

Machine Learning Project Report on "Bitcoin EDA and Prediction"

By Rajat Kashyap (1810991554)

Problem Statement:-

To predict the price/value of bitcoin using the "Bitcoin Historical Dataset"

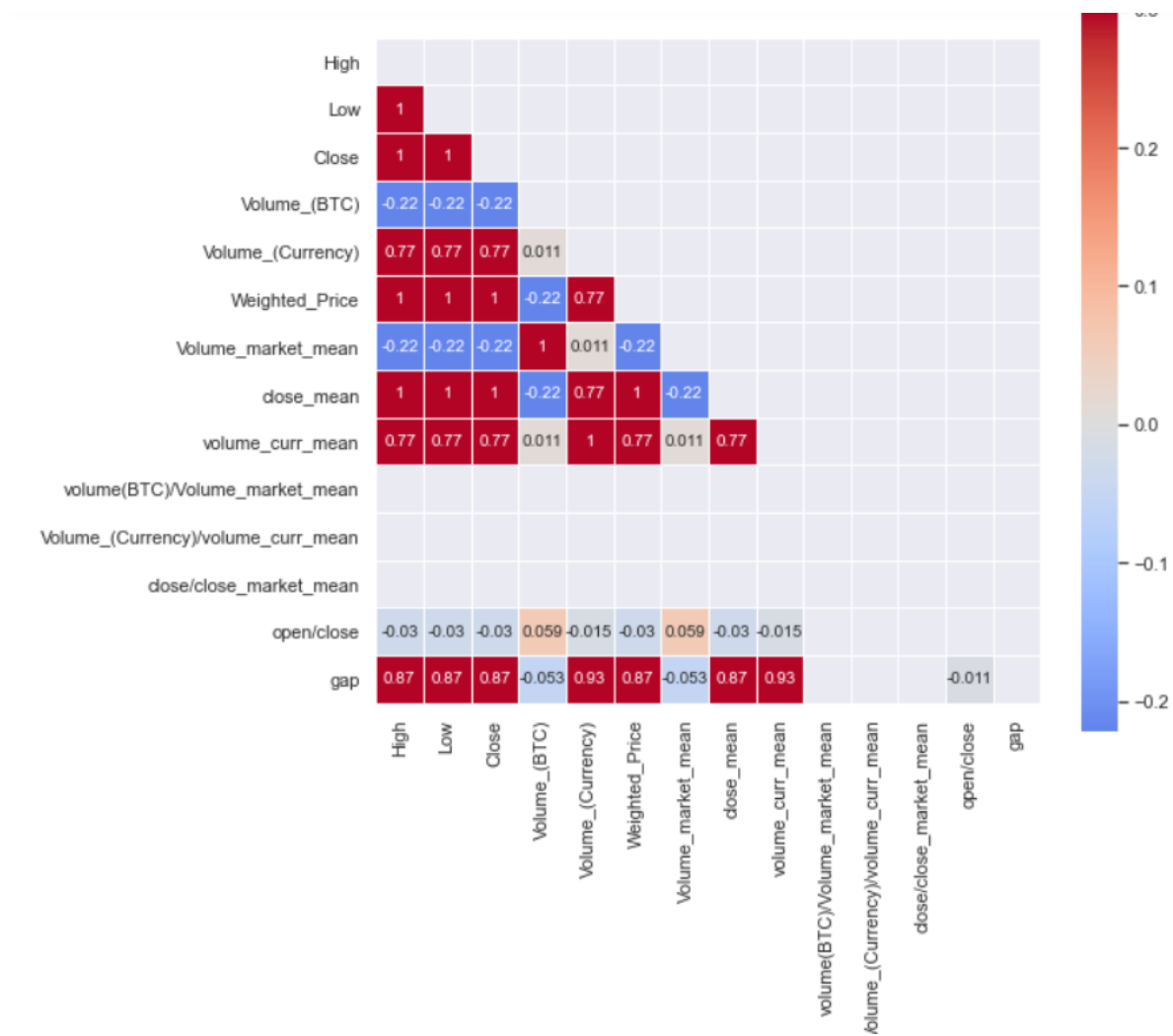
Dataset Description:-

This dataset contains all of the data values(like opening price, closing price) regarding the cryptocurrency "bitcoin" from January 2012 to March 2021

