In [1]:
```
# Machine learning



# 3 groups



# 1) Supervised Learning---------------->Data which is already labelled
# 2) Unsupervised Learning-------------->Data labels are created(clustering)

# 3) Reinforcement Learning------------->model improves based on feedback
```

In [2]:
```
# AI---->ML--------->DL

# Everything is MATH
# ML has 3 categories based on the data and behaviour
```

In [3]:
```
# """
# 1) choose an algorithm
# 2) Implement it as a program script(Python, R , Julia )
# 3) libraries with algorithms pre - defined and already implemented ( scikit learn ---> sklearn)
# 4) Choose training and testing parameters if applicable
# 5) Train, test and make predictions
# """
```

In [4]:
```
import sklearn
```

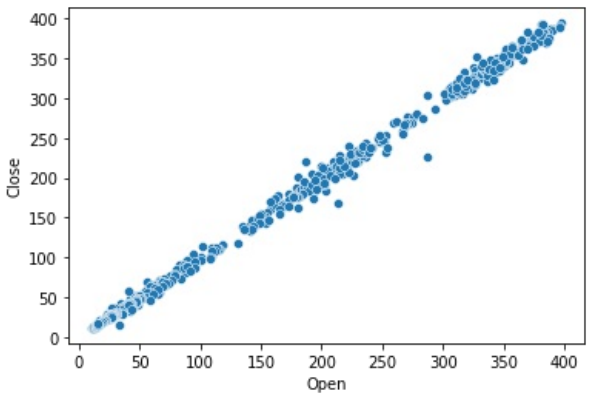
In [5]:
```
import pandas as pd
import seaborn as sns
df=pd.read_csv( "/home/harshit/DataSets/YESBANK.NS.csv")
df.sample(5)
```

Out[5]:

	Date	Open	High	Low	Close	Adj Close	Volume
683	2020-09-21	14.000000	14.050000	13.500000	13.550000	13.550000	92759761.0
575	2020-04-20	25.600000	30.100000	25.100000	29.450001	29.450001	114972573.0
482	2019-12-02	69.000000	69.300003	63.049999	64.050003	64.050003	361756071.0
95	2018-05-02	361.950012	364.700012	351.700012	354.250000	342.063446	13058846.0
44	2018-02-14	336.000000	337.850006	318.950012	320.350006	309.329620	13548524.0

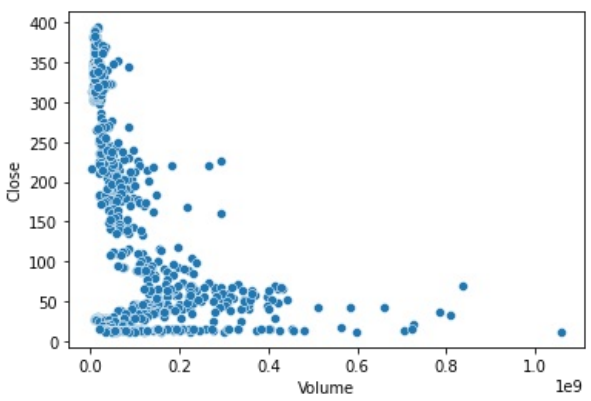
In [6]:
```
sns.scatterplot( x='Open', y = 'Close', data=df  )
```

Out[6]:
```
<AxesSubplot:xlabel='Open', ylabel='Close'>
```



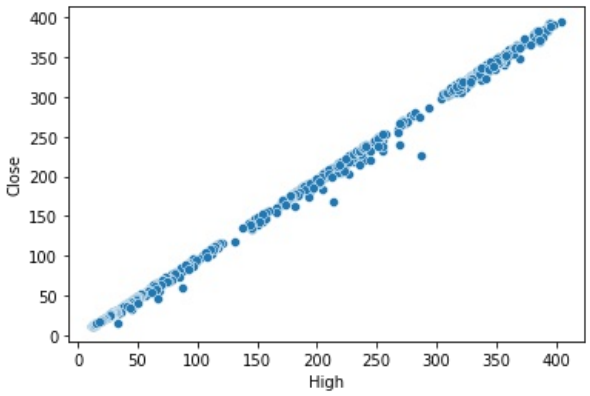
In [7]:
```
sns.scatterplot( x='Volume', y = 'Close', data=df  )
```

Out[7]:
```
<AxesSubplot:xlabel='Volume', ylabel='Close'>
```



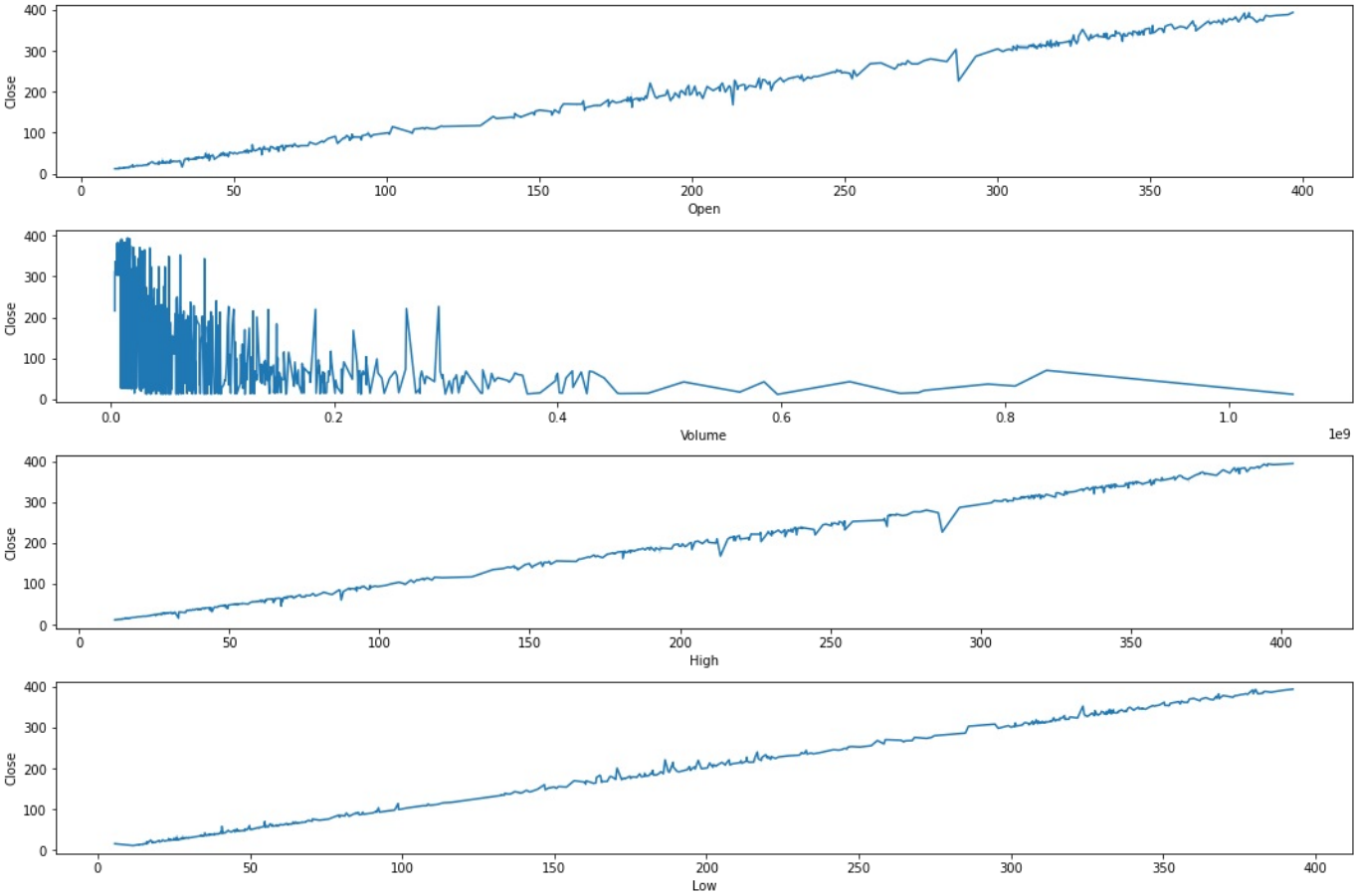
In [8]:
```
sns.scatterplot( x='High', y = 'Close', data=df  )
```

Out[8]:
```
<AxesSubplot:xlabel='High', ylabel='Close'>
```

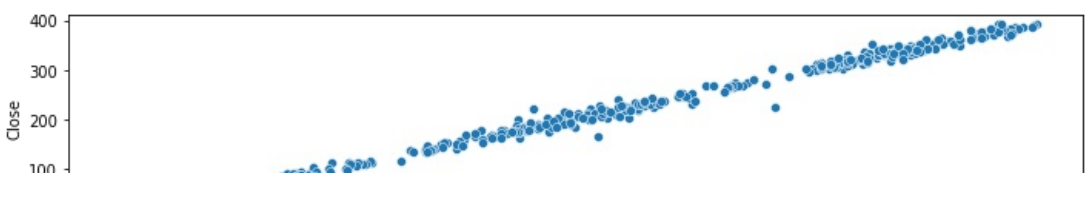


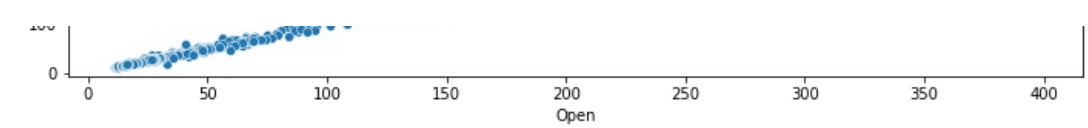
In [9]:
```
features=[  'Open', 'Volume', 'High', 'Low' ]
#close
```

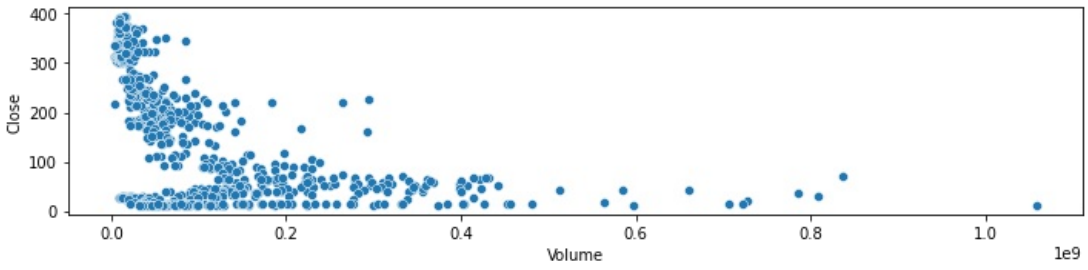
In [10]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(4,1, figsize=(15,10))

for idx,attribute in enumerate(features):
    sns.lineplot(x=attribute, y='Close',data=df,ax=ax[idx])

plt.tight_layout()
plt.show()
```

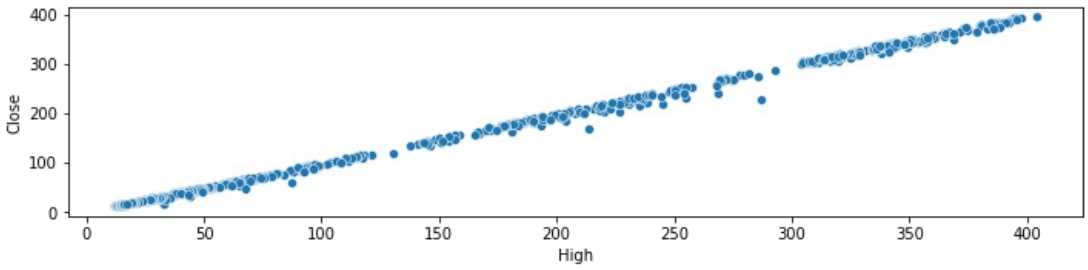


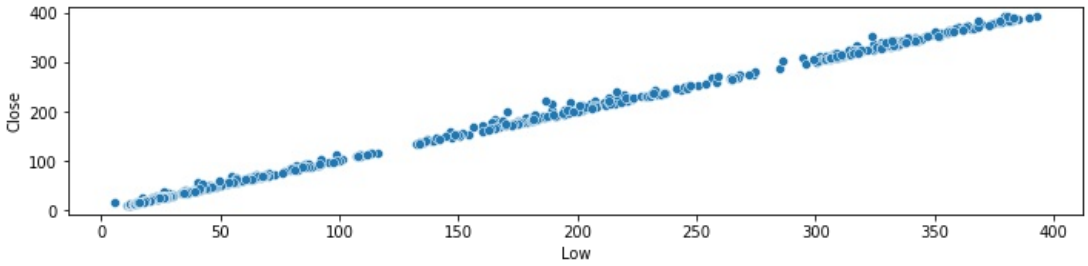
In [11]:
```
features=[  'Open', 'Volume', 'High', 'Low' ]
import matplotlib.pyplot as plt
fig, ax = plt.subplots(4,1, figsize=(10,10))

for idx,attribute in enumerate(features):
    sns.scatterplot(x=attribute, y='Close',data=df,ax=ax[idx])

plt.tight_layout()
plt.show()
```











In [12]:
```
df.corr()[['Close']] #coefficient of correlation
```

Out[12]:

	Close
Open	0.998891
High	0.999368
Low	0.999562
Close	1.000000
Adj Close	0.999966
Volume	-0.501380

In [13]:
```
features=df[['Open','Low','High']]
features.sample(2)
```

Out[13]:

	Open	Low	High
151	380.950012	379.399994	394.350006
736	15.650000	15.500000	15.850000

In [14]:
```
target=df[['Close']] #target

target.sample(2)
```

Out[14]:

	Close
464	68.300003
177	370.600006

In [15]:
```
features.shape
```

Out[15]:
```
(738, 3)
```

In [16]:
```
features.isna().sum()
```

Out[16]:
```
Open    2
Low     2
High    2
dtype: int64
```

In [17]:
```
target.isna().sum()
```

Out[17]:
```
Close    2
```

```
dtype: int64
```

In [18]:
```
#dropping records / rows with missing values

#IF ANY COLUMN IN A SINGLE row HAS MISSING VALUE
#IF ALL COLUMNS IN A SINGLE ROW HAVE MISSING VALUE

features.dropna(axis=0, how='any',inplace=True)
target.dropna(axis=0, how='any',inplace=True)
```

```
<ipython-input-18-b9eea41cbf96>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  features.dropna(axis=0, how='any',inplace=True)
<ipython-input-18-b9eea41cbf96>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  target.dropna(axis=0, how='any',inplace=True)
```

In [19]:
```
features.isna().sum()
```

Out[19]:
```
Open    0
Low     0
High    0
dtype: int64
```

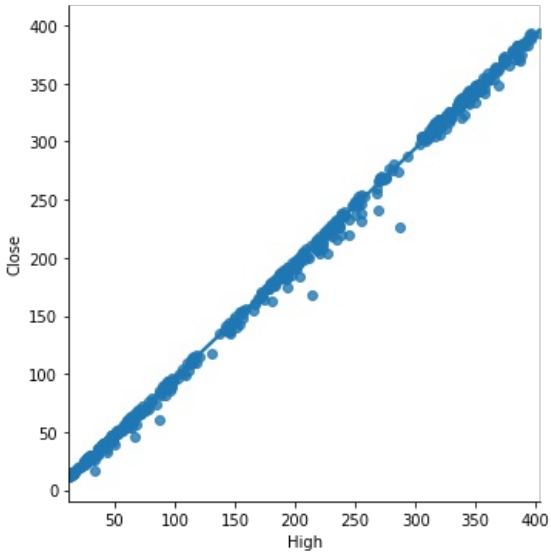
In [20]:
```
target.isna().sum()
```

Out[20]:
```
Close    0
dtype: int64
```

In [ ]:
```
"""
Linear Regression ------------> predicting a numerical value based on a set of features which are also numeric

Background Math -----> Equation of a straight line => y = m*x + C


we want to predict close price----> this is Y
We are making predictions on the basis of High value-----> High value is X

We need to calculate slope and y-intercept of a hypothetical line called BEST-FITTING-LINE


steps:
    1) Divide the available data into training and testing sets
    2) Training of the model
    3) test your model for accuracy
        -3a) report the error and accuracy for recording purpose
    4) Make predictions
"""
```

In [21]:
```
sns.lmplot(x='High',y='Close',data=df,)
```

Out[21]:
```
<seaborn.axisgrid.FacetGrid at 0x7fee62075ac0>
```



In [23]:
```
features
```

Out[23]:

	Open	Low	High
0	212.500000	210.600006	215.700088

0	313.500000	310.600006	315.799988
1	312.000000	305.899994	312.000000
2	306.350006	301.049988	307.350006
3	303.899994	301.750000	304.649994
4	307.000000	307.000000	317.450012
...	...	...	...
733	15.700000	14.850000	15.900000
734	15.650000	15.250000	15.800000
735	15.600000	15.050000	15.600000
736	15.650000	15.500000	15.850000
737	16.000000	16.000000	17.299999

736 rows × 3 columns

In [22]:
```
target
```

Out[22]:

	Close
0	311.600006
1	306.799988
2	301.899994
3	303.899994
4	315.899994
...	...
733	15.450000
734	15.450000
735	15.350000
736	15.750000
737	17.299999

736 rows × 1 columns

In [25]:
```
#if you have highly correlated records, go for lower percentage of testing

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(features,target,test_size=0.2)
```

In [26]:
```
x_train
```

Out[26]:

	Open	Low	High
271	204.800003	203.500000	209.149994
459	54.349998	53.099998	59.900002
132	333.600006	329.500000	335.700012
94	350.899994	350.299988	367.200012
188	318.000000	310.600006	319.950012
...	...	...	...
477	64.949997	62.299999	65.500000
535	37.500000	37.500000	40.200001
512	41.750000	36.549999	41.750000
6	314.000000	309.549988	314.399994
623	26.799999	26.000000	26.900000

588 rows × 3 columns

In [27]:
```
x_test
```

Out[27]:

	Open	Low	High
527	36.150002	34.450001	36.400002
110	335.000000	331.500000	338.750000

689	13.600000	13.300000	13.650000
682	14.100000	13.950000	14.250000
22	339.000000	335.299988	344.250000
...	...	...	...
283	195.899994	181.250000	197.199997
7	312.399994	310.000000	313.399994
586	26.400000	25.000000	26.750000
241	180.350006	175.250000	182.449997
725	14.550000	14.050000	14.700000

148 rows × 3 columns

In [28]:
```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
```

In [29]:
```
#training step

model.fit(  x_train, y_train ) #multivariate linear regression
```

Out[29]:
```
LinearRegression()
```

In [30]:
```
model.score(x_train, y_train) #R squared value--->
```

Out[30]:
```
0.9997184231459111
```

In [36]:
```
predicted_values=pd.DataFrame( model.predict( x_test  ) )



predicted_values
```

Out[36]:

	0
0	35.011541
1	335.281110
2	13.434802
3	14.128441
4	340.173707
...	...
143	185.126240
144	311.622717
145	25.562857
146	177.935862
147	14.292423

148 rows × 1 columns

In [35]:
```
y_test
```

Out[35]:

	Close
527	34.950001
110	333.700012
689	13.350000
682	14.000000
22	343.149994
...	...
283	185.600006
7	311.700012
586	26.299999
241	176.500000
725	14.250000

725 14.250000

148 rows × 1 columns

In [55]:
```
y_test.reset_index(inplace=True,drop=True)

y_test
# y_test.drop(columns='index',inplace=True)
```

Out[55]:

	Close
0	34.950001
1	333.700012
2	13.350000
3	14.000000
4	343.149994
...	...
143	185.600006
144	311.700012
145	26.299999
146	176.500000
147	14.250000

148 rows × 1 columns

In [62]:
```
ans=pd.concat( [predicted_values, y_test],
          ignore_index=True,
          axis=1,
         )
# ans
ans.rename(  columns={0:'Predicted', 1: 'Actual'},inplace=True  )
```

In [63]:
```
ans
```

Out[63]:

	Predicted	Actual
0	35.011541	34.950001
1	335.281110	333.700012
2	13.434802	13.350000
3	14.128441	14.000000
4	340.173707	343.149994
...	...	...
143	185.126240	185.600006
144	311.622717	311.700012
145	25.562857	26.299999
146	177.935862	176.500000
147	14.292423	14.250000

148 rows × 2 columns

In [67]:
```
model.coef_
```

Out[67]:
```
array([[-0.51795132,  0.81909961,  0.70033333]])
```

In [68]:
```
model.intercept_ #y_intercept
```

Out[68]:
```
array([0.02536508])
```

In [72]:
```
from sklearn.metrics import r2_score
r2_score(predicted_values, y_test)
```

Out[72]:
```
0.9997340860928497
```

In [66]:
```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(predicted_values, y_test)
```

Out[66]:
```
1.3066980809847957
```

In [64]:
```
from sklearn.metrics import mean_squared_error
```

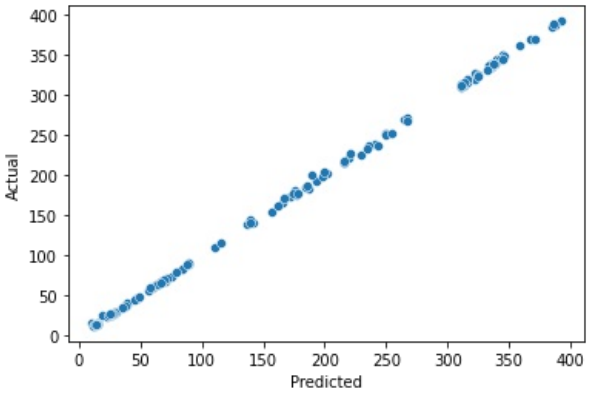
```
mean_squared_error(predicted_values, y_test)
```

Out[64]: 4.29806779336632

In [ ]:
```
#y =  m1x1 + m2x2 + m3x3 + ..... + mnxn + C
```

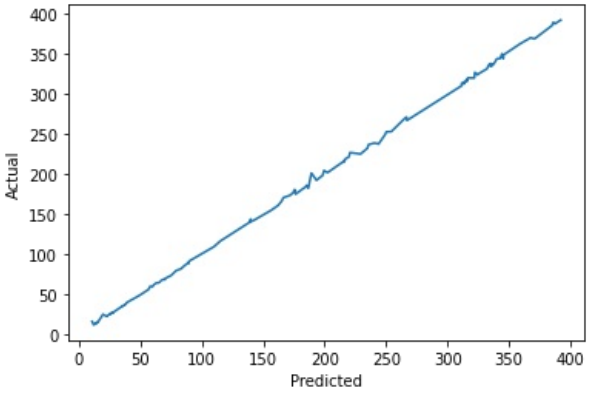
In [69]:
```
sns.scatterplot( x='Predicted', y='Actual',data=ans  )
```

Out[69]: <AxesSubplot:xlabel='Predicted', ylabel='Actual'>



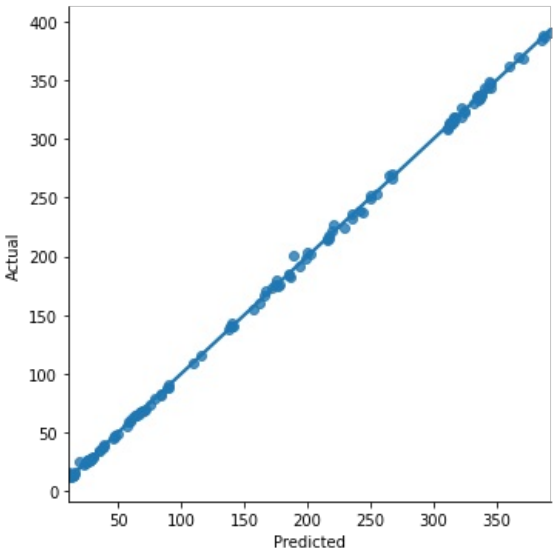
In [70]:
```
sns.lineplot( x='Predicted', y='Actual',data=ans )
```

Out[70]: <AxesSubplot:xlabel='Predicted', ylabel='Actual'>



In [75]:
```
sns.lmplot( x='Predicted', y='Actual',data=ans )
```

Out[75]: <seaborn.axisgrid.FacetGrid at 0x7fee55507d30>



In [71]:
```
ans.corr()
```

Out[71]:

	Predicted	Actual
Predicted	1.000000	0.999869
Actual	0.999869	1.000000

In [74]:
```
features.shape
```

Out[74]: (736, 3)

In [ ]:
```
"""
we can create csv file or excel file
"""
```

In [80]:
```
import numpy as np

l1= [  [311.21, 309.43, 314.65]  ]

data=np.array(   l1  )
```

In [81]:
```
model.predict(data) #close
```

Out[81]:
```
array([[312.64761005]])
```

In [83]:
```
def make_prediction():
    f1=float(input("Enter the opening price: "))
    f2=float(input("Enter the Low price: "))
    f3=float(input("Enter the High price: "))
    data=np.array(  [  [f1,f2,f3]  ]  )
    print(f"The closing price based on your input SHOULD BE: {model.predict(data)}")
```

In [84]:
```
make_prediction() #function call
```

```
Enter the opening price: 245.21
Enter the Low price: 239.76
Enter the High price: 248.34
The closing price based on your input SHOULD BE: [[243.32662411]]
```

In [ ]: