D1

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df=pd.read_csv(r"C:\Users\user\Downloads\1_fiat500_VehicleSelection_Dataset.csv
df
```

Out[2]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	l
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
...	...	...	...	...	...	...	...	
1544	NaN	NaN	NaN	NaN	NaN	NaN	NaN	leng
1545	NaN	NaN	NaN	NaN	NaN	NaN	NaN	con
1546	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Null valu
1547	NaN	NaN	NaN	NaN	NaN	NaN	NaN	fi
1548	NaN	NaN	NaN	NaN	NaN	NaN	NaN	sear

1549 rows × 11 columns

In [3]:
```
df.head(10)
```

Out[3]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon
0	1.0	lounge	51.0	882.0	25000.0	1.0	44.907242	8.611559868
1	2.0	pop	51.0	1186.0	32500.0	1.0	45.666359	12.24188995
2	3.0	sport	74.0	4658.0	142228.0	1.0	45.503300	11.41784
3	4.0	lounge	51.0	2739.0	160000.0	1.0	40.633171	17.63460922
4	5.0	pop	73.0	3074.0	106880.0	1.0	41.903221	12.49565029
5	6.0	pop	74.0	3623.0	70225.0	1.0	45.000702	7.68227005
6	7.0	lounge	51.0	731.0	11600.0	1.0	44.907242	8.611559868
7	8.0	lounge	51.0	1521.0	49076.0	1.0	41.903221	12.49565029
8	9.0	sport	73.0	4049.0	76000.0	1.0	45.548000	11.54946995
9	10.0	sport	51.0	3653.0	89000.0	1.0	45.438301	10.99170017

In [4]:
```
df.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 [5]:
```
df.describe()
```

Out[5]:

	ID	engine_power	age_in_days	km	previous_owners	lat	l
count	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	
mean	769.500000	51.904421	1650.980494	53396.011704	1.123537	43.541361	
std	444.126671	3.988023	1289.522278	40046.830723	0.416423	2.133518	
min	1.000000	51.000000	366.000000	1232.000000	1.000000	36.855839	
25%	385.250000	51.000000	670.000000	20006.250000	1.000000	41.802990	
50%	769.500000	51.000000	1035.000000	39031.000000	1.000000	44.394096	
75%	1153.750000	51.000000	2616.000000	79667.750000	1.000000	45.467960	
max	1538.000000	77.000000	4658.000000	235000.000000	4.000000	46.795612	

In [6]:
```
dft=df.drop(["Unnamed: 9","Unnamed: 10"],axis=1)
```

In [7]:
```
dff=dft.dropna()
```

In [8]:
```
dff.isnull().sum()
```

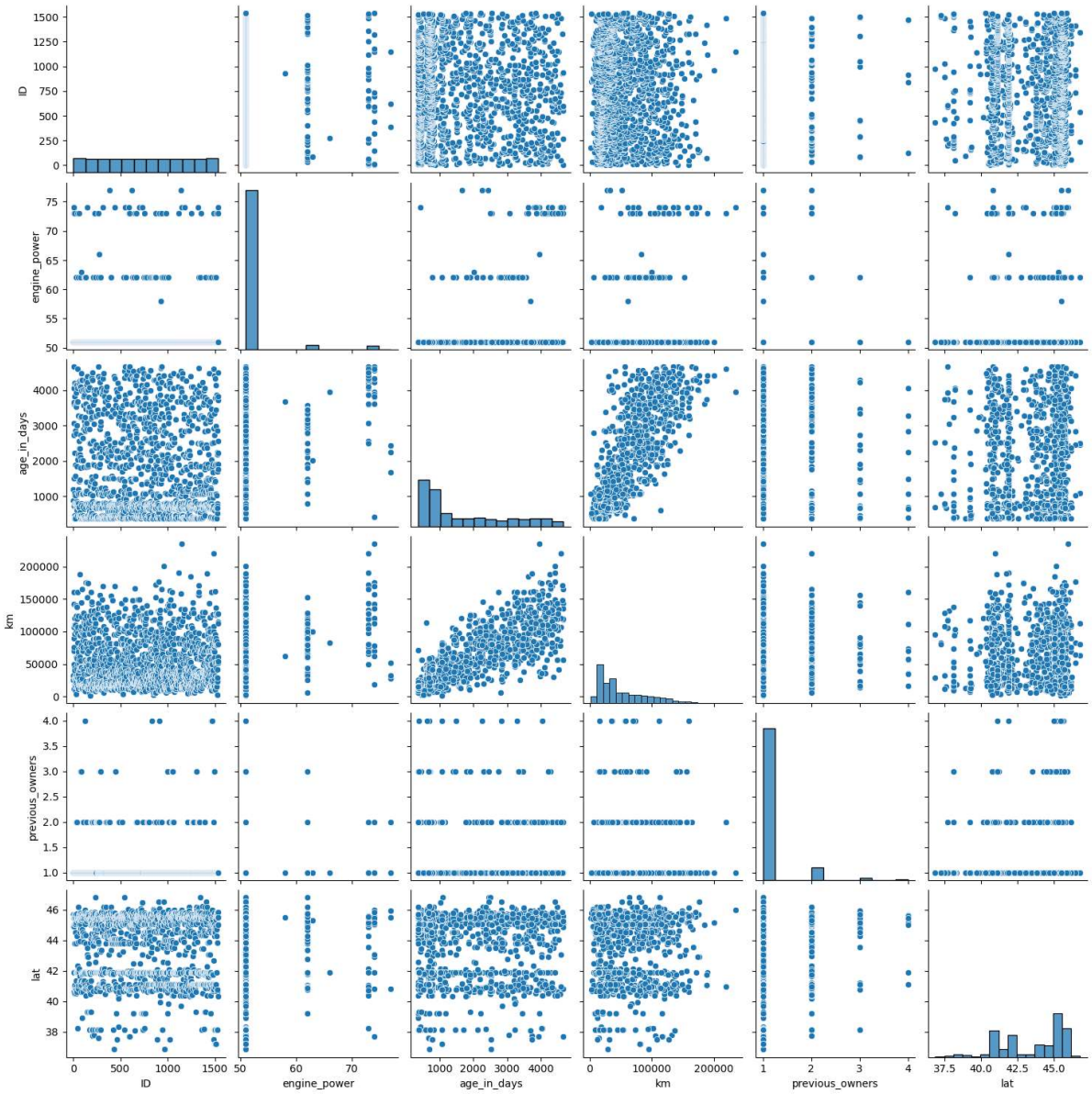
Out[8]:
```
ID                0
model             0
engine_power      0
age_in_days       0
km                0
previous_owners   0
lat               0
lon               0
price             0
dtype: int64
```

