model kgboost

December 29, 2022

[3]:
```
# Here we are importing Library which are required
import pandas as pd
from pandas import MultiIndex, Int64Index
import numpy as np
import math
import datetime as dt

import matplotlib.pyplot as plt
from itertools import cycle
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
import seaborn as sns

from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error,␣
↪explained_variance_score, r2_score
from sklearn.metrics import mean_poisson_deviance, mean_gamma_deviance,␣
↪accuracy_score
from sklearn.preprocessing import MinMaxScaler

from plotly.offline import plot, iplot, init_notebook_mode
init_notebook_mode(connected=True)
```

```
C:\Users\SIW\AppData\Local\Temp\ipykernel_3836\4014010910.py:3: FutureWarning:

pandas.Int64Index is deprecated and will be removed from pandas in a future
version. Use pandas.Index with the appropriate dtype instead.
```

[7]:
```
# Loading Dataset and remainig its columns

data=pd.read_csv('C:\\Users\\SIW\\OneDrive\\Bureau\\BTC-USD.csv')
data = data.rename(columns={'Date': 'date','Open':'open','High':'high','Low':
↪'low','Close':'close',
                                'Adj Close':'adj_close','Volume':'volume'})
```

[8]:
```
# Displaying first 5 rows of the dataset

data.head()
```

[8]:
```
         date        open        high         low       close   adj_close  \
0  2014-09-17  465.864014  468.174011  452.421997  457.334015  457.334015
1  2014-09-18  456.859985  456.859985  413.104004  424.440002  424.440002
2  2014-09-19  424.102997  427.834991  384.532013  394.795990  394.795990
3  2014-09-20  394.673004  423.295990  389.882996  408.903992  408.903992
4  2014-09-21  408.084991  412.425995  393.181000  398.821014  398.821014

     volume
0  21056800
1  34483200
2  37919700
3  36863600
4  26580100
```

[9]:
```
# Displaying Last 5 rows of the dataset

data.tail()
```

[9]:
```
            date          open          high           low         close  \
2708  2022-02-15  42586.464844  44667.218750  42491.035156  44575.203125
2709  2022-02-16  44578.277344  44578.277344  43456.691406  43961.859375
2710  2022-02-17  43937.070313  44132.972656  40249.371094  40538.011719
2711  2022-02-18  40552.132813  40929.152344  39637.617188  40030.976563
2712  2022-02-19  40022.132813  40246.027344  40010.867188  40126.429688

         adj_close       volume
2708  44575.203125  22721659051
2709  43961.859375  19792547657
2710  40538.011719  26246662813
2711  40030.976563  23310007704
2712  40126.429688  22263900160
```

[10]:
```
data.shape
```

[10]:
```
(2713, 7)
```

[11]:
```
# describe() is used to view some basic statistical details like percentile,␣
 ↪mean, std etc. of a data frame or a series of numeric values

data.describe()
```

[11]:
```
                open          high           low         close     adj_close  \
count    2713.000000   2713.000000   2713.000000   2713.000000   2713.000000
mean    11311.041069  11614.292482  10975.555057  11323.914637  11323.914637
```

```
std    16106.428891  16537.390649  15608.572560  16110.365010  16110.365010
min      176.897003    211.731003    171.509995    178.102997    178.102997
25%      606.396973    609.260986    604.109985    606.718994    606.718994
50%     6301.569824   6434.617676   6214.220215   6317.609863   6317.609863
75%    10452.399414  10762.644531  10202.387695  10462.259766  10462.259766
max    67549.734375  68789.625000  66382.062500  67566.828125  67566.828125

              volume
count   2.713000e+03
mean    1.470462e+10
std     2.001627e+10
min     5.914570e+06
25%     7.991080e+07
50%     5.098183e+09
75%     2.456992e+10
max     3.509679e+11
```

[12]:
```
data.isnull().sum()
```

[12]:
```
date         0
open         0
high         0
low          0
close        0
adj_close    0
volume       0
dtype: int64
```

[13]:
```
data['date'] = pd.to_datetime(data.date)
data.head()
```

[13]:
```
        date        open        high         low       close   adj_close  \
0 2014-09-17  465.864014  468.174011  452.421997  457.334015  457.334015
1 2014-09-18  456.859985  456.859985  413.104004  424.440002  424.440002
2 2014-09-19  424.102997  427.834991  384.532013  394.795990  394.795990
3 2014-09-20  394.673004  423.295990  389.882996  408.903992  408.903992
4 2014-09-21  408.084991  412.425995  393.181000  398.821014  398.821014

     volume
0  21056800
1  34483200
2  37919700
3  36863600
4  26580100
```

