Stock_Price_Prediction

PROJECT BY:

CB.EN.U4CSE20218

CB.EN.U4CSE20219

CB.EN.U4CSE20221

CB.EN.U4CSE20244

CB.EN.U4CSE20265

IMPORTING MODULES

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
import seaborn as sns
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
df = pd.read_csv("/content/drive/MyDrive/zyne.us.txt",parse_dates=['Date'])
df
```

Date Open High Low Close Volume OpenInt

EXPOLARATORY DATA ANALYSIS

```
df.shape
```

```
(574, 7)
```

```
df.describe()
```

	Open	High	Low	Close	Volume	OpenInt
count	574.000000	574.00000	574.000000	574.000000	5.740000e+02	574.0
mean	13.584548	14.16217	12.998530	13.526147	3.639375e+05	0.0
std	5.985249	6.36847	5.647351	5.944354	8.045633e+05	0.0
min	4.750000	5.40000	4.640000	4.990000	1.265400e+04	0.0
25%	8.910000	9.36875	8.602500	8.860000	1.091920e+05	0.0
50%	12.805000	13.38500	12.355000	12.795000	2.144885e+05	0.0
75%	18.220000	18.80000	17.810000	18.167500	3.817345e+05	0.0
max	37.980000	43.00000	33.680000	36.490000	1.363256e+07	0.0

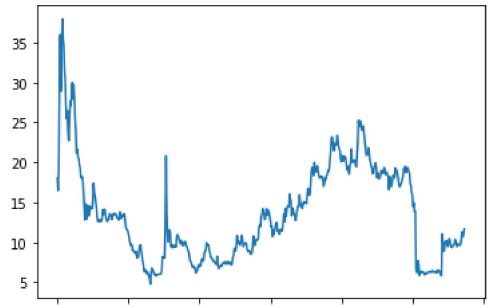
```
df.dtypes
```

```
Date       datetime64[ns]
Open              float64
High              float64
Low               float64
Close             float64
Volume              int64
OpenInt             int64
dtype: object
```

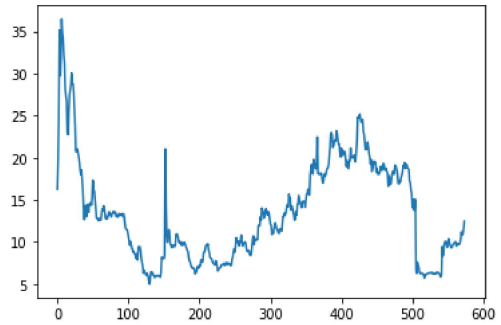
```
df.Open.value_counts()
```

```
6.26     4
14.50    4
6.20     4
9.95     4
9.60     4
        ..
7.67     1
8.13     1
8.94     1
9.34     1
11.68    1
Name: Open, Length: 459, dtype: int64
```

```
df.Open.plot(kind="line");
```




```
df.Close.plot(kind="line",x=None);
```




```
df.info
```

```
<bound method DataFrame.info of           Date   Open     High      Low  Close   Volume  OpenInt
0   2015-08-05  18.00  22.2500  15.5000  16.25  1535500        0
1   2015-08-06  16.48  19.5000  16.4600  19.32   331692        0
2   2015-08-07  19.31  24.9000  19.3100  24.54   269300        0
3   2015-08-10  29.70  42.8500  27.5000  35.14   539200        0
4   2015-08-11  35.98  37.6880  28.1600  29.73   243632        0
..         ...    ...      ...      ...    ...      ...      ...
569 2017-11-06  10.42  11.5400  10.4200  11.19   977948        0
570 2017-11-07  11.30  11.4200  10.6700  10.83   451210        0
571 2017-11-08  10.70  11.0600  10.3500  10.90   336449        0
572 2017-11-09  11.00  11.8563  10.9700  11.60   463067        0
573 2017-11-10  11.68  13.1500  11.3043  12.46   885587        0

[574 rows x 7 columns]>
```

```
df.sample()
```

	Date	Open	High	Low	Close	Volume	OpenInt
558	2017-10-20	9.65	9.88	9.525	9.73	299544	0

```
df.loc[2]
```

```
Date       2015-08-07 00:00:00
Open                     19.31
High                      24.9
```

```
Low                    19.31
Close                  24.54
Volume                269300
OpenInt                    0
Name: 2, dtype: object
```

```
df.head(10)
```

	Date	Open	High	Low	Close	Volume	OpenInt
0	2015-08-05	18.00	22.250	15.50	16.25	1535500	0
1	2015-08-06	16.48	19.500	16.46	19.32	331692	0
2	2015-08-07	19.31	24.900	19.31	24.54	269300	0
3	2015-08-10	29.70	42.850	27.50	35.14	539200	0
4	2015-08-11	35.98	37.688	28.16	29.73	243632	0
5	2015-08-12	28.91	36.000	28.00	32.05	238000	0
6	2015-08-13	35.00	43.000	32.52	36.49	407800	0
7	2015-08-14	37.98	39.500	33.68	34.84	233000	0
8	2015-08-17	35.42	36.300	32.04	34.11	124000	0
9	2015-08-18	34.72	34.720	31.45	32.22	141500	0

```
df.tail(10)
```

	Date	Open	High	Low	Close	Volume	OpenInt
564	2017-10-30	9.80	10.2699	9.6300	9.76	256138	0
565	2017-10-31	9.74	9.9492	9.6300	9.80	233242	0
566	2017-11-01	9.70	9.9300	9.4100	9.69	301604	0
567	2017-11-02	9.59	10.2000	9.5100	9.85	323496	0
568	2017-11-03	9.83	10.5000	9.8300	10.33	531495	0
569	2017-11-06	10.42	11.5400	10.4200	11.19	977948	0
570	2017-11-07	11.30	11.4200	10.6700	10.83	451210	0
571	2017-11-08	10.70	11.0600	10.3500	10.90	336449	0
572	2017-11-09	11.00	11.8563	10.9700	11.60	463067	0
573	2017-11-10	11.68	13.1500	11.3043	12.46	885587	0

```
df.isnull().any()
```

```
Date       False
Open       False
High       False
Low        False
Close      False
Volume     False
```

```
OpenInt    False
dtype: bool
```

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

```
Date       0
Open       0
High       0
Low        0
Close      0
Volume     0
OpenInt    0
dtype: int64
```

```
features=['Open','High','Low','Close','Volume']
features
```

```
['Open', 'High', 'Low', 'Close', 'Volume']
```

```
df['Date'].unique()
```

```
 '2017-09-22T00:00:00.000000000', '2017-09-25T00:00:00.000000000',
'2017-09-26T00:00:00.000000000', '2017-09-27T00:00:00.000000000',
'2017-09-28T00:00:00.000000000', '2017-09-29T00:00:00.000000000',
'2017-10-02T00:00:00.000000000', '2017-10-03T00:00:00.000000000',
'2017-10-04T00:00:00.000000000', '2017-10-05T00:00:00.000000000',
'2017-10-06T00:00:00.000000000', '2017-10-09T00:00:00.000000000',
'2017-10-10T00:00:00.000000000', '2017-10-11T00:00:00.000000000',
'2017-10-12T00:00:00.000000000', '2017-10-13T00:00:00.000000000',
'2017-10-16T00:00:00.000000000', '2017-10-17T00:00:00.000000000',
'2017-10-18T00:00:00.000000000', '2017-10-19T00:00:00.000000000',
'2017-10-20T00:00:00.000000000', '2017-10-23T00:00:00.000000000',
'2017-10-24T00:00:00.000000000', '2017-10-25T00:00:00.000000000',
'2017-10-26T00:00:00.000000000', '2017-10-27T00:00:00.000000000',
'2017-10-30T00:00:00.000000000', '2017-10-31T00:00:00.000000000',
'2017-11-01T00:00:00.000000000', '2017-11-02T00:00:00.000000000',
'2017-11-03T00:00:00.000000000', '2017-11-06T00:00:00.000000000',
'2017-11-07T00:00:00.000000000', '2017-11-08T00:00:00.000000000',
'2017-11-09T00:00:00.000000000', '2017-11-10T00:00:00.000000000'],
dtype='datetime64[ns]')
```

```
df['Open'].unique()
```

```
array([12.62  , 12.84  , 12.59  , 12.81  , 12.6   , 12.67  , 13.04  ,
       14.08  , 13.5   , 14.12  , 14.09  , 13.28  , 12.91  , 12.71  ,
       12.87  , 13.26  , 13.59  , 13.25  , 13.32  , 12.86  , 13.47  ,
       13.36  , 13.64  , 13.56  , 13.57  , 13.39  , 13.08  , 12.82  ,
       13.83  , 13.11  , 12.99  , 13.42  , 13.3   , 13.55  , 13.07  ,
       12.24  , 11.66  , 11.56  , 11.49  , 11.09  , 10.63  , 10.07  ,
        9.61  ,  9.81  ,  9.6   ,  9.01  ,  8.99  ,  8.78  ,  8.8   ,
        8.6   ,  8.86  ,  8.06  ,  8.27  ,  8.15  ,  9.09  ,  9.28  ,
        9.68  ,  8.9   ,  8.32  ,  7.6   ,  7.26  ,  6.37  ,  6.64  ,
        6.31  ,  6.4   ,  6.09  ,  6.28  ,  6.06  ,  5.8   ,  5.4   ,
        4.75  ,  6.76  ,  6.39  ,  6.45  ,  6.05  ,  6.1   ,  5.79  ,
        5.85  ,  5.99  ,  5.94  ,  5.98  ,  6.03  ,  6.    ,  6.04  ,
        6.08  ,  6.41  ,  8.17  ,  8.1   ,  8.18  ,  8.05  , 11.87  ,
       20.81  , 13.71  , 11.26  , 11.3   ,  9.5   ,  9.7   ,  9.31  ,
        9.62  ,  9.43  ,  9.4   ,  9.64  , 10.69  , 10.98  , 10.9   ,
       10.5   , 10.12  ,  9.9   , 10.26  ,  9.74  , 10.01  ,  9.8   ,
       10.11  ,  9.79  ,  9.34  ,  8.94  ,  8.13  ,  7.67  ,  7.54  ,
        7.29  ,  6.91  ,  6.9   ,  7.06  ,  6.75  ,  6.2   ,  6.5   ,
        7.02  ,  7.2   ,  6.92  ,  7.23  ,  7.11  ,  7.83  ,  7.96  ,
        7.64  ,  7.75  ,  8.69  ,  8.83  ,  9.11  ,  9.95  ,  8.95  ,
        8.7   ,  8.14  ,  8.09  ,  7.7   ,  7.71  ,  7.47  ,  7.43  ,
        7.34  ,  8.24  ,  7.3   ,  6.73  ,  6.78  ,  6.97  ,  7.14  ,
        7.    ,  7.35  ,  7.42  ,  7.36  ,  7.55  ,  7.33  ,  7.31  ,
        7.52  ,  7.17  ,  8.08  ,  8.3   ,  9.25  ,  9.51  , 10.83  ,
       10.    ,  9.87  , 10.1   , 10.09  ,  9.45  ,  9.89  ,  9.19  ,
        8.87  ,  8.5   ,  8.52  ,  8.66  ,  9.48  , 10.8   , 10.62  ,
       10.25  , 10.2   , 10.29  , 10.46  , 11.9   , 12.15  , 11.96  ,
       13.41  , 14.21  , 13.75  , 13.52  , 13.    , 13.99  , 14.    ,
       13.69  , 13.23  , 12.7   , 12.    , 10.73  , 11.55  , 10.99  ,
       11.5   , 12.33  , 12.48  , 11.4   , 11.07  , 11.05  , 11.6   ,
       11.62  , 11.38  , 11.48  , 13.96  , 14.57  , 14.41  , 16.03  ,
       15.4   , 15.27  , 13.31  , 14.26  , 14.1   , 13.6   , 12.74  ,
       13.72  , 14.59  , 15.08  , 14.72  , 14.44  , 14.95  , 14.88  ,
       15.09  , 14.96  , 15.04  , 14.85  , 16.04  , 16.74  , 16.25  ,
```

```
20.24  , 20.3   , 19.64  , 19.45  , 20.97  , 22.33  , 25.27  ,
24.51  , 25.17  , 24.47  , 24.17  , 24.57  , 23.52  , 22.62  ,
21.88  , 20.9   , 20.91  , 20.98  , 21.81  , 21.3   , 20.46  ,
19.97  , 19.59  , 18.59  , 18.65  , 19.32  , 19.7   , 19.93  ,
19.46  , 18.08  , 18.74  , 18.24  , 17.98  , 18.29  , 18.94  ,
18.97  , 18.67  , 18.69  , 19.37  , 19.18  , 18.8   , 18.2   ,
16.61  , 18.21  , 17.15  , 16.98  , 17.2   , 17.92  , 18.42  ,
18.15  , 18.05  , 19.3   , 17.13  , 16.95  , 17.26  , 17.58  ,
17.97  , 18.35  , 19.23  , 18.72  , 19.22  , 19.35  , 19.15  ,
18.83  , 18.7   , 16.26  , 15.84  , 14.92  , 13.8   , 13.89  ,
 6.56  ,  6.23  ,  6.27  ,  7.21  ,  6.36  ,  6.26  ,  6.24  ,
 5.93  ,  6.21  ,  6.07  ,  6.14  ,  6.25  ,  6.32  ,  6.33  ,
 6.44  ,  6.34  ,  6.3   ,  6.49  ,  6.22  ,  5.9   ,  5.82  ,
 8.88  ,  8.98  , 10.02  ,  9.69  , 10.45  , 10.1788,  9.56  ,
 9.38  ,  9.33  ,  9.65  ,  9.78  , 10.37  ,  9.59  ,  9.83  ,
10.42  , 10.7   , 11.    , 11.68  ])
```

```
df['Close'].unique()
```

```
13.53  , 13.14  , 13.1   , 13.17  , 13.22  , 13.19  , 13.07  ,
13.35  , 12.22  , 11.67  , 11.46  , 11.45  , 11.21  , 10.61  ,
10.19  ,  9.61  , 10.08  ,  9.67  ,  9.25  ,  9.1   ,  8.75  ,
 8.77  ,  8.54  ,  8.43  ,  8.04  ,  8.6   ,  7.86  ,  9.41  ,
 9.4   ,  9.45  ,  9.04  ,  8.15  ,  7.66  ,  7.15  ,  6.3   ,
 6.64  ,  6.39  ,  6.29  ,  6.05  ,  6.32  ,  6.18  ,  5.85  ,
 5.36  ,  4.99  ,  5.93  ,  6.49  ,  6.17  ,  6.07  ,  6.01  ,
 5.78  ,  5.95  ,  6.02  ,  5.94  ,  6.04  ,  6.    ,  5.82  ,
 8.21  ,  8.02  ,  8.08  ,  8.44  , 21.03  , 14.4   , 10.65  ,
 9.96  , 10.73  , 11.09  ,  9.75  ,  9.64  ,  9.36  ,  9.63  ,
 9.49  ,  9.6   , 10.92  , 10.97  , 10.55  , 10.02  ,  9.91  ,
10.07  ,  9.78  ,  9.59  ,  9.95  ,  9.7   ,  9.99  ,  9.28  ,
 9.03  ,  8.84  ,  7.77  ,  7.61  ,  7.23  ,  6.97  ,  6.9   ,
 6.99  ,  6.85  ,  6.77  ,  6.22  ,  6.34  ,  6.54  ,  7.13  ,
 6.91  ,  7.24  ,  7.14  ,  7.9   ,  7.59  ,  7.76  ,  8.74  ,
 8.8   ,  9.08  ,  9.51  ,  9.71  ,  9.77  ,  9.05  ,  8.19  ,
 8.09  ,  7.8   ,  7.55  ,  7.48  ,  7.37  ,  7.28  ,  7.65  ,
 7.09  ,  6.58  ,  6.72  ,  6.93  ,  7.    ,  7.3   ,  7.36  ,
 7.27  ,  7.5   ,  7.21  ,  7.33  ,  7.54  ,  7.56  ,  7.32  ,
 7.4   ,  7.18  ,  7.96  ,  9.2   ,  8.78  ,  9.43  , 10.5   ,
 9.94  , 10.11  ,  9.52  , 10.26  , 10.83  , 10.04  ,  9.88  ,
10.03  ,  9.87  ,  9.3   ,  8.89  ,  8.85  ,  8.99  ,  8.9799,
 8.48  ,  8.57  ,  8.47  , 10.72  ,  9.72  , 10.4   , 10.22  ,
10.31  , 10.28  , 11.9   , 12.02  , 11.92  , 12.99  , 12.63  ,
14.04  , 13.81  , 12.85  , 13.04  , 13.7   , 13.97  , 13.43  ,
13.61  , 13.38  , 12.7   , 12.12  , 12.    , 10.89  , 11.27  ,
10.99  , 11.54  , 11.97  , 12.26  , 11.48  , 11.43  , 11.13  ,
11.03  , 11.52  , 11.56  , 11.41  , 11.3   , 13.08  , 13.01  ,
13.42  , 13.28  , 13.55  , 14.23  , 14.44  , 14.17  , 15.69  ,
15.33  , 15.26  , 13.8   , 14.28  , 13.62  , 13.24  , 12.58  ,
12.79  , 13.18  , 14.51  , 15.53  , 15.07  , 14.62  , 14.8   ,
14.76  , 14.87  , 14.66  , 14.13  , 14.91  , 15.64  , 15.81  ,
15.66  , 15.59  , 17.95  , 18.75  , 19.2   , 18.25  , 18.08  ,
```

```
       18.76  , 19.25  , 19.6   , 19.36  , 19.47  , 18.27  , 18.77  ,
       18.06  , 18.91  , 18.98  , 18.52  , 18.63  , 19.35  , 19.19  ,
       18.61  , 18.53  , 18.38  , 18.2   , 16.61  , 17.89  , 17.47  ,
       16.86  , 17.02  , 17.96  , 18.33  , 19.12  , 18.93  , 18.02  ,
       16.97  , 16.9   , 17.31  , 17.13  , 17.67  , 19.06  , 18.94  ,
       19.43  , 19.05  , 18.81  , 18.73  , 18.59  , 17.26  , 17.04  ,
       16.26  , 15.82  , 15.06  , 13.75  ,  6.67  ,  6.27  ,  7.6   ,
        7.26  ,  7.03  ,  6.42  ,  6.24  ,  6.36  ,  6.25  ,  6.23  ,
        5.73  ,  6.16  ,  6.06  ,  6.2   ,  6.35  ,  6.28  ,  6.37  ,
        6.33  ,  6.21  ,  6.38  ,  5.84  ,  6.19  ,  9.44  ,  8.36  ,
        8.69  , 10.1   ,  9.66  ,  9.38  , 10.37  , 10.06  ,  9.47  ,
        9.32  ,  9.39  ,  9.73  ,  9.9   , 10.05  , 10.    ,  9.5   ,
        9.8   ,  9.76  ,  9.69  ,  9.85  , 10.33  , 11.19  , 10.9   ,
       11.6   , 12.46  ])
```

```
df['High'].unique()
```

```
       12.49  , 11.879 , 11.75  , 12.77  , 11.4   , 10.8   , 10.07  ,
       10.2   ,  9.84  ,  9.78  ,  9.38  ,  8.99  ,  9.35  ,  8.8   ,
        8.8199,  9.17  ,  8.66  ,  8.44  , 10.84  ,  9.735 , 10.    ,
        9.68  ,  9.7999,  8.33  ,  8.01  ,  7.26  ,  6.72  ,  6.64  ,
        6.48  ,  6.4   ,  6.49  ,  6.5599,  6.51  ,  6.16  ,  5.4   ,
        6.01  ,  7.0699,  7.    ,  6.225 ,  6.18  ,  6.03  ,  6.21  ,
        6.17  ,  6.28  ,  6.07  ,  6.05  ,  6.04  ,  6.45  ,  8.3   ,
        8.86  ,  8.35  ,  8.22  ,  8.5   , 21.5626, 20.81  , 13.71  ,
       11.85  , 11.    , 12.0698, 11.98  , 11.72  , 10.24  , 10.1599,
        9.66  , 10.09  ,  9.8   ,  9.92  ,  9.97  , 11.3   , 11.6799,
       10.614 , 10.37  , 10.32  ,  9.875 , 10.01  ,  9.8824, 10.19  ,
       10.5676,  9.99  ,  9.6   ,  9.365 ,  9.48  ,  8.78  ,  8.13  ,
        8.07  ,  7.54  ,  7.29  ,  7.185 ,  7.1199,  7.11  ,  6.9199,
        6.9   ,  6.6488,  6.66  ,  7.57  ,  7.53  ,  7.293 ,  7.343 ,
        7.5151,  7.89  ,  8.4   ,  8.12  ,  7.9   ,  9.29  ,  9.24  ,
        9.15  ,  9.79  ,  9.9828, 10.17  ,  9.95  ,  9.77  ,  9.19  ,
        8.89  ,  8.4999,  8.1699,  7.8099,  7.93  ,  7.5353,  7.67  ,
        7.79  ,  7.68  ,  8.94  ,  7.449 ,  7.05  ,  7.0525,  7.1758,
        7.175 ,  7.49  ,  7.4204,  7.5   ,  7.42  ,  7.7   ,  7.55  ,
        7.45  ,  7.6198,  7.6   ,  7.48  ,  7.791 ,  7.52  ,  7.3414,
        7.4   ,  8.25  ,  9.63  ,  9.4299,  9.5   , 10.5   , 10.83  ,
       10.39  , 10.29  , 10.1   , 10.4999, 11.01  , 11.15  ,  9.89  ,
       10.1294,  9.55  ,  9.21  ,  9.2699,  9.12  ,  8.68  , 10.74  ,
       10.62  , 10.42  , 10.6147, 10.8956, 11.9   , 12.4   , 12.41  ,
       14.38  , 14.44  , 13.96  , 13.37  , 14.    , 14.32  , 14.19  ,
       13.7189, 14.4995, 13.86  , 13.49  , 12.995 , 12.38  , 12.06  ,
       11.709 , 11.56  , 11.7089, 12.058 , 12.76  , 12.55  , 12.48  ,
       11.52  , 11.5   , 11.2373, 11.7   , 11.92  , 11.795 , 11.605 ,
       13.39  , 14.3984, 14.779 , 14.82  , 16.    , 16.91  , 15.9069,
       15.29  , 14.3   , 14.34  , 14.1399, 13.9284, 13.42  , 12.82  ,
       13.75  , 14.64  , 14.66  , 15.72  , 16.59  , 15.65  , 14.8669,
       15.03  , 15.49  , 15.45  , 15.1309, 14.999 , 15.5   , 16.74  ,
       16.3797, 15.9   , 18.    , 19.943 , 19.94  , 19.689 , 18.75  ,
       20.36  , 20.2   , 19.45  , 19.4376, 23.0685, 19.69  , 18.59  ,
```

```
       18.84  , 19.36  , 18.82  , 19.3   , 20.73  , 19.27  , 18.4714,
       17.27  , 17.41  , 17.58  , 17.7499, 19.4291, 19.34  , 19.64  ,
       20.    , 19.242 , 19.7   , 19.15  , 18.83  , 18.7999, 17.822 ,
       17.3332, 16.5   , 16.16  , 15.55  , 15.1164, 14.25  , 15.59  ,
        6.77  ,  8.4299,  7.3092,  6.69  ,  6.57  ,  6.639 ,  6.5   ,
        6.325 ,  6.2889,  6.37  ,  6.436 ,  6.443 ,  6.85  ,  6.59  ,
        6.43  ,  6.39  ,  6.35  ,  6.33  ,  6.55  ,  6.53  ,  6.52  ,
        6.25  ,  6.    , 11.28  , 10.1472,  9.27  , 10.26  , 10.4485,
       10.35  ,  9.85  , 10.12  , 10.64  , 10.499 , 10.1788,  9.76  ,
        9.49  ,  9.69  ,  9.88  , 10.06  , 10.59  ,  9.93  , 10.2699,
        9.9492, 11.54  , 11.42  , 11.06  , 11.8563, 13.15  ])
```

```
df['Low'].unique()
```

```
       13.12  , 12.92  , 13.31  , 13.04  , 12.6   , 13.03  , 13.02  ,
       12.85  , 12.62  , 12.84  , 12.93  , 13.18  , 12.55  , 12.12  ,
       10.89  , 11.101 , 11.06  , 10.99  , 10.6   ,  9.78  ,  9.3971,
        9.33  ,  9.    ,  9.25  ,  8.92  ,  8.5408,  8.72  ,  8.08  ,
        8.19  ,  7.9   ,  7.42  ,  7.7   ,  7.55  ,  8.15  ,  8.84  ,
        8.99  ,  8.1291,  7.6115,  7.05  ,  6.2   ,  6.35  ,  6.36  ,
        6.1735,  5.8   ,  6.09  ,  6.18  ,  5.14  ,  4.78  ,  4.64  ,
        6.05  ,  5.7407,  6.25  ,  5.76  ,  5.59  ,  5.17  ,  5.85  ,
        5.54  ,  5.69  ,  5.75  ,  5.86  ,  6.41  ,  7.7706,  7.75  ,
        7.8   , 11.87  , 14.01  , 10.335 ,  9.72  ,  9.71  , 10.58  ,
       10.8894,  9.5   ,  9.22  ,  9.26  ,  8.55  ,  9.2   ,  9.35  ,
        9.2888,  9.45  ,  9.4   , 10.22  , 10.526 , 10.357 ,  9.83  ,
        9.01  ,  9.52  ,  9.6   ,  9.7   ,  9.63  ,  9.36  ,  9.1   ,
        8.91  ,  8.81  ,  7.87  ,  7.6   ,  6.85  ,  6.72  ,  6.9   ,
        6.75  ,  6.74  ,  6.02  ,  6.1218,  6.47  ,  6.8801,  6.62  ,
        6.7501,  7.1   ,  7.11  ,  7.3101,  8.59  ,  8.6501,  8.7966,
        9.13  ,  9.55  ,  8.79  ,  8.61  ,  8.14  ,  8.    ,  7.8001,
        7.5   ,  7.31  ,  7.18  ,  7.34  ,  6.95  ,  6.5   ,  6.31  ,
        6.67  ,  6.82  ,  6.9101,  7.    ,  7.1659,  7.25  ,  7.2   ,
        7.22  ,  7.33  ,  7.14  ,  7.1901,  7.41  ,  7.28  ,  7.09  ,
        7.02  ,  7.3001,  7.8101,  8.57  ,  8.71  ,  9.67  ,  9.691 ,
        9.65  ,  9.51  ,  9.64  ,  9.84  , 10.26  ,  9.99  ,  9.11  ,
        9.18  ,  9.74  ,  9.69  ,  9.1259,  8.6401,  8.6   ,  8.44  ,
        8.345 ,  8.2513,  8.65  ,  9.48  , 10.1   ,  9.6416,  9.95  ,
       10.08  ,  9.94  , 10.2   , 10.4   , 11.1   , 11.26  , 11.819 ,
       12.27  , 13.306 , 13.07  , 12.87  , 13.62  , 13.11  , 13.52  ,
       12.9011, 12.56  , 11.891 , 11.59  , 10.56  , 10.57  , 10.9   ,
       10.8   , 11.4   , 11.96  , 11.84  , 10.87  , 11.    , 10.66  ,
       10.98  , 11.33  , 10.81  , 10.8301, 11.3958, 12.5   , 13.0273,
       12.72  , 13.6524, 14.    , 14.02  , 14.201 , 14.88  , 14.8211,
       13.8   , 13.09  , 13.95  , 13.4897, 12.28  , 12.7844, 12.88  ,
       13.38  , 13.75  , 15.    , 14.21  , 14.22  , 13.86  , 14.125 ,
       14.66  , 14.75  , 14.55  , 14.03  , 14.43  , 14.85  , 15.73  ,
       15.88  , 15.4   , 15.8   , 18.02  , 18.7313, 17.69  , 18.29  ,
       18.651 , 18.8   , 18.51  , 18.7   , 19.55  , 17.81  , 18.01  ,
       17.9   , 17.8   , 18.    , 17.33  , 16.0907, 16.87  , 17.34  ,
       17.5   , 17.82  , 17.92  , 18.09  , 18.61  , 18.69  , 20.29  ,
       21.32  , 22.27  , 21.17  , 21.12  , 21.26  , 21.39  , 21.3   ,
       21.86  , 22.16  , 22.42  , 22.03  , 21.3101, 21.311 , 20.5716,
```

```
       18.35  , 18.6165, 18.63  , 18.8501, 18.66  , 18.31  , 17.21  ,
       16.85  , 15.59  , 13.46  , 13.705 , 13.8479,  5.9301,  7.0508,
        6.7   ,  5.42  ,  6.0674,  6.04  ,  6.17  ,  6.15  ,  5.92  ,
        5.62  ,  5.65  ,  5.9   ,  6.03  ,  6.0542,  6.101 ,  6.161 ,
        6.21  ,  6.2178,  6.2975,  6.1   ,  6.13  ,  6.1901,  6.221 ,
        6.22  ,  5.821 ,  5.81  ,  9.38  ,  8.36  ,  8.39  ,  9.5422,
       10.02  ,  9.3801,  9.9   ,  9.92  ,  9.31  ,  9.09  ,  9.28  ,
        9.525 ,  9.851 ,  9.8101,  9.41  , 10.42  , 10.67  , 10.35  ,
       10.97  , 11.3043])
```

```
df.drop(['OpenInt'], axis=1, inplace=True)
df
```

	Date	Open	High	Low	Close	Volume
0	2015-08-05	18.00	22.2500	15.5000	16.25	1535500
1	2015-08-06	16.48	19.5000	16.4600	19.32	331692
2	2015-08-07	19.31	24.9000	19.3100	24.54	269300
3	2015-08-10	29.70	42.8500	27.5000	35.14	539200
4	2015-08-11	35.98	37.6880	28.1600	29.73	243632
...	...	...	...	...	...	...
569	2017-11-06	10.42	11.5400	10.4200	11.19	977948
570	2017-11-07	11.30	11.4200	10.6700	10.83	451210
571	2017-11-08	10.70	11.0600	10.3500	10.90	336449
572	2017-11-09	11.00	11.8563	10.9700	11.60	463067
573	2017-11-10	11.68	13.1500	11.3043	12.46	885587

574 rows × 6 columns

```
df['Volume'].unique()
```

```
            439114,   623643,   312896,   401342,   338423,   519097,
            480968,   542935,   365730,   289233,   470010,   574826,
            442423,   396027,   326975,   449719,   547317,   629254,
            746824,   747963,   597245,   280556,   423929,   381409,
            868937,   503069,   444336,   532095,   463156,   255061,
            346419,   381843,   275864,   220730,   161396,   458894,
            398093,   205665,   306629,   375915,   229628,   255815,
            282640,   301097,   204199,   699245,   577513,   296265,
            275703,   298255,   189363,   283755,   451162,   529024,
            670312,  1701597,   965224,   479991,   494926,   313832,
            317130,   266768,   251331,   430171,   538360,   325139,
            488716,   361134,   266165,   359273,   434241,   367133,
            314945,   494224,   312724,   821218,   407080,   296007,
            348435,   300478,   466061,   352090,   287749,   433604,
            242474,   244707,   298553,   447684,   188972,   348045,
            253279,   338354,   280977,   279040,   287164,   368570,
            417030,   404334,   602670,   475733,   280733,   257569,
            262735,   334357,   291003,   395154,   458950,  1616976,
            359149,   716435,   464360,   347517,   925534,   292268,
            283739,   177399,   275499,   283288,   370848,   526224,
            489290,   403144,   567735,   262238,   476676,   712476,
            369465,   367647,   753903,   523405,   596578,   439856,
           1077565,  1876740,   906874,   449217,   822704,  8998782,
           1678805,  4309150,  2320928,  1279757,  4253972,   914749,
           1673477,   651837,   423837,   486889,   765657,   747553,
            771350,   354753,   407243,   416635,   285956,   369405,
            257356,   179765,   292627,   588271,   344190,   315722,
            259959,   281103,   313352,   325800,   485416,   219485,
            288694,   220805,   562994,   383450,   345881,   342833,
          13632556,  3261715,  1829376,  1863068,  1275918,   672628,
            726409,   402038,   635145,   871267,   420984,   328513,
            287767,   442331,   198663,   303491,   299544,   480391,
            586296,   361461,   294371,   297751,   256138,   233242,
            301604,   323496,   531495,   977948,   451210,   336449,
            463067,   885587])
```

```
df.duplicated()
```

```
0      False
1      False
2      False
3      False
4      False
       ...
569    False
570    False
571    False
572    False
573    False
Length: 574, dtype: bool
```

```
df.duplicated().sum()
```

```
0
```

```
df.mean(axis = 0)
```

```
<ipython-input-27-df92b92bd195>:1: FutureWarning: DataFrame.mean and DataFrame.median with numeric_only=None will include datetime64 and datetime64tz columns in a future version.
  df.mean(axis = 0)
Open          13.584548
High          14.162170
```

```
Low           12.998530
Close         13.526147
Volume    363937.486063
dtype: float64
```

```
df.median(axis = 0)
```

```
<ipython-input-28-fce084e23a98>:1: FutureWarning: DataFrame.mean and DataFrame.median with numeric_only=None will include datetime64 and datetime64tz columns in a future version.
  df.median(axis = 0)
Open          12.805
High          13.385
Low           12.355
Close         12.795
Volume    214488.500
dtype: float64
```

```
df.mode(axis = 0)
```

	Date	Open	High	Low	Close	Volume
0	2015-08-05	6.20	6.4	6.2	9.45	54900.0
1	2015-08-06	6.26	NaN	12.8	NaN	140600.0
2	2015-08-07	9.60	NaN	NaN	NaN	NaN
3	2015-08-10	9.95	NaN	NaN	NaN	NaN
4	2015-08-11	14.50	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...
569	2017-11-06	NaN	NaN	NaN	NaN	NaN
570	2017-11-07	NaN	NaN	NaN	NaN	NaN
571	2017-11-08	NaN	NaN	NaN	NaN	NaN
572	2017-11-09	NaN	NaN	NaN	NaN	NaN
573	2017-11-10	NaN	NaN	NaN	NaN	NaN

574 rows × 6 columns

```
df.groupby(['Open','High'])['Close'].count().unstack()
```

High	5.400	6.000	6.010	6.030	6.040	6.050	6.070	6.160	6.170	6.180	...	31.690	32.000	33.060	34.720	36.000	36.300	37.688	39.500	42.850	43.000
Open																					
4.75	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5.40	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5.79	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

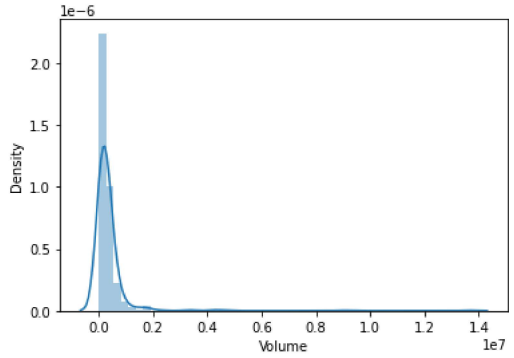
```
df.sort_values(by=['Volume'],ascending = True)
```

	Date	Open	High	Low	Close	Volume
97	2015-12-22	11.66	11.8790	11.1010	11.46	12654
124	2016-02-02	6.31	6.4800	6.1735	6.29	16609
127	2016-02-05	6.28	6.5599	6.1800	6.18	18832
19	2015-09-01	27.12	28.9400	26.6200	28.75	19600
113	2016-01-15	8.27	8.4400	7.7000	7.86	22392
...	...	...	...	...	...	...
510	2017-08-14	5.85	6.6900	5.4200	6.42	4253972
507	2017-08-09	6.27	8.4299	6.2000	7.60	4309150
152	2016-03-14	11.87	21.5626	11.8700	21.03	4681915
505	2017-08-07	6.56	6.7700	5.9301	6.67	8998782
542	2017-09-28	11.05	11.2800	9.3800	9.44	13632556

574 rows × 6 columns

```
sns.distplot(df.Volume)
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your coc
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f0ed9f40b80>
```




```
df.skew(axis = 0, skipna = True)
```

```
<ipython-input-33-c789b15bf834>:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise Ty
  df.skew(axis = 0, skipna = True)
Open       0.891791
High       1.087728
Low        0.760817
Close      0.863232
Volume    10.986774
dtype: float64
```

```
df.kurtosis(skipna = True)
```

```
<ipython-input-34-b6a0fc7ed35a>:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise Ty
  df.kurtosis(skipna = True)
Open        0.766037
High        1.734913
Low         0.145212
Close       0.573069
Volume    155.195548
dtype: float64
```

```
df.kurtosis()
```

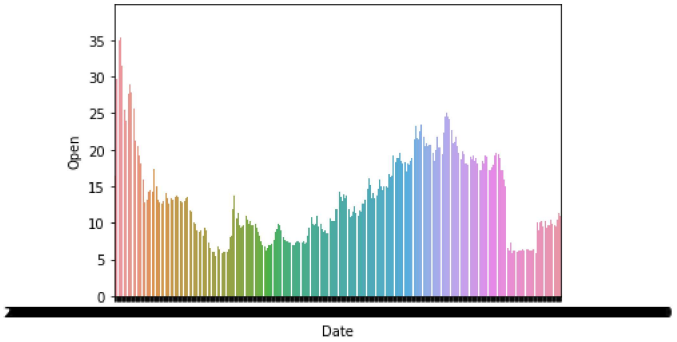
```
<ipython-input-35-c7edf97eb14c>:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise Ty
  df.kurtosis()
Open        0.766037
High        1.734913
Low         0.145212
Close       0.573069
Volume    155.195548
dtype: float64
```

DATA VISUALIZATION

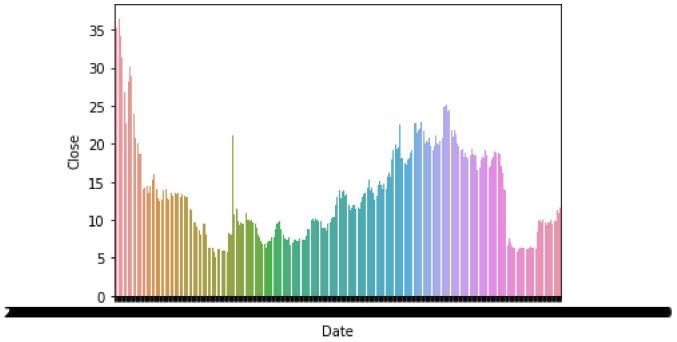
```
corr=df.corr()
sns.heatmap(corr,annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0ed9e642b0>
```

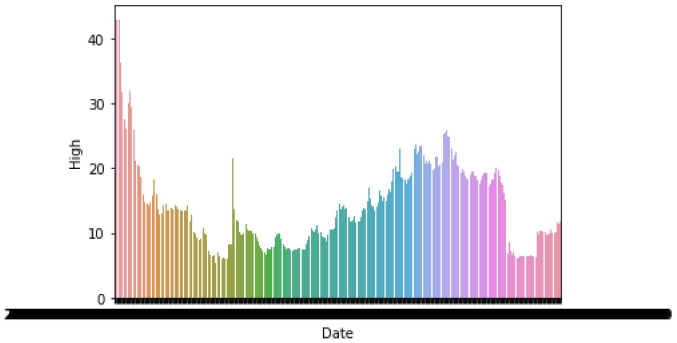
```
sns.barplot(x='Date', y='Open',data=df, )
plt.show()
```



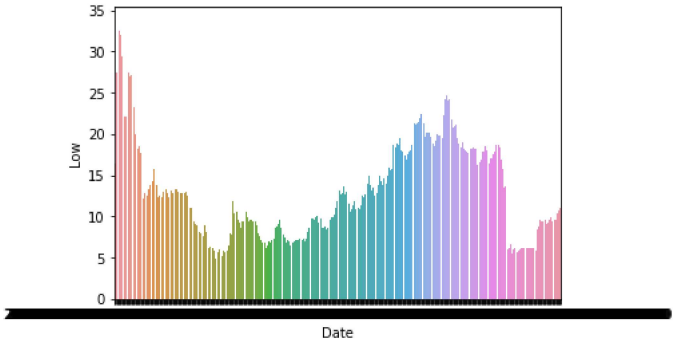

```
sns.barplot(x='Date', y='Close',data=df, )
plt.show()
```




```
sns.barplot(x='Date', y='High',data=df, )
plt.show()
```

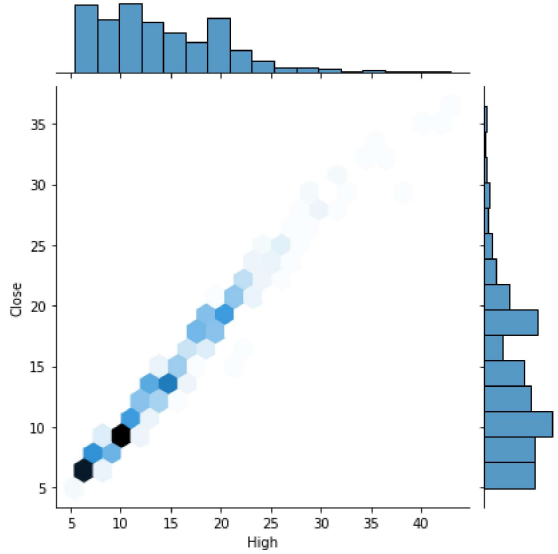


```
sns.barplot(x='Date', y='Low',data=df, )
plt.show()
```



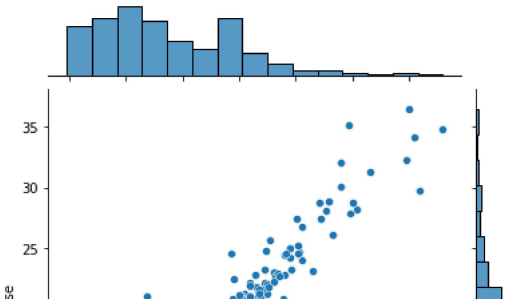

```
sns.jointplot(x = 'High', y = 'Close', data = df, kind = 'hex', gridsize = 20)
```

```
<seaborn.axisgrid.JointGrid at 0x7f0ed6dda610>
```



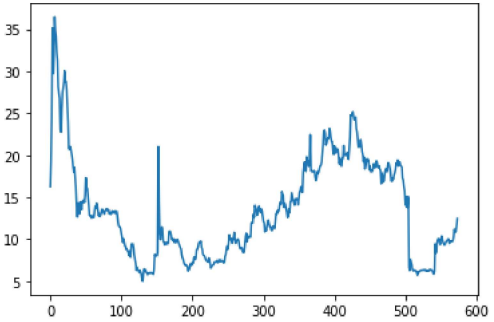

```
sns.jointplot(x="Open", y="Close", data=df)
```

```
<seaborn.axisgrid.JointGrid at 0x7f0ed5edbfa0>
```



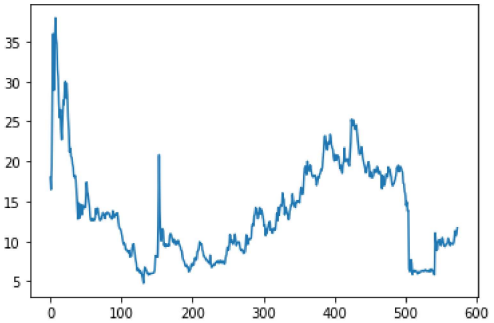

```
df['Close'].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0ed6b65640>
```




```
df['Open'].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0ed6673af0>
```

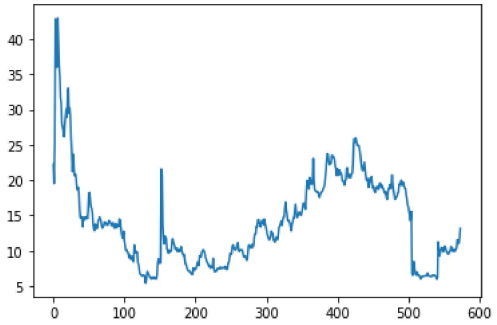



```
df['Low'].plot()
```

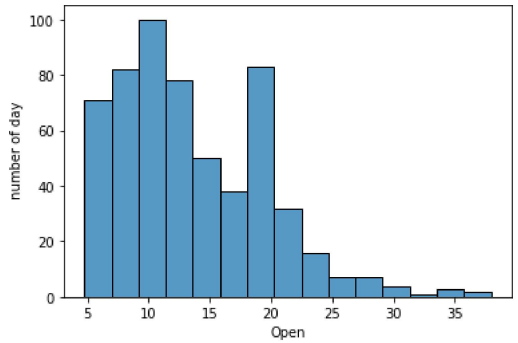
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0ed6b3b400>
```

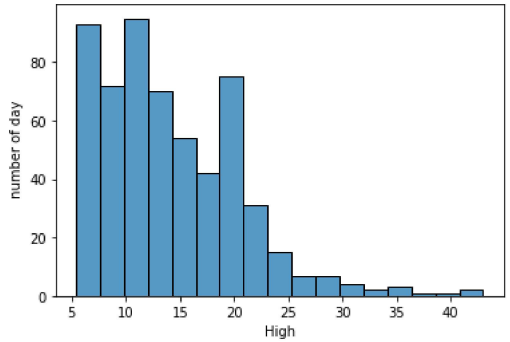

```
df['High'].plot()
```

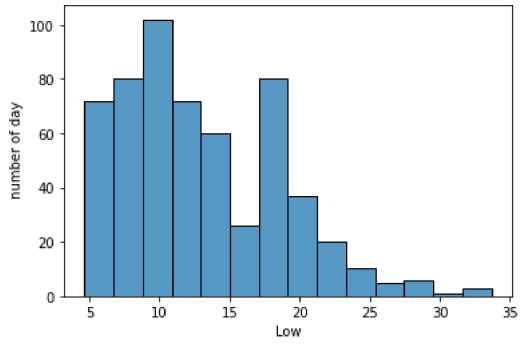
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0ed671c0a0>
```



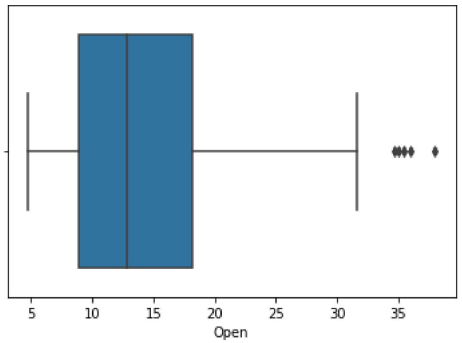

```
for i in range(0,5):
       sns.histplot(x = features[i],data = df)
       plt.ylabel(" number of day")
       plt.show()
```

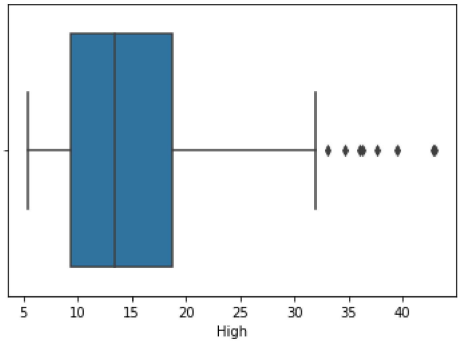


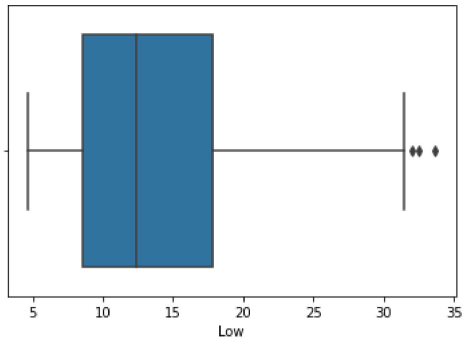


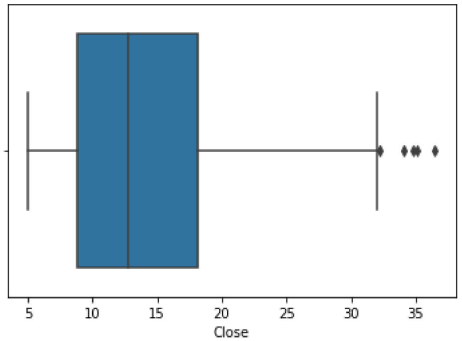



```
for i in range(0,5):
       sns.boxplot(x = features[i],data = df)
       plt.show()
```


5
10
15
20
25
30
35
Open

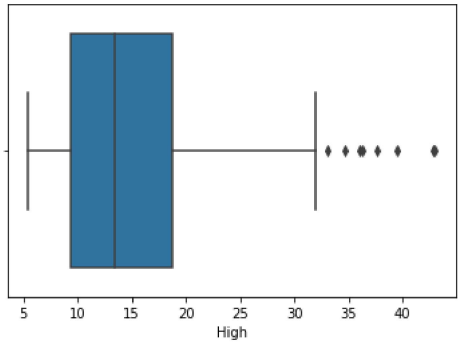

5
10
15
20
25
30
35
40
High


5
10
15
20
25
30
35
Low


5
10
15
20
25
30
35
Close

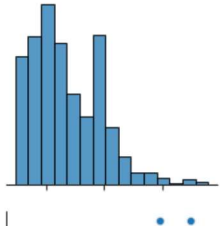
```
sns.boxplot(x="High",data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0ed6dab910>
```




