In [1]:

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

In [2]:

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

In [166]:

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

DataSet Vehicle

In [4]:

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

Out[4]:

	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 [5]:

```
a.head()
```

Out[5]:

	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 [6]:

```
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 [7]:

```
a.describe()
```

Out[7]:

	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 [8]:

```
a.columns
```

Out[8]:

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

In [9]:

```
a.isna()
```

Out[9]:

	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 [10]:

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

Out[10]:

	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 [11]:

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

	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
1547	50.0	50	50.0	50.0	50.0	50.0	50.000000	find	

In [12]:

```
c.columns
```

Out[12]:

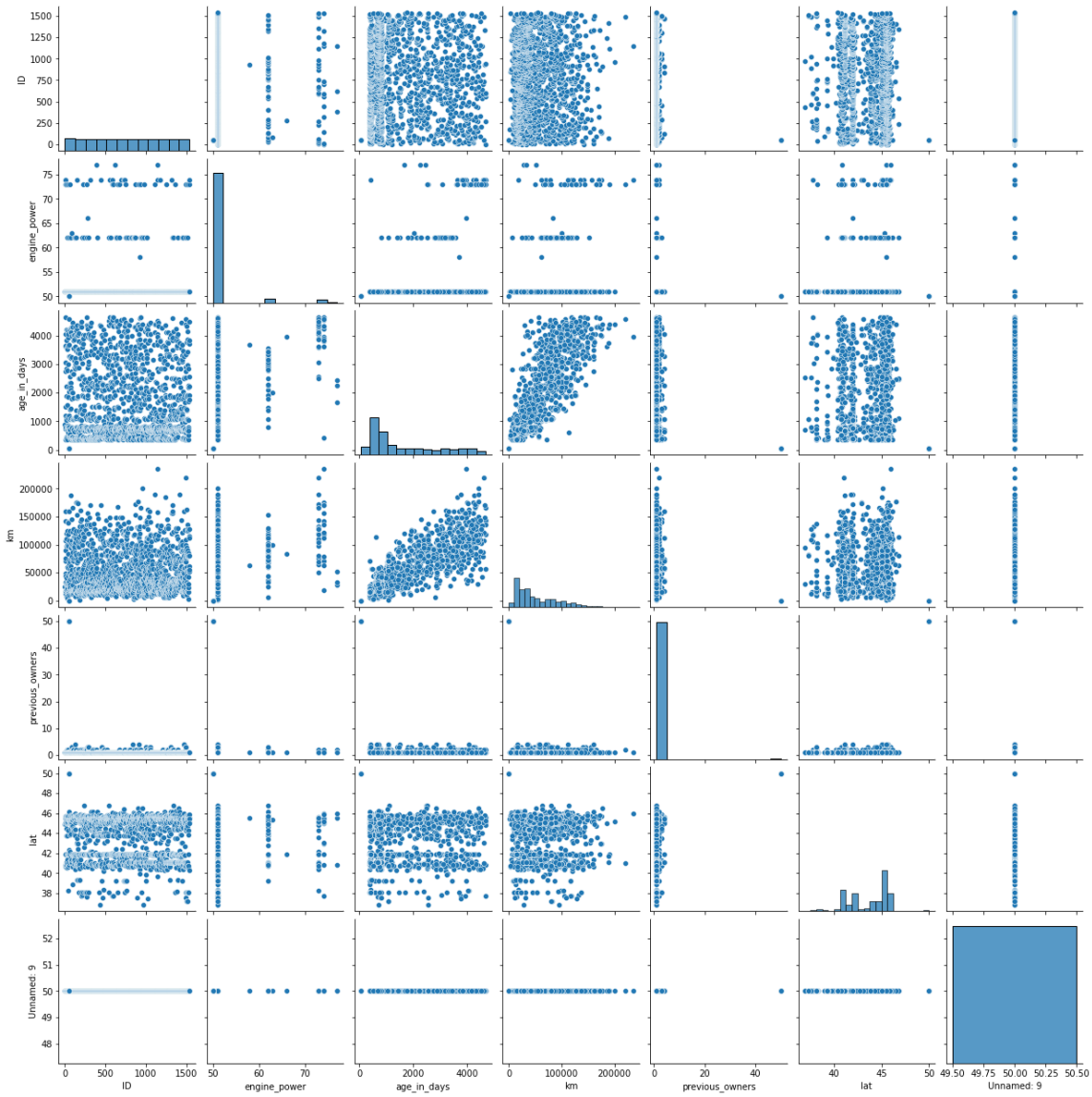
```
Index(['ID', 'model', 'engine_power', 'age_in_days', 'km', 'previous_owner
s',
       'lat', 'lon', 'price', 'Unnamed: 9', 'Unnamed: 10'],
      dtype='object')
```

In [13]:

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

Out[13]:

```
<seaborn.axisgrid.PairGrid at 0x248688ff760>
```



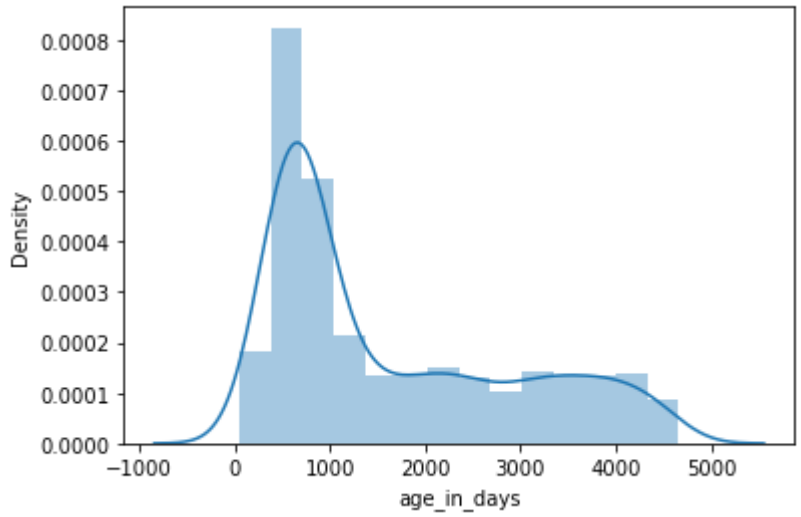
In [14]:

```
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[14]:

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



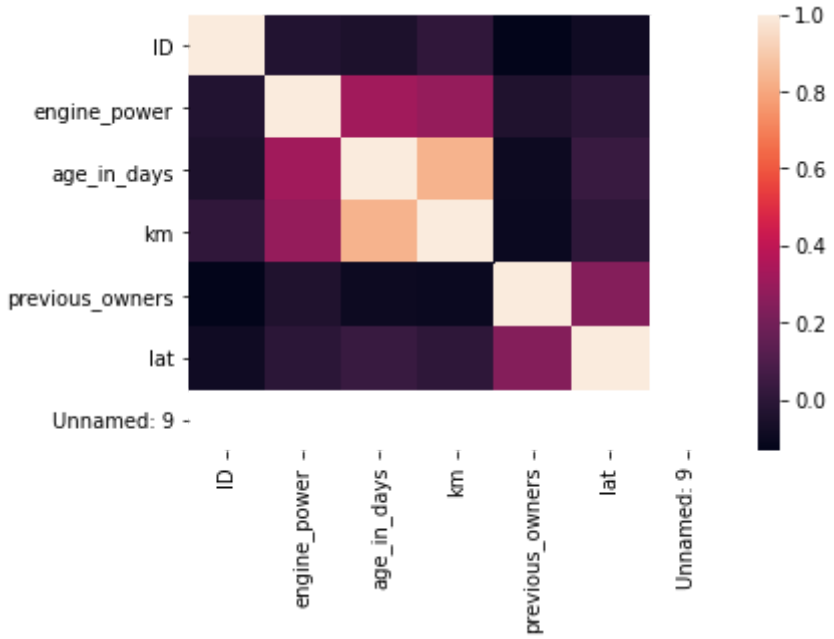
In [15]:

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

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

Out[15]:

```
<AxesSubplot:>
```



In [16]:

```
x=c[['engine_power', 'age_in_days']]
y=c['age_in_days']
```

In [17]:

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

In [18]:

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

Out[18]:

```
LinearRegression()
```

In [19]:

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

```
-2.9558577807620168e-12
```

In [20]:

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

Out[20]:

	Co-efficient
engine_power	7.586428e-14
age_in_days	1.000000e+00

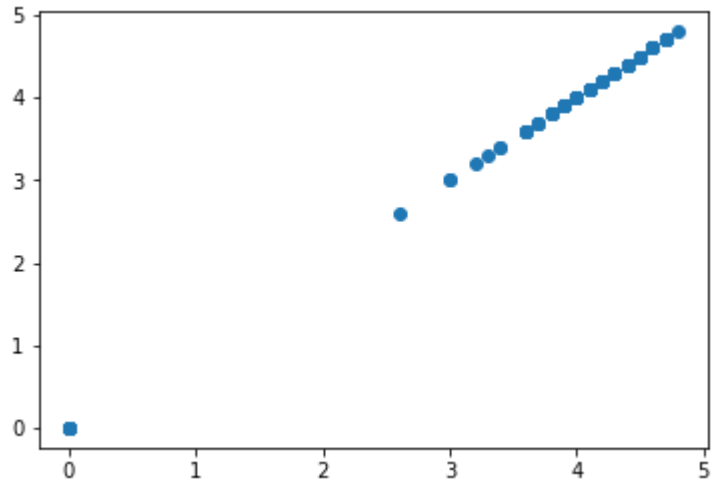
In [167]:

```
#Predicting

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

Out[167]:

```
<matplotlib.collections.PathCollection at 0x2485e422e50>
```



In [168]:

```
# Score

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

```
1.0
```

In [169]:

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

Out[169]:

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

In [170]:

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

Out[170]:

```
0.9992345048566785
```

In [171]:

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

Out[171]:

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

In [172]:

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

Out[172]:

```
-0.005181110800198008
```

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]:

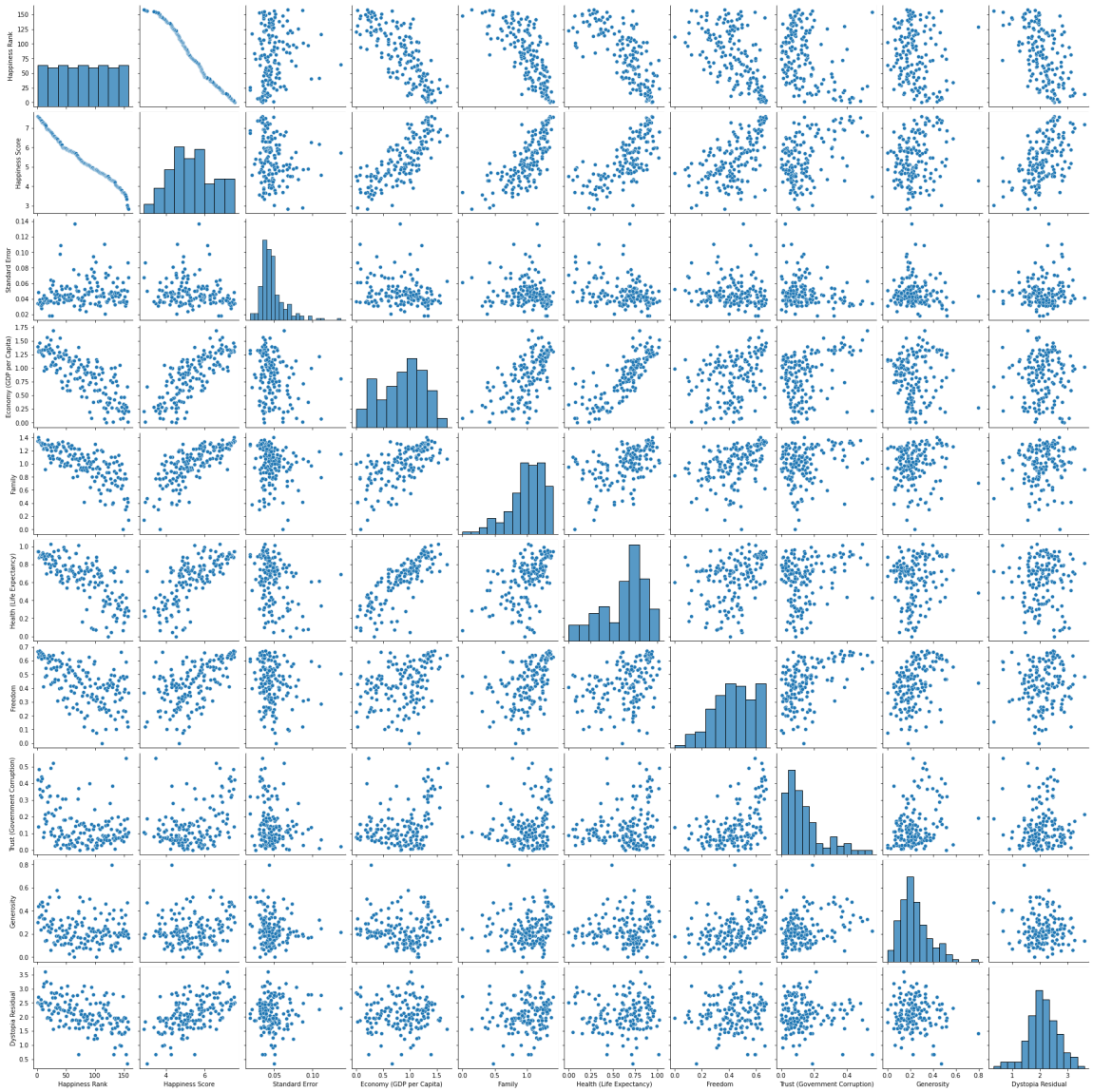
```
Index(['Country', 'Region', 'Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruptio
n)',
       'Generosity', 'Dystopia Residual'],
      dtype='object')
```

In [28]:

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

Out[28]:

```
<seaborn.axisgrid.PairGrid at 0x2486b5b2be0>
```



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]:

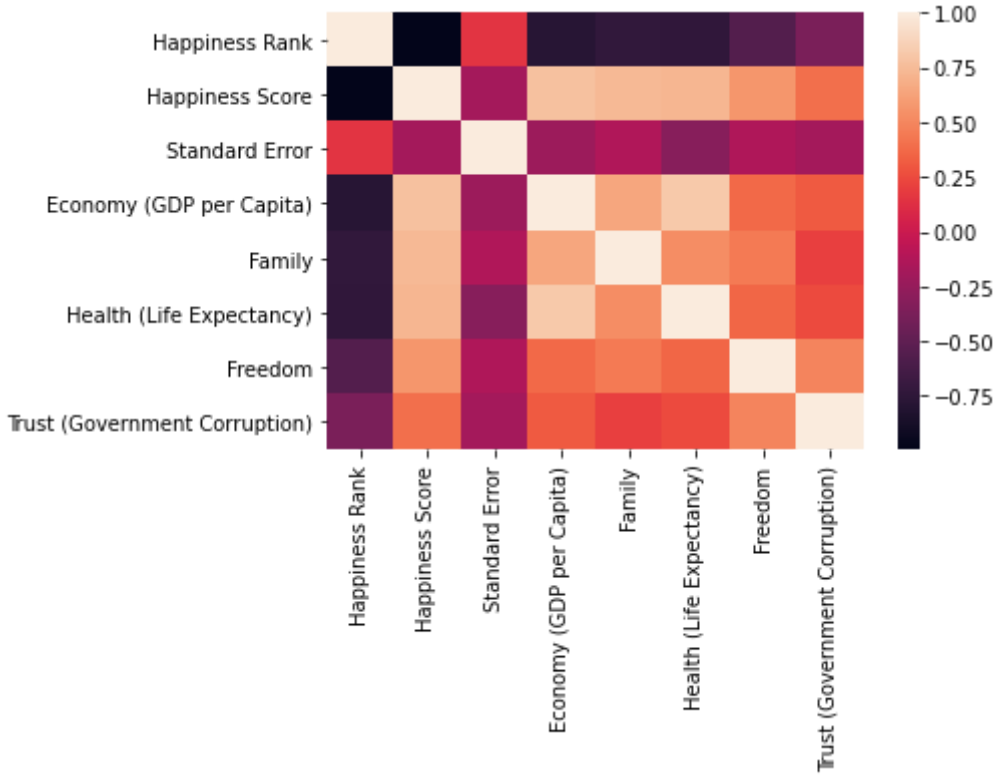
```
b=a[['Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruption)']]
```

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_)
```

```
0.0
```

In [36]:

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

Out[36]:

	Co-efficient
Happiness Score	1.000000e+00
Standard Error	1.692126e-15
Economy (GDP per Capita)	1.530309e-16
Family	-7.136055e-17
Health (Life Expectancy)	1.654037e-16
Freedom	3.766471e-16
Trust (Government Corruption)	6.728238e-17

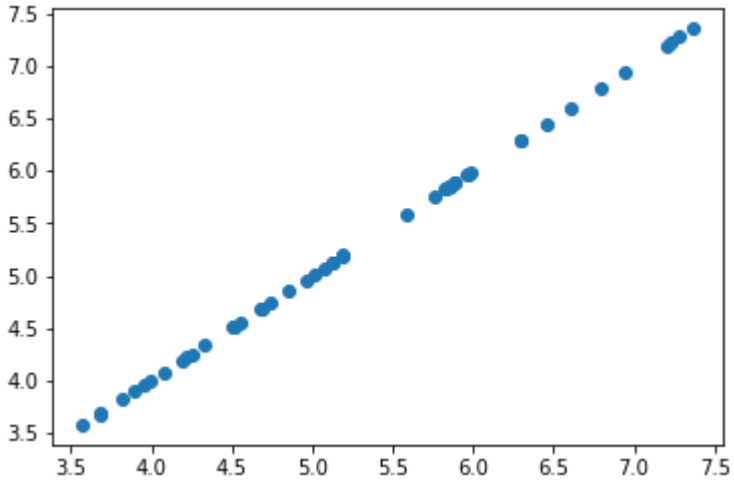
In [37]:

```
#Predicting

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

Out[37]:

```
<matplotlib.collections.PathCollection at 0x24870ac0e50>
```



In [38]:

```
# Score

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

```
1.0
```

In [173]:

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

Out[173]:

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

In [174]:

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

Out[174]:

```
0.9992345048566785
```

In [175]:

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

Out[175]:

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

In [176]:

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

Out[176]:

```
-0.005181110800198008
```

DataSet Fitness

In [39]:

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

Out[39]:

	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 [40]:

```
a.head()
```

Out[40]:

	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 [41]:

```
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 [42]:

```
a.describe()
```

Out[42]:

	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 [43]:

```
a.isna()
```

Out[43]:

	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 [44]:

```
a.columns
```

Out[44]:

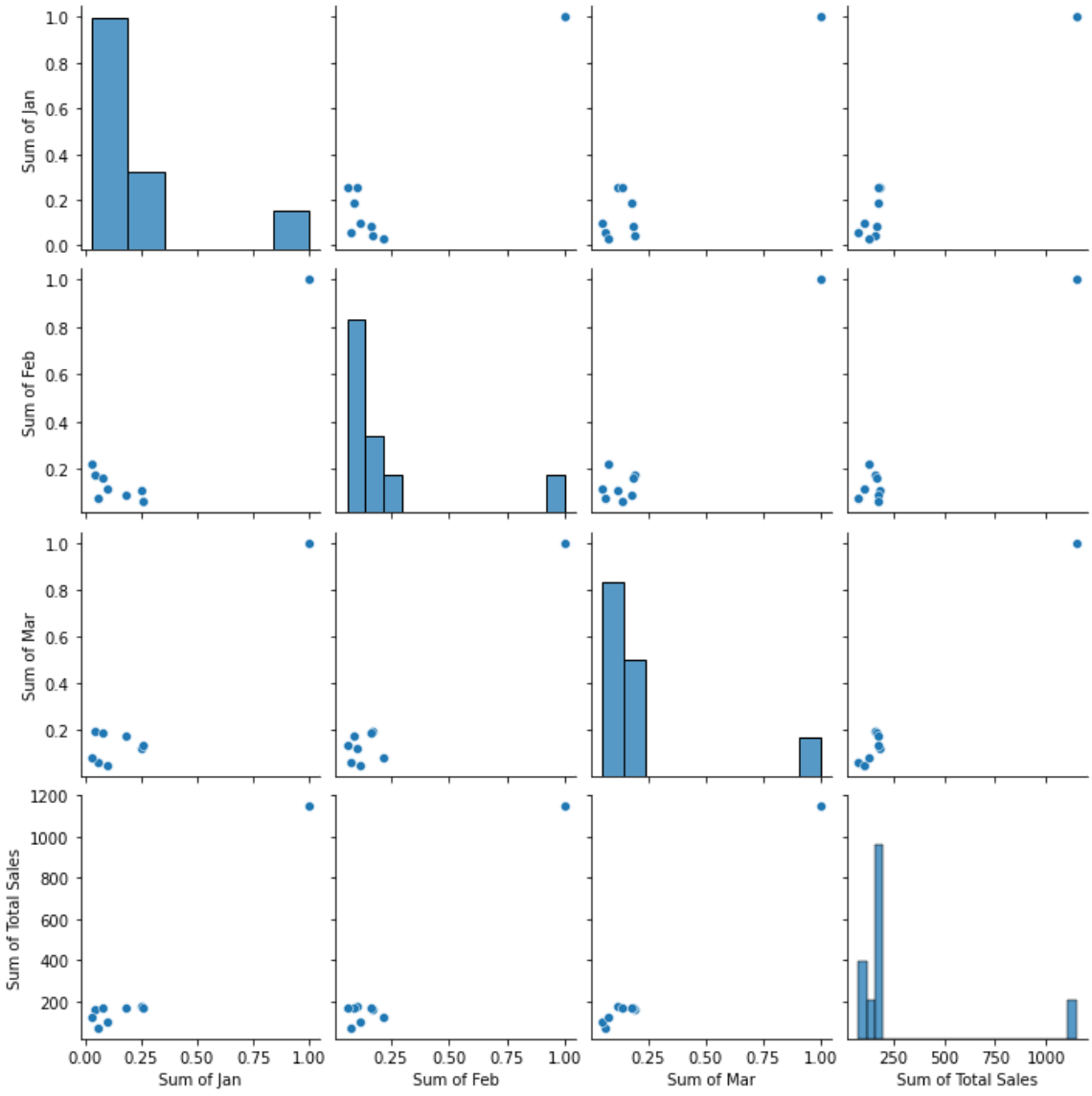
```
Index(['Row Labels', 'Sum of Jan', 'Sum of Feb', 'Sum of Mar',
       'Sum of Total Sales'],
      dtype='object')
```

In [45]:

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

Out[45]:

```
<seaborn.axisgrid.PairGrid at 0x24870af7be0>
```



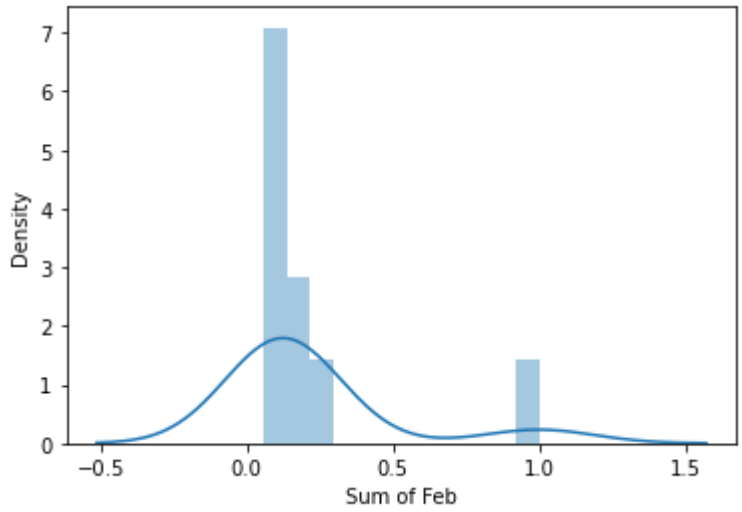
In [46]:

```
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[46]:

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



In [47]:

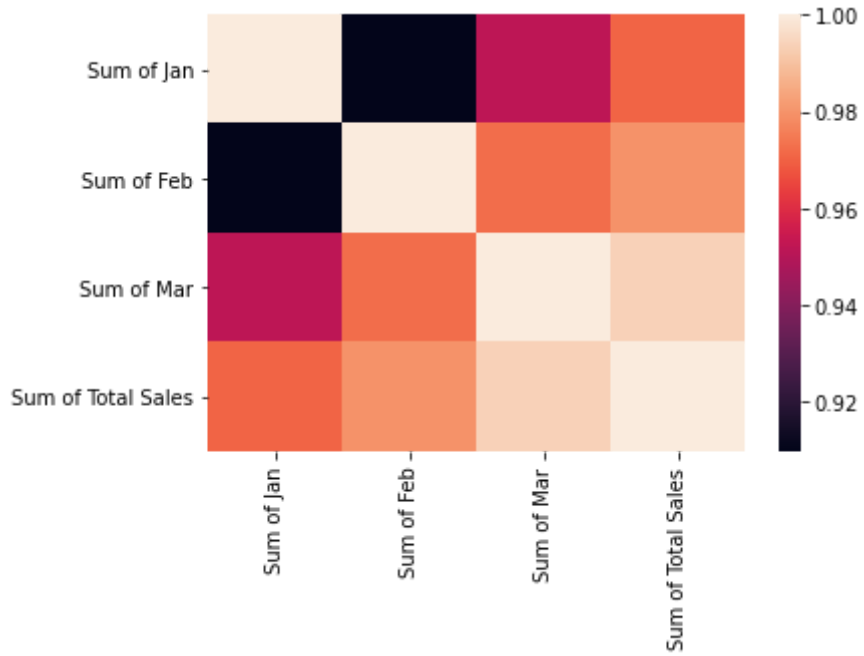
```
b=a[['Row Labels', 'Sum of Jan', 'Sum of Feb', 'Sum of Mar',
       'Sum of Total Sales']]
```

In [48]:

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

Out[48]:

```
<AxesSubplot:>
```



In [49]:

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

In [50]:

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

In [51]:

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

Out[51]:

```
LinearRegression()
```

In [52]:

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

```
-1.1102230246251565e-16
```

In [53]:

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

Out[53]:

	Co-efficient
Sum of Jan	-6.317181e-14
Sum of Feb	-6.885093e-14
Sum of Mar	1.000000e+00
Sum of Total Sales	1.779671e-16

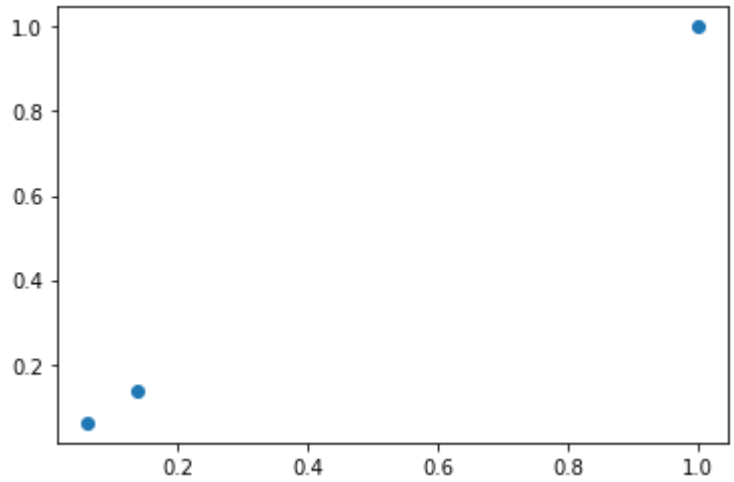
In [54]:

```
#Predicting

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

Out[54]:

```
<matplotlib.collections.PathCollection at 0x2487261e760>
```



In [55]:

```
# Score

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

```
1.0
```

In [177]:

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

Out[177]:

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

In [178]:

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

Out[178]:

```
0.9992345048566785
```

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]:

```
-0.005181110800198008
```

DataSet Drug

In [56]:

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

Out[56]:

	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 [57]:

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

Out[57]:

	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 [58]:

```
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 [59]:

```
a.describe()
```

Out[59]:

	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 [60]:

```
a.isna()
```

Out[60]:

	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 [61]:

```
a.columns
```

Out[61]:

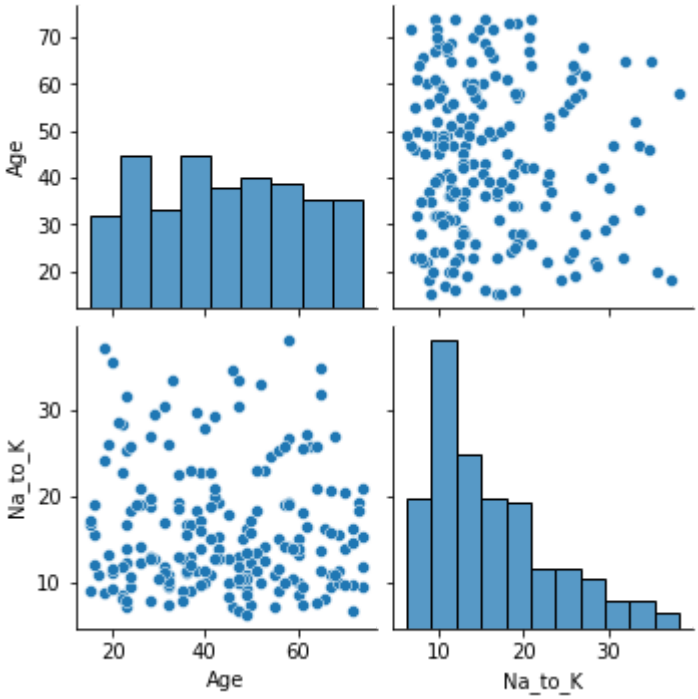
```
Index(['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K', 'Drug'], dtype='objec
t')
```

In [62]:

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

Out[62]:

```
<seaborn.axisgrid.PairGrid at 0x2487261e7c0>
```



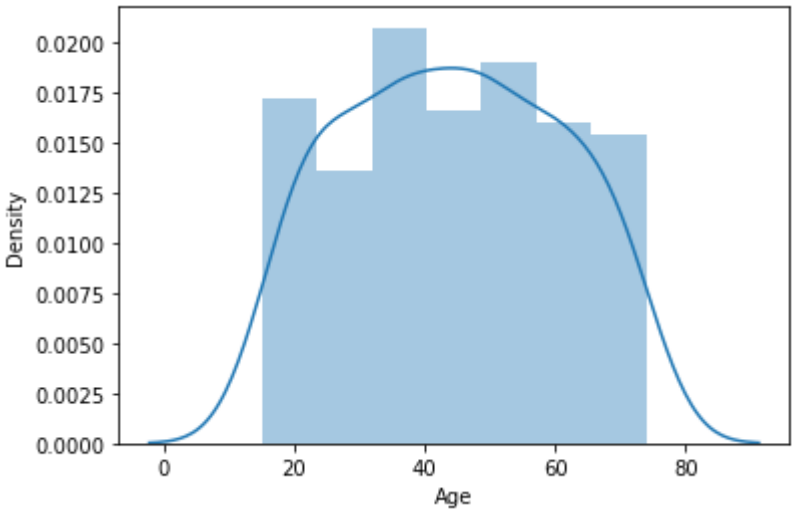
In [63]:

```
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[63]:

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

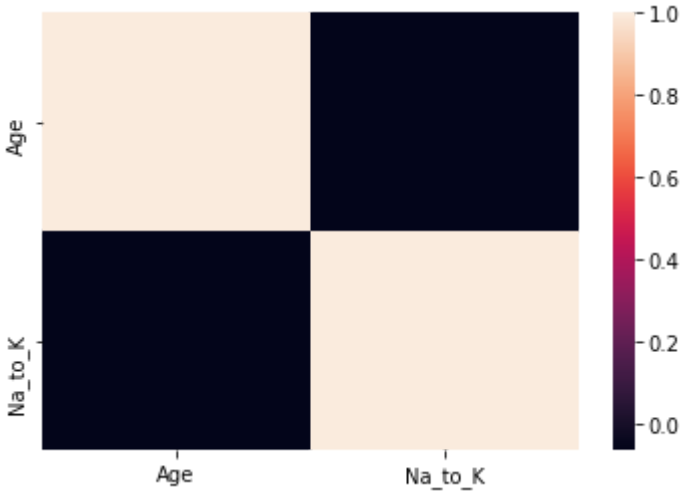


In [64]:

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

Out[64]:

```
<AxesSubplot:>
```



In [65]:

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

In [66]:

```
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[66]:

```
LinearRegression()
```

In [67]:

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

```
0.0
```

In [68]:

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

Out[68]:

	Co-efficient
Age	1.000000e+00
Na_to_K	8.124040e-17

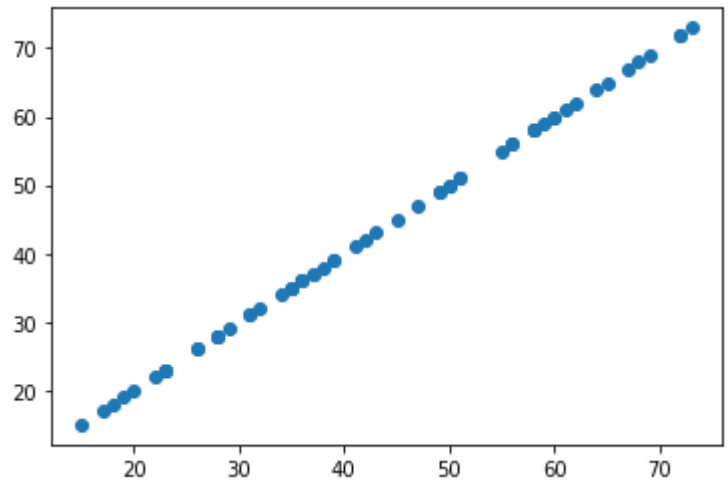
In [69]:

```
#Predicting

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

Out[69]:

```
<matplotlib.collections.PathCollection at 0x24872943b20>
```



In [70]:

```
# Score

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

```
1.0
```

In [181]:

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

Out[181]:

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

In [182]:

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

Out[182]:

```
0.9992345048566785
```

In [183]:

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

Out[183]:

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

In [184]:

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

Out[184]:

```
-0.005181110800198008
```

DataSet Instagram

In [71]:

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

Out[71]:

	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 [72]:

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

Out[72]:

	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 [73]:
```
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 [74]:

```
a.describe()
```

Out[74]:

	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 [75]:

```
a.isna()
```

Out[75]:

	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 [76]:

```
a.columns
```

Out[76]:

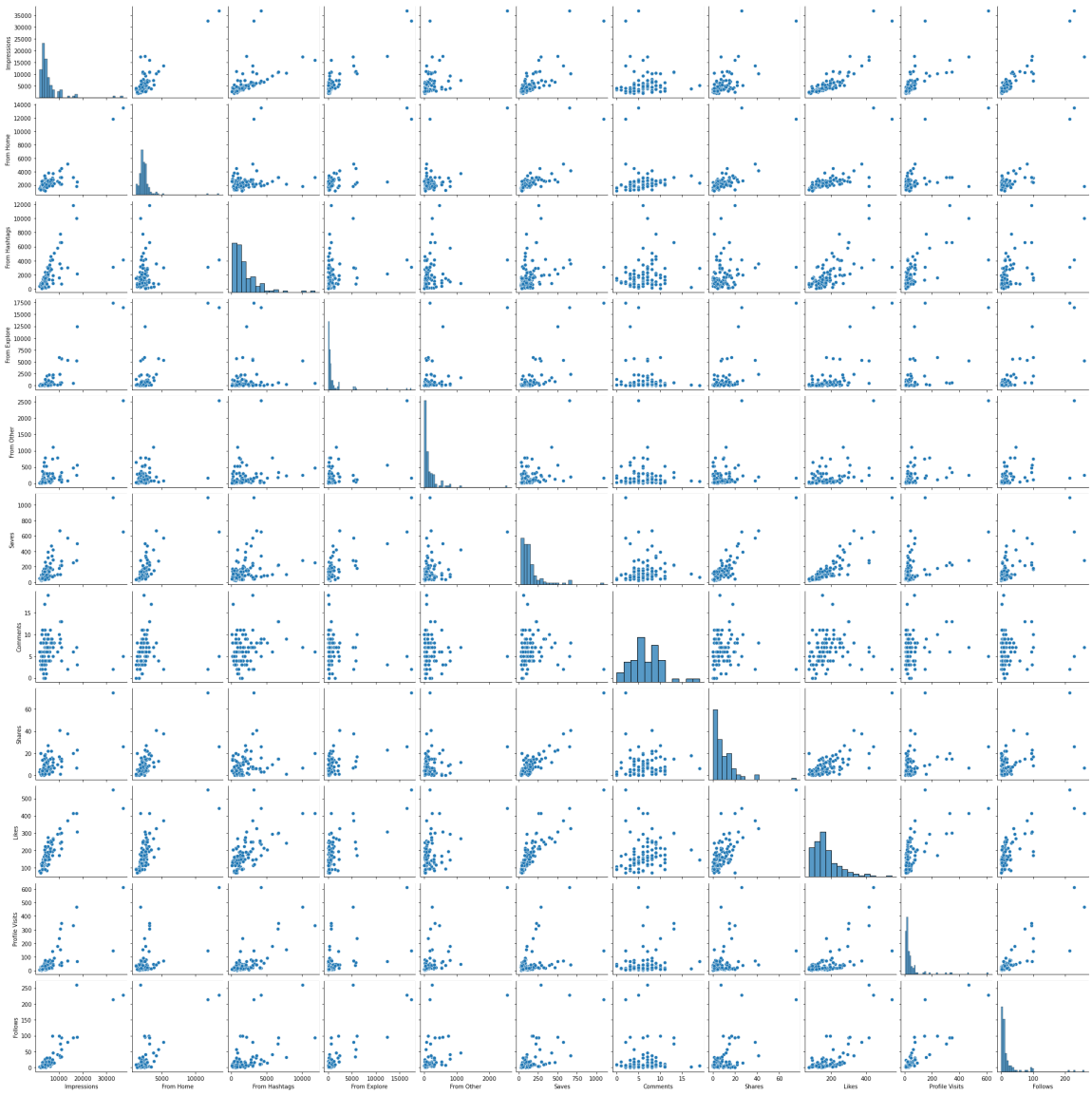
```
Index(['Impressions', 'From Home', 'From Hashtags', 'From Explore',
       'From Other', 'Saves', 'Comments', 'Shares', 'Likes', 'Profile Visi
ts',
       'Follows', 'Caption', 'Hashtags'],
      dtype='object')
```

In [77]:

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

Out[77]:

```
<seaborn.axisgrid.PairGrid at 0x24872943f10>
```



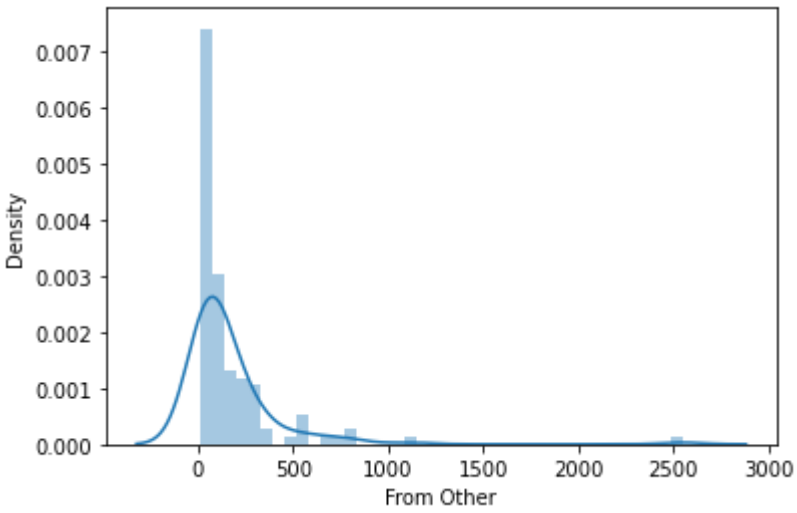
In [78]:

```
#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[78]:

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

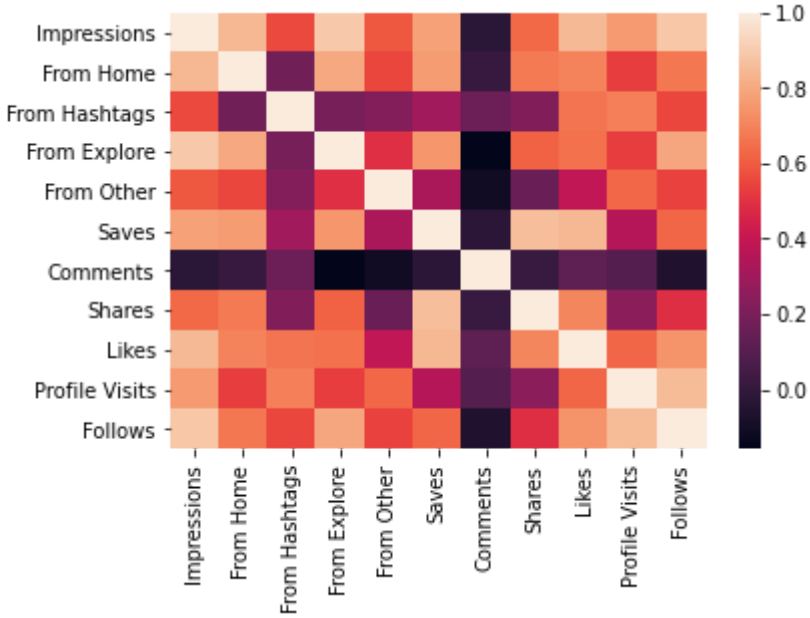


In [79]:

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

Out[79]:

```
<AxesSubplot:>
```



In [80]:

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

In [81]:

```
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[81]:

```
LinearRegression()
```

In [82]:

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

```
-5.002220859751105e-12
```

In [83]:

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

Out[83]:

	Co-efficient
Impressions	2.201944e-16
From Home	1.000000e+00
From Hashtags	-2.880966e-16
From Explore	-6.238995e-16
From Other	-4.251498e-16
Saves	2.382412e-16
Comments	-5.244320e-15
Shares	-1.790658e-15
Likes	2.006460e-15
Profile Visits	8.851035e-17
Follows	2.222233e-15

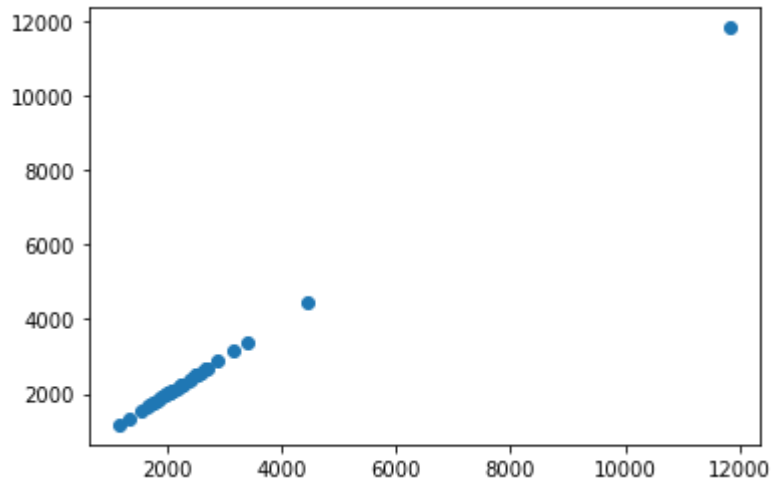
In [84]:

```
#Predicting

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

Out[84]:

```
<matplotlib.collections.PathCollection at 0x24878280280>
```



In [85]:

```
# Score

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

```
1.0
```

In [185]:

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

Out[185]:

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

In [186]:

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

Out[186]:

```
0.9992345048566785
```

In [187]:

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

Out[187]:

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

In [188]:

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

Out[188]:

```
-0.005181110800198008
```

DataSet SalesWorkLoad

In [86]:

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

Out[86]:

	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 [87]:

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

Out[87]:

	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 [88]:

```
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 [89]:

```
a.describe()
```

Out[89]:

	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 [90]:

```
a.isna()
```

Out[90]:

	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 [91]:

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

Out[91]:

	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 [92]:

```
c.columns
```

Out[92]:

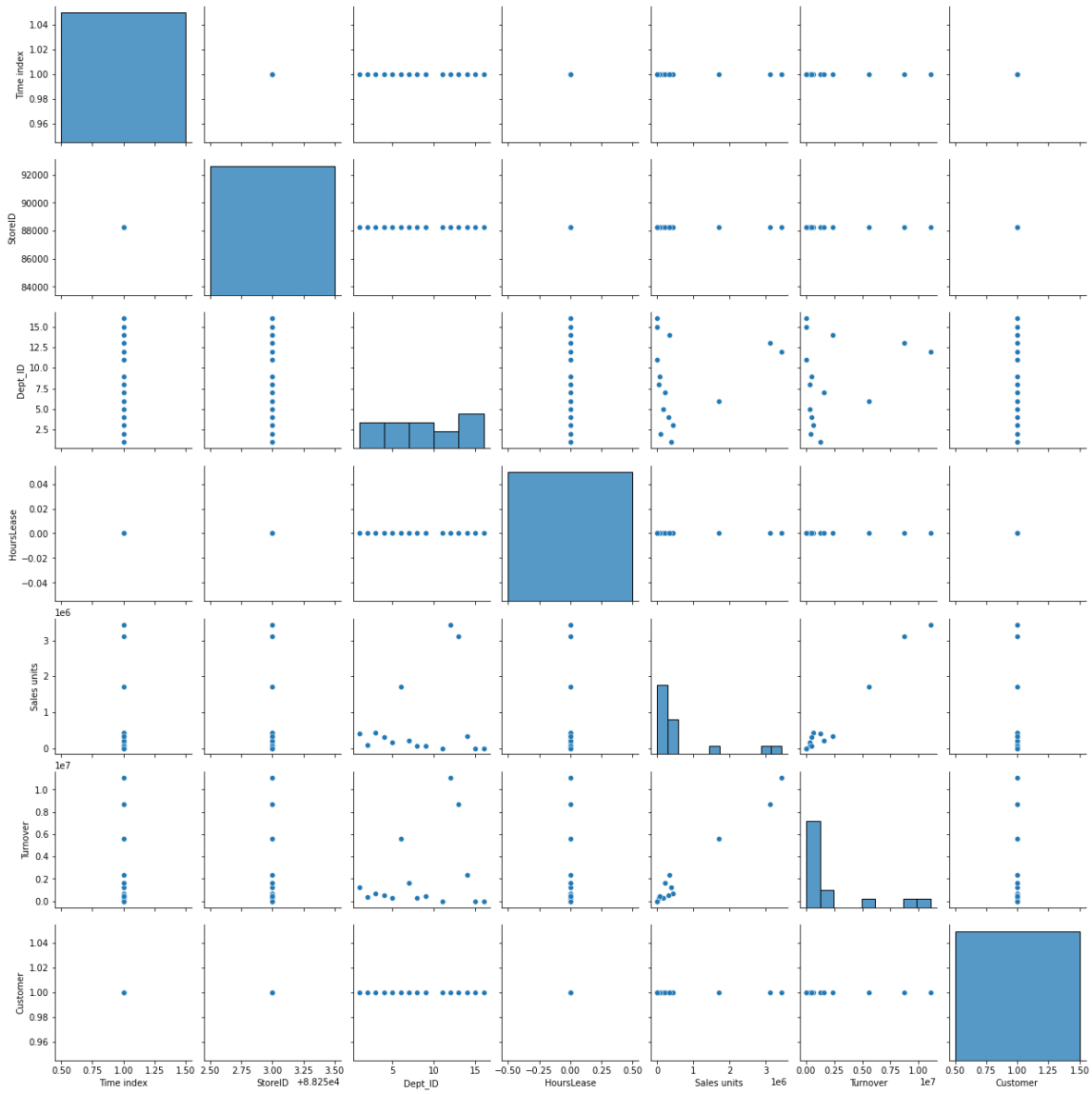
```
Index(['MonthYear', 'Time index', 'Country', 'StoreID', 'City', 'Dept_ID',
       'Dept. Name', 'HoursOwn', 'HoursLease', 'Sales units', 'Turnover',
       'Customer', 'Area (m2)', 'Opening hours'],
      dtype='object')
```

In [93]:

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

Out[93]:

```
<seaborn.axisgrid.PairGrid at 0x248782952e0>
```



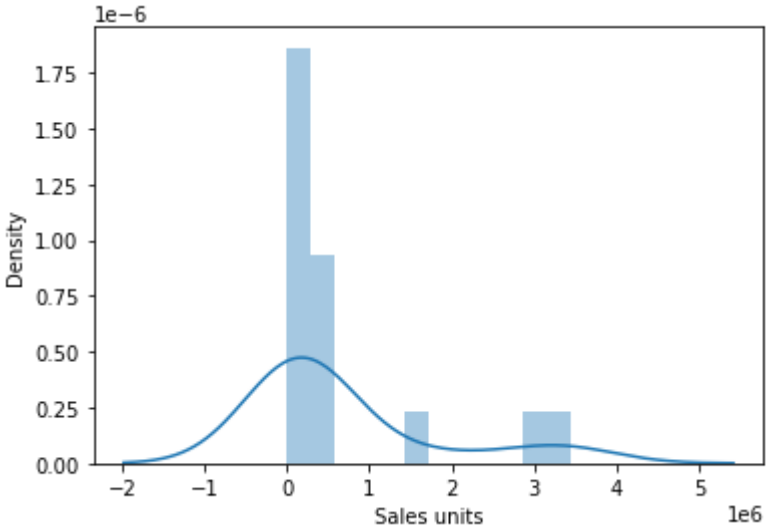
In [94]:

```
#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[94]:

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

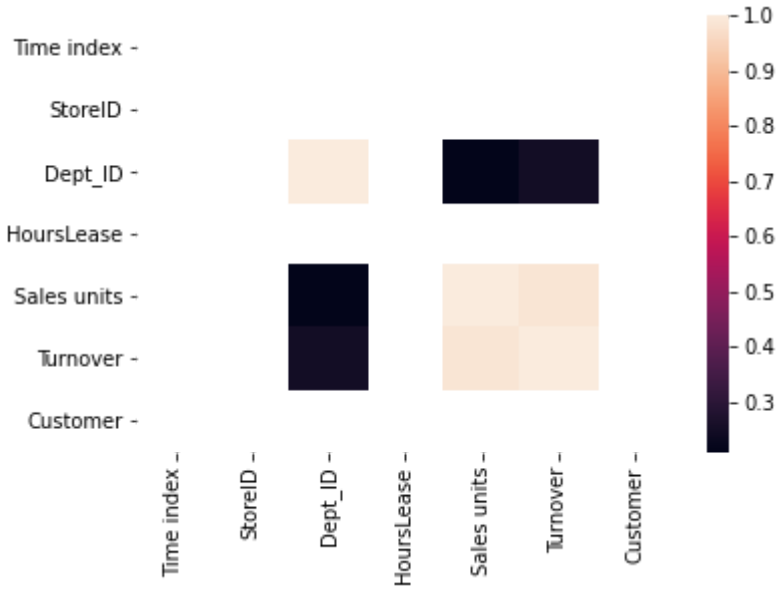


In [95]:

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

Out[95]:

```
<AxesSubplot:>
```



In [96]:

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

In [97]:

```
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[97]:

```
LinearRegression()
```

In [98]:

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

```
5.656359469696781e-08
```

In [99]:

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

Out[99]:

	Co-efficient
Time index	0.000000e+00
StoreID	-6.408073e-13
Dept_ID	1.000000e+00
HoursOwn	1.889643e-15
HoursLease	0.000000e+00
Sales units	-4.488756e-17
Turnover	8.856305e-18
Customer	0.000000e+00

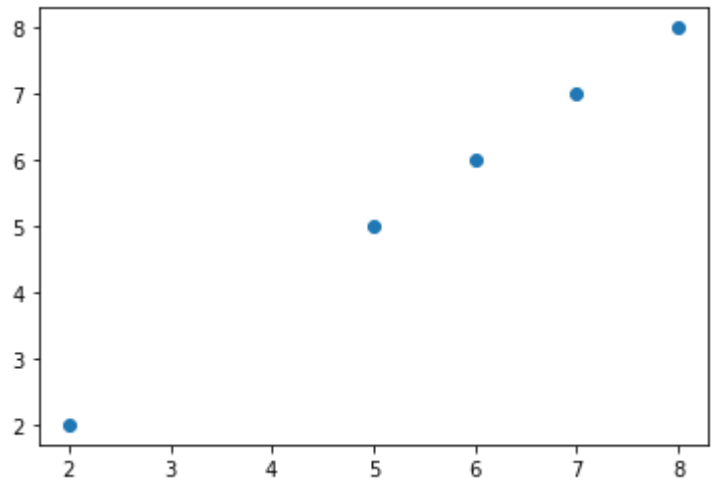
In [100]:

```
#Predicting

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

Out[100]:

```
<matplotlib.collections.PathCollection at 0x2487c061d30>
```



In [101]:

```
# Score

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

```
1.0
```

In [189]:

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

Out[189]:

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

In [190]:

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

Out[190]:

```
0.9992345048566785
```

In [191]:

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

Out[191]:

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

In [192]:

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

Out[192]:

```
-0.005181110800198008
```

DataSet Uber

In [102]:

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

Out[102]:

	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 [103]:

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

Out[103]:

	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 [104]:

```
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 [105]:

```
b.describe()
```

Out[105]:

	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 [106]:

```
b.isna()
```

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

In [107]:

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

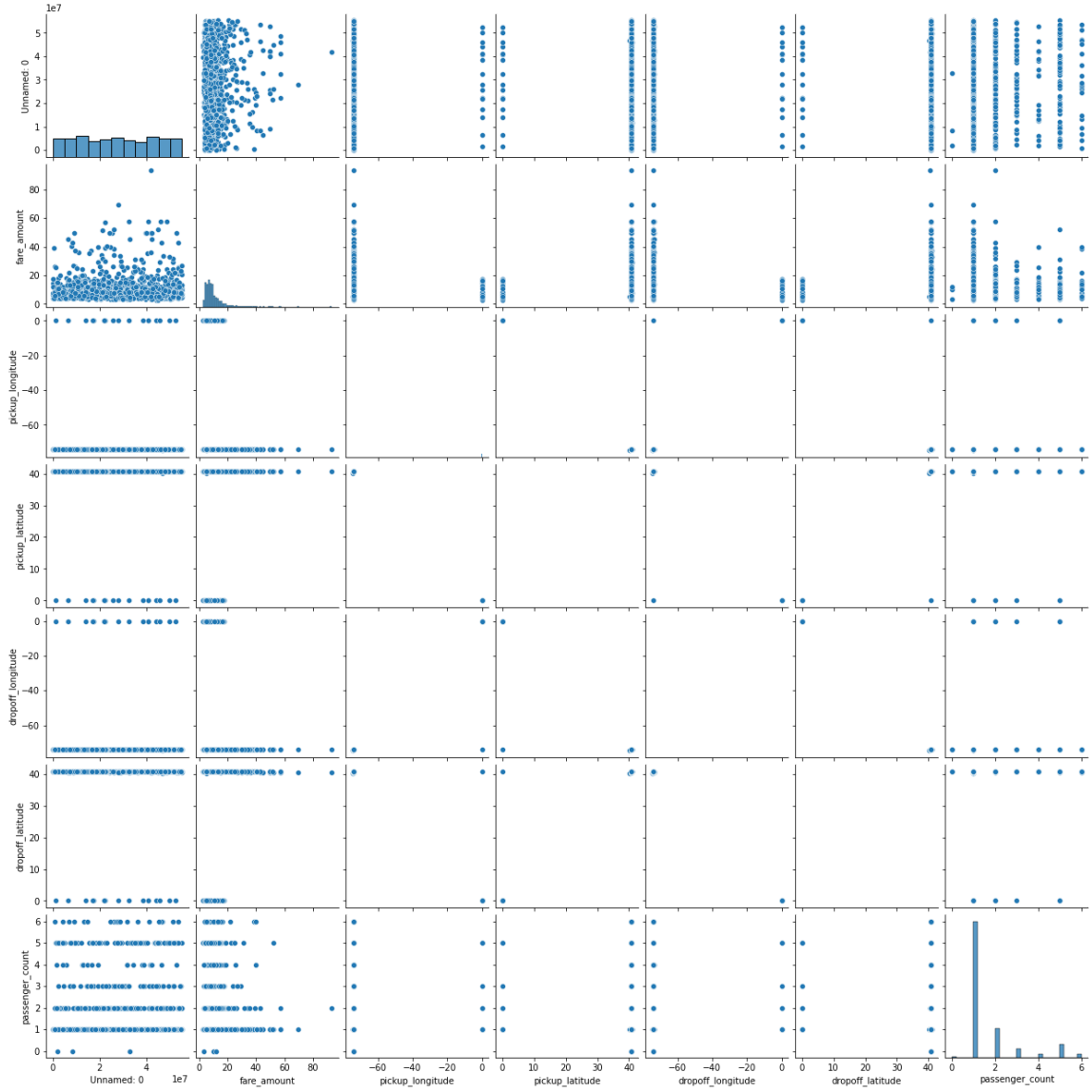
```
Index(['Unnamed: 0', 'key', 'fare_amount', 'pickup_datetime',
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'passenger_count'],
      dtype='object')
```

In [108]:

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

Out[108]:

```
<seaborn.axisgrid.PairGrid at 0x2487c0847c0>
```



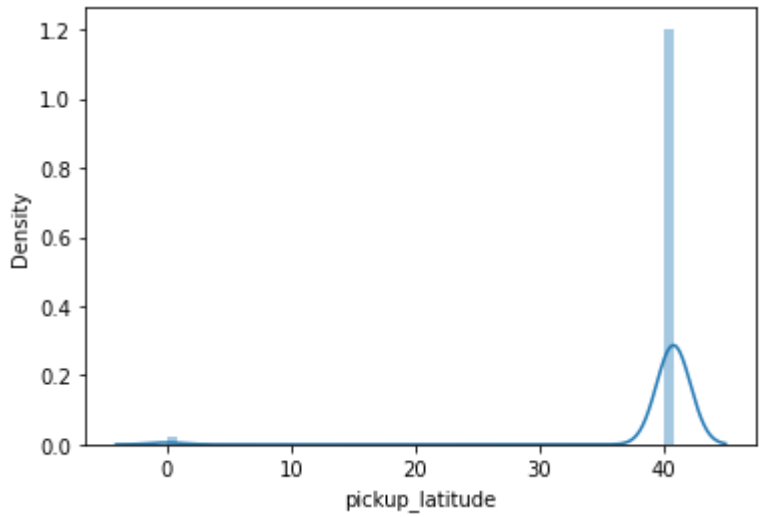
In [109]:

```
#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[109]:

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

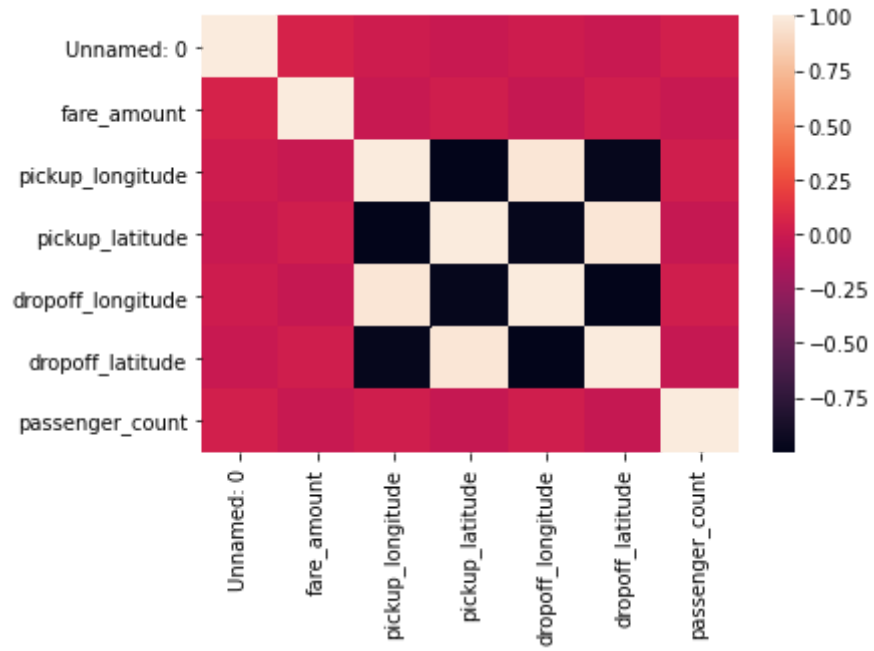


In [110]:

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

Out[110]:

```
<AxesSubplot:>
```



In [111]:

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

In [112]:

```
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[112]:

```
LinearRegression()
```

In [113]:

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

```
-2.717825964282383e-13
```

In [114]:

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

Out[114]:

	Co-efficient
Unnamed: 0	9.743538e-21
fare_amount	1.000000e+00
pickup_longitude	-6.194682e-16
pickup_latitude	-1.081274e-15
dropoff_longitude	1.081156e-15
dropoff_latitude	2.015830e-15
passenger_count	-7.250519e-17

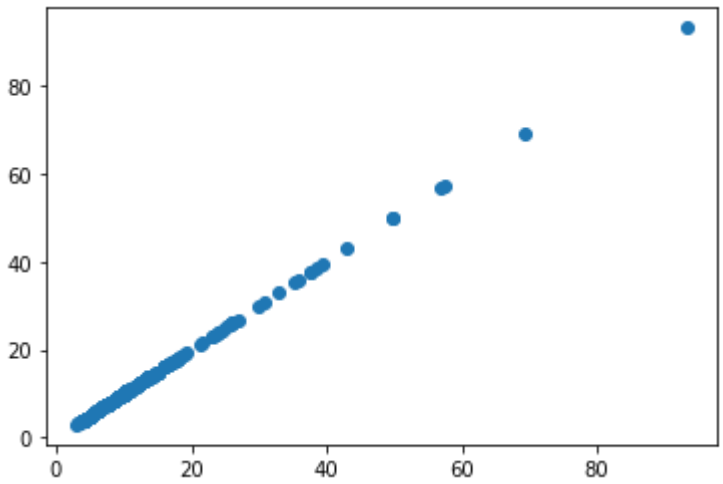
In [115]:

```
#Predicting

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

Out[115]:

```
<matplotlib.collections.PathCollection at 0x2481e06e550>
```



In [116]:

```
# Score

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

```
1.0
```

In [193]:

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

Out[193]:

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

In [194]:

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

Out[194]:

```
0.9992345048566785
```

In [195]:

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

Out[195]:

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

In [196]:

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

Out[196]:

```
-0.005181110800198008
```

DataSet Cancer

In [117]:

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

Out[117]:

	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 [118]:

```
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 [119]:

```
a.describe()
```

Out[119]:

	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 [120]:

```
a.isna()
```

Out[120]:

	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 [121]:

```
a.columns
```

Out[121]:

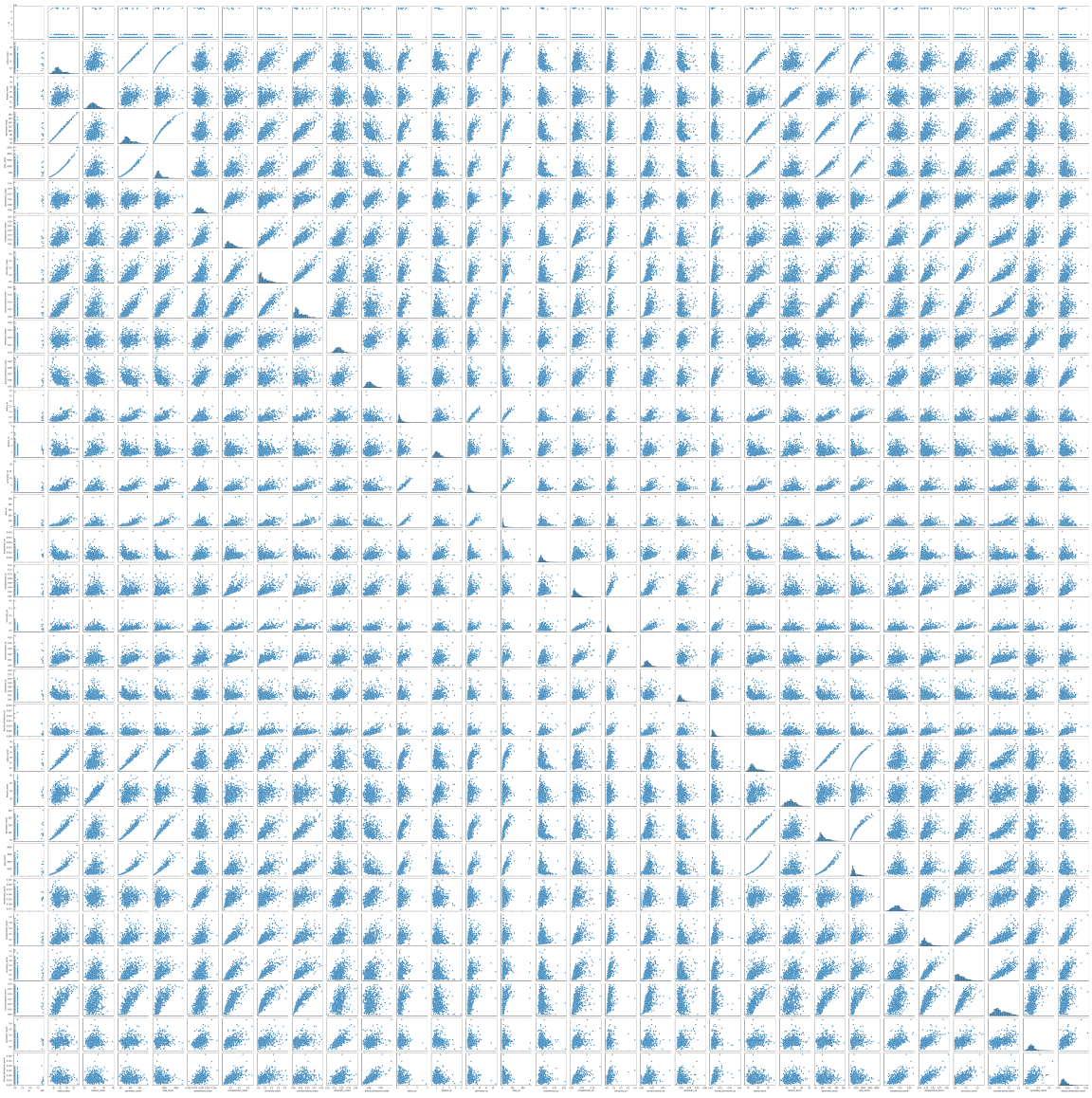
```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mea
n',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_s
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 [122]:

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

Out[122]:

```
<seaborn.axisgrid.PairGrid at 0x2481e07d5e0>
```



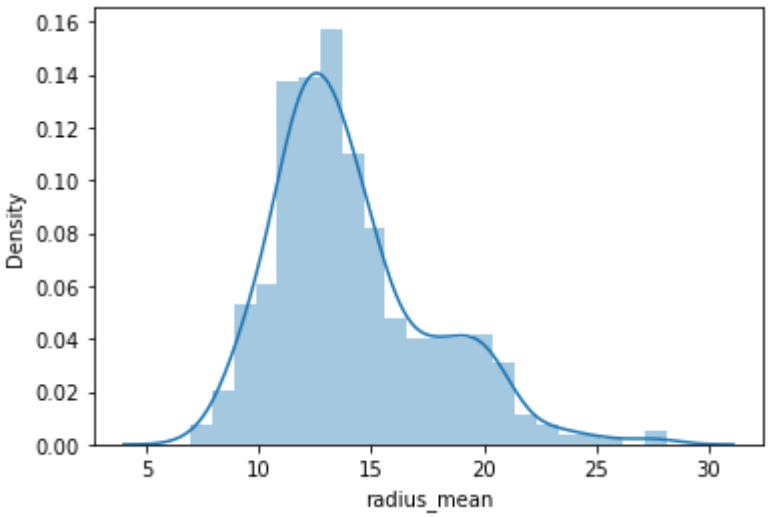
In [123]:

```
#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[123]:

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

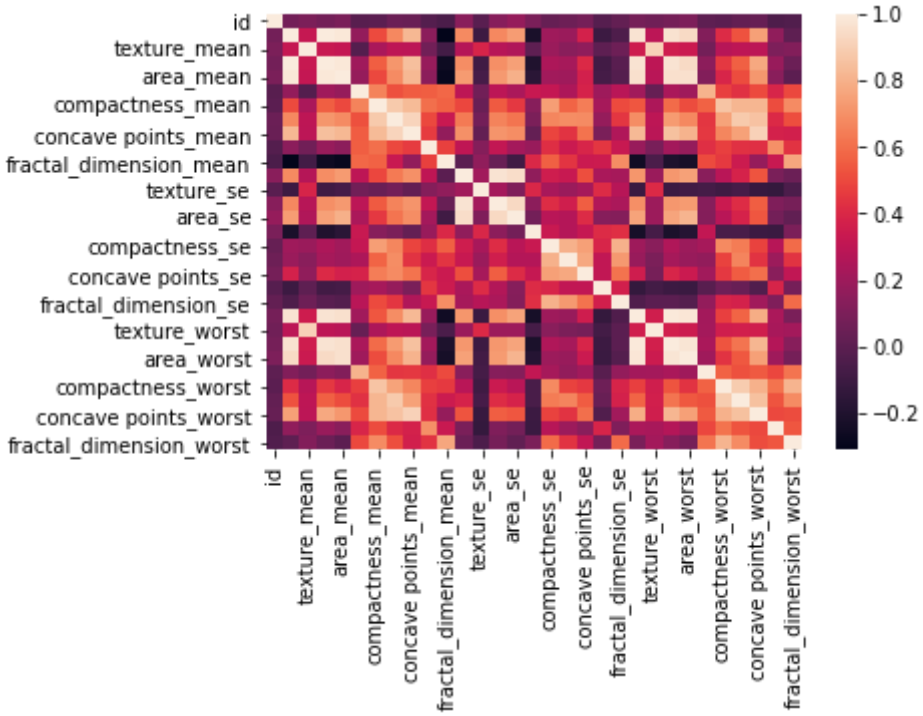


In [124]:

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

Out[124]:

```
<AxesSubplot:>
```



In [125]:

```
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 [126]:

```
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[126]:

```
LinearRegression()
```

In [127]:

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

```
-9.201528428093297e-13
```

In [128]:

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

Out[128]:

	Co-efficient
id	1.437536e-20
radius_mean	1.000000e+00
texture_mean	-1.778050e-16
perimeter_mean	9.391279e-16
area_mean	9.894769e-17
smoothness_mean	3.181452e-14
compactness_mean	1.168965e-14
concavity_mean	3.091441e-15
concave points_mean	-3.297952e-14
symmetry_mean	-8.503020e-15
fractal_dimension_mean	-1.040676e-13
radius_se	8.400254e-15
texture_se	-4.936879e-17
perimeter_se	2.568910e-16
area_se	1.038350e-16
smoothness_se	-1.833527e-14
compactness_se	-1.350511e-15
concavity_se	-6.279150e-15
concave points_se	1.914955e-14
symmetry_se	-9.457721e-16
fractal_dimension_se	5.198924e-14

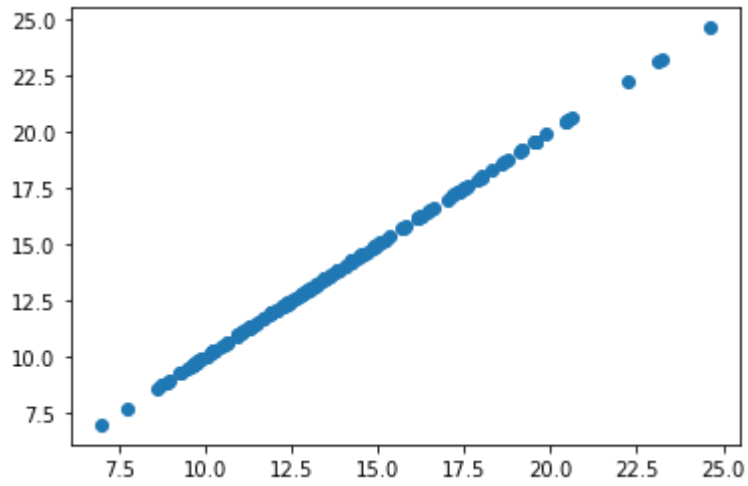
In [129]:

```
#Predicting

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

Out[129]:

```
<matplotlib.collections.PathCollection at 0x2484cc751f0>
```



In [130]:

```
# Score

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

```
1.0
```

In [197]:

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

Out[197]:

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

In [198]:

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

Out[198]:

```
0.9992345048566785
```

In [199]:

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

Out[199]:

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

In [200]:

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

Out[200]:

```
-0.005181110800198008
```

DataSet WineEquality

In [132]:

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

Out[132]:

	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 [133]:

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

Out[133]:

	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 [134]:

```
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 [135]:

```
a.describe()
```

Out[135]:

	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 [136]:

```
a.isna()
```

Out[136]:

	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 [137]:

```
a.columns
```

Out[137]:

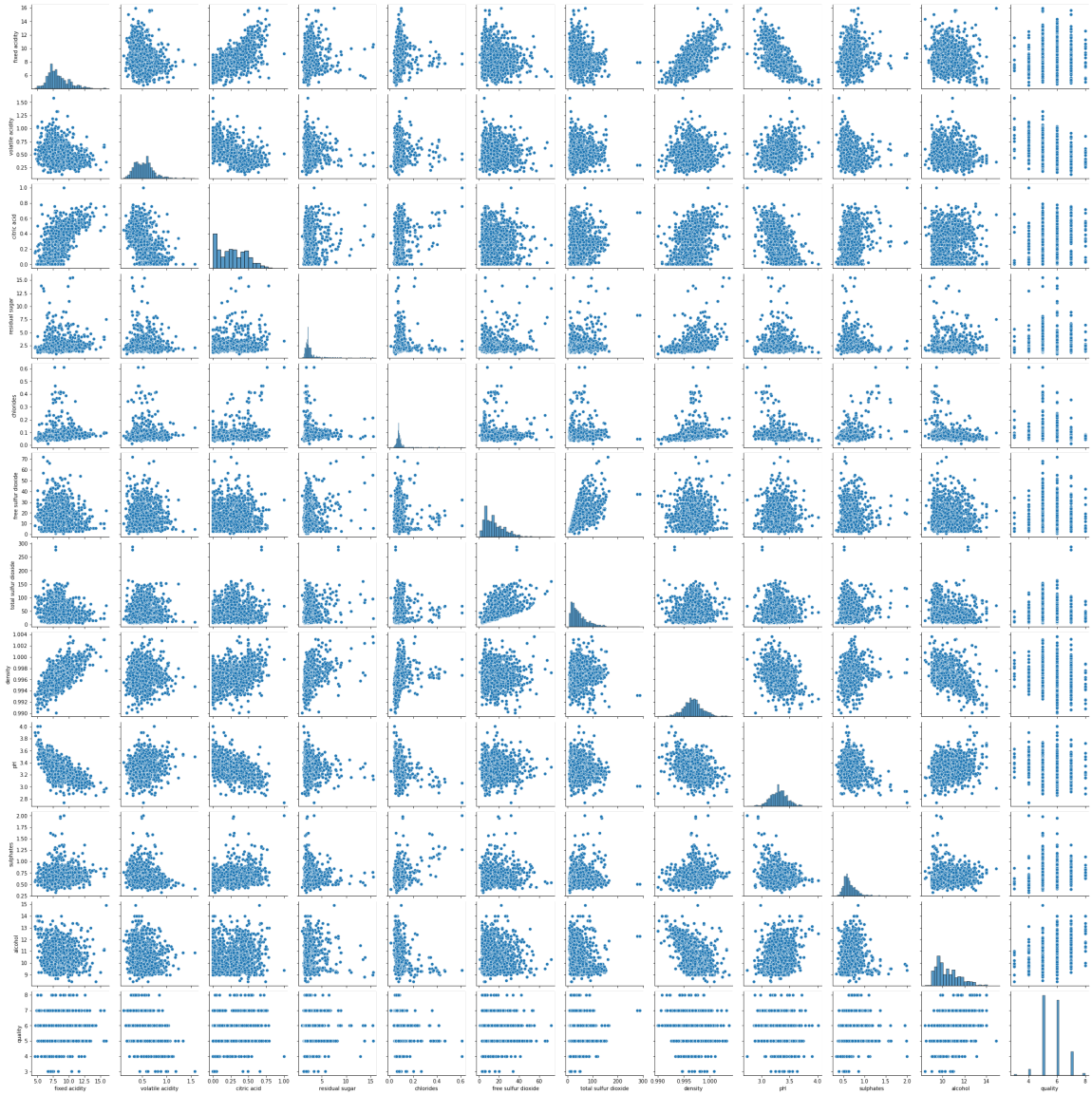
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual suga
r',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'densit
y',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [138]:

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

Out[138]:

```
<seaborn.axisgrid.PairGrid at 0x2484bb6cf70>
```



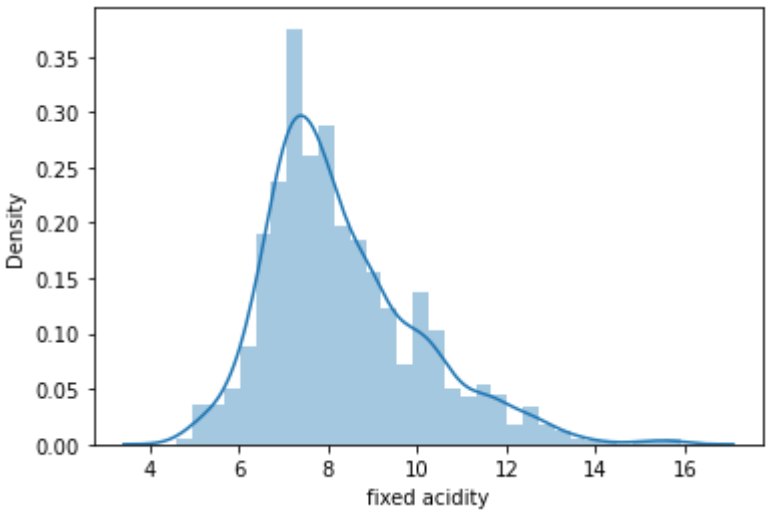
In [139]:

```
#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[139]:

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

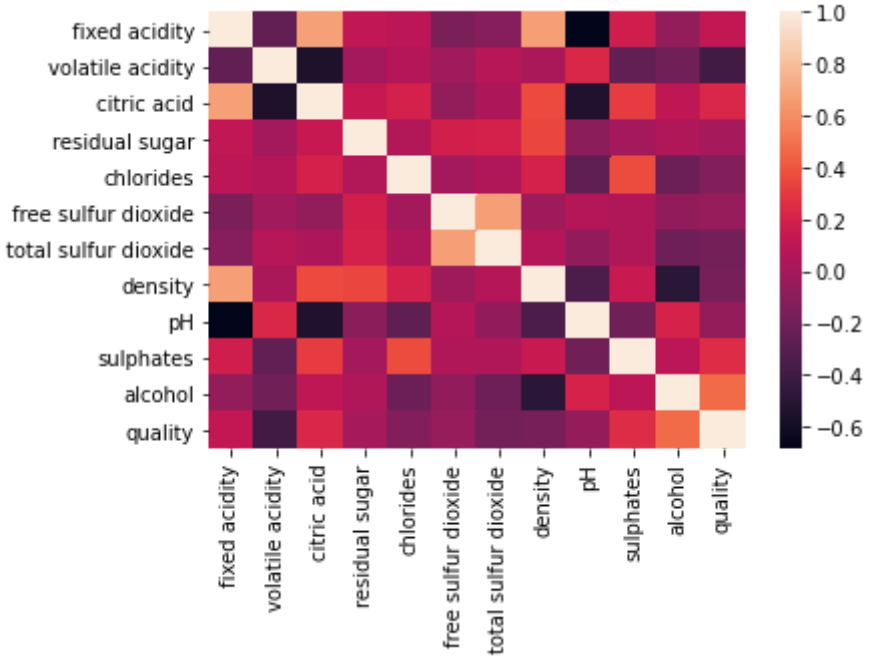


In [140]:

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

Out[140]:

```
<AxesSubplot:>
```



In [141]:

```
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 [142]:

```
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[142]:

```
LinearRegression()
```

In [143]:

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

```
-2.6645352591003757e-14
```

In [144]:

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

Out[144]:

	Co-efficient
fixed acidity	1.000000e+00
volatile acidity	-5.239349e-16
citric acid	1.714706e-16
residual sugar	1.549800e-16
chlorides	-4.348205e-16
free sulfur dioxide	2.146002e-15
total sulfur dioxide	-4.533420e-16
density	4.336154e-15
pH	-1.214305e-16
sulphates	-3.433291e-16
alcohol	-1.201025e-16
quality	7.572438e-17

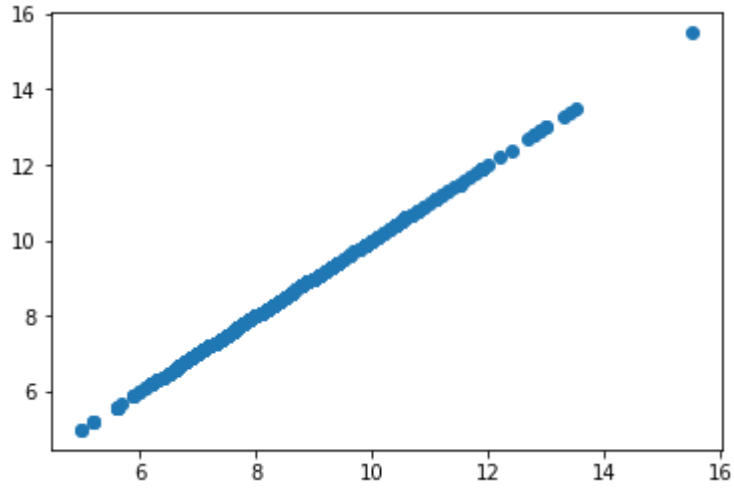
In [145]:

```
#Predicting

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

Out[145]:

```
<matplotlib.collections.PathCollection at 0x2485cfa1b80>
```



In [146]:

```
# Score

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

```
1.0
```

In [201]:

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

Out[201]:

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

In [202]:

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

Out[202]:

```
0.9992345048566785
```

In [203]:

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

Out[203]:

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

In [204]:

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

Out[204]:

```
-0.005181110800198008
```

DataSet MobilePrices

In [148]:

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

Out[148]:

	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 P2((MTK6762 Processo

1836 rows × 11 columns

In [149]:

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

Out[149]:

	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 [150]:

```
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 [151]:

```
a.describe()
```

Out[151]:

	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 [152]:

```
a.isna()
```

Out[152]:

	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 [153]:

```
a.dropna()
```

Out[153]:

	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 P2 (MTK6762 Processo

1291 rows × 11 columns

In [154]:

```
a.columns
```

Out[154]:

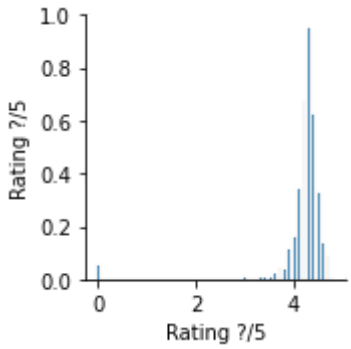
```
Index(['Phone Name', 'Rating ?/5', 'Number of Ratings', 'RAM', 'ROM/Storag
e',
       'Back/Rare Camera', 'Front Camera', 'Battery', 'Processor',
       'Price in INR', 'Date of Scraping'],
      dtype='object')
```

In [155]:

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

Out[155]:

```
<seaborn.axisgrid.PairGrid at 0x2485cfcf640>
```



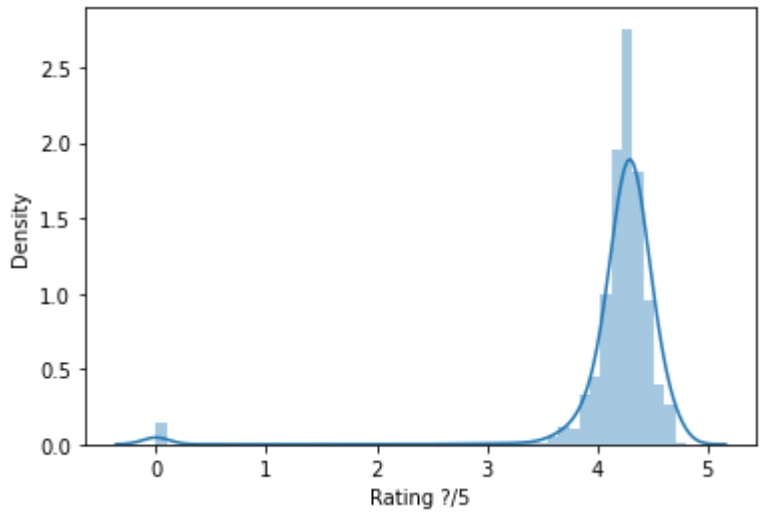
In [156]:

```
#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[156]:

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



In [157]:

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

Out[157]:

```
<AxesSubplot:>
```

In [160]:

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

In [161]:

```
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[161]:

```
LinearRegression()
```

In [162]:

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

```
8.881784197001252e-16
```

In [163]:

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

Out[163]:

	Co-efficient
Rating ?/5	1.0

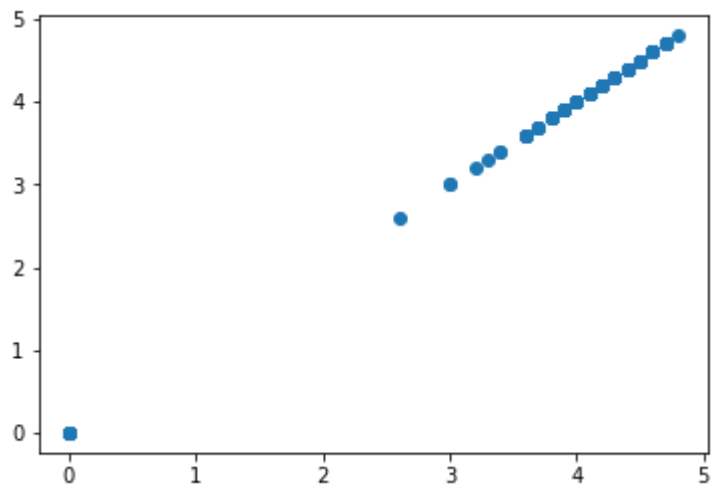
In [164]:

```
#Predicting

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

Out[164]:

```
<matplotlib.collections.PathCollection at 0x2485e391c10>
```



In [165]:

```
# Score

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

```
1.0
```

In [205]:

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

Out[205]:

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

In [206]:

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

Out[206]:

```
0.9992345048566785
```

In [207]:

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

Out[207]:

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

In [209]:

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

Out[209]:

```
-0.005181110800198008
```

In [ ]:

```

```