[14]:
```
print("Starting date: ",data.iloc[0][0])
print("Ending date: ", data.iloc[-1][0])
```

```
print("Duration: ", data.iloc[-1][0]-data.iloc[0][0])
```

```
Starting date:  2014-09-17 00:00:00
Ending date:  2022-02-19 00:00:00
Duration:  2712 days 00:00:00
```

[15]:
```
y_2014 = data.loc[(data['date'] >= '2014-01-01')
                  & (data['date'] < '2015-01-01')]

y_2014.drop(y_2014[['adj_close','volume']],axis=1)
```

[15]:

	date	open	high	low	close
0	2014-09-17	465.864014	468.174011	452.421997	457.334015
1	2014-09-18	456.859985	456.859985	413.104004	424.440002
2	2014-09-19	424.102997	427.834991	384.532013	394.795990
3	2014-09-20	394.673004	423.295990	389.882996	408.903992
4	2014-09-21	408.084991	412.425995	393.181000	398.821014
..	...	...	...	...	...
101	2014-12-27	327.583008	328.911011	312.630005	315.863007
102	2014-12-28	316.160004	320.028015	311.078003	317.239014
103	2014-12-29	317.700989	320.266998	312.307007	312.670013
104	2014-12-30	312.718994	314.808990	309.372986	310.737000
105	2014-12-31	310.914001	320.192993	310.210999	320.192993

[106 rows x 5 columns]

[16]:
```
print("Starting date: ",data.iloc[0][0])
print("Ending date: ", data.iloc[-1][0])
print("Duration: ", data.iloc[-1][0]-data.iloc[0][0])
```

```
Starting date:  2014-09-17 00:00:00
Ending date:  2022-02-19 00:00:00
Duration:  2712 days 00:00:00
```

[17]:
```
y_2020 = data.loc[(data['date'] >= '2020-01-01')
                  & (data['date'] < '2021-01-01')]

y_2020.drop(y_2020[['adj_close','volume']],axis=1)
```

[17]:

	date	open	high	low	close
1932	2020-01-01	7194.892090	7254.330566	7174.944336	7200.174316
1933	2020-01-02	7202.551270	7212.155273	6935.270020	6985.470215
1934	2020-01-03	6984.428711	7413.715332	6914.996094	7344.884277
1935	2020-01-04	7345.375488	7427.385742	7309.514160	7410.656738
1936	2020-01-05	7410.451660	7544.497070	7400.535645	7411.317383
...	...	...	...	...	...
2293	2020-12-27	26439.373047	28288.839844	25922.769531	26272.294922
2294	2020-12-28	26280.822266	27389.111328	26207.640625	27084.808594

```
2295 2020-12-29  27081.810547  27370.720703  25987.298828  27362.437500
2296 2020-12-30  27360.089844  28937.740234  27360.089844  28840.953125
2297 2020-12-31  28841.574219  29244.876953  28201.992188  29001.720703

[366 rows x 5 columns]
```

[18]:
```
monthvise= y_2020.groupby(y_2020['date'].dt.strftime('%B'))[['open','close']].
 ↪mean()
new_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July',␣
 ↪'August',
             'September', 'October', 'November', 'December']
monthvise = monthvise.reindex(new_order, axis=0)
monthvise
```

[18]:

date	open	close
January	8318.949597	8389.270476
February	9656.215113	9630.722185
March	6943.507009	6871.016113
April	7150.611328	7224.477328
May	9237.761530	9263.151745
June	9499.797005	9489.227214
July	9519.383852	9589.899729
August	11639.097215	11652.394185
September	10689.700163	10660.276856
October	11791.307491	11886.978201
November	16450.121647	16645.757422
December	21680.540827	21983.137097

[19]:
```
fig = go.Figure()

fig.add_trace(go.Bar(
    x=monthvise.index,
    y=monthvise['open'],
    name='Stock Open Price',
    marker_color='crimson'
))
fig.add_trace(go.Bar(
    x=monthvise.index,
    y=monthvise['close'],
    name='Stock Close Price',
    marker_color='lightsalmon'
))

fig.update_layout(barmode='group', xaxis_tickangle=-45,
                  title='Monthwise comparision between Stock open and close␣
 ↪price')
```

```
fig.show()
```

