ADANI ENTERPRISES LIMITED

ATTRIBUTE INFORMATION

DATA SET : 'Quote-Equity-ADANIENT-EQ-18-03-2021-to-18-03-2023 (1).csv' IS OBTAINED FROM KAGGLE

https://www.nseindia.com/get-quotes/equity?symbol=ADANIENT (https://www.nseindia.com/get-quotes/equity?symbol=ADANIENT)

ABOUT DATASET

The dataset contains details of ADANI ENTERPRISES LIMITED stocks from 2021 to 2023.

The dataset you provided is related to the stock price of Adani Enterprises Limited (ADANIENT) traded on the National Stock Exchange (NSE) in India.

The data includes the following fields:

->Date: This column represents the date for which the stock market data is being presented.

->Series: This column represents the series of the stock. Stocks can be traded in different series such as equity shares, preference shares, or debentures, among others.

->OPEN: This column represents the opening price of the stock on the given date.

->HIGH: This column represents the highest price at which the stock traded during the day on the given date.

->LOW: This column represents the lowest price at which the stock traded during the day on the given date.

->PREV. CLOSE: This column represents the closing price of the stock on the previous trading day.

->LTP: This column represents the last traded price of the stock on the given date.

->CLOSE: This column represents the closing price of the stock on the given date.

->VWAP: This column represents the Volume Weighted Average Price of the stock on the given date.

->52W H: This column represents the highest price at which the stock traded in the past 52 weeks.

->52W L: This column represents the lowest price at which the stock traded in the past 52 weeks.

->VOLUME: This column represents the total number of shares traded on the given date.

->VALUE: This column represents the total value of shares traded on the given date.

Loading the dependencies

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler, RobustScaler
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor,RandomForestRegressor,GradientBoostingReg
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error,mean_squared
from sklearn.neighbors import KNeighborsRegressor,RadiusNeighborsRegressor,NearestCentro
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
```

Loading the dataset

In [2]:

```
df=pd.read_csv('./Quote-Equity-ADANIENT-EQ-18-03-2021-to-18-03-2023 (1).csv')
```

In [3]:

```
df
```

Out[3]:

	Date	series	OPEN	HIGH	LOW	PREV. CLOSE	ltp	close	vwap	52W H
0	17-Mar-2023	EQ	1,901.00	1,918.85	1,845.00	1,843.80	1,874.00	1,876.55	1,870.80	4,190.0
1	16-Mar-2023	EQ	1,861.00	1,875.00	1,795.00	1,839.00	1,840.00	1,843.80	1,838.73	4,190.0
2	15-Mar-2023	EQ	1,760.90	1,891.45	1,728.10	1,738.20	1,838.00	1,839.00	1,809.83	4,190.0
3	14-Mar-2023	EQ	1,874.00	1,874.85	1,651.35	1,874.40	1,730.00	1,738.20	1,742.93	4,190.0
4	13-Mar-2023	EQ	1,917.00	1,985.00	1,857.40	1,896.20	1,859.00	1,874.40	1,922.43	4,190.0
...	...	...	...	...	...	...	...	...	...	..
492	24-Mar-2021	EQ	1,063.00	1,093.00	1,018.40	1,058.40	1,020.35	1,025.45	1,060.20	1,093.0
493	23-Mar-2021	EQ	999.00	1,086.70	991.05	991.05	1,060.00	1,058.40	1,057.72	1,086.7
494	22-Mar-2021	EQ	892.90	1,003.00	883.45	889.65	992.50	991.05	965.79	1,003.0
495	19-Mar-2021	EQ	866.00	895.40	840.20	871.05	885.50	889.65	870.63	944.9
496	18-Mar-2021	EQ	876.00	891.00	857.75	873.90	874.80	871.05	877.32	944.9

497 rows × 14 columns

EXPLORATORY DATA ANALYSIS(EDA)

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 497 entries, 0 to 496
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Date           497 non-null    object
 1   series         497 non-null    object
 2   OPEN           497 non-null    object
 3   HIGH           497 non-null    object
 4   LOW            497 non-null    object
 5   PREV. CLOSE    497 non-null    object
 6   ltp            497 non-null    object
 7   close          497 non-null    object
 8   vwap           497 non-null    object
 9   52W H          497 non-null    object
 10  52W L          497 non-null    object
 11  VOLUME         497 non-null    int64 
 12  VALUE          497 non-null    object
 13  No of trades   497 non-null    int64 
dtypes: int64(2), object(12)
memory usage: 54.5+ KB
```

In this data set almost numerical colums having comas so the data type is showing as object we want to change the dtype object to int or float for ML

Cheking cloumn names

In [5]:

```
df.columns
```

Out[5]:

```
Index(['Date ', 'series ', 'OPEN ', 'HIGH ', 'LOW ', 'PREV. CLOSE ', 'ltp
',
       'close ', 'vwap ', '52W H ', '52W L ', 'VOLUME ', 'VALUE ',
       'No of trades '],
      dtype='object')
```

All Column Names containing extra space so we have to remove them

In [6]:

```
df.rename(columns={'Date ': 'Date'},inplace=True)
df.rename(columns={'series ':'series'},inplace=True)
df.rename(columns={'OPEN ':'OPEN'},inplace=True)
df.rename(columns={'HIGH ':'HIGH'},inplace=True)
df.rename(columns={'LOW ':'LOW'},inplace=True)
df.rename(columns={'PREV. CLOSE ':'PREV_CLOSE'},inplace=True)
df.rename(columns={'ltp ':'Last_Traded_Price'},inplace=True)
df.rename(columns={'close ':'close'},inplace=True)
df.rename(columns={'vwap ':'Volume_weighted_avg_price'},inplace=True)
df.rename(columns={'52W H ':'52W_H'},inplace=True)
df.rename(columns={'52W L ':'52W_L'},inplace=True)
df.rename(columns={'VOLUME ':'VOLUME'},inplace=True)
df.rename(columns={'VALUE ':'VALUE'},inplace=True)
df.rename(columns={'No of trades ':'No_of_trades'},inplace=True)
```

In [7]:

```
df
```

Out[7]:

	Date	series	OPEN	HIGH	LOW	PREV_CLOSE	Last_Traded_Price	close	V
0	17-Mar-2023	EQ	1,901.00	1,918.85	1,845.00	1,843.80	1,874.00	1,876.55	
1	16-Mar-2023	EQ	1,861.00	1,875.00	1,795.00	1,839.00	1,840.00	1,843.80	
2	15-Mar-2023	EQ	1,760.90	1,891.45	1,728.10	1,738.20	1,838.00	1,839.00	
3	14-Mar-2023	EQ	1,874.00	1,874.85	1,651.35	1,874.40	1,730.00	1,738.20	
4	13-Mar-2023	EQ	1,917.00	1,985.00	1,857.40	1,896.20	1,859.00	1,874.40	
...	...	...	...	...	...	...	...	...	
492	24-Mar-2021	EQ	1,063.00	1,093.00	1,018.40	1,058.40	1,020.35	1,025.45	
493	23-Mar-2021	EQ	999.00	1,086.70	991.05	991.05	1,060.00	1,058.40	
494	22-Mar-2021	EQ	892.90	1,003.00	883.45	889.65	992.50	991.05	
495	19-Mar-2021	EQ	866.00	895.40	840.20	871.05	885.50	889.65	
496	18-Mar-2021	EQ	876.00	891.00	857.75	873.90	874.80	871.05	

497 rows × 14 columns

In [8]:

```
df.columns
```

Out[8]:

```
Index(['Date', 'series', 'OPEN', 'HIGH', 'LOW', 'PREV_CLOSE',
       'Last_Traded_Price', 'close', 'Volume_weighted_avg_price', '52W_H',
       '52W_L', 'VOLUME', 'VALUE', 'No_of_trades'],
      dtype='object')
```

In [9]:

```
print(df['HIGH'].dtype)
```

```
object
```

In [10]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 497 entries, 0 to 496
Data columns (total 14 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Date                      497 non-null    object
 1   series                    497 non-null    object
 2   OPEN                      497 non-null    object
 3   HIGH                      497 non-null    object
 4   LOW                       497 non-null    object
 5   PREV_CLOSE                497 non-null    object
 6   Last_Traded_Price         497 non-null    object
 7   close                     497 non-null    object
 8   Volume_weighted_avg_price 497 non-null    object
 9   52W_H                     497 non-null    object
 10  52W_L                     497 non-null    object
 11  VOLUME                    497 non-null    int64 
 12  VALUE                     497 non-null    object
 13  No_of_trades              497 non-null    int64 
```

Cleaning The Dataset

In [11]:

```
df['OPEN'] = df['OPEN'].str.replace(',', '',)
df['HIGH'] = df['HIGH'].str.replace(',', '')
df['LOW'] = df['LOW'].str.replace(',', '')
df['PREV_CLOSE'] = df['PREV_CLOSE'].str.replace(',', '')
df['Last_Traded_Price'] = df['Last_Traded_Price'].str.replace(',', '')
df['close'] = df['close'].str.replace(',', '')
df['52W_H'] = df['52W_H'].str.replace(',', '')
df['Volume_weighted_avg_price'] = df['Volume_weighted_avg_price'].str.replace(',', '')
df['52W_L'] = df['52W_L'].str.replace(',', '')
```

In [12]:

```
df['OPEN'] = df['OPEN'].astype(float)
df['HIGH'] = df['HIGH'].astype(float)
df['LOW'] = df['LOW'].astype(float)
df['PREV_CLOSE'] = df['PREV_CLOSE'].astype(float)
df['Last_Traded_Price'] = df['Last_Traded_Price'].astype(float)
df['52W_H'] = df['52W_H'].astype(float)
df['Volume_weighted_avg_price'] = df['Volume_weighted_avg_price'].astype(float)
df['52W_L'] = df['52W_L'].astype(float)
df['close'] = df['close'].astype(float)
```

In [13]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 497 entries, 0 to 496
Data columns (total 14 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Date                       497 non-null    object 
 1   series                     497 non-null    object 
 2   OPEN                       497 non-null    float64
 3   HIGH                       497 non-null    float64
 4   LOW                        497 non-null    float64
 5   PREV_CLOSE                 497 non-null    float64
 6   Last_Traded_Price          497 non-null    float64
 7   close                      497 non-null    float64
 8   Volume_weighted_avg_price  497 non-null    float64
 9   52W_H                      497 non-null    float64
 10  52W_L                      497 non-null    float64
 11  VOLUME                     497 non-null    int64  
 12  VALUE                      497 non-null    object 
 13  No_of_trades               497 non-null    int64  
dtypes: float64(9), int64(2), object(3)
memory usage: 54.5+ KB
```

In [14]:

```
df.drop('series', axis=1, inplace=True)
```

In [15]:

```
df
```

Out[15]:

	Date	OPEN	HIGH	LOW	PREV_CLOSE	Last_Traded_Price	close	Volume_weig
0	17-Mar-2023	1901.0	1918.85	1845.00	1843.80	1874.00	1876.55	
1	16-Mar-2023	1861.0	1875.00	1795.00	1839.00	1840.00	1843.80	
2	15-Mar-2023	1760.9	1891.45	1728.10	1738.20	1838.00	1839.00	
3	14-Mar-2023	1874.0	1874.85	1651.35	1874.40	1730.00	1738.20	
4	13-Mar-2023	1917.0	1985.00	1857.40	1896.20	1859.00	1874.40	
...	...	...	...	...	...	...	...	
492	24-Mar-2021	1063.0	1093.00	1018.40	1058.40	1020.35	1025.45	
493	23-Mar-2021	999.0	1086.70	991.05	991.05	1060.00	1058.40	
494	22-Mar-2021	892.9	1003.00	883.45	889.65	992.50	991.05	
495	19-Mar-2021	866.0	895.40	840.20	871.05	885.50	889.65	
496	18-Mar-2021	876.0	891.00	857.75	873.90	874.80	871.05	

497 rows × 13 columns

In [16]:

```
df.isnull().sum()
```

Out[16]:

```
Date                       0
OPEN                       0
HIGH                       0
LOW                        0
PREV_CLOSE                 0
Last_Traded_Price          0
close                      0
Volume_weighted_avg_price  0
52W_H                      0
52W_L                      0
VOLUME                     0
VALUE                      0
No_of_trades               0
dtype: int64
```

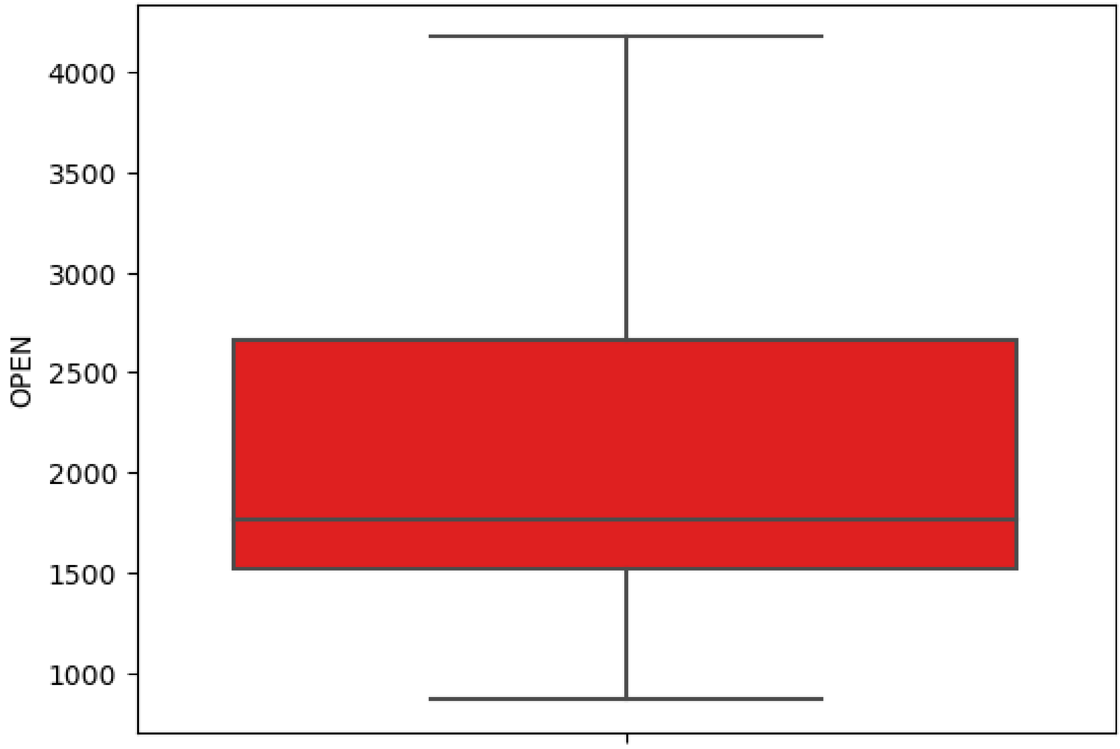
Data Visualisation Method

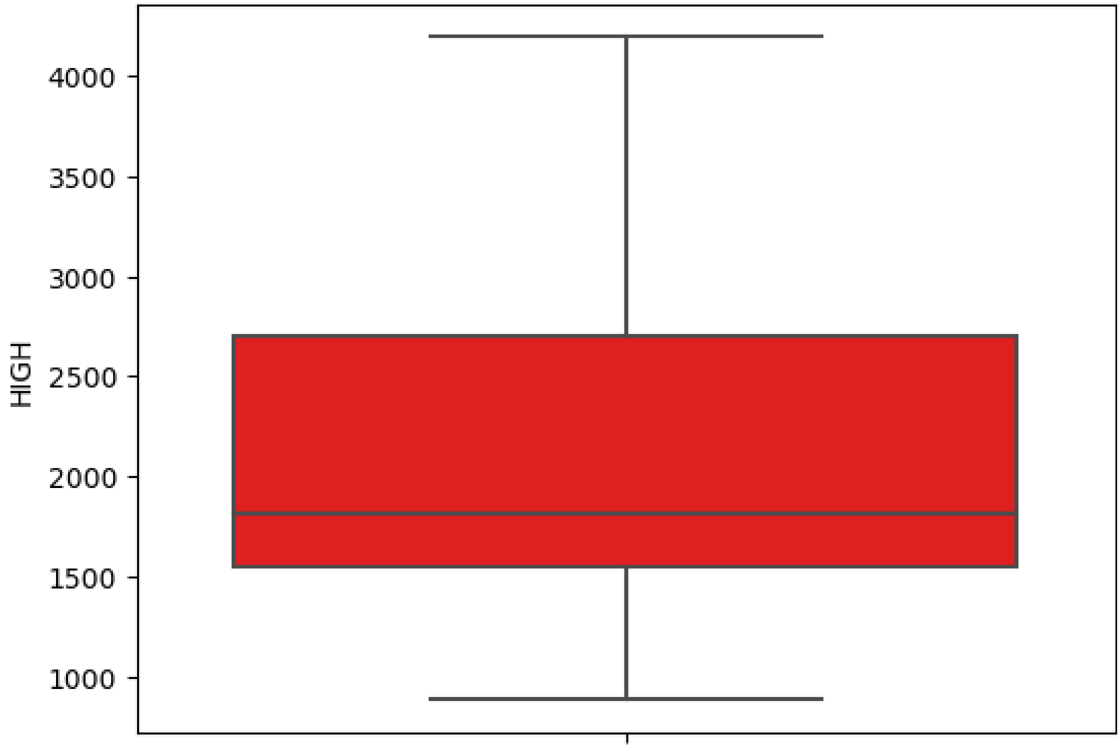
Univariate Analysis

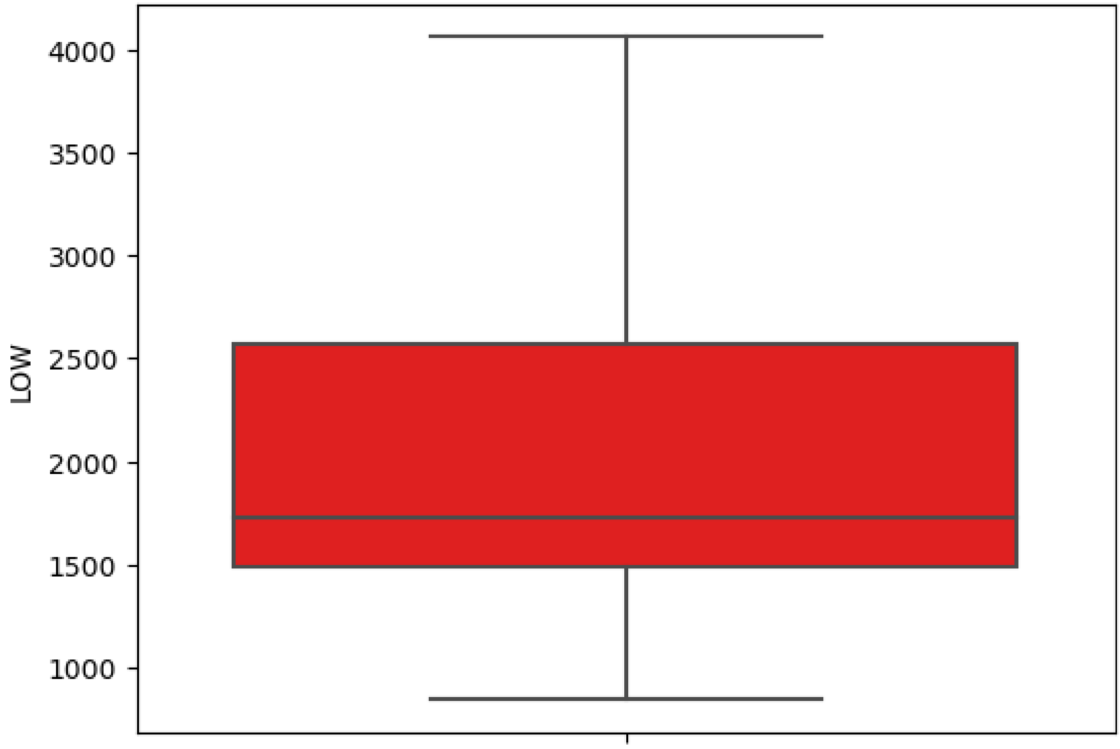
Checking Outlier

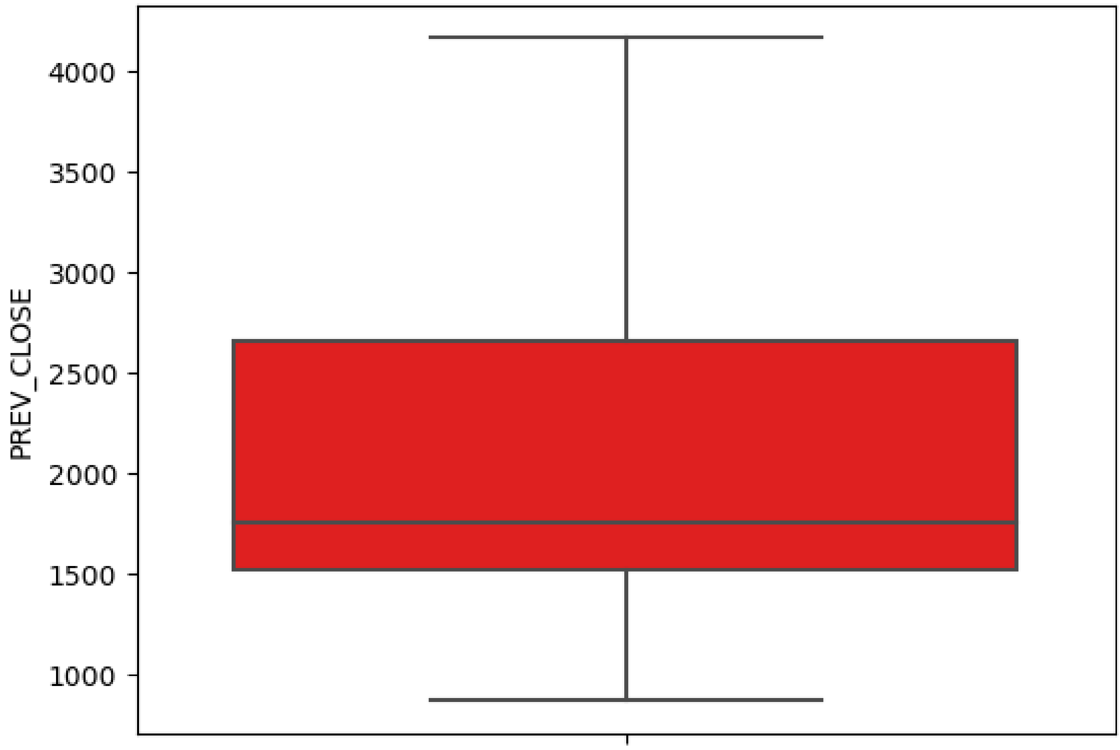
In [17]:

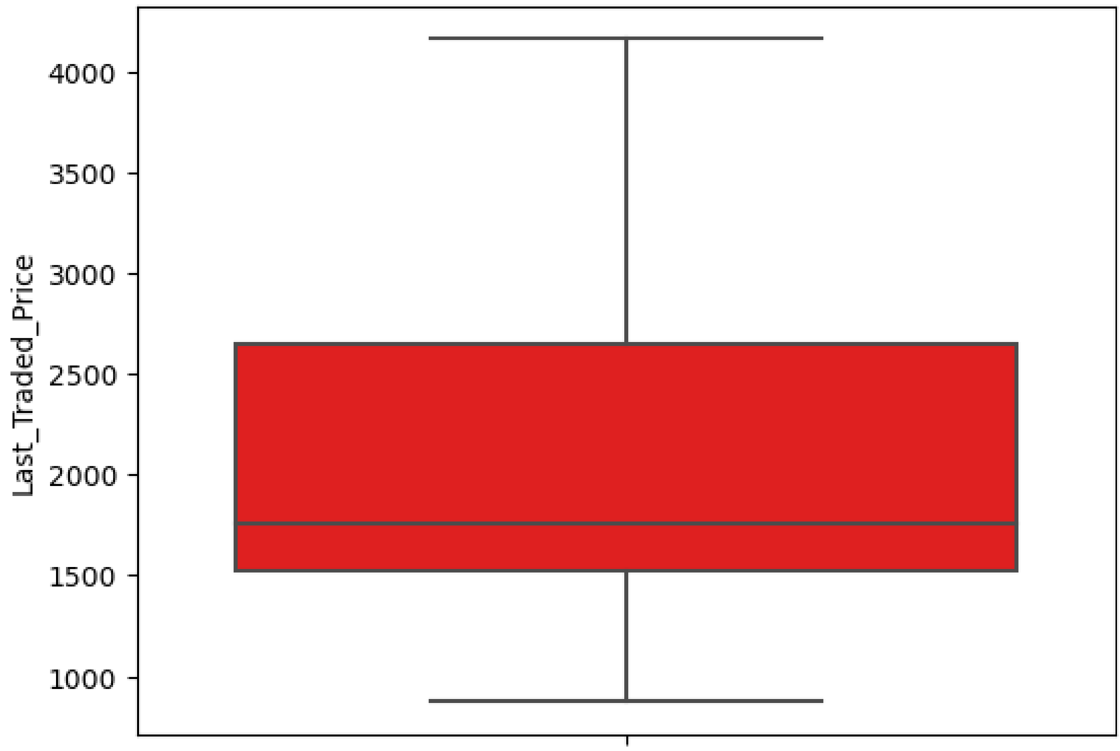
```
#Boxplot
for i in df.columns:
    if df[i].dtype!='object':
        sns.boxplot(y=df[i],color='red')
        plt.show()
```

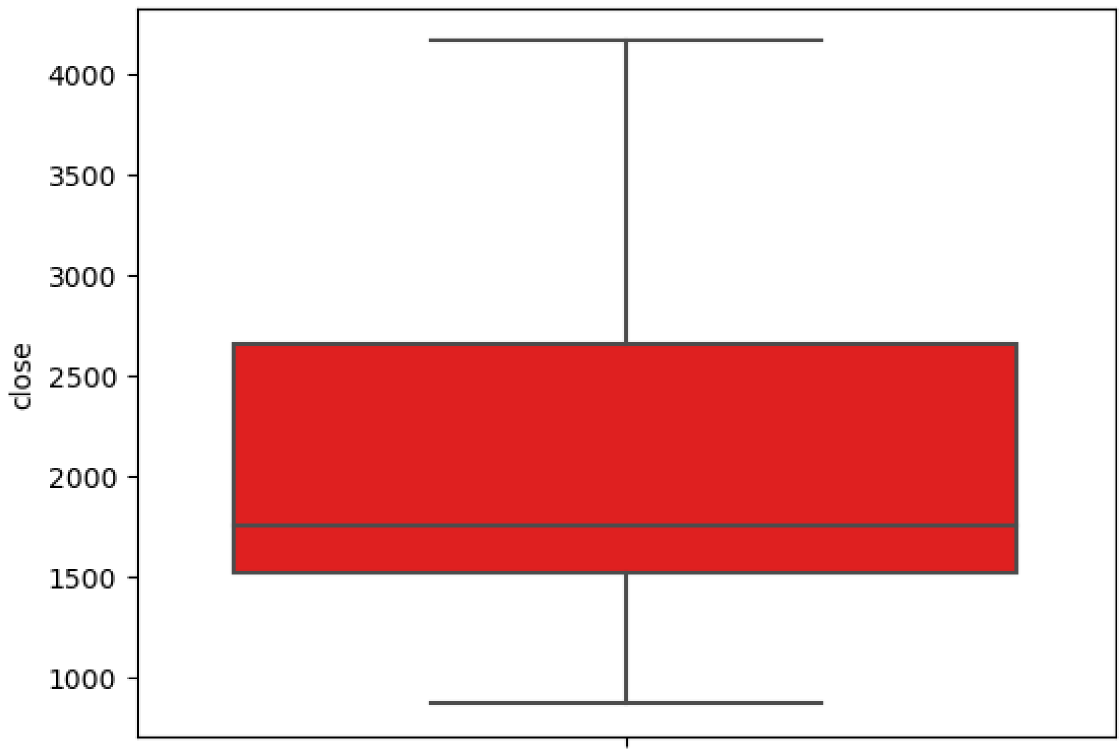


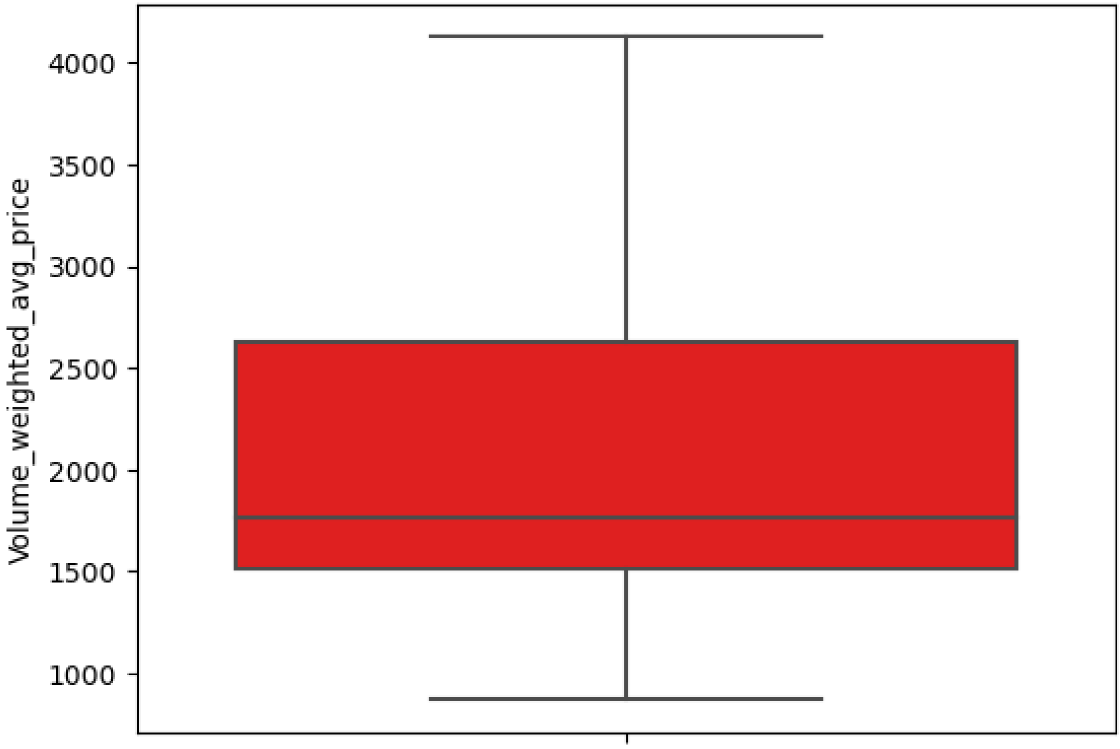


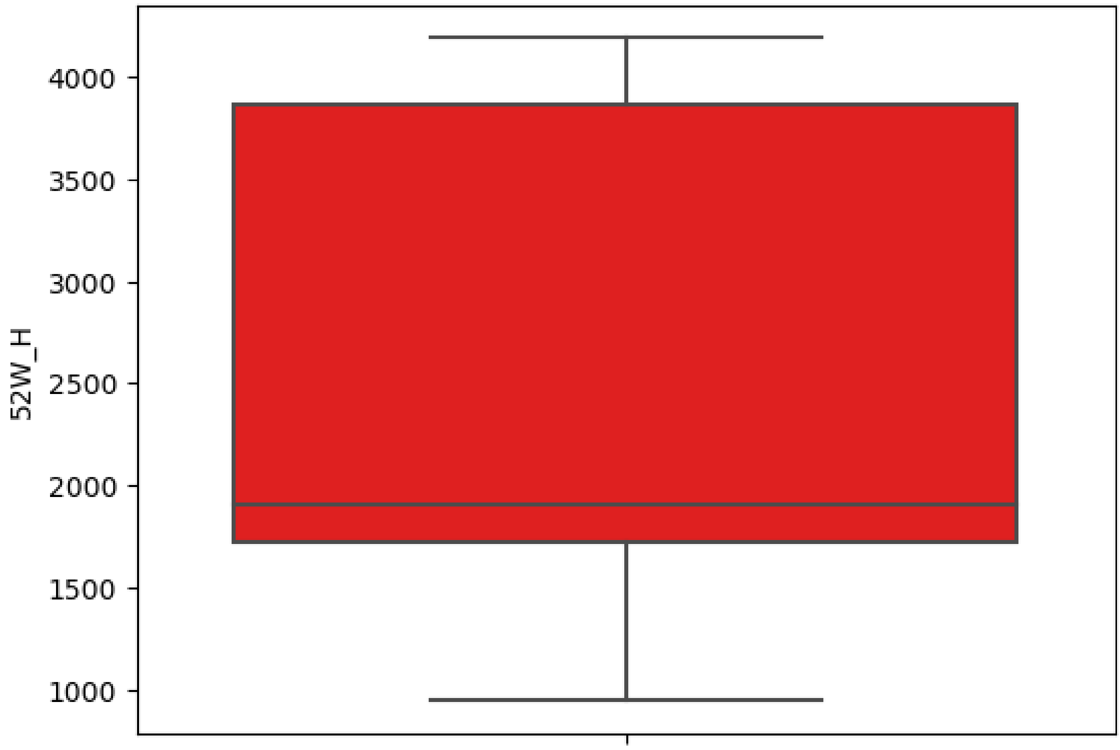

4000
3500
3000
2500
2000
1500
1000
LOW

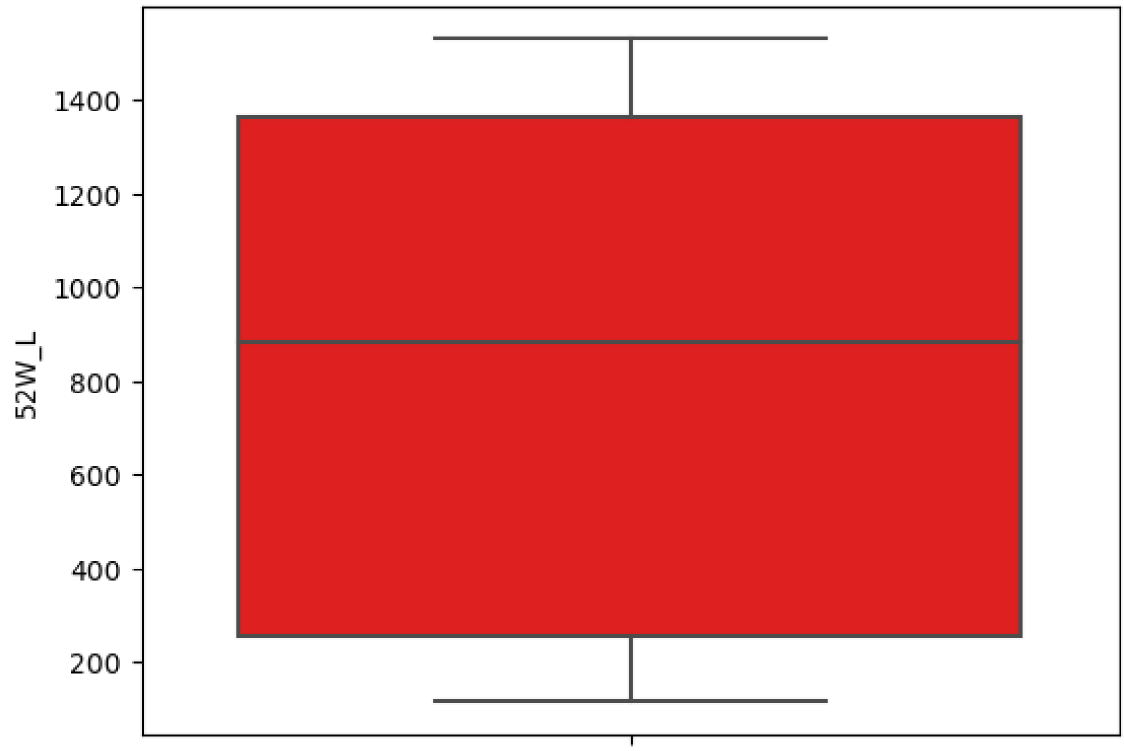

4000
3500
3000
2500
2000
1500
1000
PREV_CLOSE

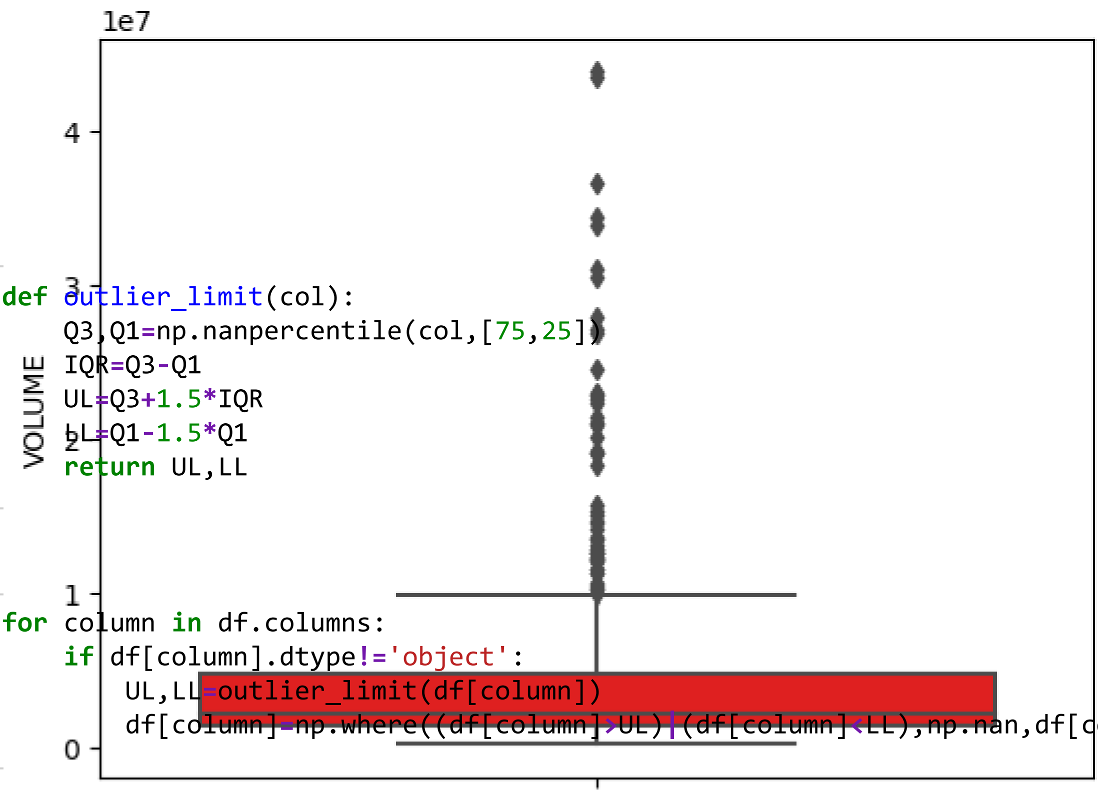

Last_Traded_Price
4000
3500
3000
2500
2000
1500
1000


close
4000
3500
3000
2500
2000
1500
1000


Volume_weighted_avg_price
4000
3500
3000
2500
2000
1500
1000


52W_H
4000
3500
3000
2500
2000
1500
1000

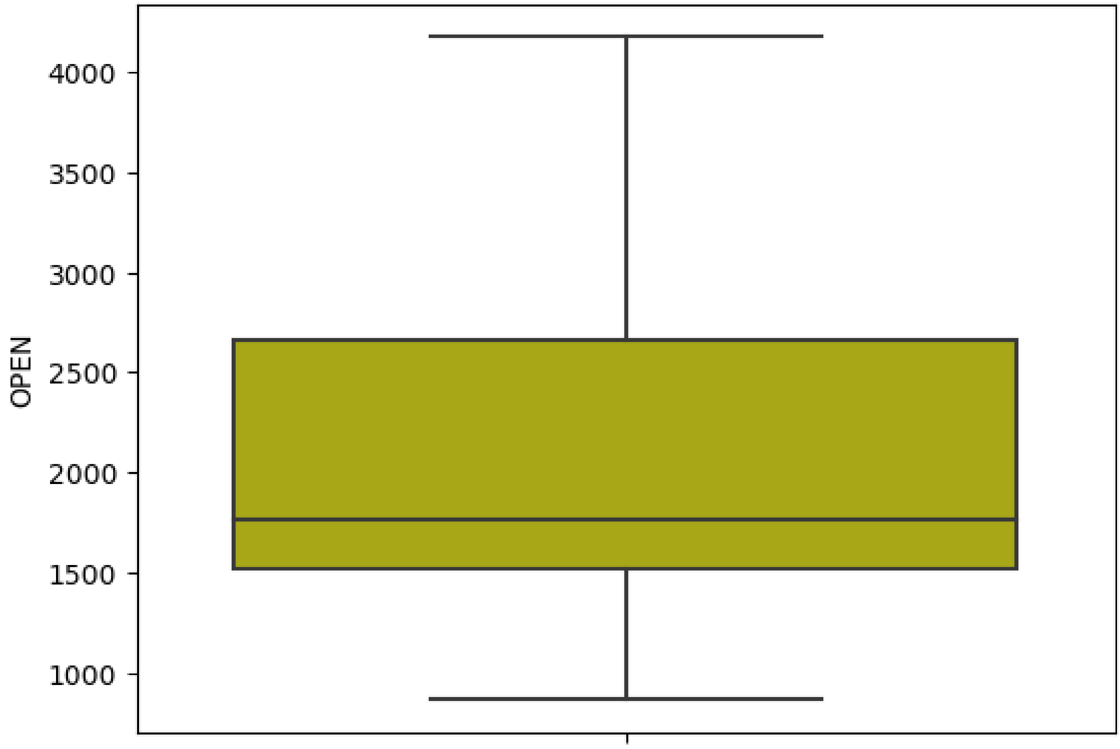


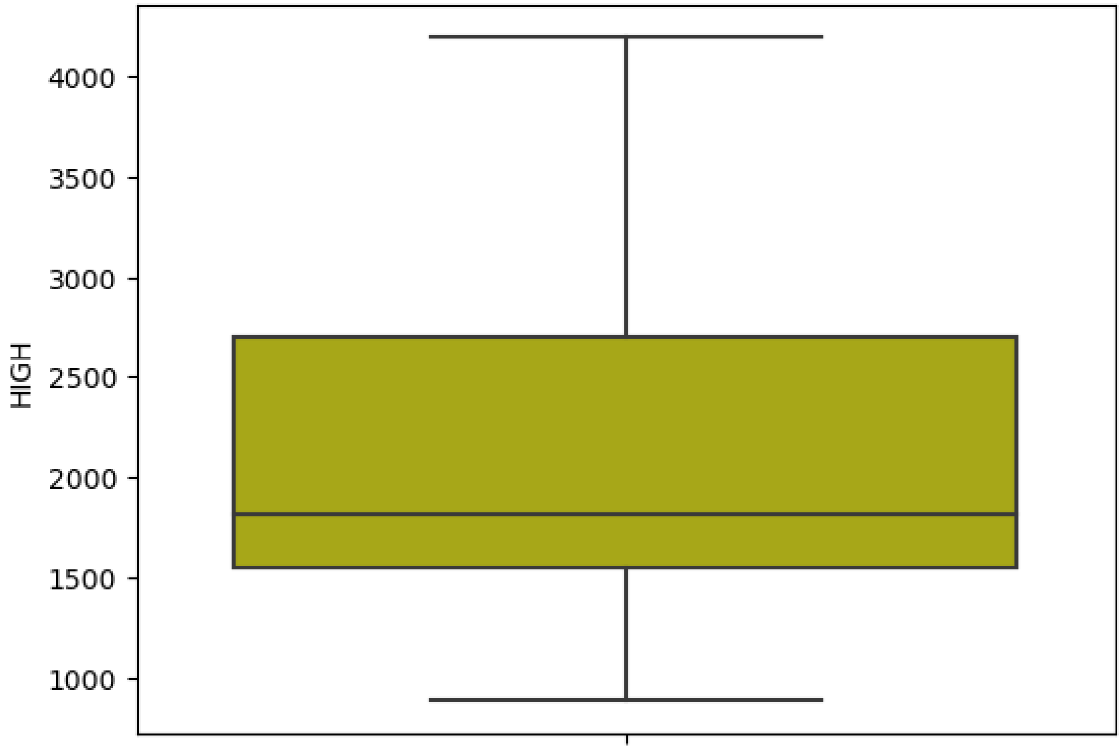


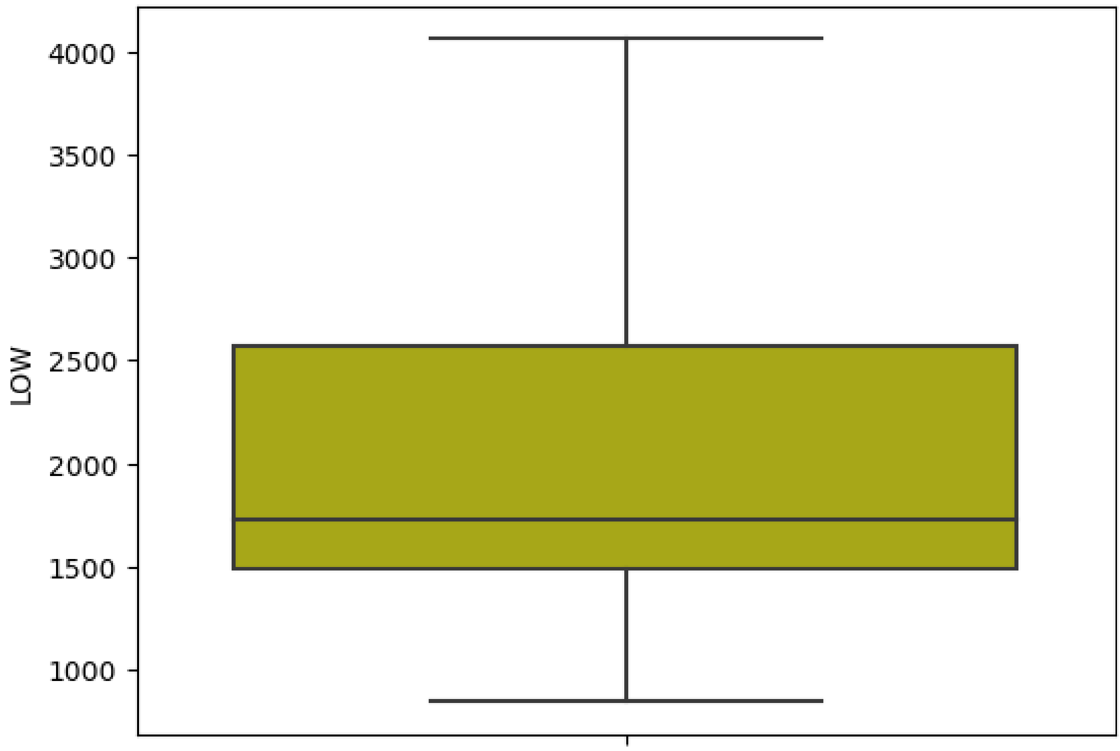

```python
def outlier_limit(col):
    Q3,Q1=np.nanpercentile(col,[75,25])
    IQR=Q3-Q1
    UL=Q3+1.5*IQR
    LL=Q1-1.5*Q1
    return UL,LL
```

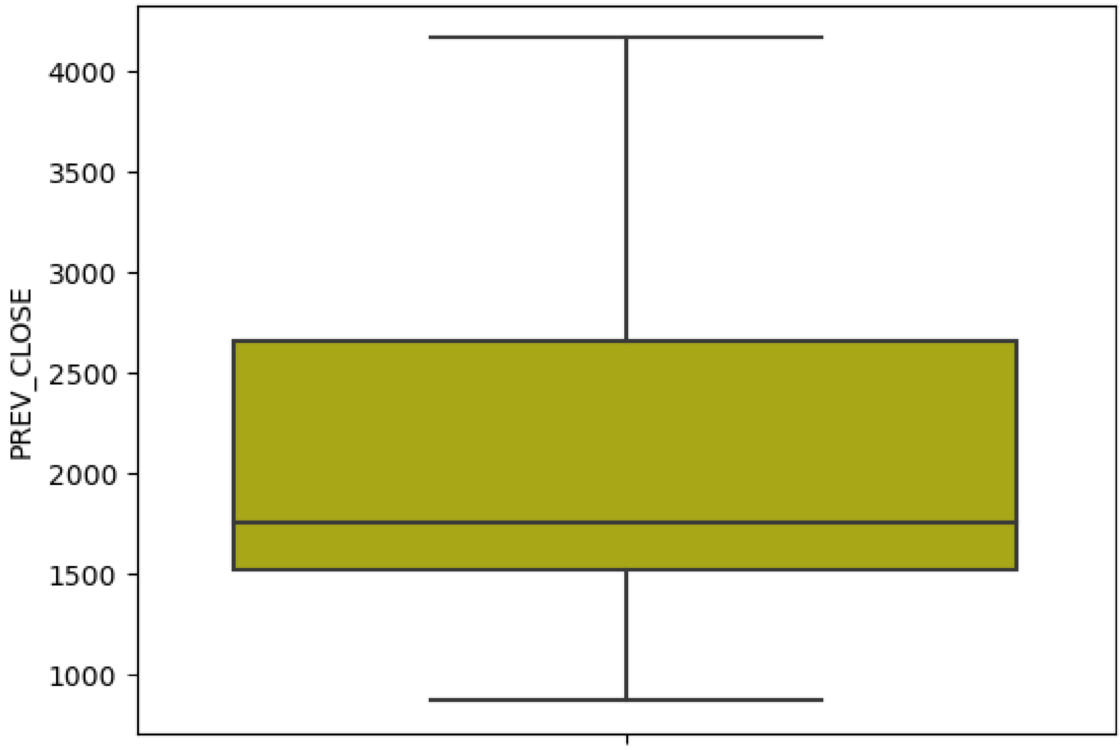
```python
for column in df.columns:
    if df[column].dtype!='object':
        UL,LL=outlier_limit(df[column])
        df[column]=np.where((df[column]>UL)|(df[column]<LL),np.nan,df[column])
```

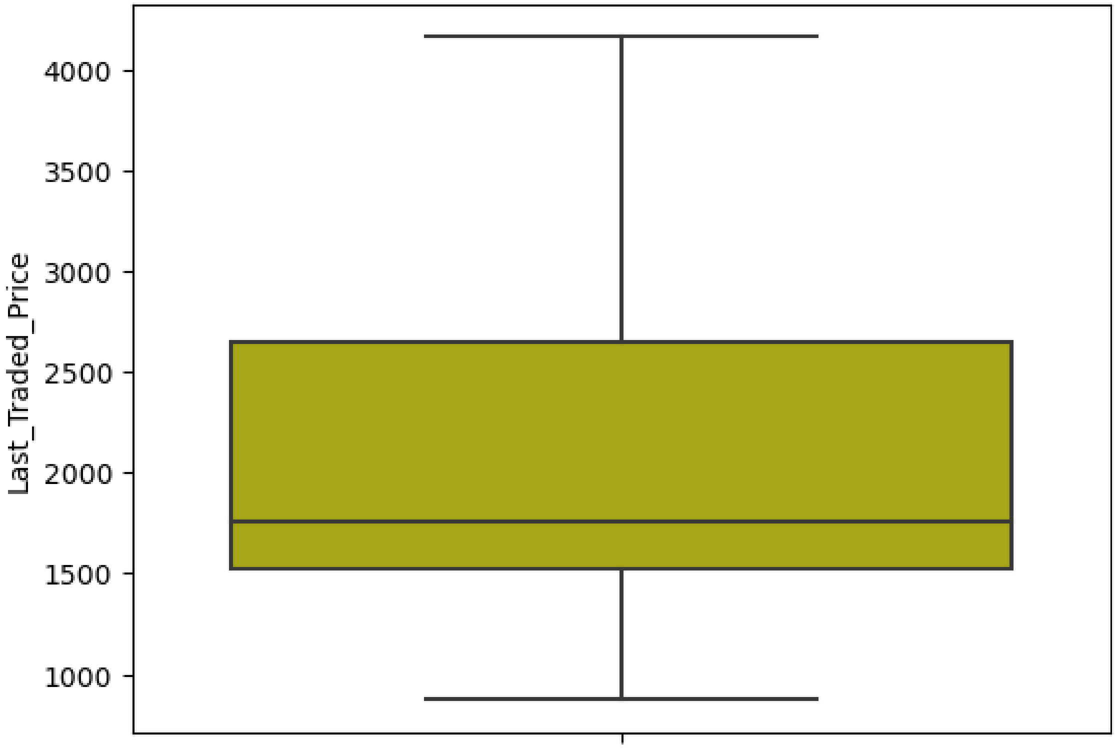
```
for i in df.columns:
    if df[i].dtype!='object':
        sns.boxplot(y=df[i],color='y')
        plt.show()
```

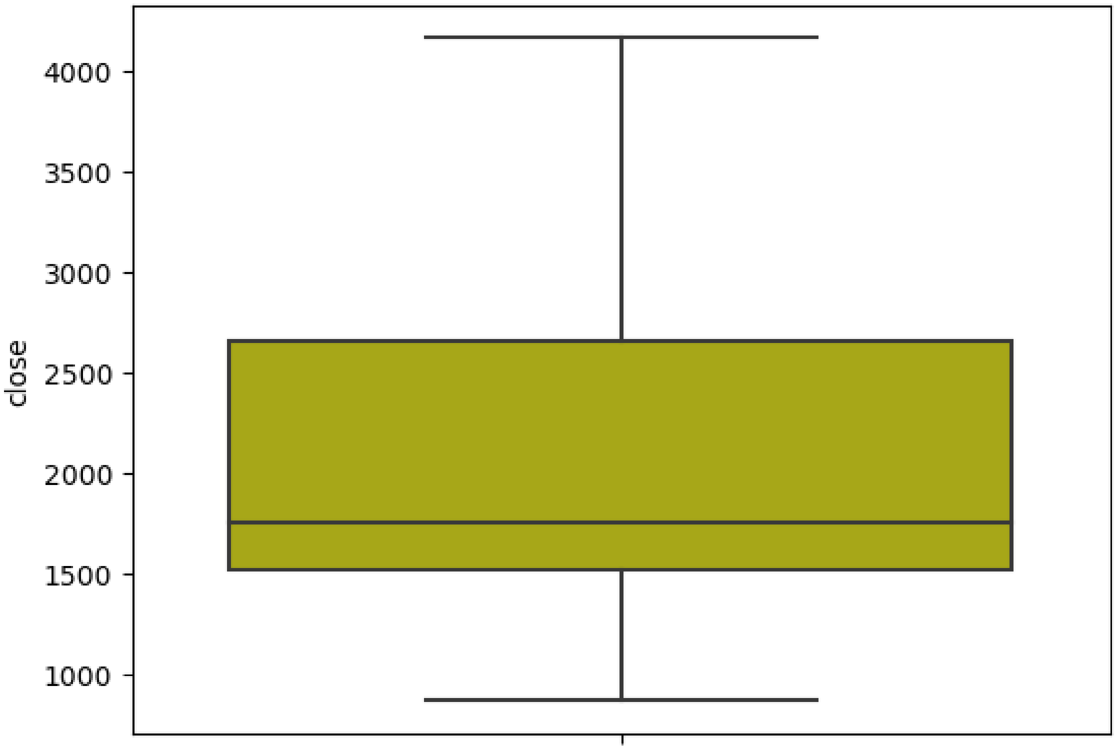


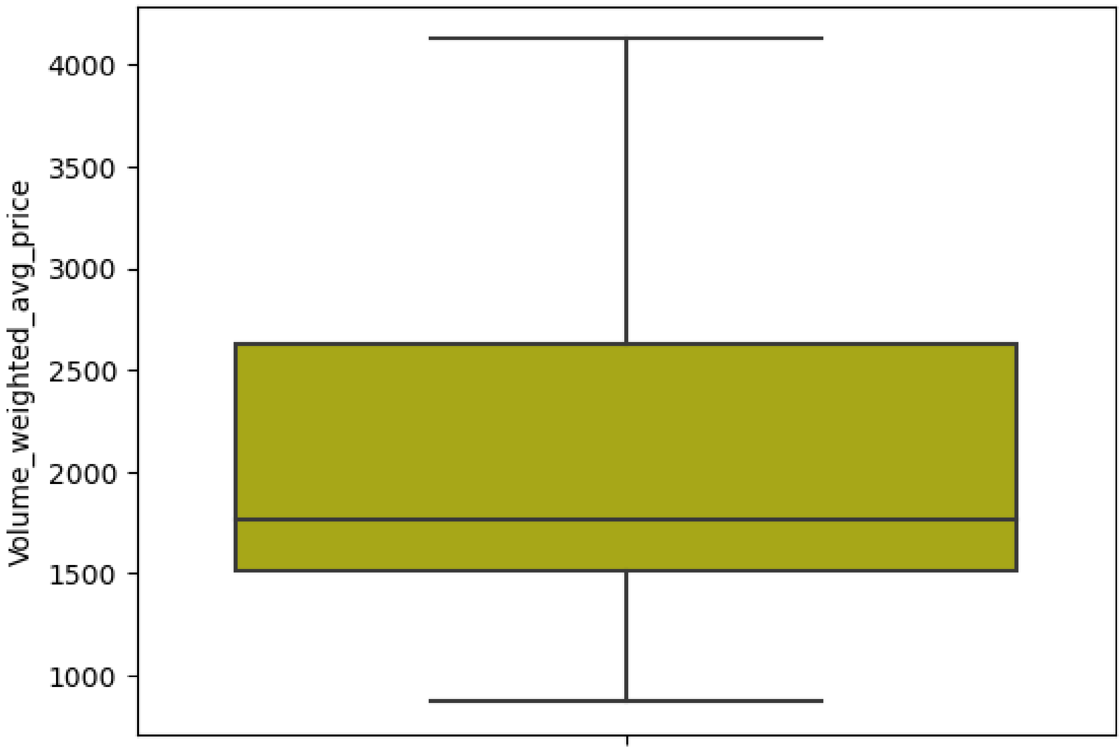


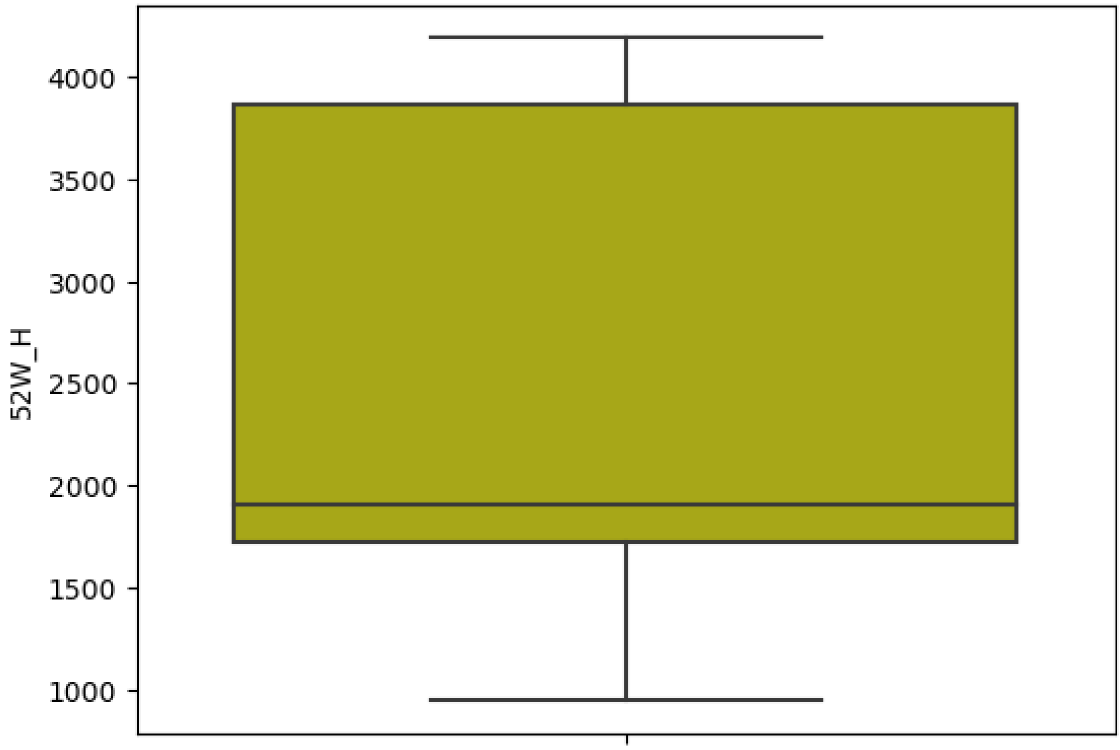

LOW
4000
3500
3000
2500
2000
1500
1000

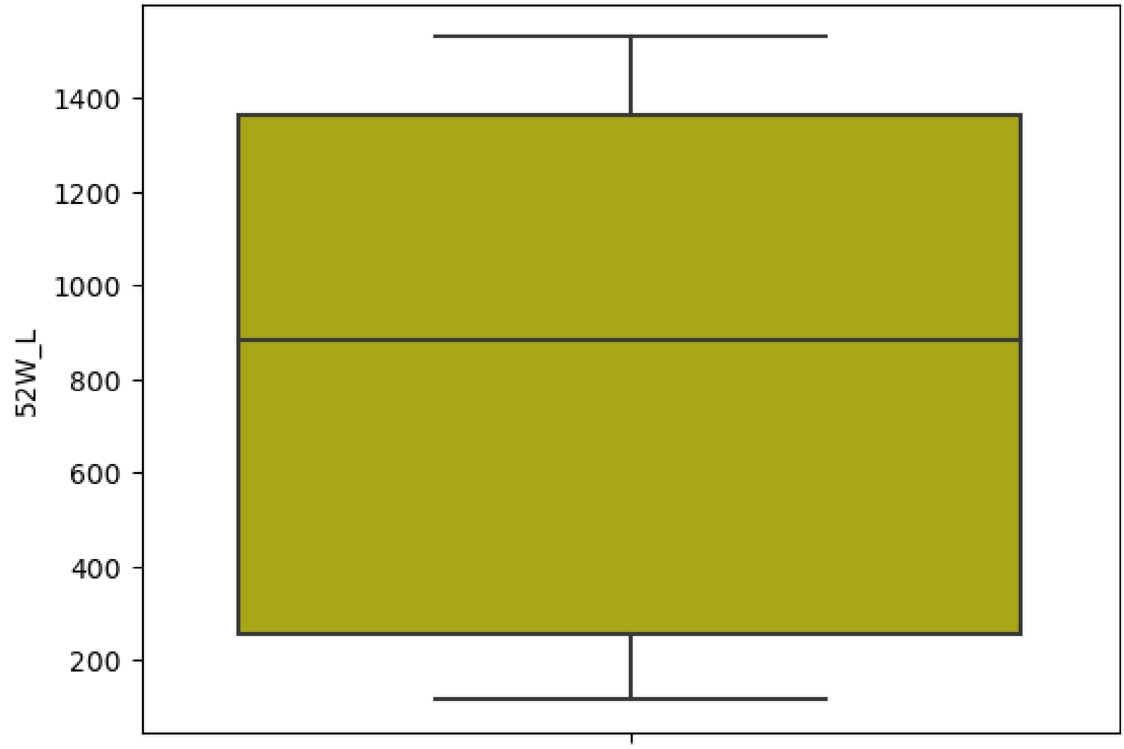

PREV_CLOSE
4000
3500
3000
2500
2000
1500
1000


4000
3500
3000
2500
2000
1500
1000
Last_Traded_Price

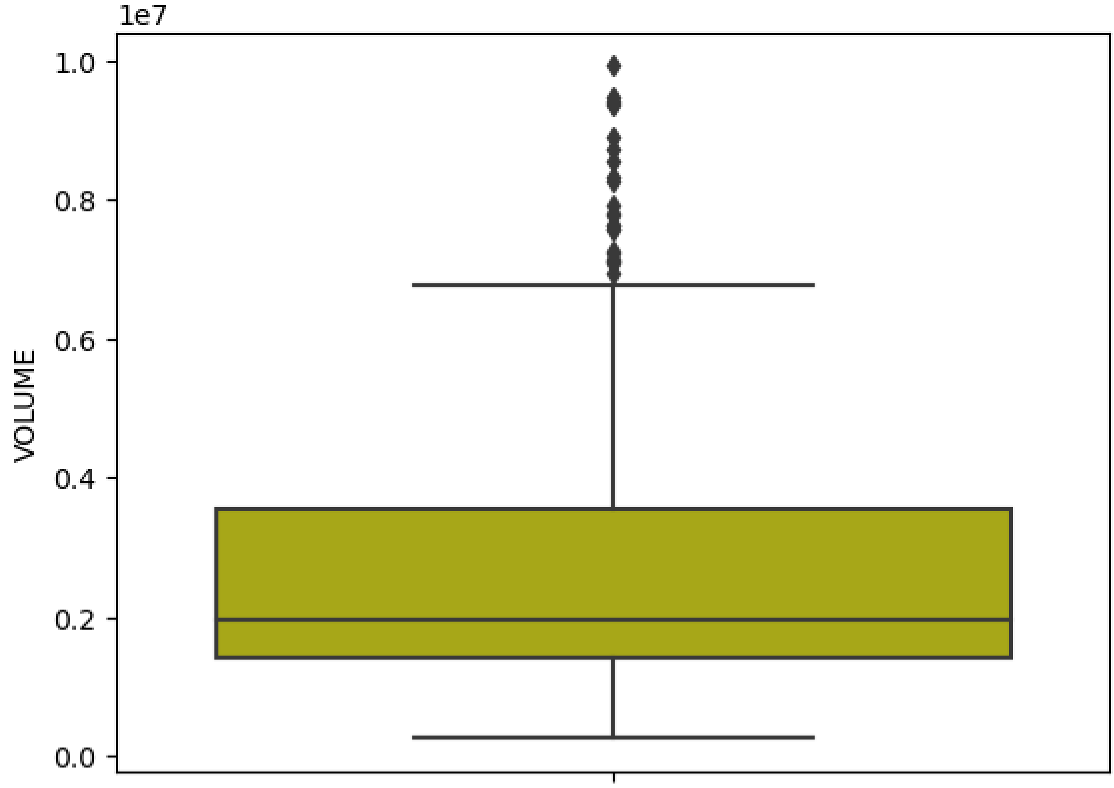

4000
3500
3000
2500
2000
1500
1000
close

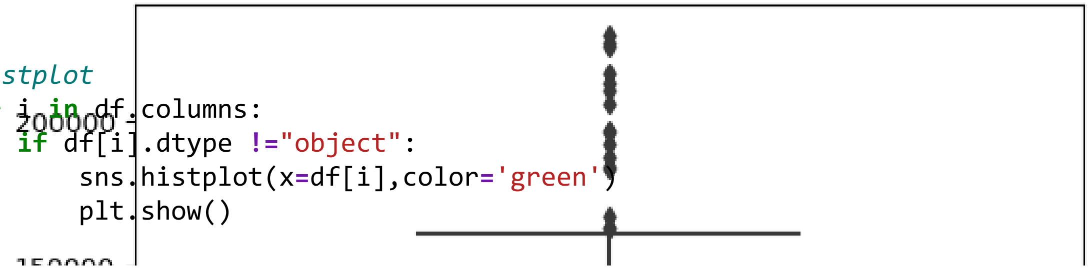

Volume_weighted_avg_price
4000
3500
3000
2500
2000
1500
1000


52W_H
4000
3500
3000
2500
2000
1500
1000

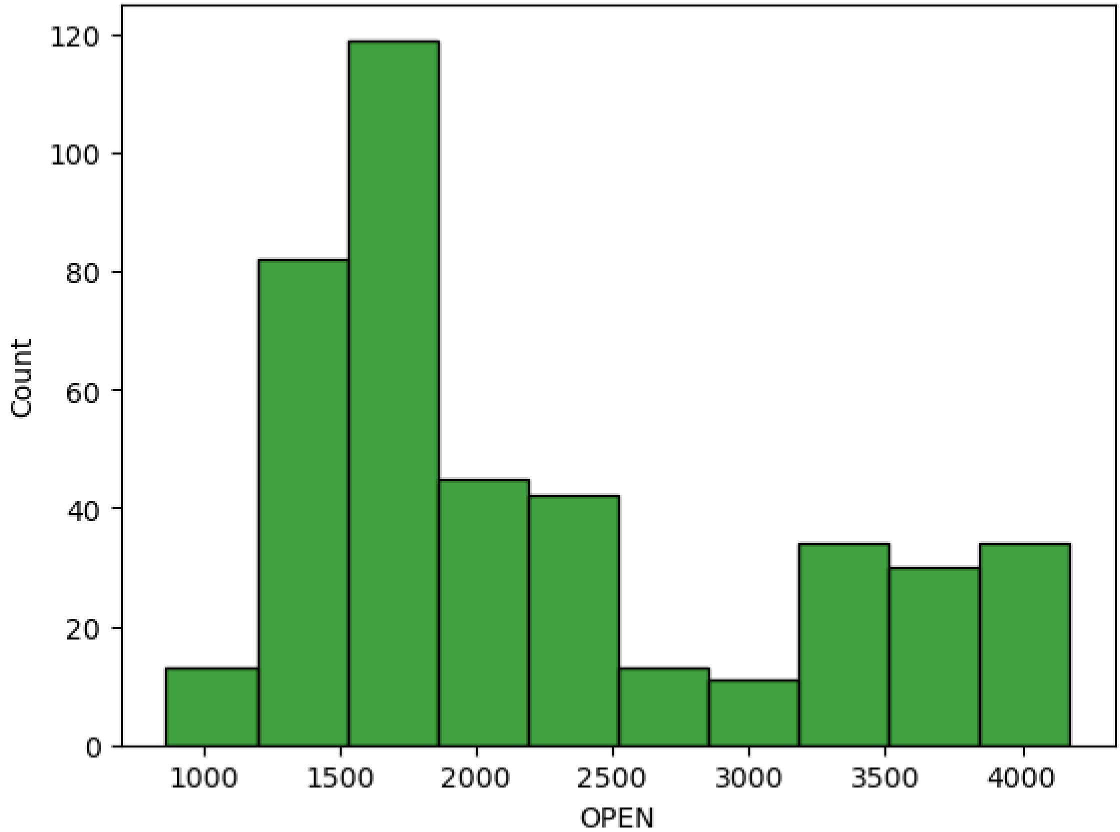


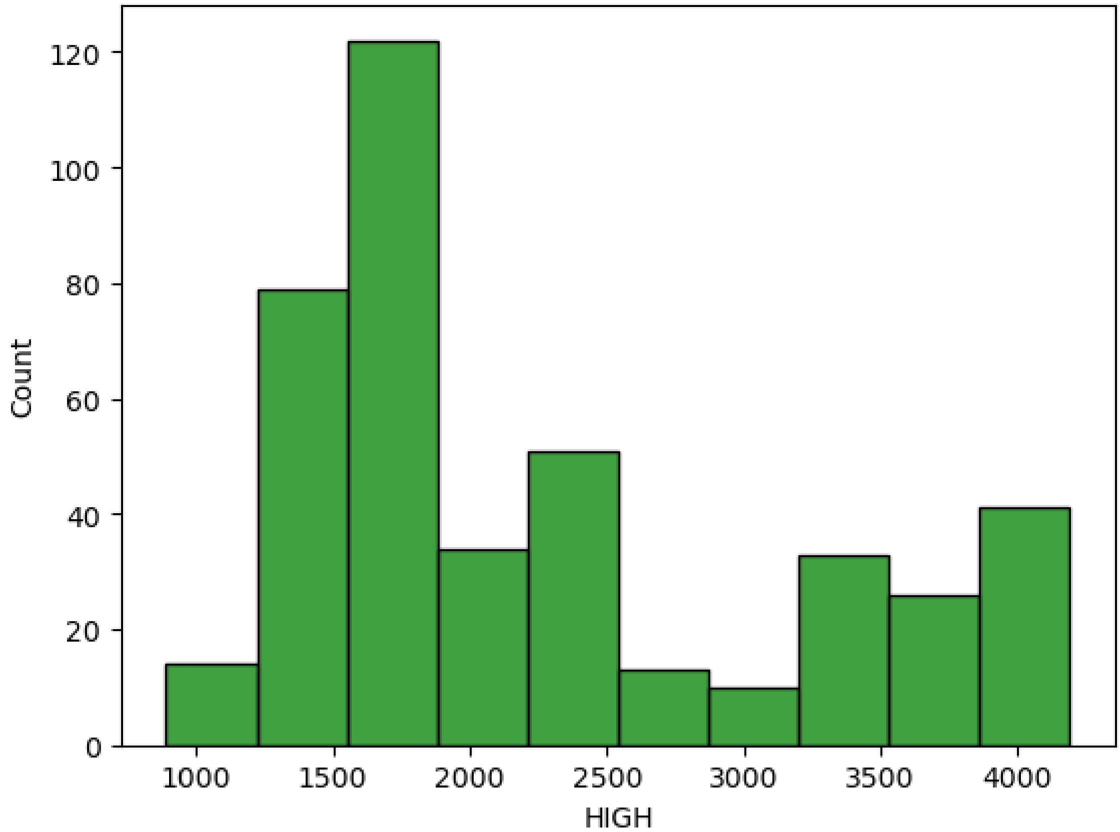

```
df.dropna(inplace=True)
```

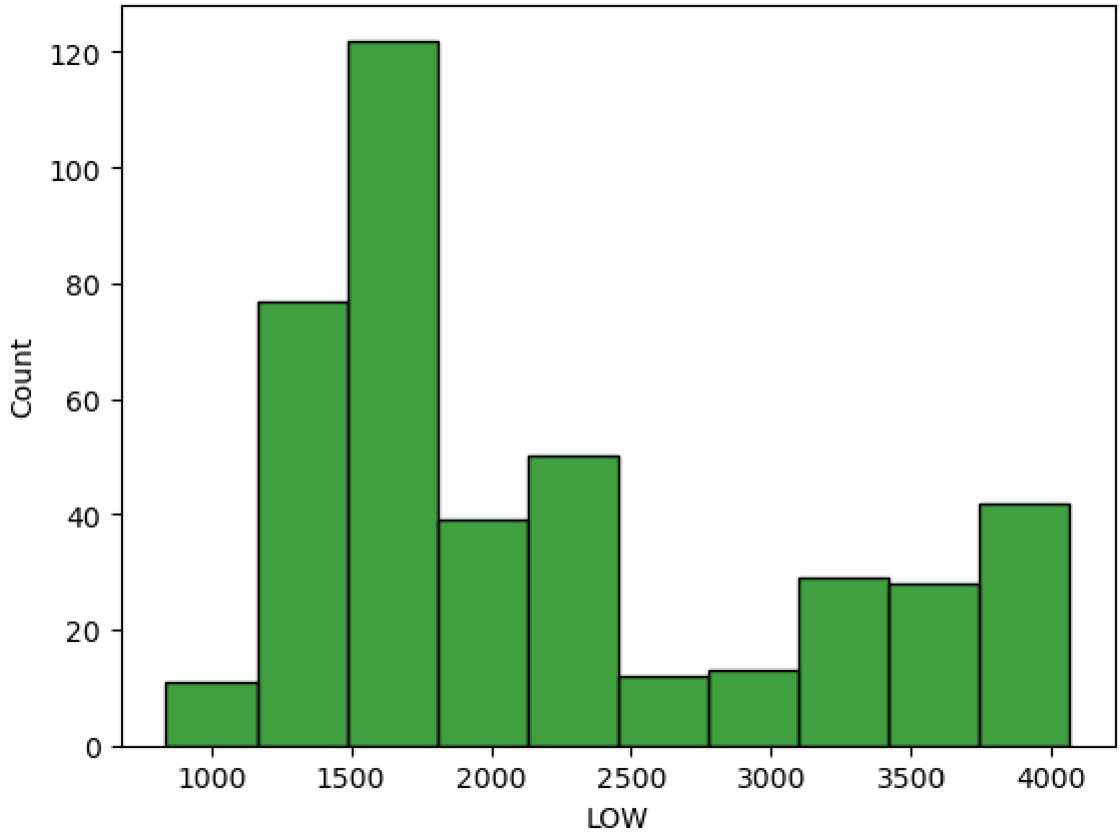


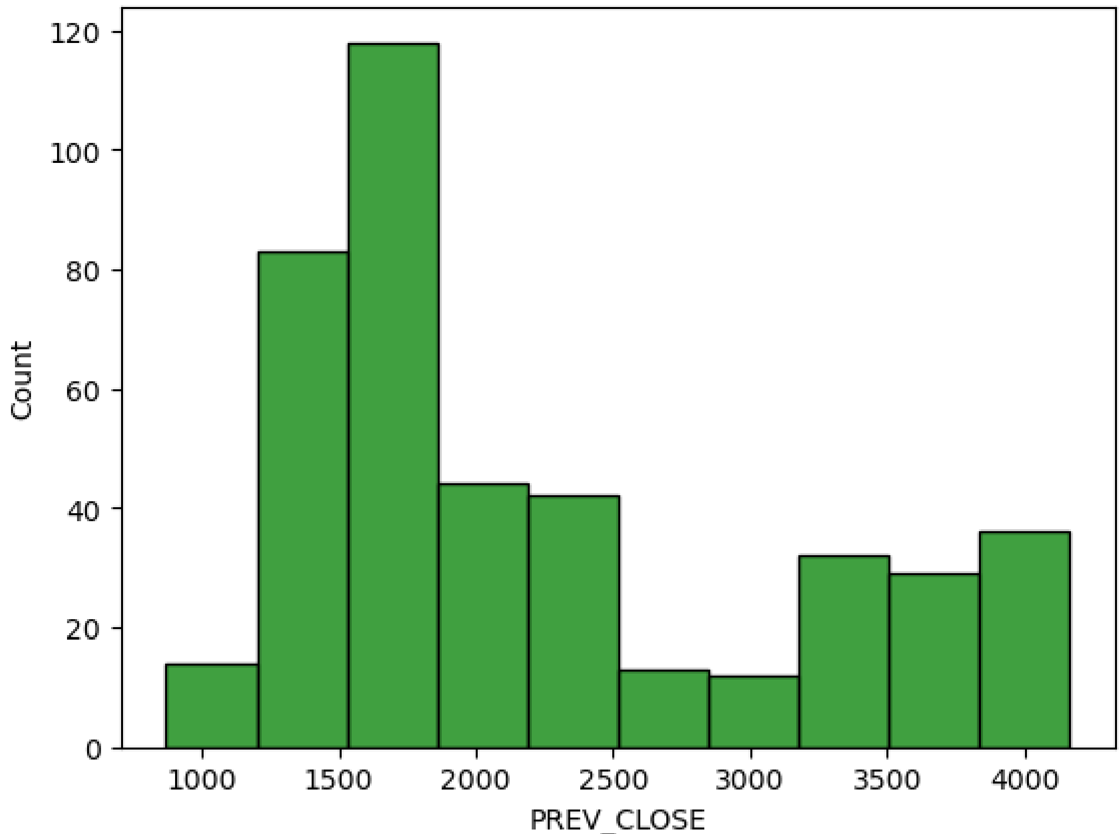


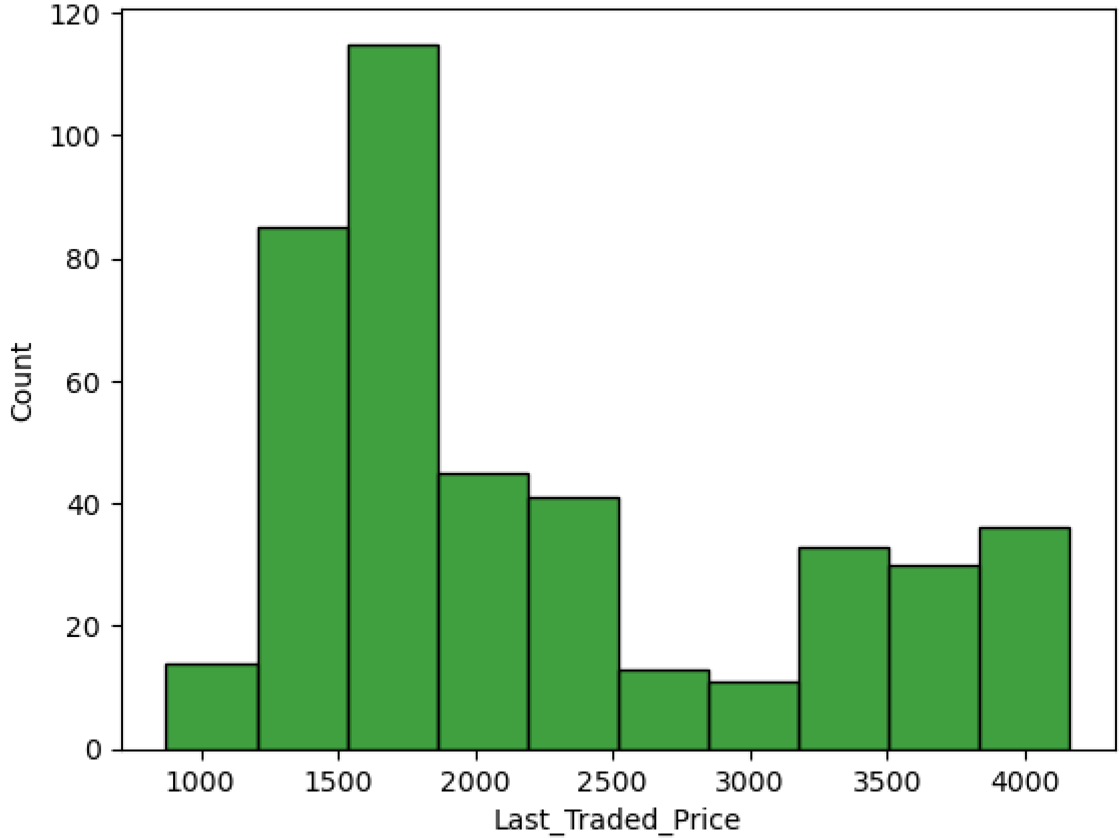

```
#histplot
for i in df.columns:
    if df[i].dtype !="object":
        sns.histplot(x=df[i],color='green')
        plt.show()
```

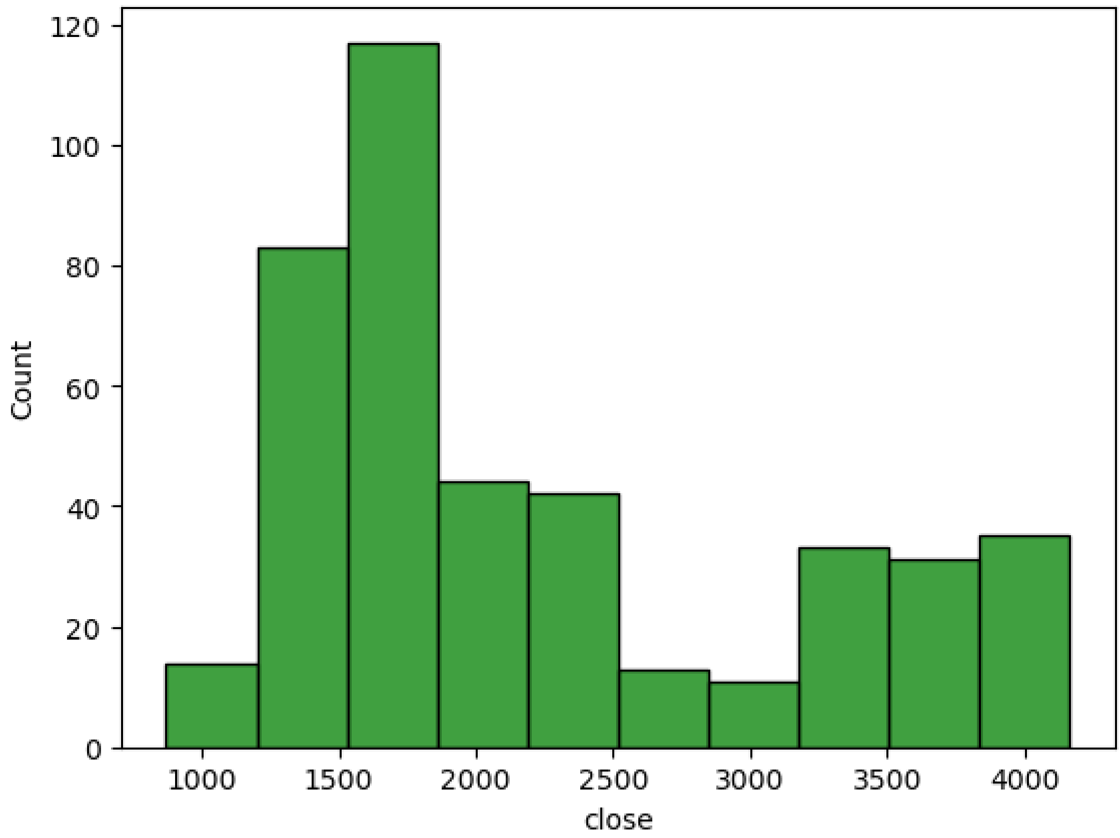


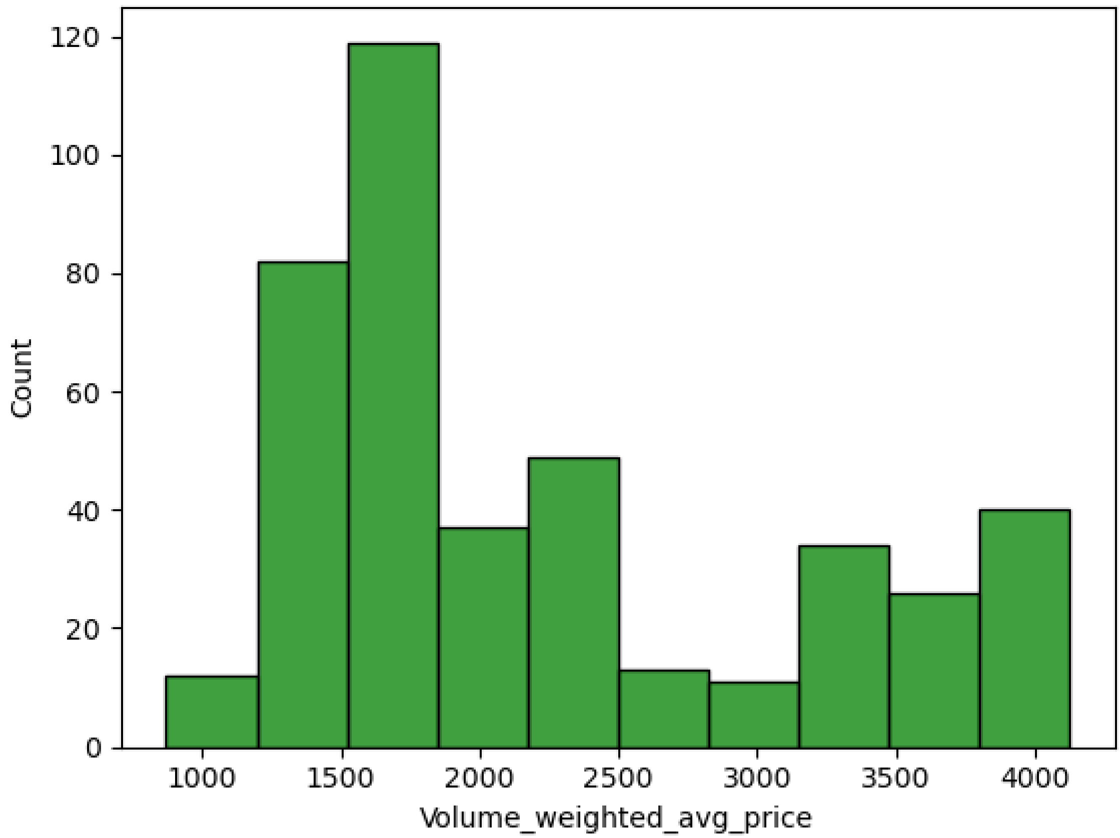



Count
LOW
0
20
40
60
80
100
120
1000
1500
2000
2500
3000
3500
4000

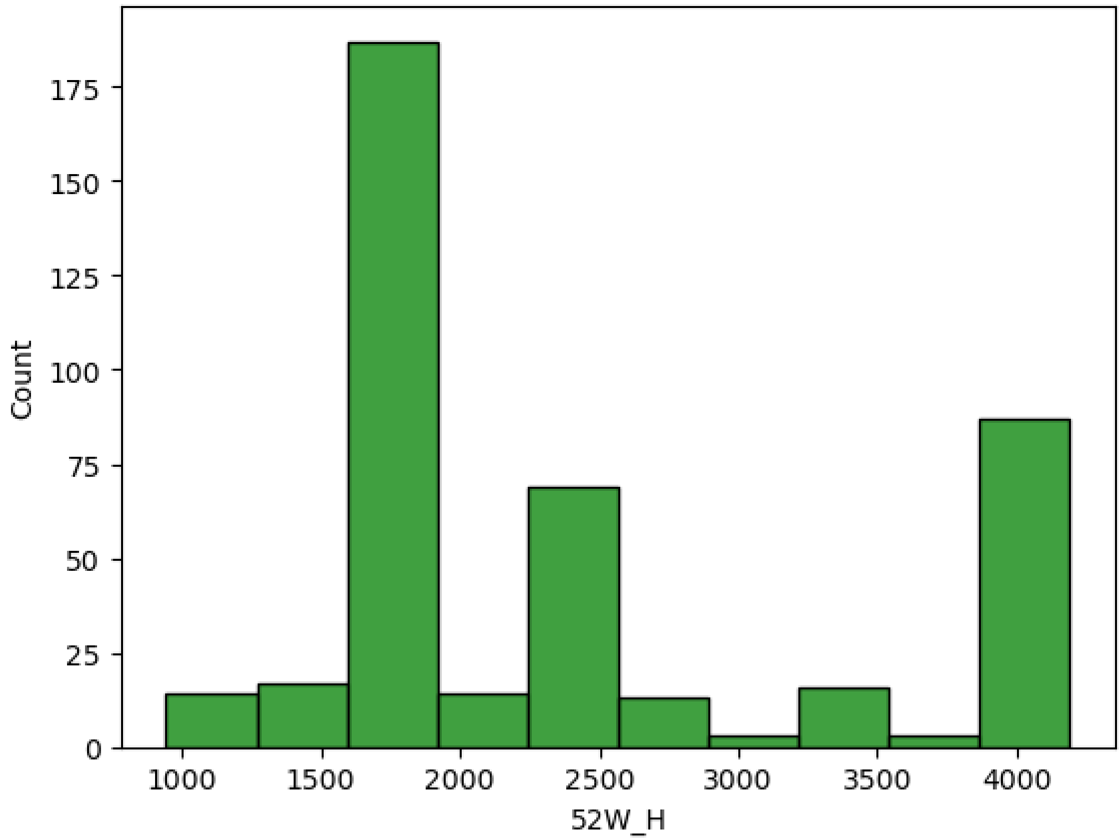

Count
PREV_CLOSE
0
20
40
60
80
100
120
1000
1500
2000
2500
3000
3500
4000


120
100
80
60
40
20
0
Count
1000
1500
2000
2500
3000
3500
4000
Last_Traded_Price

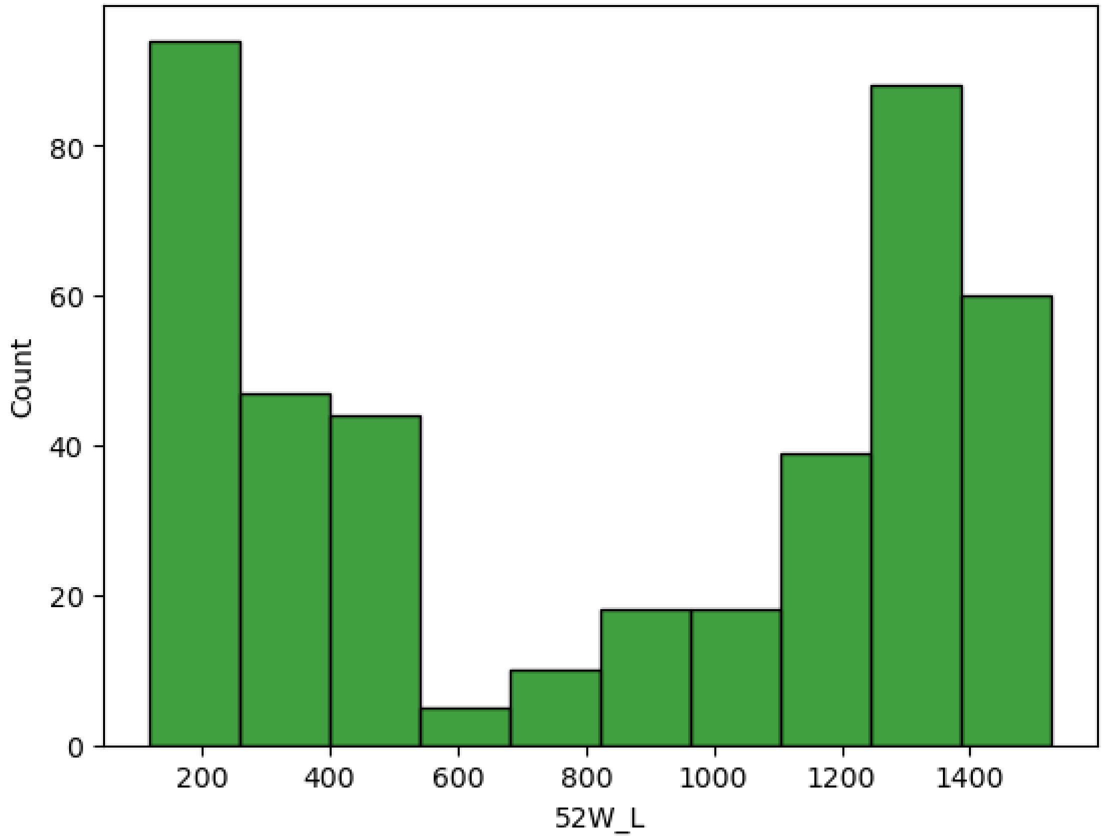

120
100
80
60
40
20
0
Count
1000
1500
2000
2500
3000
3500
4000
close

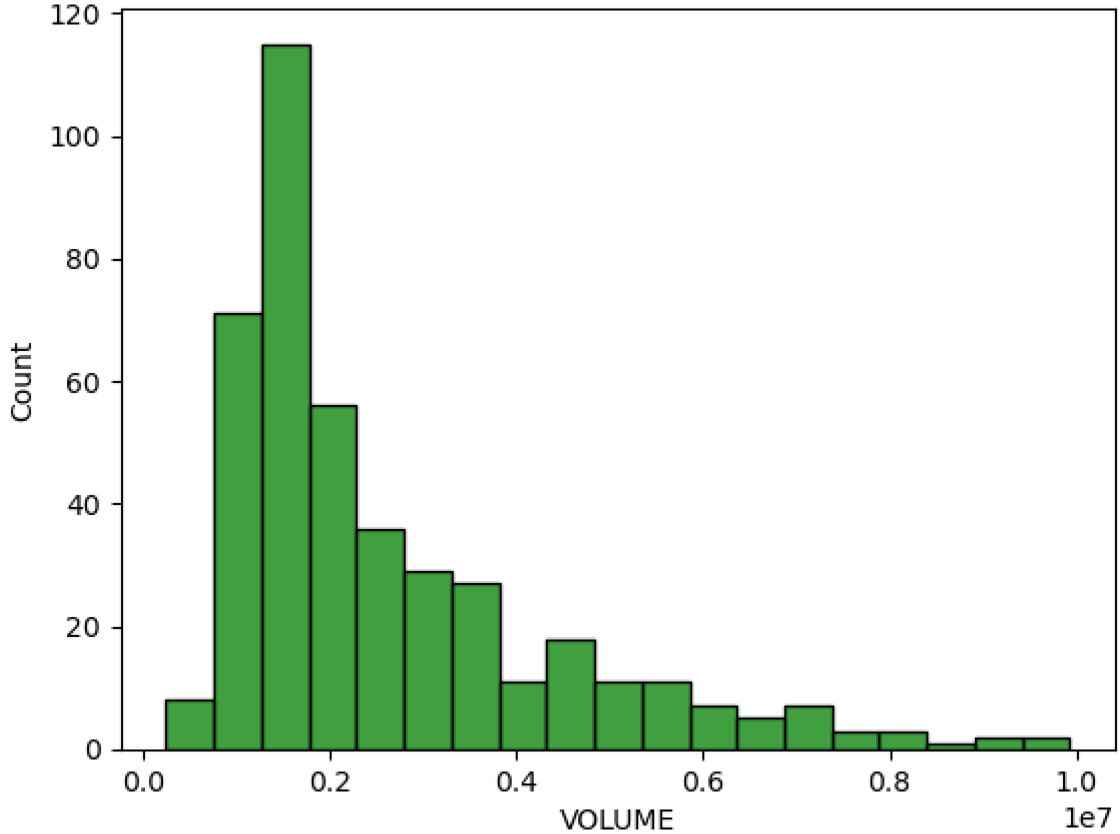


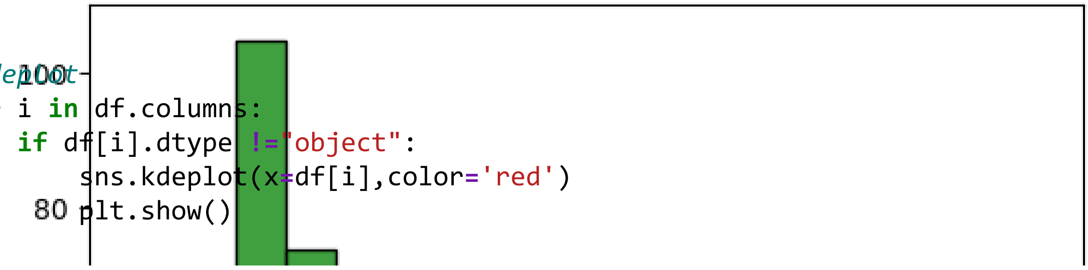
120
100
80
60
40
20
0
Count
1000
1500
2000
2500
3000
3500
4000
Volume_weighted_avg_price



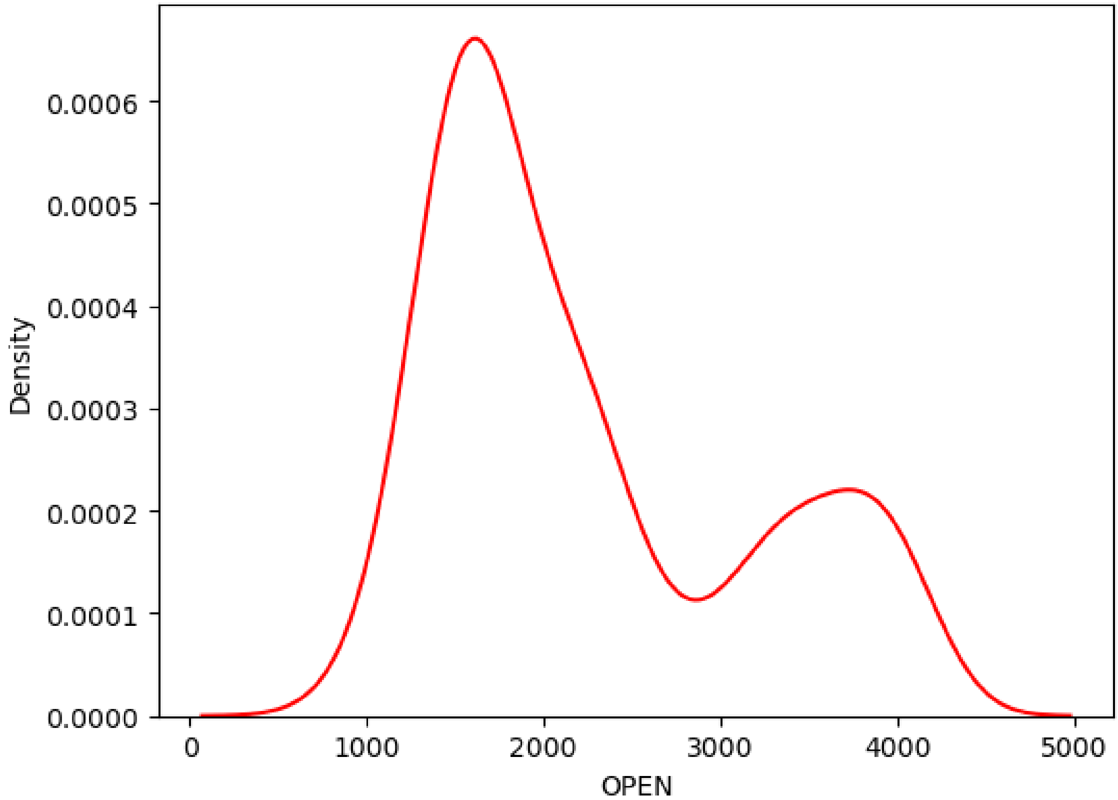
175
150
125
100
75
50
25
0
Count
1000
1500
2000
2500
3000
3500
4000
52W_H

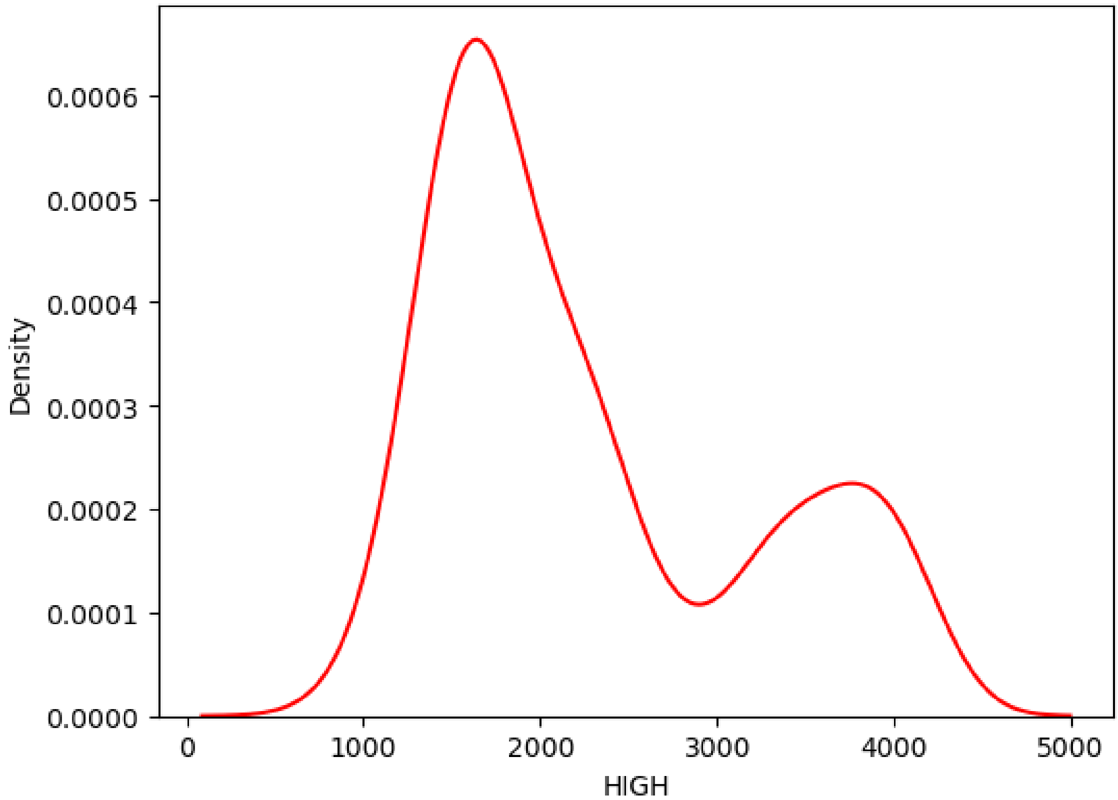

Count
0
20
40
60
80
200
400
600
800
1000
1200
1400
52W_L

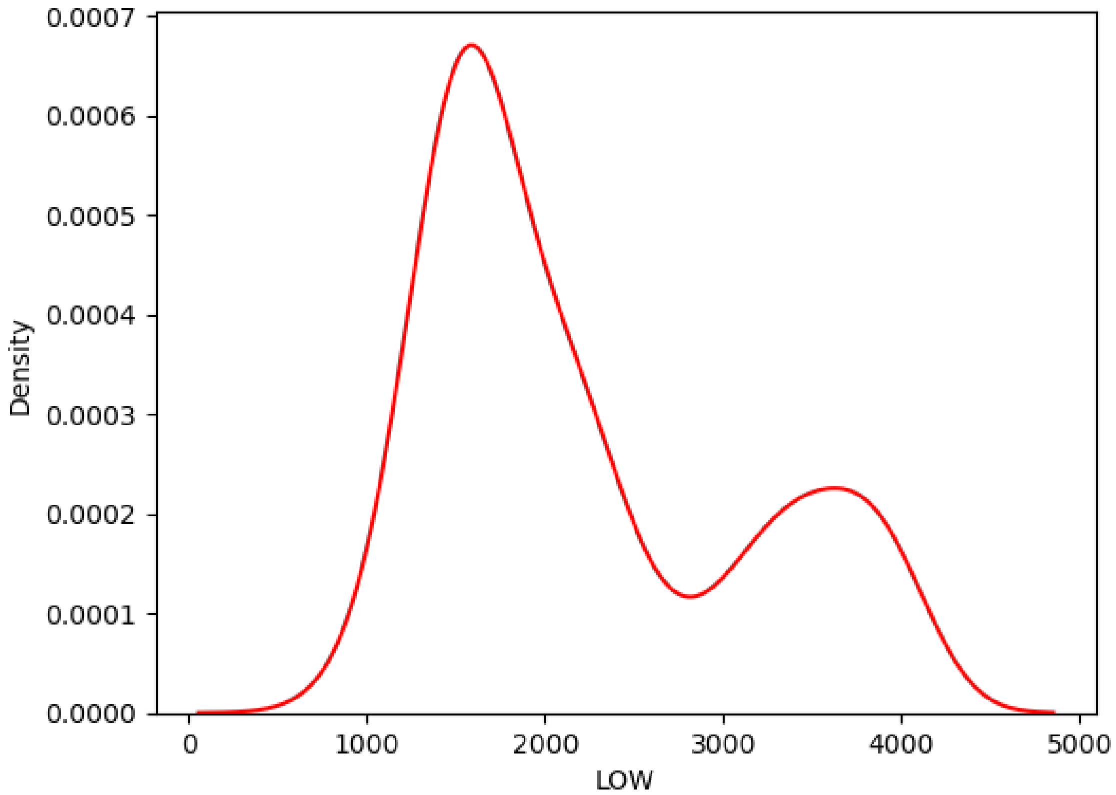

Count
0
20
40
60
80
100
120
0.0
0.2
0.4
0.6
0.8
1.0
VOLUME
1e7

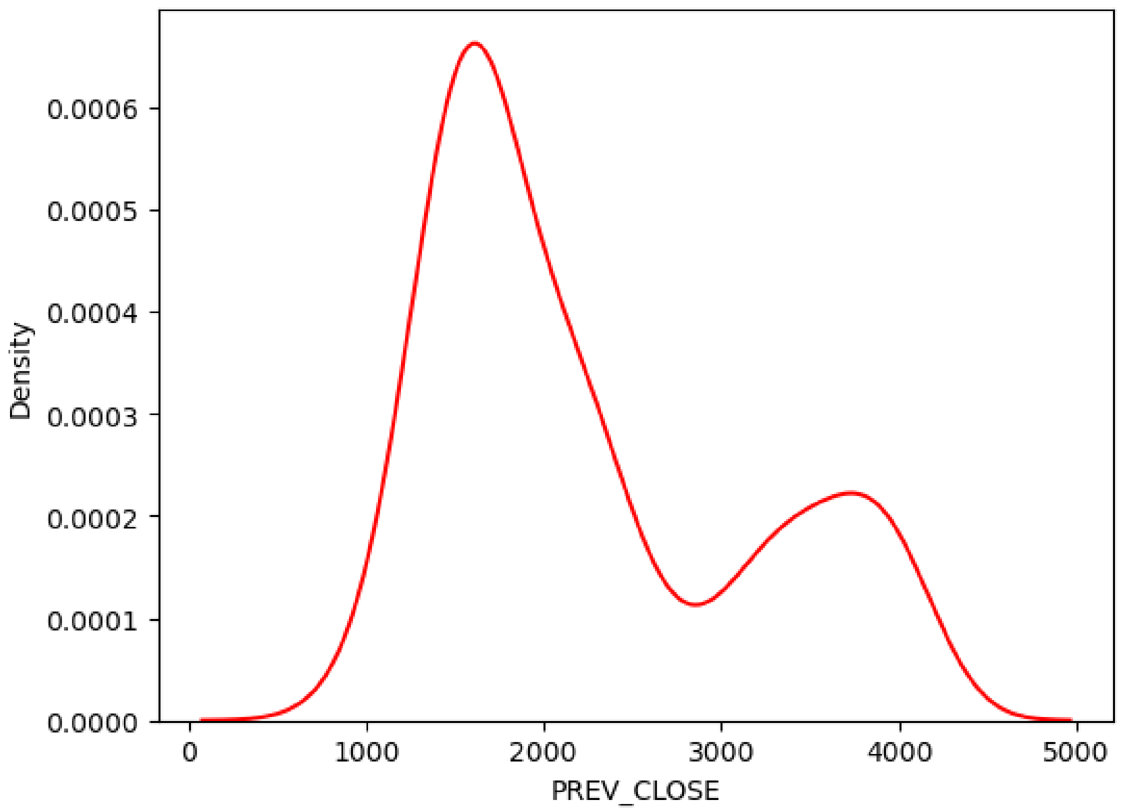


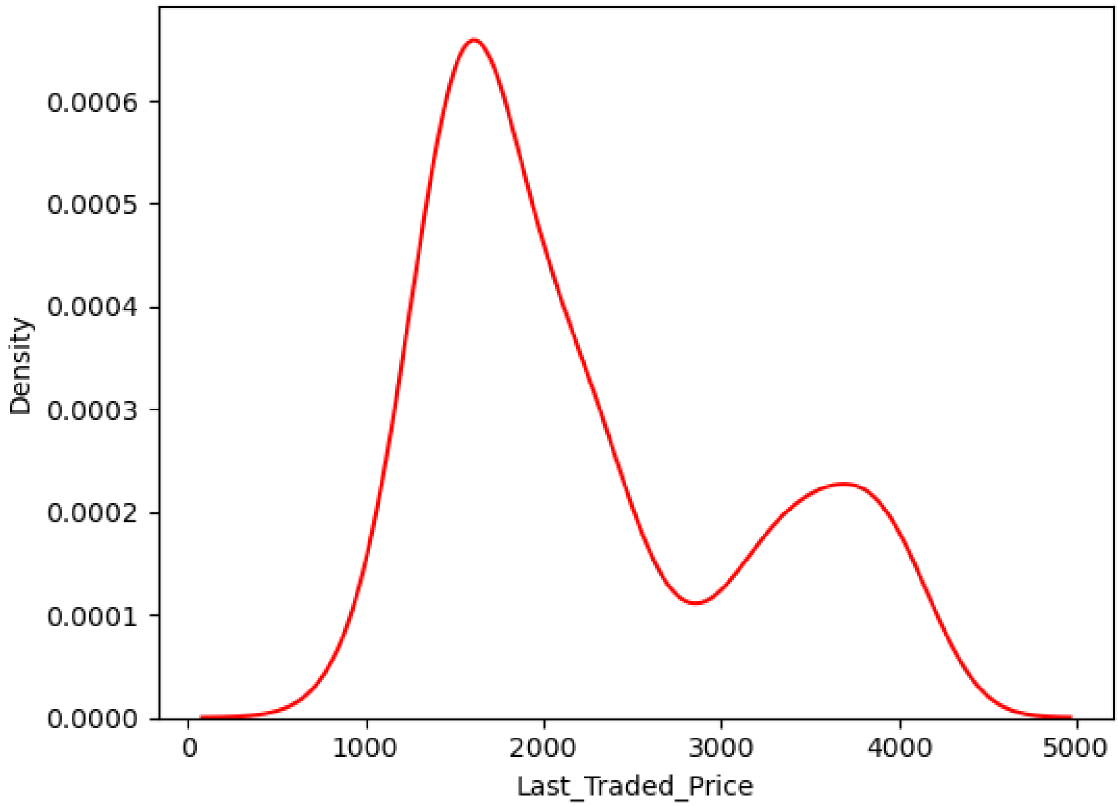

```python
#kdeplot
for i in df.columns:
    if df[i].dtype !="object":
        sns.kdeplot(x=df[i],color='red')
        plt.show()
```

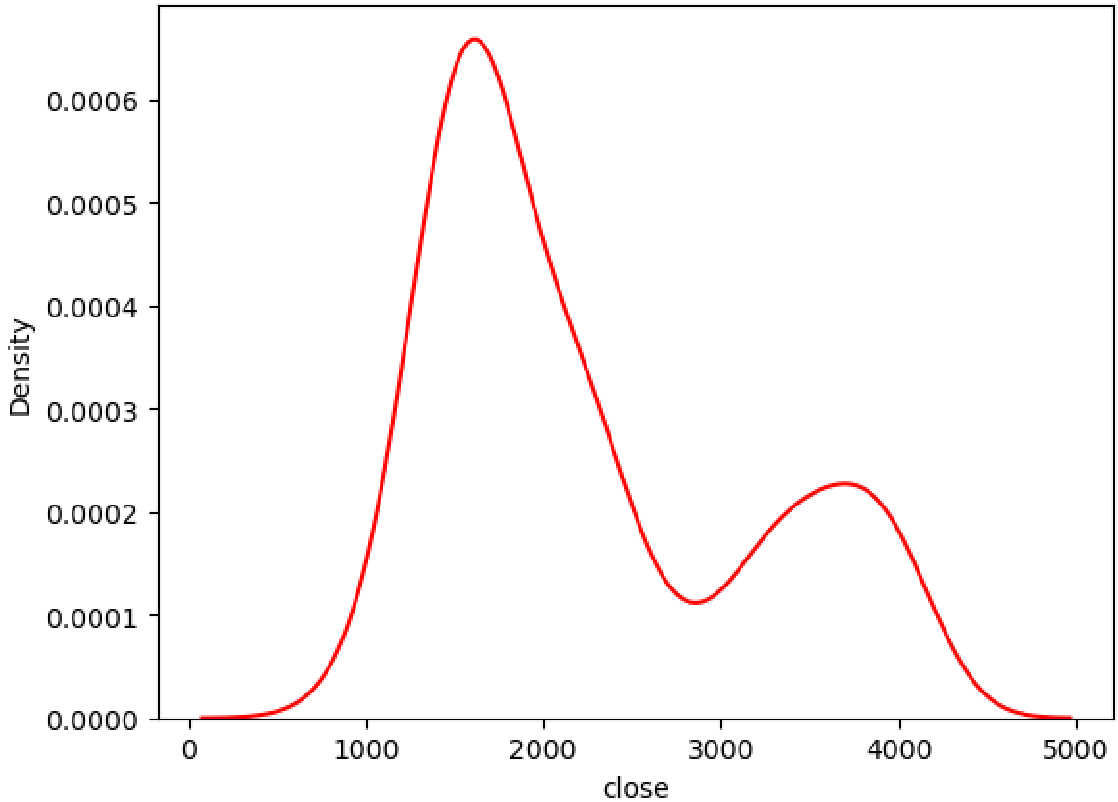


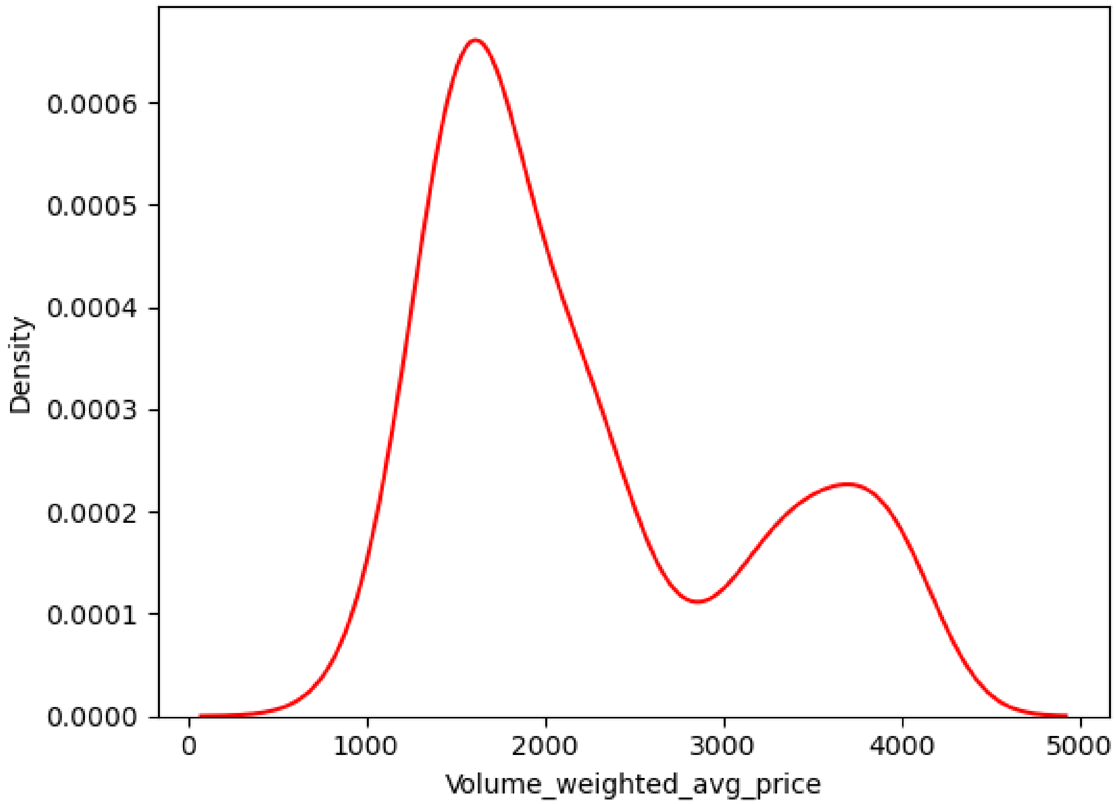



0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
Density
0
1000
2000
3000
4000
5000
LOW

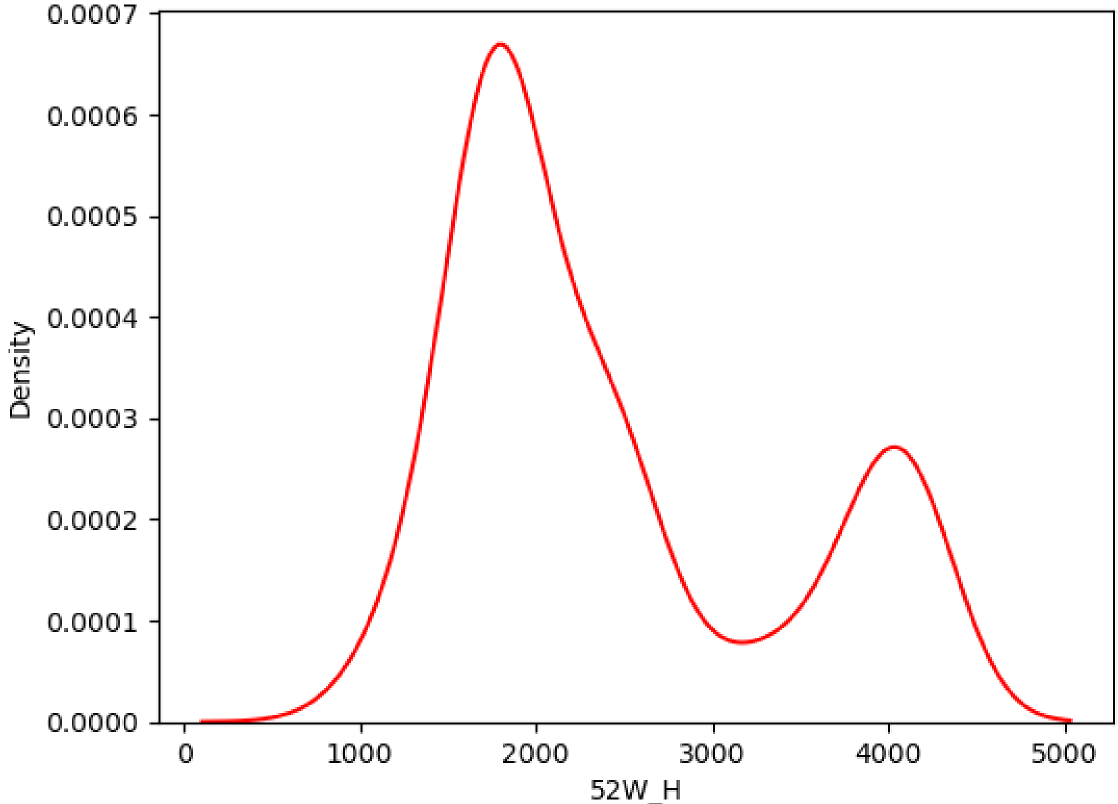

0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
Density
0
1000
2000
3000
4000
5000
PREV_CLOSE


Density
0.0000
0.0001
0.0002
0.0003
0.0004
0.0005
0.0006
0
1000
2000
3000
4000
5000
Last_Traded_Price

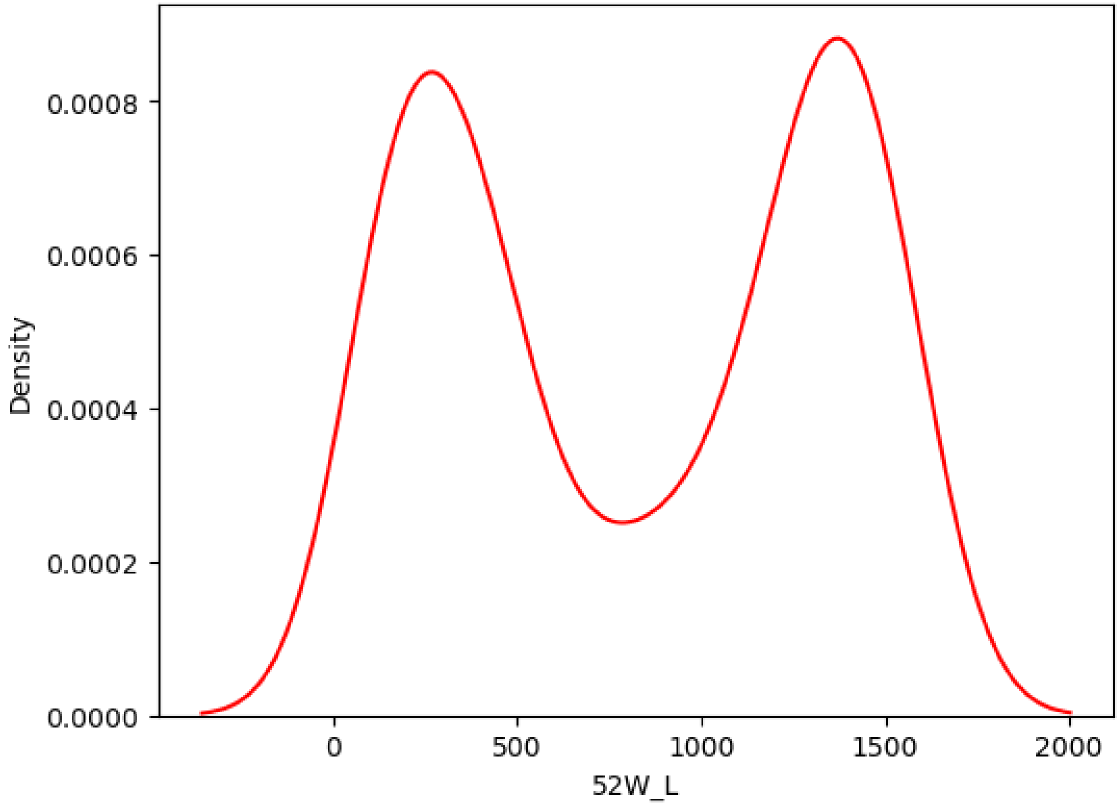

Density
0.0000
0.0001
0.0002
0.0003
0.0004
0.0005
0.0006
0
1000
2000
3000
4000
5000
close

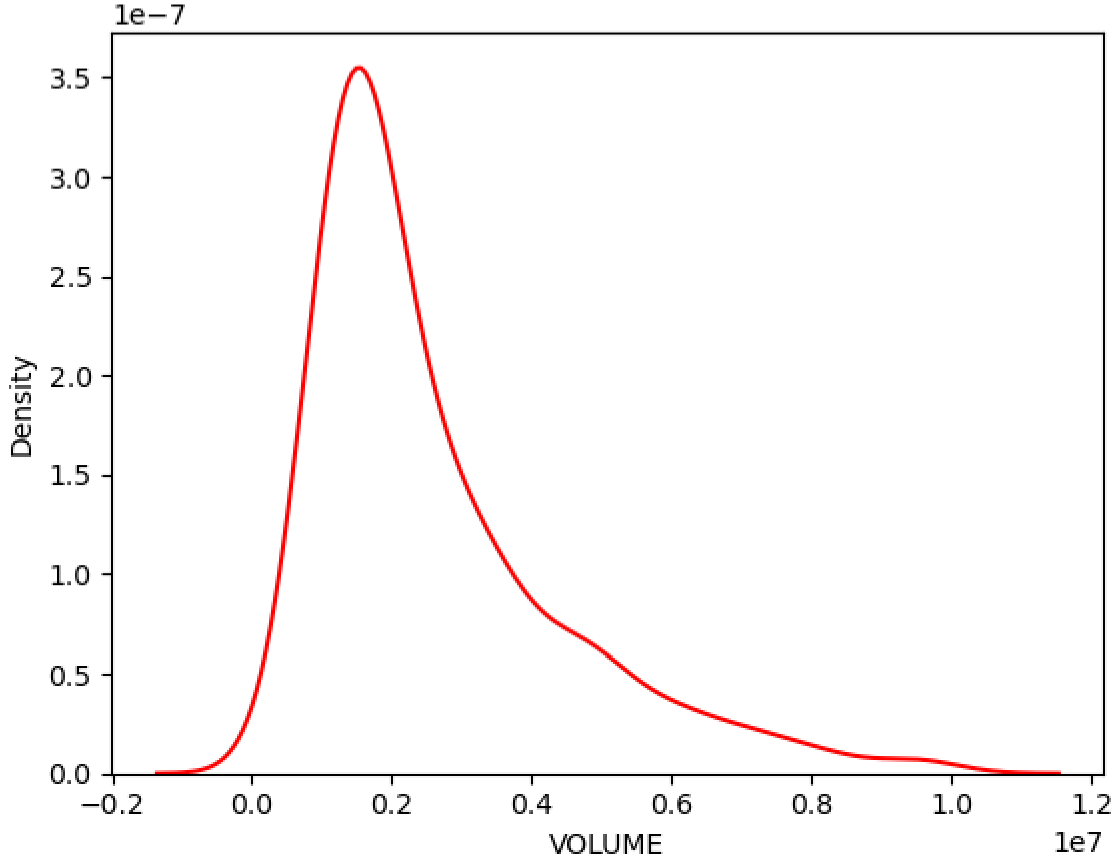


Density
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
0
1000
2000
3000
4000
5000
Volume_weighted_avg_price

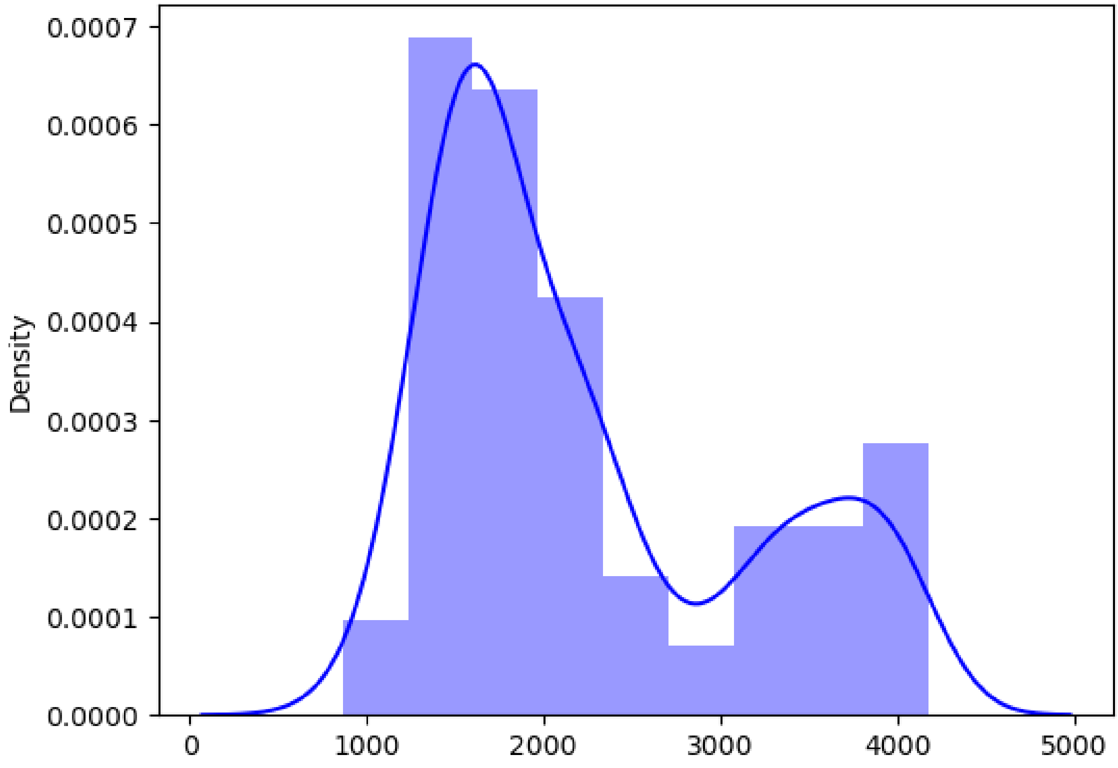


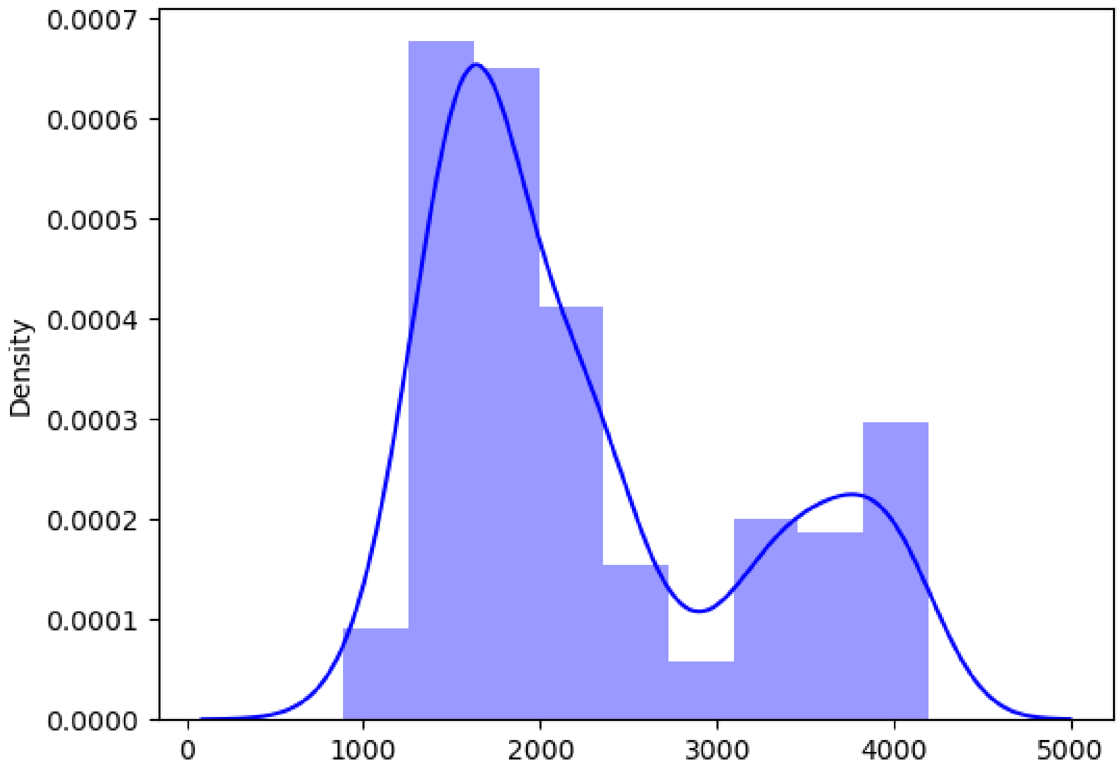
Density
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
0
1000
2000
3000
4000
5000
52W_H

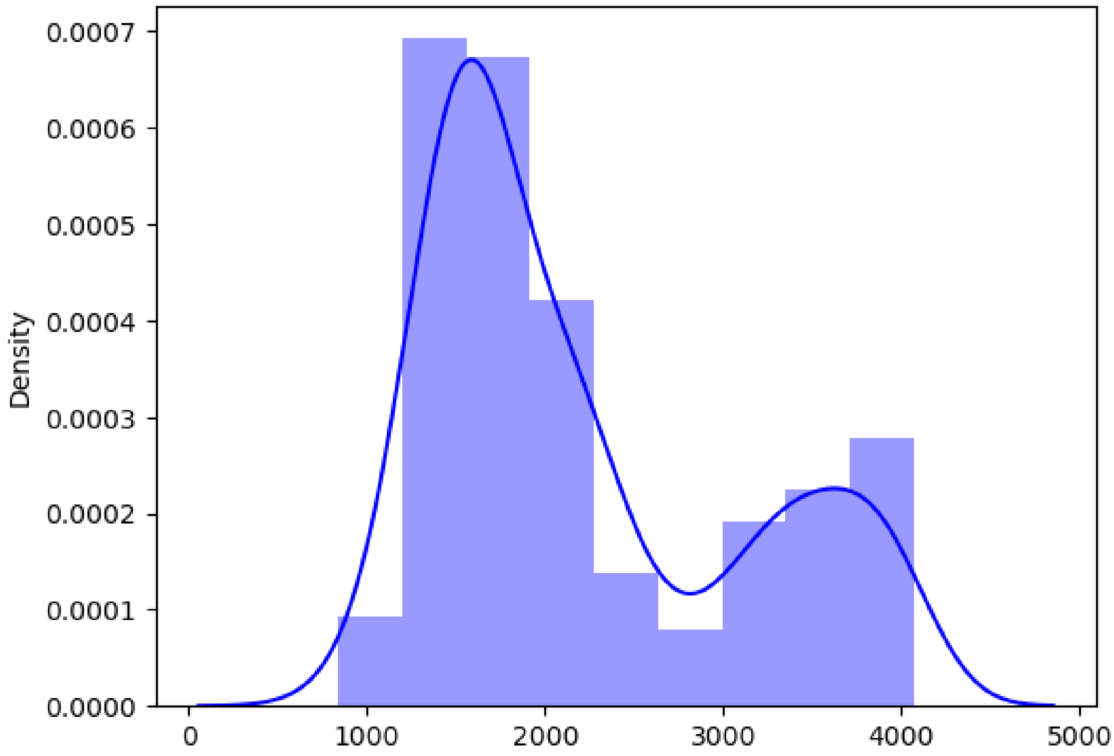

0.0008
0.0006
0.0004
0.0002
0.0000
Density
0
500
1000
1500
2000
52W_L


1e−7
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Density
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
VOLUME
1e7

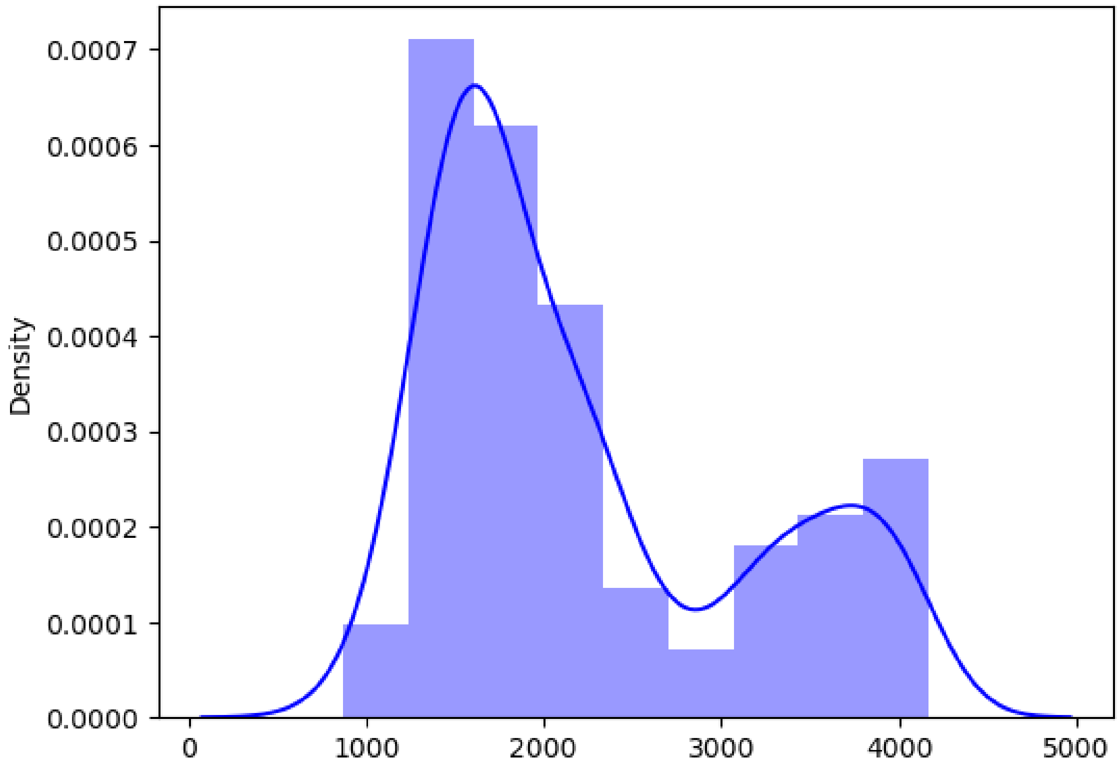

```
#Distplot
for i in df.columns:
    if df[i].dtypes != 'object":
        sns.distplot(x =df[i],color='blue' )
        plt.show()
```



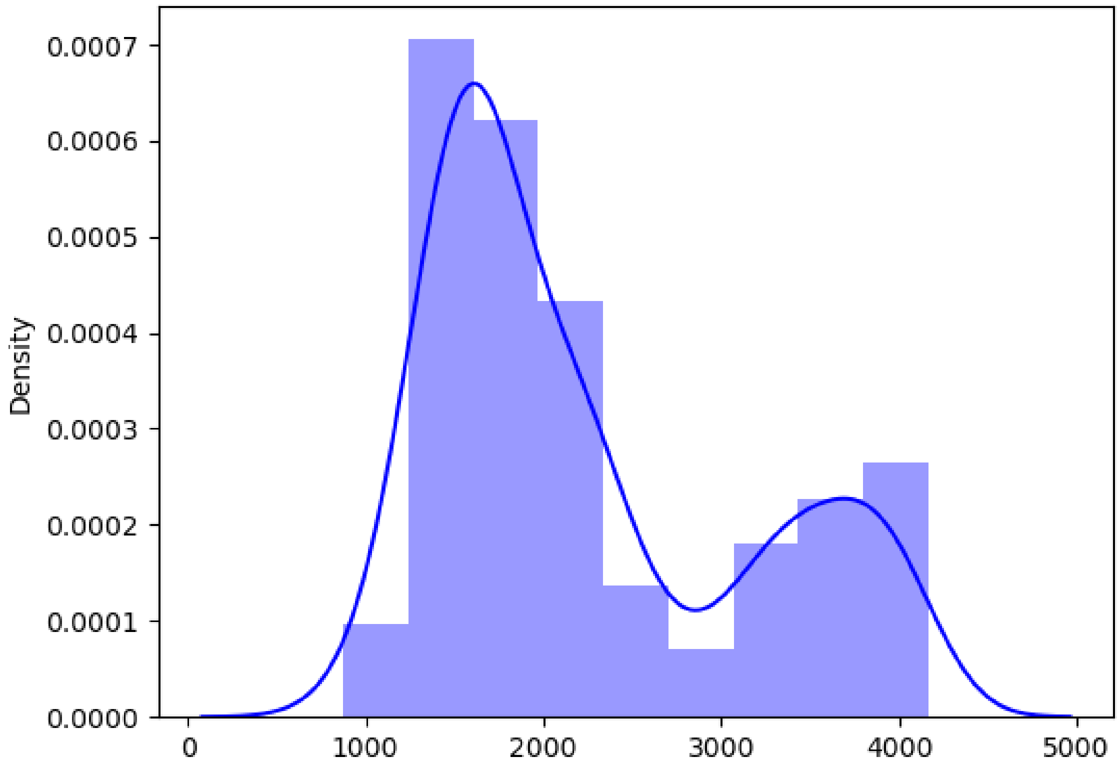


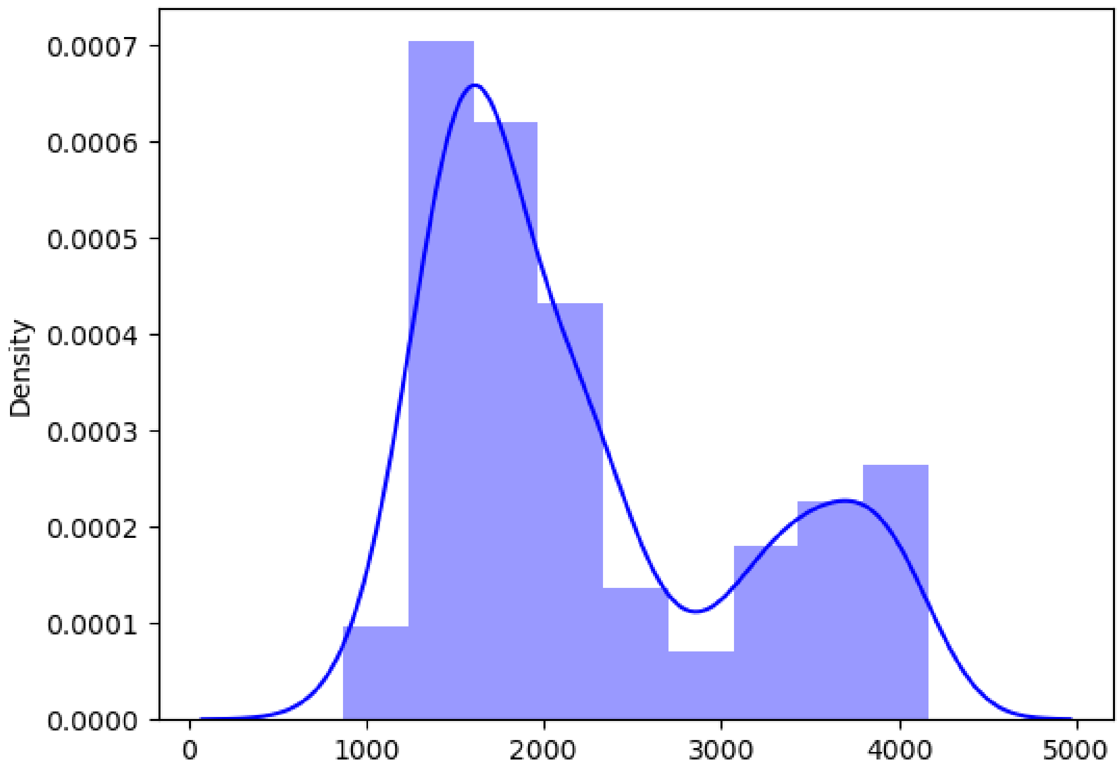


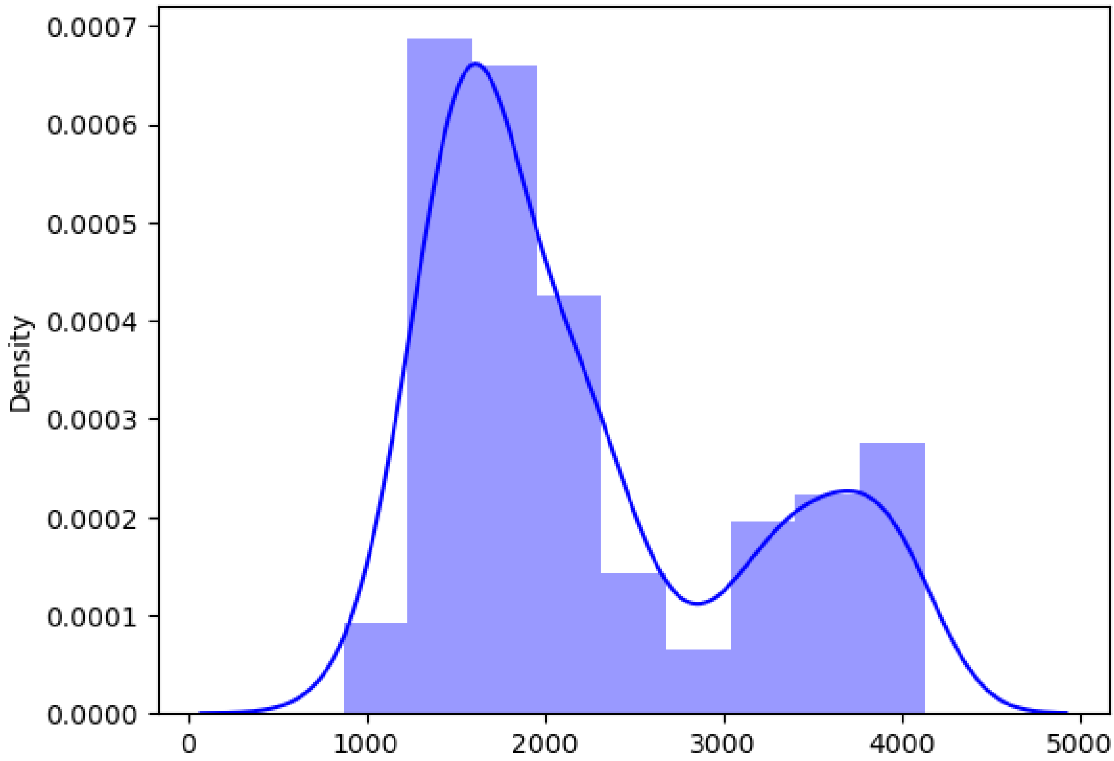
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
Density
0
1000
2000
3000
4000
5000

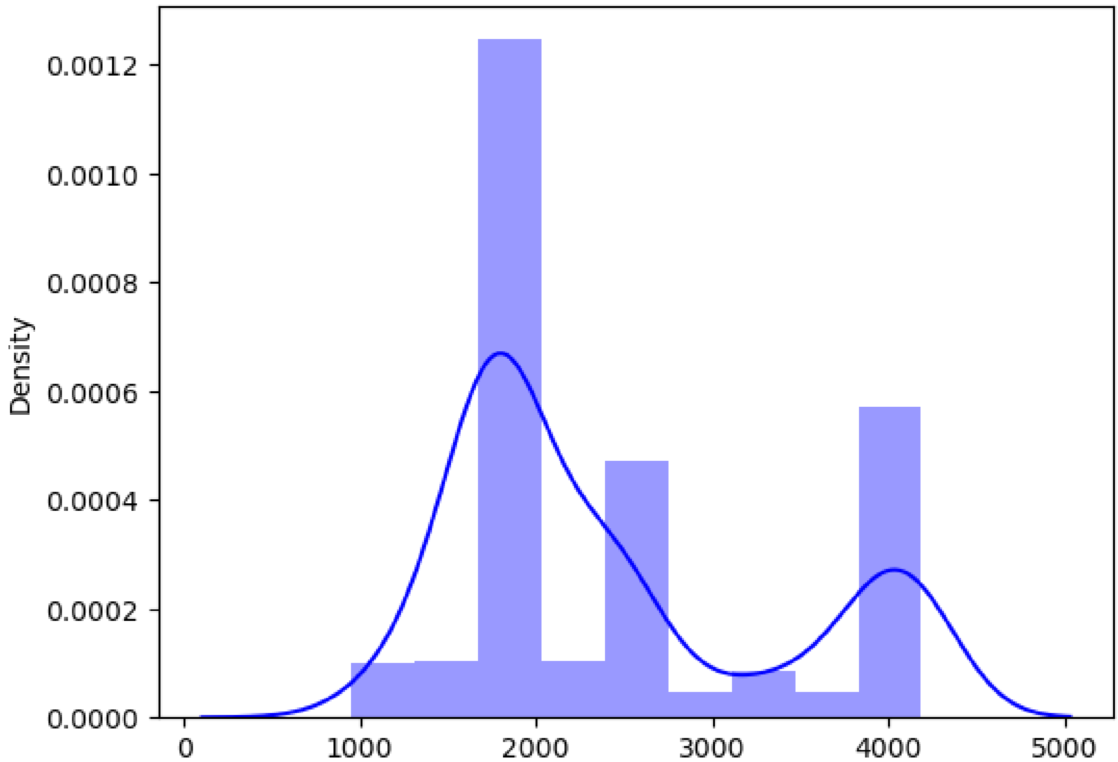


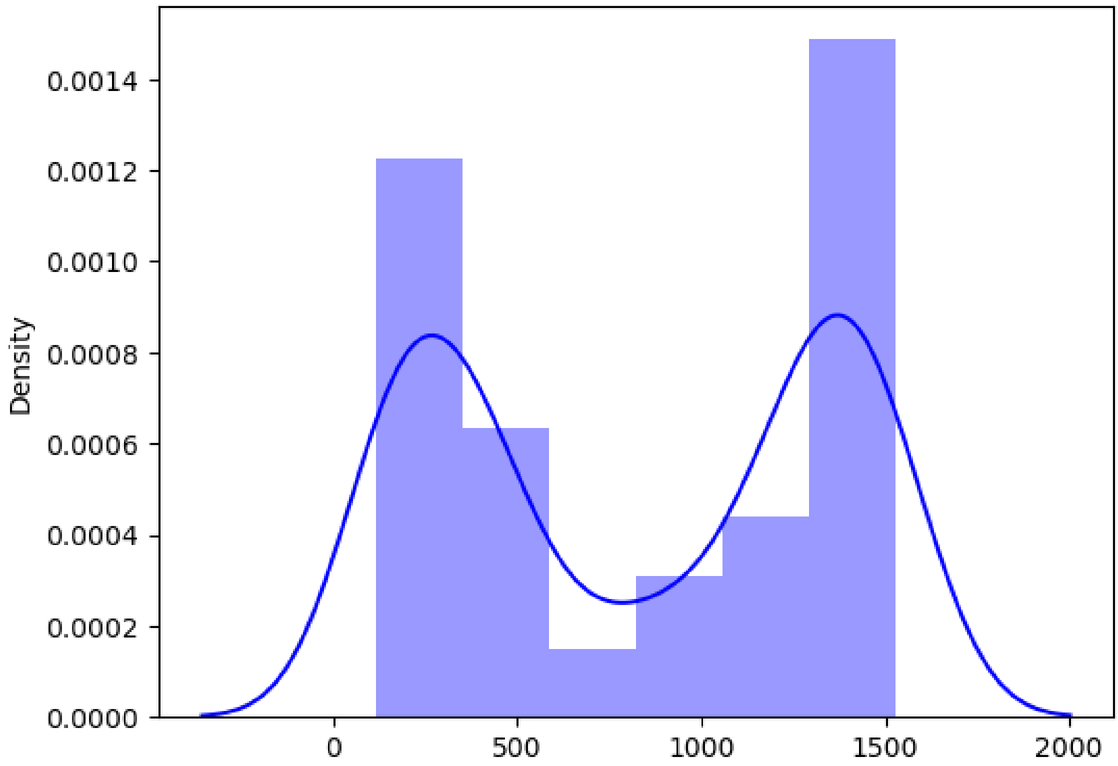
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
Density
0
1000
2000
3000
4000
5000

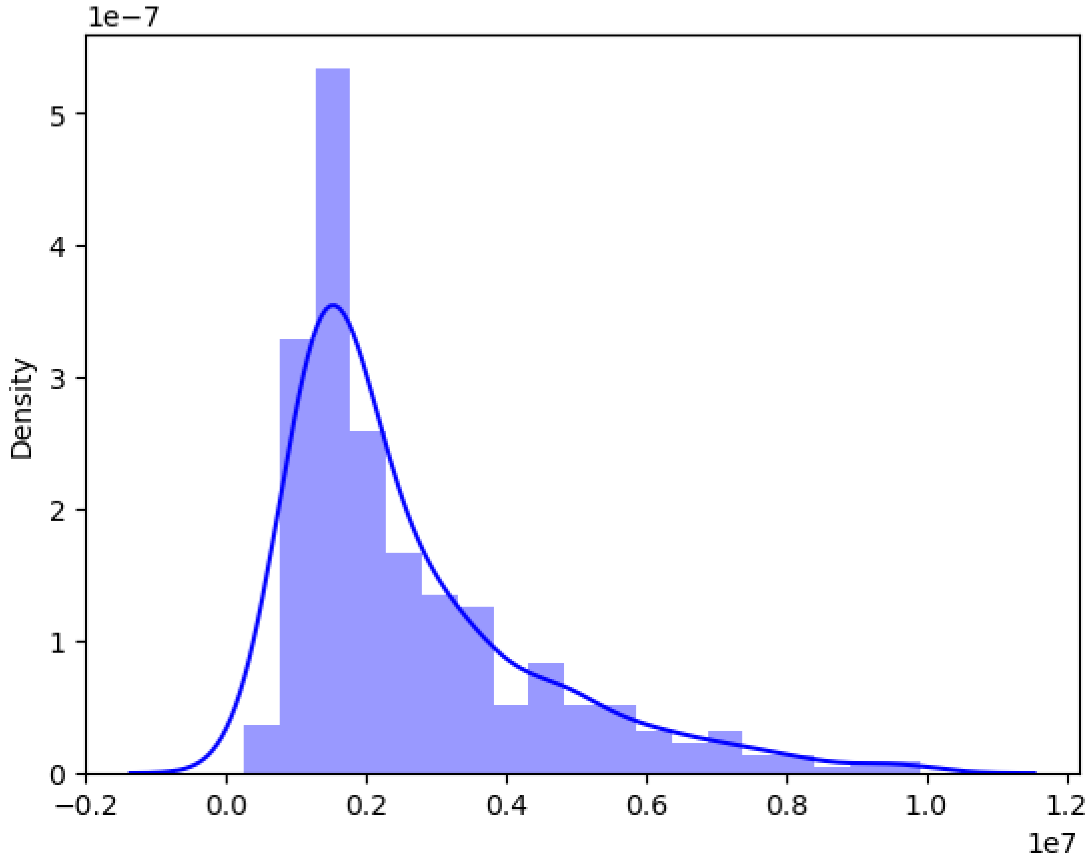

Density
0.0000
0.0001
0.0002
0.0003
0.0004
0.0005
0.0006
0.0007
0
1000
2000
3000
4000
5000


Density
0.0000
0.0001
0.0002
0.0003
0.0004
0.0005
0.0006
0.0007
0
1000
2000
3000
4000
5000

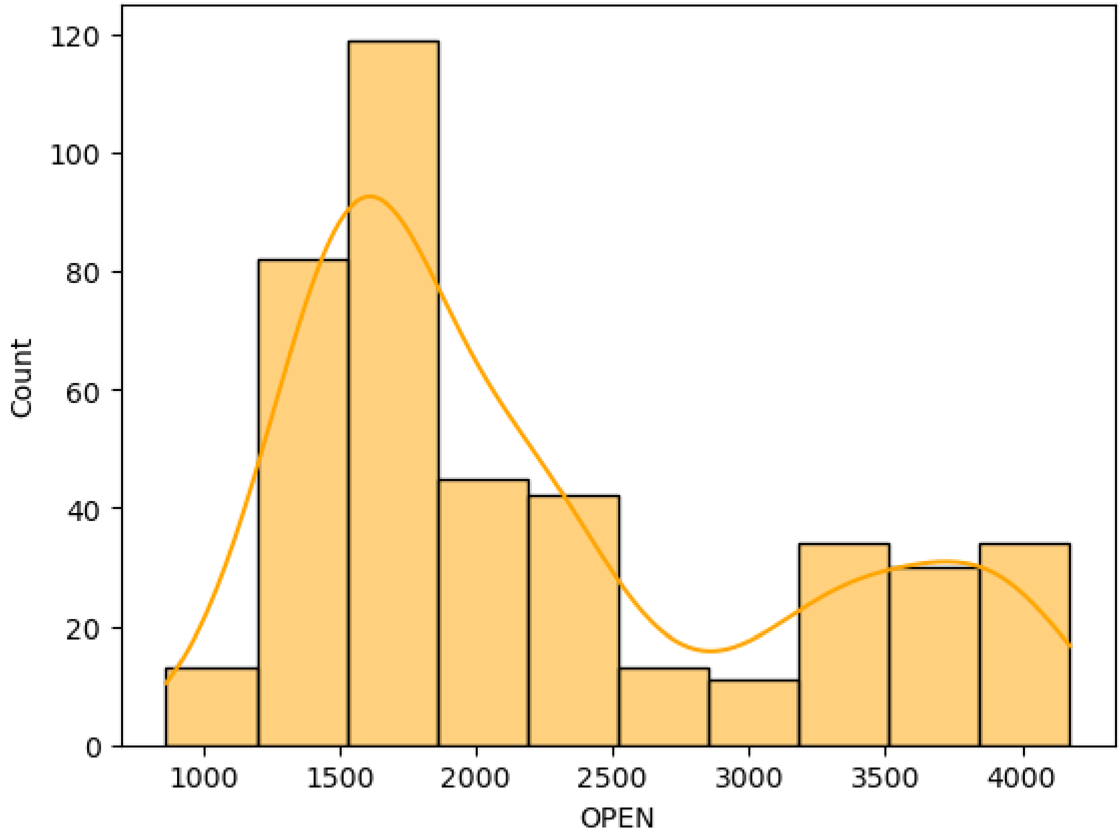

0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
Density
0
1000
2000
3000
4000
5000

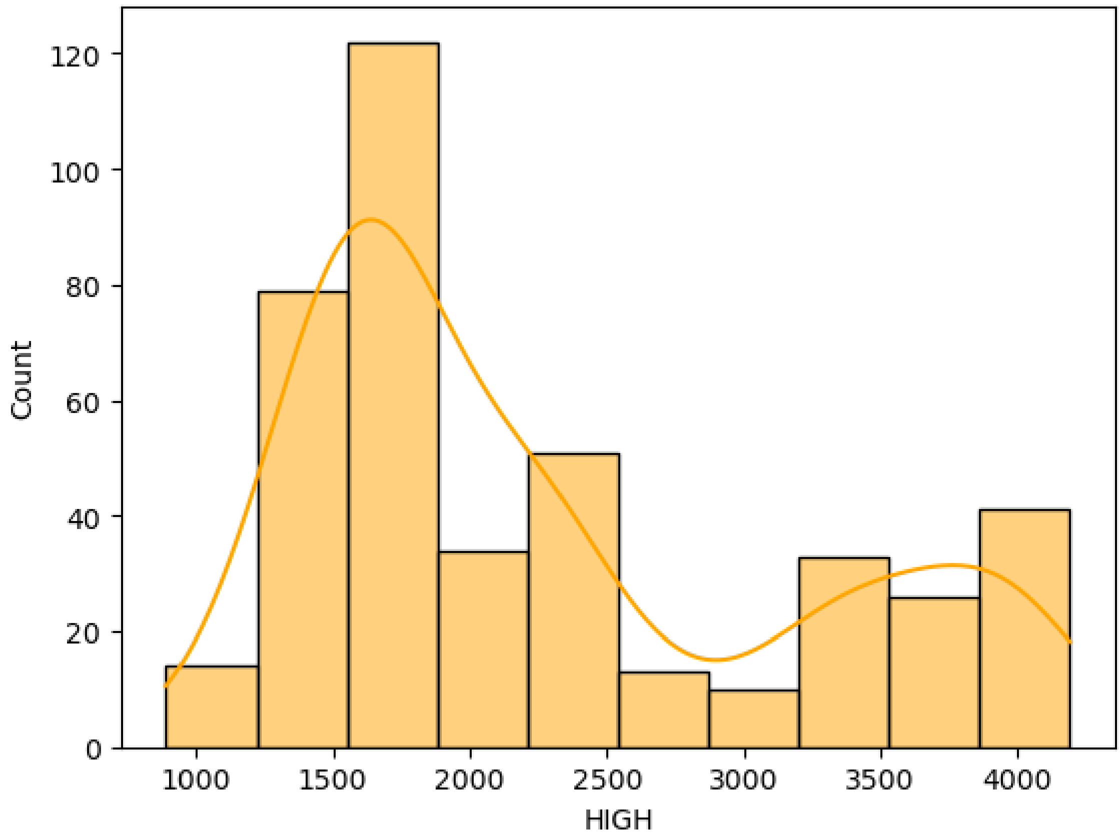

0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
Density
0
1000
2000
3000
4000
5000

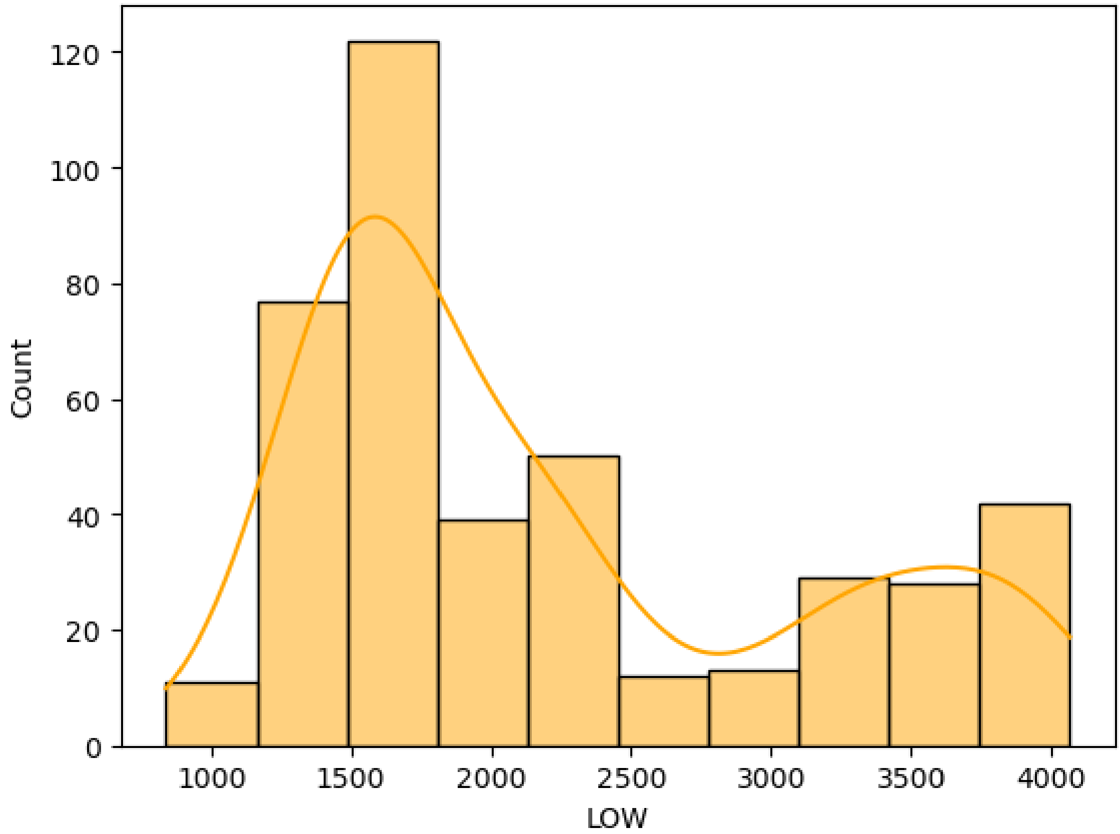

0.0014
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
Density
0
500
1000
1500
2000


1e−7
5
4
3
2
1
0
Density
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1e7

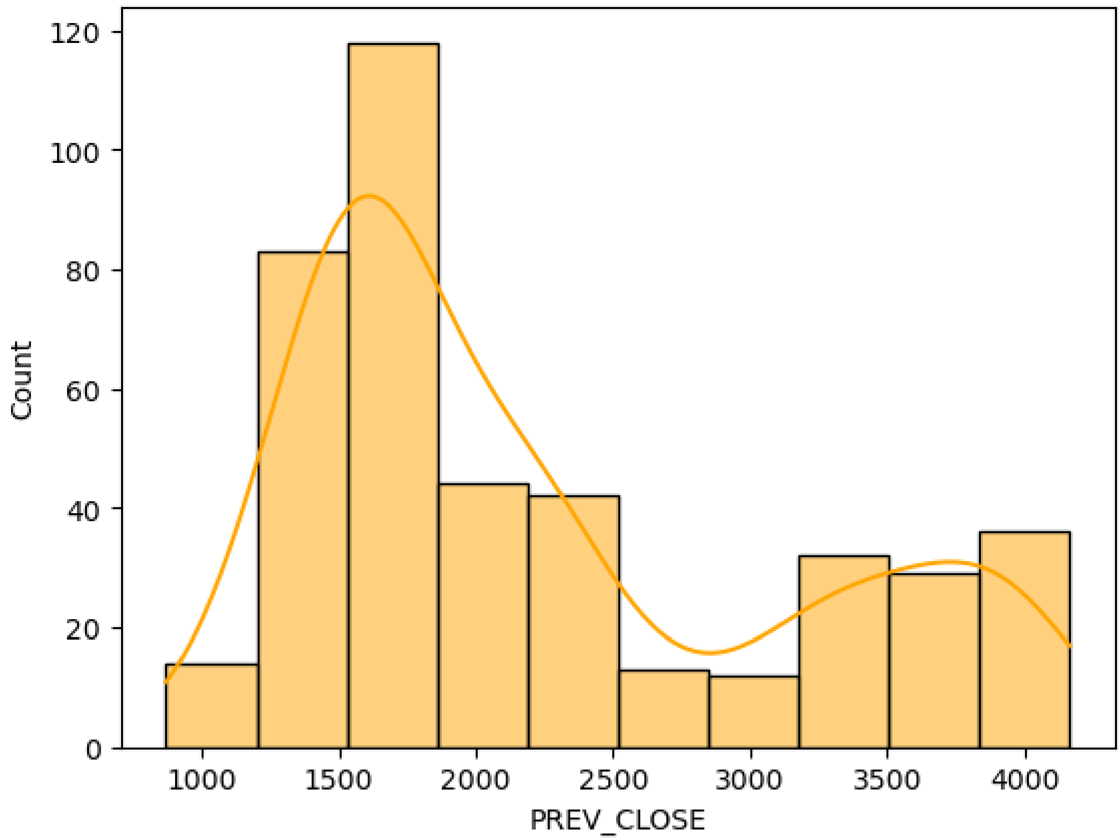

```
#Histplot
for column in df.columns:
    if df[column].dtypes !='object':
        sns.histplot(x=df[column],kde=True,color='orange')
        plt.show()
```



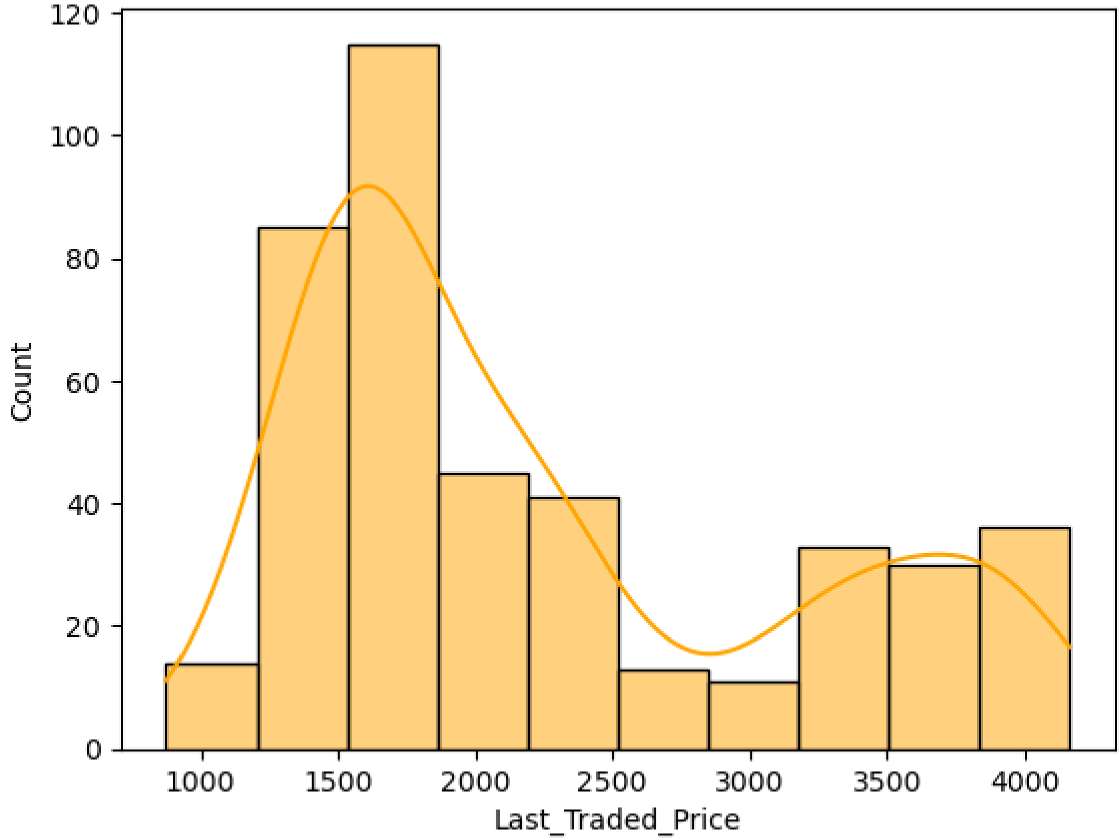


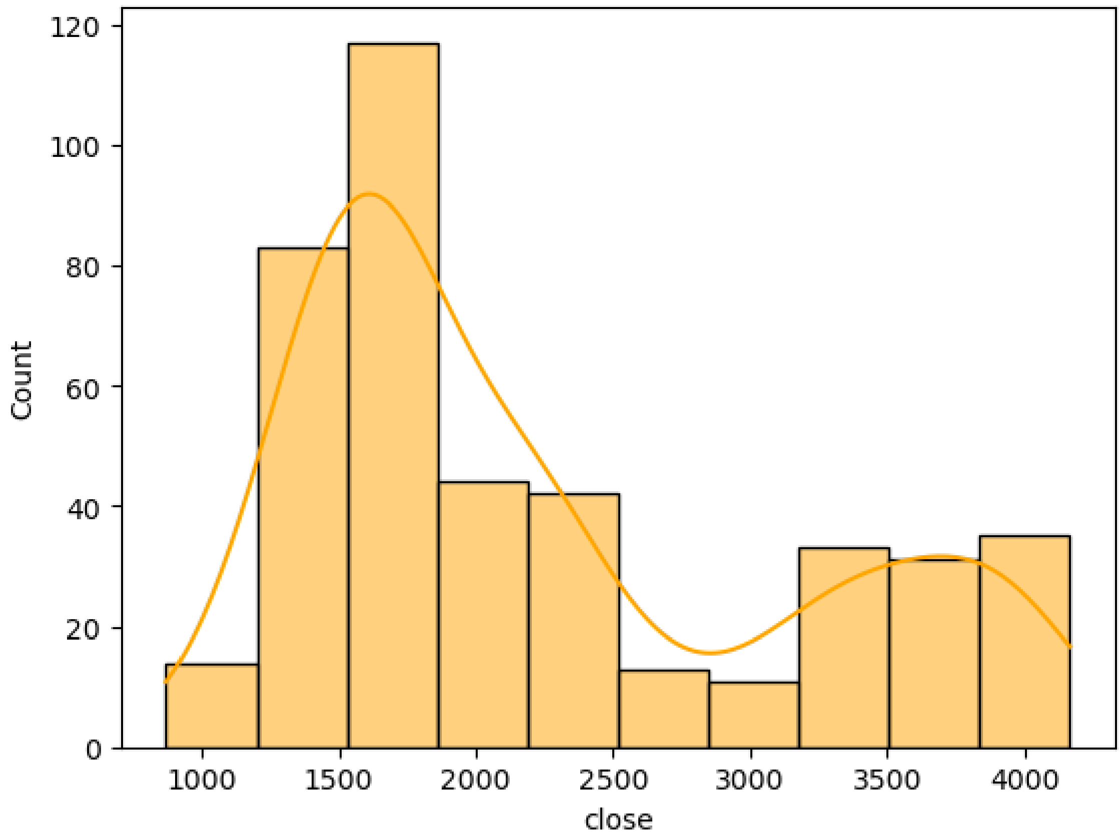


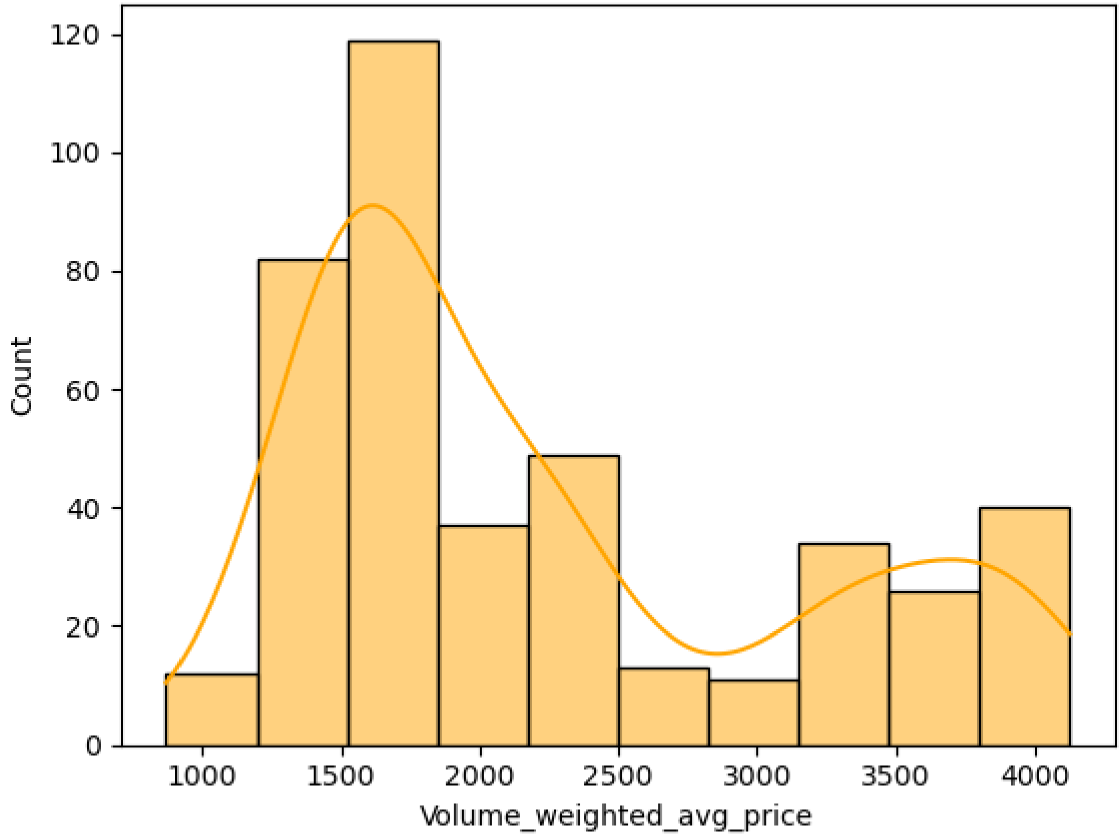
120
100
80
60
40
20
0
Count
1000
1500
2000
2500
3000
3500
4000
LOW

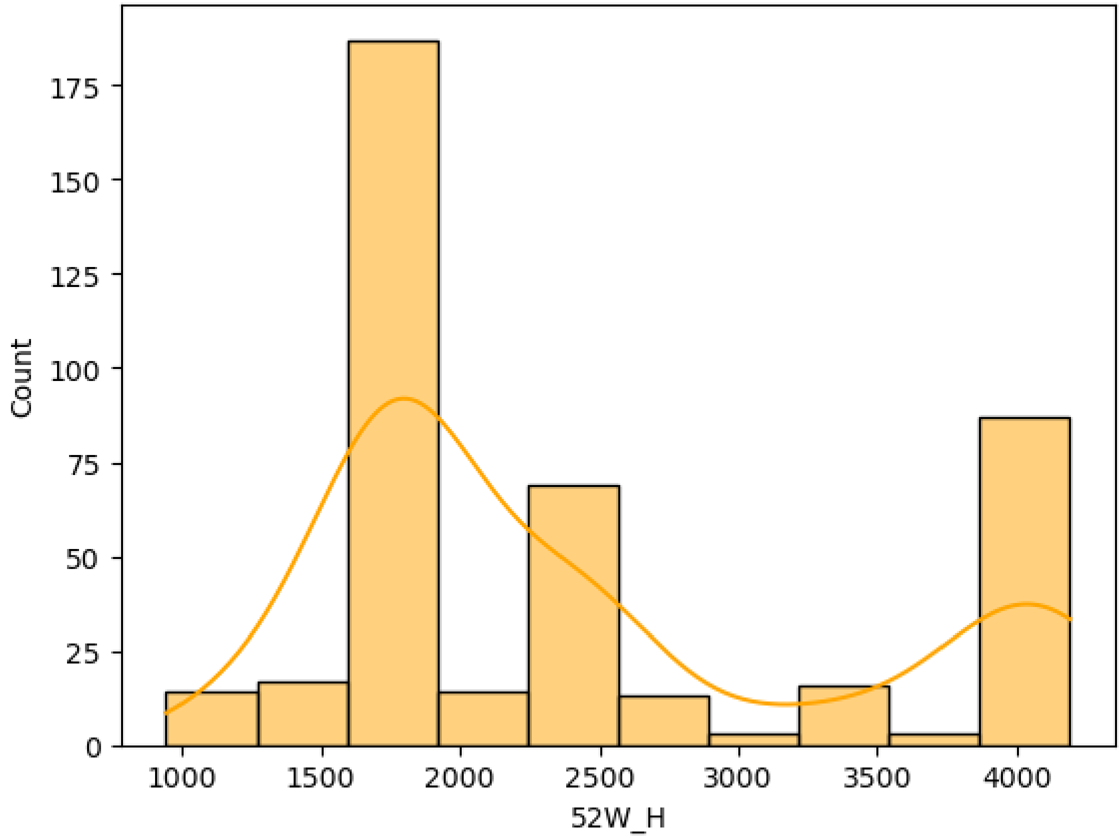


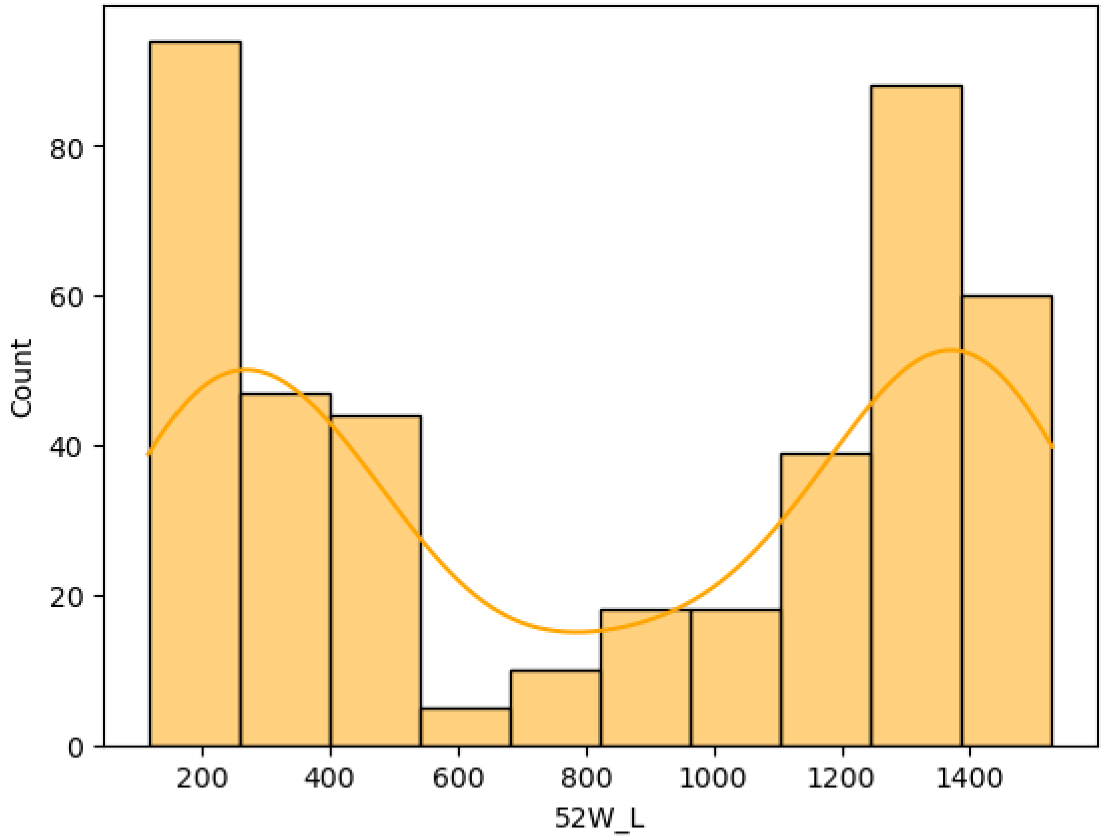
120
100
80
60
40
20
0
Count
1000
1500
2000
2500
3000
3500
4000
PREV_CLOSE


Count
Last_Traded_Price

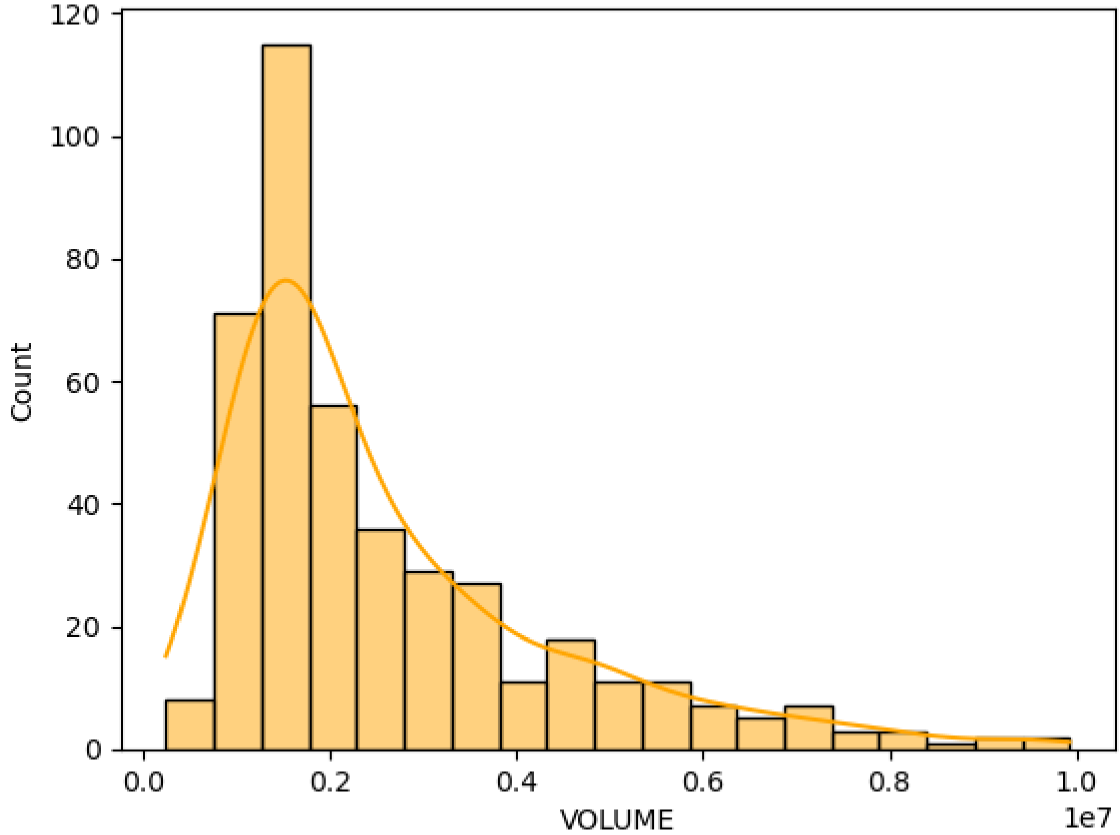

Count
close


Count
0
20
40
60
80
100
120
1000
1500
2000
2500
3000
3500
4000
Volume_weighted_avg_price


Count
0
25
50
75
100
125
150
175
1000
1500
2000
2500
3000
3500
4000
52W_H

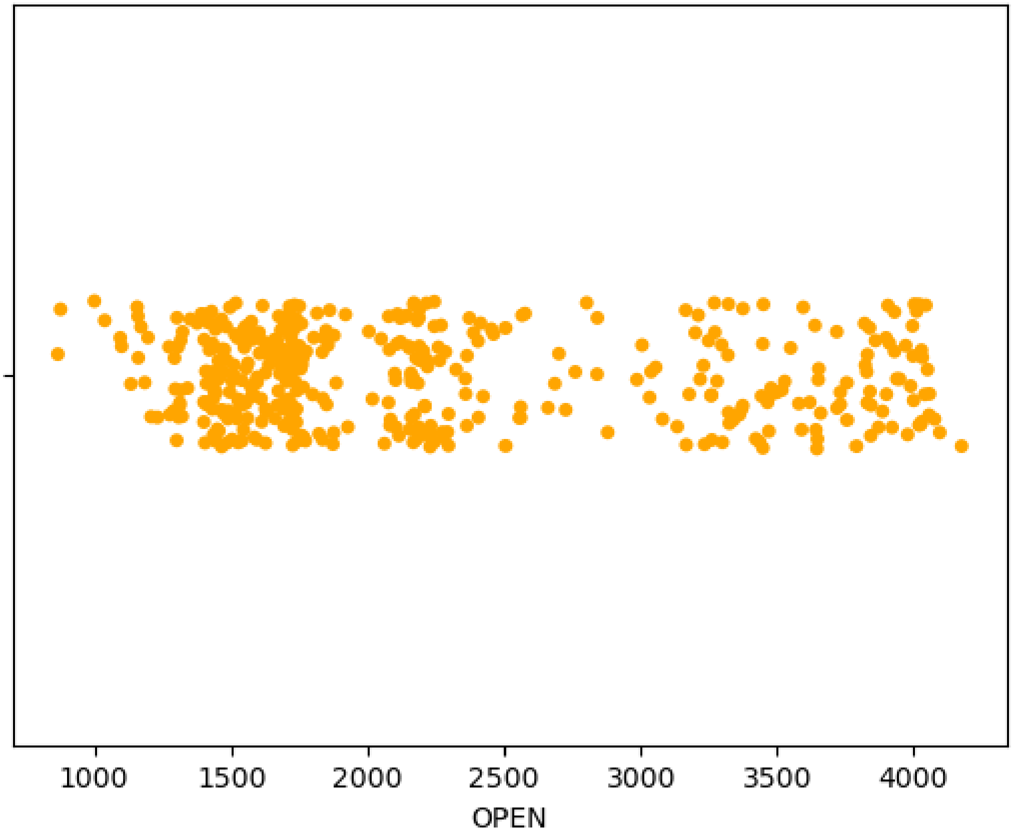


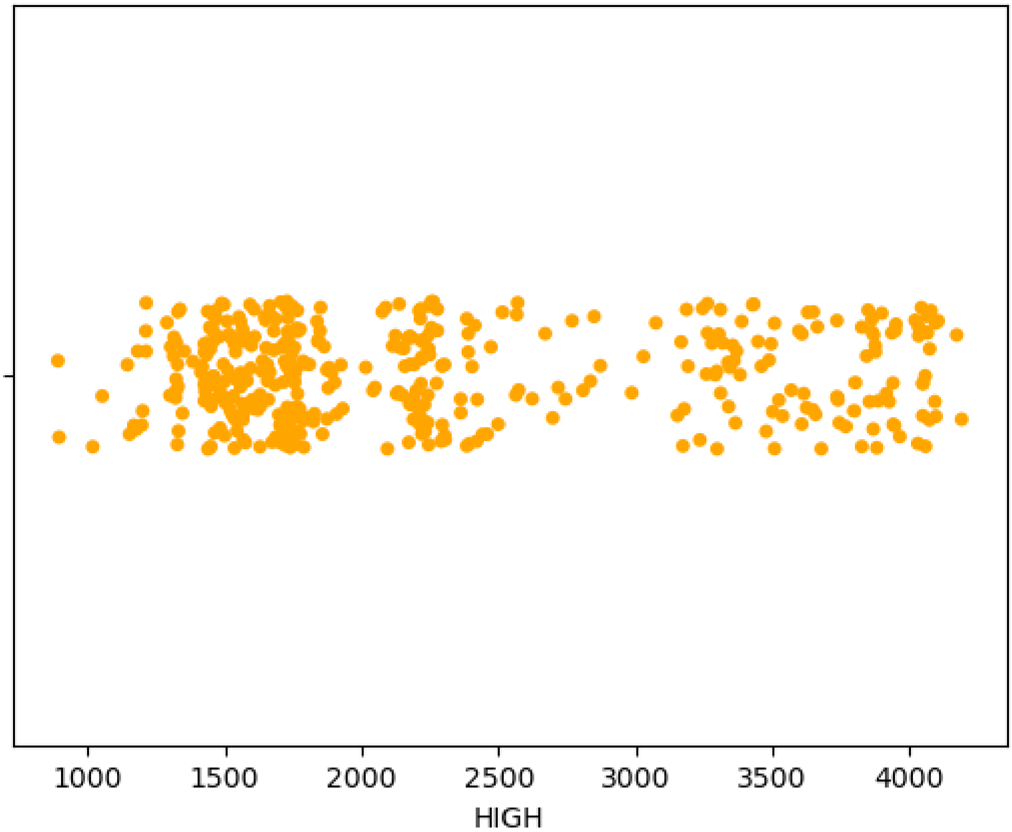
Count
0
20
40
60
80
200
400
600
800
1000
1200
1400
52W_L

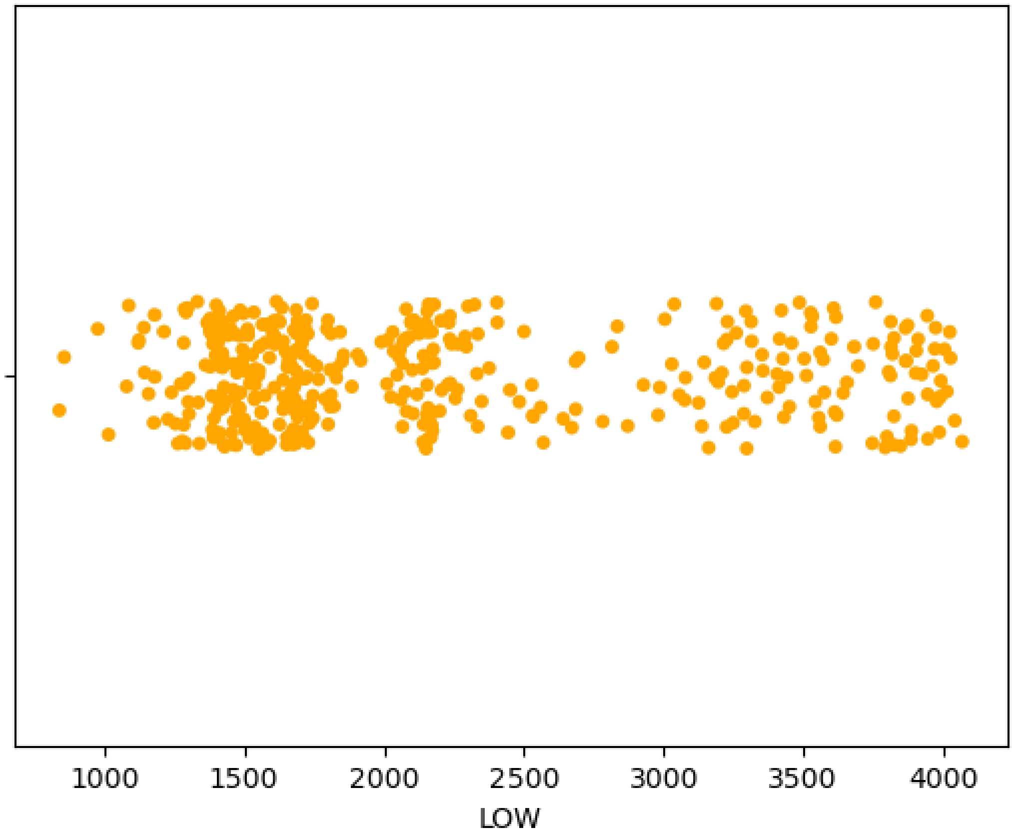


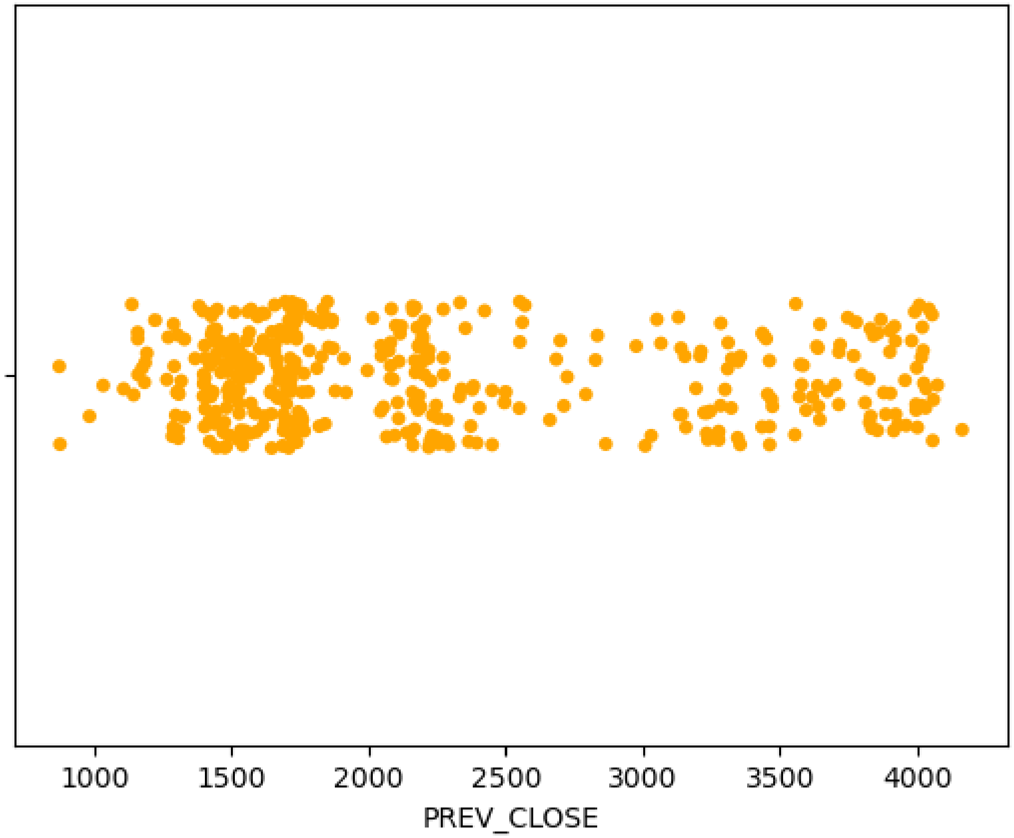
Count
0
20
40
60
80
100
120
0.0
0.2
0.4
0.6
0.8
1.0
VOLUME
1e7

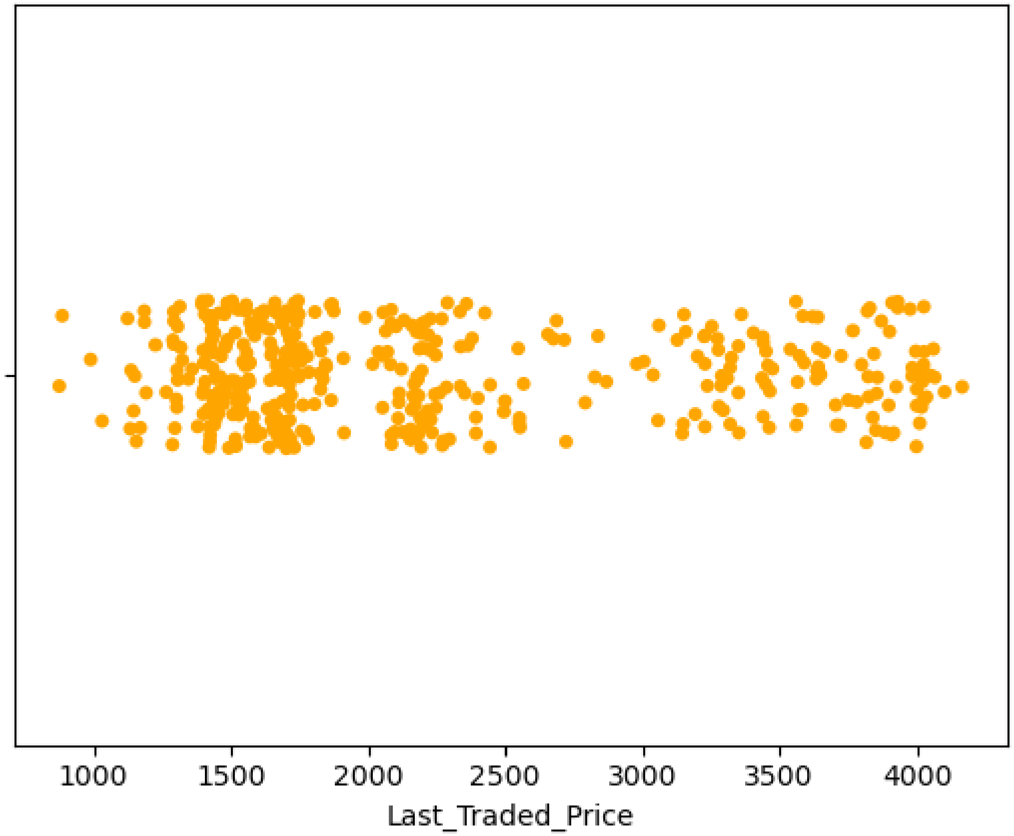
```
#Striplot
for i in df.columns:
    if df[i].dtypes != "object":
        sns.stripplot(x =df[i],color='orange' )
        plt.show()
```

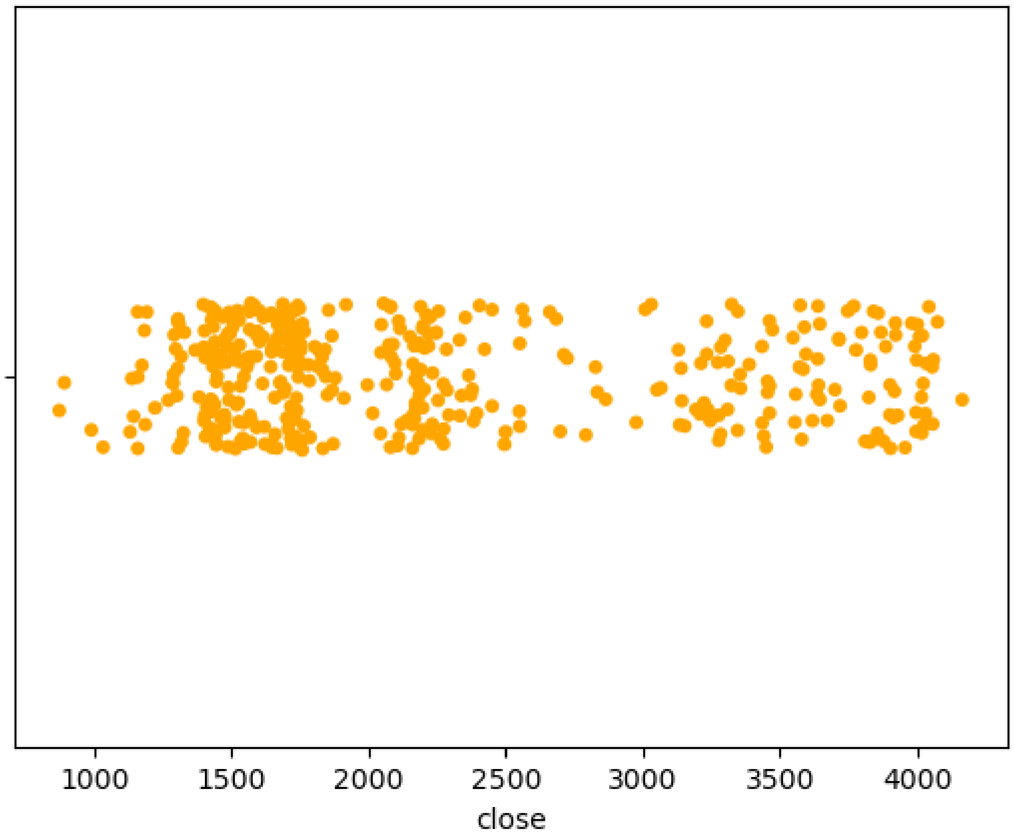


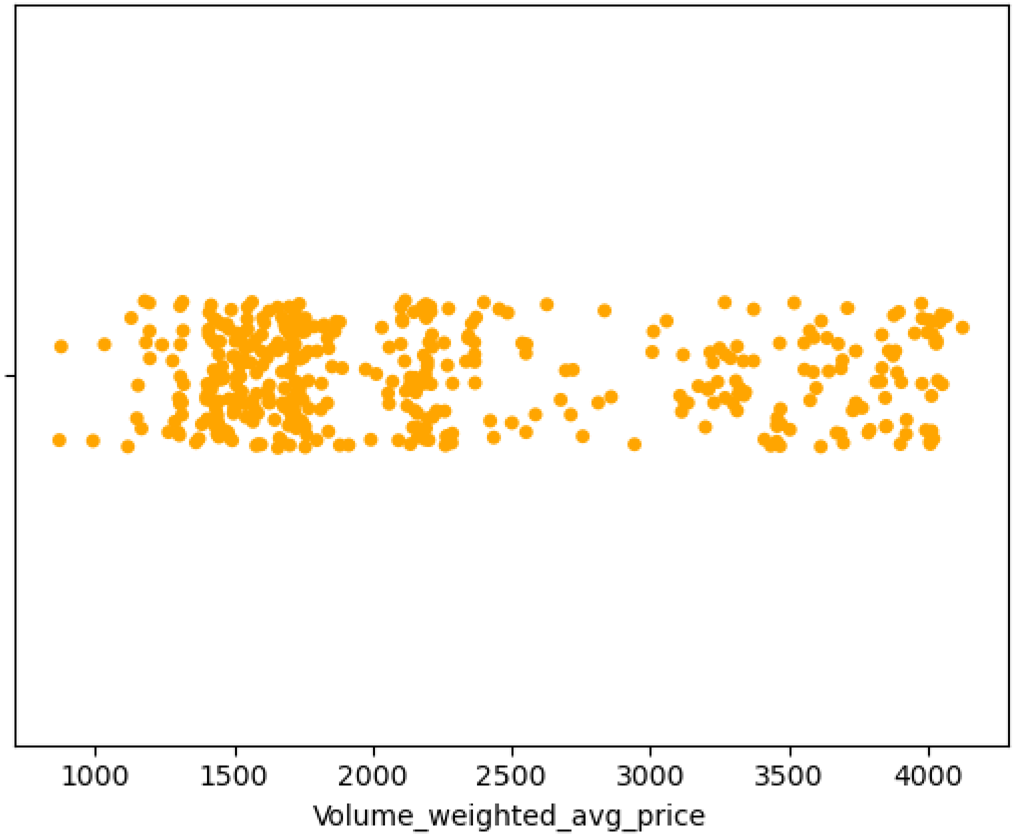


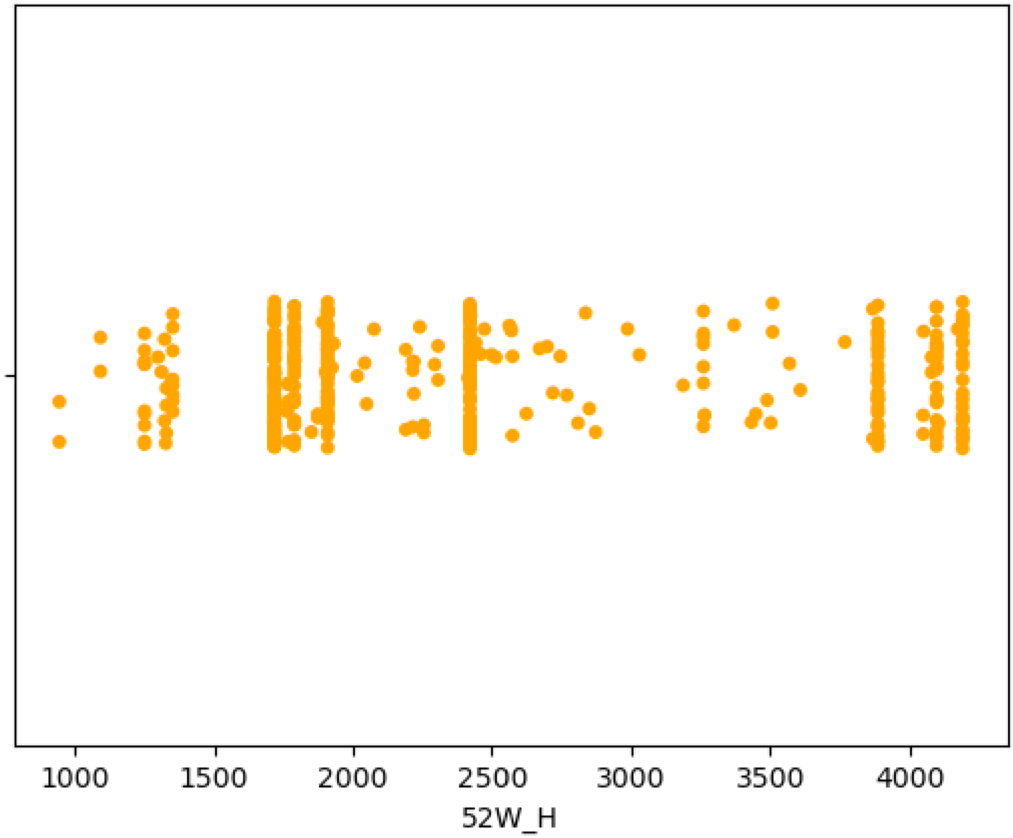

1000
1500
2000
2500
3000
3500
4000
LOW

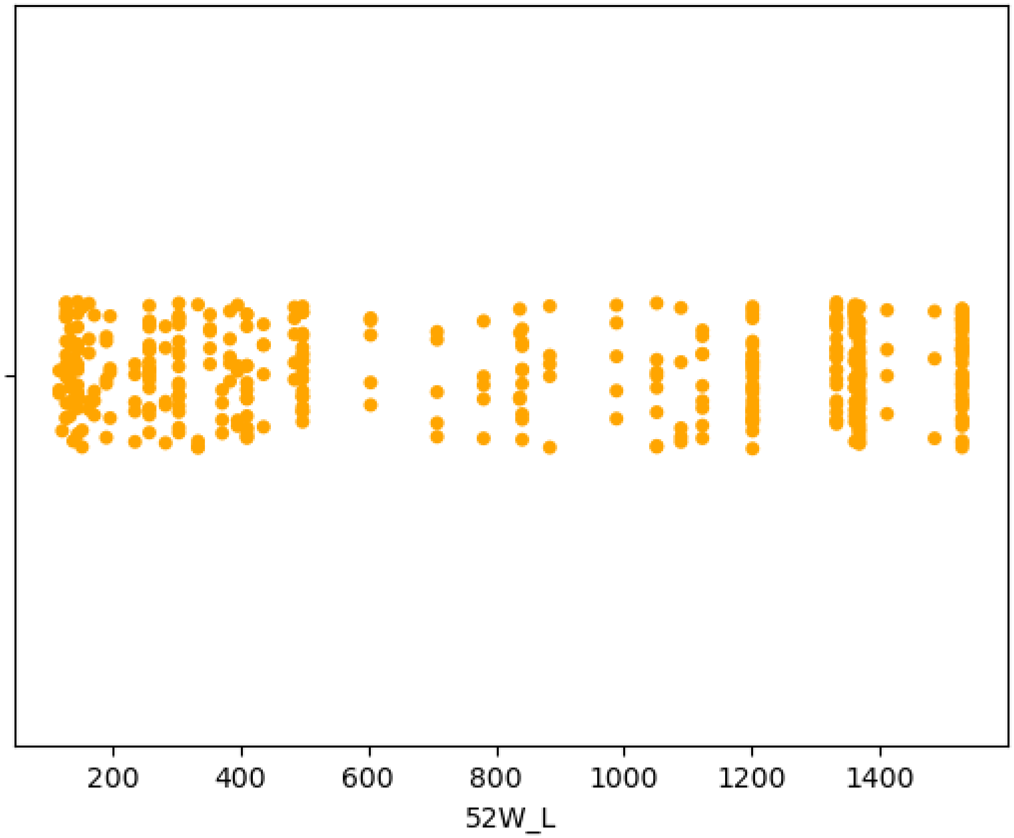

1000
1500
2000
2500
3000
3500
4000
PREV_CLOSE

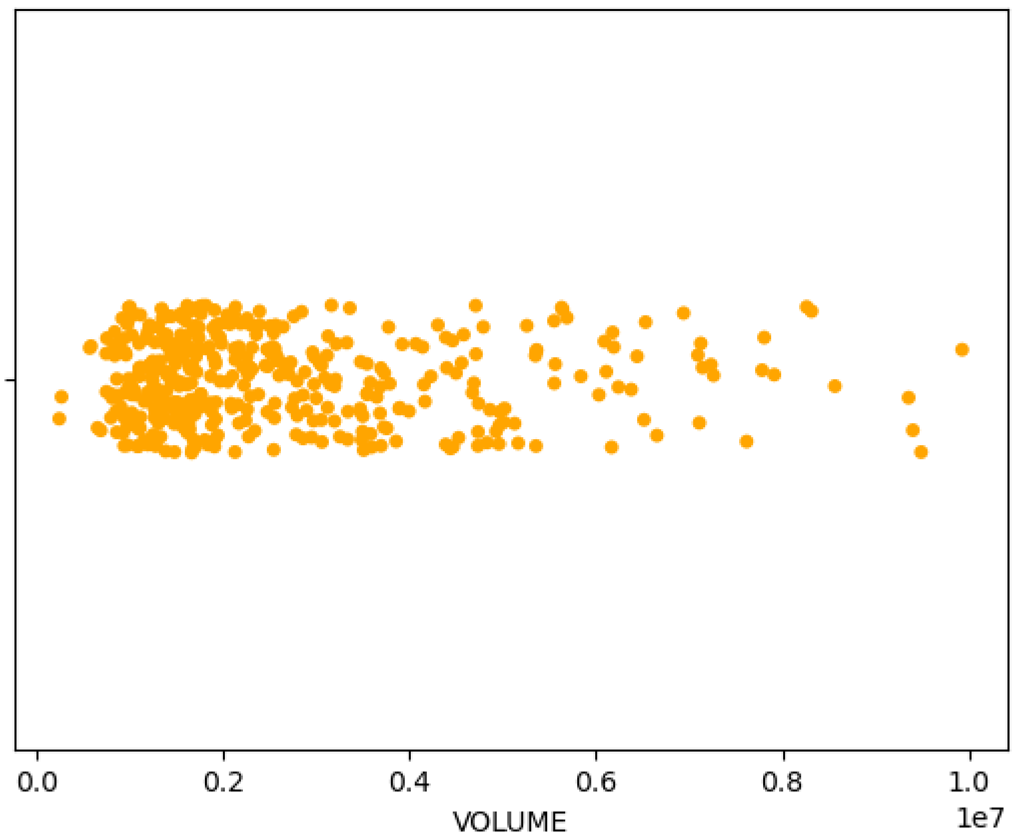

1000
1500
2000
2500
3000
3500
4000
Last_Traded_Price


1000
1500
2000
2500
3000
3500
4000
close

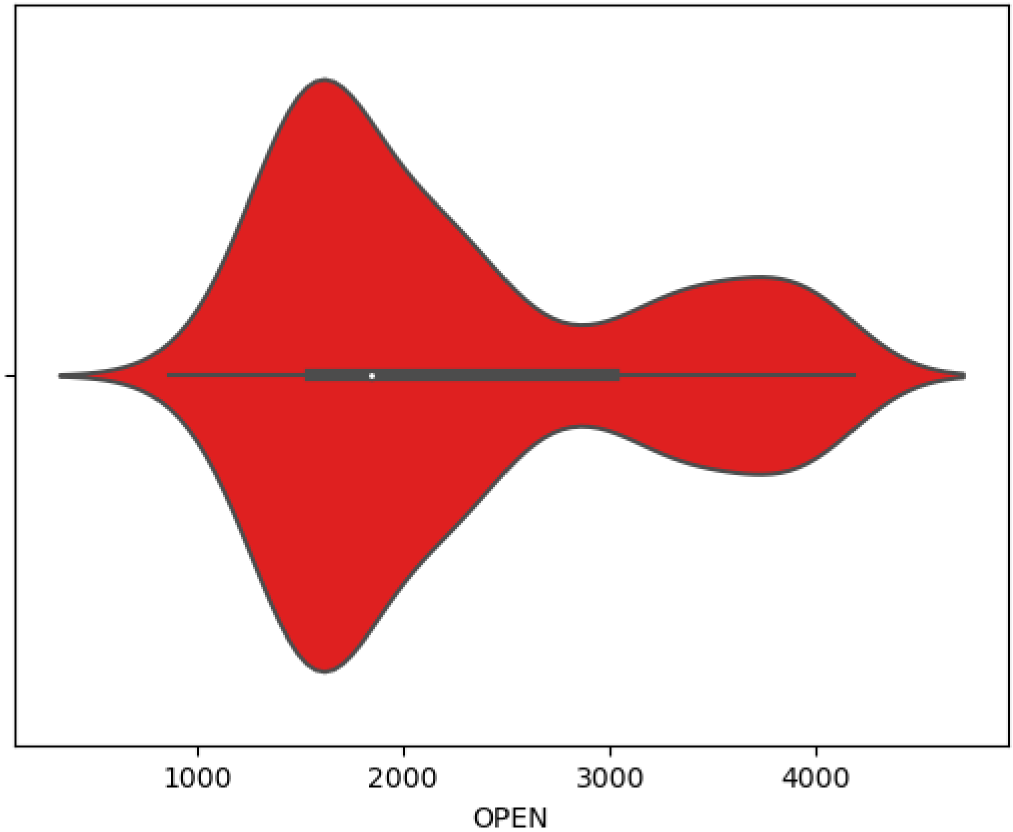

1000
1500
2000
2500
3000
3500
4000
Volume_weighted_avg_price

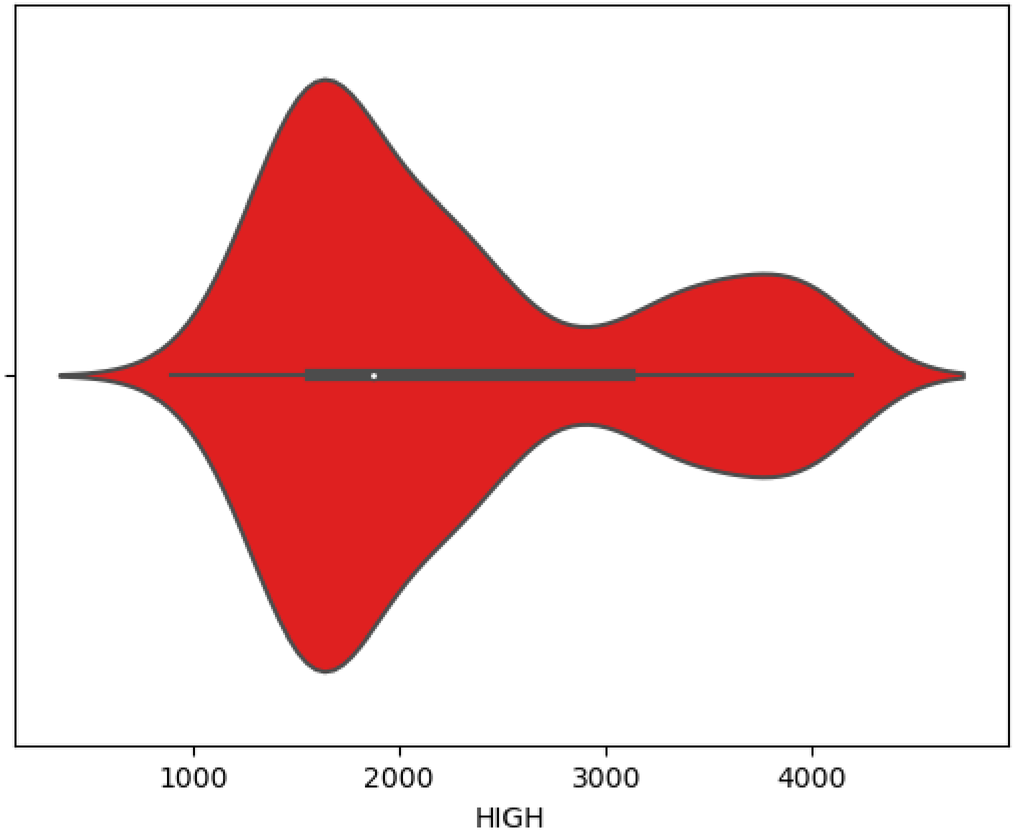

1000
1500
2000
2500
3000
3500
4000
52W_H

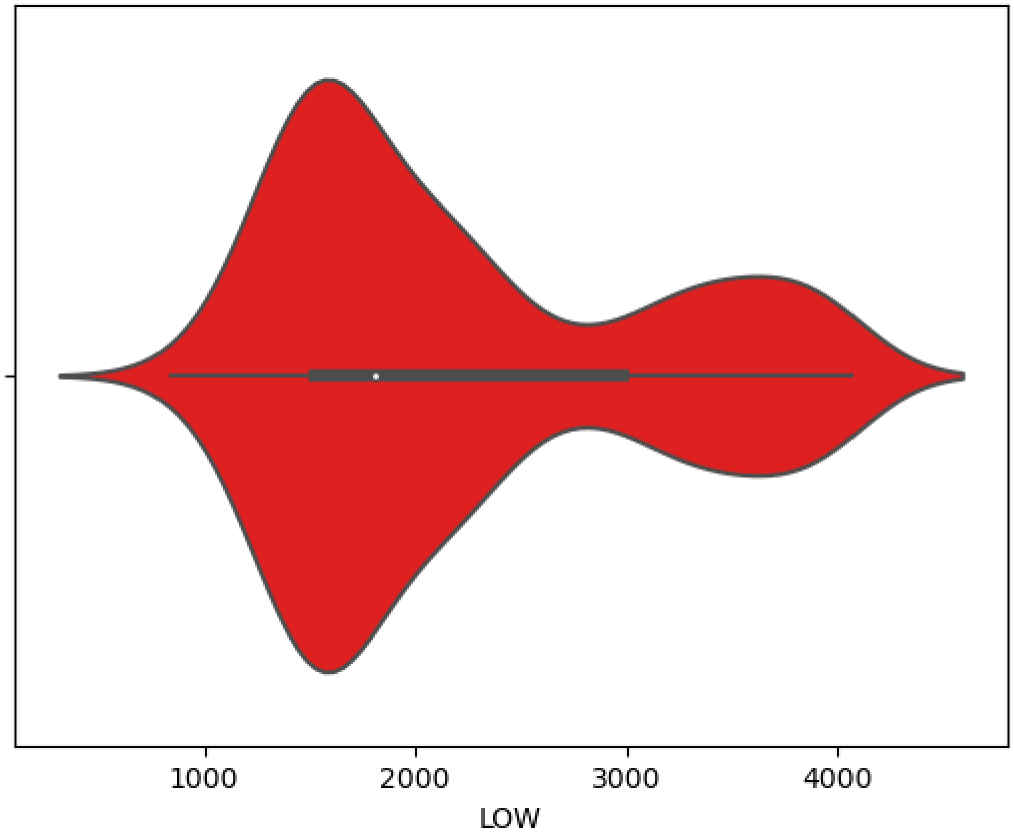

200
400
600
800
1000
1200
1400
52W_L

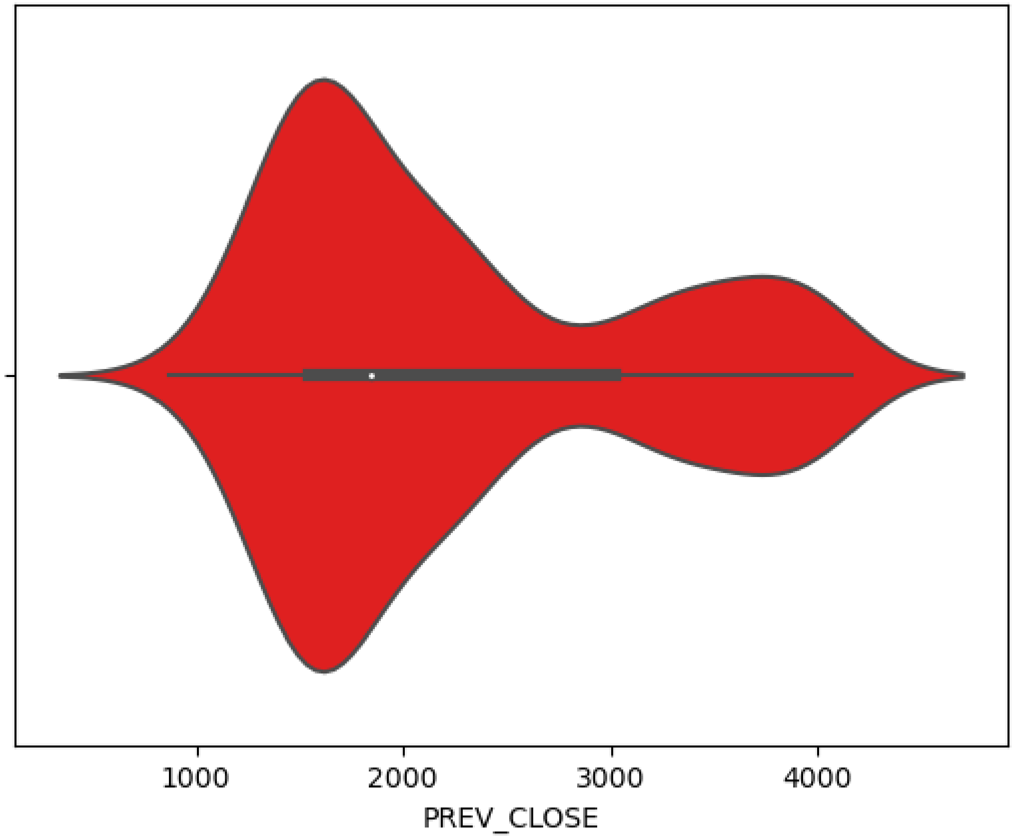

0.0
0.2
0.4
0.6
0.8
1.0
1e7
VOLUME

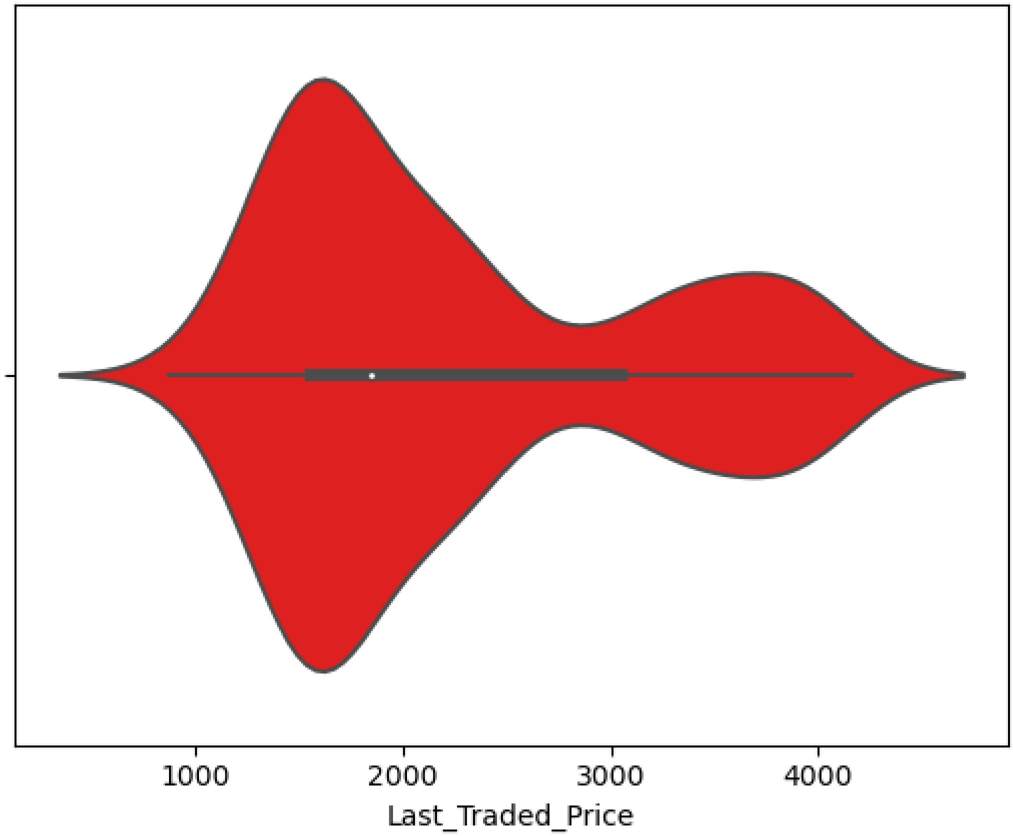
```python
#Violinplot
for i in df.columns:
    if df[i].dtypes != "object":
        sns.violinplot(x =df[i],color='red' )
        plt.show()
```

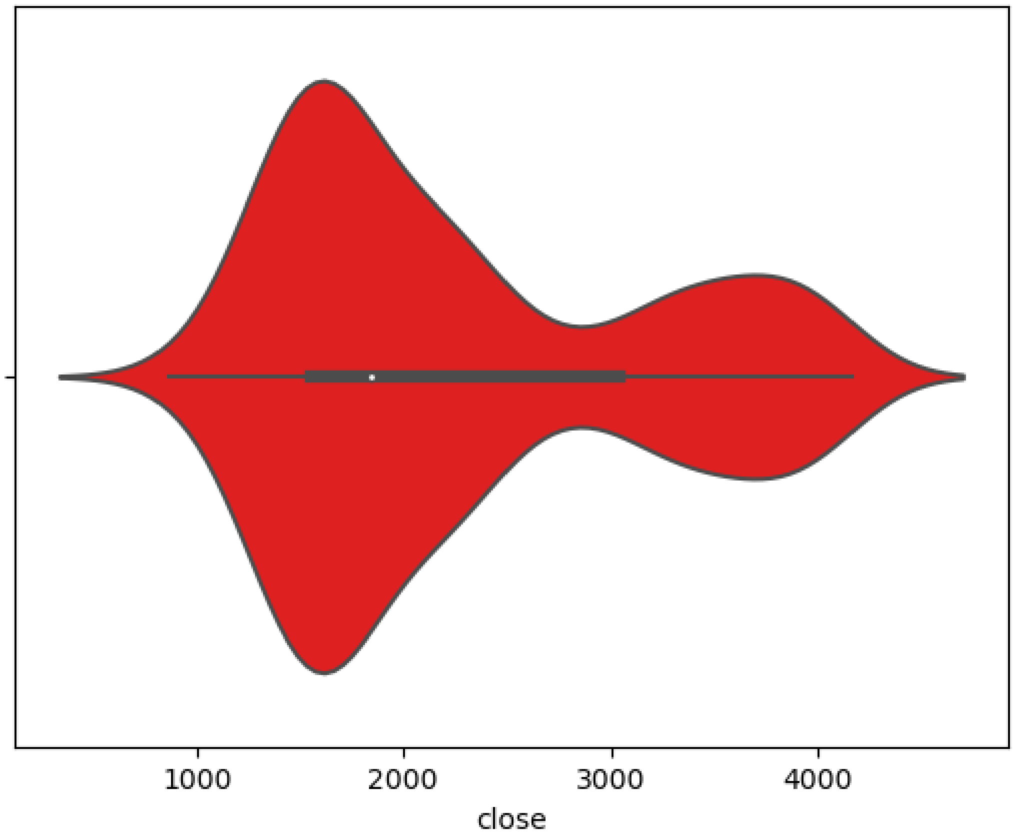


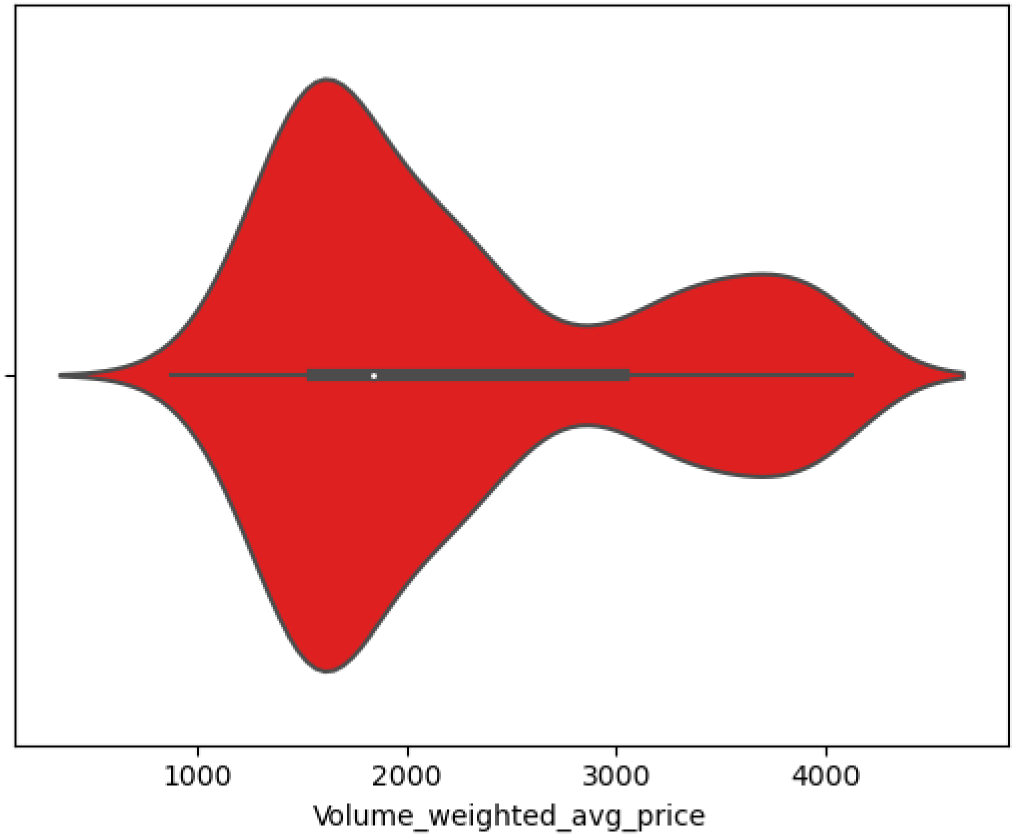


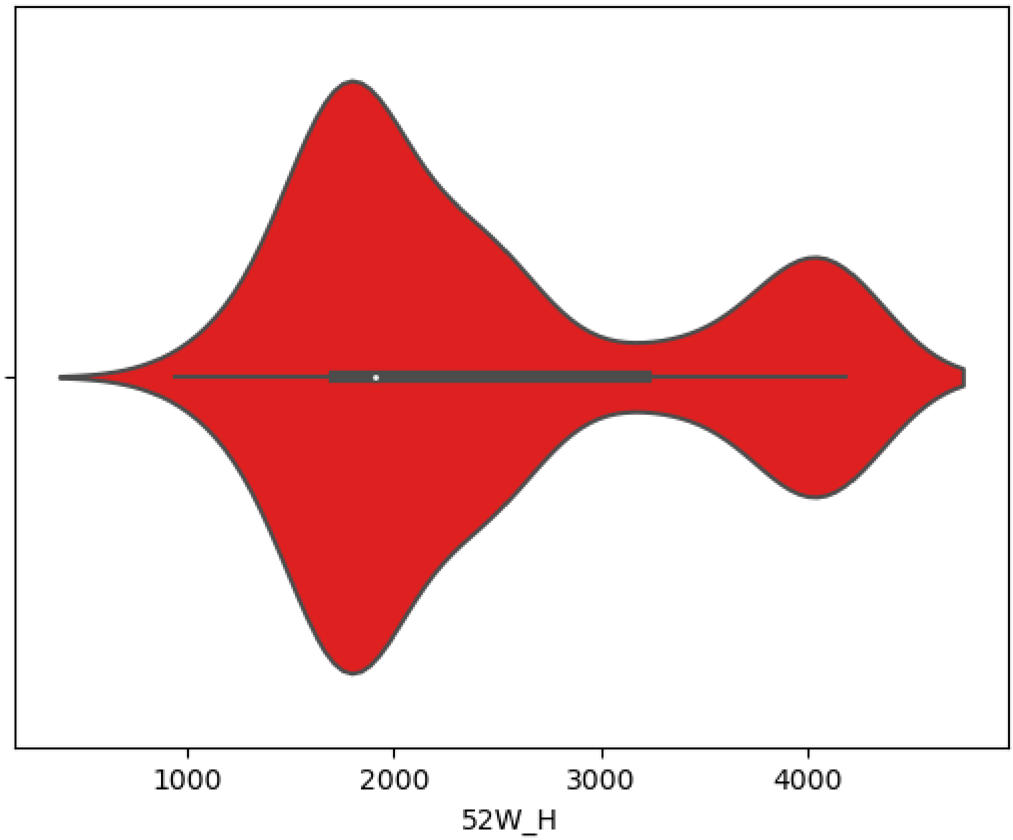

1000
2000
3000
4000
LOW

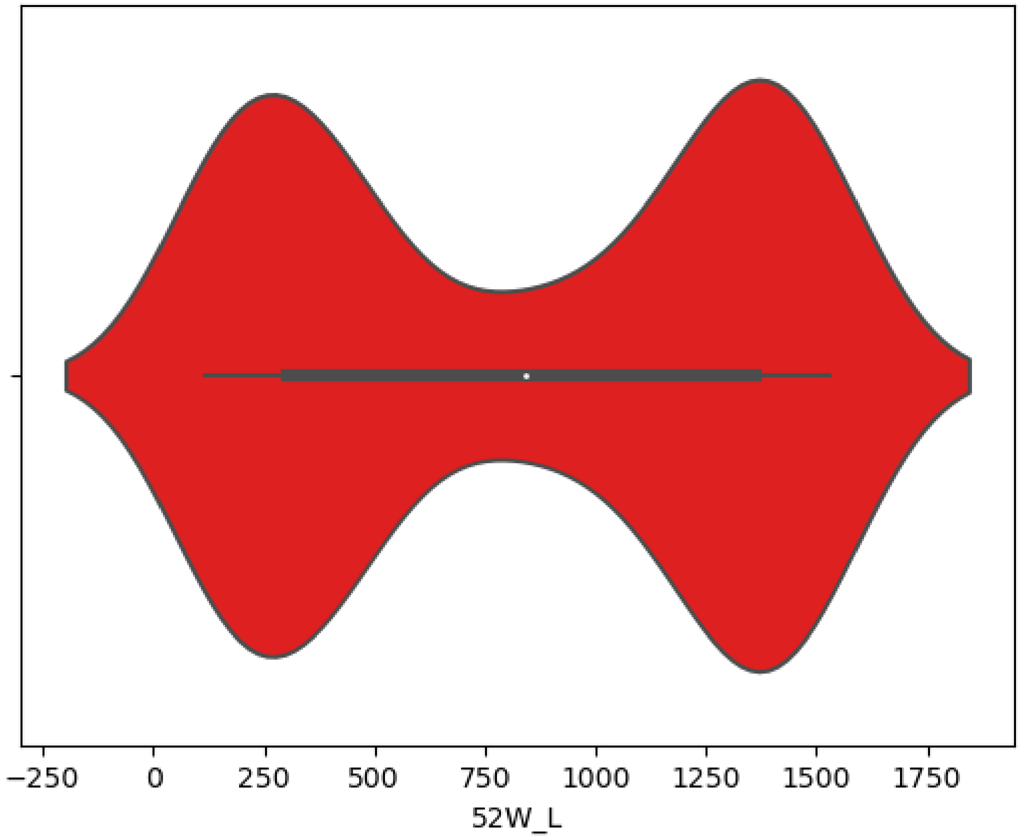

1000
2000
3000
4000
PREV_CLOSE

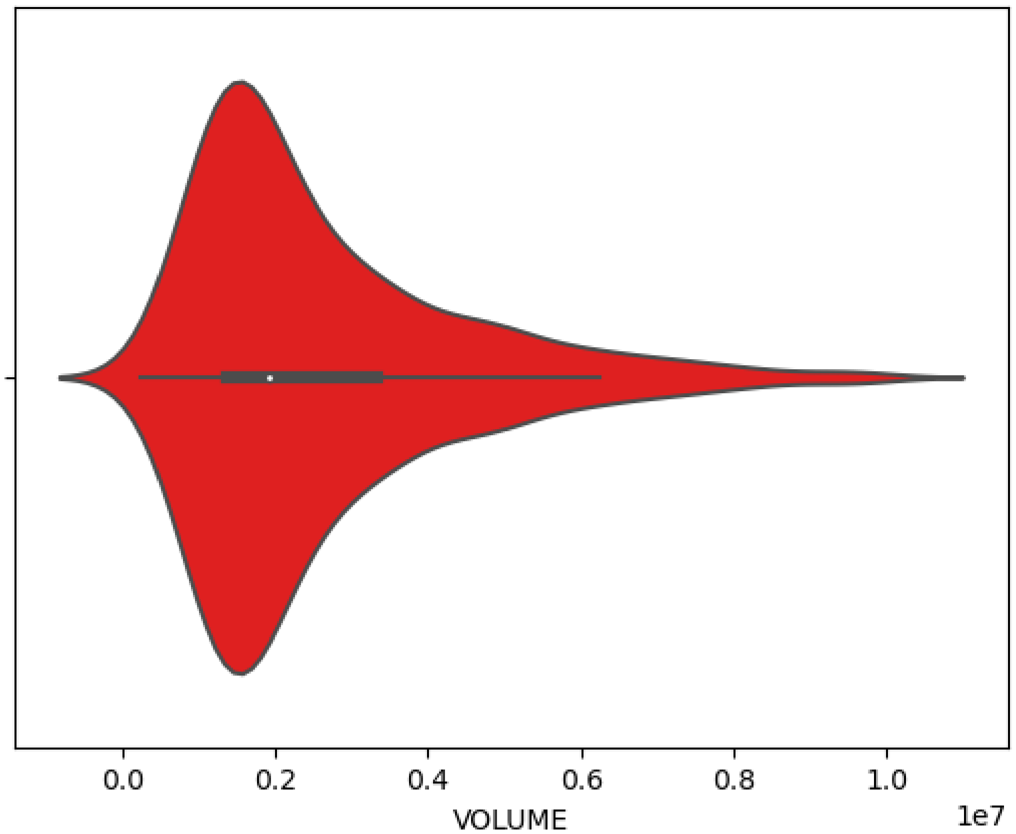

1000
2000
3000
4000
Last_Traded_Price


1000
2000
3000
4000
close

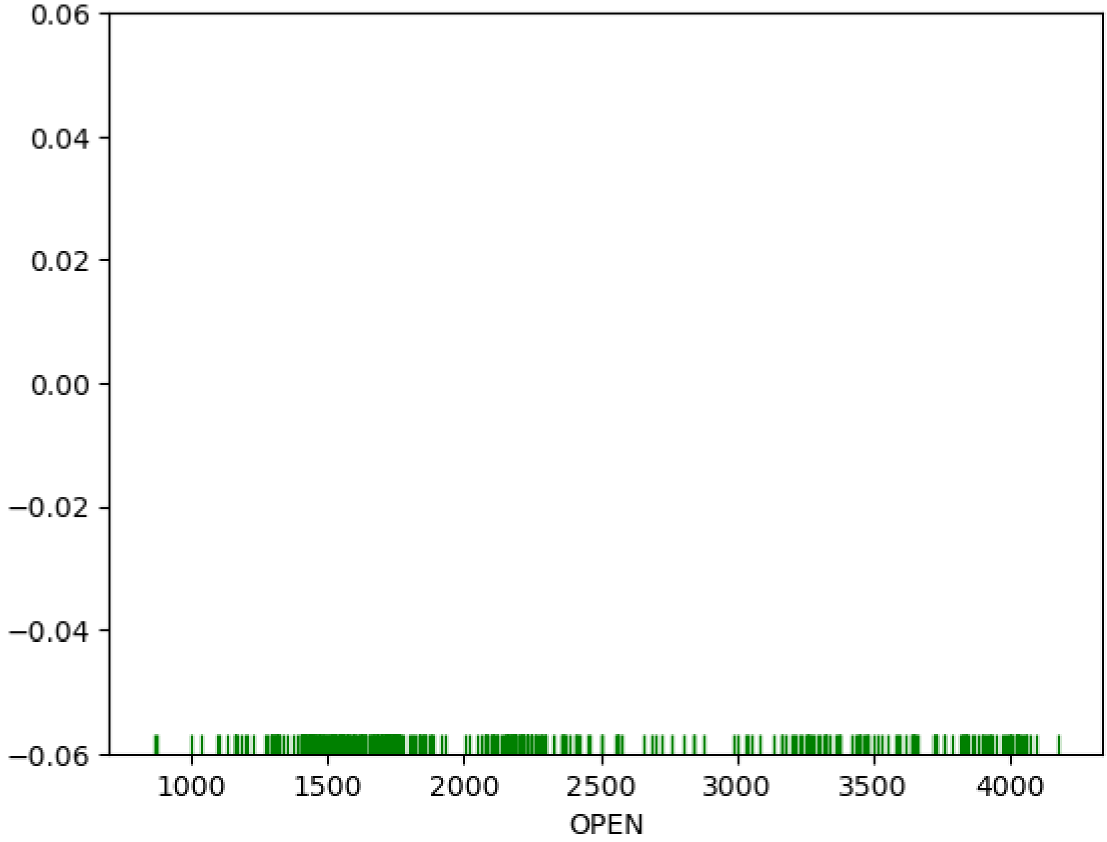

1000
2000
3000
4000
Volume_weighted_avg_price

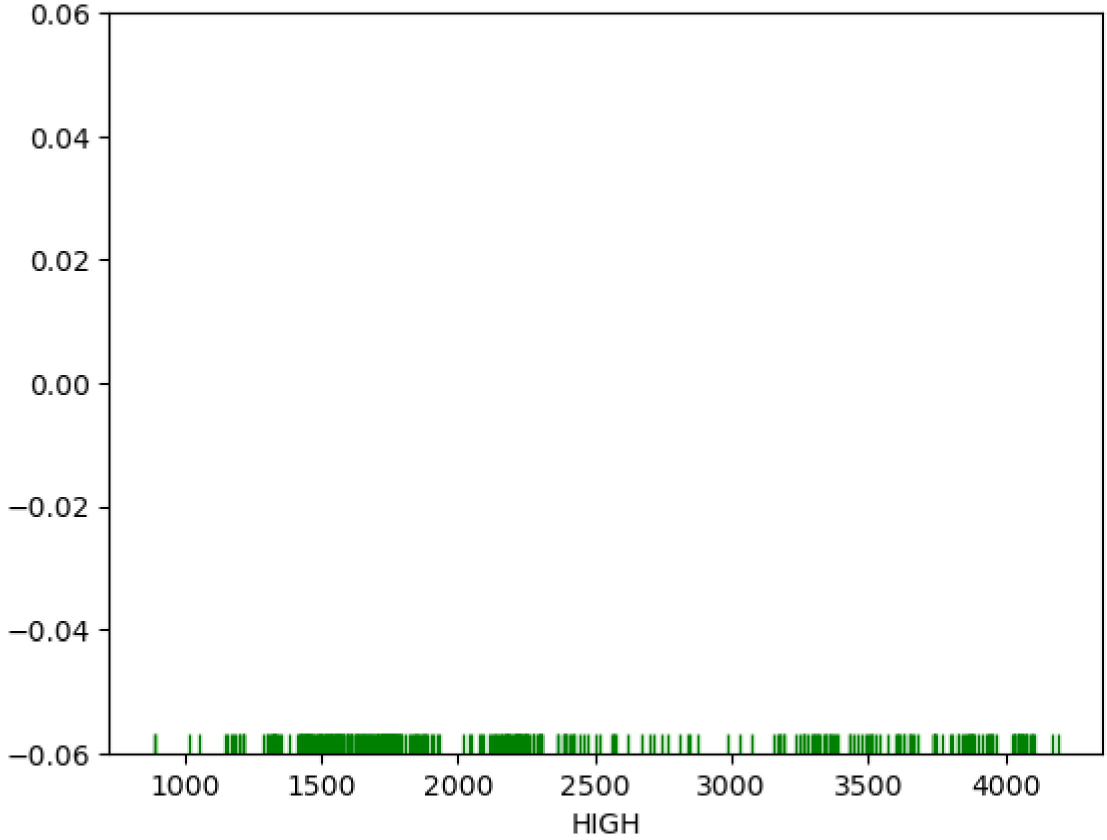

1000
2000
3000
4000
52W_H

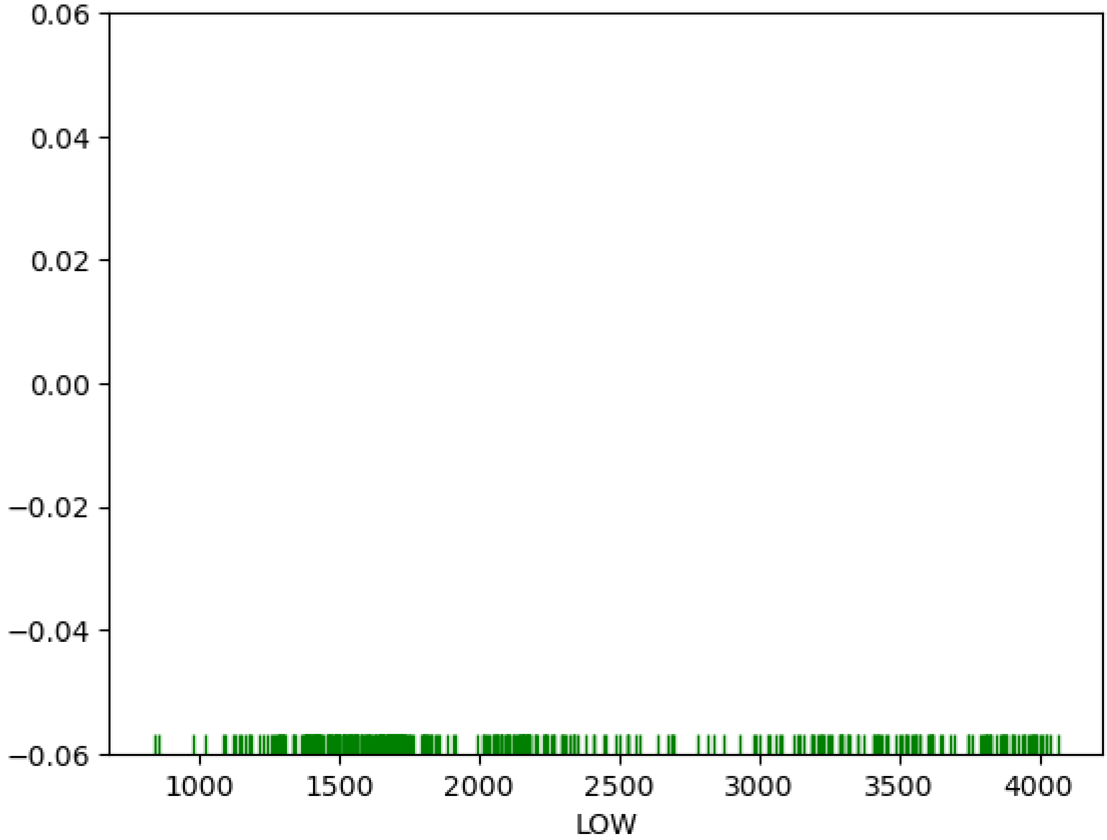

−250
0
250
500
750
1000
1250
1500
1750
52W_L

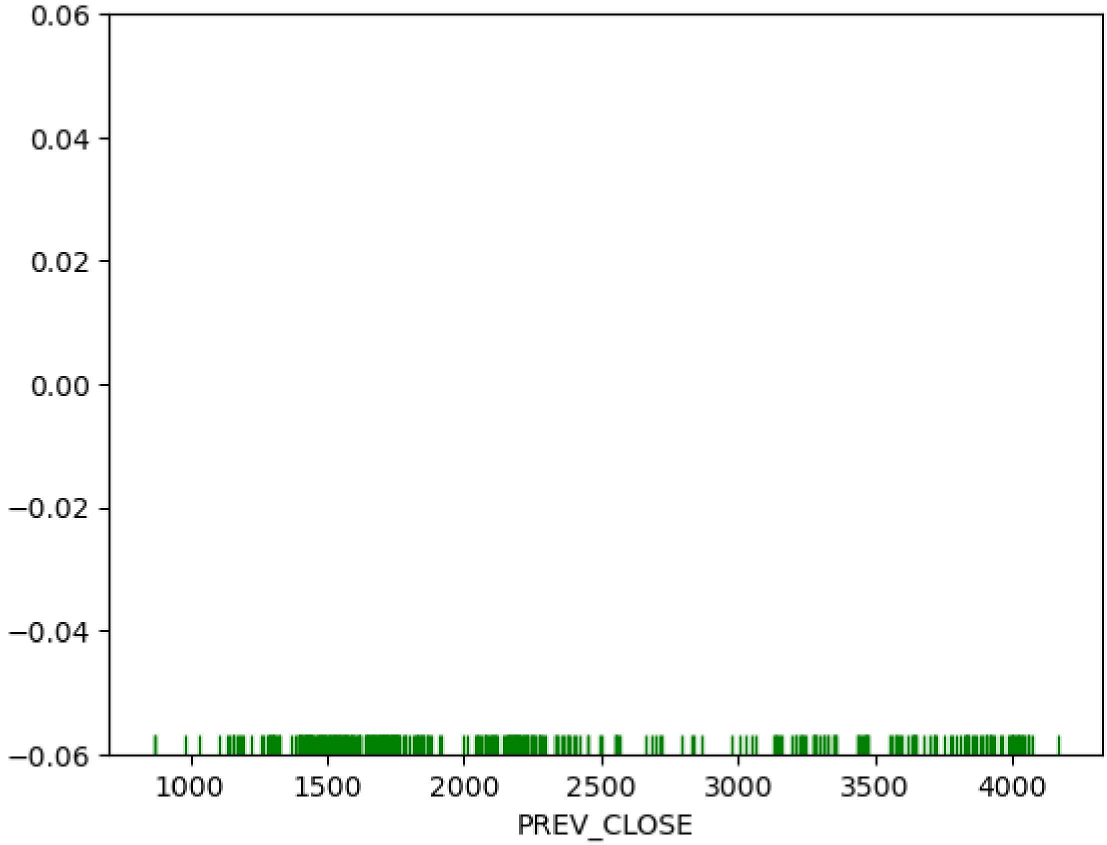

0.0
0.2
0.4
0.6
0.8
1.0
VOLUME
1e7

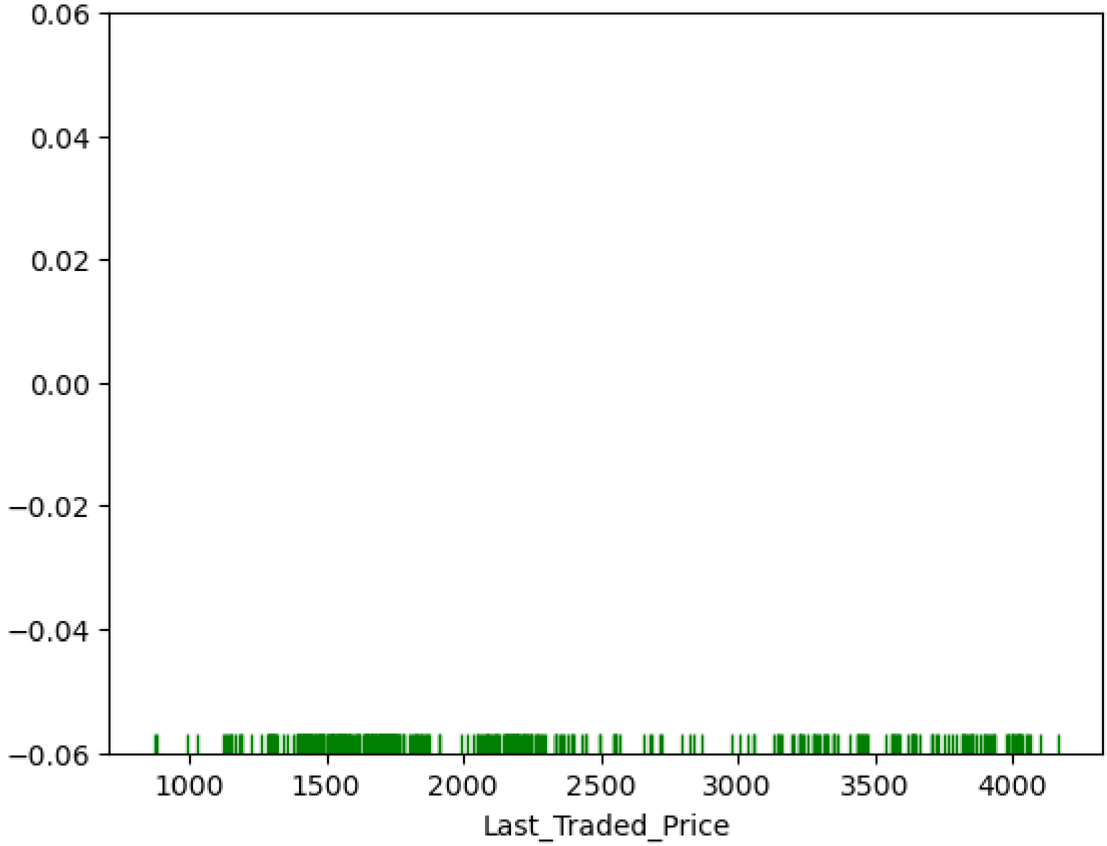
```
#Rugplot
for i in df.columns:
    if df[i].dtypes != "object":
        sns.rugplot(x =df[i],color='green' )
        plt.show()
```

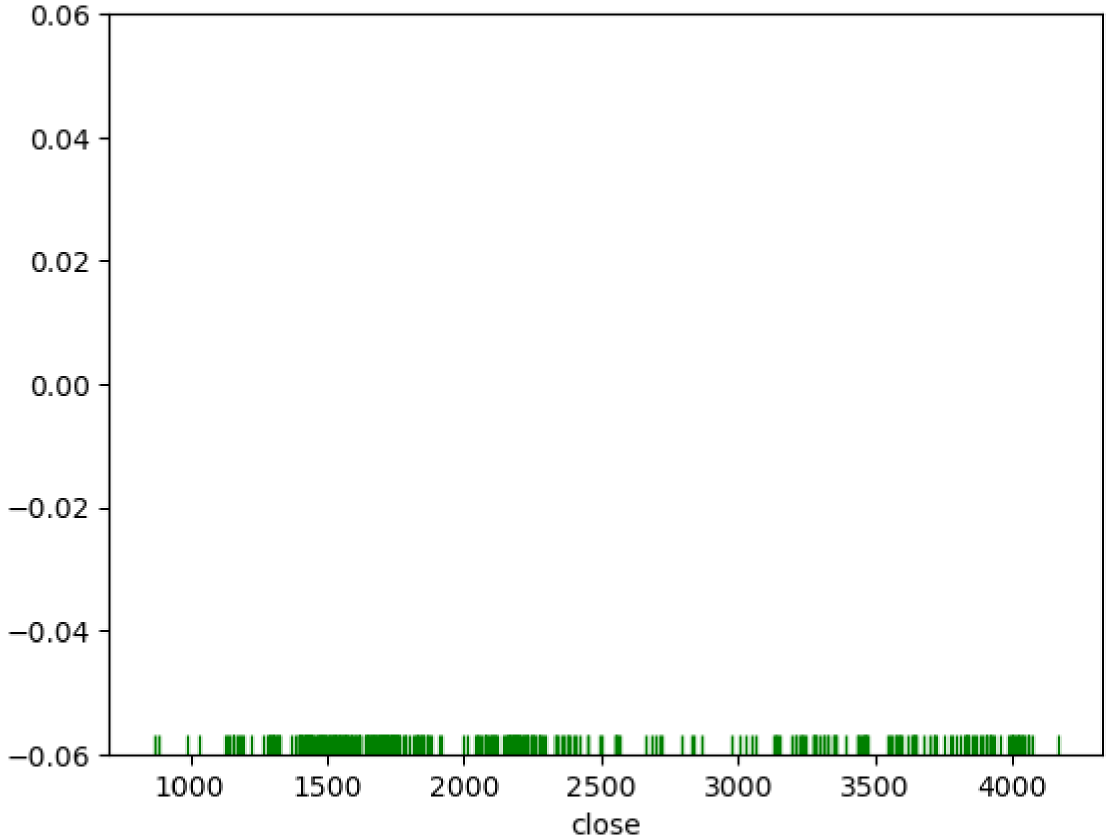


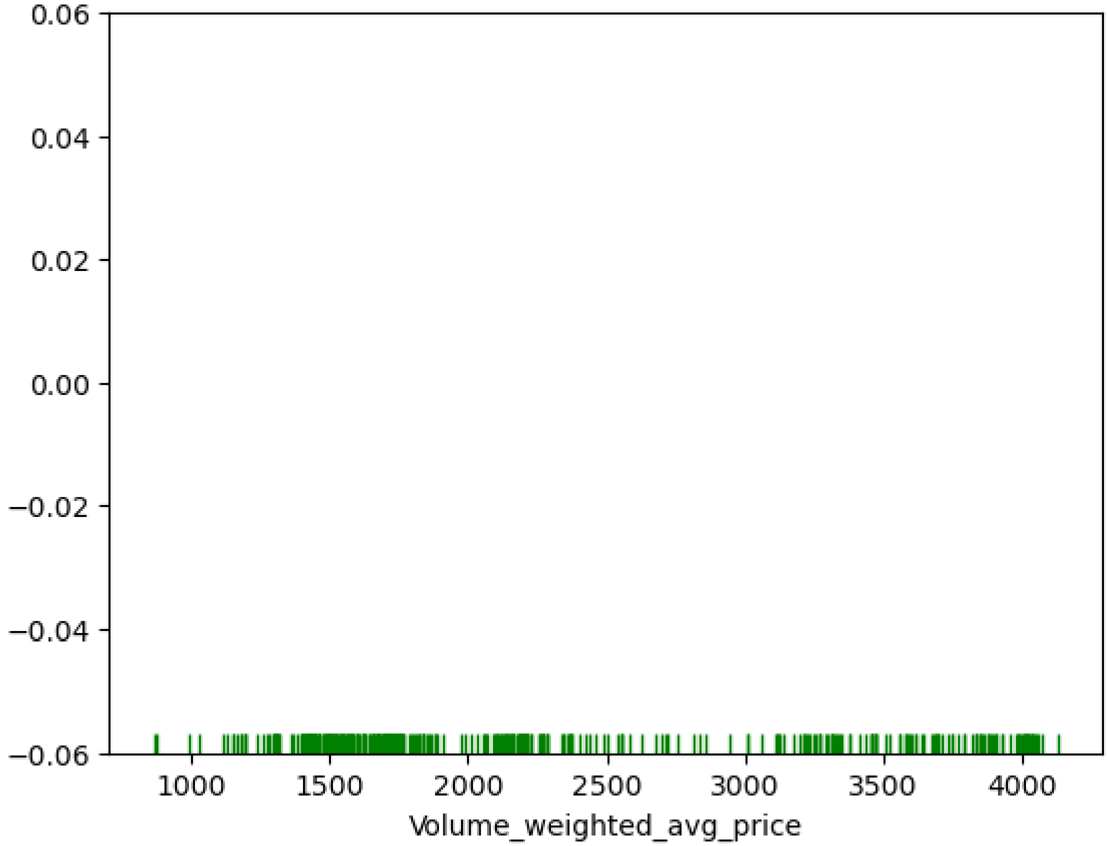


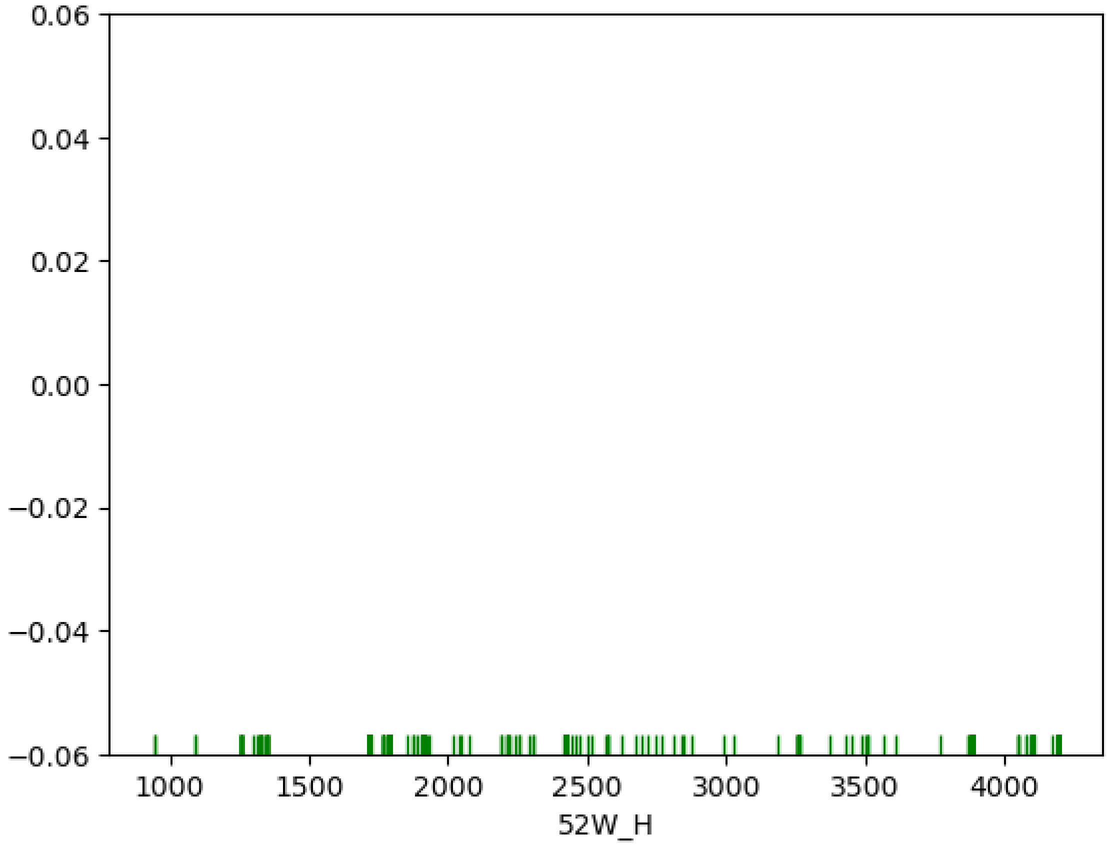

0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
1000
1500
2000
2500
3000
3500
4000
LOW

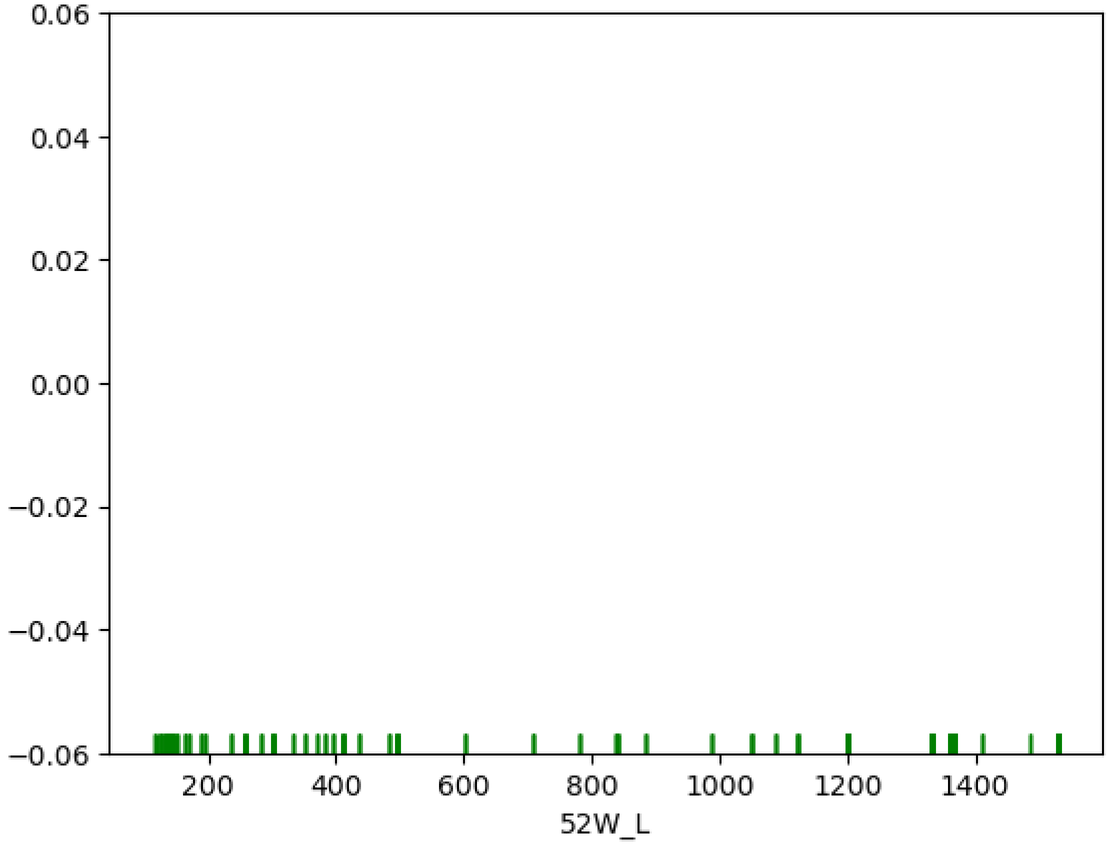

0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
1000
1500
2000
2500
3000
3500
4000
PREV_CLOSE

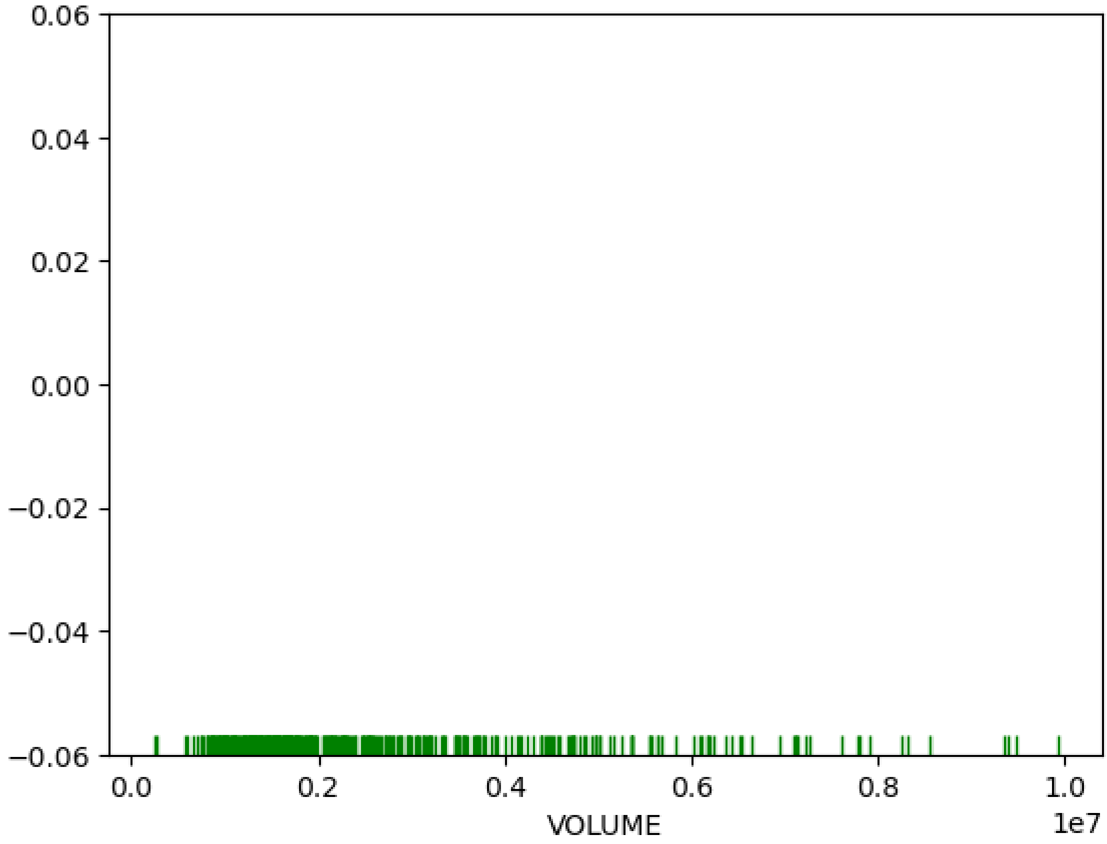

0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
1000
1500
2000
2500
3000
3500
4000
Last_Traded_Price


0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
1000
1500
2000
2500
3000
3500
4000
close

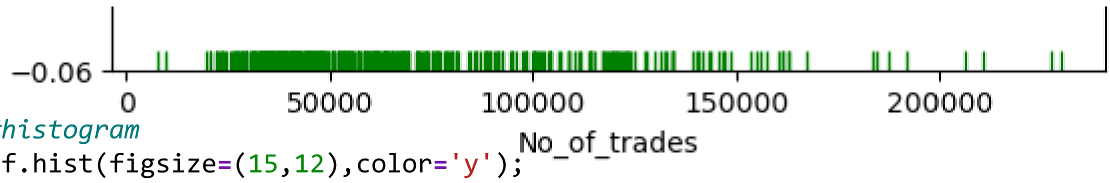

0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
1000
1500
2000
2500
3000
3500
4000
Volume_weighted_avg_price


0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
1000
1500
2000
2500
3000
3500
4000
52W_H

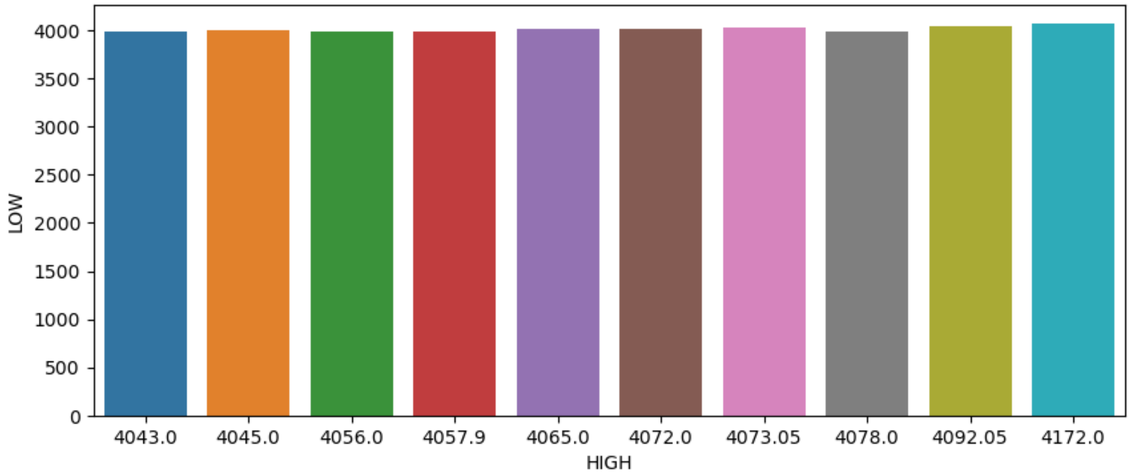




Multivariate analysis



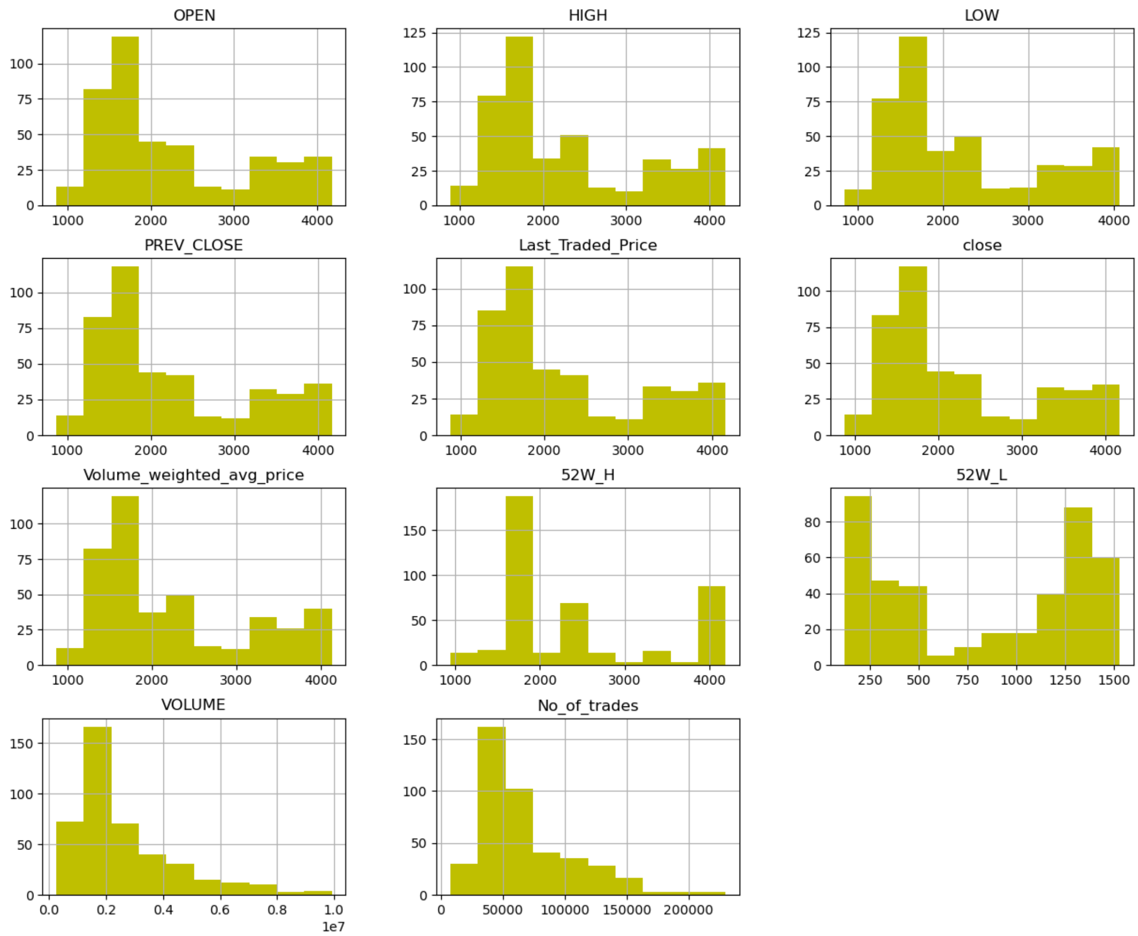

```python
#Barplot
plt.figure(figsize=(10,4))
sns.barplot(x='HIGH',y='LOW',data=df.sort_values(by='LOW',ascending=False)[:10]);
```




```python
#histogram
df.hist(figsize=(15,12),color='y');
plt.show
```

Out[66]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```

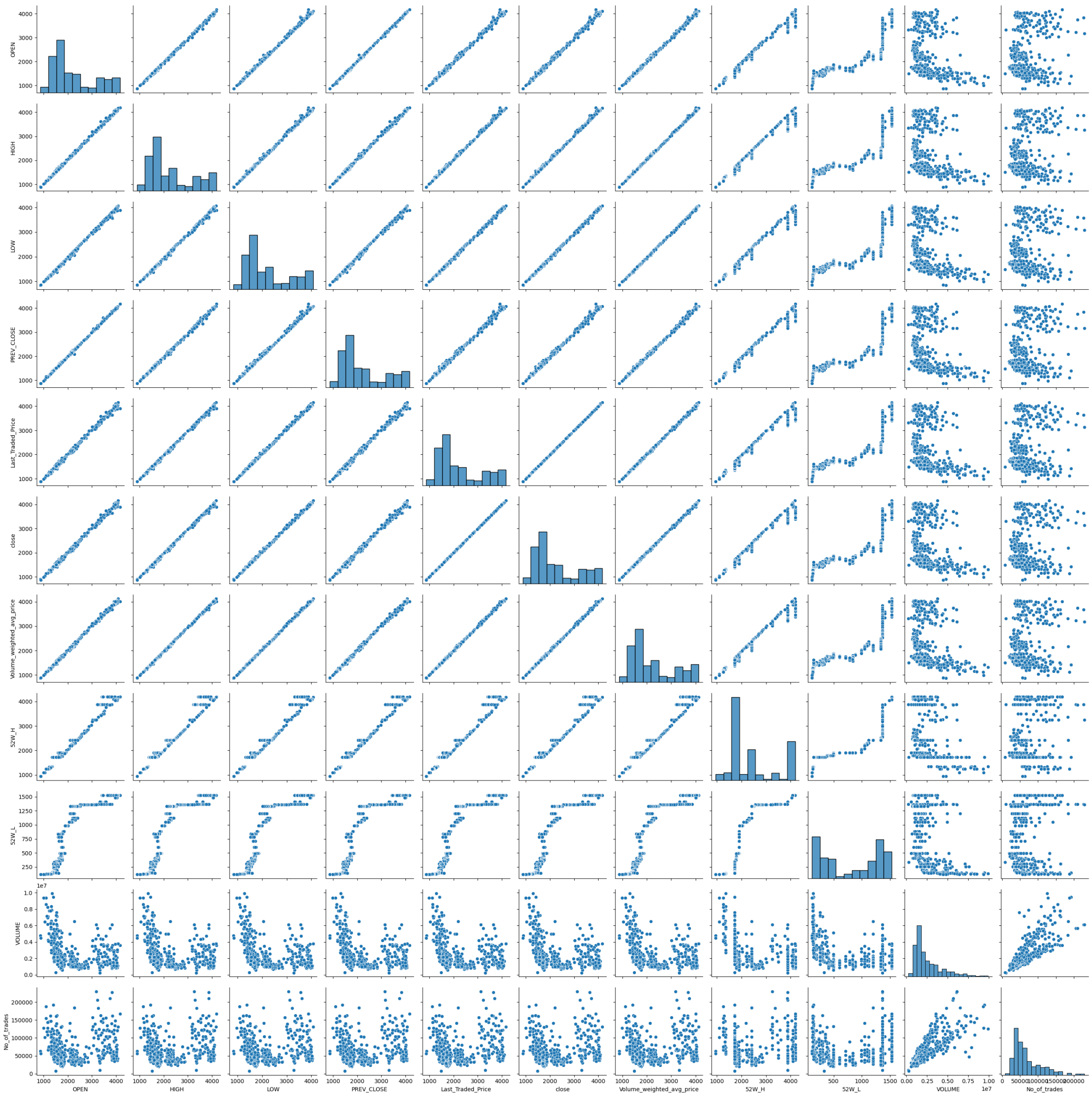


In [32]:

```
# pairplot of dataframe
sns.pairplot( df )
```

Out[32]:

<seaborn.axisgrid.PairGrid at 0x252378699d0>

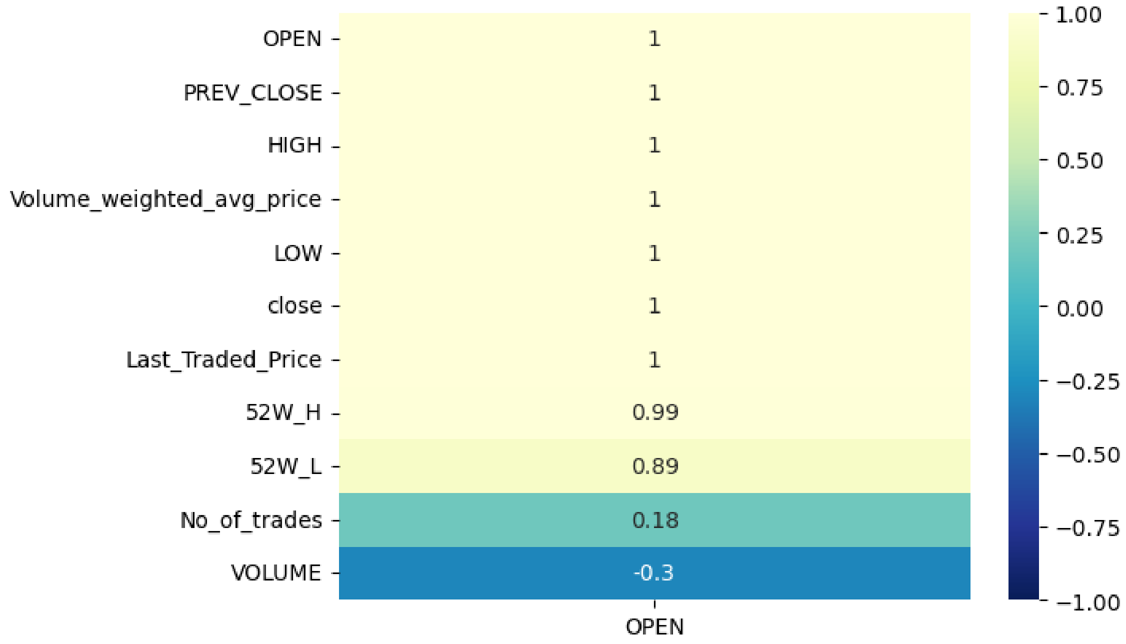


In [41]:

```
### Heatmap
sns.heatmap(df.corr()[['OPEN']].sort_values(by='OPEN', ascending=False), vmin=-1, vmax=1
```

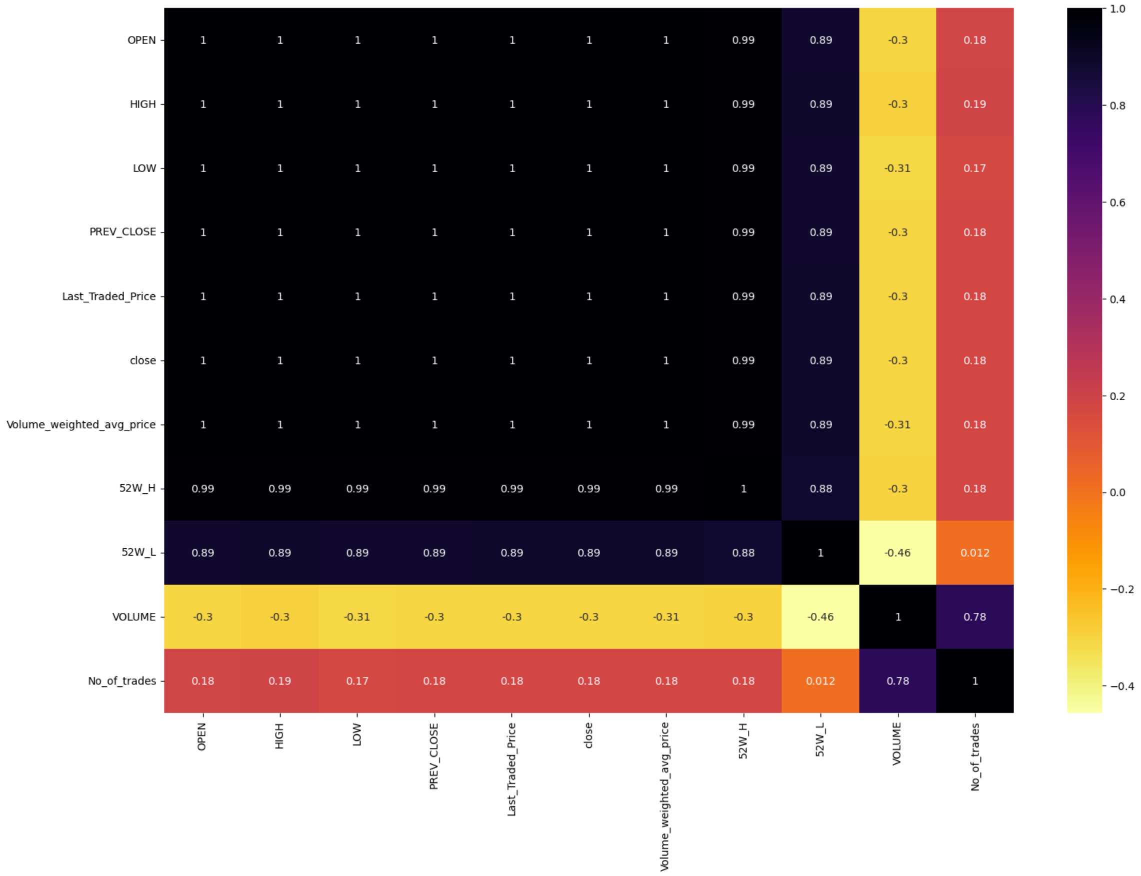
Out[41]:

```
<AxesSubplot:>
```



In [60]:

```
plt.figure(figsize=(18,12))
sns.heatmap(df.corr(),annot=True,cmap="inferno_r")
plt.show()
```



In [42]:

```
df.columns
```

Out[42]:

```
Index(['Date', 'OPEN', 'HIGH', 'LOW', 'PREV_CLOSE', 'Last_Traded_Price',
       'close', 'Volume_weighted_avg_price', '52W_H', '52W_L', 'VOLUME',
       'VALUE', 'No_of_trades'],
      dtype='object')
```

MODEL SELECTION AND TRAINING

In [43]:

```
# seperating the dependent and independent variables
x=df[['HIGH','LOW','Last_Traded_Price','close']].values
y=df[['OPEN']].values
```

In [44]:

```
sc=StandardScaler()
```

In [45]:

```
x=sc.fit_transform(x)
```

In [46]:

```
x
```

Out[46]:

```
array([[ 1.27919045,  1.2621007 ,  1.30522544,  1.28630738],
       [ 1.36833141,  1.39335622,  1.34572605,  1.34600661],
       [ 1.31484683,  1.37343962,  1.34403853,  1.33964844],
       ...,
       [-1.4056796 , -1.40523978, -1.41247785, -1.41523729],
       [-1.54278954, -1.56251813, -1.52925461, -1.52625197],
       [-1.5476923 , -1.5424874 , -1.54129229, -1.54718327]])
```

CROSS VALIDATION

In [47]:

```
from sklearn.model_selection import cross_val_score
models={
    'LinearRegression':LinearRegression(),
    'Lasso':Lasso(),
    'Ridge':Ridge(),
    'GradientBoostingRegressor':GradientBoostingRegressor(),
    'AdaBoostRegressor':AdaBoostRegressor(),
    'RandomForestRegressor':RandomForestRegressor(),
    'KneghborsRegressor':KNeighborsRegressor()
}
```

In [48]:

```
for name, model in models.items():
    scores=cross_val_score(model,x,y,scoring='neg_mean_squared_error',cv=10,n_jobs=-1)
    print('ss validaton model:{}'.format(name))
    rmse=np.sqrt(-scores)
    rmse_avarage=np.mean(rmse)
    print('AVARAGE RMSE:',rmse_avarage)
    print('*'*100)
```

```
ss validaton model:LinearRegression
AVARAGE RMSE: 20.979797176604972
***************************************************************************
*************************
ss validaton model:Lasso
AVARAGE RMSE: 27.09769422283536
***************************************************************************
*************************
ss validaton model:Ridge
AVARAGE RMSE: 30.836640186504617
***************************************************************************
*************************
ss validaton model:GradientBoostingRegressor
AVARAGE RMSE: 61.492895121593584
***************************************************************************
*************************
ss validaton model:AdaBoostRegressor
AVARAGE RMSE: 92.57694479204304
***************************************************************************
*************************
ss validaton model:RandomForestRegressor
AVARAGE RMSE: 62.06051093044184
***************************************************************************
*************************
ss validaton model:KneghborsRegressor
AVARAGE RMSE: 68.06277365279205
***************************************************************************
*************************
```

Splitting into training and testing

In [49]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

MODEL BUILDING

In [50]:

```
LR=LinearRegression()
```

In [51]:

```
LR.fit(x_train,y_train)
```

Out[51]:

```
LinearRegression()
```

In [52]:

```
print("model trained with {}".format(LR))
training_score = LR.score(x_train, y_train)*100
testing_score = LR.score(x_test, y_test)*100
score = r2_score(y_test, LR.predict(x_test))*100
mae = mean_absolute_error(y_test, LR.predict(x_test))
mse = mean_squared_error(y_test, LR.predict(x_test))
rmse = np.sqrt(mse)
print("r2score: ",score)
print("training_score: ", training_score)
print("testing_score: ", testing_score)
print("mae: ", mae)
print("mse: ", mse)
print("rmse_test: ", rmse)
```

```
model trained with LinearRegression()
r2score:  99.94406775865687
training_score:  99.94116589755342
testing_score:  99.94406775865687
mae:  14.941286650447006
mse:  444.8256084104617
rmse_test:  21.09088922758976
```

EVALUATION

In [53]:

```
y_pred = LR.predict(x)
```

In [54]:

```
OUTPUT = pd.DataFrame(zip(y,y_pred), columns=("ACTUAL", "PREDICTED"), dtype=float)
OUTPUT
```

Out[54]:

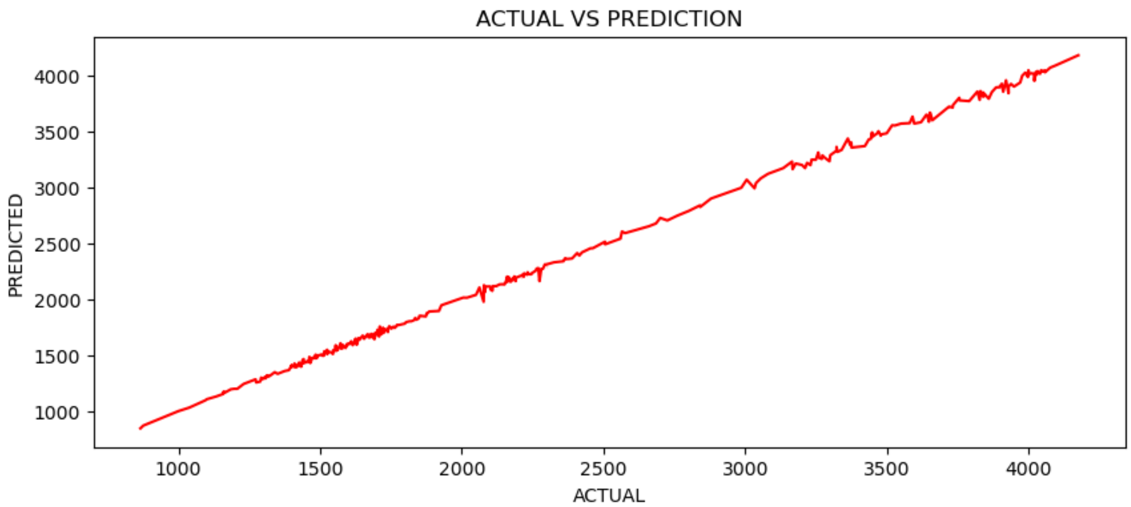
	ACTUAL	PREDICTED
0	3422.00	3370.979466
1	3447.45	3493.021352
2	3443.05	3435.564114
3	3450.00	3457.031431
4	3470.00	3503.933837
...	...	...
418	1095.00	1100.461267
419	1038.00	1035.182940
420	1000.00	1004.223901
421	866.00	850.431001
422	876.00	874.835999

423 rows × 2 columns

Visualizing the Prediction

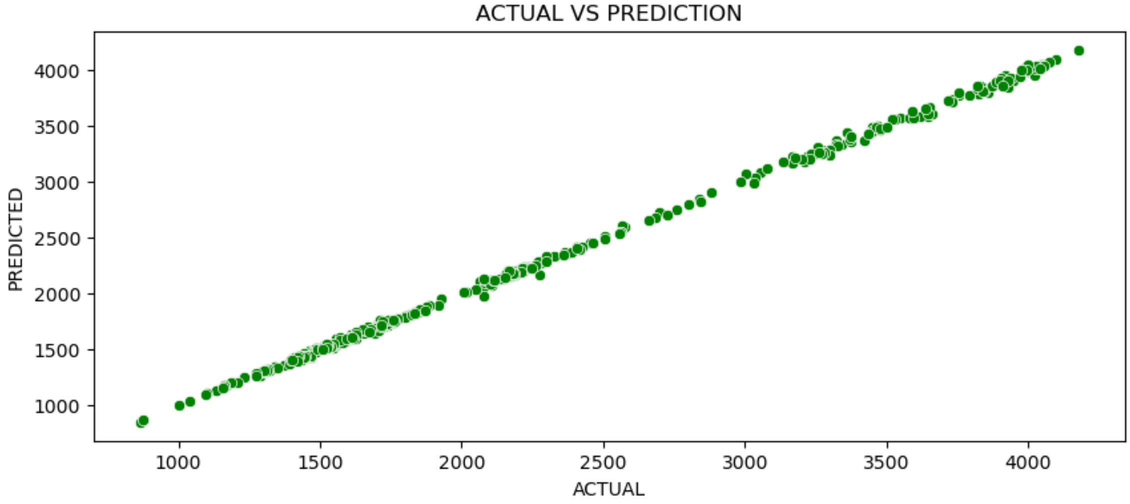
In [62]:

```
#Lineplot
plt.figure(figsize=(10,4))
sns.lineplot(x='ACTUAL', y='PREDICTED', data=OUTPUT, color="red")
plt.title("ACTUAL VS PREDICTION")
plt.show()
```



In [61]:

```
#Scatter Plot
plt.figure(figsize=(10,4))
sns.scatterplot(data = OUTPUT, x="ACTUAL", y = "PREDICTED",color="green")
plt.title("ACTUAL VS PREDICTION")
plt.show();
```



In [57]:

```
x
```

Out[57]:

```
array([[ 1.27919045,  1.2621007 ,  1.30522544,  1.28630738],
       [ 1.36833141,  1.39335622,  1.34572605,  1.34600661],
       [ 1.31484683,  1.37343962,  1.34403853,  1.33964844],
       ...,
       [-1.4056796 , -1.40523978, -1.41247785, -1.41523729],
       [-1.54278954, -1.56251813, -1.52925461, -1.52625197],
       [-1.5476923 , -1.5424874 , -1.54129229, -1.54718327]])
```

Conclusion

Based on the regression analysis performed on the Adani stock prediction, the following conclusions can be drawn:

->Relationship between variables: The regression analysis helps identify the relationship between the independent variables (such as historical stock prices, trading volumes, economic indicators, or sector performance) and the dependent variable (Adani stock price). It provides insights into how changes in the independent variables affect the stock price.

->Statistical significance: The regression analysis provides statistical measures, such as coefficients and p-values, to assess the significance of the independent variables in explaining the variation in Adani stock price. A significant p-value indicates that the variable has a meaningful impact on the stock price.

->Predictive power: The regression model can be used to estimate future Adani stock prices based on the identified relationships. By inputting values for the independent variables, the model can generate predictions for the stock price, aiding investors and traders in making informed decisions.

->Model evaluation: Various metrics, such as R-squared (coefficient of determination), adjusted R-squared, and root mean squared error (RMSE), can be used to evaluate the performance of the regression model. A higher R-squared value and a lower RMSE indicate a better fit of the model to the data.

->Assumptions: Regression analysis relies on assumptions, including linearity, independence of errors, homoscedasticity, and normality of residuals. It is important to assess these assumptions to ensure the validity of the regression model and the reliability of its predictions.

->Limitations: Regression analysis has its limitations, such as the assumption of a linear relationship between variables, potential presence of multicollinearity, and the inability to capture all factors that influence stock prices. Other external factors like market sentiment, news events, or regulatory changes may also impact stock prices and should be considered in conjunction with the regression analysis.

It is crucial to note that stock prediction is a challenging task, and regression analysis alone may not provide precise and accurate predictions. It is advisable to combine regression analysis with other techniques, perform thorough validation, and consider additional qualitative factors to make well-informed investment