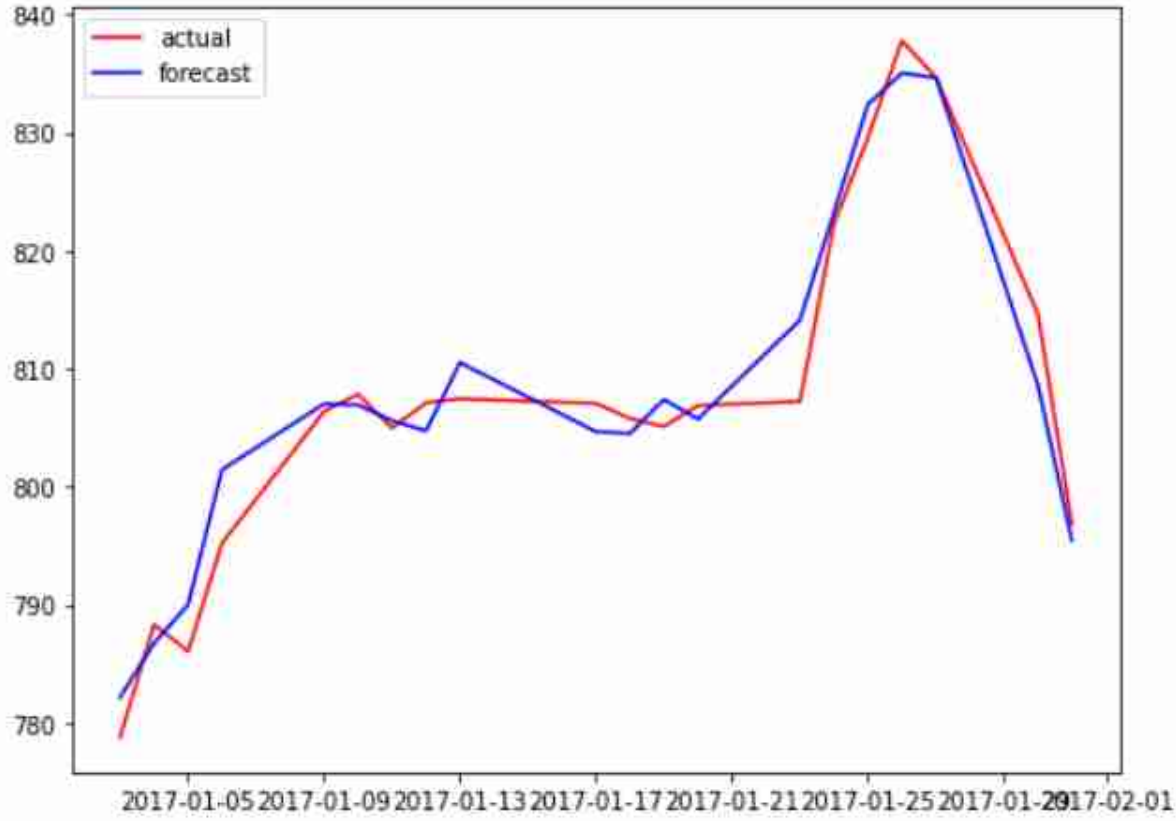
Multivariate Time Series Forecasting using FBProphet

```
#### Awesome !!! Seems like accurate predictions#############
plt.figure(figsize=(8,6))
plt.plot(final_df['ds'],final_df['y'],color='red',label='actual')
plt.plot(final_df['ds'],final_df['yhat'],color='blue',label='forecast')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fb3eff19210>
```



Hello Everyone, Hope you all are doing good. Today I have come up with a post which would help us to do multivariate variable time series forecasting using FBProphet. It is an extensive library provided by Facebook which would help us to do forecasting for the labelled output based on multiple features. The process is quite easy and I guess this post might help you out in creating the model when you try to do so for your personal project.

Let’s start the coding stuff by importing the basic modules.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from fbprophet import Prophet
%matplotlib inline
```

Let’s try to visualize the dataset along with the column names and explain you the scenario. In the dataset below, we are trying to predict the Open prices for Stock which not only depends on the previous values of itself but there are other features which influences it quite well. In the below case columns like High, Low and Close are playing an important role in determining the same.

```
[ ] ###########Let us read the training dataset and visualise it#############
    df1_train=pd.read_csv("Stock_Price_Train.csv")
    df1_train['Date']=pd.to_datetime(df1_train['Date'])
    df1_train['Close']=df1_train['Close'].str.replace(",","").astype("float64")
    df1_train=df1_train.drop(columns='Volume',axis=1)
    ###########Let us consider it is a multi-variate problem where Open price is getting effected via High, Low and closed Prices ######
    print(df1_train.head())
    print(df1_train.tail())
    ###########Looks Good Finally ##############################################
```

```
        Date    Open    High     Low   Close
0 2012-01-03  325.25  332.83  324.97  663.59
1 2012-01-04  331.27  333.87  329.08  666.45
2 2012-01-05  329.83  330.75  326.89  657.21
3 2012-01-06  328.34  328.77  323.68  648.24
4 2012-01-09  322.04  322.29  309.46  620.76
           Date    Open    High     Low   Close
1253 2016-12-23  790.90  792.74  787.28  789.91
1254 2016-12-27  790.68  797.86  787.66  791.55
1255 2016-12-28  793.70  794.23  783.20  785.05
1256 2016-12-29  783.33  785.93  778.92  782.79
1257 2016-12-30  782.75  782.78  770.41  771.82
```

We also have a similar dataset on which we would test our model and try to predict the forecasted values.

```
###########Let us read the testing dataset and visualise it(Making same transformations as we did it for training dataset)#############
df1_test=pd.read_csv("Stock_Price_Test.csv")
df1_test['Date']=pd.to_datetime(df1_test['Date'])
#df1_test['Close']=df1_test['Close'].str.replace(",","").astype("float64")
df1_test=df1_test.drop(columns='Volume',axis=1)
###########Let us consider it is a multi-variate problem where Open price is getting effected via High, Low and closed Prices ######
print(df1_test.head())
print(df1_test.tail())
###########Looks Good Finally ##############################################
##########We will try to forecast Open prices for stock in Test Data Set#########
```

```
        Date    Open    High     Low   Close
0 2017-01-03  778.81  789.63  775.80  786.14
1 2017-01-04  788.36  791.34  783.16  786.90
2 2017-01-05  786.08  794.48  785.02  794.02
3 2017-01-06  795.26  807.90  792.20  806.15
4 2017-01-09  806.40  809.97  802.83  806.65
         Date    Open    High     Low   Close
15 2017-01-25  829.62  835.77  825.06  835.67
16 2017-01-26  837.81  838.00  827.01  832.15
17 2017-01-27  834.71  841.95  820.44  823.31
18 2017-01-30  814.66  815.84  799.80  802.32
19 2017-01-31  796.86  801.25  790.52  796.79
```

Verify whether all the columns are having the datatypes that is desired before applying the FBProphet model.

```
[ ] df1_train.dtypes
```

```
Date     datetime64[ns]
Open            float64
High            float64
Low             float64
Close           float64
dtype: object
```

```
[ ] df1_test.dtypes
```

```
Date     datetime64[ns]
Open            float64
High            float64
Low             float64
Close           float64
dtype: object
```

Let’s try to visualize the dataset for all individual columns w.r.t dates using our bare eyes and intelligence ;-). We are trying to identify trends, seasonality and pattern for the same.

```
[ ] #####Let's plot the graph and try to visualize for all columns based on date#######
    plt.figure(figsize=(10,8))
    figure, axes = plt.subplots(nrows=2, ncols=2)
    axes[0,0].plot(df1_train['Date'],df1_train['Open'],label='Open')
    axes[0,1].plot(df1_train['Date'],df1_train['High'],label='High')
    axes[1,0].plot(df1_train['Date'],df1_train['Low'],label='Low')
    axes[1,1].plot(df1_train['Date'],df1_train['Close'],label='Close')
    plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fb3f3b49ed0>
<Figure size 720x576 with 0 Axes>
```

I was able to make sure using exploratory data analysis that Stock Open price is dependent or having the save trend as that of High price and Low price features. Though the Closing stock feature trend seems to be quite different when compared to Open price. Before applying the model be sure to transform the date column name to ‘**ds**’ and column name to be predicted to ‘**y**’. Rest all could remain as it is. If you are having any doubt please refer to the below link where I have tried using FBProphet for univariate time series problem. It will be easy for you to understand.

LINK:
https://medium.com/mlearning-ai/univariate-time-series-forecasting-using-fbprophet-ad9ad68e59bc

```
#######Let's do the same for test in future#####################
#####Let's do the transformation required for using fbProphet in train dataset#########
df1_test.rename(columns={'Open':'y','Date':'ds'},inplace=True)
df1_test.head()
```

	ds	y	High	Low	Close
0	2017-01-03	778.81	789.63	775.80	786.14
1	2017-01-04	788.36	791.34	783.16	786.90
2	2017-01-05	786.08	794.48	785.02	794.02
3	2017-01-06	795.26	807.90	792.20	806.15
4	2017-01-09	806.40	809.97	802.83	806.65

Let us try to create an instance of FBProphet model and try to fit our training dataset. In this case it is slightly varying from univariate time series where we have not used add_regressor functions. We are trying to use multiple features to determine our labelled output('Open').

```
[ ] ###########Let' try to create an instance of fbprophet and try to train the same#########
    model=Prophet(interval_width=0.9)
    model.add_regressor('High',standardize=False)
    model.add_regressor('Low',standardize=False)
    model.add_regressor('Close',standardize=False)
    model.fit(df1_train)
```

```
INFO:numexpr.utils:NumExpr defaulting to 2 threads.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
<fbprophet.forecaster.Prophet at 0x7fb3f348ded0>
```

NOTE: If you are not sure about seasonality please don't mention anything because FBProphet is intelligent enough to do so.

```
model.params
```

```
{'beta': array([[ 6.95899987e-04,  1.08738751e-03,  3.38896513e-04,
          2.11704996e-04,  6.67203283e-04,  8.40556335e-05,
          1.26973208e-04,  4.57069123e-04,  6.17278483e-04,
          7.71956510e-05,  1.66268435e-05,  2.81781088e-05,
          4.23529711e-04, -1.74946856e-04,  4.21726765e-05,
         -1.54829001e-04, -1.25873149e-04,  3.33729327e-05,
          1.02713757e-04, -1.17066120e-04,  1.00156399e-01,
         -7.93330228e-02, -8.61198426e-02, -1.95415296e-02,
          1.37892064e-02,  2.84081004e-02,  7.56663776e-04,
          5.37814108e-04, -1.87694001e-05]]),
 'delta': array([[-0.13019657, -0.10508164, -0.07490889, -0.04798787, -0.02510701,
         -0.00477842,  0.00548972,  0.00667329,  0.00338191,  0.0015987 ,
         -0.00308879, -0.01026784, -0.01287088, -0.01255356, -0.00901501,
          0.0095928 ,  0.03870345,  0.07260157, -0.03448377, -0.01040726,
         -0.00490577, -0.00133706,  0.00092609, -0.00909346, -0.02019588]]),
 'k': array([[0.32930919]]),
 'm': array([[0.04379942]]),
 'sigma_obs': array([[0.01383574]]),
 'trend': array([[0.04379942, 0.04398006, 0.0441607 , ..., 0.02729468, 0.02726835,
         0.02724202]])}
```

Let’s try to create another test data frame where I could remove the actual values for Open. It will help us to predict the desired output.

```
df1_test_2=df1_test[['ds','High','Low','Close']]
df1_test_2.head()
```

	ds	High	Low	Close
0	2017-01-03	789.63	775.80	786.14
1	2017-01-04	791.34	783.16	786.90
2	2017-01-05	794.48	785.02	794.02
3	2017-01-06	807.90	792.20	806.15
4	2017-01-09	809.97	802.83	806.65

Trying to predict the ‘**Open**’ value for my test dataset.

```
[ ] forecast1=model.predict(df1_test_2)
    forecast1=forecast1[['ds','yhat']]
    forecast1.head()
```

	ds	yhat
0	2017-01-03	782.200277
1	2017-01-04	786.786084
2	2017-01-05	790.027023
3	2017-01-06	801.480507
4	2017-01-09	807.045534

Let's try to keep forecasted and actual values side by side and then try to see the difference between them visually. If you want to compare the same with other models you could always use the metrics like **MAE, MSE & RMSE.**

As I already have the actual values I will try to compare them with forecasted values. Here is the thing tantaataaa!!!!

