In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:

```
from sklearn.model_selection import train_test_split
```

In [4]:

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge,Lasso
from sklearn.linear_model import ElasticNet
from sklearn import metrics
```

DataSet Vehicle

In [6]:

```
a=pd.read_csv(r"E:\Python Data Science\Documents\1.fiat500_VehicleSelection_Dataset - fia
a
```

Out[6]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	
0	1.0	lounge	51.0	882.0	25000.0	1.0	44.907242	8.611
1	2.0	pop	51.0	1186.0	32500.0	1.0	45.666359	12.24
2	3.0	sport	74.0	4658.0	142228.0	1.0	45.503300	11
3	4.0	lounge	51.0	2739.0	160000.0	1.0	40.633171	17.63
4	5.0	pop	73.0	3074.0	106880.0	1.0	41.903221	12.49
...	...	...	...	...	...	...	...	
1544	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1545	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1546	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Null
1547	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1548	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

1549 rows × 11 columns

In [7]:

```
a.head()
```

Out[7]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lo
0	1.0	lounge	51.0	882.0	25000.0	1.0	44.907242	8.6115598
1	2.0	pop	51.0	1186.0	32500.0	1.0	45.666359	12.241889
2	3.0	sport	74.0	4658.0	142228.0	1.0	45.503300	11.417
3	4.0	lounge	51.0	2739.0	160000.0	1.0	40.633171	17.634609
4	5.0	pop	73.0	3074.0	106880.0	1.0	41.903221	12.495650

In [8]:

```
a.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1549 entries, 0 to 1548
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ID               1538 non-null   float64
 1   model            1538 non-null   object 
 2   engine_power     1538 non-null   float64
 3   age_in_days      1538 non-null   float64
 4   km               1538 non-null   float64
 5   previous_owners  1538 non-null   float64
 6   lat              1538 non-null   float64
 7   lon              1549 non-null   object 
 8   price            1549 non-null   object 
 9   Unnamed: 9       0 non-null      float64
 10  Unnamed: 10      1 non-null      object 
dtypes: float64(7), object(4)
memory usage: 133.2+ KB
```

In [9]:

```
a.describe()
```

Out[9]:

	ID	engine_power	age_in_days	km	previous_owners	la
count	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.00000
mean	769.500000	51.904421	1650.980494	53396.011704	1.123537	43.54136
std	444.126671	3.988023	1289.522278	40046.830723	0.416423	2.13351
min	1.000000	51.000000	366.000000	1232.000000	1.000000	36.85583
25%	385.250000	51.000000	670.000000	20006.250000	1.000000	41.80299
50%	769.500000	51.000000	1035.000000	39031.000000	1.000000	44.39409
75%	1153.750000	51.000000	2616.000000	79667.750000	1.000000	45.46796
max	1538.000000	77.000000	4658.000000	235000.000000	4.000000	46.79561

In [10]:

```
a.columns
```

Out[10]:

```
Index(['ID', 'model', 'engine_power', 'age_in_days', 'km', 'previous_owner
s',
       'lat', 'lon', 'price', 'Unnamed: 9', 'Unnamed: 10'],
      dtype='object')
```

In [11]:

```
a.isna()
```

Out[11]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon	price
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
1544	True	True	True	True	True	True	True	False	False
1545	True	True	True	True	True	True	True	False	False
1546	True	True	True	True	True	True	True	False	False
1547	True	True	True	True	True	True	True	False	False
1548	True	True	True	True	True	True	True	False	False

1549 rows × 11 columns

In [12]:

```
b=a.fillna(value=50)
b
```

Out[12]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	
0	1.0	lounge	51.0	882.0	25000.0	1.0	44.907242	8.611
1	2.0	pop	51.0	1186.0	32500.0	1.0	45.666359	12.24
2	3.0	sport	74.0	4658.0	142228.0	1.0	45.503300	11
3	4.0	lounge	51.0	2739.0	160000.0	1.0	40.633171	17.63
4	5.0	pop	73.0	3074.0	106880.0	1.0	41.903221	12.49
...	...	...	...	...	...	...	...	
1544	50.0	50	50.0	50.0	50.0	50.0	50.000000	
1545	50.0	50	50.0	50.0	50.0	50.0	50.000000	
1546	50.0	50	50.0	50.0	50.0	50.0	50.000000	Null
1547	50.0	50	50.0	50.0	50.0	50.0	50.000000	
1548	50.0	50	50.0	50.0	50.0	50.0	50.000000	

1549 rows × 11 columns

In [13]:

```
c=b.dropna(axis=1)
c
```

Out[13]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon	pric
0	1.0	lounge	51.0	882.0	25000.0	1.0	44.907242	8.611559868	890
1	2.0	pop	51.0	1186.0	32500.0	1.0	45.666359	12.24188995	880
2	3.0	sport	74.0	4658.0	142228.0	1.0	45.503300	11.41784	420
3	4.0	lounge	51.0	2739.0	160000.0	1.0	40.633171	17.63460922	600
4	5.0	pop	73.0	3074.0	106880.0	1.0	41.903221	12.49565029	570
...	...	...	...	...	...	...	...	...	
1544	50.0	50	50.0	50.0	50.0	50.0	50.000000	length	
1545	50.0	50	50.0	50.0	50.0	50.0	50.000000	concat	lonpric
1546	50.0	50	50.0	50.0	50.0	50.0	50.000000	Null values	N

In [14]:

```
c.columns
```

Out[14]:

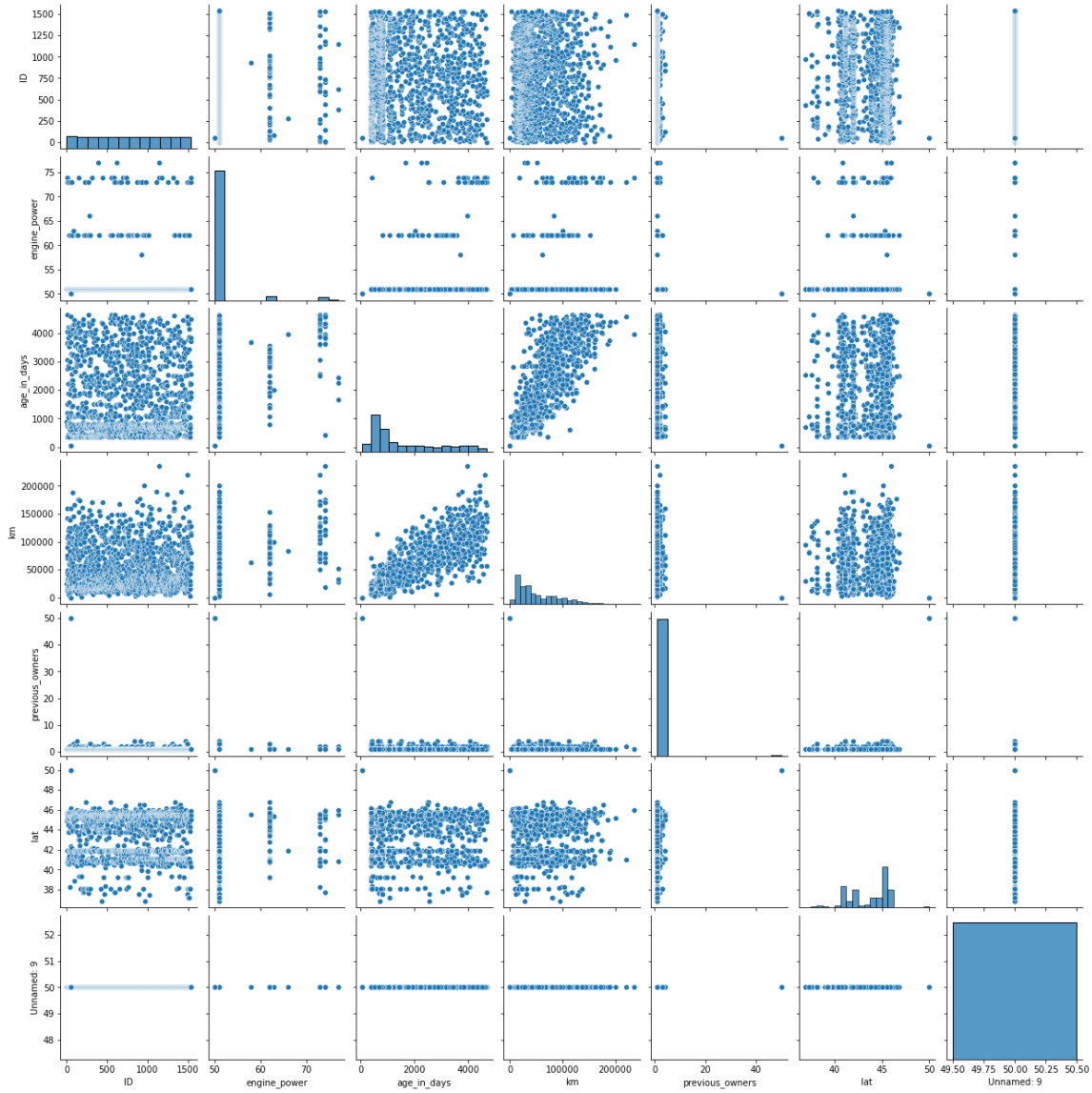
```
Index(['ID', 'model', 'engine_power', 'age_in_days', 'km', 'previous_owner
s',
       'lat', 'lon', 'price', 'Unnamed: 9', 'Unnamed: 10'],
      dtype='object')
```

In [15]:

```
sns.pairplot(c)
```

Out[15]:

```
<seaborn.axisgrid.PairGrid at 0x19942003520>
```



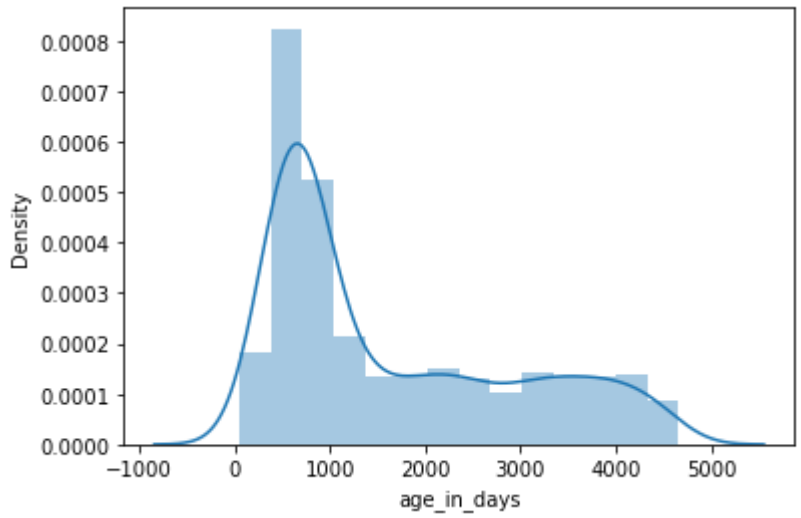
In [16]:

```
sns.distplot(c["age_in_days"])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[16]:

```
<AxesSubplot:xlabel='age_in_days', ylabel='Density'>
```

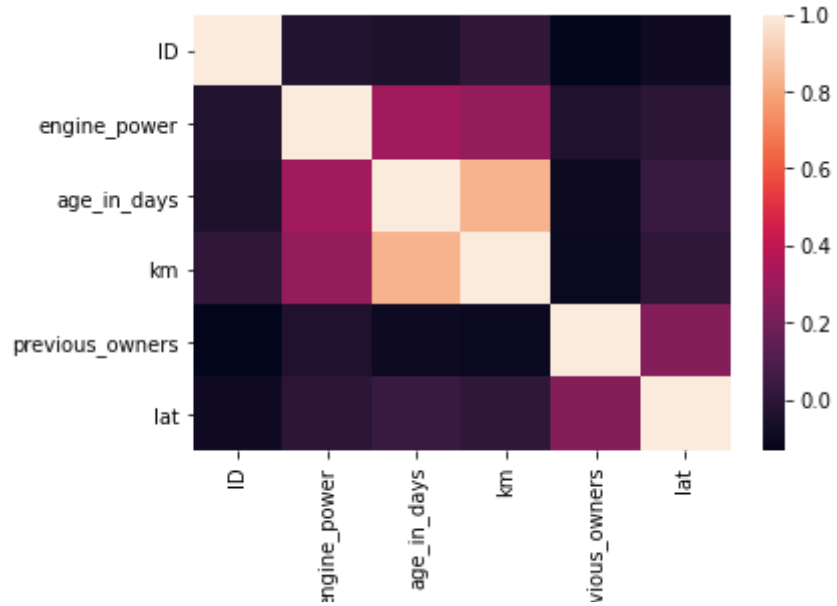


In [17]:

```
d=c[['ID', 'engine_power', 'age_in_days', 'km', 'previous_owners',
       'lat', 'lon', 'price']]

# Correlation map
sns.heatmap(d.corr())
```

```
<AxesSubplot:>
```



In [18]:

```
d.columns
```

Out[18]:

```
Index(['ID', 'engine_power', 'age_in_days', 'km', 'previous_owners', 'la
t',
       'lon', 'price'],
      dtype='object')
```

In [112]:

```
X=np.array(d['age_in_days']).reshape(-1,1)
# y=d[['km']]
y=np.array(d['km']).reshape(-1,1)
```

In [113]:

```
X_train,y_test,y_train,X_test=train_test_split(X,y,test_size=0.2)
```

In [114]:

```
lr=LinearRegression()
lr.fit(X_train,y_train)
```

Out[114]:

```
LinearRegression()
```

In [115]:

```
print(lr.intercept_)
```

```
[10789.66587942]
```

In [116]:

```
coeff=pd.DataFrame(lr.coef_,X.columns,columns=['Co-efficient'])
coeff
```

```
---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent call las
t)
<ipython-input-116-bb20e98dfb50> in <module>
----> 1 coeff=pd.DataFrame(lr.coef_,X.columns,columns=['Co-efficient'])
      2 coeff

AttributeError: 'numpy.ndarray' object has no attribute 'columns'
```

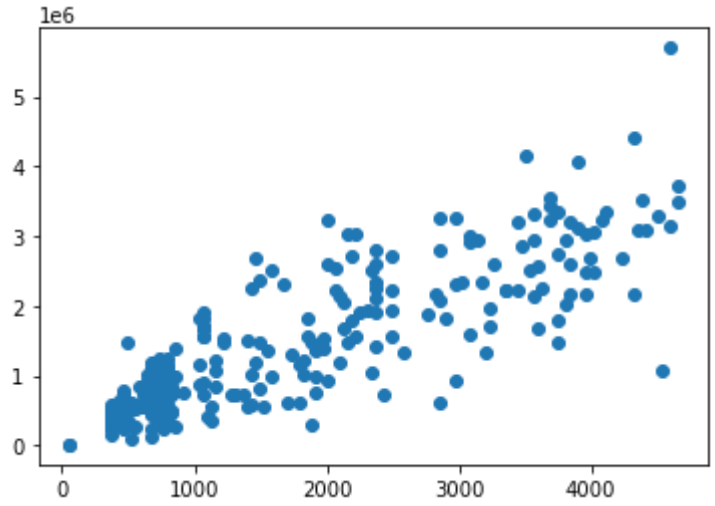
In [117]:

```
#Predicting

prediction=lr.predict(X_test)
plt.scatter(y_test,prediction)
```

Out[117]:

```
<matplotlib.collections.PathCollection at 0x199461058e0>
```



In [118]:

```
# Score

print(lr.score(X_test,y_test))
```

```
-1803187.8211347149
```

In [119]:

```
#Ridge fitting
rr=Ridge(alpha=10)
rr.fit(X_train,y_train)
```

Out[119]:

```
Ridge(alpha=10)
```

In [120]:

```
#Ridge Score
rr.score(X_test,y_test)
```

Out[120]:

```
-1803187.8044480123
```

In [124]:

```
# Lasso Fitting
la=Lasso(alpha=10)
la.fit(X_train,y_train)
```

Out[124]:

```
Lasso(alpha=10)
```

In [125]:

```
# Lasso Score
la.score(X_test,y_test)
```

Out[125]:

```
-1803187.0216804468
```

In [126]:

```
# ElasticNet
en=ElasticNet()
en.fit(X_train,y_train)
```

Out[126]:

```
ElasticNet()
```

In [127]:

```
print(en.coef_)
```

```
[25.86116044]
```

In [128]:

```
print(en.intercept_)
```

```
[10789.67891037]
```

In [129]:

```
print(en.predict(X_test))
```

```
[1045236.09642245 1913886.61436779 1818898.57207974  864207.97335784
 3088267.77100882 2312432.95787476 2079682.51393454  968169.8383178
 1184886.36278658  649560.34172408 3346879.37538685 2131404.83481014
 2157265.99524795 1950376.71174553 1042649.98037867  457463.64199208
 2933100.80838201  414559.97682577  502151.72722861  279745.74746351
 1077821.15857408  350191.54849608 1093596.46644114 1043063.75894568
  734902.17116883 2370155.06797193  993513.77554685  553874.04810421
 1678834.52714861  889060.54853857  230609.54263168  718609.64009301
  349700.18644776  493591.68312369  994030.9987556  3010684.28969542
  528012.88766641   12082.73693226  953635.86615176 2157265.99524795
 3088267.77100882  941403.53726468  563727.15023101  657318.68985542
 3062406.61057102 1821070.90955652  114234.32066157 1795209.74911872
  760763.33160663 2672730.64509422  344838.28828546 1103656.45785145
 1096958.41729806 2596905.72269058 3295157.05451124  342097.00527905
 1523589.98104048  444093.42204574 2224453.29006536 3424462.85670025
  934317.57930472 2519322.24137718  450429.406353   1001660.04108476
 1790890.9353256  1239815.46755648  246048.65541305  649560.34172408
 1001272.12367819  605596.36897982  261642.93515705  902068.71223878
 2234849.47656135  583562.66028681  292909.07812635 2596905.72269058
  782745.31797876  502151.72722861  680593.73424944  441196.97207671
 1200403.05904927 1071097.25686025 1024728.19619528 1174852.23253672
  722281.92487518  522840.65557885  424568.2459152  3041278.04249334
  458187.75448434  249953.69063916  886681.32177829  841708.76377695
  465946.10261568  432326.59404654  528012.88766641  564218.51227933
 1717626.26780531 1226264.21948707  353450.05471124  615940.83315494
 1709583.44690915  362242.8492601  1717626.26780531  628871.41337384
  398707.0854774  4407186.95333674 1023564.44397557 1574277.85549857
  789520.94201346  857303.04352094  807184.11459248  553874.04810421
 1329708.86123828  896094.78417765  864207.97335784 1898654.39086992
  346984.76460179 2803795.006193   3036545.45013322 1588320.4656163
  618526.94919872  941791.45467124  336640.30042667  953377.25454738
  595510.51640907 2027960.19305893 4070991.86764531 1245013.56080447
 1471945.24364619 2687885.28511077  833200.44199291 2053821.35349674
 2803795.006193    854716.92747716  838346.81292004  353139.72078599
 1063338.90872891  507349.82047661  725592.15341122  990927.65950307
 1691765.10736751  523719.93503373 3502046.33801366 3562664.89807987
  839820.89906499   88373.16022377 3269373.47755475 1014202.70389709
  734902.17116883   12082.73693226 1501995.91207491 1092484.43654232
  605596.36897982  450429.406353   2493461.08093937 2958961.96881981
  539676.27102386  866018.25458848 1479160.50740834 1482289.70782131
 2131404.83481014  511746.21775103  593984.70794324 2247780.05678025
 2212453.71162222 1818484.79351274 2519322.24137718  770978.48997956
 2910265.40371543 2741986.83274665 1479160.50740834 1305373.5092663
 2157265.99524795 3521752.54226726  432326.59404654 2536830.24699357
  786624.49204443  842225.98698571  243540.12285059 3230245.54181236
  597838.02084848  460773.87052812 1959660.8683427   734902.17116883
  372845.9250396  1084027.83707916  266815.16724461  398707.0854774
  935377.88688267  930205.65479511  437498.8261341   595484.65524864
  410732.52508098 2855517.3270686  1691765.10736751  352544.91409592
  494393.37909727 3321018.21494904  646922.50335942 1045236.09642245
 3114128.93144663 1355776.91095958 2121060.37063502  553874.04810421
 1226264.21948707  720419.92132366 2687911.14627121 1030727.98541685
  760763.33160663  770331.96096862 1102311.67750868  484048.91492215
  362501.46086448 2338294.11831256  483428.24707164 2472772.15258913
 2596905.72269058  256470.70306949 2726211.52487959  890379.46772089
  419396.01382764 1030081.4564059   300434.67581375 1562459.3051785
 1910369.49654825 2234849.47656135 1099415.22753965  402586.25954307
 1413757.63266113 2338294.11831256  721092.31149504  372535.59111434
 1412412.85231837  734126.33635569 1474531.35968997  383992.08518829
 2338294.11831256  335993.77141573 2265003.58963183  295262.44372619
 1387715.44410027 2312432.95787476  846829.27354363  528012.88766641
```

```
 2575699.57113159 1331855.33755461  760763.33160663  424568.2459152
  471118.33470324  456325.75093282  721221.61729723 3217573.57319783
 1536598.14474069 3708935.62151608 3139990.09188443 2700350.36444179
  675214.61287838 1400620.16315873 5700244.97522684  657318.68985542
  864207.97335784  464885.79503773 1812872.92169773 2939669.54313321
 3243434.73363564 1356242.41184746 1019995.60383516 1561450.71992142
  553874.04810421 1203196.06437655 2157265.99524795  838346.81292004
 3036545.45013322  621113.0652425   269401.28328839  450429.406353
 3217573.57319783  485005.77785834  313365.25603265 1561450.71992142
 1683437.81370653  424568.2459152  1174541.89861146  162594.69068026
  509806.6307182  1479160.50740834  476290.5667908   279745.74746351
  644388.10963652 2221918.89634245 1148680.73817366  528012.88766641
 3269295.89407344  295262.44372619 1386137.91331356  290090.21163863
 1893068.38021536 3243434.73363564 3346879.37538685 1562459.3051785
 2260710.63699915 4148575.34895871  518625.28642749 2596905.72269058
  502151.72722861  766452.78690295 2519322.24137718  551727.57178787
  760763.33160663 1950376.71174553]
```

In [130]:

```
print(en.score(X_test,y_test))
```

```
-1803186.747421288
```

In [131]:

```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 1332498.9628358942
```

In [132]:

```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 2763410848436.0493
```

In [133]:

```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root Mean Squared Error: 1662351.0003714766
```

DataSet 2015

In [22]:

```
a=pd.read_csv(r"E:\Python Data Science\Documents\2.2015.csv")
a
```

Out[22]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)
0	Switzerland	Western Europe	1	7.587	0.03411	1.39651	1.34951	0.94143
1	Iceland	Western Europe	2	7.561	0.04884	1.30232	1.40223	0.94784
2	Denmark	Western Europe	3	7.527	0.03328	1.32548	1.36058	0.87464
3	Norway	Western Europe	4	7.522	0.03880	1.45900	1.33095	0.88521
4	Canada	North America	5	7.427	0.03553	1.32629	1.32261	0.90563
...	...	...	...	...	...	...	...	...
153	Rwanda	Sub-Saharan Africa	154	3.465	0.03464	0.22208	0.77370	0.42864
154	Benin	Sub-Saharan Africa	155	3.340	0.03656	0.28665	0.35386	0.31910
155	Syria	Middle East and Northern Africa	156	3.006	0.05015	0.66320	0.47489	0.72193
156	Burundi	Sub-Saharan Africa	157	2.905	0.08658	0.01530	0.41587	0.22396
157	Togo	Sub-Saharan Africa	158	2.839	0.06727	0.20868	0.13995	0.28443

158 rows × 12 columns

In [23]:

```
a.head()
```

Out[23]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Fr
0	Switzerland	Western Europe	1	7.587	0.03411	1.39651	1.34951	0.94143	(
1	Iceland	Western Europe	2	7.561	0.04884	1.30232	1.40223	0.94784	(
2	Denmark	Western Europe	3	7.527	0.03328	1.32548	1.36058	0.87464	(
3	Norway	Western Europe	4	7.522	0.03880	1.45900	1.33095	0.88521	(
4	Canada	North America	5	7.427	0.03553	1.32629	1.32261	0.90563	(

In [24]:

```
a.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 158 entries, 0 to 157
Data columns (total 12 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Country                        158 non-null    object
 1   Region                         158 non-null    object
 2   Happiness Rank                 158 non-null    int64
 3   Happiness Score                158 non-null    float64
 4   Standard Error                 158 non-null    float64
 5   Economy (GDP per Capita)       158 non-null    float64
 6   Family                         158 non-null    float64
 7   Health (Life Expectancy)       158 non-null    float64
 8   Freedom                        158 non-null    float64
 9   Trust (Government Corruption)  158 non-null    float64
 10  Generosity                     158 non-null    float64
 11  Dystopia Residual              158 non-null    float64
dtypes: float64(9), int64(1), object(2)
memory usage: 14.9+ KB
```

In [25]:

```
a.describe()
```

Out[25]:

	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom
count	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000
mean	79.493671	5.375734	0.047885	0.846137	0.991046	0.630259	0.428615
std	45.754363	1.145010	0.017146	0.403121	0.272369	0.247078	0.150693
min	1.000000	2.839000	0.018480	0.000000	0.000000	0.000000	0.000000
25%	40.250000	4.526000	0.037268	0.545808	0.856823	0.439185	0.328330
50%	79.500000	5.232500	0.043940	0.910245	1.029510	0.696705	0.435515
75%	118.750000	6.243750	0.052300	1.158448	1.214405	0.811013	0.549092
max	158.000000	7.587000	0.136930	1.690420	1.402230	1.025250	0.669730

In [26]:

```
a.isna()
```

Out[26]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Free
0	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	
153	False	False	False	False	False	False	False	False	
154	False	False	False	False	False	False	False	False	
155	False	False	False	False	False	False	False	False	
156	False	False	False	False	False	False	False	False	
157	False	False	False	False	False	False	False	False	

158 rows × 12 columns

In [27]:

```
a.columns
```

Out[27]:

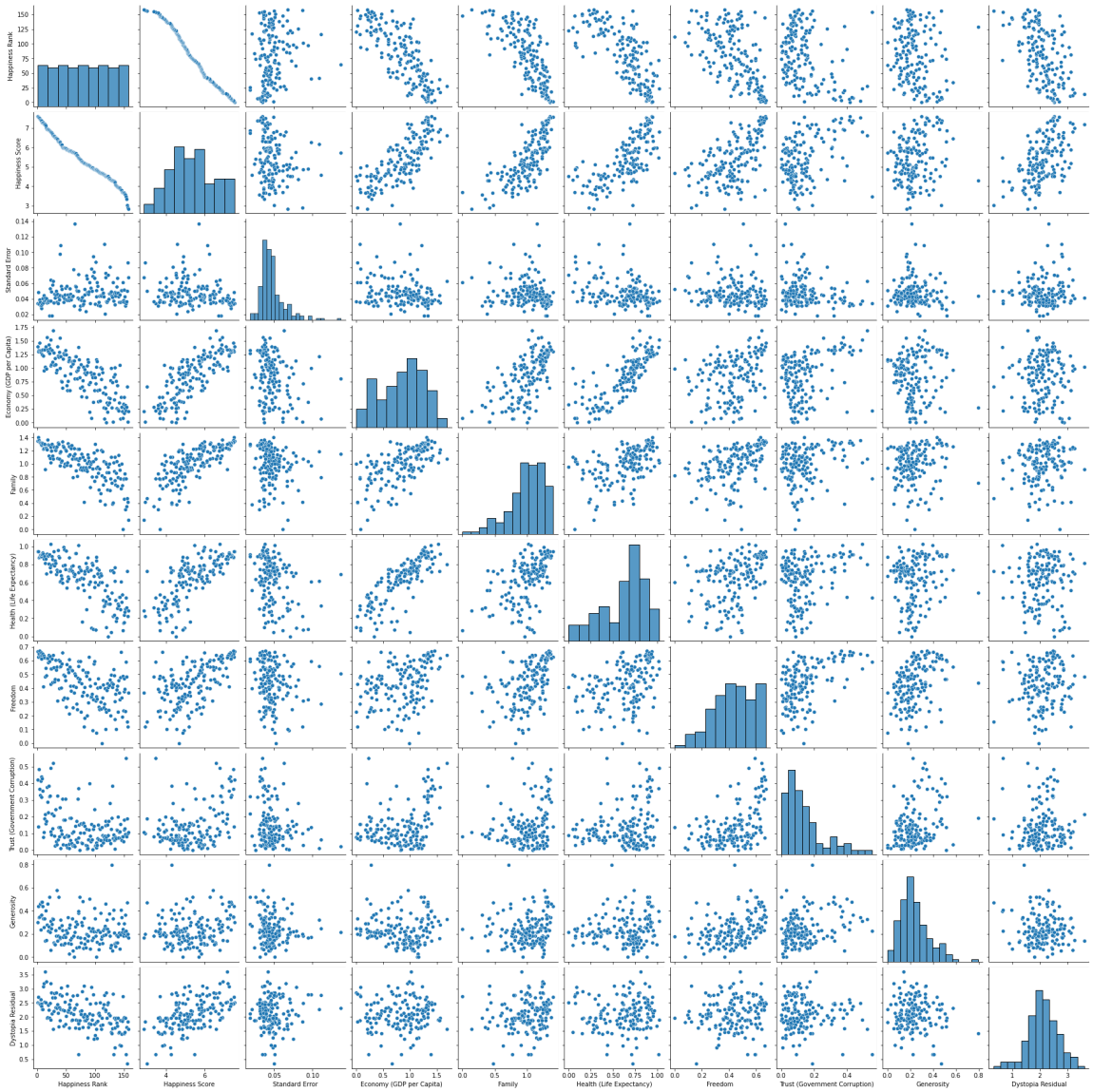
```
Index(['Country', 'Region', 'Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruptio
n)',
       'Generosity', 'Dystopia Residual'],
      dtype='object')
```

In [28]:

```
sns.pairplot(a)
```

Out[28]:

```
<seaborn.axisgrid.PairGrid at 0x1afec1fb3d0>
```



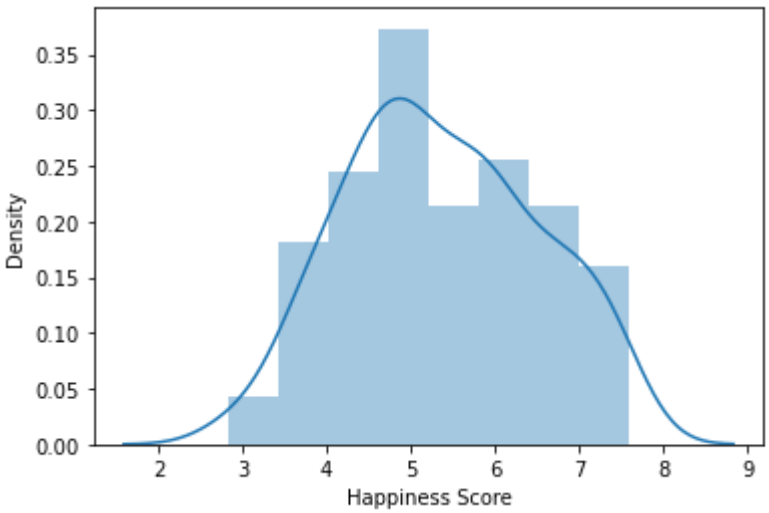
In [29]:

```
#normal distribution
sns.distplot(a['Happiness Score'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[29]:

```
<AxesSubplot:xlabel='Happiness Score', ylabel='Density'>
```



In [30]:

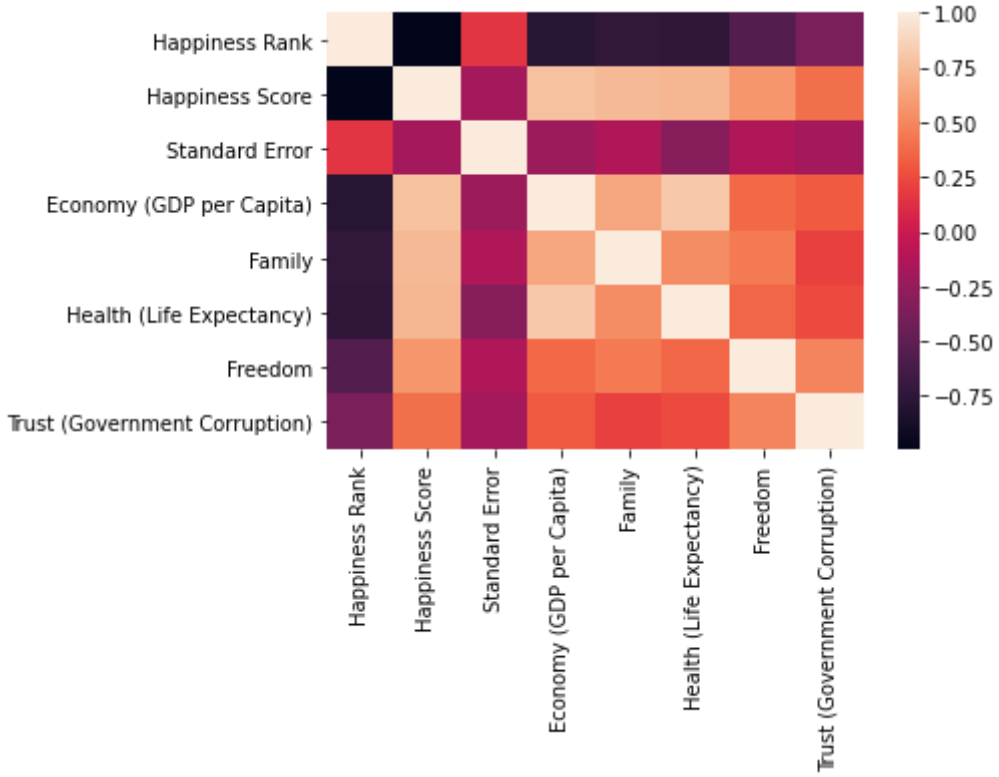
```
b=a[['Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruption)']]
```

In [31]:

```
# Correlation map
sns.heatmap(b.corr())
```

Out[31]:

```
<AxesSubplot:>
```



In [32]:

```
x=a[['Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruption)']]
y=a['Happiness Score']
```

In [33]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [34]:

```
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[34]:

```
LinearRegression()
```

In [35]:

```
print(lr.intercept_)
```

```
-2.6645352591003757e-15
```

In [36]:

```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[36]:

	Co-efficient
Happiness Score	1.000000e+00
Standard Error	5.239707e-16
Economy (GDP per Capita)	1.256233e-16
Family	-1.143704e-17
Health (Life Expectancy)	-3.894106e-17
Freedom	-1.520411e-16
Trust (Government Corruption)	3.351441e-17

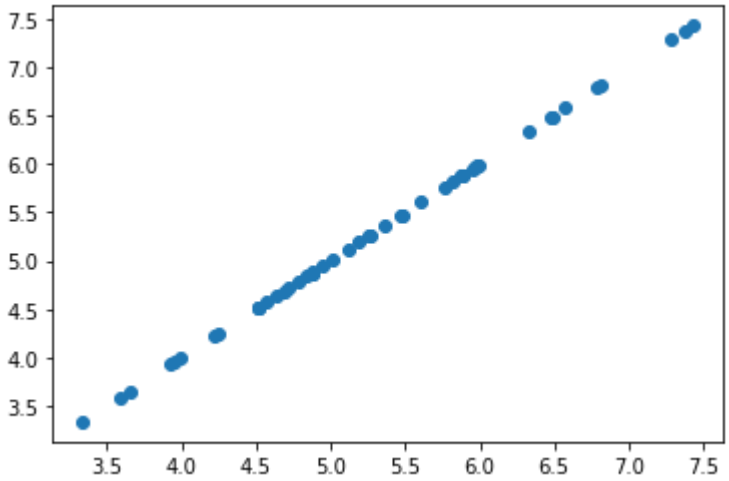
In [37]:

```
#Predicting

prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[37]:

```
<matplotlib.collections.PathCollection at 0x1aff16bc4c0>
```



In [38]:

```
# Score

print(lr.score(x_test,y_test))
```

```
1.0
```

In [39]:

```
#Ridge fitting
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[39]:

```
Ridge(alpha=10)
```

In [40]:

```
#Ridge Score
rr.score(x_test,y_test)
```

Out[40]:

```
0.9945136868586665
```

In [41]:

```
# Lasso Fitting
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[41]:

```
Lasso(alpha=10)
```

In [42]:

```
# Lasso Score
la.score(x_test,y_test)
```

Out[42]:

```
-0.00977273306138926
```

In [43]:

```
# ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[43]:

```
ElasticNet()
```

In [44]:

```
print(en.coef_)
```

```
[ 0.48203983 -0.          0.          0.          0.          0.
  0.        ]
```

In [45]:

```
print(en.intercept_)
```

```
2.799993782714357
```

In [46]:

```
print(en.predict(x_test))
```

```
[4.72574291 4.69489236 5.05883244 5.44012594 5.63342392 5.10703642
 5.92216578 5.60739377 5.43867983 5.00339786 5.96940568 5.13258453
 4.52907066 5.14945593 5.07281159 4.70694336 4.41000682 5.68982258
 5.50182704 5.68452014 5.67295118 6.07111609 6.08268504 4.97592159
 5.18560891 5.57606118 4.84962715 5.9260221  5.38372728 5.21356722
 4.97784975 5.66716671 5.26996588 5.33937962 4.8332378  6.35648367
 6.38010362 5.15475836 5.30370867 5.33214902 4.97495751 5.63872636
 5.85082388 4.56233141 5.6392084  6.30827968 5.15042001 5.03328433]
```

In [47]:

```
print(en.score(x_test,y_test))
```

```
0.729095409228625
```

In [48]:

```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 4.533410683886056e-16
```

In [49]:

```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 4.149737053506364e-31
```

In [50]:

```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root Mean Squared Error: 6.441845274070438e-16
```

DataSet Fitness

In [51]:

```
a=pd.read_csv(r"E:\Python Data Science\Documents\3.Fitness-1.csv")
a
```

Out[51]:

	Row Labels	Sum of Jan	Sum of Feb	Sum of Mar	Sum of Total Sales
0	A	0.0562	0.0773	0.0616	75
1	B	0.0421	0.1727	0.1921	160
2	C	0.0983	0.1160	0.0517	101
3	D	0.0281	0.2191	0.0788	127
4	E	0.2528	0.1057	0.1182	179
5	F	0.0815	0.1624	0.1847	167
6	G	0.1854	0.0876	0.1749	171
7	H	0.2556	0.0593	0.1379	170
8	Grand Total	1.0000	1.0000	1.0000	1150

In [52]:

```
a.head()
```

Out[52]:

	Row Labels	Sum of Jan	Sum of Feb	Sum of Mar	Sum of Total Sales
0	A	0.0562	0.0773	0.0616	75
1	B	0.0421	0.1727	0.1921	160
2	C	0.0983	0.1160	0.0517	101
3	D	0.0281	0.2191	0.0788	127
4	E	0.2528	0.1057	0.1182	179

In [53]:

```
a.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9 entries, 0 to 8
Data columns (total 5 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Row Labels          9 non-null      object 
 1   Sum of Jan          9 non-null      float64
 2   Sum of Feb          9 non-null      float64
 3   Sum of Mar          9 non-null      float64
 4   Sum of Total Sales  9 non-null      int64  
dtypes: float64(3), int64(1), object(1)
memory usage: 488.0+ bytes
```

In [54]:

```
a.describe()
```

Out[54]:

	Sum of Jan	Sum of Feb	Sum of Mar	Sum of Total Sales
count	9.000000	9.000000	9.000000	9.000000
mean	0.222222	0.222233	0.222211	255.555556
std	0.304383	0.296123	0.296410	337.332963
min	0.028100	0.059300	0.051700	75.000000
25%	0.056200	0.087600	0.078800	127.000000
50%	0.098300	0.116000	0.137900	167.000000
75%	0.252800	0.172700	0.184700	171.000000
max	1.000000	1.000000	1.000000	1150.000000

In [55]:

```
a.isna()
```

Out[55]:

	Row Labels	Sum of Jan	Sum of Feb	Sum of Mar	Sum of Total Sales
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
5	False	False	False	False	False
6	False	False	False	False	False
7	False	False	False	False	False
8	False	False	False	False	False

In [56]:

```
a.columns
```

Out[56]:

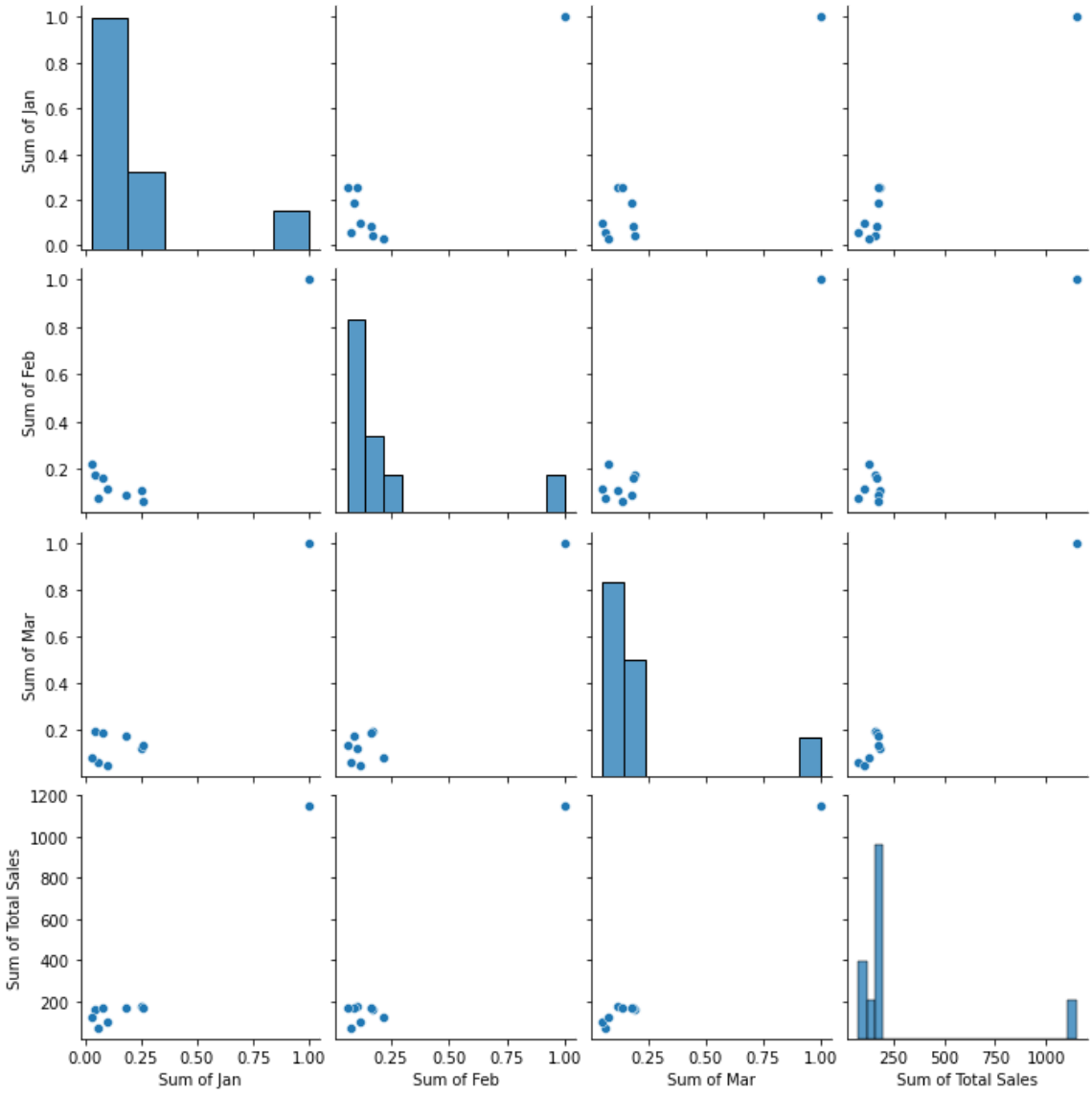
```
Index(['Row Labels', 'Sum of Jan', 'Sum of Feb', 'Sum of Mar',
       'Sum of Total Sales'],
      dtype='object')
```

In [57]:

```
sns.pairplot(a)
```

Out[57]:

```
<seaborn.axisgrid.PairGrid at 0x1aff16f3b50>
```



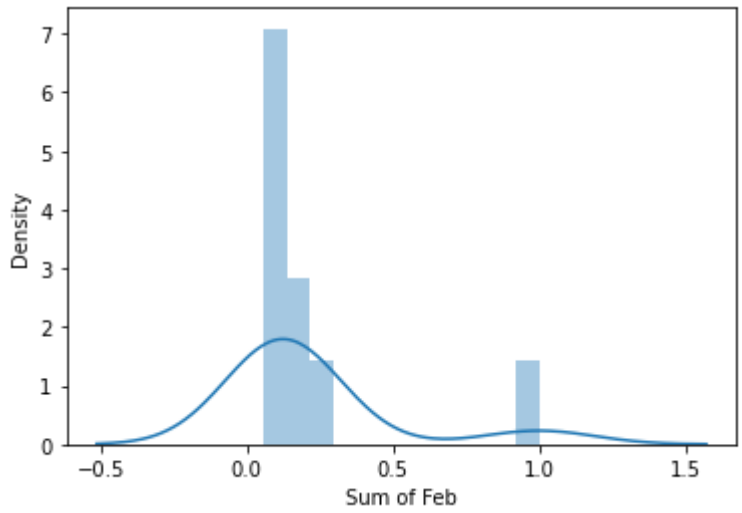
In [58]:

```
sns.distplot(a["Sum of Feb"])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[58]:

```
<AxesSubplot:xlabel='Sum of Feb', ylabel='Density'>
```



In [59]:

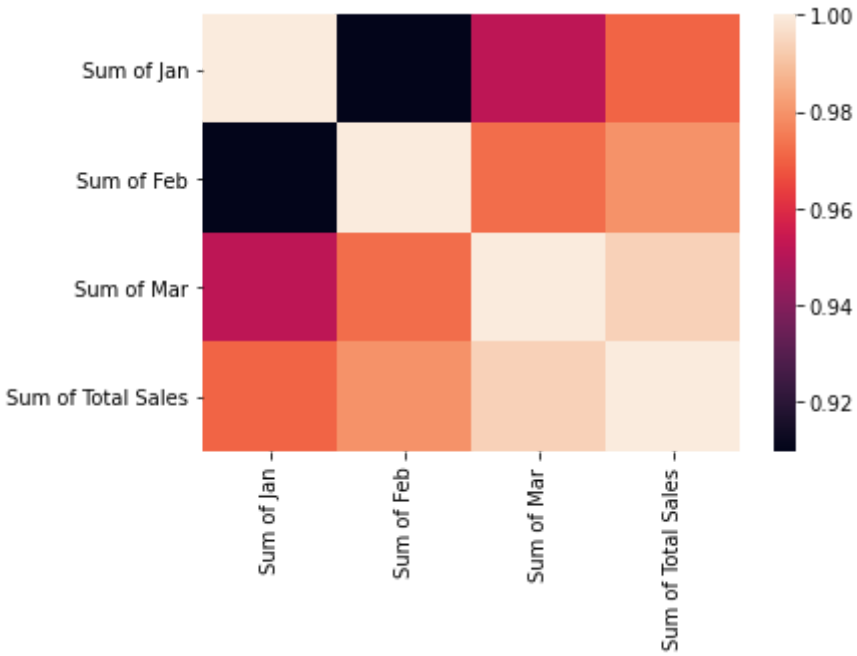
```
b=a[['Row Labels', 'Sum of Jan', 'Sum of Feb', 'Sum of Mar',
       'Sum of Total Sales']]
```

In [60]:

```
# Correlation map
sns.heatmap(b.corr())
```

Out[60]:

<AxesSubplot:>



In [61]:

```
x=a[['Sum of Jan', 'Sum of Feb', 'Sum of Mar',
       'Sum of Total Sales']]
y=a['Sum of Mar']
```

In [62]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [63]:

```
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[63]:

```
LinearRegression()
```

In [64]:

```
print(lr.intercept_)
```

```
1.1102230246251565e-16
```

In [65]:

```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[65]:

	Co-efficient
Sum of Jan	-6.217215e-14
Sum of Feb	-6.775999e-14
Sum of Mar	1.000000e+00
Sum of Total Sales	1.742011e-16

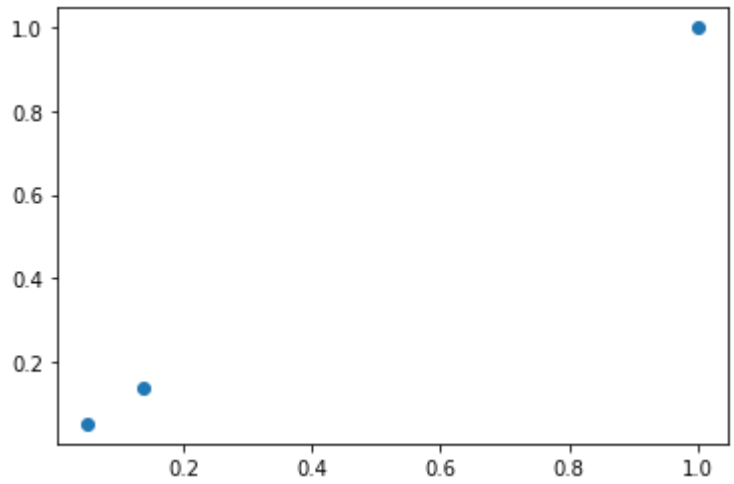
In [66]:

```
#Predicting

prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[66]:

```
<matplotlib.collections.PathCollection at 0x1aff365d4c0>
```



In [67]:

```
# Score

print(lr.score(x_test,y_test))
```

```
1.0
```

In [68]:

```
#Ridge fitting
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[68]:

```
Ridge(alpha=10)
```

In [69]:

```
#Ridge Score
rr.score(x_test,y_test)
```

Out[69]:

```
0.8899880354662043
```

In [70]:

```
# Lasso Fitting
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[70]:

```
Lasso(alpha=10)
```

In [71]:

```
# Lasso Score
la.score(x_test,y_test)
```

Out[71]:

```
-0.37296468887632117
```

In [72]:

```
# ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[72]:

```
ElasticNet()
```

In [73]:

```
print(en.coef_)
```

```
[-0.          0.          0.          0.00071768]
```

In [74]:

```
print(en.intercept_)
```

```
0.029909871687407302
```

In [75]:

```
print(en.predict(x_test))
```

```
[0.10239556 0.85524194 0.15191548]
```

In [76]:

```
print(en.score(x_test,y_test))
```

```
0.9568681381800686
```

In [77]:

```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 2.1510571102112408e-16
```

In [78]:

```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 1.0427305707236559e-31
```

In [79]:

```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root Mean Squared Error: 3.2291338942875317e-16
```

DataSet Drug

In [80]:

```
a=pd.read_csv(r"E:\Python Data Science\Documents\4_drug200 - 4_drug200.csv")
a
```

Out[80]:

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	drugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	drugY
...	...	...	...	...	...	...
195	56	F	LOW	HIGH	11.567	drugC
196	16	M	LOW	HIGH	12.006	drugC
197	52	M	NORMAL	HIGH	9.894	drugX
198	23	M	NORMAL	NORMAL	14.020	drugX
199	40	F	LOW	NORMAL	11.349	drugX

200 rows × 6 columns

In [81]:

```
a.head(10)
```

Out[81]:

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	drugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	drugY
5	22	F	NORMAL	HIGH	8.607	drugX
6	49	F	NORMAL	HIGH	16.275	drugY
7	41	M	LOW	HIGH	11.037	drugC
8	60	M	NORMAL	HIGH	15.171	drugY
9	43	M	LOW	NORMAL	19.368	drugY

In [82]:

```
a.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Age          200 non-null    int64  
 1   Sex          200 non-null    object 
 2   BP           200 non-null    object 
 3   Cholesterol  200 non-null    object 
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object 
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
```

In [83]:

```
a.describe()
```

Out[83]:

	Age	Na_to_K
count	200.000000	200.000000
mean	44.315000	16.084485
std	16.544315	7.223956
min	15.000000	6.269000
25%	31.000000	10.445500
50%	45.000000	13.936500
75%	58.000000	19.380000
max	74.000000	38.247000

In [84]:

```
a.isna()
```

Out[84]:

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
...	...	...	...	...	...	...
195	False	False	False	False	False	False
196	False	False	False	False	False	False
197	False	False	False	False	False	False
198	False	False	False	False	False	False
199	False	False	False	False	False	False

200 rows × 6 columns

In [85]:

```
a.columns
```

Out[85]:

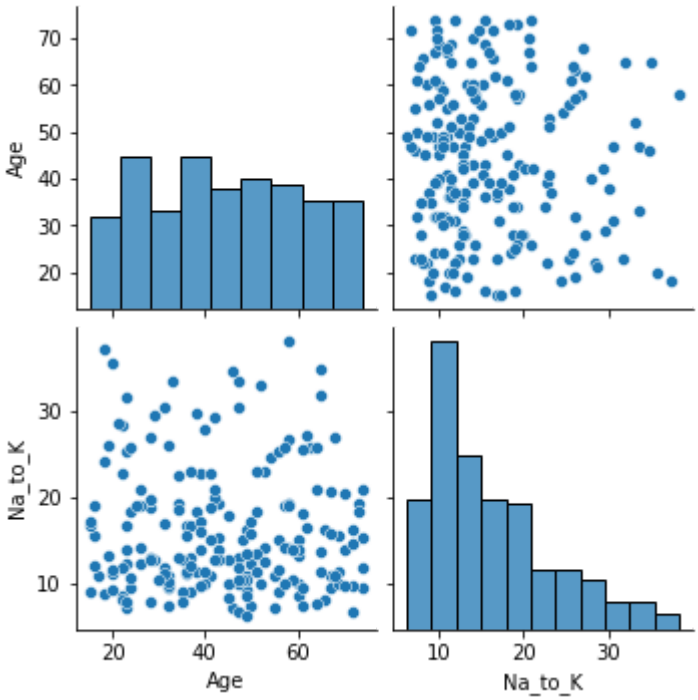
```
Index(['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K', 'Drug'], dtype='objec
t')
```

In [86]:

```
sns.pairplot(a)
```

Out[86]:

```
<seaborn.axisgrid.PairGrid at 0x1aff36130a0>
```



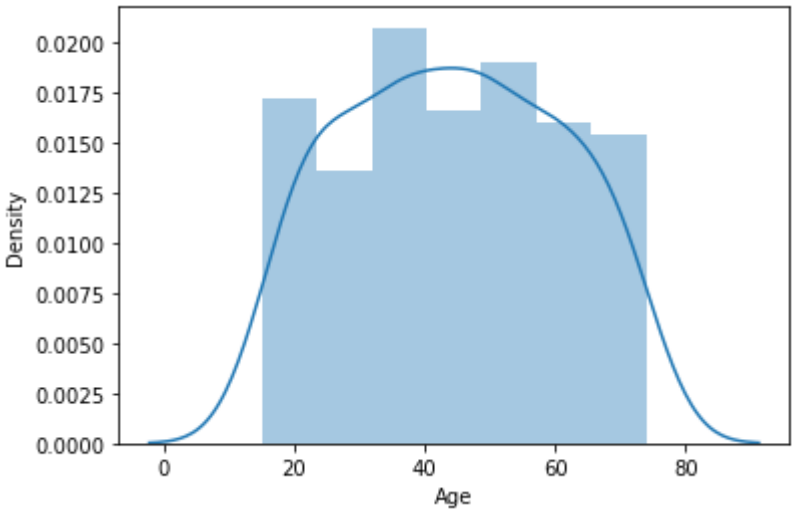
In [87]:

```
sns.distplot(a["Age"])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[87]:

```
<AxesSubplot:xlabel='Age', ylabel='Density'>
```

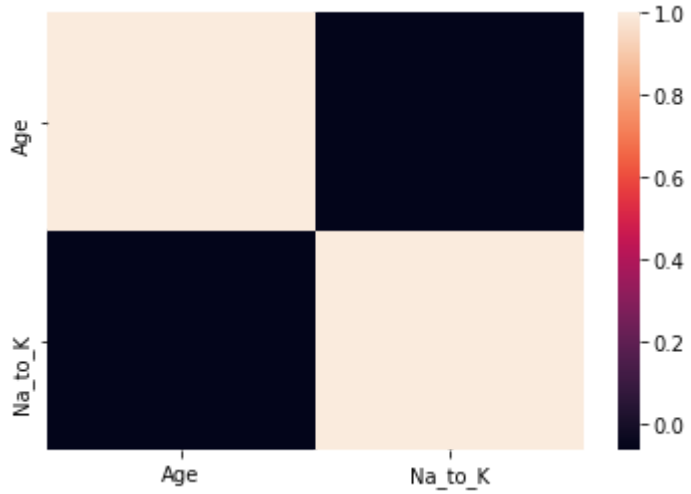


In [88]:

```
# Correlation map
sns.heatmap(a.corr())
```

Out[88]:

```
<AxesSubplot:>
```



In [89]:

```
x=a[['Age','Na_to_K']]
y=a['Age']
```

In [90]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)


lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[90]:

```
LinearRegression()
```

In [91]:

```
print(lr.intercept_)
```

```
-2.1316282072803006e-14
```

In [92]:

```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[92]:

	Co-efficient
Age	1.000000e+00
Na_to_K	9.076028e-18

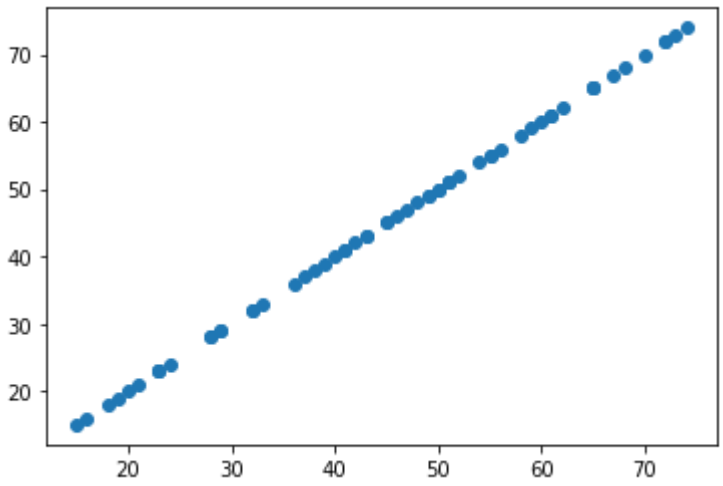
In [93]:

```
#Predicting

prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[93]:

```
<matplotlib.collections.PathCollection at 0x1aff3994910>
```



In [94]:

```
# Score

print(lr.score(x_test,y_test))
```

```
1.0
```

In [95]:

```
#Ridge fitting
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[95]:

```
Ridge(alpha=10)
```

In [96]:

```
#Ridge Score
rr.score(x_test,y_test)
```

Out[96]:

```
0.999999919752533
```

In [97]:

```
# Lasso Fitting
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[97]:

```
Lasso(alpha=10)
```

In [98]:

```
# Lasso Score
la.score(x_test,y_test)
```

Out[98]:

```
0.9986540629101466
```

In [99]:

```
# ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[99]:

```
ElasticNet()
```

In [100]:

```
print(en.coef_)
```

```
[ 0.9963519 -0.       ]
```

In [101]:

```
print(en.intercept_)
```

```
0.16009934736287335
```

In [102]:

```
print(en.predict(x_test))
```

```
[37.02511975 72.89378826 44.99593498 28.05795263 21.08348931 23.07619311
 49.97769449 32.04336024 48.98134259 39.01782356 29.05430453 50.97404639
 41.01052737 60.93756542 71.89743636 44.99593498 51.9703983  59.94121352
 33.03971214 47.98499069 60.93756542 69.90473255 67.91202874 23.07619311
 36.02876785 15.10537789 41.01052737 57.94850972 46.98863878 54.95945401
 71.89743636 45.99228688 54.95945401 50.97404639 43.00323117 42.00687927
 43.00323117 38.02147166 18.0944336  20.08713741 64.92297304 73.89014016
 19.0907855  50.97404639 49.97769449 55.95580591 53.9631021  61.93391733
 66.91567684 58.94486162 40.01417546 28.05795263 23.07619311 29.05430453
 24.07254502 64.92297304 32.04336024 16.10172979 48.98134259 64.92297304]
```

In [103]:

```
print(en.score(x_test,y_test))
```

```
0.9999865896854121
```

In [104]:

```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 4.322468309207276e-15
```

In [105]:

```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 4.2282944519846234e-29
```

In [106]:

```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root Mean Squared Error: 6.5025337000161895e-15
```

DataSet Instagram

In [107]:

```
a=pd.read_csv(r"E:\Python Data Science\Documents\5_Instagram data - 5_Instagram data.csv"
a
```

Out[107]:

	Impressions	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profil Visit
0	3920	2586	1028	619	56	98	9	5	162	3
1	5394	2727	1838	1174	78	194	7	14	224	4
2	4021	2085	1188	0	533	41	11	1	131	6
3	4528	2700	621	932	73	172	10	7	213	2
4	2518	1704	255	279	37	96	5	4	123	
...	...	...	...	...	...	...	...	...	...	.
114	13700	5185	3041	5352	77	573	2	38	373	7
115	5731	1923	1368	2266	65	135	4	1	148	2
116	4139	1133	1538	1367	33	36	0	1	92	3
117	32695	11815	3147	17414	170	1095	2	75	549	14
118	36919	13473	4176	16444	2547	653	5	26	443	61

119 rows × 13 columns

In [108]:

```
a.head(20)
```

Out[108]:

	Impressions	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profile Visits
0	3920	2586	1028	619	56	98	9	5	162	35
1	5394	2727	1838	1174	78	194	7	14	224	48
2	4021	2085	1188	0	533	41	11	1	131	62
3	4528	2700	621	932	73	172	10	7	213	23
4	2518	1704	255	279	37	96	5	4	123	8
5	3884	2046	1214	329	43	74	7	10	144	9
6	2621	1543	599	333	25	22	5	1	76	26
7	3541	2071	628	500	60	135	4	9	124	12
8	3749	2384	857	248	49	155	6	8	159	36
9	4115	2609	1104	178	46	122	6	3	191	31
10	2218	1597	411	162	15	28	6	3	81	29
11	3234	2414	476	185	75	122	8	14	151	15
12	4344	2168	1274	673	40	119	7	11	162	8

	Impressions	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profile Visits
13	3216	2524	212	201	223	121	5	5	142	20
14	9453	2525	5799	208	794	100	6	10	294	181
15	5055	2017	2351	298	108	101	7	11	159	17
16	4002	3401	278	128	73	111	17	18	205	16
17	3169	1979	707	341	32	106	8	1	121	21
18	6168	2177	3450	153	296	82	6	6	151	77
19	2407	1338	655	276	39	40	8	20	72	10

In [109]:
```
a.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119 entries, 0 to 118
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Impressions     119 non-null    int64 
 1   From Home       119 non-null    int64 
 2   From Hashtags   119 non-null    int64 
 3   From Explore    119 non-null    int64 
 4   From Other      119 non-null    int64 
 5   Saves           119 non-null    int64 
 6   Comments        119 non-null    int64 
 7   Shares          119 non-null    int64 
 8   Likes           119 non-null    int64 
 9   Profile Visits  119 non-null    int64 
 10  Follows         119 non-null    int64 
 11  Caption         119 non-null    object
 12  Hashtags        119 non-null    object
dtypes: int64(11), object(2)
memory usage: 12.2+ KB
```

In [110]:

```
a.describe()
```

Out[110]:

	Impressions	From Home	From Hashtags	From Explore	From Other	Saves	C
count	119.000000	119.000000	119.000000	119.000000	119.000000	119.000000	11
mean	5703.991597	2475.789916	1887.512605	1078.100840	171.092437	153.310924	
std	4843.780105	1489.386348	1884.361443	2613.026132	289.431031	156.317731	
min	1941.000000	1133.000000	116.000000	0.000000	9.000000	22.000000	
25%	3467.000000	1945.000000	726.000000	157.500000	38.000000	65.000000	
50%	4289.000000	2207.000000	1278.000000	326.000000	74.000000	109.000000	
75%	6138.000000	2602.500000	2363.500000	689.500000	196.000000	169.000000	
max	36919.000000	13473.000000	11817.000000	17414.000000	2547.000000	1095.000000	1

In [111]:

```
a.isna()
```

Out[111]:

	Impressions	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profil Visit
0	False	False	False	False	False	False	False	False	False	Fals
1	False	False	False	False	False	False	False	False	False	Fals
2	False	False	False	False	False	False	False	False	False	Fals
3	False	False	False	False	False	False	False	False	False	Fals
4	False	False	False	False	False	False	False	False	False	Fals
...	...	...	...	...	...	...	...	...	...	.
114	False	False	False	False	False	False	False	False	False	Fals
115	False	False	False	False	False	False	False	False	False	Fals
116	False	False	False	False	False	False	False	False	False	Fals
117	False	False	False	False	False	False	False	False	False	Fals
118	False	False	False	False	False	False	False	False	False	Fals

119 rows × 13 columns

In [112]:

```
a.columns
```

Out[112]:

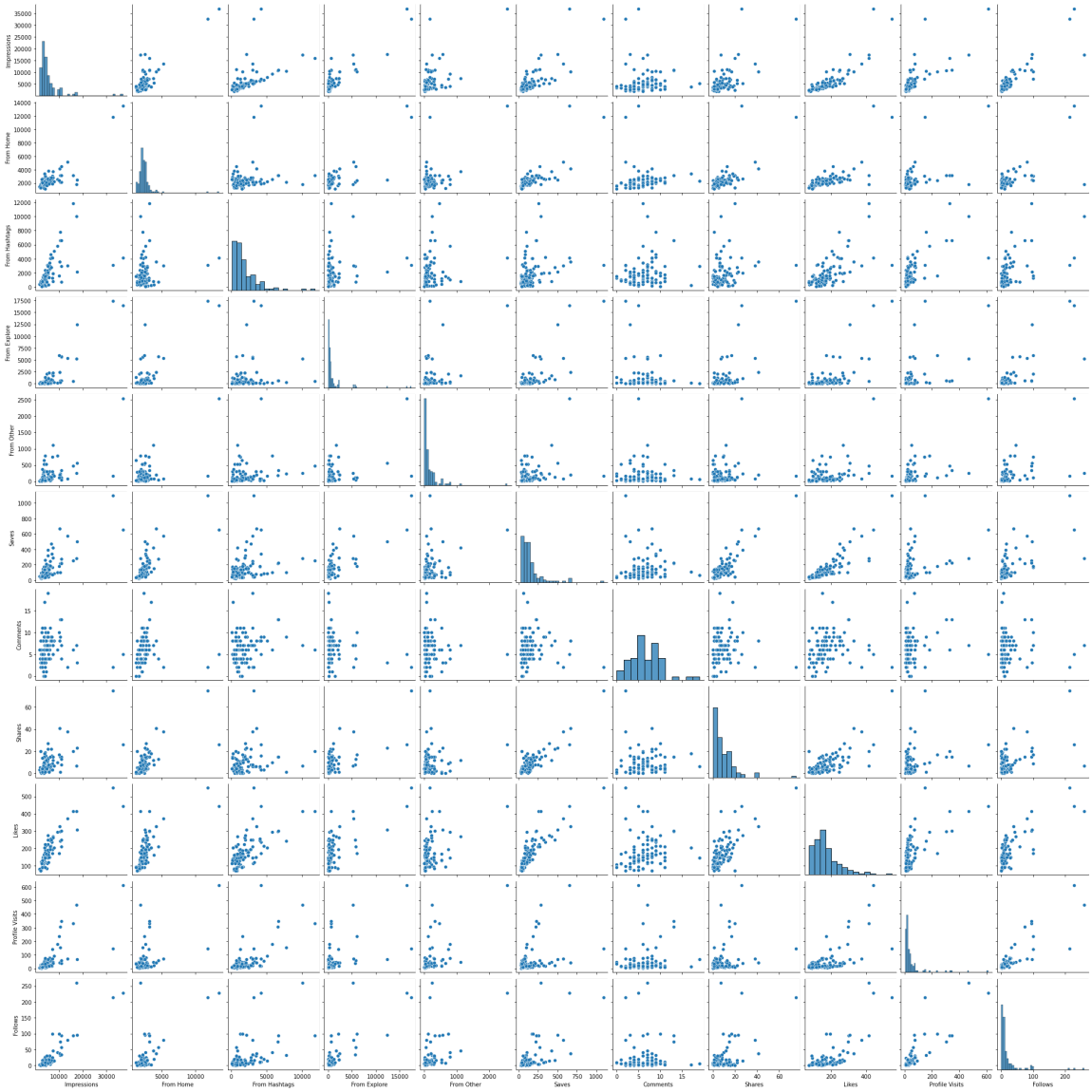
```
Index(['Impressions', 'From Home', 'From Hashtags', 'From Explore',
       'From Other', 'Saves', 'Comments', 'Shares', 'Likes', 'Profile Visi
ts',
       'Follows', 'Caption', 'Hashtags'],
      dtype='object')
```

In [113]:

```
sns.pairplot(a)
```

Out[113]:

```
<seaborn.axisgrid.PairGrid at 0x1aff39e6d60>
```



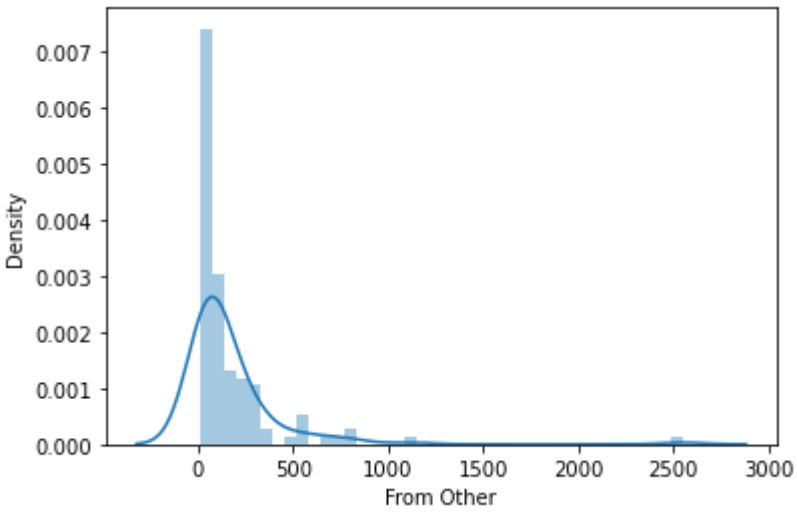
In [114]:

```
#normal distribution
sns.distplot(a["From Other"])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[114]:

```
<AxesSubplot:xlabel='From Other', ylabel='Density'>
```

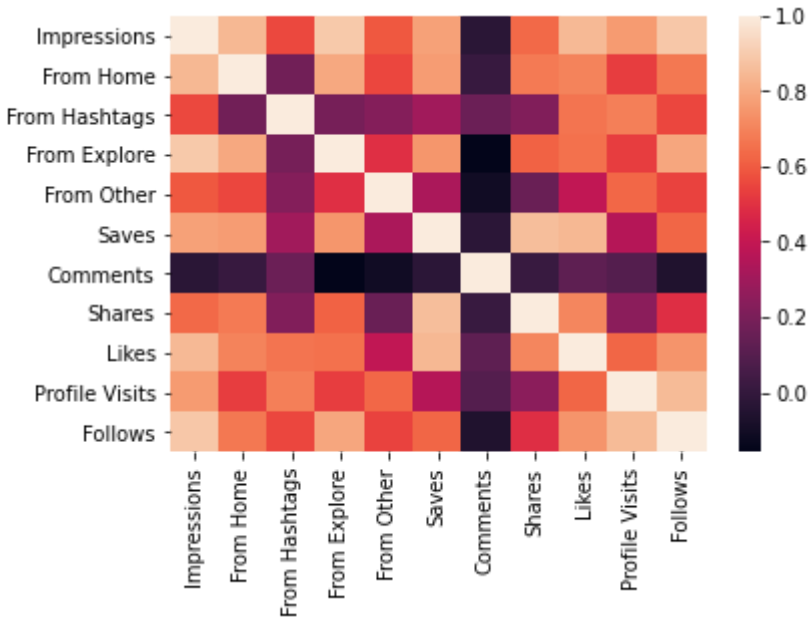


In [115]:

```
# Correlation map
sns.heatmap(a.corr())
```

Out[115]:

```
<AxesSubplot:>
```



In [116]:

```
x=a[['Impressions', 'From Home', 'From Hashtags', 'From Explore',
       'From Other', 'Saves', 'Comments', 'Shares', 'Likes', 'Profile Visits',
       'Follows']]
y=a['From Home']
```

In [117]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)


lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[117]:

```
LinearRegression()
```

In [118]:

```
print(lr.intercept_)
```

```
-2.2737367544323206e-12
```

In [119]:

```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[119]:

	Co-efficient
Impressions	2.504055e-15
From Home	1.000000e+00
From Hashtags	-2.746471e-15
From Explore	-2.433762e-15
From Other	1.264992e-16
Saves	1.108227e-15
Comments	3.004869e-14
Shares	-1.234987e-14
Likes	2.324474e-15
Profile Visits	1.847849e-15
Follows	-1.499204e-15

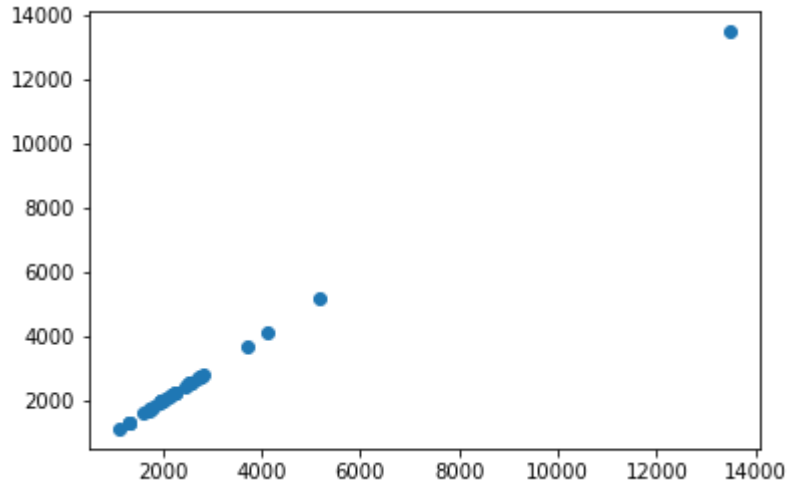
In [120]:

```
#Predicting

prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[120]:

```
<matplotlib.collections.PathCollection at 0x1affa299c10>
```



In [121]:

```
# Score

print(lr.score(x_test,y_test))
```

```
1.0
```

In [122]:

```
#Ridge fitting
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[122]:

```
Ridge(alpha=10)
```

In [123]:

```
#Ridge Score
rr.score(x_test,y_test)
```

Out[123]:

```
0.9999999999984914
```

In [124]:

```
# Lasso Fitting
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[124]:

```
Lasso(alpha=10)
```

In [125]:

```
# Lasso Score
la.score(x_test,y_test)
```

Out[125]:

```
0.9999180445774012
```

In [126]:

```
# ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[126]:

```
ElasticNet()
```

In [127]:

```
print(en.coef_)
```

```
[ 0.1125644   0.88532814 -0.1145352  -0.11413643 -0.12322048  0.0153968
 -0.          0.06958039  0.04463503  0.02165367 -0.        ]
```

In [128]:

```
print(en.intercept_)
```

```
-10.337334727763391
```

In [129]:

```
print(en.predict(x_test))
```

```
[ 1708.74471647  2270.42512248  1972.01480183  2093.87762818
  1924.40911149  2534.92625797  2486.86058479  1983.88637883
  1977.04304463  2266.48101571  2799.3313422   2239.76797742
  1741.44014465  2185.512432    1311.74885027  4149.28069196
  1785.94736114  1590.67042227  1982.16089272  2015.19762734
  2742.94176964 13449.31986998  2518.27777573  2536.63936441
  2462.37680825  2716.38426858  5183.26972409  2138.90269568
  2536.63936441  1127.86411263  3723.48594537  2814.64907303
  1306.71210919  1938.44944738  1339.28416515  1820.23685543]
```

In [130]:

```
print(en.score(x_test,y_test))
```

```
0.9999128075240743
```

In [131]:

```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 1.2821348920826698e-12
```

In [132]:

```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 8.75863638187659e-24
```

In [133]:

```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root Mean Squared Error: 2.959499346490313e-12
```

DataSet SalesWorkLoad

In [134]:

```
a=pd.read_csv(r"E:\Python Data Science\Documents\6.Salesworkload1.csv")
a
```

Out[134]:

	MonthYear	Time index	Country	StoreID	City	Dept_ID	Dept. Name	HoursOwn	Hou
0	10.2016	1.0	United Kingdom	88253.0	London (I)	1.0	Dry	3184.764	
1	10.2016	1.0	United Kingdom	88253.0	London (I)	2.0	Frozen	1582.941	
2	10.2016	1.0	United Kingdom	88253.0	London (I)	3.0	other	47.205	
3	10.2016	1.0	United Kingdom	88253.0	London (I)	4.0	Fish	1623.852	
4	10.2016	1.0	United Kingdom	88253.0	London (I)	5.0	Fruits & Vegetables	1759.173	
...	...	...	...	...	...	...	...	...	
7653	06.2017	9.0	Sweden	29650.0	Gothenburg	12.0	Checkout	6322.323	
7654	06.2017	9.0	Sweden	29650.0	Gothenburg	16.0	Customer Services	4270.479	
7655	06.2017	9.0	Sweden	29650.0	Gothenburg	11.0	Delivery	0	
7656	06.2017	9.0	Sweden	29650.0	Gothenburg	17.0	others	2224.929	
7657	06.2017	9.0	Sweden	29650.0	Gothenburg	18.0	all	39652.2	

7658 rows × 14 columns

In [135]:

```
b=a.head(15)
b
```

Out[135]:

	MonthYear	Time index	Country	StoreID	City	Dept_ID	Dept. Name	HoursOwn	HoursLeas
0	10.2016	1.0	United Kingdom	88253.0	London (I)	1.0	Dry	3184.764	0.
1	10.2016	1.0	United Kingdom	88253.0	London (I)	2.0	Frozen	1582.941	0.
2	10.2016	1.0	United Kingdom	88253.0	London (I)	3.0	other	47.205	0.
3	10.2016	1.0	United Kingdom	88253.0	London (I)	4.0	Fish	1623.852	0.
4	10.2016	1.0	United Kingdom	88253.0	London (I)	5.0	Fruits & Vegetables	1759.173	0.
5	10.2016	1.0	United Kingdom	88253.0	London (I)	6.0	Meat	8270.316	0.
6	10.2016	1.0	United Kingdom	88253.0	London (I)	13.0	Food	16468.251	0.
7	10.2016	1.0	United Kingdom	88253.0	London (I)	7.0	Clothing	4698.471	0.
8	10.2016	1.0	United Kingdom	88253.0	London (I)	8.0	Household	1183.272	0.
9	10.2016	1.0	United Kingdom	88253.0	London (I)	9.0	Hardware	2029.815	0.
10	10.2016	1.0	United Kingdom	88253.0	London (I)	14.0	Non Food	7911.558	0.
11	10.2016	1.0	United Kingdom	88253.0	London (I)	15.0	Admin	4308.243	0.
12	10.2016	1.0	United Kingdom	88253.0	London (I)	12.0	Checkout	5825.097	0.
13	10.2016	1.0	United Kingdom	88253.0	London (I)	16.0	Customer Services	3320.085	0.
14	10.2016	1.0	United Kingdom	88253.0	London (I)	11.0	Delivery	0	0.

In [136]:

```
a.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7658 entries, 0 to 7657
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   MonthYear      7658 non-null   object 
 1   Time index     7650 non-null   float64
 2   Country        7650 non-null   object 
 3   StoreID        7650 non-null   float64
 4   City           7650 non-null   object 
 5   Dept_ID        7650 non-null   float64
 6   Dept. Name     7650 non-null   object 
 7   HoursOwn       7650 non-null   object 
 8   HoursLease     7650 non-null   float64
 9   Sales units    7650 non-null   float64
 10  Turnover       7650 non-null   float64
 11  Customer       0 non-null      float64
 12  Area (m2)      7650 non-null   object 
 13  Opening hours  7650 non-null   object 
dtypes: float64(7), object(7)
memory usage: 837.7+ KB
```

In [137]:

```
a.describe()
```

Out[137]:

	Time index	StoreID	Dept_ID	HoursLease	Sales units	Turnover	Cu
count	7650.000000	7650.000000	7650.000000	7650.000000	7.650000e+03	7.650000e+03	
mean	5.000000	61995.220000	9.470588	22.036078	1.076471e+06	3.721393e+06	
std	2.582158	29924.581631	5.337429	133.299513	1.728113e+06	6.003380e+06	
min	1.000000	12227.000000	1.000000	0.000000	0.000000e+00	0.000000e+00	
25%	3.000000	29650.000000	5.000000	0.000000	5.457125e+04	2.726798e+05	
50%	5.000000	75400.500000	9.000000	0.000000	2.932300e+05	9.319575e+05	
75%	7.000000	87703.000000	14.000000	0.000000	9.175075e+05	3.264432e+06	
max	9.000000	98422.000000	18.000000	3984.000000	1.124296e+07	4.271739e+07	

In [138]:

```
a.isna()
```

Out[138]:

	MonthYear	Time index	Country	StoreID	City	Dept_ID	Dept. Name	HoursOwn	HoursLease	S u
0	False	False	False	False	False	False	False	False	False	F
1	False	False	False	False	False	False	False	False	False	F
2	False	False	False	False	False	False	False	False	False	F
3	False	False	False	False	False	False	False	False	False	F
4	False	False	False	False	False	False	False	False	False	F
...	...	...	...	...	...	...	...	...	...	
7653	False	False	False	False	False	False	False	False	False	F
7654	False	False	False	False	False	False	False	False	False	F
7655	False	False	False	False	False	False	False	False	False	F
7656	False	False	False	False	False	False	False	False	False	F
7657	False	False	False	False	False	False	False	False	False	F

7658 rows × 14 columns

In [139]:

```
c=b.fillna(value=1)
c
```

Out[139]:

	MonthYear	Time index	Country	StoreID	City	Dept_ID	Dept. Name	HoursOwn	HoursLease	Sales units
0	10.2016	1.0	United Kingdom	88253.0	London (I)	1.0	Dry	3184.764	0.0	398560.0
1	10.2016	1.0	United Kingdom	88253.0	London (I)	2.0	Frozen	1582.941	0.0	82725.0
2	10.2016	1.0	United Kingdom	88253.0	London (I)	3.0	other	47.205	0.0	438400.0
3	10.2016	1.0	United Kingdom	88253.0	London (I)	4.0	Fish	1623.852	0.0	309425.0
4	10.2016	1.0	United Kingdom	88253.0	London (I)	5.0	Fruits & Vegetables	1759.173	0.0	165515.0
5	10.2016	1.0	United Kingdom	88253.0	London (I)	6.0	Meat	8270.316	0.0	1713310.0

In [140]:

```
c.columns
```

Out[140]:

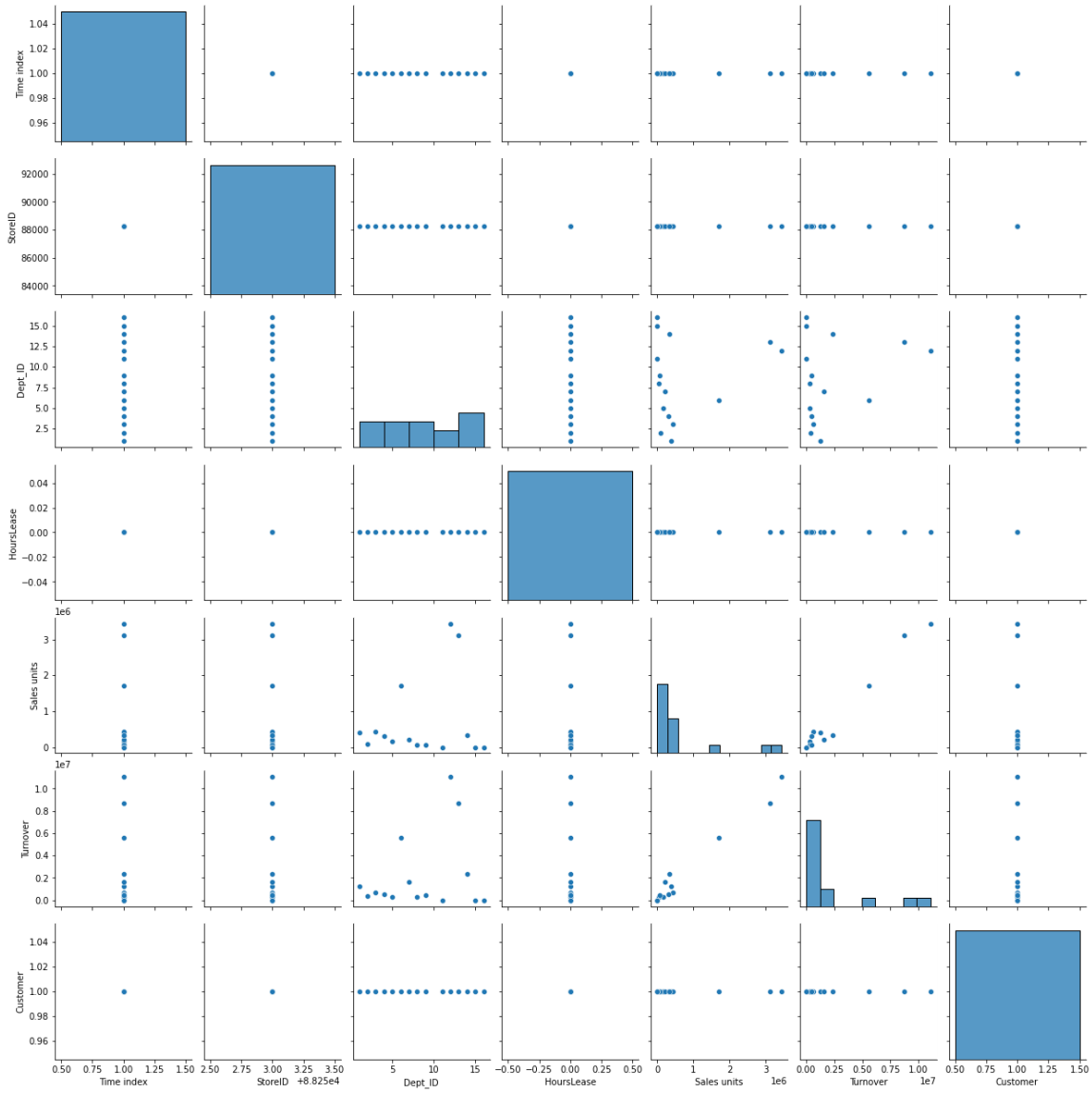
```
Index(['MonthYear', 'Time index', 'Country', 'StoreID', 'City', 'Dept_ID',
       'Dept. Name', 'HoursOwn', 'HoursLease', 'Sales units', 'Turnover',
       'Customer', 'Area (m2)', 'Opening hours'],
      dtype='object')
```

In [141]:

```
sns.pairplot(c)
```

Out[141]:

```
<seaborn.axisgrid.PairGrid at 0x1affa313820>
```



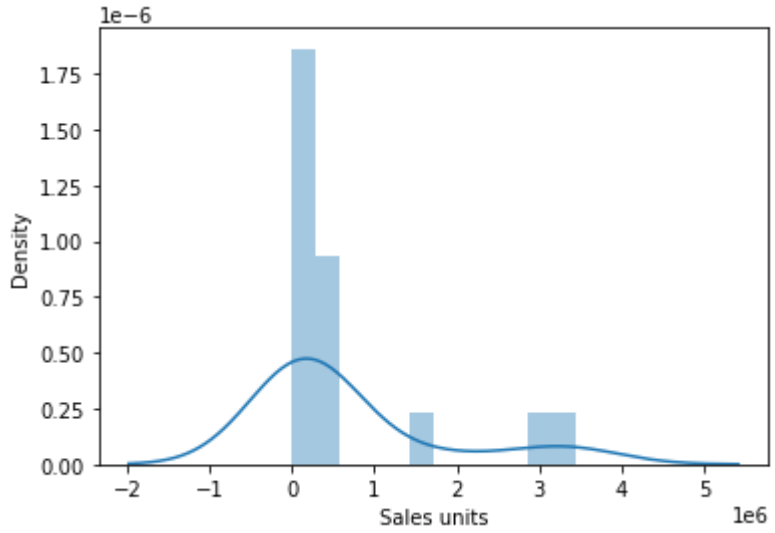
In [142]:

```
#normal distribution
sns.distplot(c["Sales units"])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[142]:

```
<AxesSubplot:xlabel='Sales units', ylabel='Density'>
```

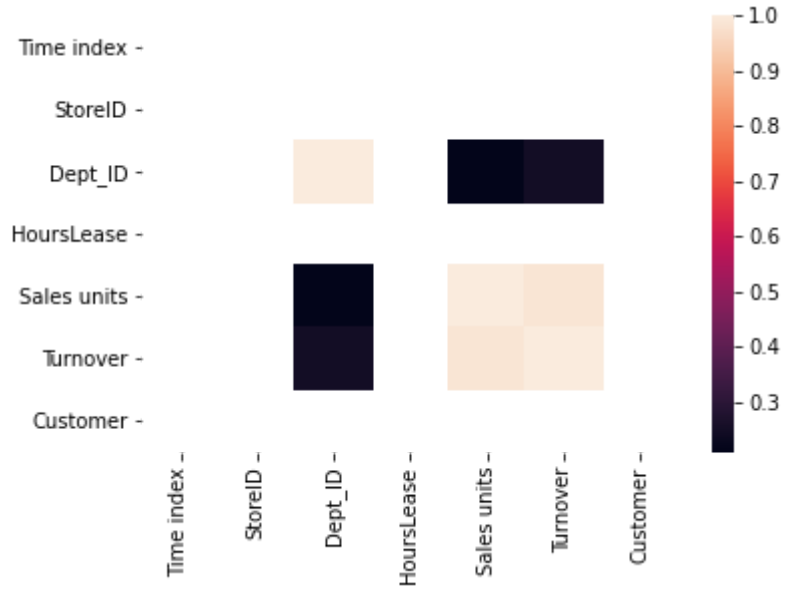


In [143]:

```
# Correlation map
sns.heatmap(c.corr())
```

Out[143]:

```
<AxesSubplot:>
```



In [144]:

```
x=c[[ 'Time index', 'StoreID', 'Dept_ID'
    , 'HoursOwn', 'HoursLease', 'Sales units', 'Turnover',
       'Customer']]
y=c[ 'Dept_ID']
```

In [145]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)


lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[145]:

```
LinearRegression()
```

In [146]:

```
print(lr.intercept_)
```

```
4.1303509412671247e-07
```

In [147]:

```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[147]:

	Co-efficient
Time index	0.000000e+00
StoreID	-4.679937e-12
Dept_ID	1.000000e+00
HoursOwn	-3.818117e-15
HoursLease	2.220446e-16
Sales units	-1.183735e-16
Turnover	3.662533e-17
Customer	0.000000e+00

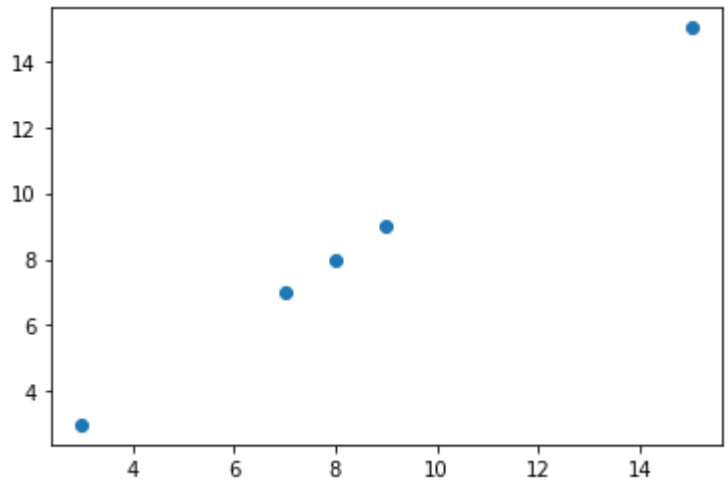
In [148]:

```
#Predicting

prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[148]:

```
<matplotlib.collections.PathCollection at 0x1affd0d9580>
```



In [149]:

```
# Score

print(lr.score(x_test,y_test))
```

```
1.0
```

In [150]:

```
#Ridge fitting
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[150]:

```
Ridge(alpha=10)
```

In [151]:

```
#Ridge Score
rr.score(x_test,y_test)
```

Out[151]:

```
0.9978864394084306
```

In [152]:

```
# Lasso Fitting
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[152]:

```
Lasso(alpha=10)
```

In [153]:

```
# Lasso Score
la.score(x_test,y_test)
```

Out[153]:

```
0.766851921400335
```

In [154]:

```
# ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[154]:

```
ElasticNet()
```

In [155]:

```
print(en.coef_)
```

```
[ 0.00000000e+00  0.00000000e+00  9.50698238e-01  2.38266072e-05
  0.00000000e+00 -3.74196671e-07  1.22186628e-07  0.00000000e+00]
```

In [156]:

```
print(en.intercept_)
```

```
0.2831411742027736
```

In [157]:

```
print(en.predict(x_test))
```

```
[ 7.93185442  7.16739219  3.05230313 14.64626556  8.92059984]
```

In [158]:

```
print(en.score(x_test,y_test))
```

```
0.9977814890372924
```

In [159]:

```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 1.986002473586268e-11
```

In [160]:

```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 4.97748323620404e-22
```

In [161]:

```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root Mean Squared Error: 2.23102739476772e-11
```

DataSet Uber

In [162]:

```
a=pd.read_csv(r"E:\Python Data Science\Documents\7.uber - uber.csv")
a
```

Out[162]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude
0	24238194	2015-05-07 19:52:06	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354
1	27835199	2009-07-17 20:04:56	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225
2	44984355	2009-08-24 21:45:00	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770
3	25894730	2009-06-26 8:22:21	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844
4	17610152	2014-08-28 17:47:00	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085
...	...	...	...	...	...	...
199995	42598914	2012-10-28 10:49:00	3.0	2012-10-28 10:49:00 UTC	-73.987042	40.739367
199996	16382965	2014-03-14 1:09:00	7.5	2014-03-14 01:09:00 UTC	-73.984722	40.736837
199997	27804658	2009-06-29 0:42:00	30.9	2009-06-29 00:42:00 UTC	-73.986017	40.756487
199998	20259894	2015-05-20 14:56:25	14.5	2015-05-20 14:56:25 UTC	-73.997124	40.725452
199999	11951496	2010-05-15 4:08:00	14.1	2010-05-15 04:08:00 UTC	-73.984395	40.720077

200000 rows × 9 columns

In [163]:

```
b=a.head(1000)
b
```

Out[163]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dı
0	24238194	2015-05-07 19:52:06	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	
1	27835199	2009-07-17 20:04:56	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	
2	44984355	2009-08-24 21:45:00	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	
3	25894730	2009-06-26 8:22:21	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	
4	17610152	2014-08-28 17:47:00	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	
...	...	...	...	...	...	...	
995	13439193	2011-05-04 6:39:00	5.7	2011-05-04 06:39:00 UTC	-73.969720	40.757577	
996	32405310	2011-11-23 20:43:20	8.1	2011-11-23 20:43:20 UTC	-73.993784	40.757054	
997	51612001	2010-01-11 20:58:00	8.5	2010-01-11 20:58:00 UTC	-73.972338	40.765078	
998	937243	2013-06-12 17:01:24	5.5	2013-06-12 17:01:24 UTC	-73.979054	40.784730	
999	47946613	2011-12-11 21:48:22	9.7	2011-12-11 21:48:22 UTC	-73.983675	40.729944	

1000 rows × 9 columns

In [164]:

```
b.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         1000 non-null   int64  
 1   key                1000 non-null   object 
 2   fare_amount        1000 non-null   float64
 3   pickup_datetime    1000 non-null   object 
 4   pickup_longitude   1000 non-null   float64
 5   pickup_latitude    1000 non-null   float64
 6   dropoff_longitude  1000 non-null   float64
 7   dropoff_latitude   1000 non-null   float64
 8   passenger_count    1000 non-null   int64  
dtypes: float64(5), int64(2), object(2)
memory usage: 70.4+ KB
```

In [165]:

```
b.describe()
```

Out[165]:

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	drop
count	1.000000e+03	1000.000000	1000.000000	1000.000000	1000.000000	1
mean	2.744197e+07	11.116280	-72.646044	40.017119	-72.717954	
std	1.616669e+07	9.216902	9.840422	5.420590	9.567772	
min	6.250000e+02	2.500000	-74.689831	0.000000	-74.689831	
25%	1.305167e+07	6.075000	-73.992702	40.735620	-73.991797	
50%	2.724411e+07	8.500000	-73.982114	40.752500	-73.980328	
75%	4.185744e+07	12.500000	-73.969435	40.765824	-73.963108	
max	5.525106e+07	93.160000	0.001782	40.850558	0.000875	

In [166]:

```
b.isna()
```

Out[166]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	drop
0	False	False	False	False	False	False	
1	False	False	False	False	False	False	
2	False	False	False	False	False	False	
3	False	False	False	False	False	False	
4	False	False	False	False	False	False	
...	...	...	...	...	...	...	
995	False	False	False	False	False	False	
996	False	False	False	False	False	False	
997	False	False	False	False	False	False	
998	False	False	False	False	False	False	
999	False	False	False	False	False	False	

1000 rows × 9 columns

In [167]:

```
print(b.columns)
```

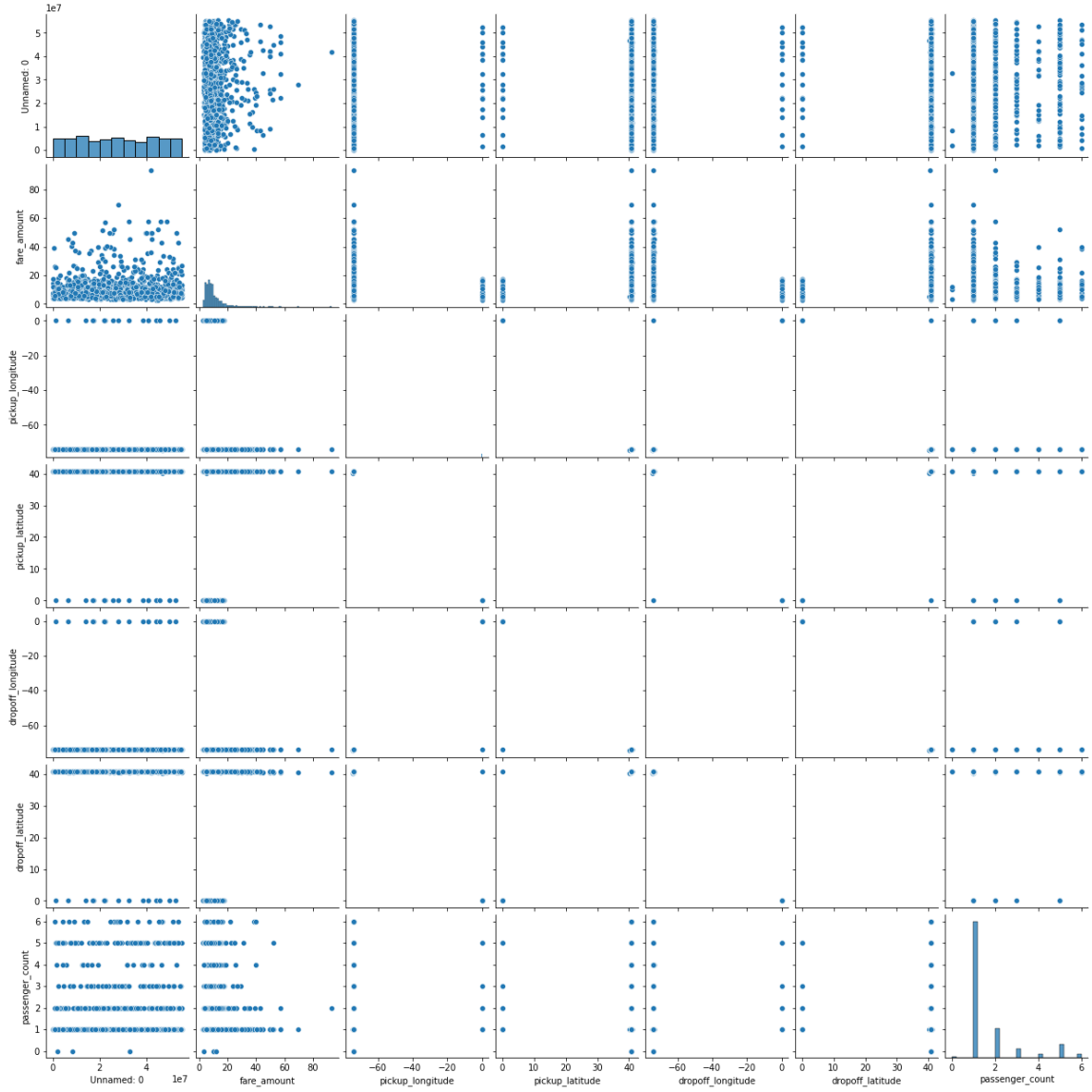
```
Index(['Unnamed: 0', 'key', 'fare_amount', 'pickup_datetime',
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'passenger_count'],
      dtype='object')
```

In [168]:

```
sns.pairplot(b)
```

Out[168]:

```
<seaborn.axisgrid.PairGrid at 0x1aff3636c40>
```



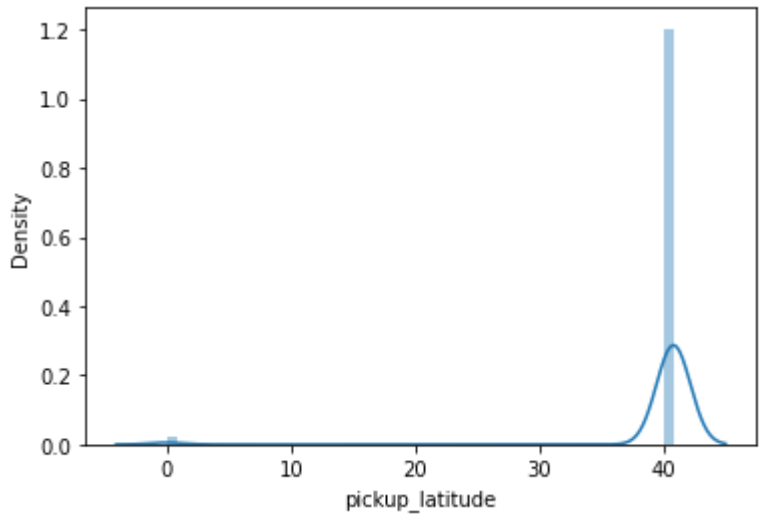
In [169]:

```
#normal distribution
sns.distplot(b["pickup_latitude"])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[169]:

```
<AxesSubplot:xlabel='pickup_latitude', ylabel='Density'>
```

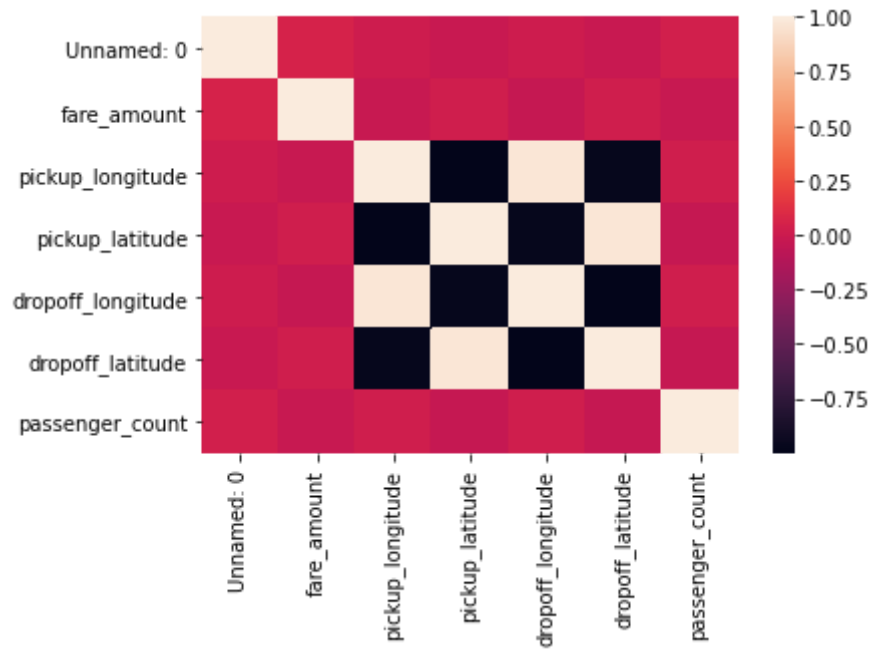


In [170]:

```
# Correlation map
sns.heatmap(b.corr())
```

Out[170]:

```
<AxesSubplot:>
```



In [171]:

```
x=b[['Unnamed: 0', 'fare_amount',
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'passenger_count']]
y=b[ 'fare_amount']
```

In [172]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)


lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[172]:

```
LinearRegression()
```

In [173]:

```
print(lr.intercept_)
```

```
-4.121147867408581e-13
```

In [174]:

```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[174]:

	Co-efficient
Unnamed: 0	1.585221e-20
fare_amount	1.000000e+00
pickup_longitude	-4.650093e-16
pickup_latitude	-1.828973e-15
dropoff_longitude	9.413685e-16
dropoff_latitude	2.290411e-15
passenger_count	4.840940e-18

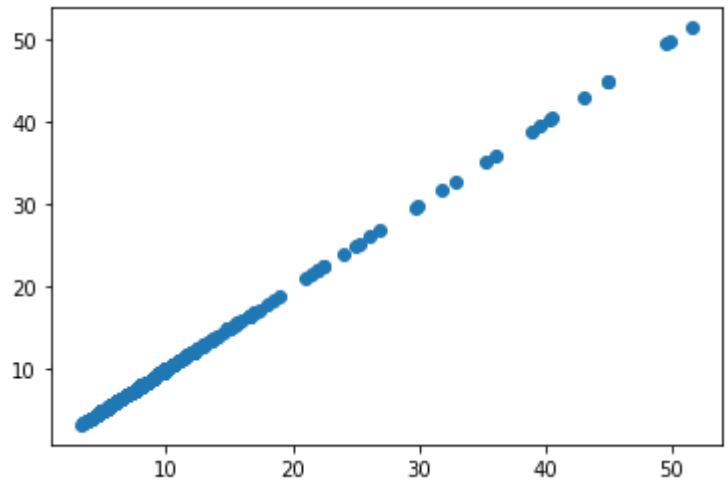
In [175]:

```
#Predicting

prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[175]:

```
<matplotlib.collections.PathCollection at 0x1af9daad400>
```



In [176]:

```
# Score

print(lr.score(x_test,y_test))
```

```
1.0
```

In [177]:

```
#Ridge fitting
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_ridge.py:
147: LinAlgWarning: Ill-conditioned matrix (rcond=4.04705e-17): result may
not be accurate.
  return linalg.solve(A, Xy, sym_pos=True,
```

Out[177]:

```
Ridge(alpha=10)
```

In [178]:

```
#Ridge Score
rr.score(x_test,y_test)
```

Out[178]:

```
0.999999967781721
```

In [179]:

```
# Lasso Fitting
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[179]:

```
Lasso(alpha=10)
```

In [180]:

```
# Lasso Score
la.score(x_test,y_test)
```

Out[180]:

```
0.9876966271035826
```

In [181]:

```
# ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[181]:

```
ElasticNet()
```

In [182]:

```
print(en.coef_)
```

```
[ 5.12498747e-10  9.89018696e-01 -0.00000000e+00  0.00000000e+00
 -0.00000000e+00  0.00000000e+00 -0.00000000e+00]
```

In [183]:

```
print(en.intercept_)
```

```
0.10982732375875592
```

In [184]:

```
print(en.predict(x_test))
```

```
[ 7.35777606 11.68569429 12.09515838  7.73276997 17.22751792 12.07699923
  9.71141757  4.96440756  4.18067998  4.16812482 16.9477906  11.30304609
  4.58173677  4.97213683 35.73590884 10.00544768  7.53988959  4.57169296
  9.31456459 11.28719486 14.9626999   7.74665196  9.3209422  10.09992464
  7.73181947  5.76151125  4.97763926  7.54843053 10.51186491  3.58524389
  3.77113299 10.50283008  8.53260512  6.95695764 16.43566874  5.55543652
  8.91439784  6.14451634 12.09062144  8.03343256  5.7752799   8.53007702
  6.55463341  7.5368188  29.5940268  10.51759486  6.54214527 21.400441
  8.02567497  7.73717824  6.15840932  8.12753773 10.50928427 15.45280959
  6.56255615 14.94614541  8.93917524 22.38191746  8.53206466  6.04791493
  9.70882086  6.5450813   7.33830116  6.95886708  5.07478313  4.17416743
  7.34719507 16.43756451  4.06878632  5.5578824   4.57721333  7.52756809
  9.32285545  9.52152925 35.05979164  3.79508866 16.8439356  14.87387343
 13.96195089  4.18101328  6.5647628  15.94701463 14.84994243 21.88771163
  8.52369655  7.34944111 26.72930525  8.52891205  8.93354821  4.09080838
 12.8747994  14.86814431 10.10695511  5.0568413   6.14979124 14.8714112
  7.74264452 44.63740769  6.15680168  7.75250476 10.51464835 23.86595404
  5.05566046 14.94968386  5.08300258  8.03791091  7.73118368  6.93550142
 21.87174754  5.77003168  7.34003472  5.75849649  4.17207679  7.35761167
  9.72349945  9.7210697   5.36604259 49.38336992  5.37559934  8.51871407
  8.52596893  4.57450861 49.1624708  40.16928821 29.498563    8.92278435
  7.74800049  8.51975549 16.43584931  6.95610588 20.89355657  6.06008829
  5.7474473  10.99300541  6.04487571 11.98351461  4.18830911 15.64085098
 31.60125954  6.96224199  4.16525465  8.52777549 12.97211184  5.07536452
  9.71509738  6.0627161   5.37893108  9.01439939  5.55704372  4.9675868
  6.93831465  8.13795934 10.00292151  5.56470255 12.49580347 11.00397383
  7.73134167 11.69035382 10.01873761  8.13914749 12.08595214  7.55242928
 13.47620407  8.22257975  5.75094588 13.6748035  12.99535963 16.43841633
 10.1196691  15.2577147  32.67412188  6.06810479 11.48424896 13.66601828
 14.9668209   4.97602975  4.19057798  5.55872673  4.58124282  7.73814613
 10.01427463  8.14028821  4.09329925 38.51366204  4.95869359  8.03170583
 12.07942125  7.33276929  5.56976925  4.95726505 10.02778087  6.04661203
  5.75983732  4.58778592 13.46611274 12.48526086  5.57742166 13.26807524
  8.0427782  12.08313675  5.35613188 15.44512499  5.08309518  4.57493913
  8.54340379  4.56408681  3.58186375  3.59713442  8.34214241  6.93533893
  4.57544069 10.11142974 14.47581468 18.91656694 11.00674054 14.07607709
 10.51663818 42.66142953  4.58143275  9.02207458  7.53195616  7.33924124
  9.03784941 18.41152937 22.38045026  4.58088475 51.05524992 11.00855549
  8.14640026  9.72876709 11.70292294  7.35756275  6.54470133  9.0114267
  7.3301398   4.58757616 44.63256525  4.16960051  7.74345362  8.92867526
 12.00035549  3.79464295 11.49913951  8.14410236  4.97739994  5.56581249
  7.72621576  8.03194585 10.5094653   7.75141199  6.56348885 40.00879163
  5.75455259  7.35749334  4.56748485 12.00079526  4.56404157 12.09939073
 16.4502912  25.1502232   7.75058652  7.34009597  6.9397128  15.64341596
  5.35925613  8.12198273 10.51861158  9.51069547  8.03061145  7.55166833
  8.12569869 10.10971985  7.54912897  7.34787732  6.161881    5.77001537
 17.91792549 12.10271786  9.51780653  8.54457473  4.58841586 25.93359757
 10.99824137  3.77045046  5.36143429  9.0209976   5.37528255 13.48296524
  3.77960968  4.07204021  5.05970429  6.16764446  7.54084627 10.01640789
 10.52007219 24.84058583  3.3889495   5.452654   39.18862613 12.47551647]
```

In [185]:

```
print(en.score(x_test,y_test))
```

```
0.9998783136265867
```

In [186]:

```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 2.261331862503842e-13
```

In [187]:

```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 6.746784825245631e-26
```

In [188]:

```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root Mean Squared Error: 2.597457376983428e-13
```

DataSet Cancer

In [189]:

```
a=pd.read_csv(r"E:\Python Data Science\Documents\8_BreastCancerPrediction - 8_BreastCance
a
```

Out[189]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothn
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	
...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	
565	926682	M	20.13	28.25	131.20	1261.0	
566	926954	M	16.60	28.08	108.30	858.1	
567	927241	M	20.60	29.33	140.10	1265.0	
568	92751	B	7.76	24.54	47.92	181.0	

569 rows × 32 columns

In [190]:

```
a.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

In [191]:

```
a.describe()
```

Out[191]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.0
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.0
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.0
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.0
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.0
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.0
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.1
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.1

8 rows × 31 columns

In [192]:

```
a.isna()
```

Out[192]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_
0	False	False	False	False	False	False	
1	False	False	False	False	False	False	
2	False	False	False	False	False	False	
3	False	False	False	False	False	False	
4	False	False	False	False	False	False	
...	...	...	...	...	...	...	
564	False	False	False	False	False	False	
565	False	False	False	False	False	False	
566	False	False	False	False	False	False	
567	False	False	False	False	False	False	
568	False	False	False	False	False	False	

569 rows × 32 columns

In [193]:

```
a.columns
```

Out[193]:

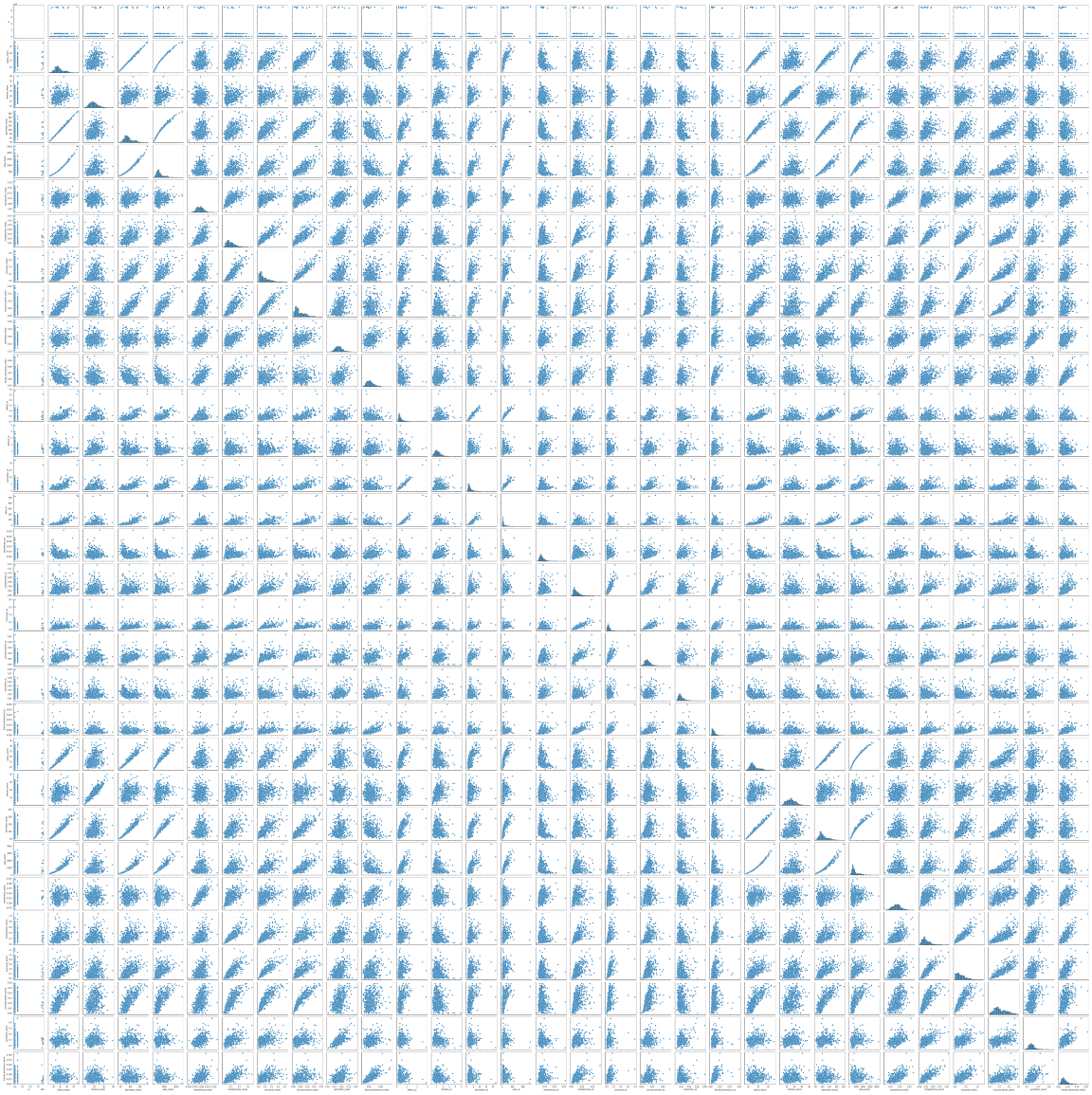
```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mea
n',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_s
e',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_s
e',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

In [194]:

```
sns.pairplot(a)
```

Out[194]:

```
<seaborn.axisgrid.PairGrid at 0x1af9ec2abe0>
```



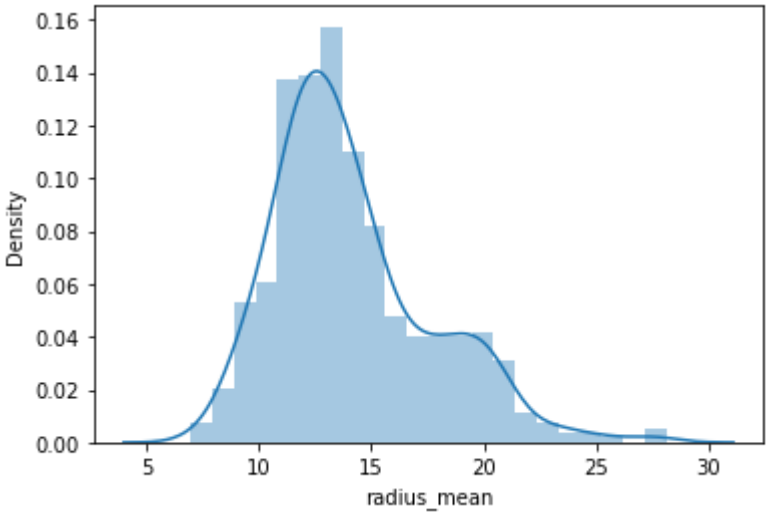
In [195]:

```
#normal distribution
sns.distplot(a["radius_mean"])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[195]:

```
<AxesSubplot:xlabel='radius_mean', ylabel='Density'>
```

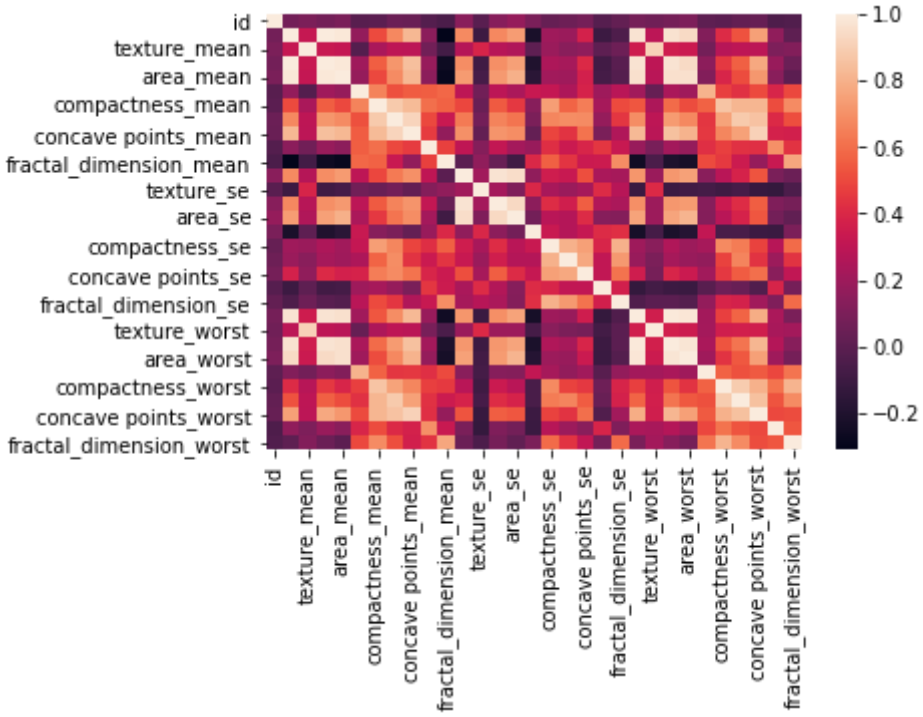


In [196]:

```
# Correlation map
sns.heatmap(a.corr())
```

Out[196]:

```
<AxesSubplot:>
```



In [197]:

```
x=a[['id', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se']]
y=a['radius_mean']
```

In [198]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)


lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[198]:

```
LinearRegression()
```

In [199]:

```
print(lr.intercept_)
```

```
-1.099564883588755e-12
```

In [200]:

```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[200]:

	Co-efficient
id	2.533739e-20
radius_mean	1.000000e+00
texture_mean	3.572192e-16
perimeter_mean	-5.465799e-16
area_mean	8.242682e-17
smoothness_mean	1.676818e-14
compactness_mean	-3.472568e-15
concavity_mean	-1.186681e-14
concave points_mean	3.537624e-14
symmetry_mean	3.483451e-15
fractal_dimension_mean	-1.192330e-14
radius_se	-7.006197e-15
texture_se	-5.584958e-16
perimeter_se	-2.650578e-16
area_se	4.021869e-17
smoothness_se	-1.995293e-14
compactness_se	-7.646949e-15
concavity_se	7.186394e-15
concave points_se	2.963253e-14
symmetry_se	1.166081e-14
fractal_dimension_se	-1.935157e-14

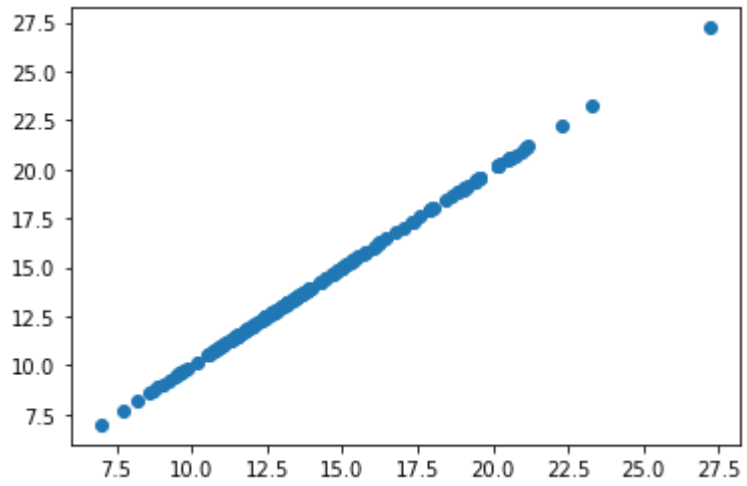
In [201]:

```
#Predicting

prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[201]:

```
<matplotlib.collections.PathCollection at 0x1afcca368e0>
```



In [202]:

```
# Score

print(lr.score(x_test,y_test))
```

```
1.0
```

In [203]:

```
#Ridge fitting
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_ridge.py:
147: LinAlgWarning: Ill-conditioned matrix (rcond=1.35258e-18): result may
not be accurate.
  return linalg.solve(A, Xy, sym_pos=True,
```

Out[203]:

```
Ridge(alpha=10)
```

In [204]:

```
#Ridge Score
rr.score(x_test,y_test)
```

Out[204]:

```
0.9993669836813074
```

In [205]:

```
# Lasso Fitting
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[205]:

```
Lasso(alpha=10)
```

In [206]:

```
# Lasso Score
la.score(x_test,y_test)
```

Out[206]:

```
0.9766791616816647
```

In [207]:

```
# ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[207]:

```
ElasticNet()
```

In [208]:

```
print(en.coef_)
```

```
[-5.09142885e-11  0.00000000e+00  0.00000000e+00  7.85289461e-02
  5.14092936e-03 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00  0.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -5.82581273e-03 -0.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
 -0.00000000e+00]
```

In [209]:

```
print(en.intercept_)
```

```
3.777973583643554
```

In [210]:

```
print(en.predict(x_test))
```

```
[17.2271819  14.80698514 14.57591782 19.3435149  12.31710543 13.7643844
 13.11451102 18.72293323 12.32872921 14.71829788 11.5977481   9.15466748
 19.4291046   9.99016522 18.60670541 12.6748452  10.3656592   9.76847173
 14.49645147 13.2448104  10.89083013 11.92392604 12.68041823 20.89444721
 11.23329782 15.70828306  9.29638818 11.17554717 14.83702436 13.05181878
 12.24990574 15.08537872 18.91765206 17.36290341 11.49480288 14.32011209
 12.17553321 11.76238854 10.16576834 20.44429769 20.56138441 12.56529508
 13.01381729 13.12954386 19.98664755 12.07056588  9.8005446  12.55639925
 15.77932093 11.39285478 14.34396853 17.09793345 13.61577014 11.5798597
 13.50195424 15.78988879 19.60120878 17.72986169 12.65192736 12.49435549
 17.03291821 12.00029233 12.65911162 11.91385563 12.62920639 10.7013396
 11.60540449  9.57950726  7.89715035 13.84651129 18.5930876  13.25300125
 12.49935117 28.89535914 16.40142469  9.83261216 12.80852426 13.66396932
 13.37941536 11.61603744 12.56984177 13.64519397 12.12414959  9.15557956
 13.36256047 10.7702631  11.55914475 11.44625374 14.79521819 14.79280212
 13.62796534 15.60880435 16.52483925 13.44854375 14.55362969 10.8544939
 18.34432901 13.41973405 11.58630403 21.21440054 14.72837227 19.30795386
 10.91251422 20.77806622 11.23309609 16.13404417 12.84073104 13.64791585
 13.57104813 19.85016254 15.42735308 20.40546512 12.62007208 10.88756083
 11.73301023 20.78447717 11.39067476 13.98505774 13.58545678 13.67979729
 13.48680252 12.87553158 13.48125715 12.35194426 15.00194641 12.24548592
 11.94718596 11.89813952  9.93461943 20.1614222  20.94782417 13.36560974
 13.86452833 11.4839055  18.84554827 18.54990136 18.4106241  20.81216129
 11.48616234 12.89018577 15.95406257 13.28533931 22.54446638 17.67367589
 16.45458614 23.81933382 13.15697104 12.91381961 14.3013234  20.41061235
 15.21864293  8.82449418 14.07628229 13.29936976 13.17978466 10.03899651
 11.09631437 15.61227596 21.22239831 11.21492534 15.49933779 21.24919739
 11.74240528  8.35343763 10.74706532 12.44683133 19.13248258 17.69244832
 13.13695288 14.05809164 15.18639061]
```

In [211]:

```
print(en.score(x_test,y_test))
```

```
0.9947140210151686
```

In [212]:

```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 1.1888761580141566e-12
```

In [213]:

```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 9.494913742869005e-24
```

In [214]:

```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root Mean Squared Error: 3.0813817911562023e-12
```

DataSet WineEquality

In [215]:

```
a=pd.read_csv(r"E:\Python Data Science\Documents\11_winequality-red - 11_winequality-red
a
```

Out[215]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56
...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66

1599 rows × 12 columns

In [216]:

```
a.head(12)
```

Out[216]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alc
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	
9	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	
10	6.7	0.58	0.08	1.8	0.097	15.0	65.0	0.9959	3.28	0.54	
11	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	

In [217]:

```
a.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In [218]:

```
a.describe()
```

Out[218]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total d
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.0
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.4
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.8
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.0
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.0
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.0
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.0
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.0

In [219]:

```
a.isna()
```

Out[219]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
0	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...
1594	False	False	False	False	False	False	False	False	False	False
1595	False	False	False	False	False	False	False	False	False	False
1596	False	False	False	False	False	False	False	False	False	False
1597	False	False	False	False	False	False	False	False	False	False
1598	False	False	False	False	False	False	False	False	False	False

1599 rows × 12 columns

In [220]:

```
a.columns
```

Out[220]:

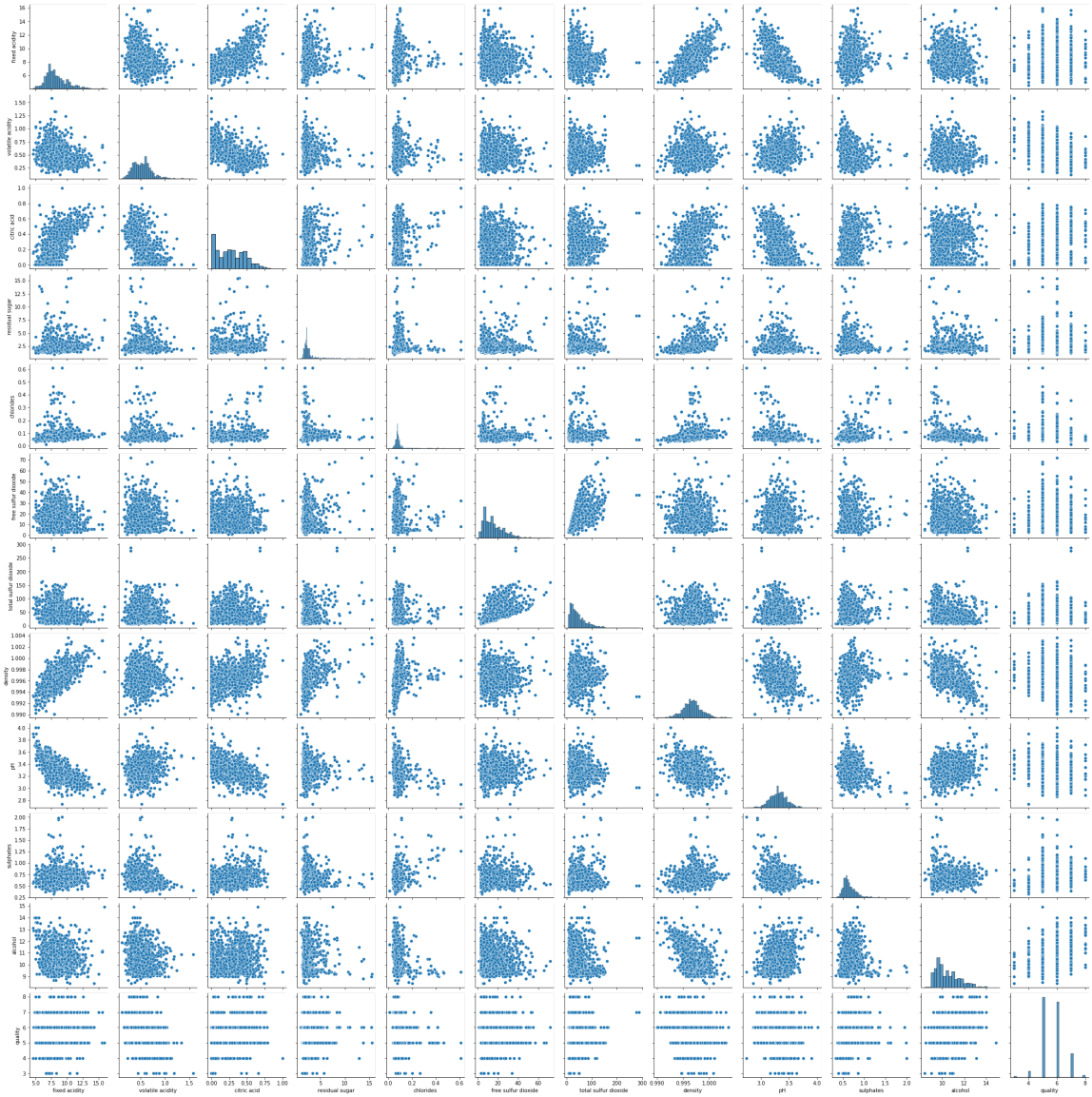
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual suga
r',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'densit
y',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [221]:

```
sns.pairplot(a)
```

Out[221]:

```
<seaborn.axisgrid.PairGrid at 0x1afcd764be0>
```



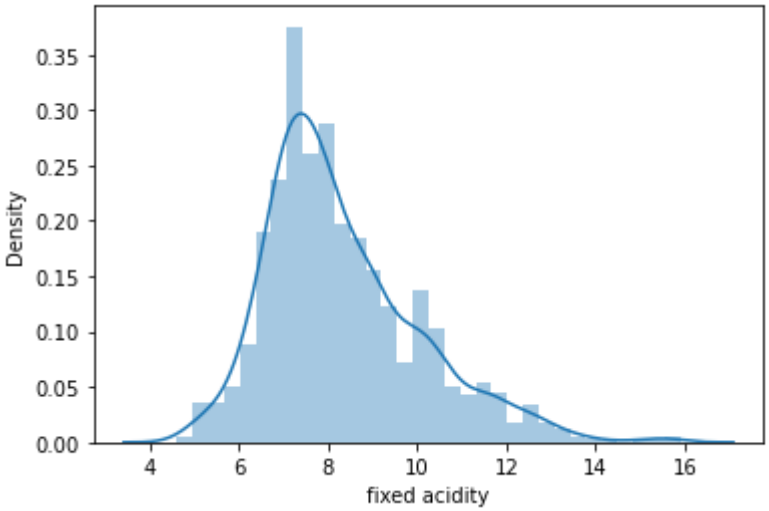
In [222]:

```
#normal distribution
sns.distplot(a['fixed acidity'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[222]:

```
<AxesSubplot:xlabel='fixed acidity', ylabel='Density'>
```

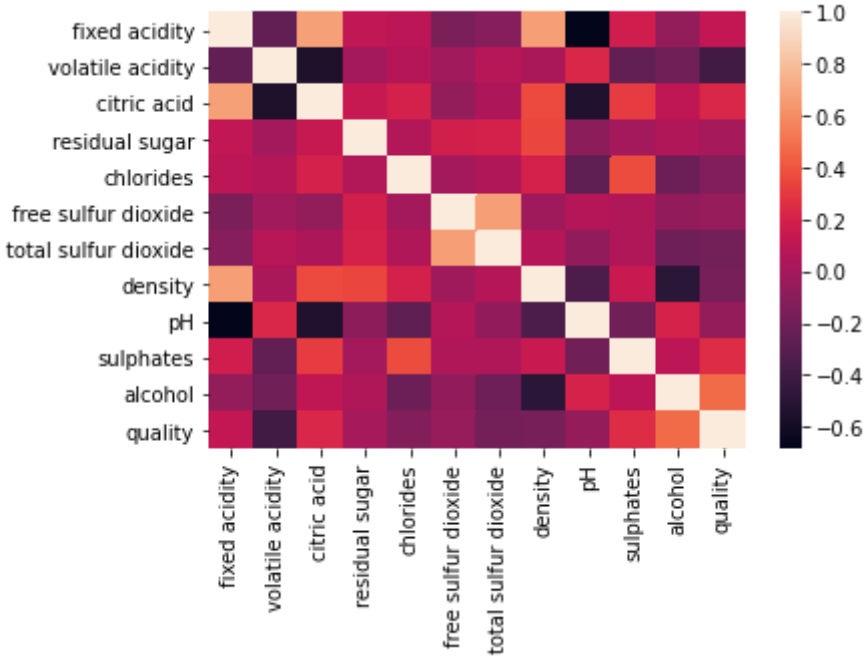


In [223]:

```
# Correlation map
sns.heatmap(a.corr())
```

Out[223]:

```
<AxesSubplot:>
```



In [224]:

```
x=a[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality']]
y=a['fixed acidity']
```

In [225]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)


lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[225]:

```
LinearRegression()
```

In [226]:

```
print(lr.intercept_)
```

```
4.796163466380676e-14
```

In [227]:

```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[227]:

	Co-efficient
fixed acidity	1.000000e+00
volatile acidity	-6.867950e-17
citric acid	4.432490e-16
residual sugar	2.825829e-16
chlorides	-1.065192e-16
free sulfur dioxide	1.503163e-15
total sulfur dioxide	-2.950678e-16
density	-6.093525e-14
pH	-3.990648e-17
sulphates	1.415208e-16
alcohol	-6.004868e-17
quality	1.827960e-17

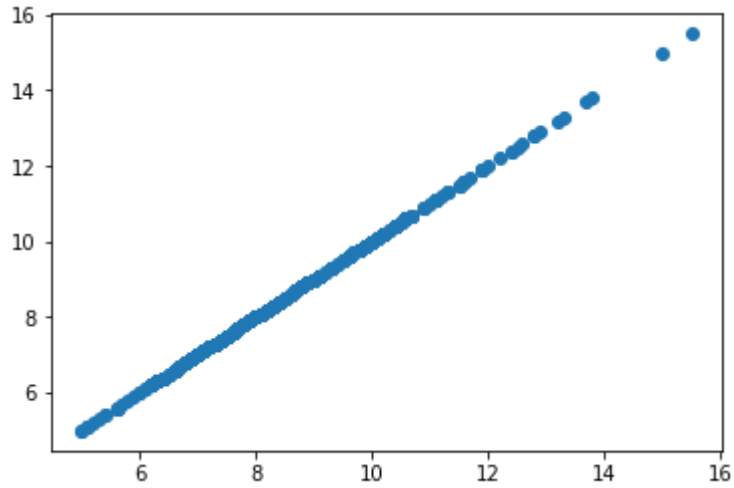
In [228]:

```
#Predicting

prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[228]:

```
<matplotlib.collections.PathCollection at 0x1afdd241880>
```



In [229]:

```
# Score

print(lr.score(x_test,y_test))
```

```
1.0
```

In [230]:

```
#Ridge fitting
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[230]:

```
Ridge(alpha=10)
```

In [231]:

```
#Ridge Score
rr.score(x_test,y_test)
```

Out[231]:

```
0.9999859286841725
```

In [232]:

```
# Lasso Fitting
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[232]:

```
Lasso(alpha=10)
```

In [233]:

```
# Lasso Score
la.score(x_test,y_test)
```

Out[233]:

```
-2.679837826313758e-06
```

In [234]:

```
# ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[234]:

```
ElasticNet()
```

In [235]:

```
print(en.coef_)
```

```
[ 0.72004476 -0.          0.          0.          0.         -0.00090433
 -0.00119676  0.         -0.          0.         -0.          0.        ]
```

In [236]:

```
print(en.intercept_)
```

```
2.399170785117132
```

In [237]:

```
print(en.predict(x_test))
```

```
[ 7.99006844 12.31241102  9.07611938  9.26992513  9.80311591  7.54870584
  7.67522765  8.6762574   6.41554406  7.83167133  6.95769702  7.40770232
  9.58019685  8.25292734  7.39299467  8.35768155  8.3735414   7.57876963
 11.85102298  9.71785638  7.18270921  8.73655965  7.67686449  8.58701366
  7.4112751   9.86037614  7.43073032 10.002284    8.31229822  8.3099047
  7.09537565  6.84617242  8.57372202  7.73848124  9.58388577  8.2123581
  7.46503574 11.57409539  8.56526356 10.90658748  8.30204573  7.91112174
  7.55013808  9.76231788  9.80311591  7.68881459  9.35634482 10.00377319
  8.77339383  9.34587595  7.95738235  6.88239469  7.27869344  7.61308969
  6.7461782   9.51302396 10.71275468  7.67522765  8.50487061  7.65553409
  7.54870584  9.76231788  9.55318304  8.11551409 10.73940287 10.51229913
  6.81876753 10.72950932  8.63221561  7.46610236  9.40681716  7.50615325
  7.69695357  7.55503617  8.91515405  8.21174619  7.40107957  7.05097779
  9.23845998  7.33391622  7.32072482  9.14182058  7.37331575 10.04758619
  7.32694409  9.38607162  7.72091264  5.95581704  8.65615081  7.67353959
  9.89073235  7.47430785  8.35575515  7.56671134  8.18614712  7.40577306
  7.59222641  6.2418274   8.86217999  8.74227807  7.91988218 10.60625545
  9.84896635  7.54870584  9.2915209   5.99936465  7.0308537   7.31415616
  8.23734241  7.16650284  9.79944448  7.44226071  7.19708783  7.94842018
  7.54094043  7.38012274  9.40255133  8.64474102  7.92385558  8.47912675
  7.29942147  7.01259635  7.99652605  9.87024265 10.92906053  9.79798234
  8.40829199 10.25180385  7.74679206  8.20877737  8.01467919  7.46920133
  8.26997441  9.5019066   7.33884805  7.88654759  7.81879939  7.57273173
  8.13256116  8.9350888   8.91716445  6.67475858 10.65630808  7.91544166
  7.7543045   8.64906094  8.83070021  7.9574094   7.88870278  7.12298227
  7.56671134  7.91754275 10.21827038  6.78185221  8.2068656   9.90281769
  7.78781378  7.74536937  8.42201129  7.33298198  8.20906025  7.5173613
  7.66574713  9.36600958  8.71367643  6.86505519  6.84278387  8.03178035
  7.40233042  8.05454298  7.54365342  7.34360804 11.41362423  9.07970966
  8.21025701  7.81474199  8.33328209 10.90658748  9.85707829  7.49938285
  8.13585901  8.30809317  7.78631504  8.81813821  8.97565518  7.28975384
  6.00246363 10.78983862  7.5253555   7.75453329  6.09442704  7.81903773
  7.85809073 10.29269543  7.1701663   9.19591025  9.43792337  8.04289771
  6.92650012  7.70622568  7.34200112  9.64103782  7.54870584  7.46619591
  7.12298227  9.55945926  8.61231077  9.46527416  7.63600282 10.36738585
  6.82206539  7.80050147  7.80055557  8.76320785  7.67026878  9.51532679
  7.493372    9.85678586  8.12955574  8.2267434   8.71309158  6.87521413
 11.28552923  6.41043755  9.65154328 10.03144345  7.26043211  9.71785638
  7.04557885  8.25292734  7.66227458  7.13211627  6.31525456  8.99268761
  8.29639381  9.16770922 10.002284    7.41044236 11.18392523  7.47168554
  8.6410871   7.78127217  7.87445939  9.4826948   7.62686595  9.19587365
  7.74679206  8.63221561  7.12957797  7.9216272   6.2178922   6.90880455
  9.25074992  6.47193656  7.56671134  8.17789039  8.84390912  8.45779636
  8.54360414  7.77534247  9.68551681  7.98913706  7.49202091  7.26398579
  7.79563329  9.25480446  7.77971934  7.44550447  9.91951824 10.32095054
  5.86059236  6.91990441  7.65841892  7.19081161  7.40207459  5.94980619
  8.2601079   5.87742814  8.20622665  9.42715253  6.89797962  7.65841892
  7.32939457  7.96456291  7.55089762  7.52594036  7.47168554 10.38577446
  7.23731055  7.61113624  6.83467194  7.20249632  7.54572747  8.14289193
  7.57611073  7.17627579  8.18387419  7.53522201  7.78062367  8.18539043
  7.9028227   7.93732986  8.93418447  9.92021128  7.70952353  8.99338066
  8.00174075  8.76799489  8.21446874  6.29756854  8.19393958  7.46610236
  7.48361655  7.25318791 10.07997985  7.92702614  7.99001149  7.43598448
  7.1496941   7.90426066  8.90750923 11.64133987  7.66875255  8.89127294
  7.75695385  7.98855221  7.93937686  8.19686386  8.83882965  9.68788329
  8.69356985  8.93120609  8.18539043  9.44367838  8.79712729  8.71309158
  7.43531848  6.65289743  7.98884464  7.70839041  8.7419045   8.62013028
  8.33687237  9.20130919  7.60571025 10.51229913  9.43825239  7.97203304
 10.50751209  7.34200112  8.71008615  7.20129956  7.97568028  7.12808878
  8.28704852 10.35991286 10.01040549 10.17962754  9.54956571  8.39302653
  8.2820327  10.65721242  8.92399849  9.64675624  7.72876238  7.78421395
```

```
  7.48659492 11.57409539  6.44394396  7.40207459  8.35813117  9.62997455
  7.23372027  7.34844917  7.42154222  9.22676063  7.40233042  9.25480446
 11.89696412  7.70800729  8.50735482  7.39719686  9.93626333  7.61680056
  8.05083496  7.94829958  7.11037572  8.91051465  6.86135672  8.04021176
  9.30209     7.63465841  6.97716465  8.76352732  7.01259635  7.98945654
  8.20912389  7.87995189  8.35768155  7.52447822  7.3024269   7.83167133
 10.19941465  7.9096596   8.60040681  9.78028678  7.49140901  7.31307714
  9.79345114  7.86705291  6.71288021  8.00808348  7.36661186  7.83335939
  7.77709703 12.19694962  6.57216911  9.54746462 10.72921689  8.05755126
 10.92812916  7.37728915  7.5572009   8.14843851  9.13922532  7.96304668
  7.45364345  9.05542793 11.00553257  7.06508308  7.45364345 13.16207656
  7.34769249  7.85472924  9.70346821  6.93317696  7.86053835  7.4928982
  7.95554664  7.06814259  8.03186149  8.0509651  11.14954153  7.55290803
 11.22073236  7.37477789  7.02704259 10.00558186  7.56319425  8.26816574
  8.41637688  8.52472136  8.13256116  8.331181    7.22467695  9.38607162
  7.39240982  6.8364806   7.4656747   8.0476577   7.46610236 13.5232957
  9.58019685  8.57372202  8.07979553  9.71785638  8.74744823  8.9688606
  8.5267655   9.18446386  7.52120741  7.80558094  7.73567756  9.92549249
  7.27898586  8.32852209  8.65512588  8.5885474   7.69274184 10.37698698]
```

In [238]:

```
print(en.score(x_test,y_test))
```

```
0.9224034092831258
```

In [239]:

```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 8.652338105245387e-15
```

In [240]:

```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 1.573054383419466e-28
```

In [241]:

```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root Mean Squared Error: 1.2542146480644635e-14
```

DataSet MobilePrices

In [242]:

```
a=pd.read_csv(r"E:\Python Data Science\Documents\12_mobile_prices_2023 - 12_mobile_prices
a
```

Out[242]:

	Phone Name	Rating ?/5	Number of Ratings	RAM	ROM/Storage	Back/Rare Camera	Front Camera	Battery	Processo
0	POCO C50 (Royal Blue, 32 GB)	4.2	33,561	2 GB RAM	32 GB ROM	8MP Dual Camera	5MP Front Camera	5000 mAh	Mediatel Helio A22 Processor Upto 2.(GHz Pro..
1	POCO M4 5G (Cool Blue, 64 GB)	4.2	77,128	4 GB RAM	64 GB ROM	50MP + 2MP	8MP Front Camera	5000 mAh	Mediatel Dimensity 70(Processo
2	POCO C51 (Royal Blue, 64 GB)	4.3	15,175	4 GB RAM	64 GB ROM	8MP Dual Rear Camera	5MP Front Camera	5000 mAh	Helio G3(Processo
3	POCO C55 (Cool Blue, 64 GB)	4.2	22,621	4 GB RAM	64 GB ROM	50MP Dual Rear Camera	5MP Front Camera	5000 mAh	Mediatel Helio G8 Processo
4	POCO C51 (Power Black, 64 GB)	4.3	15,175	4 GB RAM	64 GB ROM	8MP Dual Rear Camera	5MP Front Camera	5000 mAh	Helio G3(Processo
...	...	...	...	...	...	...	...	...	..
1831	Infinix Note 7 (Forest Green, 64 GB)	4.3	25,582	4 GB RAM	64 GB ROM	48MP + 2MP + 2MP + AI Lens Camera	16MP Front Camera	5000 mAh	MediaTel Helio G7(Processo
1832	Infinix Note 7 (Bolivia Blue, 64 GB)	4.3	25,582	4 GB RAM	64 GB ROM	48MP + 2MP + 2MP + AI Lens Camera	16MP Front Camera	5000 mAh	MediaTel Helio G7(Processo
1833	Infinix Note 7 (Aether Black, 64 GB)	4.3	25,582	4 GB RAM	64 GB ROM	48MP + 2MP + 2MP + AI Lens Camera	16MP Front Camera	5000 mAh	MediaTel Helio G7(Processo
1834	Infinix Zero 8i (Silver Diamond, 128 GB)	4.2	7,117	8 GB RAM	128 GB ROM	48MP + 8MP + 2MP + AI Lens Camera	16MP + 8MP Dual Front Camera	4500 mAh	MediaTel Helic G90T Processo
1835	Infinix S5 (Quetzal Cyan, 64 GB)	4.3	15,701	4 GB RAM	64 GB ROM	16MP + 5MP + 2MP + Low Light Sensor	32MP Front Camera	4000 mAh	Helio P22 (MTK6762 Processo

1836 rows × 11 columns

In [243]:

```
a.head(5)
```

Out[243]:

	Phone Name	Rating ?/5	Number of Ratings	RAM	ROM/Storage	Back/Rare Camera	Front Camera	Battery	Processor	P in
0	POCO C50 (Royal Blue, 32 GB)	4.2	33,561	2 GB RAM	32 GB ROM	8MP Dual Camera	5MP Front Camera	5000 mAh	Mediatek Helio A22 Processor, Upto 2.0 GHz Pro...	₹5
1	POCO M4 5G (Cool Blue, 64 GB)	4.2	77,128	4 GB RAM	64 GB ROM	50MP + 2MP	8MP Front Camera	5000 mAh	Mediatek Dimensity 700 Processor	₹11
2	POCO C51 (Royal Blue, 64 GB)	4.3	15,175	4 GB RAM	64 GB ROM	8MP Dual Rear Camera	5MP Front Camera	5000 mAh	Helio G36 Processor	₹6
3	POCO C55 (Cool Blue, 64 GB)	4.2	22,621	4 GB RAM	64 GB ROM	50MP Dual Rear Camera	5MP Front Camera	5000 mAh	Mediatek Helio G85 Processor	₹7
4	POCO C51 (Power Black, 64 GB)	4.3	15,175	4 GB RAM	64 GB ROM	8MP Dual Rear Camera	5MP Front Camera	5000 mAh	Helio G36 Processor	₹6

In [244]:

```
a.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1836 entries, 0 to 1835
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Phone Name         1836 non-null   object 
 1   Rating ?/5         1836 non-null   float64
 2   Number of Ratings  1836 non-null   object 
 3   RAM                1836 non-null   object 
 4   ROM/Storage        1662 non-null   object 
 5   Back/Rare Camera   1827 non-null   object 
 6   Front Camera       1435 non-null   object 
 7   Battery            1826 non-null   object 
 8   Processor          1781 non-null   object 
 9   Price in INR       1836 non-null   object 
 10  Date of Scraping   1836 non-null   object 
dtypes: float64(1), object(10)
memory usage: 157.9+ KB
```

In [245]:

```
a.describe()
```

Out[245]:

	Rating ?/5
count	1836.000000
mean	4.210512
std	0.543912
min	0.000000
25%	4.200000
50%	4.300000
75%	4.400000
max	4.800000

In [246]:

```
a.isna()
```

Out[246]:

	Phone Name	Rating ?/5	Number of Ratings	RAM	ROM/Storage	Back/Rare Camera	Front Camera	Battery	Processor
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
1831	False	False	False	False	False	False	False	False	False
1832	False	False	False	False	False	False	False	False	False
1833	False	False	False	False	False	False	False	False	False
1834	False	False	False	False	False	False	False	False	False
1835	False	False	False	False	False	False	False	False	False

1836 rows × 11 columns

In [247]:

```
a.dropna()
```

Out[247]:

	Phone Name	Rating ?/5	Number of Ratings	RAM	ROM/Storage	Back/Rare Camera	Front Camera	Battery	Processo
0	POCO C50 (Royal Blue, 32 GB)	4.2	33,561	2 GB RAM	32 GB ROM	8MP Dual Camera	5MP Front Camera	5000 mAh	Mediatel Helio A22 Processor Upto 2.(GHz Pro..
1	POCO M4 5G (Cool Blue, 64 GB)	4.2	77,128	4 GB RAM	64 GB ROM	50MP + 2MP	8MP Front Camera	5000 mAh	Mediatel Dimensity 70(Processo
2	POCO C51 (Royal Blue, 64 GB)	4.3	15,175	4 GB RAM	64 GB ROM	8MP Dual Rear Camera	5MP Front Camera	5000 mAh	Helio G3(Processo
3	POCO C55 (Cool Blue, 64 GB)	4.2	22,621	4 GB RAM	64 GB ROM	50MP Dual Rear Camera	5MP Front Camera	5000 mAh	Mediatel Helio G8! Processo
4	POCO C51 (Power Black, 64 GB)	4.3	15,175	4 GB RAM	64 GB ROM	8MP Dual Rear Camera	5MP Front Camera	5000 mAh	Helio G3(Processo
...	...	...	...	...	...	...	...	...	..
1831	Infinix Note 7 (Forest Green, 64 GB)	4.3	25,582	4 GB RAM	64 GB ROM	48MP + 2MP + 2MP + AI Lens Camera	16MP Front Camera	5000 mAh	MediaTel Helio G7(Processo
1832	Infinix Note 7 (Bolivia Blue, 64 GB)	4.3	25,582	4 GB RAM	64 GB ROM	48MP + 2MP + 2MP + AI Lens Camera	16MP Front Camera	5000 mAh	MediaTel Helio G7(Processo
1833	Infinix Note 7 (Aether Black, 64 GB)	4.3	25,582	4 GB RAM	64 GB ROM	48MP + 2MP + 2MP + AI Lens Camera	16MP Front Camera	5000 mAh	MediaTel Helio G7(Processo
1834	Infinix Zero 8i (Silver Diamond, 128 GB)	4.2	7,117	8 GB RAM	128 GB ROM	48MP + 8MP + 2MP + AI Lens Camera	16MP + 8MP Dual Front Camera	4500 mAh	MediaTel Helic G90T Processo
1835	Infinix S5 (Quetzal Cyan, 64 GB)	4.3	15,701	4 GB RAM	64 GB ROM	16MP + 5MP + 2MP + Low Light Sensor	32MP Front Camera	4000 mAh	Helio P22 (MTK6762 Processo

1291 rows × 11 columns

In [248]:

```
a.columns
```

Out[248]:

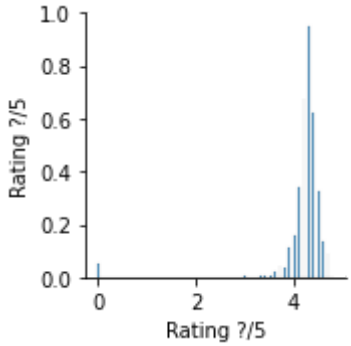
```
Index(['Phone Name', 'Rating ?/5', 'Number of Ratings', 'RAM', 'ROM/Storag
e',
       'Back/Rare Camera', 'Front Camera', 'Battery', 'Processor',
       'Price in INR', 'Date of Scraping'],
      dtype='object')
```

In [249]:

```
sns.pairplot(a)
```

Out[249]:

```
<seaborn.axisgrid.PairGrid at 0x1afdd3e85b0>
```



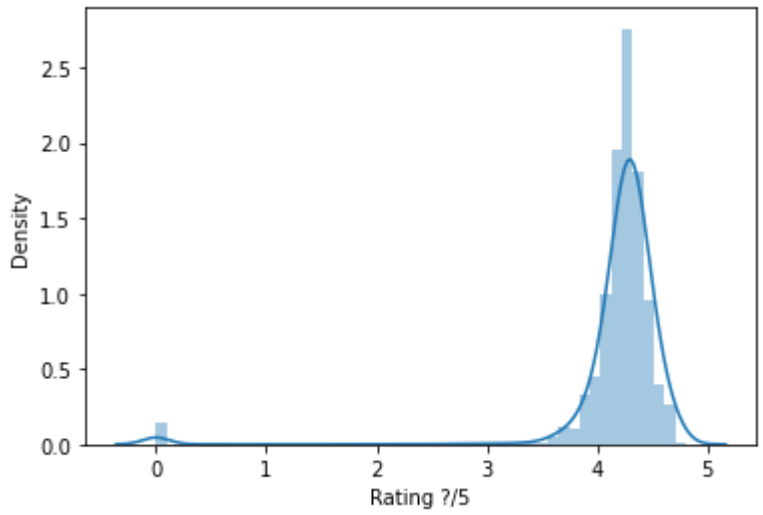
In [250]:

```
#normal distribution
sns.distplot(a['Rating ?/5'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[250]:

```
<AxesSubplot:xlabel='Rating ?/5', ylabel='Density'>
```



In [251]:

```
# Correlation map
sns.heatmap(a.corr())
```

Out[251]:

```
<AxesSubplot:>
```

In [252]:

```
x=a[['Rating ?/5']]
y=a['Rating ?/5']
```

In [253]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)


lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[253]:

```
LinearRegression()
```

In [254]:

```
print(lr.intercept_)
```

```
-8.881784197001252e-16
```

In [255]:

```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[255]:

	Co-efficient
Rating ?/5	1.0

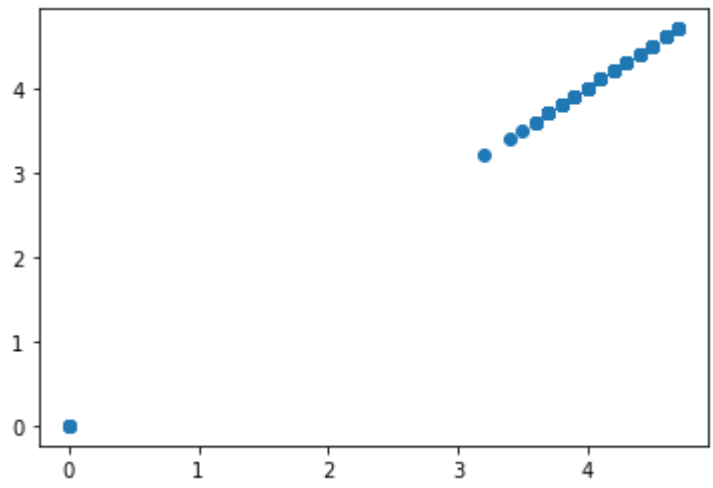
In [256]:

```
#Predicting

prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[256]:

```
<matplotlib.collections.PathCollection at 0x1afdd6a4700>
```



In [257]:

```
# Score

print(lr.score(x_test,y_test))
```

```
1.0
```

In [258]:

```
#Ridge fitting
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[258]:

```
Ridge(alpha=10)
```

In [259]:

```
#Ridge Score
rr.score(x_test,y_test)
```

Out[259]:

```
0.9995124203267691
```

In [260]:

```
# Lasso Fitting
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[260]:

```
Lasso(alpha=10)
```

In [261]:

```
# Lasso Score
la.score(x_test,y_test)
```

Out[261]:

```
-0.0085612035041025
```

In [262]:

```
# ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[262]:

```
ElasticNet()
```

In [263]:

```
print(en.coef_)
```

```
[0.]
```

In [264]:

```
print(en.intercept_)
```

```
4.198832684824903
```

In [265]:

```
print(en.predict(x_test))
```

```
[4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
```

```
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268
 4.19883268 4.19883268 4.19883268 4.19883268 4.19883268]
```

In [266]:

```
print(en.score(x_test,y_test))
```

```
-0.0085612035041025
```

In [267]:

```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 6.4477562228684225e-18
```

In [268]:

```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 5.726757932638924e-33
```

In [269]:

```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root Mean Squared Error: 7.567534560633947e-17
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```