[20]:
```
y_2020.groupby(y_2020['date'].dt.strftime('%B'))['low'].min()
monthvise_high = y_2020.groupby(data['date'].dt.strftime('%B'))['high'].max()
monthvise_high = monthvise_high.reindex(new_order, axis=0)

monthvise_low = y_2020.groupby(y_2020['date'].dt.strftime('%B'))['low'].min()
monthvise_low = monthvise_low.reindex(new_order, axis=0)

fig = go.Figure()
fig.add_trace(go.Bar(
    x=monthvise_high.index,
    y=monthvise_high,
    name='Stock high Price',
    marker_color='rgb(0, 153, 204)'
))
fig.add_trace(go.Bar(
    x=monthvise_low.index,
    y=monthvise_low,
    name='Stock low Price',
    marker_color='rgb(255, 128, 0)'
))

fig.update_layout(barmode='group',
                  title=' Monthwise High and Low stock price')
fig.show()
```

[21]:
```
names = cycle(['Stock Open Price','Stock Close Price','Stock High Price','Stock␣
 ↪Low Price'])

fig = px.line(y_2020, x=y_2020.date, y=[y_2020['open'], y_2020['close'],
                                        y_2020['high'], y_2020['low']],
              labels={'Date': 'Date','value':'Stock value'})
fig.update_layout(title_text='Stock analysis chart', font_size=15,␣
 ↪font_color='black',legend_title_text='Stock Parameters')
fig.for_each_trace(lambda t:  t.update(name = next(names)))
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)

fig.show()
```

[22]:
```
y_overall=data
y_overall.drop(y_overall[['adj_close','volume']],axis=1)
```

[22]:

	date	open	high	low	close
0	2014-09-17	465.864014	468.174011	452.421997	457.334015
1	2014-09-18	456.859985	456.859985	413.104004	424.440002

```
2    2014-09-19    424.102997    427.834991    384.532013    394.795990
3    2014-09-20    394.673004    423.295990    389.882996    408.903992
4    2014-09-21    408.084991    412.425995    393.181000    398.821014
...         ...           ...           ...           ...           ...
2708 2022-02-15  42586.464844  44667.218750  42491.035156  44575.203125
2709 2022-02-16  44578.277344  44578.277344  43456.691406  43961.859375
2710 2022-02-17  43937.070313  44132.972656  40249.371094  40538.011719
2711 2022-02-18  40552.132813  40929.152344  39637.617188  40030.976563
2712 2022-02-19  40022.132813  40246.027344  40010.867188  40126.429688

[2713 rows x 5 columns]
```

[23]:
```
monthvise= y_overall.groupby(y_overall['date'].dt.
 ↪strftime('%B'))[['open','close']].mean()
new_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July',␣
 ↪'August',
             'September', 'October', 'November', 'December']
monthvise = monthvise.reindex(new_order, axis=0)
```

[24]:
```
names = cycle(['Stock Open Price','Stock Close Price','Stock High Price','Stock␣
 ↪Low Price'])

fig = px.line(y_overall, x=y_overall.date, y=[y_overall['open'],␣
 ↪y_overall['close'],
                                              y_overall['high'], y_overall['low']],
              labels={'Date': 'Date','value':'Stock value'})
fig.update_layout(title_text='Stock analysis chart', font_size=15,␣
 ↪font_color='black',legend_title_text='Stock Parameters')
fig.for_each_trace(lambda t:  t.update(name = next(names)))
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)

fig.show()
```

[25]:
```
#Creating new dataframe of Bitcoin data containing only date and closing price
closedf = data[['date','close']]
print("Shape of close dataframe:", closedf.shape)
```

```
Shape of close dataframe: (2713, 2)
```

[26]:
```
closedf = closedf[closedf['date'] > '2020-09-13']
close_stock = closedf.copy()
print("Total data for prediction: ",closedf.shape[0])
```

```
Total data for prediction:  524
```

[27]:
```
del closedf['date']
scaler=MinMaxScaler(feature_range=(0,1))
```

```
closedf=scaler.fit_transform(np.array(closedf).reshape(-1,1))
print(closedf.shape)
```