```
sns.pairplot(df,corner=True)
```

```
<seaborn.axisgrid.PairGrid at 0x7f0ed65565b0>
```



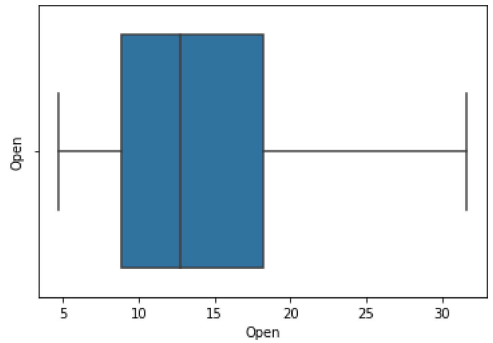
FEATURE ENGINEERING

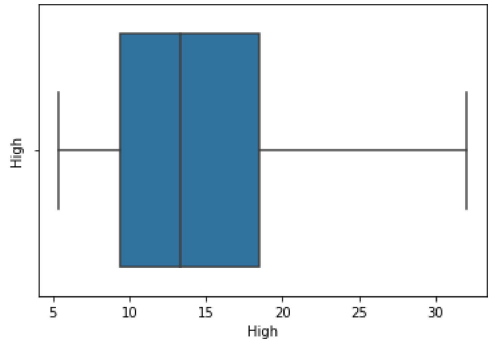
```
for col in features:
  Q1 = df[col].quantile(0.25)
  Q3 = df[col].quantile(0.75)
  IQR = Q3 - Q1
  lower_limit = Q1 - 1.5 * IQR
  upper_limit = Q3 + 1.5 * IQR
  df=df[(df[col] > lower_limit) & (df[col] < upper_limit)]
#after removing outliers
df
```

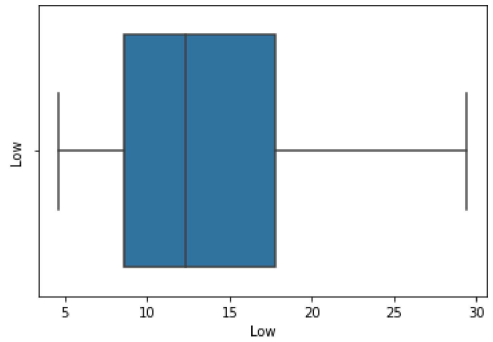
	Date	Open	High	Low	Close	Volume
1	2015-08-06	16.48	19.5000	16.46	19.32	331692
2	2015-08-07	19.31	24.9000	19.31	24.54	269300
10	2015-08-19	31.60	31.6900	29.38	31.28	119800
11	2015-08-20	30.43	31.0000	28.00	28.18	156400
12	2015-08-21	27.23	27.9400	25.27	27.39	140600
...	...	...	...	...	...	...
567	2017-11-02	9.59	10.2000	9.51	9.85	323496
568	2017-11-03	9.83	10.5000	9.83	10.33	531495
570	2017-11-07	11.30	11.4200	10.67	10.83	451210
571	2017-11-08	10.70	11.0600	10.35	10.90	336449
572	2017-11-09	11.00	11.8563	10.97	11.60	463067

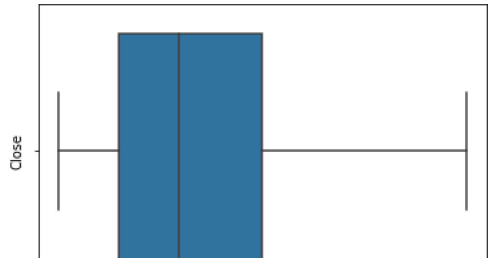
530 rows × 6 columns

```
for i in range(0,4):
       sns.boxplot(x = features[i],data = df)
       plt.ylabel(features[i])
       plt.show()
```










```
df = df.fillna({
    "Open": df["Open"].mean(),
    "Close":df["Close"].mean(),
    "Low":df["Low"].mean(),
    "High":df["High"].mean(),
```

```
})
```

```
df
```

	Date	Open	High	Low	Close	Volume
1	2015-08-06	16.48	19.5000	16.46	19.32	331692
2	2015-08-07	19.31	24.9000	19.31	24.54	269300
10	2015-08-19	31.60	31.6900	29.38	31.28	119800
11	2015-08-20	30.43	31.0000	28.00	28.18	156400
12	2015-08-21	27.23	27.9400	25.27	27.39	140600
...	...	...	...	...	...	...
567	2017-11-02	9.59	10.2000	9.51	9.85	323496
568	2017-11-03	9.83	10.5000	9.83	10.33	531495
570	2017-11-07	11.30	11.4200	10.67	10.83	451210
571	2017-11-08	10.70	11.0600	10.35	10.90	336449
572	2017-11-09	11.00	11.8563	10.97	11.60	463067

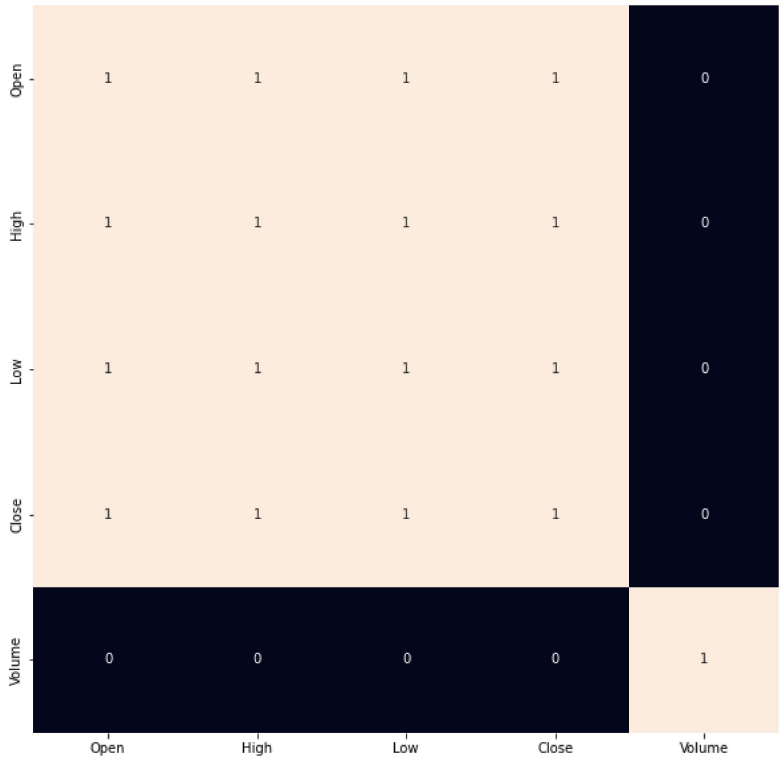
530 rows × 6 columns

```
df.describe()
df
```

	Date	Open	High	Low	Close	Volume
1	2015-08-06	16.48	19.5000	16.46	19.32	331692
2	2015-08-07	19.31	24.9000	19.31	24.54	269300
10	2015-08-19	31.60	31.6900	29.38	31.28	119800
11	2015-08-20	30.43	31.0000	28.00	28.18	156400
12	2015-08-21	27.23	27.9400	25.27	27.39	140600
...	...	...	...	...	...	...
567	2017-11-02	9.59	10.2000	9.51	9.85	323496
568	2017-11-03	9.83	10.5000	9.83	10.33	531495
570	2017-11-07	11.30	11.4200	10.67	10.83	451210
571	2017-11-08	10.70	11.0600	10.35	10.90	336449
572	2017-11-09	11.00	11.8563	10.97	11.60	463067

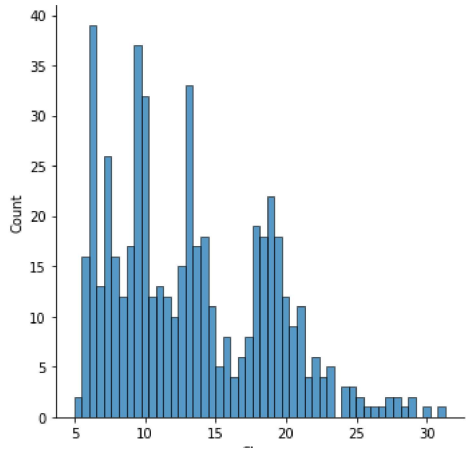
530 rows × 6 columns

```
#/*
#splitted = df['Date'].str.split('-', expand=True)
#df['day'] = splitted[2].astype('int')
#df['month'] = splitted[1].astype('int')
#df['year'] = splitted[0].astype('int')
#df


#encoding
#df['open-close']  = df['Open'] - df['Close']
#df['low-high']  = df['Low'] - df['High']
#df['target'] = np.where(df['Close'].shift(-1) > df['Close'], 1, 0)


#plt.pie(df['target'].value_counts().values,
#        labels=[0, 1], autopct='%1.1f%%')
#plt.show()


plt.figure(figsize=(10, 10))
sns.heatmap(df.corr() > 0.9, annot=True, cbar=False)
plt.show()
```




```
#Transform
sns.displot(df,x='Close',bins=50)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f0ed63d3d60>
```




```
df['Close'].skew()
```

```
0.5987071339793048
```

```
df
```

	Date	Open	High	Low	Close	Volume
1	2015-08-06	16.48	19.5000	16.46	19.32	331692
2	2015-08-07	19.31	24.9000	19.31	24.54	269300
10	2015-08-19	31.60	31.6900	29.38	31.28	119800
11	2015-08-20	30.43	31.0000	28.00	28.18	156400
12	2015-08-21	27.23	27.9400	25.27	27.39	140600
...	...	...	...	...	...	...
567	2017-11-02	9.59	10.2000	9.51	9.85	323496
568	2017-11-03	9.83	10.5000	9.83	10.33	531495
570	2017-11-07	11.30	11.4200	10.67	10.83	451210
571	2017-11-08	10.70	11.0600	10.35	10.90	336449
572	2017-11-09	11.00	11.8563	10.97	11.60	463067

530 rows × 6 columns

```
#scaling
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
feature_transform = scaler.fit_transform(df[features])
feature_transform= pd.DataFrame(columns=features, data=feature_transform, index=df.index)
feature_transform.dropna(axis=0,inplace=True)
feature_transform
```

	Open	High	Low	Close	Volume
1	0.436872	0.530075	0.477769	0.545074	0.414681
2	0.542272	0.733083	0.592967	0.743629	0.333585
10	1.000000	0.988346	1.000000	1.000000	0.139267
11	0.956425	0.962406	0.944220	0.882084	0.186839
12	0.837244	0.847368	0.833872	0.852035	0.166302
...	...	...	...	...	...
567	0.180261	0.180451	0.196847	0.184861	0.404028
568	0.189199	0.191729	0.209782	0.203119	0.674382
570	0.243948	0.226316	0.243735	0.222138	0.570029
571	0.221601	0.212782	0.230800	0.224800	0.420864
572	0.232775	0.242718	0.255861	0.251426	0.585440

530 rows × 5 columns

feature_transform

	Open	High	Low	Close	Volume
1	0.436872	0.530075	0.477769	0.545074	0.414681
2	0.542272	0.733083	0.592967	0.743629	0.333585
10	1.000000	0.988346	1.000000	1.000000	0.139267
11	0.956425	0.962406	0.944220	0.882084	0.186839
12	0.837244	0.847368	0.833872	0.852035	0.166302
...	...	...	...	...	...
567	0.180261	0.180451	0.196847	0.184861	0.404028
568	0.189199	0.191729	0.209782	0.203119	0.674382
570	0.243948	0.226316	0.243735	0.222138	0.570029
571	0.221601	0.212782	0.230800	0.224800	0.420864
572	0.232775	0.242718	0.255861	0.251426	0.585440

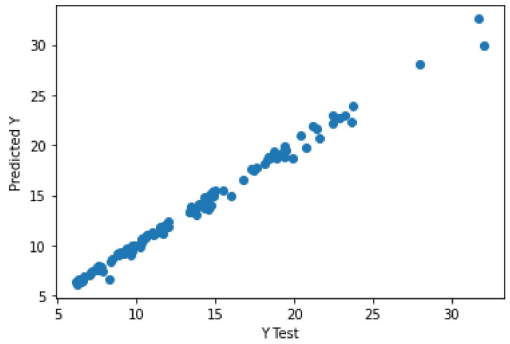
530 rows × 5 columns

1/2 BASE MODEL

```
X=df['Open']
y=df['High']
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=101)
X_train, X_test = X_train.values, X_test.values
X_train= X_train.reshape(-1, 1)
X_test = X_test.reshape(-1, 1)
```

```
from sklearn.linear_model import LinearRegression
lm=LinearRegression()
lm.fit(X_train,y_train)
print("Coefficients: ",lm.coef_)
predictions=lm.predict(X_test)
plt.scatter(y_test,predictions)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```

```
Coefficients:  [1.02798556]
Text(0, 0.5, 'Predicted Y')
```

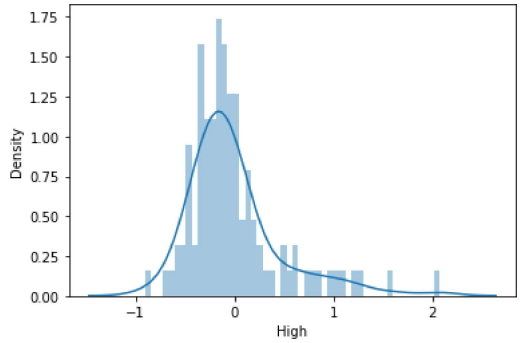



```
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MAE: 0.33153766002287127
MSE: 0.22640290570167013
RMSE: 0.4758181435187924
```

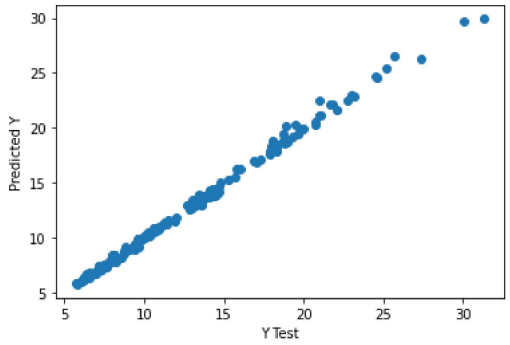
```
sns.distplot((y_test-predictions),bins=50)
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your coc
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f0ed5d700d0>
```



```
y = df['Close']
X = df[['Open', 'High', 'Low', 'Volume']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
lm.fit(X_train,y_train)
print('Coefficients: \n', lm.coef_)
predictions = lm.predict(X_test)
plt.scatter(y_test,predictions)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```

```
Coefficients:
 [-4.76365802e-01  7.68492743e-01  7.01840928e-01  1.92569141e-07]
Text(0, 0.5, 'Predicted Y')
```

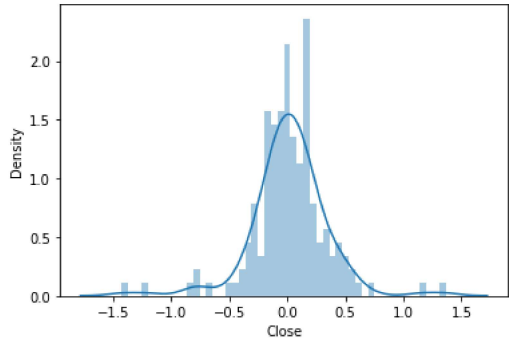



```
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MAE: 0.2209430197533412
MSE: 0.10712848485221717
RMSE: 0.3273048805811138
```

```
sns.distplot((y_test-predictions),bins=50)
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your coc
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f0ed9f0d2e0>
```



```
y_test.mean()
```

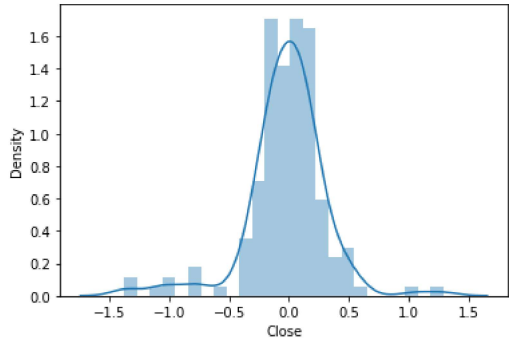
```
13.203206918238996
```

```
predictions.mean()
```

```
13.186245132147656
```

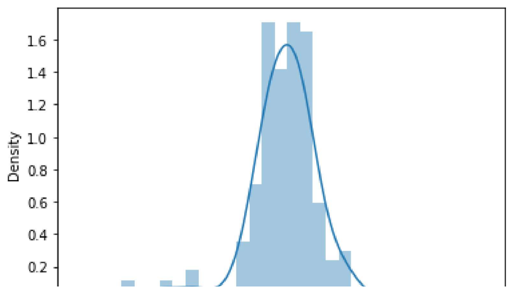
```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
ridge=Ridge()
parameters={'alpha':[1e-15,1e-10,1e-8,1e-3,1e-2,1,5,10,20,30,35,40,45,50,55,100]}
ridge_regressor=GridSearchCV(ridge,parameters,scoring='neg_mean_squared_error',cv=5)
ridge_regressor.fit(X,y)
print(ridge_regressor.best_params_)
print(ridge_regressor.best_score_)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
prediction_ridge=ridge_regressor.predict(X_test)
sns.distplot(y_test-prediction_ridge)
```

```
{'alpha': 1}
-0.10491537864394454
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your coc
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f0ed690dbe0>
```




```
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
lasso=Lasso()
parameters={'alpha':[1e-15,1e-10,1e-8,1e-3,1e-2,1,5,10,20,30,35,40,45,50,55,100]}
lasso_regressor=GridSearchCV(lasso,parameters,scoring='neg_mean_squared_error',cv=5)
lasso_regressor.fit(X,y)
print(lasso_regressor.best_params_)
print(lasso_regressor.best_score_)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
prediction_ridge=ridge_regressor.predict(X_test)
sns.distplot(y_test-prediction_ridge)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterat
  model = cd_fast.enet_coordinate_descent(
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterat
  model = cd_fast.enet_coordinate_descent(
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterat
  model = cd_fast.enet_coordinate_descent(
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterat
  model = cd_fast.enet_coordinate_descent(
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterat
  model = cd_fast.enet_coordinate_descent(
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterat
  model = cd_fast.enet_coordinate_descent(
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterat
  model = cd_fast.enet_coordinate_descent(
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterat
  model = cd_fast.enet_coordinate_descent(
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterat
  model = cd_fast.enet_coordinate_descent(
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterat
  model = cd_fast.enet_coordinate_descent(
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterat
  model = cd_fast.enet_coordinate_descent(
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterat
  model = cd_fast.enet_coordinate_descent(
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterat
  model = cd_fast.enet_coordinate_descent(
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterat
  model = cd_fast.enet_coordinate_descent(
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterat
  model = cd_fast.enet_coordinate_descent(
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterat
  model = cd_fast.enet_coordinate_descent(
{'alpha': 0.001}
-0.105263515485685
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your coc
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f0ed7080340>
```




```
df.drop(['Date'],axis=1,inplace=True)
df
```

	Open	High	Low	Close	Volume
1	16.48	19.5000	16.46	19.32	331692
2	19.31	24.9000	19.31	24.54	269300
10	31.60	31.6900	29.38	31.28	119800
11	30.43	31.0000	28.00	28.18	156400
12	27.23	27.9400	25.27	27.39	140600
...	...	...	...	...	...
567	9.59	10.2000	9.51	9.85	323496

Review 2

```
ans=[]
for i in range(530):
    if(df.iloc[i,3]>=df.iloc[i,0]):
        ans.append(1)
    else:
        ans.append(0)
df["increase value"]=ans
```

```
df
```

	Open	High	Low	Close	Volume	increase value
1	16.48	19.5000	16.46	19.32	331692	1
2	19.31	24.9000	19.31	24.54	269300	1
10	31.60	31.6900	29.38	31.28	119800	0
11	30.43	31.0000	28.00	28.18	156400	0
12	27.23	27.9400	25.27	27.39	140600	1
...	...	...	...	...	...	...
567	9.59	10.2000	9.51	9.85	323496	1
568	9.83	10.5000	9.83	10.33	531495	1
570	11.30	11.4200	10.67	10.83	451210	0
571	10.70	11.0600	10.35	10.90	336449	1
572	11.00	11.8563	10.97	11.60	463067	1

530 rows × 6 columns

```
from sklearn.model_selection import train_test_split

x = df.drop(['increase value'], axis=1)
y = df['increase value']

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

Decision tree

```
print(x_train.shape,y_train.shape)
```

```
(424, 5) (424,)
```

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(x_train,y_train)
y_pred1 = model.predict(x_test)
print(y_pred1)
```

```
[1 0 1 0 0 1 0 0 1 1 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 1 0 0 0
 0 0 0 0 0 1 0 1 1 1 0 0 0 0 0 1 1 0 1 1 0 0 1 1 0 1 0 1 0 0 1 1 1 1 0 0 0
 0 1 0 1 1 1 0 1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 0 1 0 1 1 1 0 1 1 0]
```

```
result = pd.DataFrame({'actual' : y_test ,'predicted' : y_pred1})
result
```

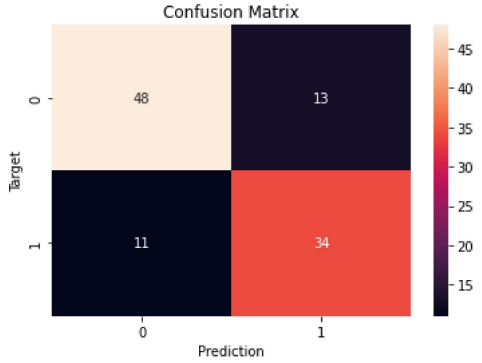
	actual	predicted
485	1	1
319	0	0
452	1	1
329	0	0
324	0	0
...	...	...
101	0	1
214	0	0
76	0	1
524	1	1
124	0	0

106 rows × 2 columns

```
from sklearn.metrics import confusion_matrix
cf = confusion_matrix(y_test, y_pred1)
plt.figure()
sns.heatmap(cf, annot=True)
plt.xlabel('Prediction')
```

```
plt.ylabel('Target')
plt.title('Confusion Matrix')
```

```
Text(0.5, 1.0, 'Confusion Matrix')
```




```
from sklearn.tree import plot_tree
plt.figure(figsize=(30,50))
plot_tree(model,class_names=['0','1'], feature_names=x.columns,filled=True,fontsize=15)
```

```
Text(0.24253144654408805, 0.8214285714285714, 'Volume <= 59261.0\ngini = 0.491\nsamples = 222\nvalue = [126, 96]\nclass = 0'),
Text(0.10062893081761007, 0.75, 'High <= 9.53\ngini = 0.367\nsamples = 33\nvalue = [25, 8]\nclass = 0'),
Text(0.0880503144654088, 0.6785714285714286, 'Close <= 8.88\ngini = 0.435\nsamples = 25\nvalue = [17, 8]\nclass = 0'),
Text(0.07547169811320754, 0.6071428571428571, 'Volume <= 41519.5\ngini = 0.386\nsamples = 23\nvalue = [17, 6]\nclass = 0'),
Text(0.06289308176100629, 0.5357142857142857, 'Volume <= 38203.0\ngini = 0.469\nsamples = 16\nvalue = [10, 6]\nclass = 0'),
Text(0.03773584905660377, 0.4642857142857143, 'Volume <= 28562.5\ngini = 0.32\nsamples = 10\nvalue = [8, 2]\nclass = 0'),
Text(0.025157232704402517, 0.39285714285714285, 'Volume <= 26551.5\ngini = 0.48\nsamples = 5\nvalue = [3, 2]\nclass = 0'),
Text(0.012578616352201259, 0.32142857142857145, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]\nclass = 0'),
Text(0.03773584905660377, 0.32142857142857145, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]\nclass = 1'),
Text(0.050314465408805034, 0.39285714285714285, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]\nclass = 0'),
Text(0.0880503144654088, 0.4642857142857143, 'Volume <= 40196.0\ngini = 0.444\nsamples = 6\nvalue = [2, 4]\nclass = 1'),
Text(0.07547169811320754, 0.39285714285714285, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]\nclass = 1'),
Text(0.10062893081761007, 0.39285714285714285, 'Close <= 8.515\ngini = 0.444\nsamples = 3\nvalue = [2, 1]\nclass = 0'),
Text(0.0880503144654088, 0.32142857142857145, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]\nclass = 0'),
Text(0.11320754716981132, 0.32142857142857145, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]\nclass = 1'),
Text(0.0880503144654088, 0.5357142857142857, 'gini = 0.0\nsamples = 7\nvalue = [7, 0]\nclass = 0'),
Text(0.10062893081761007, 0.6071428571428571, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]\nclass = 1'),
Text(0.11320754716981132, 0.6785714285714286, 'gini = 0.0\nsamples = 8\nvalue = [8, 0]\nclass = 0'),
Text(0.38443396226415094, 0.75, 'Open <= 12.775\ngini = 0.498\nsamples = 189\nvalue = [101, 88]\nclass = 0'),
Text(0.3718553459119497, 0.6785714285714286, 'Close <= 9.765\ngini = 0.499\nsamples = 183\nvalue = [95, 88]\nclass = 0'),
Text(0.26257861635220126, 0.6071428571428571, 'Open <= 9.67\ngini = 0.482\nsamples = 121\nvalue = [72, 49]\nclass = 0'),
Text(0.25, 0.5357142857142857, 'Close <= 6.335\ngini = 0.498\nsamples = 105\nvalue = [56, 49]\nclass = 0'),
Text(0.1761006289308176, 0.4642857142857143, 'Open <= 6.17\ngini = 0.393\nsamples = 26\nvalue = [19, 7]\nclass = 0'),
Text(0.1509433962264151, 0.39285714285714285, 'Close <= 5.945\ngini = 0.48\nsamples = 10\nvalue = [4, 6]\nclass = 1'),
Text(0.13836477987421383, 0.32142857142857145, 'Open <= 5.075\ngini = 0.32\nsamples = 5\nvalue = [4, 1]\nclass = 0'),
Text(0.12578616352201258, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]\nclass = 1'),
Text(0.1509433962264151, 0.25, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]\nclass = 0'),
Text(0.16352201257861634, 0.32142857142857145, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]\nclass = 1'),
Text(0.20125786163522014, 0.39285714285714285, 'High <= 6.415\ngini = 0.117\nsamples = 16\nvalue = [15, 1]\nclass = 0'),
Text(0.18867924528301888, 0.32142857142857145, 'Close <= 6.255\ngini = 0.32\nsamples = 5\nvalue = [4, 1]\nclass = 0'),
Text(0.1761006289308176, 0.25, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]\nclass = 0'),
Text(0.20125786163522014, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]\nclass = 1'),
Text(0.2138364779874214, 0.32142857142857145, 'gini = 0.0\nsamples = 11\nvalue = [11, 0]\nclass = 0'),
Text(0.3238993710691824, 0.4642857142857143, 'Open <= 7.39\ngini = 0.498\nsamples = 79\nvalue = [37, 42]\nclass = 1'),
Text(0.25157232704402516, 0.39285714285714285, 'Open <= 6.455\ngini = 0.367\nsamples = 33\nvalue = [8, 25]\nclass = 1'),
Text(0.2389937106918239, 0.32142857142857145, 'gini = 0.0\nsamples = 11\nvalue = [0, 11]\nclass = 1'),
Text(0.2641509433962264, 0.32142857142857145, 'Close <= 6.95\ngini = 0.463\nsamples = 22\nvalue = [8, 14]\nclass = 1'),
Text(0.2389937106918239, 0.25, 'Volume <= 74201.0\ngini = 0.245\nsamples = 7\nvalue = [6, 1]\nclass = 0'),
Text(0.22641509433962265, 0.17857142857142858, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]\nclass = 1'),
Text(0.25157232704402516, 0.17857142857142858, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]\nclass = 0'),
Text(0.2893081761006289, 0.25, 'Close <= 7.19\ngini = 0.231\nsamples = 15\nvalue = [2, 13]\nclass = 1'),
Text(0.27672955974842767, 0.17857142857142858, 'Open <= 7.08\ngini = 0.48\nsamples = 5\nvalue = [2, 3]\nclass = 1'),
Text(0.2641509433962264, 0.10714285714285714, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]\nclass = 1'),
Text(0.2893081761006289, 0.10714285714285714, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]\nclass = 0'),
Text(0.30188679245283020, 0.17857142857142858, 'gini = 0.0\nsamples = 10\nvalue = [0, 10]\nclass = 1'),
Text(0.39622641509433965, 0.39285714285714285, 'Close <= 9.055\ngini = 0.466\nsamples = 46\nvalue = [29, 17]\nclass = 0'),
Text(0.36477987421383645, 0.32142857142857145, 'Volume <= 229950.5\ngini = 0.312\nsamples = 31\nvalue = [25, 6]\nclass = 0'),
Text(0.3522012578616352, 0.25, 'Close <= 8.17\ngini = 0.238\nsamples = 29\nvalue = [25, 4]\nclass = 0'),
Text(0.3270440251572327, 0.17857142857142858, 'Volume <= 90005.5\ngini = 0.1\nsamples = 19\nvalue = [18, 1]\nclass = 0'),
Text(0.31446540880503143, 0.10714285714285714, 'Volume <= 85127.5\ngini = 0.32\nsamples = 5\nvalue = [4, 1]\nclass = 0'),
Text(0.30188679245283020, 0.03571428571428571, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]\nclass = 0'),
Text(0.3270440251572327, 0.03571428571428571, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]\nclass = 1'),
Text(0.33962264150943394, 0.10714285714285714, 'gini = 0.0\nsamples = 14\nvalue = [14, 0]\nclass = 0'),
Text(0.37735849056603776, 0.17857142857142858, 'Open <= 8.5\ngini = 0.42\nsamples = 10\nvalue = [7, 3]\nclass = 0'),
Text(0.36477987421383645, 0.10714285714285714, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]\nclass = 1'),
Text(0.389937106918239, 0.10714285714285714, 'gini = 0.0\nsamples = 7\nvalue = [7, 0]\nclass = 0'),
Text(0.37735849056603776, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]\nclass = 1'),
Text(0.4276729559748428, 0.32142857142857145, 'Open <= 9.355\ngini = 0.391\nsamples = 15\nvalue = [4, 11]\nclass = 1'),
Text(0.41509433962264153, 0.25, 'gini = 0.0\nsamples = 7\nvalue = [0, 7]\nclass = 1'),
Text(0.44025157232704404, 0.25, 'Close <= 9.565\ngini = 0.5\nsamples = 8\nvalue = [4, 4]\nclass = 0'),
Text(0.4276729559748428, 0.17857142857142858, 'Low <= 9.33\ngini = 0.32\nsamples = 5\nvalue = [4, 1]\nclass = 0'),
Text(0.41509433962264153, 0.10714285714285714, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]\nclass = 0'),
Text(0.44025157232704404, 0.10714285714285714, 'Close <= 9.47\ngini = 0.5\nsamples = 2\nvalue = [1, 1]\nclass = 0'),
Text(0.4276729559748428, 0.03571428571428571, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]\nclass = 1'),
```

```
Text(0.4528301886792453, 0.03571428571428571, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]\nclass = 0'),
```

```
from sklearn.metrics import accuracy_score
print("Accuracy of Decision tree:",accuracy_score(y_test,y_pred1))
```

```
Accuracy of Decision tree: 0.7735849056603774
```

```
from sklearn.metrics import classification_report
print("Decision tree:",classification_report(y_test, y_pred1))
```

```
Decision tree:               precision    recall  f1-score   support

           0       0.81      0.79      0.80        61
           1       0.72      0.76      0.74        45

    accuracy                           0.77       106
   macro avg       0.77      0.77      0.77       106
weighted avg       0.78      0.77      0.77       106
```

Random forest

```
Text(0.5974842767295597, 0.39285714285714285, 'gini = 0.0\nsamples = 6\nvalue = [0, 6]\nclass = 1')
```

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(x_train, y_train)
y_pred2 = rf.predict(x_test)
y_pred2
```

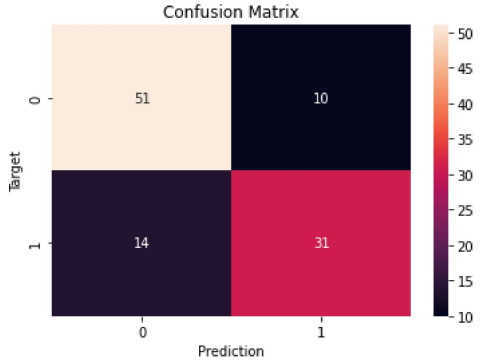
```
array([1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0,
       0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0])
Text(0.6540880503144654, 0.6071428571428571, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]\nclass = 1'),
```

```
result = pd.DataFrame({'actual' : y_test ,'predicted' : y_pred2})
result
```

	actual	predicted
485	1	1

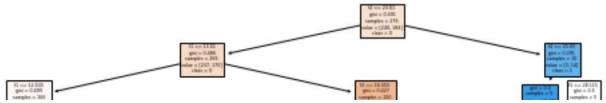
```
from sklearn.metrics import confusion_matrix
cf = confusion_matrix(y_test, y_pred2)
plt.figure()
sns.heatmap(cf, annot=True)
plt.xlabel('Prediction')
plt.ylabel('Target')
plt.title('Confusion Matrix')
```

```
Text(0.5, 1.0, 'Confusion Matrix')
```




```
Text(0.9622641509433962, 0.4642857142857143, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]\nclass = 0'),
```

```
plt.figure(figsize=(20, 10))
plot_tree(rf.estimators_[0], filled=True, feature_names=['f' + str(i) for i in range(1, 11)], class_names=['0', '1'])
plt.show()
```



```
print("Accuracy of random forest:",accuracy_score(y_test,y_pred2))
```

```
Accuracy of random forest: 0.7735849056603774
```

```
print("random forest:",classification_report(y_test, y_pred2))
```

```
random forest:               precision    recall  f1-score   support

           0       0.78      0.84      0.81        61
           1       0.76      0.69      0.72        45

    accuracy                           0.77       106
   macro avg       0.77      0.76      0.77       106
weighted avg       0.77      0.77      0.77       106
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(x_train, y_train)
y_pred3= knn.predict(x_test)
y_pred3
```

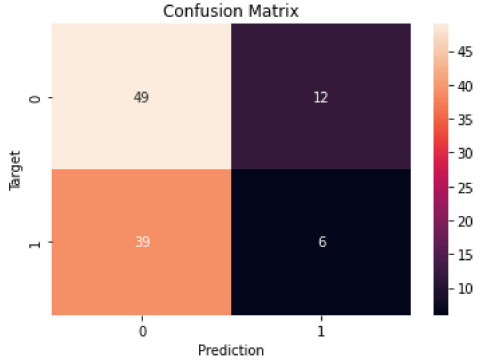
```
array([0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
result = pd.DataFrame({'actual' : y_test ,'predicted' : y_pred3})
result
```

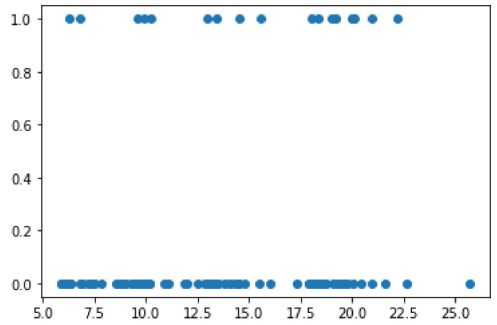
	actual	predicted
485	1	0
319	0	1

```
from sklearn.metrics import confusion_matrix
cf = confusion_matrix(y_test, y_pred3)
plt.figure()
sns.heatmap(cf, annot=True)
plt.xlabel('Prediction')
plt.ylabel('Target')
plt.title('Confusion Matrix')
```

```
Text(0.5, 1.0, 'Confusion Matrix')
```

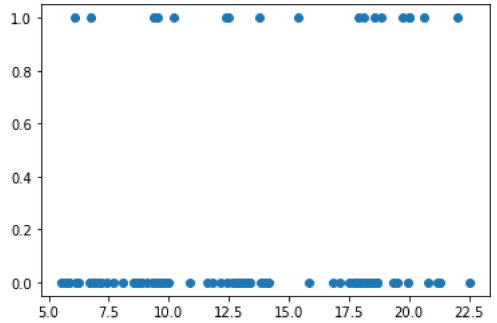



```
plt.scatter(x_test['Close'], y_pred3)
plt.show()
```

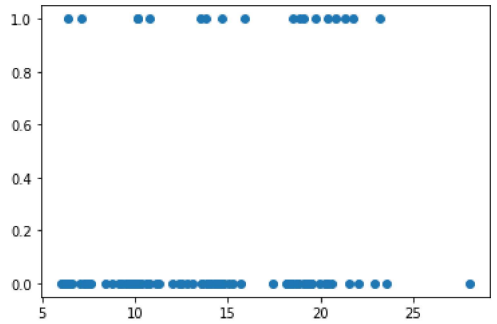



```
plt.scatter(x_test['Open'], y_pred3)
plt.show()
```


```
plt.scatter(x_test['Low'], y_pred3)
plt.show()
```




```
plt.scatter(x_test['High'], y_pred3)
plt.show()
```




```
print("Accuracy of KNN:",accuracy_score(y_test,y_pred3))
```

```
Accuracy of KNN: 0.5188679245283019
```

```
print("KNN:",classification_report(y_test, y_pred3))
```

```
KNN:               precision    recall  f1-score   support

           0       0.56      0.80      0.66        61
           1       0.33      0.13      0.19        45

    accuracy                           0.52       106
   macro avg       0.45      0.47      0.42       106
weighted avg       0.46      0.52      0.46       106
```

Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
nb=GaussianNB()
nb.fit(x_train,y_train)
y_pred4=nb.predict(x_test)
y_pred4
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

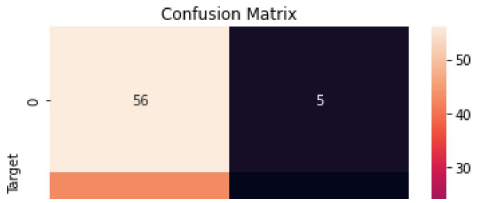
```
result = pd.DataFrame({'actual' : y_test ,'predicted' : y_pred4})
result
```

	actual	predicted
485	1	0
319	0	0
452	1	0
329	0	0
324	0	0
...	...	...
101	0	0
214	0	0
76	0	0
524	1	0
124	0	0

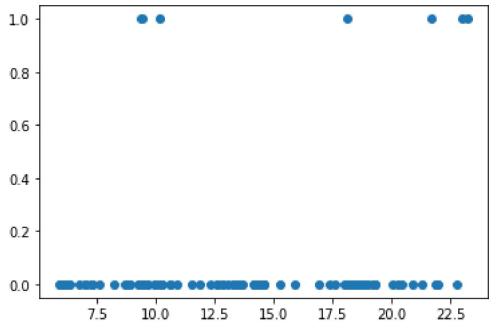
106 rows × 2 columns

```
from sklearn.metrics import confusion_matrix
cf = confusion_matrix(y_test, y_pred4)
plt.figure()
sns.heatmap(cf, annot=True)
plt.xlabel('Prediction')
plt.ylabel('Target')
plt.title('Confusion Matrix')
```

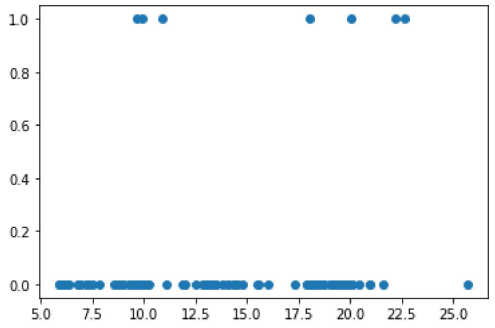
```
Text(0.5, 1.0, 'Confusion Matrix')
```




```
plt.scatter(x_test['Open'], y_pred4)
plt.show()
```

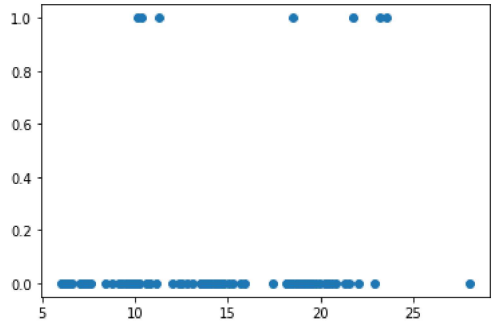



```
plt.scatter(x_test['Close'], y_pred4)
plt.show()
```




```
plt.scatter(x_test['Low'], y_pred4)
plt.show()
```


```
plt.scatter(x_test['High'], y_pred4)
plt.show()
```




```
print("Accuracy of Naive Bayes:",accuracy_score(y_test,y_pred4))
```

```
Accuracy of Naive Bayes: 0.5471698113207547
```

```
print("KNN:",classification_report(y_test, y_pred4))
```

```
KNN:               precision    recall  f1-score   support

           0       0.57      0.92      0.70        61
           1       0.29      0.04      0.08        45

    accuracy                           0.55       106
   macro avg       0.43      0.48      0.39       106
weighted avg       0.45      0.55      0.44       106
```

SVM

```
from sklearn.svm import SVC
svc = SVC()
svc.fit(x_train,y_train)
y_pred5 = svc.predict(x_test)
y_pred5
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

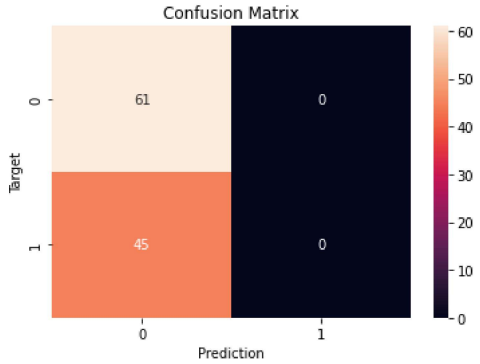
```
result = pd.DataFrame({'actual' : y_test ,'predicted' : y_pred5})
result
```

	actual	predicted
485	1	0
319	0	0
452	1	0
329	0	0
324	0	0
...	...	...
101	0	0
214	0	0
76	0	0
524	1	0
124	0	0

106 rows × 2 columns

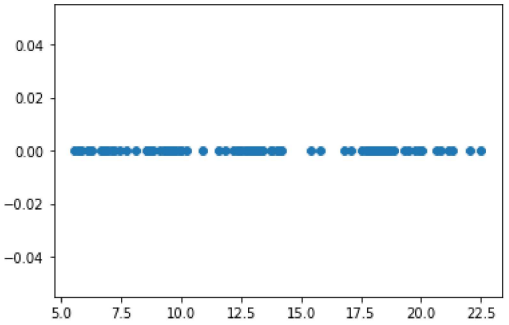
```
from sklearn.metrics import confusion_matrix
cf = confusion_matrix(y_test, y_pred5)
plt.figure()
sns.heatmap(cf, annot=True)
plt.xlabel('Prediction')
plt.ylabel('Target')
plt.title('Confusion Matrix')
```

```
Text(0.5, 1.0, 'Confusion Matrix')
```

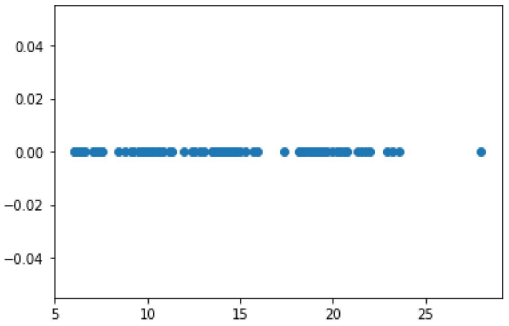



```
plt.scatter(x_test['Open'], y_pred5)
plt.show()
```


```
plt.scatter(x_test['Low'], y_pred5)
plt.show()
```




```
plt.scatter(x_test['High'], y_pred5)
plt.show()
```




```
plt.scatter(x_test['Close'], y_pred5)
plt.show()
```

```
print("Accuracy of svm:",accuracy_score(y_test,y_pred5))
```

```
Accuracy of svm: 0.5754716981132075
```

```
print("svm:",classification_report(y_test, y_pred5))
```

```
svm:               precision    recall  f1-score   support

           0       0.58      1.00      0.73        61
           1       0.00      0.00      0.00        45

    accuracy                           0.58       106
   macro avg       0.29      0.50      0.37       106
weighted avg       0.33      0.58      0.42       106

/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no p
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no p
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no p
  _warn_prf(average, modifier, msg_start, len(result))
```

K- Means

```
from sklearn.cluster import KMeans
model2 = KMeans(n_clusters=2)
model2.fit(x_train,y_train)
y_pred6 = model2.predict(x_test)
y_pred6
```

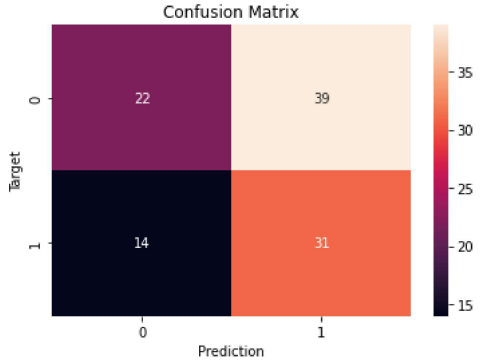
```
array([1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int32)
```

```
result = pd.DataFrame({'actual' : y_test ,'predicted' : y_pred6})
result
```

	actual	predicted
485	1	1
319	0	1
452	1	1

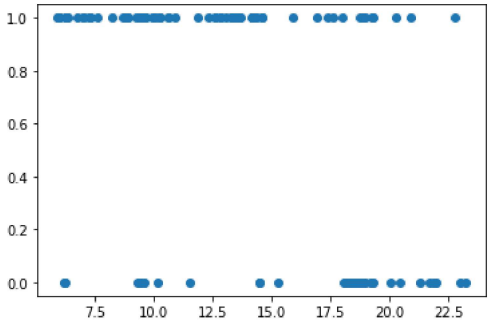
```
from sklearn.metrics import confusion_matrix
cf = confusion_matrix(y_test, y_pred6)
plt.figure()
sns.heatmap(cf, annot=True)
plt.xlabel('Prediction')
plt.ylabel('Target')
plt.title('Confusion Matrix')
```

```
Text(0.5, 1.0, 'Confusion Matrix')
```

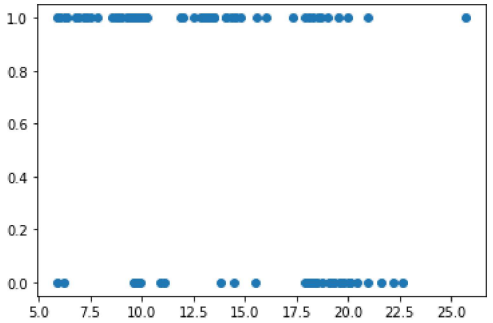



```
sse = []
for k in range(1, 3):
    model2 = KMeans(n_clusters=k)
    model2.fit(x_train,y_train)
    sse.append(model2.inertia_)

plt.plot(range(1,3), sse)
plt.xticks(range(1,3))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.show()
```

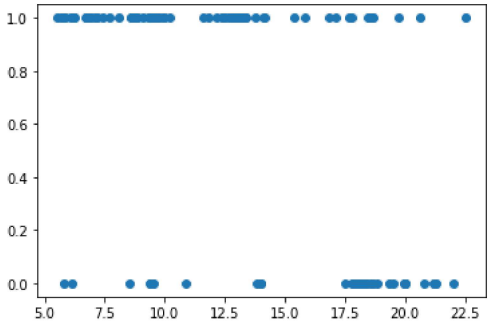

```
plt.scatter(x_test['Open'], y_pred6)
plt.show()
```



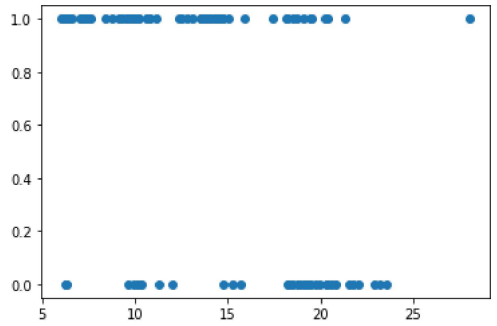

```
plt.scatter(x_test['Close'], y_pred6)
plt.show()
```




```
plt.scatter(x_test['Low'], y_pred6)
plt.show()
```




```
plt.scatter(x_test['High'], y_pred6)
plt.show()
```




```
print("Accuracy of K means:",accuracy_score(y_test,y_pred6))
```

```
Accuracy of K means: 0.5
```

```
print("K means:",classification_report(y_test, y_pred6))
```

```
K means:               precision    recall  f1-score   support

           0       0.61      0.36      0.45        61
           1       0.44      0.69      0.54        45

    accuracy                           0.50       106
   macro avg       0.53      0.52      0.50       106
weighted avg       0.54      0.50      0.49       106
```

Performance metric

```
from sklearn.metrics import accuracy_score
print("Accuracy of Naive Bayes:",accuracy_score(y_test,y_pred4))
print("Accuracy of Decision tree:",accuracy_score(y_test,y_pred1))
print("Accuracy of random forest:",accuracy_score(y_test,y_pred2))
print("Accuracy of KNN:",accuracy_score(y_test,y_pred3))
print("Accuracy of svc:",accuracy_score(y_test,y_pred5))
print("Accuracy of K means:",accuracy_score(y_test,y_pred6))
```

```
Accuracy of Naive Bayes: 0.5471698113207547
Accuracy of Decision tree: 0.7735849056603774
Accuracy of random forest: 0.7735849056603774
Accuracy of KNN: 0.5188679245283019
Accuracy of svc: 0.5754716981132075
Accuracy of K means: 0.5
```

```
from sklearn.metrics import classification_report, confusion_matrix
print("naive bayes")
print(confusion_matrix(y_test, y_pred4))
print("Decision tree")
print(confusion_matrix(y_test, y_pred1))
print("Random forest")
print(confusion_matrix(y_test, y_pred2))
```

```
print("Knn")
print(confusion_matrix(y_test, y_pred3))
print("SVC")
print(confusion_matrix(y_test, y_pred5))
print("K-means")
print(confusion_matrix(y_test, y_pred6))
```

```
naive bayes
[[56  5]
 [43  2]]
Decision tree
[[48 13]
 [11 34]]
Random forest
[[51 10]
 [14 31]]
Knn
[[49 12]
 [39  6]]
SVC
[[61  0]
 [45  0]]
K-means
[[22 39]
 [14 31]]
```

```
from sklearn.metrics import classification_report
print("Naive Bayes:",classification_report(y_test, y_pred4))
print("Decision tree:",classification_report(y_test, y_pred1))
print("random forest:",classification_report(y_test, y_pred2))
print("KNN:",classification_report(y_test, y_pred3))
print("svc:",classification_report(y_test, y_pred5))
print("K means:",classification_report(y_test, y_pred6))
```

```
              1       0.29      0.04      0.08        45

       accuracy                           0.55       106
      macro avg       0.43      0.48      0.39       106
   weighted avg       0.45      0.55      0.44       106

   Decision tree:              precision    recall  f1-score   support

              0       0.81      0.79      0.80        61
              1       0.72      0.76      0.74        45

       accuracy                           0.77       106
      macro avg       0.77      0.77      0.77       106
   weighted avg       0.78      0.77      0.77       106

   random forest:              precision    recall  f1-score   support

              0       0.78      0.84      0.81        61
              1       0.76      0.69      0.72        45

       accuracy                           0.77       106
      macro avg       0.77      0.76      0.77       106
   weighted avg       0.77      0.77      0.77       106
```