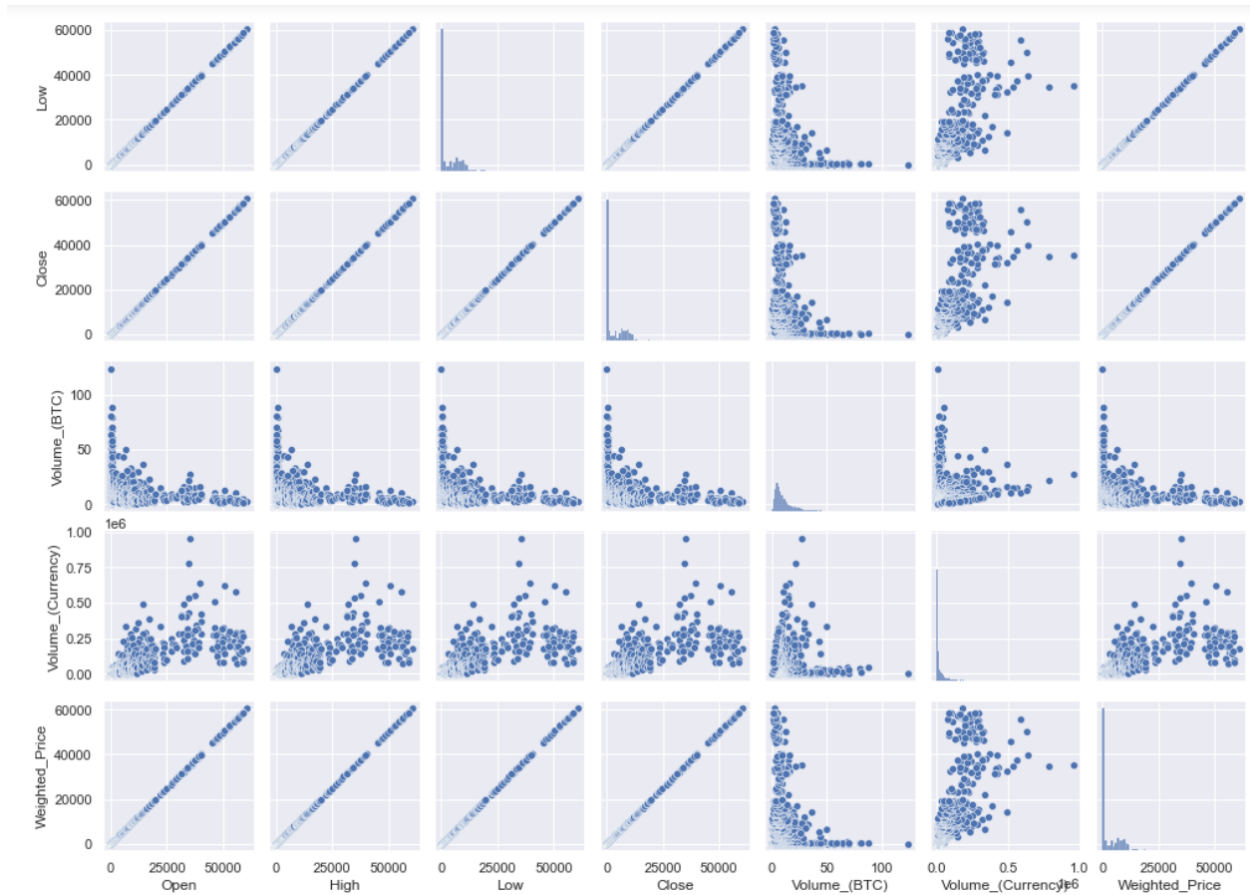
Feature Engineering :-

Opening price, Closing price, Highest Price, Lowest Price, Volume_market_mean

Correlation Metrics:-



Regression Plots:-



Checking for Multicollinearity

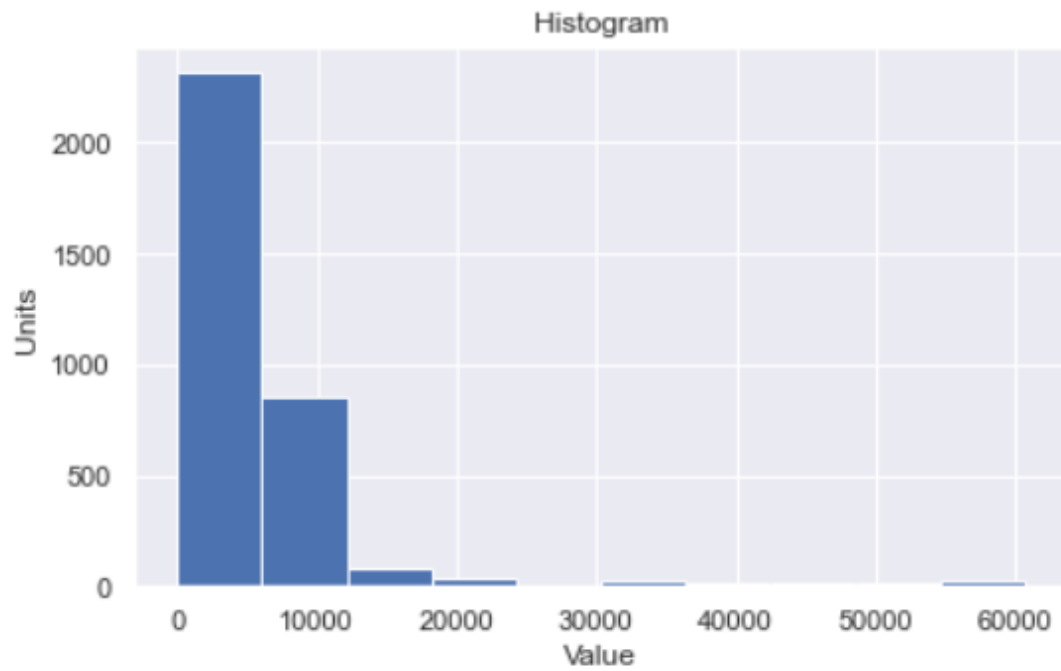
Feature Selection:-

The following columns were selected -

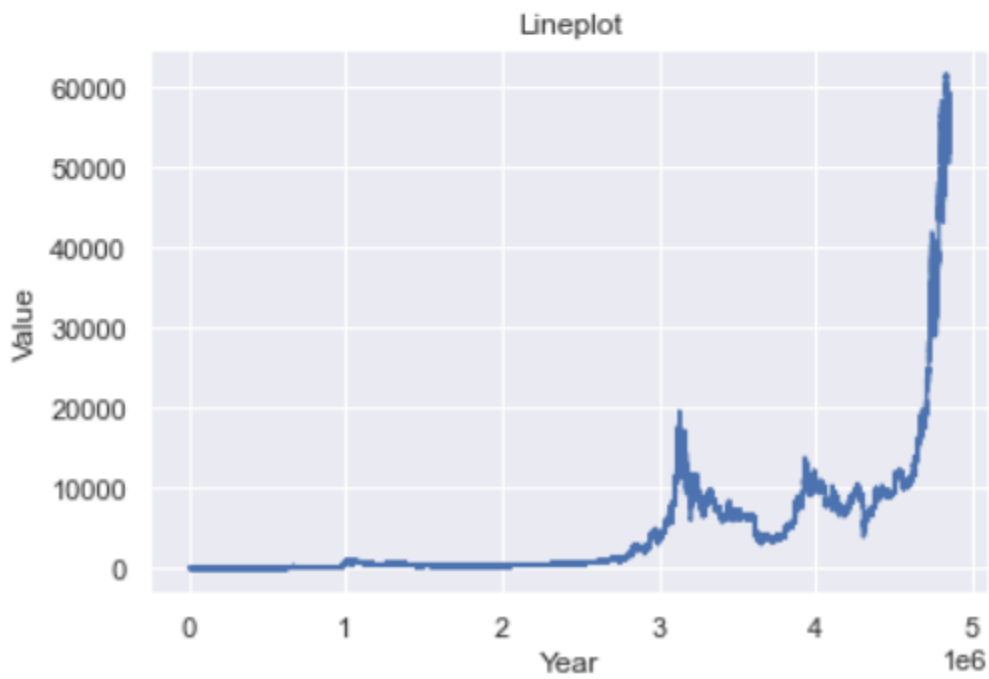
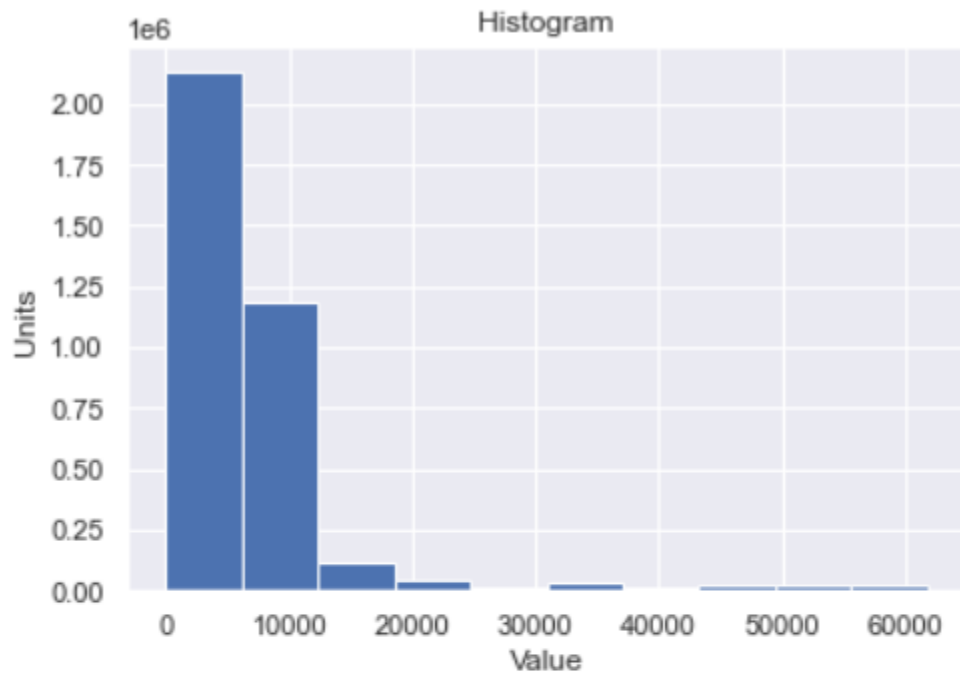
Open, High, Low, Close, Volume_(BTC), Volume_(Currency),
Weighted_Price

Data Visualization-Other Plots:-

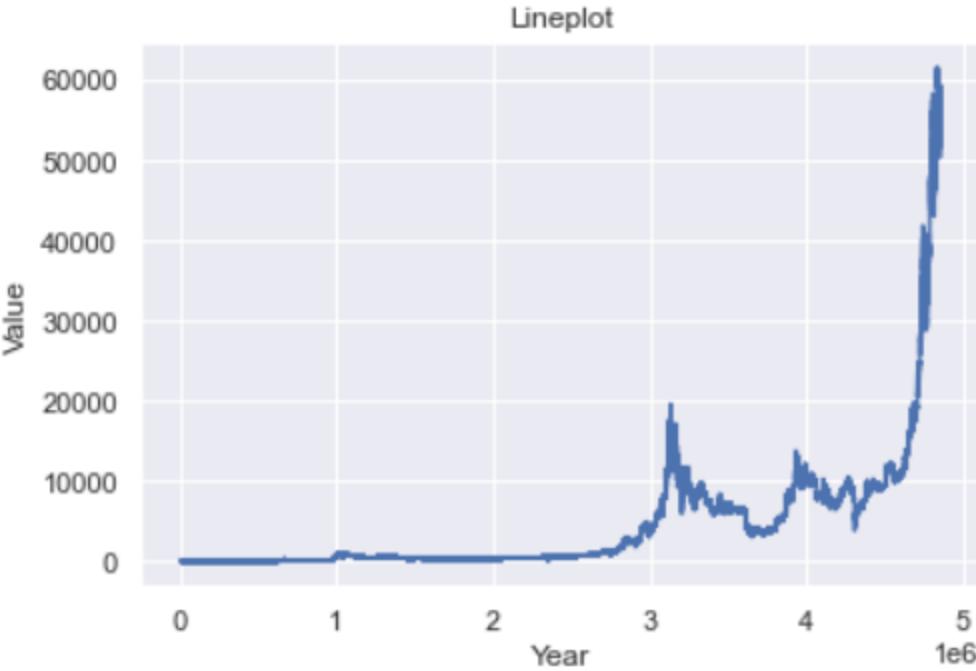
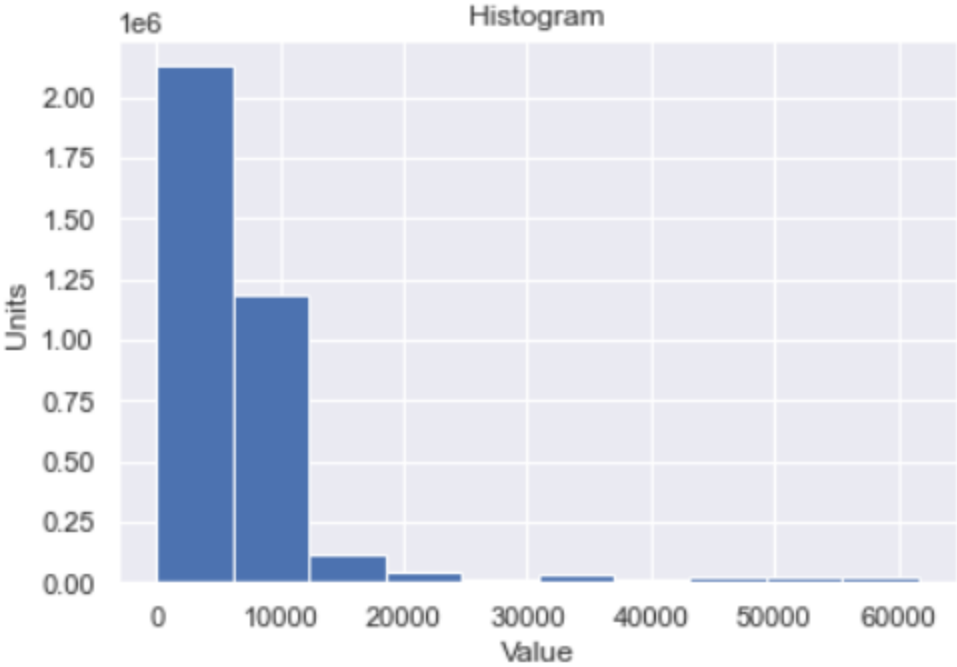
DISTRIBUTION AT OPENING PRICE



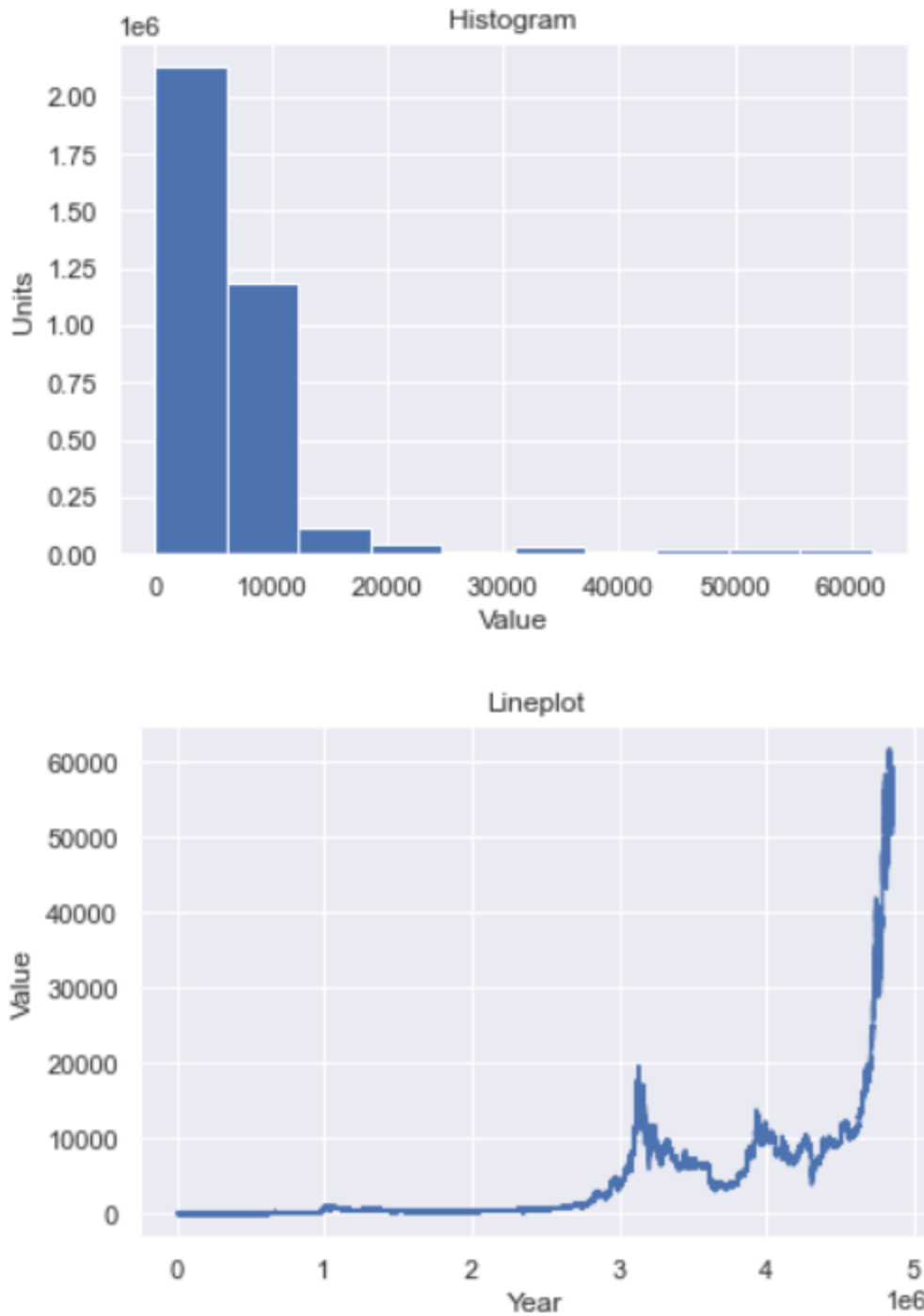
DISTRIBUTION OF HIGHEST PRICE



DISTRIBUTION OF LOWEST PRICE



DISTRIBUTION OF CLOSING PRICE



Selection of Machine Learning Algorithms:-

- LSTM(Long Short Term Memory)

- **XGBoost**(eXtreme Gradient Boosting)
- Prophet(By Facebook)
- ARIMA(AutoRegressive Integrated Moving Average)
- RNN(Recurrent neural network).

Create all models:-

LSTM-

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers import Activation
```

```
model = Sequential()
model.add(LSTM(128,activation="sigmoid",input_shape=(1,1)))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, y_train, epochs=100, batch_size=50, verbose=2)
```

XGBoost-

```
X_train, y_train = create_features(data_train, label='Weighted_Price')
X_test, y_test = create_features(data_test, label='Weighted_Price')
import xgboost as xgb
from xgboost import plot_importance, plot_tree
model = xgb.XGBRegressor(objective='reg:linear',min_child_weight=10,
booster='gbtree', colsample_bytree = 0.3, learning_rate = 0.1,
max_depth = 5, alpha = 10, n_estimators = 100)
model.fit(X_train, y_train,
eval_set=[(X_train, y_train), (X_test, y_test)],
early_stopping_rounds=50,
verbose=True)
```



```
Prophet-  
model = Prophet()  
model.fit(data_train)
```

```
ARIMA  
results = []  
best_aic = float("inf")  
warnings.filterwarnings('ignore')  
for param in parameters_list:  
    try:  
        model=sm.tsa.statespace.SARIMAX(data.Weighted_Price, order=(param[0], d,  
param[1]),  
                                         seasonal_order=(param[2], D, param[3],  
12),enforce_stationarity=False,  
                                         enforce_invertibility=False).fit(dispatch=-1)  
    except ValueError:  
        #print('wrong parameters:', param)  
        continue  
    aic = model.aic  
    if aic < best_aic:  
        best_model = model  
        best_aic = aic  
        best_param = param  
    results.append([param, model.aic])  
  
result_table = pd.DataFrame(results)  
result_table.columns = ['parameters', 'aic']  
print(result_table.sort_values(by = 'aic', ascending=True).head())  
print(best_model.summary())
```

```
RNN-  
from keras.models import Sequential  
from keras.layers import Dense  
from keras.layers import LSTM
```

```
regressor = Sequential()
```

```
regressor.add(LSTM(units = 4, activation = 'sigmoid', input_shape = (None, 1)))
```

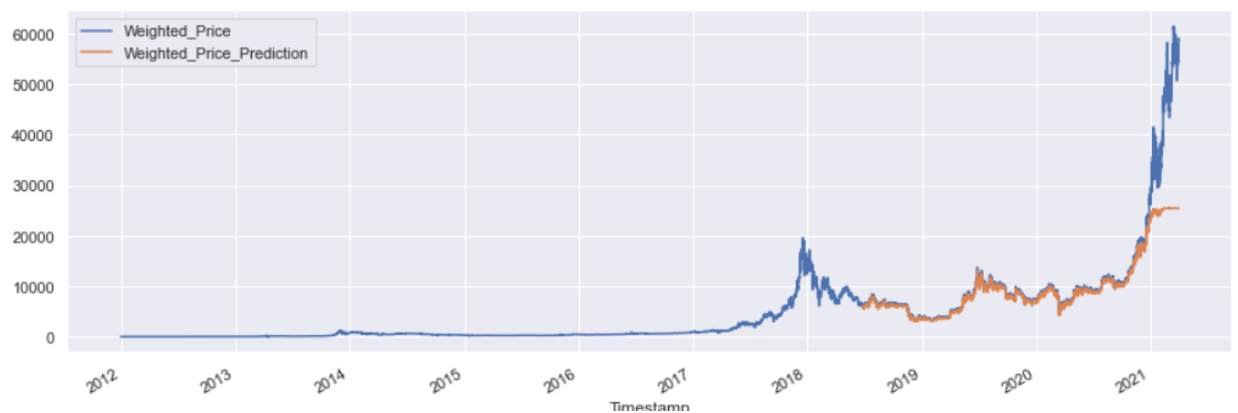
```
regressor.add(Dense(units = 1))
```

```
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

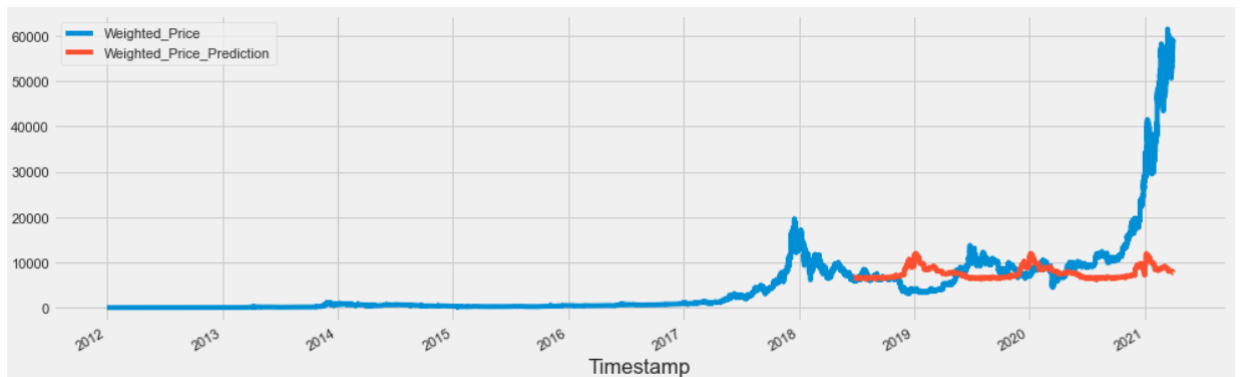
```
regressor.fit(X_train, y_train, batch_size = 5, epochs = 100)
```

Plot Classification Summary:-

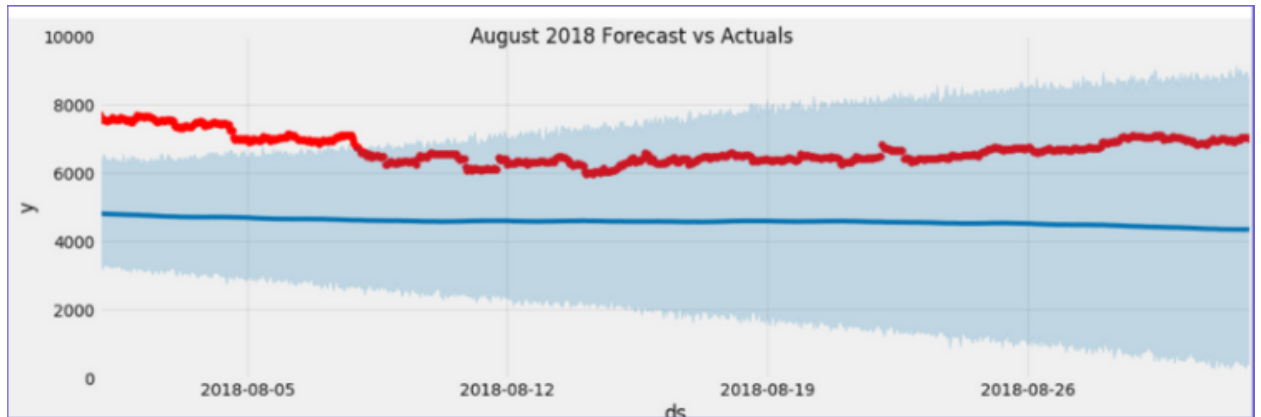
LSTM



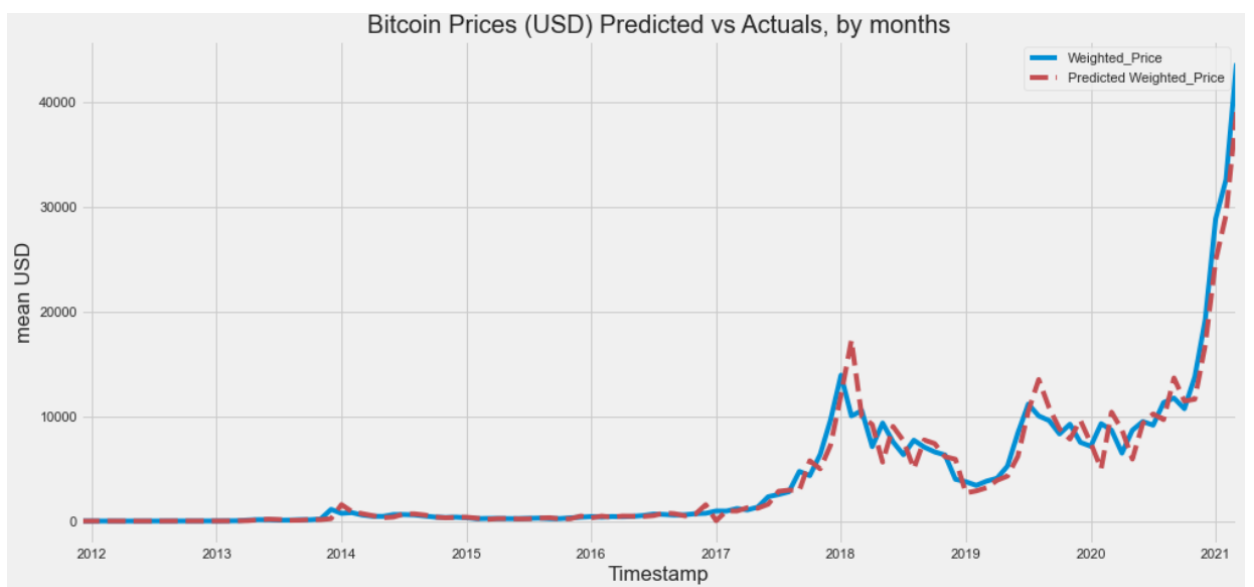
XGBoost



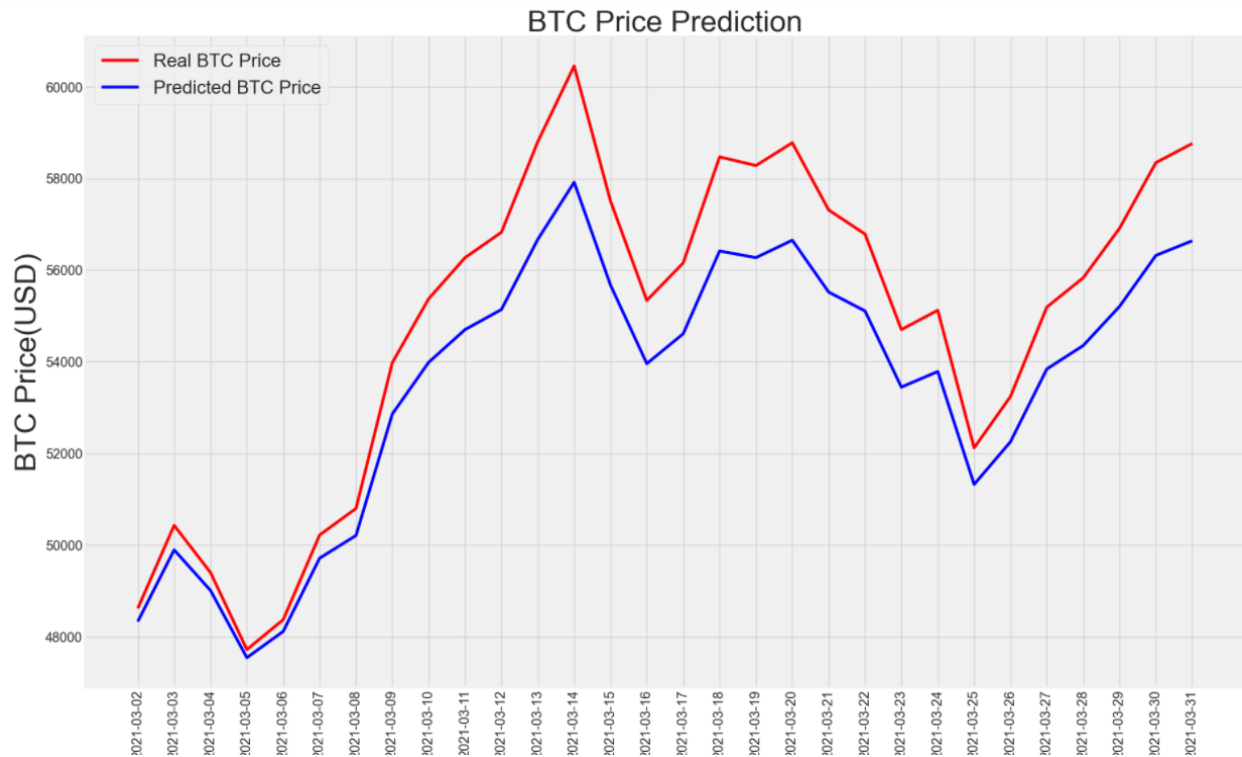
Prophet



ARIMA



RNN



FINAL CODE:-

```

1  Final Code:-<div>
2      <h1><center style="background-color:#87CEFA;
   color:white; font-size: 80px;"> Bitcoin
3          EDA and Prediction</center></h1>
4 </div>
5
6
7 <div>
8 
9 </div>
10
11 <a id="top"></a>
12
13 <div class="list-group" id="list-tab" role="tablist">

```

```

14 <h3 class="list-group-item list-group-item-action active"
    data-toggle="list"
    style='background-color:#87CEFA;font-size: 50px; border:0'
    role="tab" aria-controls="home" color=black><center>Quick
    navigation</center></h3>
15
16 * [1. Required Libraries](#1)
17 * [2. Dataset Quick Overview & Pre-Processing](#2)
18 * [3. Features generation](#3)
19 * [4. Distribution of Features](#4)
20 * [5. Correlation Analysis ](#5)
21
22
23
24
25 <div class="alert alert-info">
26 <h3><center>Bitcoin[a] (฿) is a cryptocurrency invented in
    2008 by an unknown person or group of people using the name
    Satoshi Nakamoto.</center></h3>
27 <h4>Some interesting facts about Bitcoin(BTC):</h4>
28 <h5>1.Bitcoin is a decentralized digital currency, without
    a central bank or single administrator, that can be sent
    from user to user on the peer-to-peer bitcoin network
    without the need for intermediaries. Transactions are
    verified by network nodes through cryptography and recorded
    in a public distributed ledger called a blockchain.</h5>
29 <h5>2. In fact, there are only 21 million bitcoins that can
    be mined in total.Once miners have unlocked this amount of
    bitcoins, the supply will be exhausted.</h5>
30 <h5>3. Currently, around 18.5 million bitcoin have been
    mined. This leaves less than three million that have yet to
    be introduced into circulation.</h5>
31 </div>
32
33 <a id="1"></a>
34 <h2 style='background-color:#87CEFA; border:0;

```

```

    color:black'><center>Required Libraries</center><h2>
35
36
37 #Data Pre-Processing packages:
38 import numpy as np
39 import pandas as pd
40 from datetime import datetime
41
42
43 #Data Visualization Packages:
44 #Seaborn
45 import seaborn as sns
46 sns.set(rc={'figure.figsize':(10,6)})
47 custom_colors = ["#4e89ae", "#c56183", "#ed6663", "#ffa372"]
48
49 #Matplotlib
50 import matplotlib.pyplot as plt
51 %matplotlib inline
52 import matplotlib.image as mpimg
53
54 #Colorama
55 from colorama import Fore, Back, Style # For text colors
56 y_ = Fore.CYAN
57 m_ = Fore.WHITE
58
59 #garbage collector - To free up unused space
60 import gc
61 gc.collect()
62
63
64 #Time -To find how long each cell takes to run
65 import time
66 import datetime
67
68 #Importing of Data
69 data=pd.read_csv(r'C:\Users\rajat\Downloads\bitcoindata.csv
    ')

```

```

70
71 import warnings
72 warnings.simplefilter(action="ignore",
    category=FutureWarning)
73
74
75
76 <a id="2"></a>
77 <h2 style='background-color:#87CEFA; border:0;
    color:black'><center>Data set Overview &
    Pre-Processing</center><h2>
78
79 print(f"{m_}Total records:{y_}{data.shape}\n")
80 print(f"{m_}Data types of data columns:
    \n{y_}{data.dtypes}")
81
82 <div class="alert alert-info">
83     <h3 color:black><center><b> Data Pre-processing
    steps</b></center></h3>
84     <p> 1. <b>Date</b> - We need to convert the Hourly data
    to Daily based data </p>
85     <p> 2. <b>Fill in</b> the missing values interpolation</p>
86
87 </div>
88
89 <div class='alert alert-info'>
90 <h3> The data is available on a Hourly based on each day,
    So we need to resample them to day based.</h3>
91 </div>
92
93 data['Timestamp'] = [datetime.datetime.fromtimestamp(x) for
    x in data['Timestamp']]
94 data = data.set_index('Timestamp')
95 data = data.resample("24H").mean()
96 data.head()
97

```

```

98 <div class="alert alert-info">
99     <h3><b><center>Missing values</center><b></h3>
100 </div>
101
102
103 missed = pd.DataFrame()
104 missed['column'] = data.columns
105
106 missed['percent'] = [round(100* data[col].isnull().sum() /
    len(data), 2) for col in data.columns]
107 missed = missed.sort_values('percent',ascending=False)
108 missed = missed[missed['percent']>0]
109
110 fig = sns.barplot(
111     x=missed['percent'],
112     y=missed["column"],
113     orientation='horizontal',palette="winter"
114 ).set_title('Missed values percent for every column')
115
116 data.isnull().sum()
117
118 <div class="alert alert-info">
119     <h3><b><center>Let's interpolate to fill in the
    missing values</center><b></h3>
120 </div>
121
122
123 def fill_missing(df):
124     """ function to impute missing values using
    interpolation """
125     df['Open'] = df['Open'].interpolate()
126     df['Close'] = df['Close'].interpolate()
127     df['Weighted_Price'] =
    df['Weighted_Price'].interpolate()
128     df['Volume_(BTC)'] = df['Volume_(BTC)'].interpolate()
129     df['Volume_(Currency)'] =

```



```

    df['Volume_(Currency)'].interpolate()
130     df['High'] = df['High'].interpolate()
131     df['Low'] = df['Low'].interpolate()
132     print(f'{m_}No. of Missing values after
    interpolation:\n{y_}{df.isnull().sum()}')
133 fill_missing(data)
134
135 <a id="3"></a>
136 <h2 style='background-color:#87CEFA; border:0;
    color:black'><center>Feature Generation</center><h2>
137
138 data.columns
139
140 new_df=data.groupby('Timestamp').mean()
141 new_df=new_df[['Volume_(BTC)',
    'Close','Volume_(Currency)']]
142 new_df.rename(columns={'Volume_(BTC)': 'Volume_market_mean'
    , 'Close': 'close_mean', 'Volume_(Currency)': 'volume_curr_mean'
    }, inplace=True)
143 new_df.head()
144
145 data_df = data.merge(new_df, left_on='Timestamp',
146                     right_index=True)
147 data_df['volume(BTC)/Volume_market_mean'] =
    data_df['Volume_(BTC)'] / data_df['Volume_market_mean']
148 data_df['Volume_(Currency)/volume_curr_mean'] =
    data_df['Volume_(Currency)'] / data_df['volume_curr_mean']
149
150 data_df['close/close_market_mean'] = data_df['Close'] /
    data_df['close_mean']
151 data_df['open/close'] = data_df['Open'] / data_df['Close']
152 data_df["gap"] = data_df["High"] - data_df["Low"]
153 data_df.head()
154
155 <div class='alert alert-info'>
156     <p> Sometimes, the data set might be too huge to
    process, since we are using dataframe. To make sure we dont

```

```

    hold up too much RAM. We could try other approaches
    like</p>
157     <p> 1. use gc.collect() - collects all the garbage
    values </p>
158     <p> 2. del dataframe - free up some space by deleting
    the unused dataframe using the del command </p>
159     <p> 3. Reduce the memory usage based on the data
    types of the columns in the dataframe(shown below)</p>
160 </div>
161
162 def mem_usage(pandas_obj):
163     if isinstance(pandas_obj, pd.DataFrame):
164         usage_b = pandas_obj.memory_usage(deep=True).sum()
165     else: # we assume if not a df it's a series
166         usage_b = pandas_obj.memory_usage(deep=True)
167     usage_mb = usage_b / 1024 ** 2 # convert bytes to
    megabytes
168     return "{:03.2f} MB".format(usage_mb)
169 print(f'{m_}Memory of the
    dataframe:\n{y_}{mem_usage(data_df)}')
170
171 #All the columns in float64 format, we can downsize them
    to float32 to reduce memory usage
172 data_df.info()
173
174
175 <div class='alert alert-info'>
176 <h3>We can use the function pd.to_numeric() to downcast
    our float types. We'll use DataFrame.select_dtypes to
    select only the float columns, then we'll optimize the
    types and compare the memory usage.</h3>
177 </div>
178
179 gl_float = data_df.select_dtypes(include=['float'])
180 converted_float =
    gl_float.apply(pd.to_numeric, downcast='float')
181 compare_floats =

```

```

    pd.concat([gl_float.dtypes,converted_float.dtypes],axis=1)
182 compare_floats.columns = ['Before','After']
183 compare_floats.apply(pd.Series.value_counts)
184
185 print(f'{m_}Before float
    conversion:\n{y_}{mem_usage(data_df)}")
186 data_df[converted_float.columns] = converted_float
187 print(f'{m_}After float
    conversion:\n{y_}{mem_usage(data_df)}")
188
189 <div class='alert alert-info'>
190     <h3>We have successfully reduced the size of the
    dataframe by 50%. Eventhough, the size of the dataframe
    used here is small in this analysis. It's always a good
    approach to reduce the memory usage</h3>
191 </div>
192
193 <a id="4"></a>
194 <h2 style='background-color:#87CEFA; border:0;
    color:black'><center>Distribution of Features</center><h2>
195
196 <div class="alert alert-info">
197     <h3><b>Let's Visualize the distribution of the key
    variables like Opening price, Highest price, Lowest price
    and Volume in Bitcoin<b></h3>
198 </div>
199
200
201
202 print("                                DISTRIBUTION AT
    OPENING PRICE")
203
204 plt.hist(data['Open'])
205 plt.title("Histogram")
206 plt.xlabel("Value")
207 plt.ylabel("Units")
208 plt.show()

```

```
209
210 plt.plot(data['Open'])
211 plt.title("Lineplot")
212 plt.xlabel("Year")
213 plt.ylabel("Value")
214 plt.show()
```

```
215
```

```
216
```

```
217
```

```
218 print("
    HIGHEST PRICE")
```

DISTRIBUTION OF

```
219
```

```
220 plt.hist(data['High'])
221 plt.title("Histogram")
222 plt.xlabel("Value")
223 plt.ylabel("Units")
224 plt.show()
```

```
225
```

```
226 plt.plot(data['High'])
227 plt.title("Lineplot")
228 plt.xlabel("Year")
229 plt.ylabel("Value")
230 plt.show()
```

```
231
```

```
232
```

```
233
```

```
234
```

```
235 print("
    LOWEST PRICE")
```

DISTRIBUTION OF

```
236
```

```
237 plt.hist(data['Low'])
238 plt.title("Histogram")
239 plt.xlabel("Value")
240 plt.ylabel("Units")
241 plt.show()
```

```
242
```

```

243 plt.plot(data['Low'])
244 plt.title("Lineplot")
245 plt.xlabel("Year")
246 plt.ylabel("Value")
247 plt.show()
248
249
250 print("                                DISTRIBUTION OF
      CLOSING PRICE")
251
252 plt.hist(data['Close'])
253 plt.title("Histogram")
254 plt.xlabel("Value")
255 plt.ylabel("Units")
256 plt.show()
257
258 plt.plot(data['Close'])
259 plt.title("Lineplot")
260 plt.xlabel("Year")
261 plt.ylabel("Value")
262 plt.show()
263
264 <a id="5"></a>
265 <h2 style='background-color:#87CEFA; border:0;
      color:black'><center>Correlation Analysis</center></h2>
266
267 plt.figure(figsize=(8,8))
268 corr=data_df[data_df.columns[1:]].corr()
269 mask = np.triu(np.ones_like(corr, dtype=bool))
270 sns.heatmap(data_df[data_df.columns[1:]].corr(),
      mask=mask, cmap='coolwarm', vmax=.3, center=0,
271             square=True, linewidths=.5, annot=True)
272 plt.show()
273
274 sns.pairplot(data,height=2)
275

```

```

276 <a id="5"></a>
277 <h2 style='background-color:#87CEFA; border:0;
    color:black'><center>Prediction</center></h2>
278
279 ## Predictiong using LSTM
280
281 import datetime, pytz
282 #define a conversion function for the native timestamps in
    the csv file
283 def dateparse (time_in_secs):
284     return
    pytz.utc.localize(datetime.datetime.fromtimestamp(float(time
    e_in_secs)))
285
286
287
288 #data=pd.read_csv(r'C:\Users\rajat\Downloads\bitcoindata.c
    sv')
289 data =
    pd.read_csv(r'C:\Users\rajat\Downloads\bitcoindata.csv',par
    se_dates=[0], date_parser=dateparse)
290 data['Timestamp'] = data['Timestamp'].dt.tz_localize(None)
291 data = data.groupby([pd.Grouper(key='Timestamp',
    freq='H')]).first().reset_index()
292 data = data.set_index('Timestamp')
293 data = data[['Weighted_Price']]
294 data['Weighted_Price'].fillna(method='ffill',
    inplace=True)
295
296 # split data
297 split_date = '25-Jun-2018'
298 data_train = data.loc[data.index <= split_date].copy()
299 data_test = data.loc[data.index > split_date].copy()
300
301
302 # Data preprocess

```

```

303 training_set = data_train.values
304 training_set = np.reshape(training_set,
    (len(training_set), 1))
305 from sklearn.preprocessing import MinMaxScaler
306 sc = MinMaxScaler()
307 training_set = sc.fit_transform(training_set)
308 X_train = training_set[0:len(training_set)-1]
309 y_train = training_set[1:len(training_set)]
310 X_train = np.reshape(X_train, (len(X_train), 1, 1))
311
312 color_pal = ["#F8766D", "#D39200", "#93AA00", "#00BA38",
    "#00C19F", "#00B9E3", "#619CFF", "#DB72FB"]
313 _ = data.plot(style='', figsize=(15,5),
    color=color_pal[0], title='BTC Weighted_Price Price (USD)
    by Hours')
314
315
316 _ = data_test \
317     .rename(columns={'Weighted_Price': 'Test Set'}) \
318     .join(data_train.rename(columns={'Weighted_Price':
    'Training Set'}), how='outer') \
319     .plot(figsize=(15,5), title='BTC Weighted_Price Price
    (USD) by Hours', style='')
320
321 # Importing the Keras libraries and packages
322 from keras.models import Sequential
323 from keras.layers import Dense
324 from keras.layers import LSTM
325 from keras.layers import Dropout
326 from keras.layers import Activation
327
328
329 model = Sequential()
330 model.add(LSTM(128,activation="sigmoid",input_shape=(1,1))
    )
331 model.add(Dropout(0.2))

```

```
332 model.add(Dense(1))
333 model.compile(loss='mean_squared_error', optimizer='adam')
334 model.fit(X_train, y_train, epochs=100, batch_size=50,
    verbose=2)
335
336 model.summary()
337
338 # Making the predictions
339 test_set = data_test.values
340 inputs = np.reshape(test_set, (len(test_set), 1))
341 inputs = sc.transform(inputs)
342 inputs = np.reshape(inputs, (len(inputs), 1, 1))
343 predicted_BTC_price = model.predict(inputs)
344 predicted_BTC_price =
    sc.inverse_transform(predicted_BTC_price)
345
346
347 data_test['Weighted_Price_Prediction'] =
    predicted_BTC_price
348 data_all = pd.concat([data_test, data_train], sort=False)
349
350
351 _ =
    data_all[['Weighted_Price', 'Weighted_Price_Prediction']].pl
    ot(figsize=(15, 5))
352
353 f, ax = plt.subplots(1)
354 f.set_figheight(5)
355 f.set_figwidth(15)
356 _ =
    data_all[['Weighted_Price_Prediction', 'Weighted_Price']].pl
    ot(ax=ax,
357
    style=['-', '.'])
358 ax.set_xbound(lower='08-01-2018', upper='09-01-2018')
359 ax.set_ylim(0, 10000)
```



```

360 plot = plt.suptitle('August 2018 Forecast vs Actuals')
361
362 # Plot the forecast with the actuals
363 f, ax = plt.subplots(1)
364 f.set_figheight(5)
365 f.set_figwidth(15)
366 _ =
    data_all[['Weighted_Price_Prediction', 'Weighted_Price']].pl
    ot(ax=ax,
367
    style=['-', '.'])
368 ax.set_xbound(lower='08-01-2018', upper='08-08-2018')
369 ax.set_ylim(0, 10000)
370 plot = plt.suptitle('First Week of August 2018 Forecast vs
    Actuals')
371
372 #calculate MSE and MAE
373 from sklearn.metrics import mean_squared_error,
    mean_absolute_error
374 mean_squared_error(y_true=data_test['Weighted_Price'],
375
    y_pred=data_test['Weighted_Price_Prediction'])
376
377 mean_absolute_error(y_true=data_test['Weighted_Price'],
378
    y_pred=data_test['Weighted_Price_Prediction'])
379
380 ## Predictiong using XGBoost
381
382 import seaborn as sns
383 import matplotlib.pyplot as plt
384 #from fbprophet import Prophet
385 from sklearn.metrics import mean_squared_error,
    mean_absolute_error
386 plt.style.use('fivethirtyeight')
387
388 data =

```

```

    pd.read_csv(r'C:\Users\rajat\Downloads\bitcoindata.csv',par
    se_dates=[0], date_parser=dateparse)
389 data['Timestamp'] = data['Timestamp'].dt.tz_localize(None)
390 data = data.groupby([pd.Grouper(key='Timestamp',
    freq='H')]).first().reset_index()
391 data = data.set_index('Timestamp')
392 data = data[['Weighted_Price']]
393 data['Weighted_Price'].fillna(method='ffill',
    inplace=True)
394
395 color_pal = ["#F8766D", "#D39200", "#93AA00", "#00BA38",
    "#00C19F", "#00B9E3", "#619CFF", "#DB72FB"]
396 _ = data.plot(style='', figsize=(15,5),
    color=color_pal[0], title='BTC Weighted_Price Price (USD)
    by Hours')
397
398 split_date = '25-Jun-2018'
399 data_train = data.loc[data.index <= split_date].copy()
400 data_test = data.loc[data.index > split_date].copy()
401
402 _ = data_test \
403     .rename(columns={'Weighted_Price': 'Test Set'}) \
404     .join(data_train.rename(columns={'Weighted_Price':
    'Training Set'}), how='outer') \
405     .plot(figsize=(15,5), title='BTC Weighted_Price Price
    (USD) by Hours', style='')
406
407 def create_features(df, label=None):
408     """
409     Creates time series features from datetime index
410     """
411     df['date'] = df.index
412     df['hour'] = df['date'].dt.hour
413     df['dayofweek'] = df['date'].dt.dayofweek
414     df['quarter'] = df['date'].dt.quarter
415     df['month'] = df['date'].dt.month
416     df['year'] = df['date'].dt.year

```

```

417     df['dayofyear'] = df['date'].dt.dayofyear
418     df['dayofmonth'] = df['date'].dt.day
419     df['weekofyear'] = df['date'].dt.weekofyear
420     X = df[['hour', 'dayofweek', 'quarter', 'month', 'year',
421            'dayofyear', 'dayofmonth', 'weekofyear']]
422     if label:
423         y = df[label]
424         return X, y
425     return X
426
427 X_train, y_train = create_features(data_train,
428                                   label='Weighted_Price')
429 X_test, y_test = create_features(data_test,
430                                  label='Weighted_Price')
431
432 import xgboost as xgb
433 from xgboost import plot_importance, plot_tree
434 model = xgb.XGBRegressor(objective
435                           = 'reg:linear', min_child_weight=10, booster='gbtree',
436                           colsample_bytree = 0.3, learning_rate = 0.1,
437                           max_depth = 5, alpha = 10, n_estimators =
438                           100)
439 model.fit(X_train, y_train,
440          eval_set=[(X_train, y_train), (X_test, y_test)],
441          early_stopping_rounds=50,
442          verbose=True)
443
444 data_test['Weighted_Price_Prediction'] =
445     model.predict(X_test)
446 data_all = pd.concat([data_test, data_train], sort=False)
447
448 _ =
449     data_all[['Weighted_Price', 'Weighted_Price_Prediction']].plot(figsize=(15, 5))
450
451 # Plot the forecast with the actuals

```

```

445 f, ax = plt.subplots(1)
446 f.set_figheight(5)
447 f.set_figwidth(15)
448 _ =
    data_all[['Weighted_Price_Prediction', 'Weighted_Price']].plot(
        ax=ax,
449
        style=['-', '.'])
450 ax.set_xbound(lower='08-01-2018', upper='09-01-2018')
451 ax.set_ylim(0, 10000)
452 plot = plt.suptitle('August 2018 Forecast vs Actuals')
453
454 # Plot the forecast with the actuals
455 f, ax = plt.subplots(1)
456 f.set_figheight(5)
457 f.set_figwidth(15)
458 _ =
    data_all[['Weighted_Price_Prediction', 'Weighted_Price']].plot(
        ax=ax,
459
        style=['-', '.'])
460 ax.set_xbound(lower='08-01-2018', upper='08-08-2018')
461 ax.set_ylim(0, 10000)
462 plot = plt.suptitle('First Week of August 2018 Forecast vs
    Actuals')
463
464 mean_squared_error(y_true=data_test['Weighted_Price'],
465
    y_pred=data_test['Weighted_Price_Prediction'])
466
467 mean_absolute_error(y_true=data_test['Weighted_Price'],
468
    y_pred=data_test['Weighted_Price_Prediction'])
469
470 ## Predictiong using Prophet
471
472 color_pal = ["#F8766D", "#D39200", "#93AA00", "#00BA38",

```

```

    "#00C19F", "#00B9E3", "#619CFF", "#DB72FB"]
473 _ = data.plot(style='', figsize=(15,5),
    color=color_pal[0], title='BTC Weighted_Price Price (USD)
    by Hours')
474
475 split_date = '25-Jun-2018'
476 data_train = data.loc[data.index <= split_date].copy()
477 data_test = data.loc[data.index > split_date].copy()
478
479
480 data_train =
    data_train.reset_index().rename(columns={'Timestamp':'ds',
    'Weighted_Price':'y'})
481
482 # Setup and train model
483 model = Prophet()
484 model.fit(data_train)
485
486 # Plot the forecast
487 f, ax = plt.subplots(1)
488 f.set_figheight(5)
489 f.set_figwidth(15)
490 fig = model.plot(data_test_fcst, ax=ax)
491
492 # Plot the components
493 fig = model.plot_components(data_test_fcst)
494
495 # Plot the forecast with the actuals
496 f, ax = plt.subplots(1)
497 f.set_figheight(5)
498 f.set_figwidth(15)
499 ax.scatter(data_test.index, data_test['Weighted_Price'],
    color='r')
500 fig = model.plot(data_test_fcst, ax=ax)
501 ax.set_xbound(lower='08-01-2018', upper='08-08-2018')
502 ax.set_ylim(0, 10000)
503 plot = plt.suptitle('First Week of August 2018 Forecast vs

```

```

    Actuals')
504
505 mean_squared_error(y_true=data_test['Weighted_Price'],
506                     y_pred=data_test_fcst['yhat'])
507
508 mean_absolute_error(y_true=data_test['Weighted_Price'],
509                     y_pred=data_test_fcst['yhat'])
510
511 ## Prediction using ARIMA
512
513 from scipy import stats
514 import statsmodels.api as sm
515 import warnings
516 from itertools import product
517
518 data =
    pd.read_csv(r'C:\Users\rajat\Downloads\bitcoindata.csv',par
    se_dates=[0], date_parser=dateparse)
519
520 data['Open'].fillna(method='ffill', inplace=True)
521 data['High'].fillna(method='ffill', inplace=True)
522 data['Low'].fillna(method='ffill', inplace=True)
523 data['Close'].fillna(method='ffill', inplace=True)
524 data['Weighted_Price'].fillna(method='ffill',
    inplace=True)
525 data['Volume_(BTC)'].fillna(method='ffill', inplace=True)
526 data['Volume_(Currency)'].fillna(method='ffill',
    inplace=True)
527
528 plt.figure(figsize=[20,8])
529 plt.title('BTC Weighted_Price Price (USD) by Hours')
530 plt.plot(data.Weighted_Price, '-', label='By Hours')
531
532 data['Timestamp'] = data['Timestamp'].dt.tz_localize(None)
533 data = data.groupby([pd.Grouper(key='Timestamp',
    freq='M')]).first().reset_index()
534 data = data.set_index('Timestamp')

```

```
535 data['Weighted_Price'].fillna(method='ffill',
    inplace=True)
536
537 plt.figure(figsize=[20,8])
538 plt.title('BTC Weighted_Price Price (USD) by Months')
539 plt.plot(data.Weighted_Price, '-', label='By Months')
540
541 decomposition =
    sm.tsa.seasonal_decompose(data.Weighted_Price)
542
543 trend = decomposition.trend
544 seasonal = decomposition.seasonal
545 residual = decomposition.resid
546
547 fig = plt.figure(figsize=(20,8))
548
549 plt.subplot(411)
550 plt.plot(data.Weighted_Price, label='Original')
551 plt.legend(loc='best')
552 plt.subplot(412)
553 plt.plot(trend, label='Trend')
554 plt.legend(loc='best')
555 plt.subplot(413)
556 plt.plot(seasonal, label='Seasonality')
557 plt.legend(loc='best')
558 plt.subplot(414)
559 plt.plot(residual, label='Residuals')
560 plt.legend(loc='best')
561
562 fig.suptitle('Decomposition of Prices Data')
563 plt.show()
564
565 print("Dickey-Fuller test: p=%f" %
    sm.tsa.stattools.adfuller(data.Weighted_Price)[1])
566
567 from statsmodels.graphics.tsaplots import plot_acf
```

```

568 from statsmodels.graphics.tsaplots import plot_pacf
569 from matplotlib import pyplot
570 pyplot.figure(figsize=(20,8))
571 pyplot.subplot(211)
572 plot_acf(data.Weighted_Price, ax=pyplot.gca(),lags=40)
573 pyplot.subplot(212)
574 plot_pacf(data.Weighted_Price, ax=pyplot.gca(), lags=50)
575 pyplot.show()
576
577 # Initial approximation of parameters
578 Qs = range(0, 2)
579 qs = range(0, 3)
580 Ps = range(0, 3)

```

```

1 ps = range(0, 3)
2 D=1
3 d=1
4 parameters = product(ps, qs, Ps, Qs)
5 parameters_list = list(parameters)
6 len(parameters_list)
7
8 # Model Selection
9 results = []
10 best_aic = float("inf")
11 warnings.filterwarnings('ignore')
12 for param in parameters_list:
13     try:
14
15         model=sm.tsa.statespace.SARIMAX(data.Weighted_Price,
16         order=(param[0], d, param[1]),
17         seasonal_order=(param[2], D, param[3],
18         12),enforce_stationarity=False,
19         enforce_invertibility=False).fit(dis=-1)

```



```

17     except ValueError:
18         #print('wrong parameters:', param)
19         continue
20     aic = model.aic
21     if aic < best_aic:
22         best_model = model
23         best_aic = aic
24         best_param = param
25     results.append([param, model.aic])
26
27 # Best Models
28 result_table = pd.DataFrame(results)
29 result_table.columns = ['parameters', 'aic']
30 print(result_table.sort_values(by = 'aic',
    ascending=True).head())
31 print(best_model.summary())
32
33 fig = plt.figure(figsize=(20,8))
34 best_model.resid.plot()
35 fig.suptitle('Residual Plot of the Best Model')
36 print("Dickey-Fuller test:: p=%f" %
    sm.tsa.stattools.adfuller(best_model.resid)[1])
37
38 df_month2 = data[['Weighted_Price']]
39 future = pd.DataFrame()
40 df_month2 = pd.concat([df_month2, future])
41 df_month2['forecast'] = best_model.predict(start=0,
    end=200)
42 plt.figure(figsize=(15,7))
43 df_month2.Weighted_Price.plot()
44 df_month2.forecast.plot(color='r', ls='--',
    label='Predicted Weighted_Price')
45 plt.legend()
46 plt.title('Bitcoin Prices (USD) Predicted vs Actuals, by
    months')
47 plt.ylabel('mean USD')

```

```

48 plt.show()
49
50 ## Prediction using rNN
51
52 df =
    data=pd.read_csv(r'C:\Users\rajat\Downloads\bitcoindata.csv
    ')
53 df['date'] =
    pd.to_datetime(df['Timestamp'],unit='s').dt.date
54 group = df.groupby('date')
55 Real_Price = group['Weighted_Price'].mean()
56
57 # split data
58 prediction_days = 30
59 df_train= Real_Price[:len(Real_Price)-prediction_days]
60 df_test= Real_Price[len(Real_Price)-prediction_days:]
61
62 # Data preprocess
63 training_set = df_train.values
64 training_set = np.reshape(training_set, (len(training_set),
    1))
65 from sklearn.preprocessing import MinMaxScaler
66 sc = MinMaxScaler()
67 training_set = sc.fit_transform(training_set)
68 X_train = training_set[0:len(training_set)-1]
69 y_train = training_set[1:len(training_set)]
70 X_train = np.reshape(X_train, (len(X_train), 1, 1))
71
72 # Importing the Keras libraries and packages
73 from keras.models import Sequential
74 from keras.layers import Dense
75 from keras.layers import LSTM
76
77 # Initialising the RNN
78 regressor = Sequential()
79

```

```
80 # Adding the input layer and the LSTM layer
81 regressor.add(LSTM(units = 4, activation = 'sigmoid',
    input_shape = (None, 1)))
82
83 # Adding the output layer
84 regressor.add(Dense(units = 1))
85
86 # Compiling the RNN
87 regressor.compile(optimizer = 'adam', loss =
    'mean_squared_error')
88
89 # Fitting the RNN to the Training set
90 regressor.fit(X_train, y_train, batch_size = 5, epochs =
    100)
91
92 model.summary()
93
94 # Making the predictions
95 test_set = df_test.values
96 inputs = np.reshape(test_set, (len(test_set), 1))
97 inputs = sc.transform(inputs)
98 inputs = np.reshape(inputs, (len(inputs), 1, 1))
99 predicted_BTC_price = regressor.predict(inputs)
100 predicted_BTC_price =
    sc.inverse_transform(predicted_BTC_price)
101
102 # Visualising the results
103 plt.figure(figsize=(25,15), dpi=80, facecolor='w',
    edgecolor='k')
104 ax = plt.gca()
105 plt.plot(test_set, color = 'red', label = 'Real BTC
    Price')
106 plt.plot(predicted_BTC_price, color = 'blue', label =
    'Predicted BTC Price')
107 plt.title('BTC Price Prediction', fontsize=40)
108 df_test = df_test.reset_index()
```

```
109 x=df_test.index
110 labels = df_test['date']
111 plt.xticks(x, labels, rotation = 'vertical')
112 for tick in ax.xaxis.get_major_ticks():
113     tick.label1.set_fontsize(18)
114 for tick in ax.yaxis.get_major_ticks():
115     tick.label1.set_fontsize(18)
116 plt.xlabel('Time', fontsize=40)
117 plt.ylabel('BTC Price(USD)', fontsize=40)
118 plt.legend(loc=2, prop={'size': 25})
119 plt.show()
120
121
```