In [1]:

```
"""
Load the data
"""


import pandas as pd

ydf=pd.read_csv('/home/harshit/Downloads/YESBANK.csv')
ydf
```

Out[1]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2018-01-16	334.000000	338.500000	328.000000	333.899994	319.873657	470267.0
1	2018-01-17	336.000000	343.750000	331.250000	342.500000	328.112457	653618.0
2	2018-01-18	350.000000	356.500000	333.100006	340.250000	325.956970	2419109.0
3	2018-01-19	348.000000	352.000000	339.250000	348.299988	333.668793	1659646.0
4	2018-01-22	349.000000	358.000000	349.000000	355.250000	340.326874	663569.0
...	...	...	...	...	...	...	...
484	2020-01-09	47.150002	48.450001	46.299999	47.299999	47.299999	6835915.0
485	2020-01-10	47.599998	48.349998	43.900002	44.799999	44.799999	15918973.0
486	2020-01-13	43.400002	44.000000	41.200001	42.099998	42.099998	10763969.0
487	2020-01-14	41.750000	41.750000	36.549999	38.549999	38.549999	18250917.0
488	2020-01-15	38.549999	41.099998	36.650002	39.799999	39.799999	19876620.0

489 rows × 7 columns

In [2]:

```
#import LinearRegression Class
from sklearn.linear_model import LinearRegression
```

In [3]:

```
#Algorithm---------------->step by step solution to a problem
#training

#model-------------->system of prediction which will give you your final output/answer

#feature!!!
```

In [4]:

```
#step 1-------------> select your target Attribute
```

In [5]:

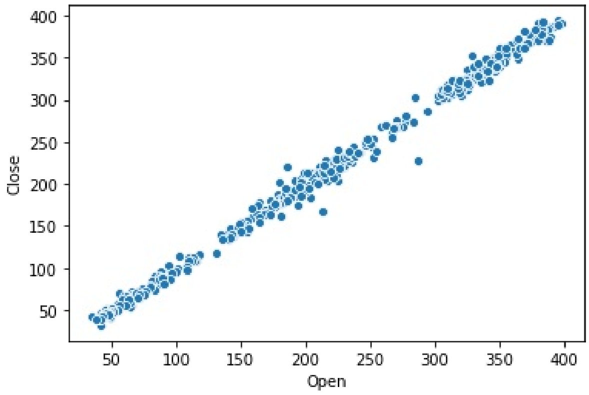
```
import seaborn as sns
```

In [6]:

```
sns.scatterplot(x='Open',y='Close',data=ydf)
```

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f629cfdc520>
```

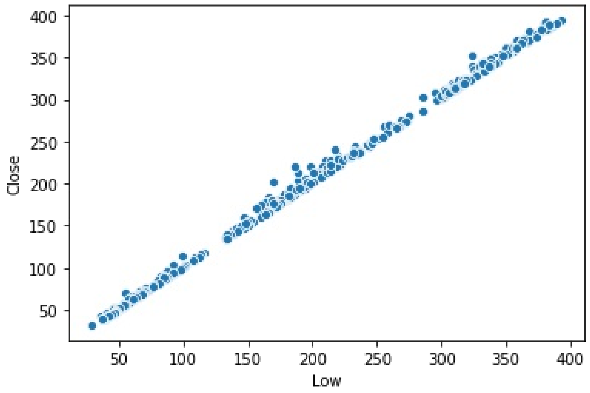


In [7]:

```
sns.scatterplot(x='Low',y='Close',data=ydf)
```

Out[7]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f629a8cda60>
```



In [8]:

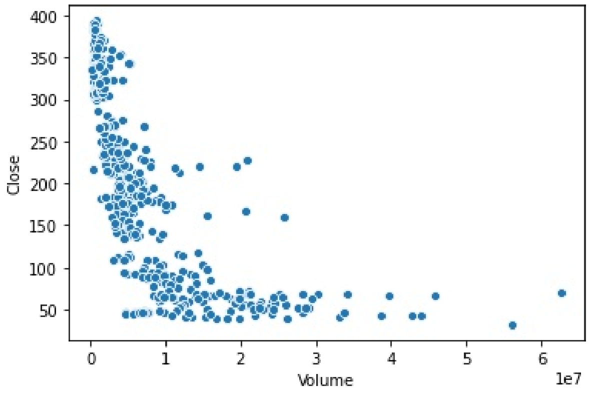
```
# Univariate Linear Regression
```

In [9]:

```
sns.scatterplot(x='Volume',y='Close',data=ydf)
```

Out[9]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f629f48c2e0>
```



In [10]:

```
ydf.isna().sum() #check for missing values
```

Out[10]:

```
Date         0
Open         1
High         1
Low          1
Close        1
Adj Close    1
Volume       1
dtype: int64
```

In [11]:

```
ydf.dropna(axis=0,how='any',inplace=True) #drop missing values
```

In [12]:

```
ydf.isna().sum()
```

Out[12]:

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

In [13]:

```
#separate the column which is target
target=ydf.pop('Close')
ydf
```

Out[13]:

	Date	Open	High	Low	Adj Close	Volume
0	2018-01-16	334.000000	338.500000	328.000000	319.873657	470267.0
1	2018-01-17	336.000000	343.750000	331.250000	328.112457	653618.0
2	2018-01-18	350.000000	356.500000	333.100006	325.956970	2419109.0
3	2018-01-19	348.000000	352.000000	339.250000	333.668793	1659646.0
4	2018-01-22	349.000000	358.000000	349.000000	340.326874	663569.0
...	...	...	...	...	...	...
484	2020-01-09	47.150002	48.450001	46.299999	47.299999	6835915.0
485	2020-01-10	47.599998	48.349998	43.900002	44.799999	15918973.0
486	2020-01-13	43.400002	44.000000	41.200001	42.099998	10763969.0
487	2020-01-14	41.750000	41.750000	36.549999	38.549999	18250917.0
488	2020-01-15	38.549999	41.099998	36.650002	39.799999	19876620.0

488 rows × 6 columns

In [14]:

```
target
```

Out[14]:

```
0      333.899994
1      342.500000
2      340.250000
3      348.299988
4      355.250000
          ...    
484     47.299999
485     44.799999
486     42.099998
487     38.549999
488     39.799999
Name: Close, Length: 488, dtype: float64
```

In [15]:

```
feature=ydf.pop('Low') #feature to be used on x-axis
ydf
```

Out[15]:

	Date	Open	High	Adj Close	Volume
0	2018-01-16	334.000000	338.500000	319.873657	470267.0
1	2018-01-17	336.000000	343.750000	328.112457	653618.0
2	2018-01-18	350.000000	356.500000	325.956970	2419109.0
3	2018-01-19	348.000000	352.000000	333.668793	1659646.0
4	2018-01-22	349.000000	358.000000	340.326874	663569.0
...	...	...	...	...	...
484	2020-01-09	47.150002	48.450001	47.299999	6835915.0
485	2020-01-10	47.599998	48.349998	44.799999	15918973.0
486	2020-01-13	43.400002	44.000000	42.099998	10763969.0
487	2020-01-14	41.750000	41.750000	38.549999	18250917.0
488	2020-01-15	38.549999	41.099998	39.799999	19876620.0

488 rows × 5 columns

In [16]:

```
feature
```

Out[16]:

```
0      328.000000
1      331.250000
2      333.100006
3      339.250000
4      349.000000
          ...
484     46.299999
485     43.900002
486     41.200001
487     36.549999
488     36.650002
Name: Low, Length: 488, dtype: float64
```

In [17]:

```
from sklearn.model_selection import train_test_split

#split data into training and testing set
xtrain,xtest,ytraining,ytest=train_test_split(feature ,target,test_size=0.2)
```

In [18]:

```
xtrain
```

Out[18]:

```
391     53.150002
90     332.799988
223    172.000000
370     81.750000
263    173.300003
          ...
464     40.700001
181    217.199997
400     58.900002
442     64.349998
111    327.350006
Name: Low, Length: 390, dtype: float64
```

In [19]:

```
ytraining
```

Out[19]:

```
391     56.299999
90     343.149994
223    174.699997
370     91.150002
263    174.800003
          ...
464     42.799999
181    240.000000
400     61.900002
442     69.000000
111    329.200012
Name: Close, Length: 390, dtype: float64
```

In [20]:

```
#predictor model


model=LinearRegression()
```

In [21]:

```
import numpy as np


#reshape the data in 1 column layout
xtrain=np.reshape(np.array(xtrain),(-1,1))
ytraining=np.reshape(np.array(ytraining),(-1,1))
```

In [22]:

```
#Training Model


model.fit(xtrain,ytraining) #training process!!!!!
```

Out[22]:

```
LinearRegression()
```

In [23]:

```
xtest
```

Out[23]:

```
166    319.000000
286    243.399994
249    191.100006
441     66.000000
477     46.349998
          ...
58     305.899994
162    314.600006
360     85.699997
86     325.700012
366     92.300003
Name: Low, Length: 98, dtype: float64
```

In [25]:

```
ytest
```

Out[25]:

```
166    323.149994
286    245.050003
249    192.100006
441     66.500000
477     46.950001
          ...
58     309.399994
162    316.700012
360     93.099998
86     330.500000
366    103.900002
Name: Close, Length: 98, dtype: float64
```

In [29]:

```
#reshapte testing x values as well
xtest=np.reshape(np.array(xtest),(-1,1))
values=model.predict(xtest)
```

In [30]:

```
from sklearn.metrics import mean_squared_error
mean_squared_error(ytest,values)
```

Out[30]:

```
10.191441691484796
```