# 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"

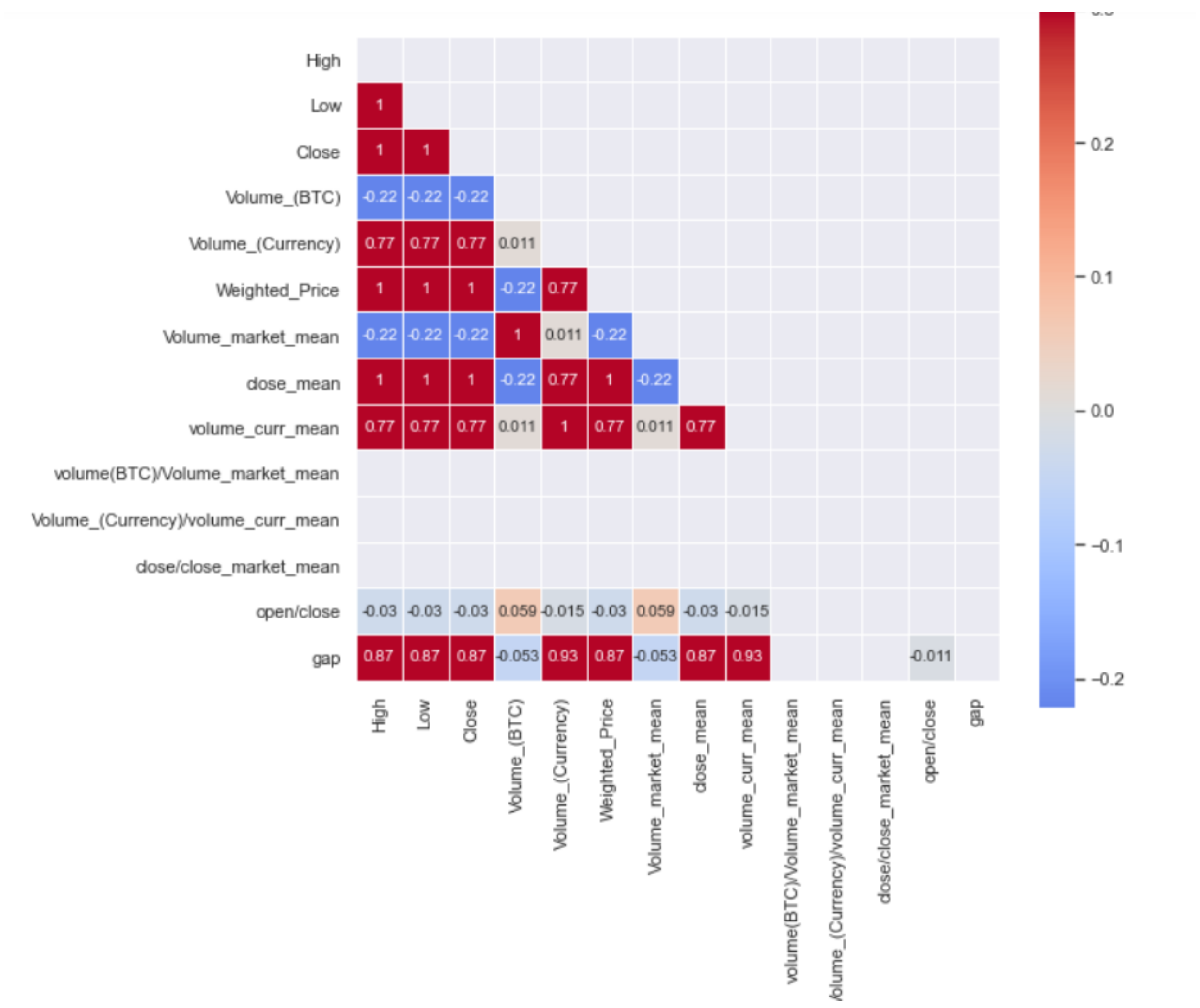
Dataset Description:-

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 :-

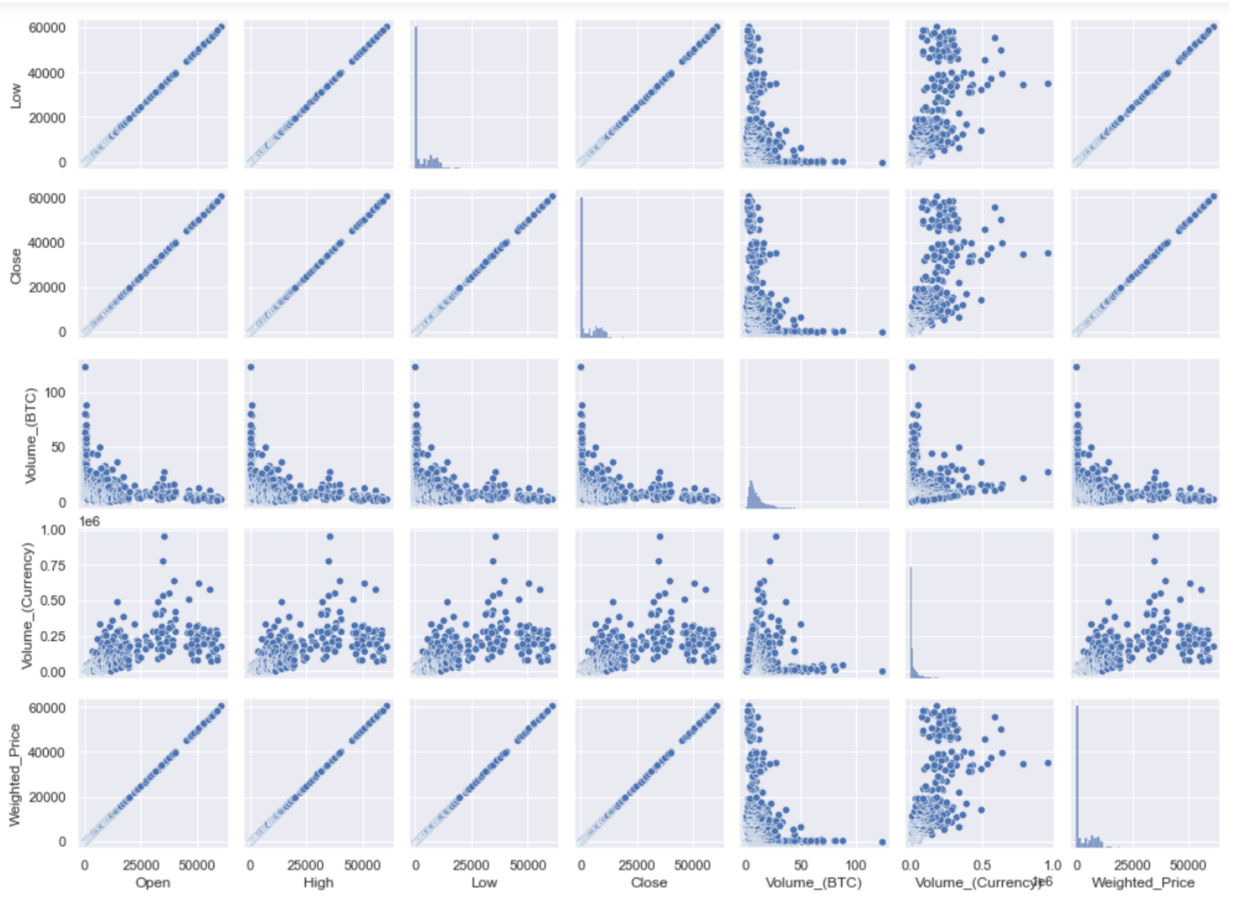
Opening price, Closing price, Highest Price, Lowest Price, Volume_market_mean

Correlation Metrics:-



|  | High | Low | Close | Volume_(BTC) | Volume_(Currency) | Weighted_Price | Volume_market_mean | close_mean | volume_curr_mean | volume(BTC)/Volume_market_mean | Volume_(Currency)/volume_curr_mean | close/close_market_mean | open/close | gap |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| High |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Low | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Close | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |
| Volume_(BTC) | -0.22 | -0.22 | -0.22 |  |  |  |  |  |  |  |  |  |  |  |
| Volume_(Currency) | 0.77 | 0.77 | 0.77 | 0.011 |  |  |  |  |  |  |  |  |  |  |
| Weighted_Price | 1 | 1 | 1 | -0.22 | 0.77 |  |  |  |  |  |  |  |  |  |
| Volume_market_mean | -0.22 | -0.22 | -0.22 | 1 | 0.011 | -0.22 |  |  |  |  |  |  |  |  |
| close_mean | 1 | 1 | 1 | -0.22 | 0.77 | 1 | -0.22 |  |  |  |  |  |  |  |
| volume_curr_mean | 0.77 | 0.77 | 0.77 | 0.011 | 1 | 0.77 | 0.011 | 0.77 |  |  |  |  |  |  |
| volume(BTC)/Volume_market_mean |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Volume_(Currency)/volume_curr_mean |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| close/close_market_mean |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| open/close | -0.03 | -0.03 | -0.03 | 0.059 | -0.015 | -0.03 | 0.059 | -0.03 | -0.015 |  |  |  |  |  |
| gap | 0.87 | 0.87 | 0.87 | -0.053 | 0.93 | 0.87 | -0.053 | 0.87 | 0.93 |  |  |  | -0.011 |  |

Regression Plots:-



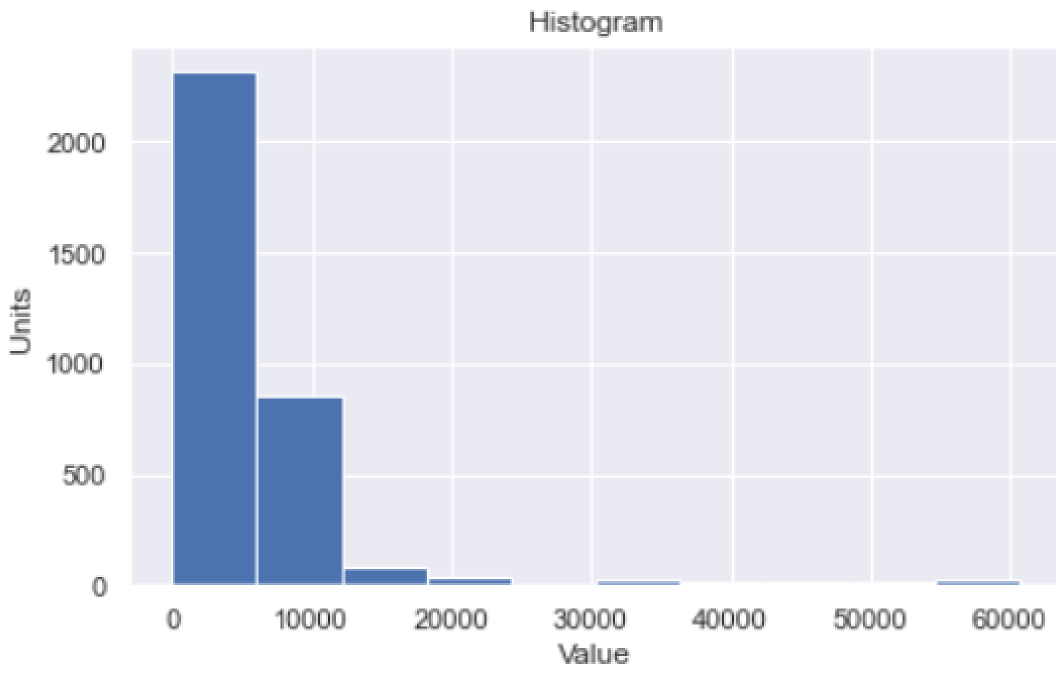
## Checking for Multicollinearity

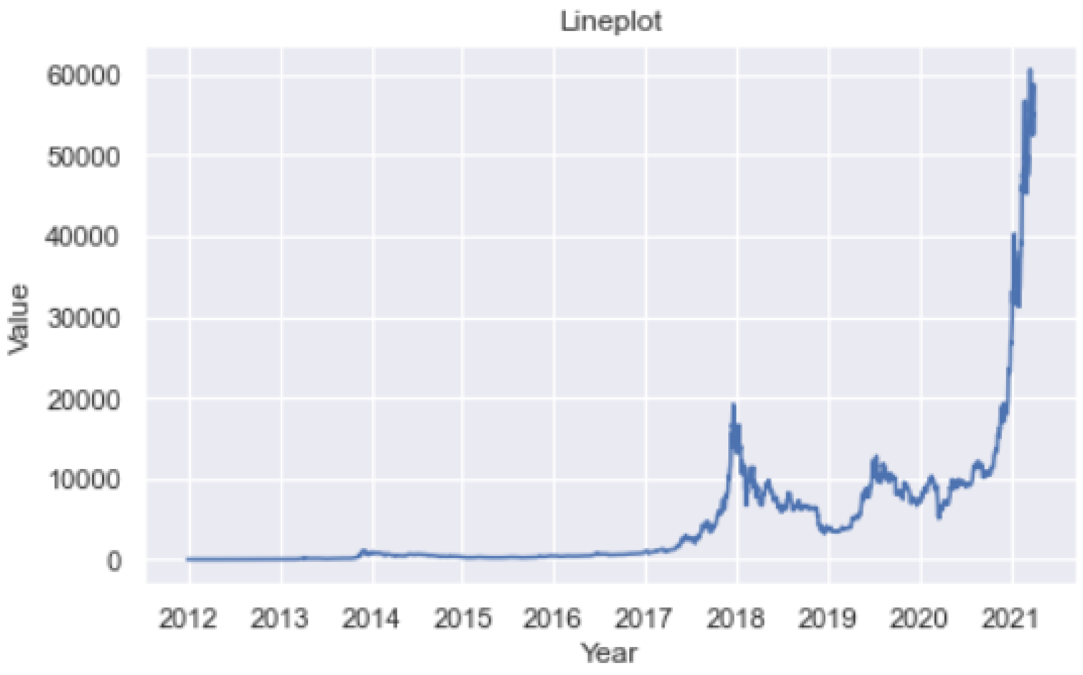
## Feature Selection:-

The following columns were selected -
Open, High, Low, Close, Volume_(BTC), Volume_(Currency), Weighted_Price

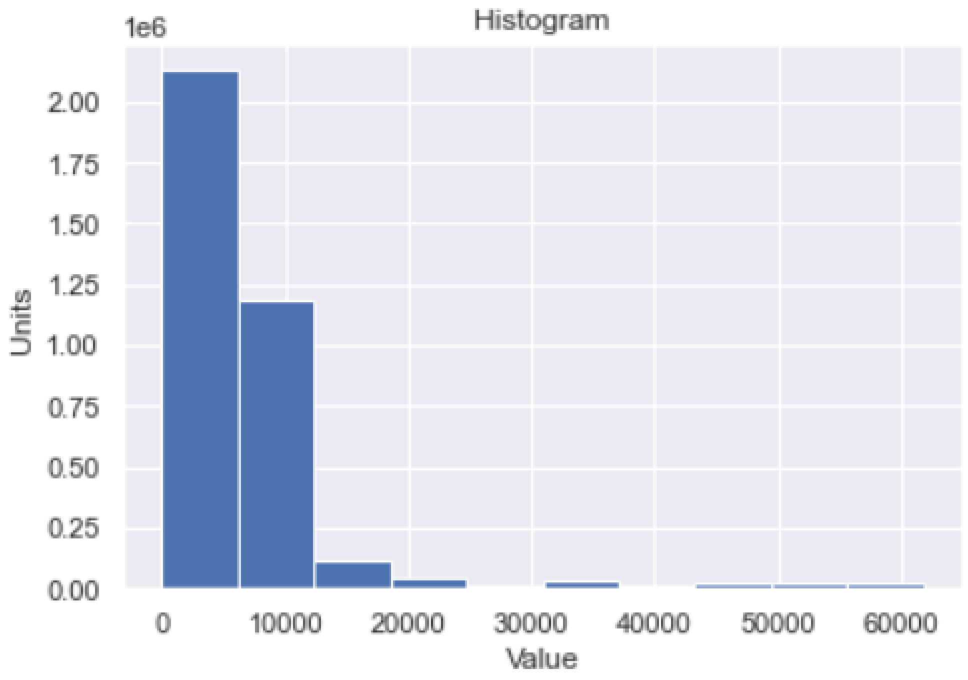
## Data Visualization-Other Plots:-

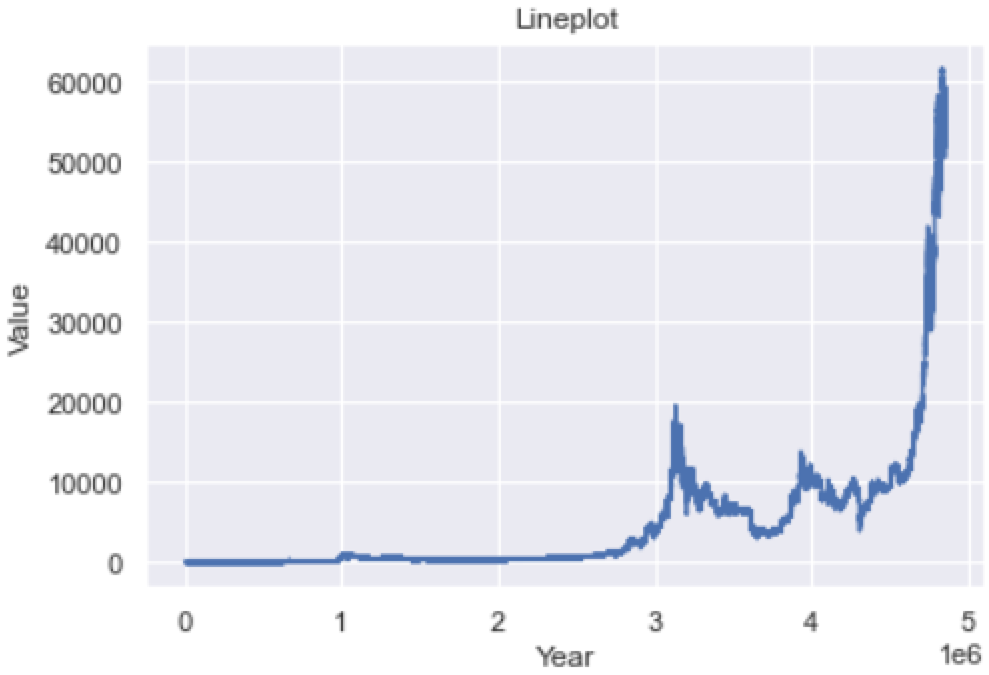
DISTRIBUTION AT OPENING PRICE



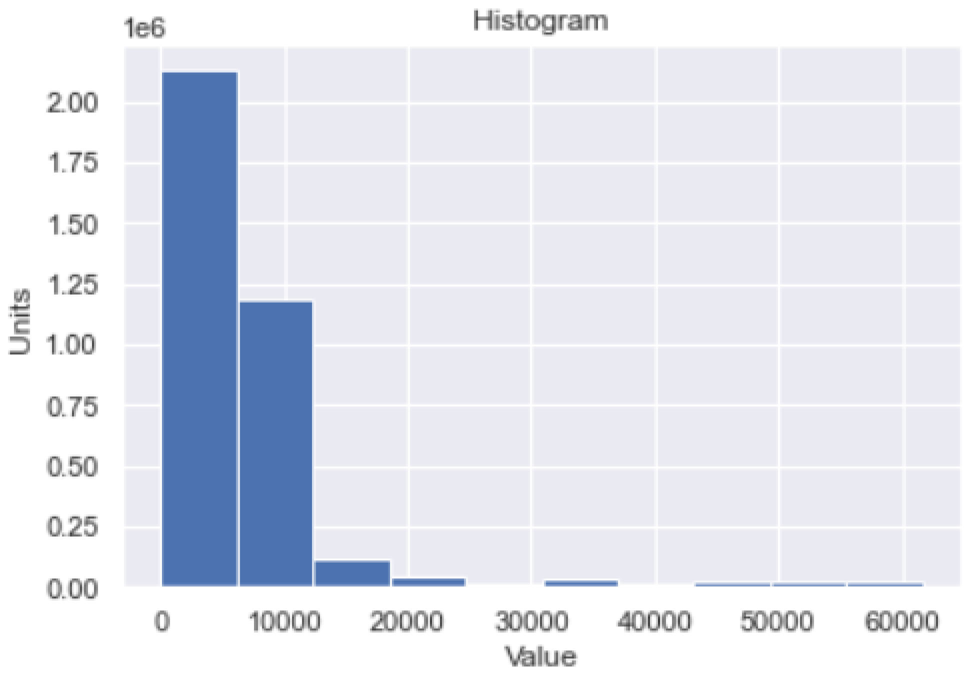


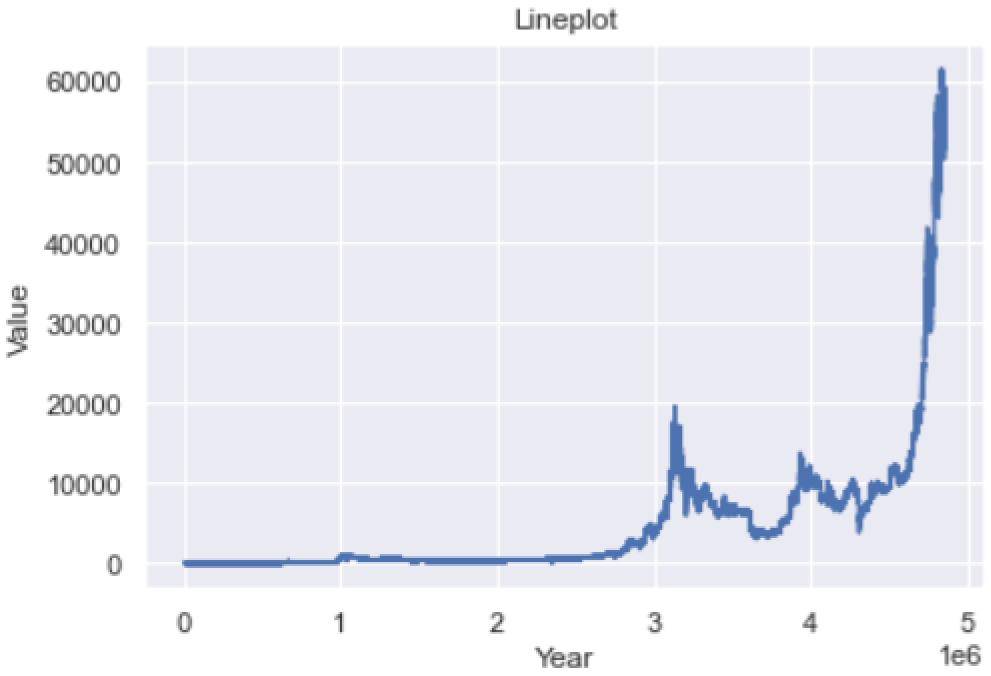
DISTRIBUTION OF HIGHEST PRICE



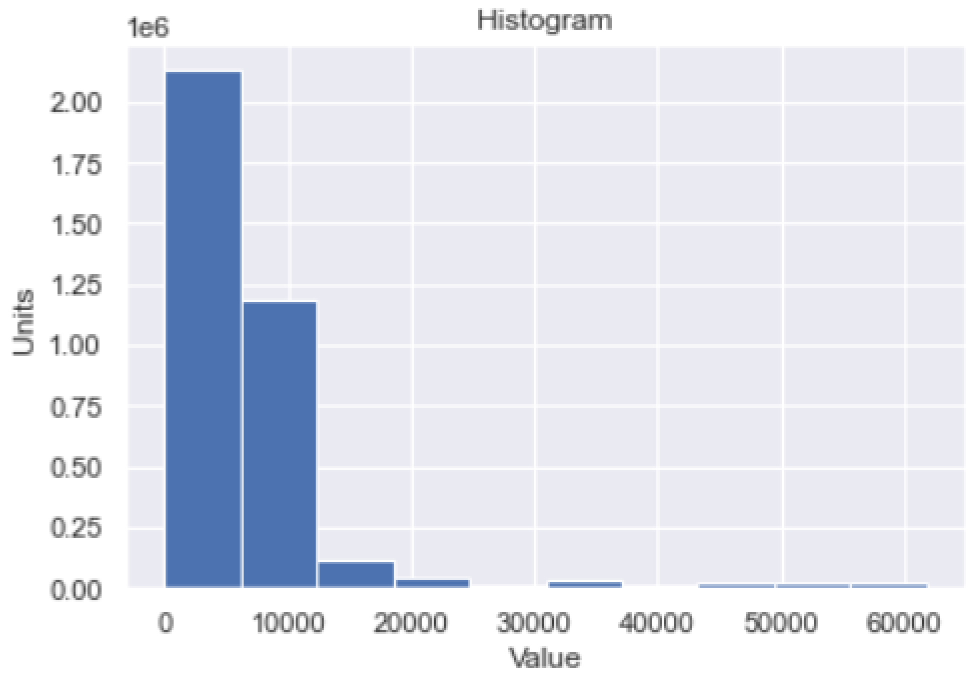


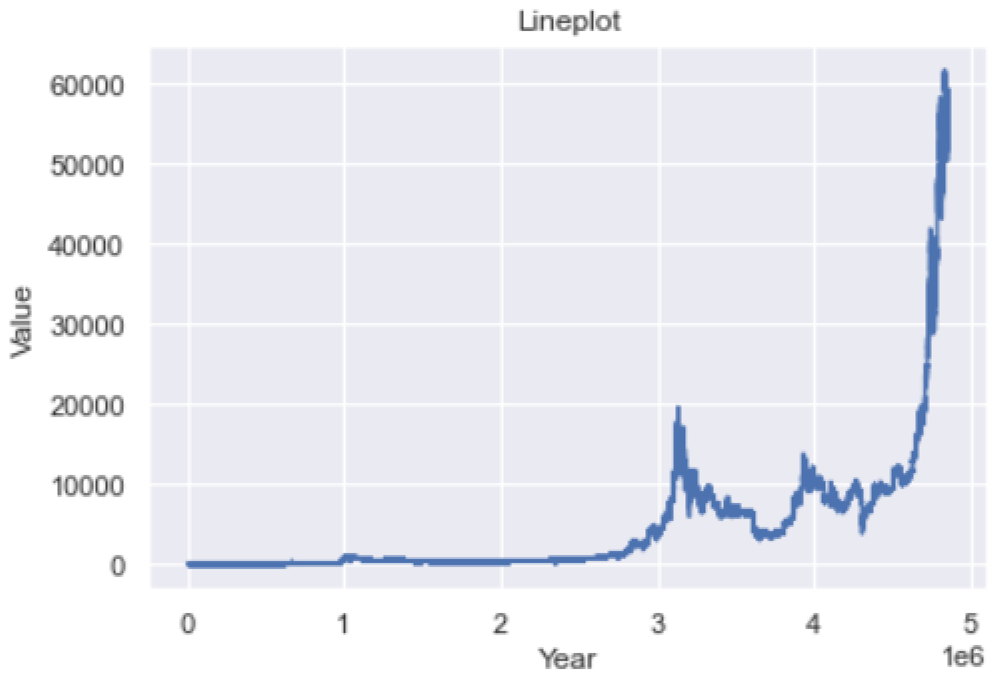
DISTRIBUTION OF LOWEST PRICE





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(disp=-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())
```

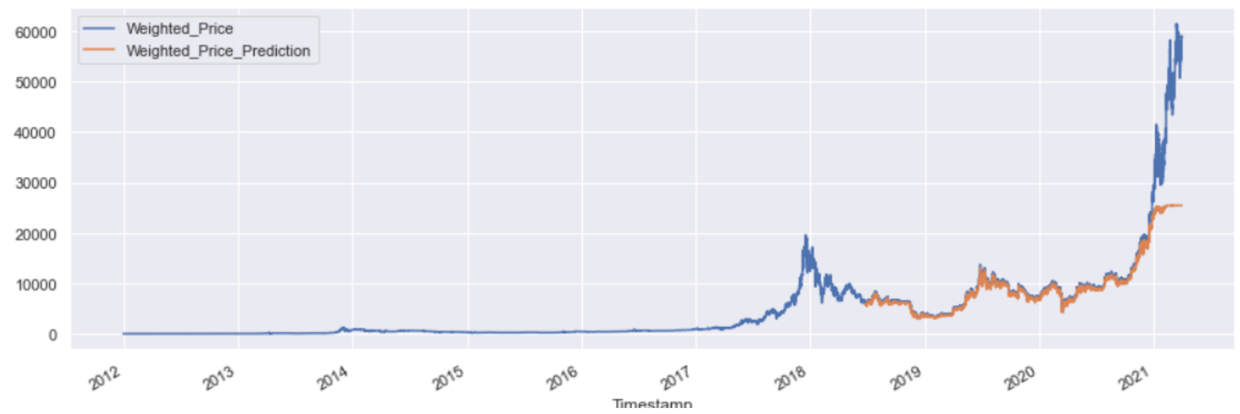
RNN-

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

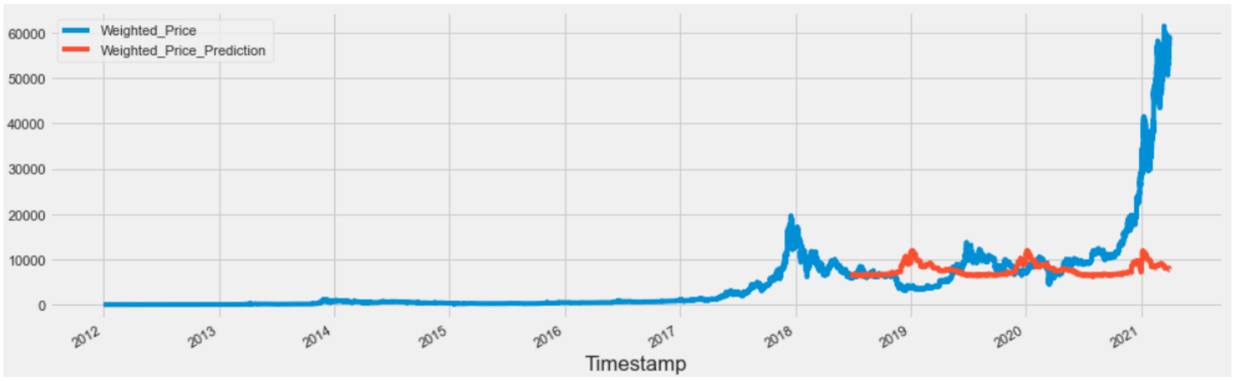
```
regressor = Sequential()


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:-

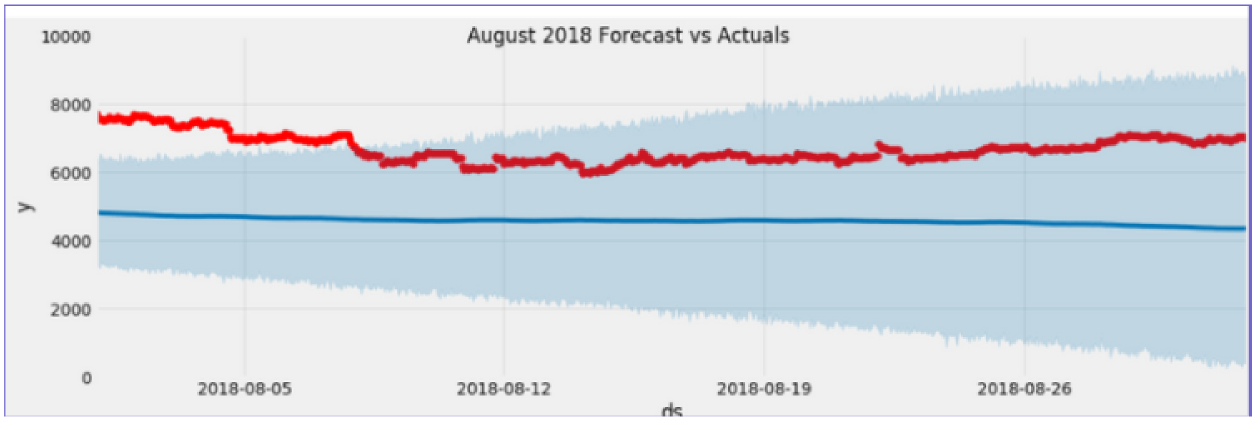
LSTM



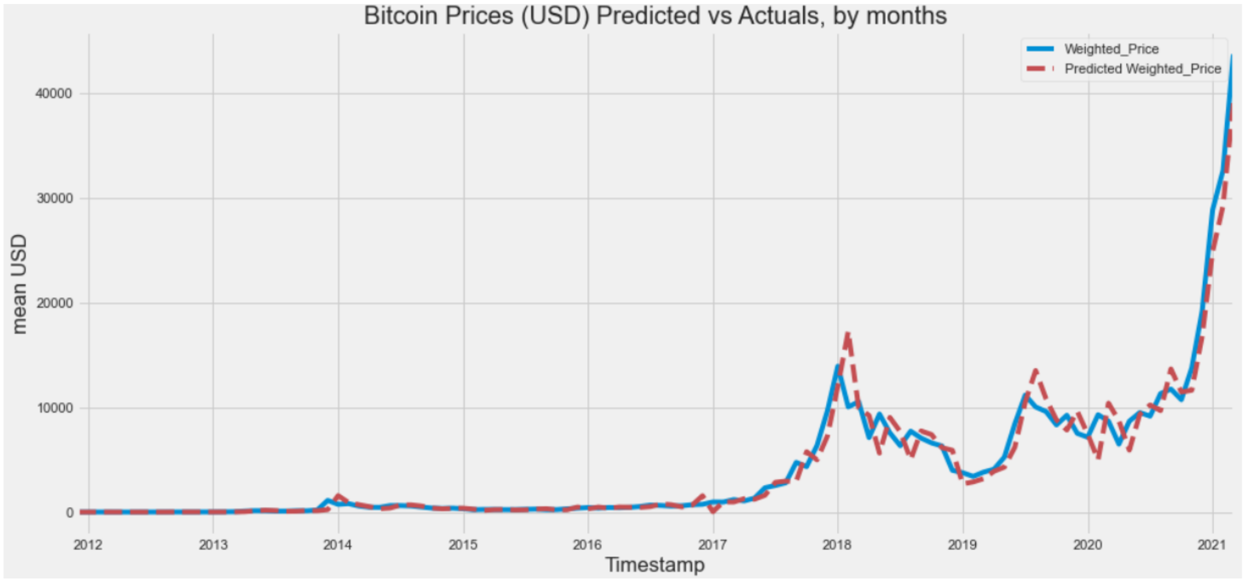
XGBoost



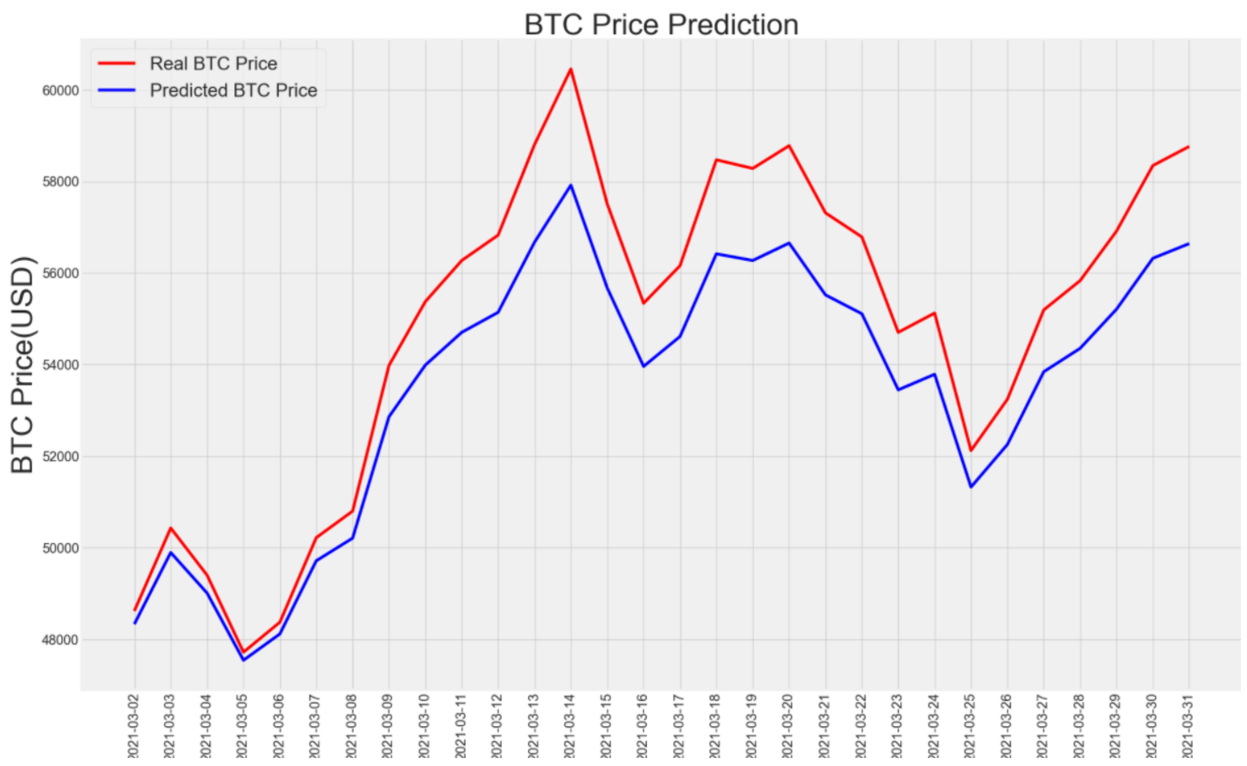
Prophet


August 2018 Forecast vs Actuals
y
10000
8000
6000
4000
2000
0
2018-08-05
2018-08-12
2018-08-19
2018-08-26
ds

ARIMA


Bitcoin Prices (USD) Predicted vs Actuals, by months
Weighted_Price
Predicted Weighted_Price
mean USD
40000
30000
20000
10000
0
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
Timestamp

RNN



FINAL CODE:-

```
 Final Code:-<div>
    <h1><center style="background-color:#87CEFA;
color:white; font-size: 80px;"> Bitcoin
        EDA and Prediction</center></h1>
</div>


<div>
<img
src="https://thumbs.gfycat.com/IllSharpCod-max-1mb.gif"
width='350'>
</div>

<a id="top"></a>

<div class="list-group" id="list-tab" role="tablist">
```

```
<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>

* [1. Required Libraries](#1)
* [2. Dataset Quick Overview & Pre-Processing](#2)
* [3. Features generation](#3)
* [4. Distribution of Features](#4)
* [5. Correlation Analysis ](#5)




<div class="alert alert-info">
<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>
<h4>Some interesting facts about Bitcoin(BTC):</h4>
<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>
<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>
<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>
</div>

<a id="1"></a>
<h2 style='background-color:#87CEFA; border:0;
```

```
color:black'><center>Required Libraries</center><h2>


#Data Pre-Processing packages:
import numpy as np
import pandas as pd
from datetime import datetime


#Data Visualization Packages:
#Seaborn
import seaborn as sns
sns.set(rc={'figure.figsize':(10,6)})
custom_colors = ["#4e89ae", "#c56183","#ed6663","#ffa372"]

#Matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib.image as mpimg

#Colorama
from colorama import Fore, Back, Style # For text colors
y_= Fore.CYAN
m_= Fore.WHITE

#garbage collector - To free up unused space
import gc
gc.collect()


#Time -To find how long each cell takes to run
import time
import datetime

#Importing of Data
data=pd.read_csv(r'C:\Users\rajat\Downloads\bitcoindata.csv
')
```

```

import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)



<a id="2"></a>
<h2 style='background-color:#87CEFA; border:0; color:black'><center>Data set Overview & Pre-Processing</center><h2>

print(f"{m_}Total records:{y_}{data.shape}\n")
print(f"{m_}Data types of data columns: \n{y_}{data.dtypes}")

<div class="alert alert-info">
    <h3 color:black><center><b> Data Pre-processing steps</b></center></h3>
    <p> 1. <b>Date</b> - We need to convert the Hourly data to Daily based data </p>
    <p> 2. <b>Fill in the missing values interpolation</p>

</div>

<div class='alert alert-info'>
<h3> The data is available on a Hourly based on each day, So we need to resample them to day based.</h3>
</div>

data['Timestamp'] = [datetime.datetime.fromtimestamp(x) for x in data['Timestamp']]
data = data.set_index('Timestamp')
data = data.resample("24H").mean()
data.head()

```

```
<div class="alert alert-info">
    <h3><b><center>Missing values</center><b></h3>
</div>


missed = pd.DataFrame()
missed['column'] = data.columns

missed['percent'] = [round(100* data[col].isnull().sum() / len(data), 2) for col in data.columns]
missed = missed.sort_values('percent',ascending=False)
missed = missed[missed['percent']>0]

fig = sns.barplot(
    x=missed['percent'],
    y=missed["column"],
    orientation='horizontal',palette="winter"
).set_title('Missed values percent for every column')

data.isnull().sum()

<div class="alert alert-info">
    <h3><b><center>Let's interpolate to fill in the missing values</center><b></h3>
</div>


def fill_missing(df):
    ### function to impute missing values using interpolation ###
    df['Open'] = df['Open'].interpolate()
    df['Close'] = df['Close'].interpolate()
    df['Weighted_Price'] = df['Weighted_Price'].interpolate()
    df['Volume_(BTC)'] = df['Volume_(BTC)'].interpolate()
    df['Volume_(Currency)'] =
df['Volume_(Currency)'].interpolate()
    df['High'] = df['High'].interpolate()
    df['Low'] = df['Low'].interpolate()
    print(f'{m_}No. of Missing values after interpolation:\n{y_}{df.isnull().sum()}')
fill_missing(data)

<a id="3"></a>
<h2 style='background-color:#87CEFA; border:0; color:black'><center>Feature Generation</center><h2>

data.columns

new_df=data.groupby('Timestamp').mean()
new_df=new_df[['Volume_(BTC)', 'Close','Volume_(Currency)']]
new_df.rename(columns={'Volume_(BTC)':'Volume_market_mean','Close':'close_mean','Volume_(Currency)':'volume_curr_mean'},inplace=True)
new_df.head()

data_df = data.merge(new_df, left_on='Timestamp',
                                 right_index=True)
data_df['volume(BTC)/Volume_market_mean'] = data_df['Volume_(BTC)'] / data_df['Volume_market_mean']
data_df['Volume_(Currency)/volume_curr_mean'] = data_df['Volume_(Currency)'] / data_df['volume_curr_mean']

data_df['close/close_market_mean'] = data_df['Close'] / data_df['close_mean']
data_df['open/close'] = data_df['Open'] / data_df['Close']
data_df["gap"] = data_df["High"] - data_df["Low"]
data_df.head()

<div class='alert alert-info'>
     <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>
     <p> 1. use gc.collect() - collects all the garbage values </p>
     <p> 2. del dataframe - free up some space by deleting the unused dataframe using the del command </p>
     <p> 3. Reduce the memory usage based on the data types of the columns in the dataframe(shown below)</p>
</div>

def mem_usage(pandas_obj):
    if isinstance(pandas_obj,pd.DataFrame):
        usage_b = pandas_obj.memory_usage(deep=True).sum()
    else: # we assume if not a df it's a series
        usage_b = pandas_obj.memory_usage(deep=True)
    usage_mb = usage_b / 1024 ** 2 # convert bytes to megabytes
    return "{:03.2f} MB".format(usage_mb)
print(f'{m_}Memory of the dataframe:\n{y_}{mem_usage(data_df)}')

#All the columns in float64 format, we can downsize them to float32 to reduce memory usage
data_df.info()


<div class='alert alert-info'>
<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>
</div>

gl_float = data_df.select_dtypes(include=['float'])
converted_float = gl_float.apply(pd.to_numeric,downcast='float')
compare_floats =
pd.concat([gl_float.dtypes,converted_float.dtypes],axis=1)
compare_floats.columns = ['Before','After']
compare_floats.apply(pd.Series.value_counts)

print(f"{m_}Before float conversion:\n{y_}{mem_usage(data_df)}")
data_df[converted_float.columns] = converted_float
print(f"{m_}After float conversion:\n{y_}{mem_usage(data_df)}")

<div class='alert alert-info'>
    <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 aprroach to reduce the memory usage</h3>
</div>

<a id="4"></a>
<h2 style='background-color:#87CEFA; border:0; color:black'><center>Distribution of Features</center><h2>

<div class="alert alert-info">
    <h3><b>Let's Visualize the distribution of the key variables like Opening price, Highest price, Lowest price and Volume in Bitcoin<b></h3>
</div>



print("                                 DISTRIBUTION AT OPENING PRICE")

plt.hist(data['Open'])
plt.title("Histogram")
plt.xlabel("Value")
plt.ylabel("Units")
plt.show()
```

```python

plt.plot(data['Open'])
plt.title("Lineplot")
plt.xlabel("Year")
plt.ylabel("Value")
plt.show()



print("                                          DISTRIBUTION OF HIGHEST PRICE")

plt.hist(data['High'])
plt.title("Histogram")
plt.xlabel("Value")
plt.ylabel("Units")
plt.show()

plt.plot(data['High'])
plt.title("Lineplot")
plt.xlabel("Year")
plt.ylabel("Value")
plt.show()




print("                                          DISTRIBUTION OF LOWEST PRICE")

plt.hist(data['Low'])
plt.title("Histogram")
plt.xlabel("Value")
plt.ylabel("Units")
plt.show()

plt.plot(data['Low'])
plt.title("Lineplot")
plt.xlabel("Year")
plt.ylabel("Value")
plt.show()


print("                                 DISTRIBUTION OF CLOSING PRICE")

plt.hist(data['Close'])
plt.title("Histogram")
plt.xlabel("Value")
plt.ylabel("Units")
plt.show()

plt.plot(data['Close'])
plt.title("Lineplot")
plt.xlabel("Year")
plt.ylabel("Value")
plt.show()

<a id="5"></a>
<h2 style='background-color:#87CEFA; border:0; color:black'><center>Correlation Analysis</center><h2>

plt.figure(figsize=(8,8))
corr=data_df[data_df.columns[1:]].corr()
mask = np.triu(np.ones_like(corr, dtype=bool))
sns.heatmap(data_df[data_df.columns[1:]].corr(), mask=mask, cmap='coolwarm', vmax=.3, center=0,
            square=True, linewidths=.5,annot=True)
plt.show()

sns.pairplot(data,height=2)

```

```
<a id="5"></a>
<h2 style='background-color:#87CEFA; border:0; color:black'><center>Prediction</center><h2>

## Predictiong using LSTM

import datetime, pytz
#define a conversion function for the native timestamps in the csv file
def dateparse (time_in_secs):
    return pytz.utc.localize(datetime.datetime.fromtimestamp(float(time_in_secs)))



#data=pd.read_csv(r'C:\Users\rajat\Downloads\bitcoindata.csv')
data = pd.read_csv(r'C:\Users\rajat\Downloads\bitcoindata.csv',parse_dates=[0], date_parser=dateparse)
data['Timestamp'] = data['Timestamp'].dt.tz_localize(None)
data = data.groupby([pd.Grouper(key='Timestamp', freq='H')]).first().reset_index()
data = data.set_index('Timestamp')
data = data[['Weighted_Price']]
data['Weighted_Price'].fillna(method='ffill', inplace=True)

# split data
split_date = '25-Jun-2018'
data_train = data.loc[data.index <= split_date].copy()
data_test = data.loc[data.index > split_date].copy()


# Data preprocess
```

```python
training_set = data_train.values
training_set = np.reshape(training_set, (len(training_set), 1))
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler()
training_set = sc.fit_transform(training_set)
X_train = training_set[0:len(training_set)-1]
y_train = training_set[1:len(training_set)]
X_train = np.reshape(X_train, (len(X_train), 1, 1))

color_pal = ["#F8766D", "#D39200", "#93AA00", "#00BA38", "#00C19F", "#00B9E3", "#619CFF", "#DB72FB"]
_ = data.plot(style='', figsize=(15,5), color=color_pal[0], title='BTC Weighted_Price Price (USD) by Hours')


_ = data_test \
    .rename(columns={'Weighted_Price': 'Test Set'}) \
    .join(data_train.rename(columns={'Weighted_Price': 'Training Set'}), how='outer') \
    .plot(figsize=(15,5), title='BTC Weighted_Price Price (USD) by Hours', style='')

# Importing the Keras libraries and packages
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)

model.summary()

# Making the predictions
test_set = data_test.values
inputs = np.reshape(test_set, (len(test_set), 1))
inputs = sc.transform(inputs)
inputs = np.reshape(inputs, (len(inputs), 1, 1))
predicted_BTC_price = model.predict(inputs)
predicted_BTC_price = sc.inverse_transform(predicted_BTC_price)


data_test['Weighted_Price_Prediction'] = predicted_BTC_price
data_all = pd.concat([data_test, data_train], sort=False)


_ = data_all[['Weighted_Price','Weighted_Price_Prediction']].plot(figsize=(15, 5))

f, ax = plt.subplots(1)
f.set_figheight(5)
f.set_figwidth(15)
_ = data_all[['Weighted_Price_Prediction','Weighted_Price']].plot(ax=ax,
style=['-','.'])
ax.set_xbound(lower='08-01-2018', upper='09-01-2018')
ax.set_ylim(0, 10000)
```

```
plot = plt.suptitle('August 2018 Forecast vs Actuals')

# Plot the forecast with the actuals
f, ax = plt.subplots(1)
f.set_figheight(5)
f.set_figwidth(15)
_ =
  data_all[['Weighted_Price_Prediction','Weighted_Price']].pl
  ot(ax=ax,

  style=['-','.'])
ax.set_xbound(lower='08-01-2018', upper='08-08-2018')
ax.set_ylim(0, 10000)
plot = plt.suptitle('First Week of August 2018 Forecast vs
  Actuals')

#calculate MSE and MAE
from sklearn.metrics import mean_squared_error,
  mean_absolute_error
mean_squared_error(y_true=data_test['Weighted_Price'],

  y_pred=data_test['Weighted_Price_Prediction'])

mean_absolute_error(y_true=data_test['Weighted_Price'],

  y_pred=data_test['Weighted_Price_Prediction'])

## Predictiong using XGBoost

import seaborn as sns
import matplotlib.pyplot as plt
#from fbprophet import Prophet
from sklearn.metrics import mean_squared_error,
  mean_absolute_error
plt.style.use('fivethirtyeight')

data =
```

```python
pd.read_csv(r'C:\Users\rajat\Downloads\bitcoindata.csv',parse_dates=[0], date_parser=dateparse)
data['Timestamp'] = data['Timestamp'].dt.tz_localize(None)
data = data.groupby([pd.Grouper(key='Timestamp', freq='H')]).first().reset_index()
data = data.set_index('Timestamp')
data = data[['Weighted_Price']]
data['Weighted_Price'].fillna(method='ffill', inplace=True)

color_pal = ["#F8766D", "#D39200", "#93AA00", "#00BA38", "#00C19F", "#00B9E3", "#619CFF", "#DB72FB"]
_ = data.plot(style='', figsize=(15,5), color=color_pal[0], title='BTC Weighted_Price Price (USD) by Hours')

split_date = '25-Jun-2018'
data_train = data.loc[data.index <= split_date].copy()
data_test = data.loc[data.index > split_date].copy()

_ = data_test \
    .rename(columns={'Weighted_Price': 'Test Set'}) \
    .join(data_train.rename(columns={'Weighted_Price': 'Training Set'}), how='outer') \
    .plot(figsize=(15,5), title='BTC Weighted_Price Price (USD) by Hours', style='')

def create_features(df, label=None):
    """
    Creates time series features from datetime index
    """
    df['date'] = df.index
    df['hour'] = df['date'].dt.hour
    df['dayofweek'] = df['date'].dt.dayofweek
    df['quarter'] = df['date'].dt.quarter
    df['month'] = df['date'].dt.month
    df['year'] = df['date'].dt.year
    df['dayofyear'] = df['date'].dt.dayofyear
    df['dayofmonth'] = df['date'].dt.day
    df['weekofyear'] = df['date'].dt.weekofyear
    X = df[['hour','dayofweek','quarter','month','year',
           'dayofyear','dayofmonth','weekofyear']]
    if label:
        y = df[label]
        return X, y
    return X

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)

data_test['Weighted_Price_Prediction'] = model.predict(X_test)
data_all = pd.concat([data_test, data_train], sort=False)

_ = data_all[['Weighted_Price','Weighted_Price_Prediction']].plot(figsize=(15, 5))

# Plot the forecast with the actuals
```

```python
f, ax = plt.subplots(1)
f.set_figheight(5)
f.set_figwidth(15)
_ = data_all[['Weighted_Price_Prediction','Weighted_Price']].plot(ax=ax,
    style=['-','.'])
ax.set_xbound(lower='08-01-2018', upper='09-01-2018')
ax.set_ylim(0, 10000)
plot = plt.suptitle('August 2018 Forecast vs Actuals')

# Plot the forecast with the actuals
f, ax = plt.subplots(1)
f.set_figheight(5)
f.set_figwidth(15)
_ = data_all[['Weighted_Price_Prediction','Weighted_Price']].plot(ax=ax,
    style=['-','.'])
ax.set_xbound(lower='08-01-2018', upper='08-08-2018')
ax.set_ylim(0, 10000)
plot = plt.suptitle('First Week of August 2018 Forecast vs Actuals')

mean_squared_error(y_true=data_test['Weighted_Price'],
    y_pred=data_test['Weighted_Price_Prediction'])

mean_absolute_error(y_true=data_test['Weighted_Price'],
    y_pred=data_test['Weighted_Price_Prediction'])

## Predictiong using Prophet

color_pal = ["#F8766D", "#D39200", "#93AA00", "#00BA38",
    "#00C19F", "#00B9E3", "#619CFF", "#DB72FB"]
_ = data.plot(style='', figsize=(15,5),
    color=color_pal[0], title='BTC Weighted_Price Price (USD)
    by Hours')

split_date = '25-Jun-2018'
data_train = data.loc[data.index <= split_date].copy()
data_test = data.loc[data.index > split_date].copy()


data_train =
    data_train.reset_index().rename(columns={'Timestamp':'ds',
    'Weighted_Price':'y'})

# Setup and train model
model = Prophet()
model.fit(data_train)

# Plot the forecast
f, ax = plt.subplots(1)
f.set_figheight(5)
f.set_figwidth(15)
fig = model.plot(data_test_fcst, ax=ax)

# Plot the components
fig = model.plot_components(data_test_fcst)

# Plot the forecast with the actuals
f, ax = plt.subplots(1)
f.set_figheight(5)
f.set_figwidth(15)
ax.scatter(data_test.index, data_test['Weighted_Price'],
    color='r')
fig = model.plot(data_test_fcst, ax=ax)
ax.set_xbound(lower='08-01-2018', upper='08-08-2018')
ax.set_ylim(0, 10000)
plot = plt.suptitle('First Week of August 2018 Forecast vs
```

```python
Actuals')

mean_squared_error(y_true=data_test['Weighted_Price'],
                   y_pred=data_test_fcst['yhat'])

mean_absolute_error(y_true=data_test['Weighted_Price'],
                    y_pred=data_test_fcst['yhat'])

## Prediction using ARIMA

from scipy import stats
import statsmodels.api as sm
import warnings
from itertools import product

data = pd.read_csv(r'C:\Users\rajat\Downloads\bitcoindata.csv',parse_dates=[0], date_parser=dateparse)

data['Open'].fillna(method='ffill', inplace=True)
data['High'].fillna(method='ffill', inplace=True)
data['Low'].fillna(method='ffill', inplace=True)
data['Close'].fillna(method='ffill', inplace=True)
data['Weighted_Price'].fillna(method='ffill', inplace=True)
data['Volume_(BTC)'].fillna(method='ffill', inplace=True)
data['Volume_(Currency)'].fillna(method='ffill', inplace=True)

plt.figure(figsize=[20,8])
plt.title('BTC Weighted_Price Price (USD) by Hours')
plt.plot(data.Weighted_Price, '-', label='By Hours')

data['Timestamp'] = data['Timestamp'].dt.tz_localize(None)
data = data.groupby([pd.Grouper(key='Timestamp', freq='M')]).first().reset_index()
data = data.set_index('Timestamp')
```

```python
data['Weighted_Price'].fillna(method='ffill', inplace=True)

plt.figure(figsize=[20,8])
plt.title('BTC Weighted_Price Price (USD) by Months')
plt.plot(data.Weighted_Price, '-', label='By Months')

decomposition = sm.tsa.seasonal_decompose(data.Weighted_Price)

trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

fig = plt.figure(figsize=(20,8))

plt.subplot(411)
plt.plot(data.Weighted_Price, label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(seasonal,label='Seasonality')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual, label='Residuals')
plt.legend(loc='best')

fig.suptitle('Decomposition of Prices Data')
plt.show()

print("Dickey-Fuller test: p=%f" % sm.tsa.stattools.adfuller(data.Weighted_Price)[1])

from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
from matplotlib import pyplot
pyplot.figure(figsize=(20,8))
pyplot.subplot(211)
plot_acf(data.Weighted_Price, ax=pyplot.gca(),lags=40)
pyplot.subplot(212)
plot_pacf(data.Weighted_Price, ax=pyplot.gca(), lags=50)
pyplot.show()

# Initial approximation of parameters
Qs = range(0, 2)
qs = range(0, 3)
Ps = range(0, 3)
```

```
ps = range(0, 3)
D=1
d=1
parameters = product(ps, qs, Ps, Qs)
parameters_list = list(parameters)
len(parameters_list)

# Model Selection
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(disp=-1)
```

```python
    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])

# Best Models
result_table = pd.DataFrame(results)
result_table.columns = ['parameters', 'aic']
print(result_table.sort_values(by = 'aic', ascending=True).head())
print(best_model.summary())

fig = plt.figure(figsize=(20,8))
best_model.resid.plot()
fig.suptitle('Residual Plot of the Best Model')
print("Dickey-Fuller test:: p=%f" % sm.tsa.stattools.adfuller(best_model.resid)[1])

df_month2 = data[['Weighted_Price']]
future = pd.DataFrame()
df_month2 = pd.concat([df_month2, future])
df_month2['forecast'] = best_model.predict(start=0, end=200)
plt.figure(figsize=(15,7))
df_month2.Weighted_Price.plot()
df_month2.forecast.plot(color='r', ls='--', label='Predicted Weighted_Price')
plt.legend()
plt.title('Bitcoin Prices (USD) Predicted vs Actuals, by months')
plt.ylabel('mean USD')
plt.show()

## Prediction using rNN

df = data=pd.read_csv(r'C:\Users\rajat\Downloads\bitcoindata.csv')
df['date'] = pd.to_datetime(df['Timestamp'],unit='s').dt.date
group = df.groupby('date')
Real_Price = group['Weighted_Price'].mean()

# split data
prediction_days = 30
df_train= Real_Price[:len(Real_Price)-prediction_days]
df_test= Real_Price[len(Real_Price)-prediction_days:]

# Data preprocess
training_set = df_train.values
training_set = np.reshape(training_set, (len(training_set), 1))
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler()
training_set = sc.fit_transform(training_set)
X_train = training_set[0:len(training_set)-1]
y_train = training_set[1:len(training_set)]
X_train = np.reshape(X_train, (len(X_train), 1, 1))

# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM

# Initialising the RNN
regressor = Sequential()

```

```python
# Adding the input layer and the LSTM layer
regressor.add(LSTM(units = 4, activation = 'sigmoid', input_shape = (None, 1)))

# Adding the output layer
regressor.add(Dense(units = 1))

# Compiling the RNN
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')

# Fitting the RNN to the Training set
regressor.fit(X_train, y_train, batch_size = 5, epochs = 100)

model.summary()

# Making the predictions
test_set = df_test.values
inputs = np.reshape(test_set, (len(test_set), 1))
inputs = sc.transform(inputs)
inputs = np.reshape(inputs, (len(inputs), 1, 1))
predicted_BTC_price = regressor.predict(inputs)
predicted_BTC_price = sc.inverse_transform(predicted_BTC_price)

# Visualising the results
plt.figure(figsize=(25,15), dpi=80, facecolor='w', edgecolor='k')
ax = plt.gca()
plt.plot(test_set, color = 'red', label = 'Real BTC Price')
plt.plot(predicted_BTC_price, color = 'blue', label = 'Predicted BTC Price')
plt.title('BTC Price Prediction', fontsize=40)
df_test = df_test.reset_index()
```

```python
x=df_test.index
labels = df_test['date']
plt.xticks(x, labels, rotation = 'vertical')
for tick in ax.xaxis.get_major_ticks():
    tick.label1.set_fontsize(18)
for tick in ax.yaxis.get_major_ticks():
    tick.label1.set_fontsize(18)
plt.xlabel('Time', fontsize=40)
plt.ylabel('BTC Price(USD)', fontsize=40)
plt.legend(loc=2, prop={'size': 25})
plt.show()


```