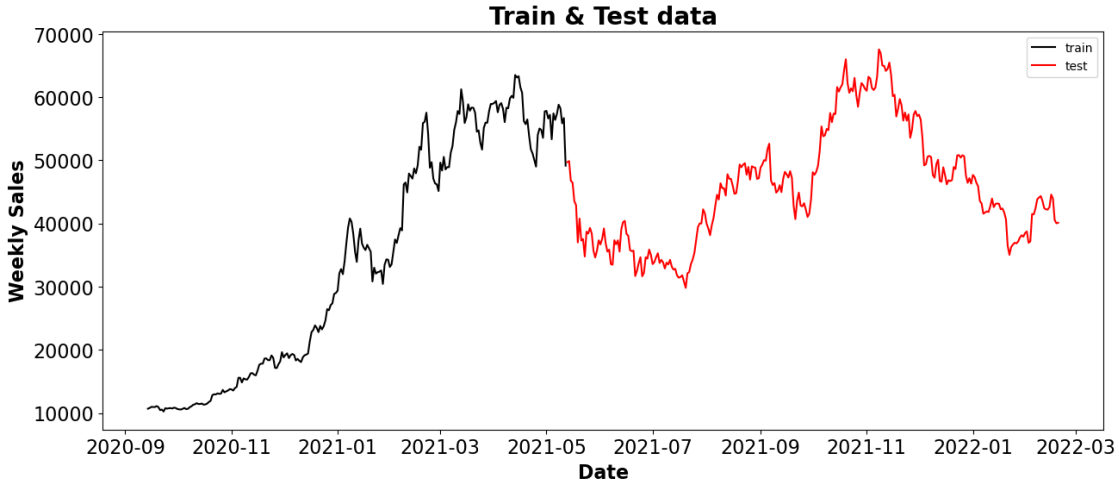
```
(524, 1)
```

[28]:
```
training_size=int(len(closedf)*0.70)
test_size=len(closedf)-training_size
train_data,test_data=closedf[0:training_size,:],closedf[training_size:
 ↪len(closedf),:1]
print("train_data: ", train_data.shape)
print("test_data: ", test_data.shape)
```

```
train_data:  (366, 1)
test_data:  (158, 1)
```

[29]:
```
fig, ax = plt.subplots(figsize=(15, 6))
sns.lineplot(x = close_stock['date'][:241], y = close_stock['close'][:241],␣
 ↪color = 'black')
sns.lineplot(x = close_stock['date'][241:], y = close_stock['close'][241:],␣
 ↪color = 'red')

# Formatting
ax.set_title('Train & Test data', fontsize = 20, loc='center',␣
 ↪fontdict=dict(weight='bold'))
ax.set_xlabel('Date', fontsize = 16, fontdict=dict(weight='bold'))
ax.set_ylabel('Weekly Sales', fontsize = 16, fontdict=dict(weight='bold'))
plt.tick_params(axis='y', which='major', labelsize=16)
plt.tick_params(axis='x', which='major', labelsize=16)
plt.legend(loc='upper right' ,labels = ('train', 'test'))
plt.show()
```



[30]:
```
# 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]
        dataX.append(a)
        dataY.append(dataset[i + time_step, 0])
    return np.array(dataX), np.array(dataY)
```

[31]:
```
time_step = 21
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)

print("X_train: ", X_train.shape)
print("y_train: ", y_train.shape)
print("X_test: ", X_test.shape)
print("y_test", y_test.shape)
```

```
X_train:  (344, 21)
y_train:  (344,)
X_test:  (136, 21)
y_test (136,)
```

[32]:
```
# Biulding Model

my_model = XGBRegressor(n_estimators=1000)
my_model.fit(X_train, y_train, verbose=False)
```

[32]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.300000012,
             max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
             monotone_constraints='()', n_estimators=1000, n_jobs=8,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
             validate_parameters=1, verbosity=None)
```

[33]:
```
predictions = my_model.predict(X_test)
print("Mean Absolute Error - MAE : " + str(mean_absolute_error(y_test,␣
 ↪predictions)))
print("Root Mean squared Error - RMSE : " + str(math.
 ↪sqrt(mean_squared_error(y_test, predictions))))
```

```
Mean Absolute Error - MAE : 0.04643015682889229
Root Mean squared Error - RMSE : 0.06031498357634065
```

[34]:
```
train_predict=my_model.predict(X_train)
test_predict=my_model.predict(X_test)

train_predict = train_predict.reshape(-1,1)
test_predict = test_predict.reshape(-1,1)

print("Train data prediction:", train_predict.shape)
print("Test data prediction:", test_predict.shape)
```

```
Train data prediction: (344, 1)
Test data prediction: (136, 1)
```

[35]:
```
# Transform back to original form

train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
original_ytrain = scaler.inverse_transform(y_train.reshape(-1,1))
original_ytest = scaler.inverse_transform(y_test.reshape(-1,1))
```

[36]:
```
# shift train predictions for plotting

look_back=time_step
trainPredictPlot = np.empty_like(closedf)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
print("Train predicted data: ", trainPredictPlot.shape)

# shift test predictions for plotting
testPredictPlot = np.empty_like(closedf)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(closedf)-1, :] =␣
 ↪test_predict
print("Test predicted data: ", testPredictPlot.shape)

names = cycle(['Original close price','Train predicted close price','Test␣
 ↪predicted close price'])

plotdf = pd.DataFrame({'date': close_stock['date'],
                       'original_close': close_stock['close'],
                      'train_predicted_close': trainPredictPlot.reshape(1,-1)[0].
 ↪tolist(),
                       'test_predicted_close': testPredictPlot.reshape(1,-1)[0].
 ↪tolist()})

fig = px.line(plotdf,x=plotdf['date'],␣
 ↪y=[plotdf['original_close'],plotdf['train_predicted_close'],
                                          plotdf['test_predicted_close']],
```

```
              labels={'value':'Close price','date': 'Date'})
fig.update_layout(title_text='Comparision between original close price vs␣
 ↪predicted close price',
                  plot_bgcolor='white', font_size=15,␣
 ↪font_color='black',legend_title_text='Close Price')
fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

```
Train predicted data:  (524, 1)
Test predicted data:  (524, 1)
```

[37]:
```
x_input=test_data[len(test_data)-time_step:].reshape(1,-1)
temp_input=list(x_input)
temp_input=temp_input[0].tolist()

from numpy import array

lst_output=[]
n_steps=time_step
i=0
pred_days = 10
while(i<pred_days):

    if(len(temp_input)>time_step):

        x_input=np.array(temp_input[1:])
        #print("{} day input {}".format(i,x_input))
        x_input=x_input.reshape(1,-1)

        yhat = my_model.predict(x_input)
        #print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat.tolist())
        temp_input=temp_input[1:]
        lst_output.extend(yhat.tolist())
        i=i+1

    else:
        yhat = my_model.predict(x_input)

        temp_input.extend(yhat.tolist())
        lst_output.extend(yhat.tolist())

        i=i+1
```

```
print("Output of predicted next days: ", len(lst_output))
```

```
Output of predicted next days:  10
```

[38]:
```
last_days=np.arange(1,time_step+1)
day_pred=np.arange(time_step+1,time_step+pred_days+1)
print(last_days)
print(day_pred)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21]
[22 23 24 25 26 27 28 29 30 31]
```

[39]:
```
temp_mat = np.empty((len(last_days)+pred_days+1,1))
temp_mat[:] = np.nan
temp_mat = temp_mat.reshape(1,-1).tolist()[0]

last_original_days_value = temp_mat
next_predicted_days_value = temp_mat

last_original_days_value[0:time_step+1] = scaler.
 ↪inverse_transform(closedf[len(closedf)-time_step:]).reshape(1,-1).tolist()[0]
next_predicted_days_value[time_step+1:] = scaler.inverse_transform(np.
 ↪array(lst_output).reshape(-1,1)).reshape(1,-1).tolist()[0]

new_pred_plot = pd.DataFrame({
    'last_original_days_value':last_original_days_value,
    'next_predicted_days_value':next_predicted_days_value
})

names = cycle(['Last 15 days close price','Predicted next 10 days close price'])

fig = px.line(new_pred_plot,x=new_pred_plot.index,␣
 ↪y=[new_pred_plot['last_original_days_value'],
                                                      ␣
 ↪new_pred_plot['next_predicted_days_value']],
              labels={'value': 'Close price','index': 'Timestamp'})
fig.update_layout(title_text='Compare last 15 days vs next 10 days',
                  plot_bgcolor='white', font_size=15,␣
 ↪font_color='black',legend_title_text='Close Price')
fig.for_each_trace(lambda t:  t.update(name = next(names)))
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

[41]:
```
my_model=closedf.tolist()
my_model.extend((np.array(lst_output).reshape(-1,1)).tolist())
my_model=scaler.inverse_transform(my_model).reshape(1,-1).tolist()[0]
```

```
names = cycle(['Close Price'])

fig = px.line(my_model,labels={'value': 'Close price','index': 'Timestamp'})
fig.update_layout(title_text='Plotting whole closing price with prediction',
                  plot_bgcolor='white', font_size=15,␣
 ↪font_color='black',legend_title_text='Stock')
fig.for_each_trace(lambda t:  t.update(name = next(names)))
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

[ ]: