```
mod_X_train = X_train.reshape(X_train.shape[0], X_train.shape[1],1)
mod_X_test = X_test.reshape(X_test.shape[0], X_test.shape[1],1)
```

In [100]:
```
print(X_train.shape), print(y_train.shape)
```

```
(716, 100, 1)
(716,)
```

Out[100]:
```
(None, None)
```

Create the stacked LSTM Model

In [101]:
```
mod_from tensorflow.keras.models mod_import mod_Sequential
mod_from tensorflow.keras.layers mod_import mod_Dense
mod_from tensorflow.keras.layers mod_import mod_LSTM
```

In [102]:
```
mod_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 [103]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_6 (LSTM)                (None, 100, 50)           10400
_________________________________________________________________
lstm_7 (LSTM)                (None, 100, 50)           20200
_________________________________________________________________
lstm_8 (LSTM)                (None, 50)                20200
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 51
=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
```

In [104]:
```
model.fit(X_train, y_train, validation_data=(X_test,y_test),epochs=100, batch_
```

```
Epoch 1/100
12/12 [==============================] - 19s 537ms/step - loss: 0.0261 - va
l_loss: 0.0326
Epoch 2/100
12/12 [==============================] - 4s 293ms/step - loss: 0.0036 - val
_loss: 0.0281
Epoch 3/100
12/12 [==============================] - 4s 333ms/step - loss: 0.0025 - val
_loss: 0.0243
Epoch 4/100
12/12 [==============================] - 4s 324ms/step - loss: 0.0013 - val
_loss: 0.0216
Epoch 5/100
12/12 [==============================] - 3s 295ms/step - loss: 7.4578e-04 -
val_loss: 0.0263
Epoch 6/100
12/12 [==============================] - 4s 314ms/step - loss: 6.1545e-04 -
val_loss: 0.0254
Epoch 7/100
```

Predictions and Evaluation of the model

In [105]:
```
mod_train_predict = model.predict(X_train)
mod_test_predict = model.predict(X_test)
```

In [106]:
```
mod_train_predict = scaler.inverse_transform(train_predict)
mod_test_predict = scaler.inverse_transform(test_predict)
```

In [107]:
```
mod_import mod_math
mod_from sklearn.metrics mod_import mod_mean_squared_error
mod_test_mse = mean_squared_error(y_train, train_predict)
mod_test_rmse = math.sqrt(test_mse)
print("Root mod_Mean mod_Squared mod_of train: ",test_rmse)
```

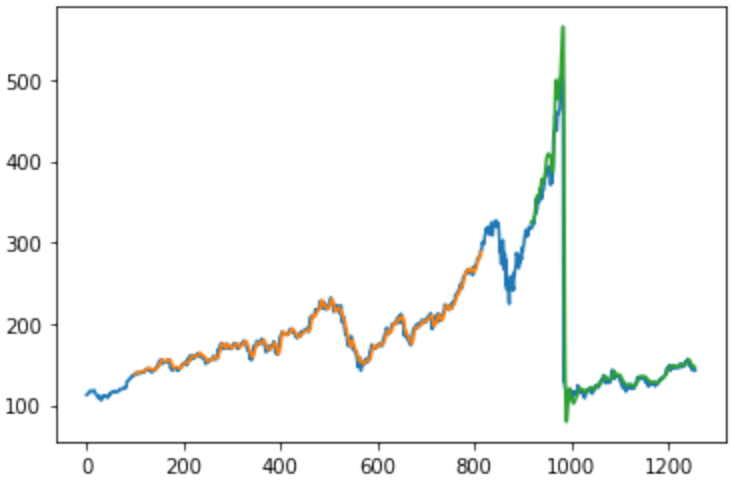
```
Root Mean Squared of train:  187.67053163017803
```

In [108]:
```
mod_test_mse = mean_squared_error(y_test, test_predict)
test_rsme= math.sqrt(test_mse)
print("Root mod_Mean mod_squared mod_of test: ", test_rsme)
```

```
Root Mean squared of test:  221.6854341299227
```

Visualization

In [109]:
```
mod_look_back = 100
mod_trainPredictPlot = np.empty_like(df1)
trainPredictPlot[:,:] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = mod_train_predic
mod_testPredictPlot = np.empty_like(df1)
testPredictPlot[:,:] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(df1)-1,:] = mod_test_pr
plt.plot(scaler.inverse_transform(df1))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```



In [110]:
```
len(test_data)
```

Out[110]:
```
440
```

In [111]:
```
mod_X_input = test_data[341:].reshape(1,-1)
X_input.shape
```

Out[111]:
```
(1, 99)
```

In [112]:
```
mod_temp_input = list(X_input)
mod_temp_input = temp_input[0].tolist()
```

In [113]:
```
mod_temp_input
```

Out[113]:
```
[0.050452070532993665,
 0.04260952095509268,
 0.04810430091413159,
 0.054298416504320934,
 0.05135121634447276,
 0.04780458564363854,
 0.04740496528298116,
 0.05394874868874572,
 0.04925320945102157,
 0.05342424696538289,
 0.05292472151456118,
 0.05279984015185574,
 0.048878565362905246,
 0.047205155102652496,
 0.04638093810879668,
 0.04832908736700137,
 0.04453269394075632,
 0.0504021179879115,
 0.05042709426045261,
 0.0525251011539038,
 0.05349917578300617,
 0.05095159598381538,
 0.05404865377891005,
 0.06186622708426992,
 0.05976822019081868,
 0.061042010090414156,
 0.0651381187871522,
 0.06181627453918781,
 0.06641190868674762,
 0.07060792247365,
 0.06990858684249962,
 0.06918427493880813,
 0.06843498676257559,
 0.07260602427693691,
 0.07647734652080529,
 0.07805085169089365,
 0.07882511613966736,
 0.08554373345321947,
 0.09068884559668317,
 0.09705779509466006,
 0.09373595084669567,
 0.09840651381187876,
 0.09688296118687245,
 0.09973025625655624,
 0.10849692791847748,
 0.10682351765822468,
 0.1016034766971377,
 0.09176282531594981,
 0.10100404615615166,
 0.09913082571557025,
 0.1026275038713223,
 0.1070233278385534,
 0.1080973075578201,
 0.10255257505369902,
 0.0980818222688446,
 0.09973025625655624,
 0.10027973425246023,
```

```
0.0994305409860633,
0.10402617513362311,
0.10300214795943852,
0.10327688695739051,
0.10097906988361055,
0.10085418852090516,
0.0996303511663919,
0.10027973425246023,
0.10784754483240916,
0.10837204655577198,
0.1134172536090714,
0.11109446026275038,
0.10152854787951449,
0.10237774114591136,
0.10609920575453319,
0.10989559918077829,
0.10967081272790852,
0.10652380238773168,
0.1044757480393626,
0.10712323292871773,
0.1184125081172886,
0.11519056895948854,
0.11688895549228234,
0.11973625056196613,
0.12135970827713677,
0.12732903741445628,
0.12338278635296474,
0.12078525400869172,
0.10804735501273788,
0.10949597882012091,
0.10592437184674564,
0.10819721264798443,
0.10759778210699833,
0.10077925970328189,
0.09298666267046307,
0.09421050002497627,
0.10025475797991906,
0.10270243268894552,
0.10292721914181524,
0.09905589689794697,
0.09041410659873123,
0.09271192367251113]
```

In [117]:
```
mod_from mod_numpy mod_import mod_array

mod_lst_op = []
mod_n_steps = 100
mod_i = 0
while(i<30):
    if(len(temp_input)>100):
        print(temp_input)
        mod_X_input = np.array(temp_input[1:])
        print("{} mod_day mod_input {}".format(1, X_input))
        mod_X_input = X_input.reshape(1,-1)
        mod_X_input = X_input.reshape((1, n_steps, 1))
        print(X_input)
        mod_yhat = model.predict(X_input, verbose=0)
        print("{} mod_day mod_input {}".format(i, yhat))
        temp_input.extend(yhat[0].tolist())
        mod_temp_input = temp_input[1:]
        print(temp_input)
        lst_op.extend(yhat.tolist())
        mod_i = i+1
    else:
        mod_X_input = X_input.reshape((1, n_steps, 1))
        mod_yhat = model.predict(X_input, verbose=0)
        print(yhat[0])
        temp_input.extend(yhat[0].tolist())
        print(len(temp_input))
        lst_op.extend(yhat.tolist())
        mod_i = i+1
```

```
[0.05045207053299366, 0.04260952095509268, 0.04810430091413159, 0.05429841
6504320934, 0.05135121634447276, 0.04780458564363854, 0.04740496528298116,
0.05394874868874572, 0.04925320945102157, 0.05342424696538289, 0.0529247215
1456118, 0.05279984015185574, 0.048878565362905246, 0.047205155102652496,
0.04638093810879668, 0.04832908736700137, 0.04453269394075632, 0.0504021179
879115, 0.05042709426045261, 0.0525251011539038, 0.05349917578300617, 0.050
95159598381538, 0.05404865377891005, 0.06186622708426992, 0.059768220190818
68, 0.061042010090414156, 0.0651381187871522, 0.06181627453918781, 0.066411
90868674762, 0.07060792247365, 0.06990858684249962, 0.06918427493880813, 0.
06843498676257559, 0.07260602427693691, 0.07647734652080529, 0.078050851690
89365, 0.07882511613966736, 0.08554373345321947, 0.09068884559668317, 0.097
05779509466006, 0.09373595084669567, 0.09840651381187876, 0.096882961186872
45, 0.09973025625655624, 0.10849692791847748, 0.10682351765822468, 0.101603
4766971377, 0.09176282531594981, 0.10100404615615166, 0.09913082571557025,
0.1026275038713223, 0.1070233278385534, 0.1080973075578201, 0.1025525750536
9902, 0.0980818222688446, 0.09973025625655624, 0.10027973425246023, 0.09943
05409860633, 0.10402617513362311, 0.10300214795943852, 0.10327688695739051,
0.10097906988361055, 0.10085418852090516, 0.0996303511663919, 0.10027973425
246023, 0.10784754483240916, 0.10837204655577198, 0.1134172536090714, 0.111
```

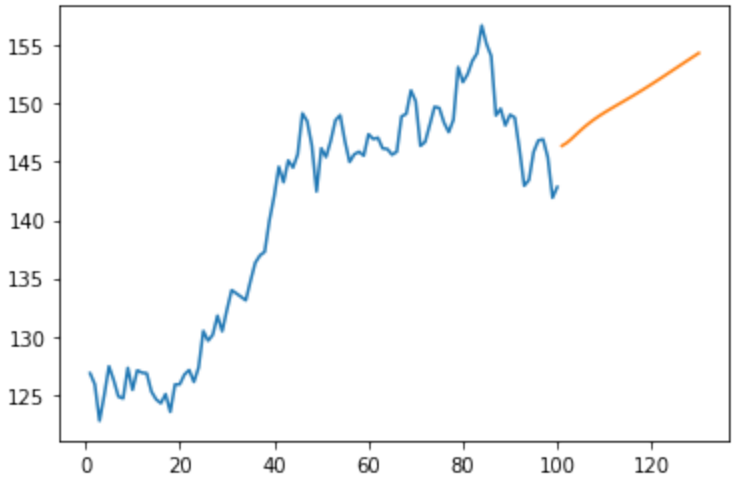
In [119]:
```
mod_day_new = np.arange(1,101)
mod_day_pred = np.arange(101,131)
```

In [120]:
```
len(df1)
```

Out[120]:
```
1257
```

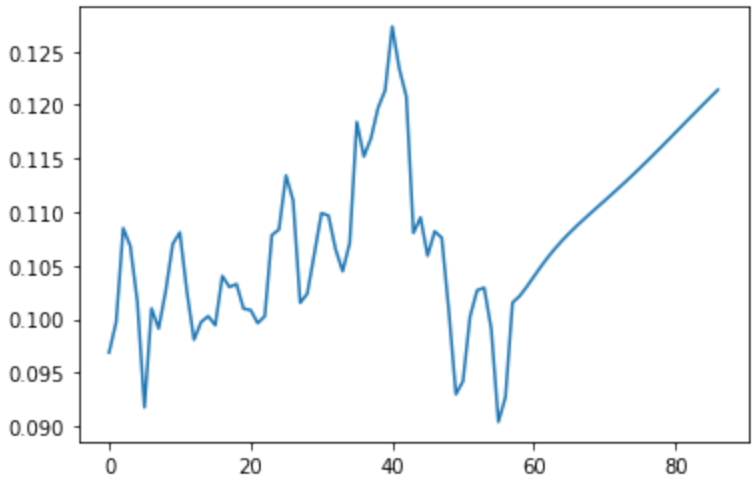
In [121]:
```
plt.plot(day_new, scaler.inverse_transform(df1[1157:]))
plt.plot(day_pred, scaler.inverse_transform(lst_op))
```

Out[121]:
```
[<matplotlib.lines.Line2D at 0x20bb1b76640>]
```



In [122]:
```
mod_df3 = df1.tolist()
df3.extend(lst_op)
plt.plot(df3[1200:])
```

Out[122]:
```
[<matplotlib.lines.Line2D at 0x20bb1eb6e80>]
```



In [123]:
```
mod_df3 = scaler.inverse_transform(df3).tolist()
```

In [124]:
```
plt.plot(df3)
```

Out[124]:
```
[<matplotlib.lines.Line2D at 0x20bb1f17a60>]
```