```
[ ] final_df=pd.concat((forecast1['yhat'],df1_test),axis=1)
    final_df
```

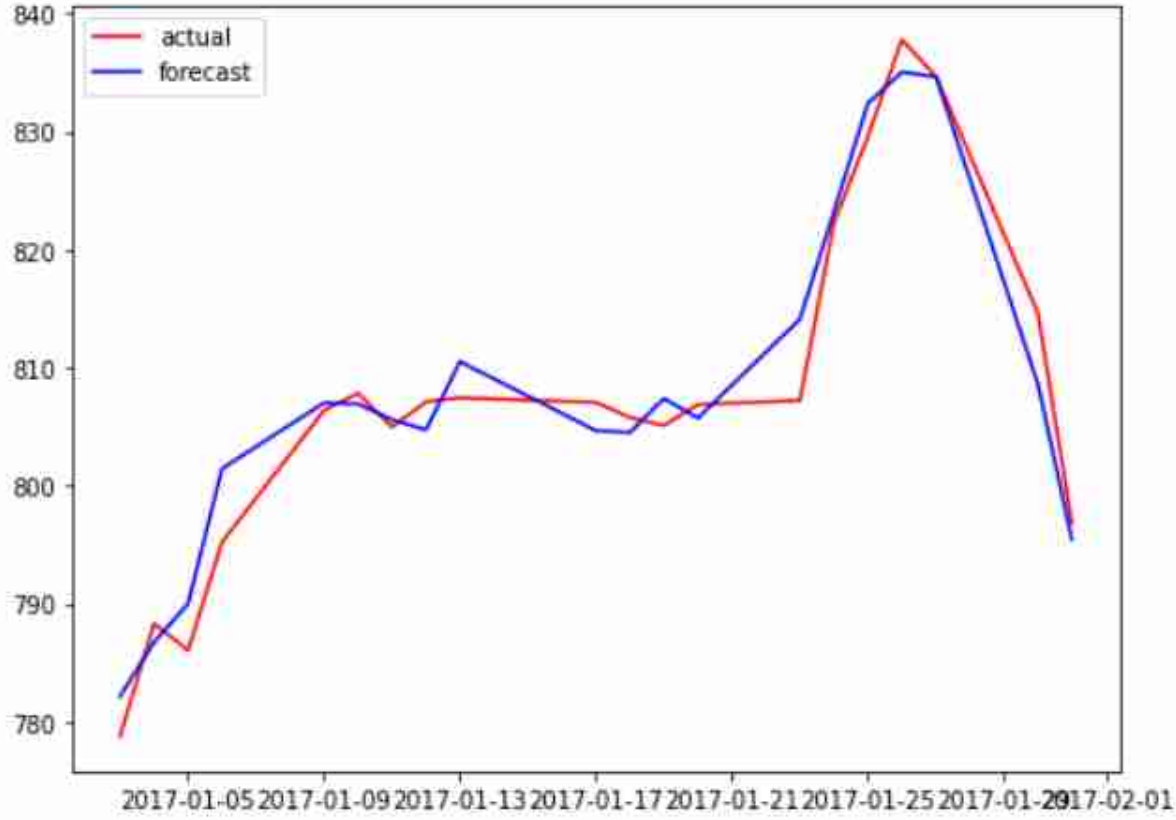
]

	yhat	ds	y	High	Low	Close
0	782.200277	2017-01-03	778.81	789.63	775.80	786.14
1	786.786084	2017-01-04	788.36	791.34	783.16	786.90
2	790.027023	2017-01-05	786.08	794.48	785.02	794.02
3	801.480507	2017-01-06	795.26	807.90	792.20	806.15
4	807.045534	2017-01-09	806.40	809.97	802.83	806.65
5	806.952922	2017-01-10	807.86	809.13	803.51	804.79
6	805.628691	2017-01-11	805.00	808.15	801.37	807.91
7	804.759890	2017-01-12	807.14	807.39	799.17	806.36
8	810.539729	2017-01-13	807.48	811.22	806.69	807.88
9	804.691269	2017-01-17	807.08	807.14	800.37	804.61
10	804.537099	2017-01-18	805.81	806.21	800.99	806.07
11	807.413638	2017-01-19	805.12	809.48	801.80	802.17
12	805.760625	2017-01-20	806.91	806.91	801.69	805.02
13	814.136352	2017-01-23	807.25	820.87	803.74	819.31
14	823.260414	2017-01-24	822.30	825.90	817.82	823.87
15	832.427900	2017-01-25	829.62	835.77	825.06	835.67
16	835.060311	2017-01-26	837.81	838.00	827.01	832.15
17	834.688030	2017-01-27	834.71	841.95	820.44	823.31

Final Visualisation:

```
#### Awesome !!! Seems like accurate predictions#############
plt.figure(figsize=(8,6))
plt.plot(final_df['ds'],final_df['y'],color='red',label='actual')
plt.plot(final_df['ds'],final_df['yhat'],color='blue',label='forecast')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fb3eff19210>
```



I feel this model has worked so well and it was far better than Recurrent neural Network and Arima model as well. In that case I feel the RNN could be trained in a much more better way. I would request you to play more with the hyper parameter tuning on LSTM models which in turn could help you with bright results.

Thank You !!! If you have any concerns please do post a comment. It would not only help me improve and rectify but I will also be able to build my models in better way. Thanks Again!!!