In [9]:
```
dff.columns
```

Out[9]:
```
Index(['ID', 'model', 'engine_power', 'age_in_days', 'km', 'previous_owners',
       'lat', 'lon', 'price'],
      dtype='object')
```

In [10]:
```
sns.pairplot(dff)
```

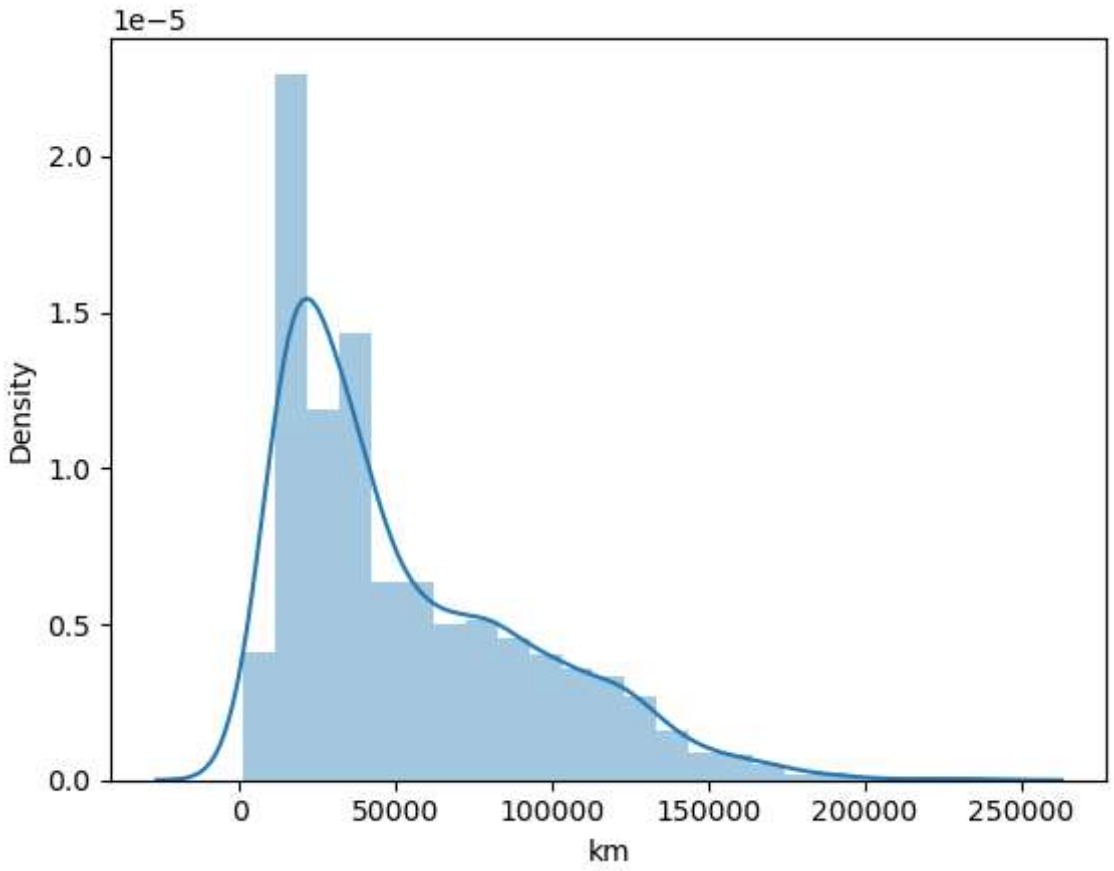
Out[10]:
```
<seaborn.axisgrid.PairGrid at 0x1ad26e0eed0>
```



In [11]:
```
sns.distplot(dff["km"])
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_8148\596799067.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(dff["km"])
```

Out[11]: <Axes: xlabel='km', ylabel='Density'>

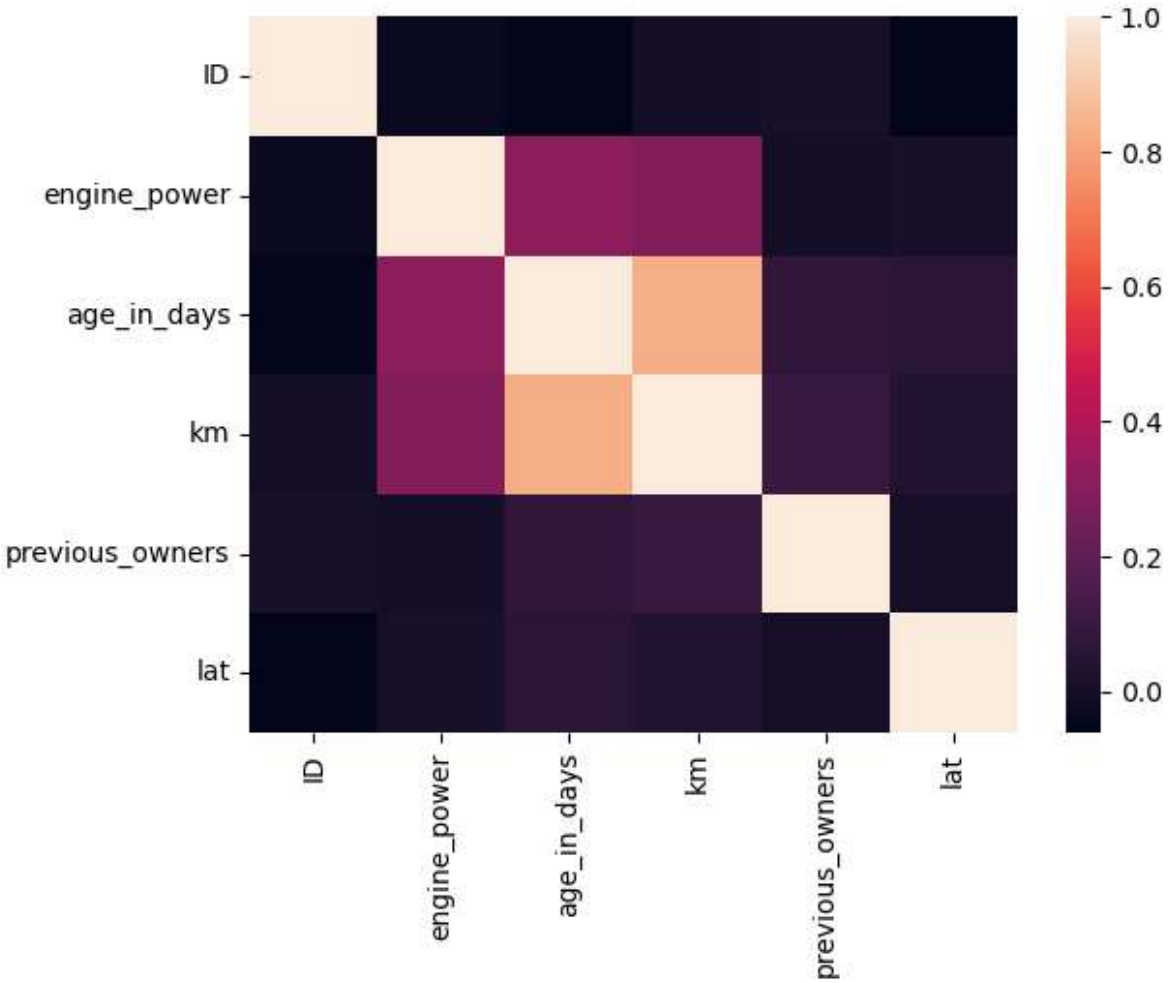


In [12]:
```
df1=dff[['ID', 'model', 'engine_power', 'age_in_days', 'km', 'previous_owners'
       'lat', 'lon', 'price']]
```

In [13]:
```
sns.heatmap(df1.corr())
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_8148\781785195.py:1: FutureWarnin
g: The default value of numeric_only in DataFrame.corr is deprecated. In a fu
ture version, it will default to False. Select only valid columns or specify
the value of numeric_only to silence this warning.
  sns.heatmap(df1.corr())
```

Out[13]: <Axes: >



In [14]:
```
x=df1[['ID', 'engine_power', 'age_in_days', 'previous_owners',
       'lat']]
y=df1['km']
```

In [15]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [16]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[16]:
```
LinearRegression
LinearRegression()
```

In [17]:
```
print(lr.intercept_)
```

```
-3742.018457785569
```

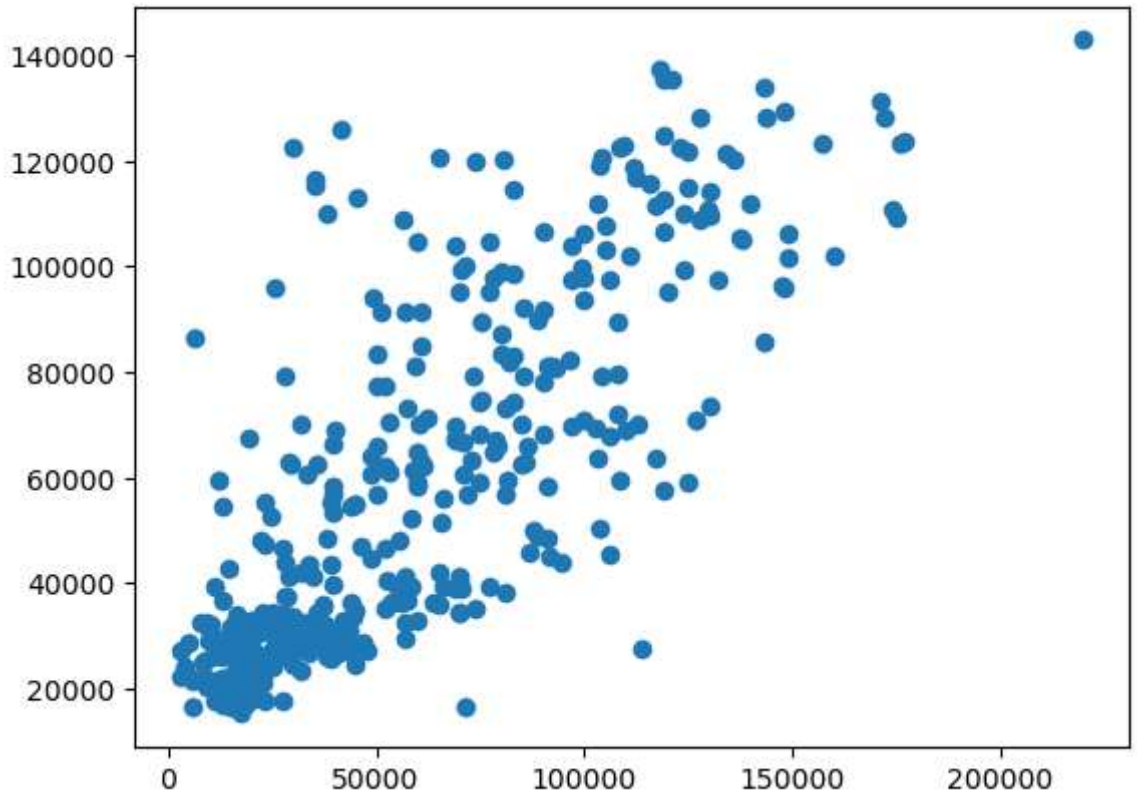
In [18]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[18]:

	Co-efficient
ID	4.733338
engine_power	310.433868
age_in_days	25.803759
previous_owners	3766.548261
lat	-224.134735

In [19]:
```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[19]:
```
<matplotlib.collections.PathCollection at 0x1ad2c5b4c10>
```



In [20]:
```
print(lr.score(x_test,y_test))
```

```
0.6518748273974636
```

In [21]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [22]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[22]:
```
Ridge(alpha=10)
```

In [23]:
```
rr.score(x_test,y_test)
```

Out[23]:
```
0.6519347562577196
```

In [24]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[24]:
```
Lasso(alpha=10)
```

In [25]:
```
la.score(x_test,y_test)
```

Out[25]:
```
0.6518956328194522
```

In [ ]:

D2

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df=pd.read_csv(r"C:\Users\user\Downloads\2_2015.csv")
df
```

Out[2]:

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

158 rows × 12 columns

In [3]:
```
df.head(10)
```

Out[3]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freec
0	Switzerland	Western Europe	1	7.587	0.03411	1.39651	1.34951	0.94143	0.66
1	Iceland	Western Europe	2	7.561	0.04884	1.30232	1.40223	0.94784	0.62
2	Denmark	Western Europe	3	7.527	0.03328	1.32548	1.36058	0.87464	0.64
3	Norway	Western Europe	4	7.522	0.03880	1.45900	1.33095	0.88521	0.66
4	Canada	North America	5	7.427	0.03553	1.32629	1.32261	0.90563	0.63
5	Finland	Western Europe	6	7.406	0.03140	1.29025	1.31826	0.88911	0.64
6	Netherlands	Western Europe	7	7.378	0.02799	1.32944	1.28017	0.89284	0.61
7	Sweden	Western Europe	8	7.364	0.03157	1.33171	1.28907	0.91087	0.65
8	New Zealand	Australia and New Zealand	9	7.286	0.03371	1.25018	1.31967	0.90837	0.63
9	Australia	Australia and New Zealand	10	7.284	0.04083	1.33358	1.30923	0.93156	0.65

In [4]:
```
df.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 [5]:
```
df.describe()
```

Out[5]:

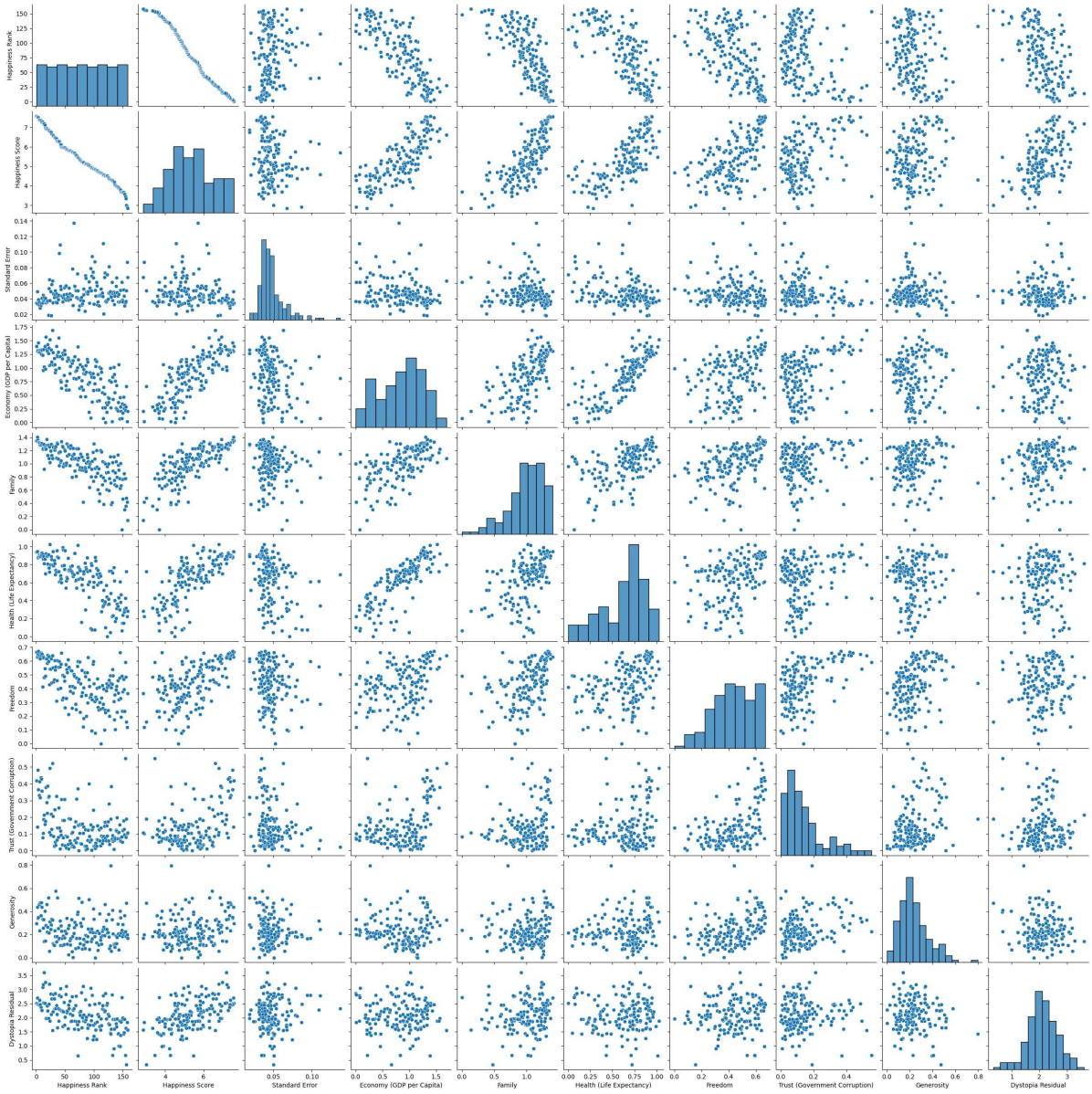
	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	(G C
count	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	1
mean	79.493671	5.375734	0.047885	0.846137	0.991046	0.630259	0.428615	
std	45.754363	1.145010	0.017146	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 [6]:
```
df.columns
```

Out[6]:
```
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 [7]:
```
sns.pairplot(df)
```

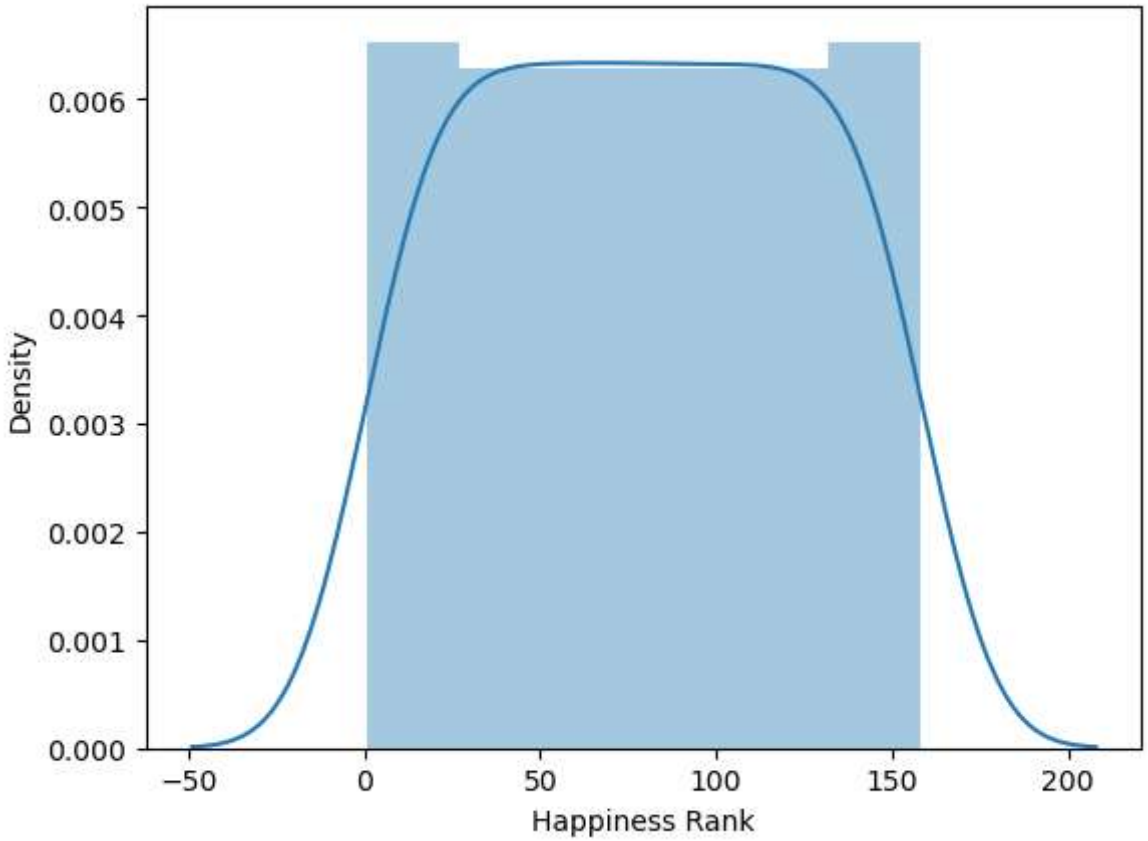
Out[7]:
```
<seaborn.axisgrid.PairGrid at 0x179361ff2d0>
```



In [8]:
```
sns.distplot(df["Happiness Rank"])
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_9896\2893626210.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(df["Happiness Rank"])
```

Out[8]: <Axes: xlabel='Happiness Rank', ylabel='Density'>

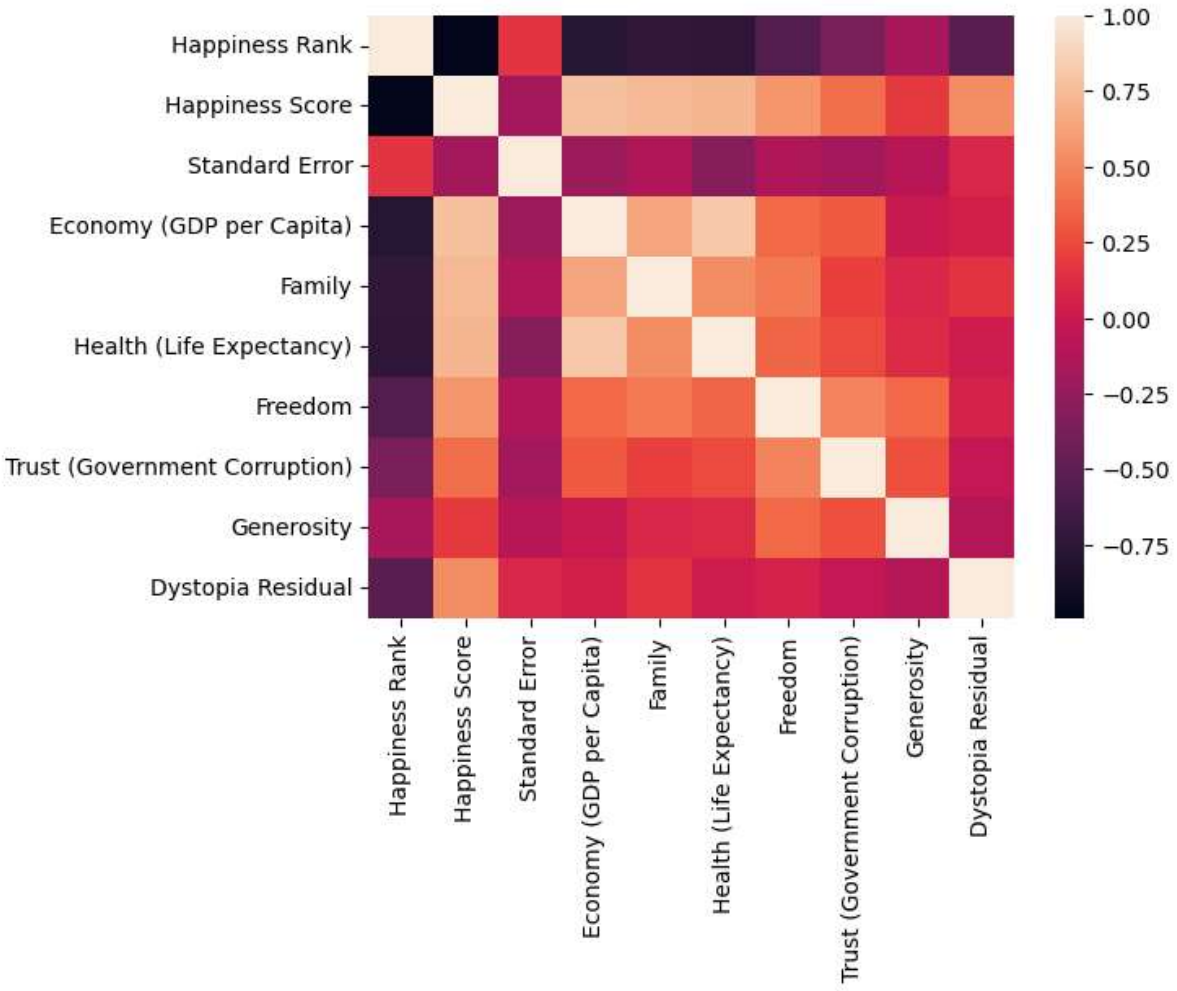


In [9]:
```
df1=df[['Country', 'Region', 'Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruption)',
       'Generosity', 'Dystopia Residual']]
```

In [10]:
```
sns.heatmap(df1.corr())
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_9896\781785195.py:1: FutureWarnin
g: The default value of numeric_only in DataFrame.corr is deprecated. In a fu
ture version, it will default to False. Select only valid columns or specify
the value of numeric_only to silence this warning.
  sns.heatmap(df1.corr())
```

Out[10]: <Axes: >



In [11]:
```
x=df1[['Happiness Rank',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruption)',
       'Generosity', 'Dystopia Residual']]
y=df1['Happiness Score']
```

In [12]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [13]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[13]:
```
LinearRegression
LinearRegression()
```

In [14]:
```
print(lr.intercept_)
```

```
0.001820616265106878
```

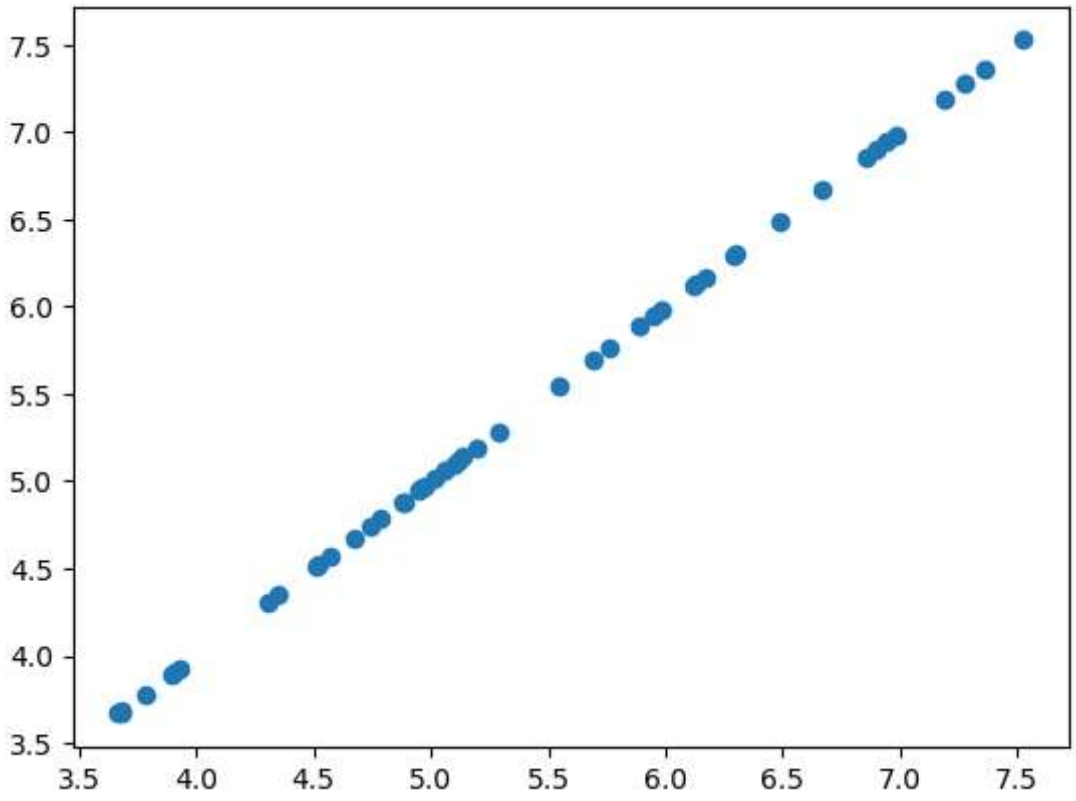
In [15]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[15]:

	Co-efficient
Happiness Rank	-0.000006
Standard Error	-0.000845
Economy (GDP per Capita)	0.999816
Family	0.999852
Health (Life Expectancy)	0.999557
Freedom	0.999355
Trust (Government Corruption)	0.999906
Generosity	0.999879
Dystopia Residual	0.999791

In [16]:
```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[16]:
```
<matplotlib.collections.PathCollection at 0x17945a78650>
```



In [17]:
```
print(lr.score(x_test,y_test))
```

```
0.9999999338448409
```

In [18]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [19]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[19]:
```
Ridge(alpha=10)
```

In [20]:
```
rr.score(x_test,y_test)
```

Out[20]:
```
0.989483996887005
```

In [21]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[21]:
```
Lasso(alpha=10)
```

In [22]:
```
la.score(x_test,y_test)
```

Out[22]:
```
0.9599137953301835
```

In [ ]:
```

```

D3

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df=pd.read_csv(r"C:\Users\user\Downloads\4_drug200.csv")
df
```

Out[2]:

	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 [3]:
```
df.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 [4]:
```
df.describe()
```

Out[4]:

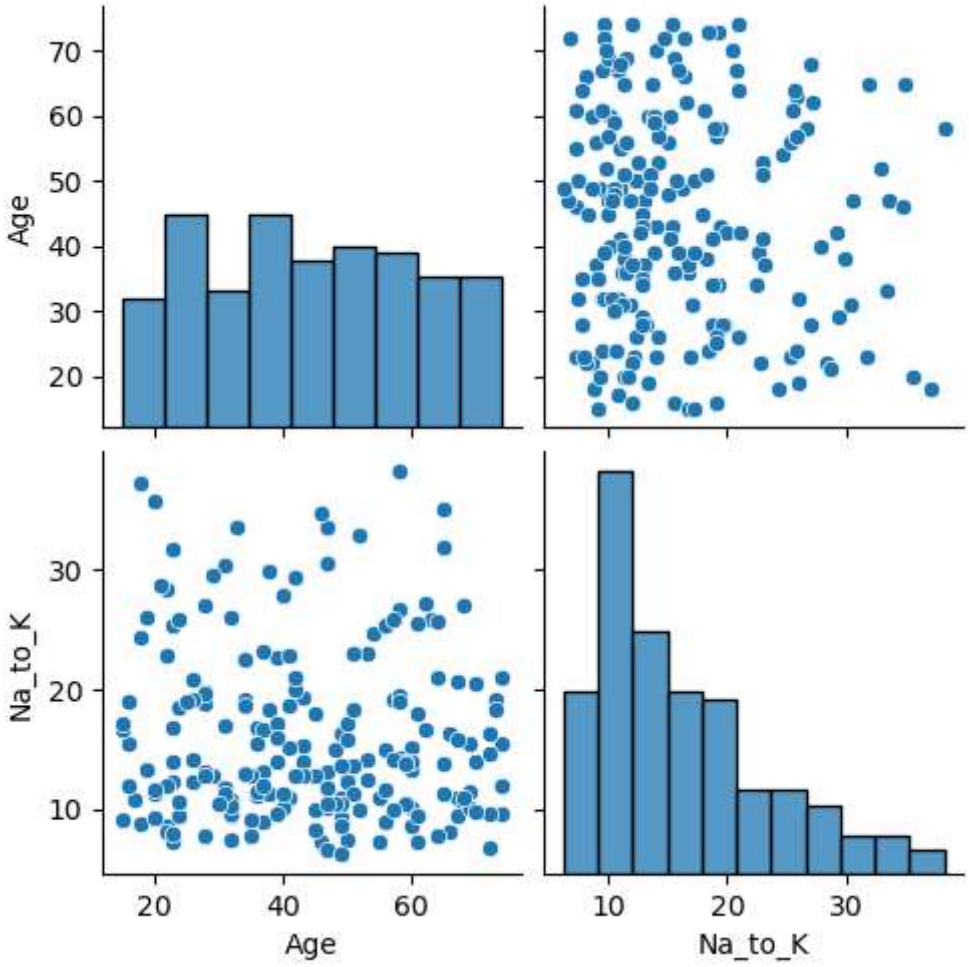
	Age	Na_to_K
count	200.000000	200.000000
mean	44.315000	16.084485
std	16.544315	7.223956
min	15.000000	6.269000
25%	31.000000	10.445500
50%	45.000000	13.936500
75%	58.000000	19.380000
max	74.000000	38.247000

In [5]:
```
df.columns
```

Out[5]:
```
Index(['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K', 'Drug'], dtype='object')
```

In [6]:
```
sns.pairplot(df)
```

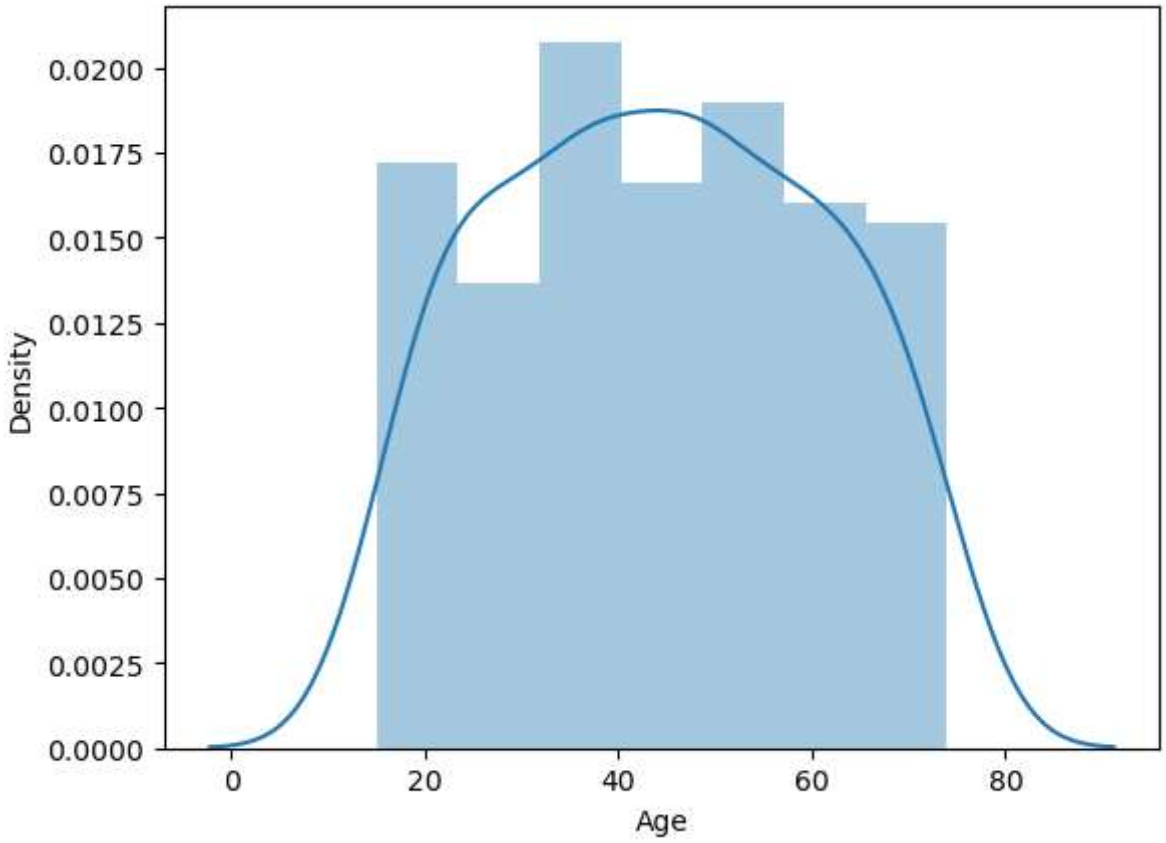
Out[6]:
```
<seaborn.axisgrid.PairGrid at 0x1eb144434d0>
```



In [7]:
```
sns.distplot(df["Age"])
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_7792\2732350774.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

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

Out[7]: <Axes: xlabel='Age', ylabel='Density'>

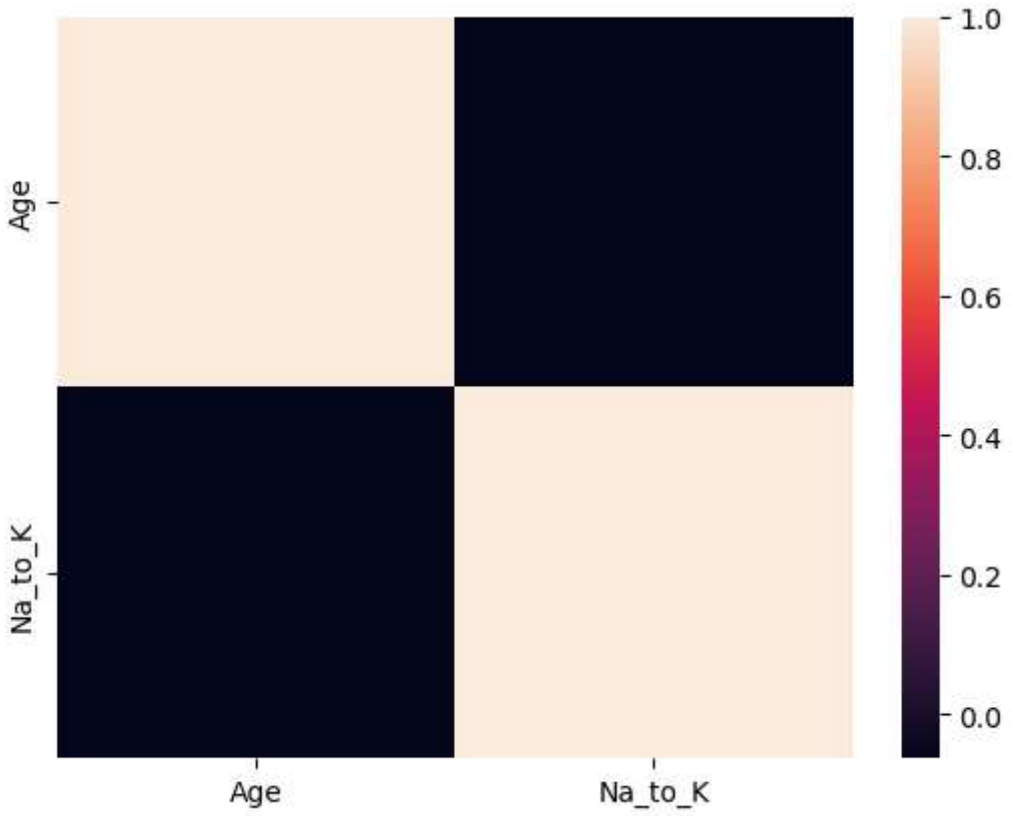


In [8]:
```
df1=df[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K', 'Drug']]
```

In [9]:
```
sns.heatmap(df1.corr())
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_7792\781785195.py:1: FutureWarnin
g: The default value of numeric_only in DataFrame.corr is deprecated. In a fu
ture version, it will default to False. Select only valid columns or specify
the value of numeric_only to silence this warning.
  sns.heatmap(df1.corr())
```

Out[9]: <Axes: >



In [10]:
```
x=df1[['Age']]
y=df1['Na_to_K']
```

In [11]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [12]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[12]:
```
LinearRegression
LinearRegression()
```

In [13]:
```
print(lr.intercept_)
```

```
15.435320212781702
```

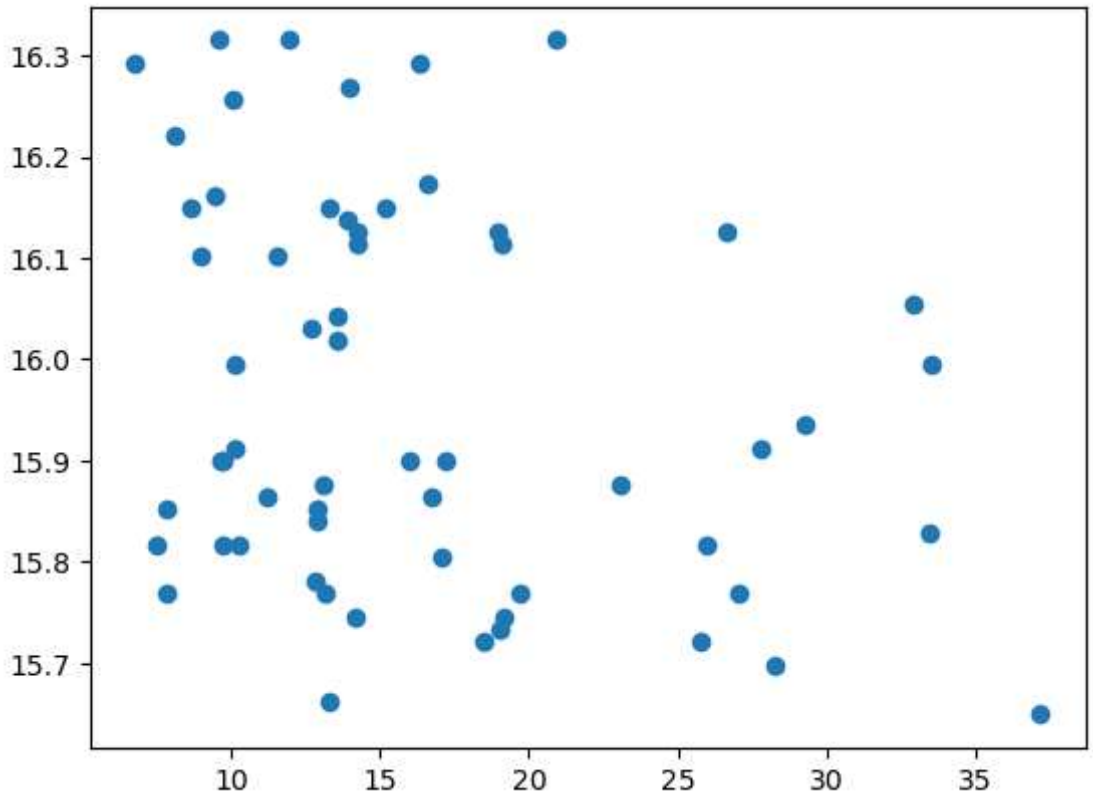
In [14]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[14]:

	Co-efficient
Age	0.011889

In [15]:
```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[15]:
```
<matplotlib.collections.PathCollection at 0x1eb0f056250>
```



In [16]:
```
print(lr.score(x_test,y_test))
```

```
-0.017709603394363116
```

In [17]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [18]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[18]:
```
Ridge(alpha=10)
```

In [19]:
```
rr.score(x_test,y_test)
```

Out[19]:
```
-0.017705688018912147
```

In [20]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[20]:
```
Lasso(alpha=10)
```

In [21]:
```
la.score(x_test,y_test)
```

Out[21]:
```
-0.0030546432738367546
```

In [ ]:

D4

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df=pd.read_csv(r"C:\Users\user\Downloads\6_Salesworkload1.csv")
df
```

Out[2]:

	MonthYear	Time index	Country	StoreID	City	Dept_ID	Dept. Name	HoursOwn	HoursLe
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 [3]:
```
df.head(10)
```

Out[3]:

	MonthYear	Time index	Country	StoreID	City	Dept_ID	Dept. Name	HoursOwn	HoursLease	
0	10.2016	1.0	United Kingdom	88253.0	London (I)	1.0	Dry	3184.764	0.0	3
1	10.2016	1.0	United Kingdom	88253.0	London (I)	2.0	Frozen	1582.941	0.0	
2	10.2016	1.0	United Kingdom	88253.0	London (I)	3.0	other	47.205	0.0	4
3	10.2016	1.0	United Kingdom	88253.0	London (I)	4.0	Fish	1623.852	0.0	3
4	10.2016	1.0	United Kingdom	88253.0	London (I)	5.0	Fruits & Vegetables	1759.173	0.0	1
5	10.2016	1.0	United Kingdom	88253.0	London (I)	6.0	Meat	8270.316	0.0	17
6	10.2016	1.0	United Kingdom	88253.0	London (I)	13.0	Food	16468.251	0.0	31
7	10.2016	1.0	United Kingdom	88253.0	London (I)	7.0	Clothing	4698.471	0.0	2
8	10.2016	1.0	United Kingdom	88253.0	London (I)	8.0	Household	1183.272	0.0	
9	10.2016	1.0	United Kingdom	88253.0	London (I)	9.0	Hardware	2029.815	0.0	

In [4]:
```
df.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 [5]:
```
dff=df.drop("Customer",axis=1)
```

In [6]:
```
dft=dff.dropna()
```

In [7]:
```
dft.isnull().sum()
```

Out[7]:
```
MonthYear        0
Time index       0
Country          0
StoreID          0
City             0
Dept_ID          0
Dept. Name       0
HoursOwn         0
HoursLease       0
Sales units      0
Turnover         0
Area (m2)        0
Opening hours    0
dtype: int64
```

In [8]:
```
dft.describe()
```

Out[8]:

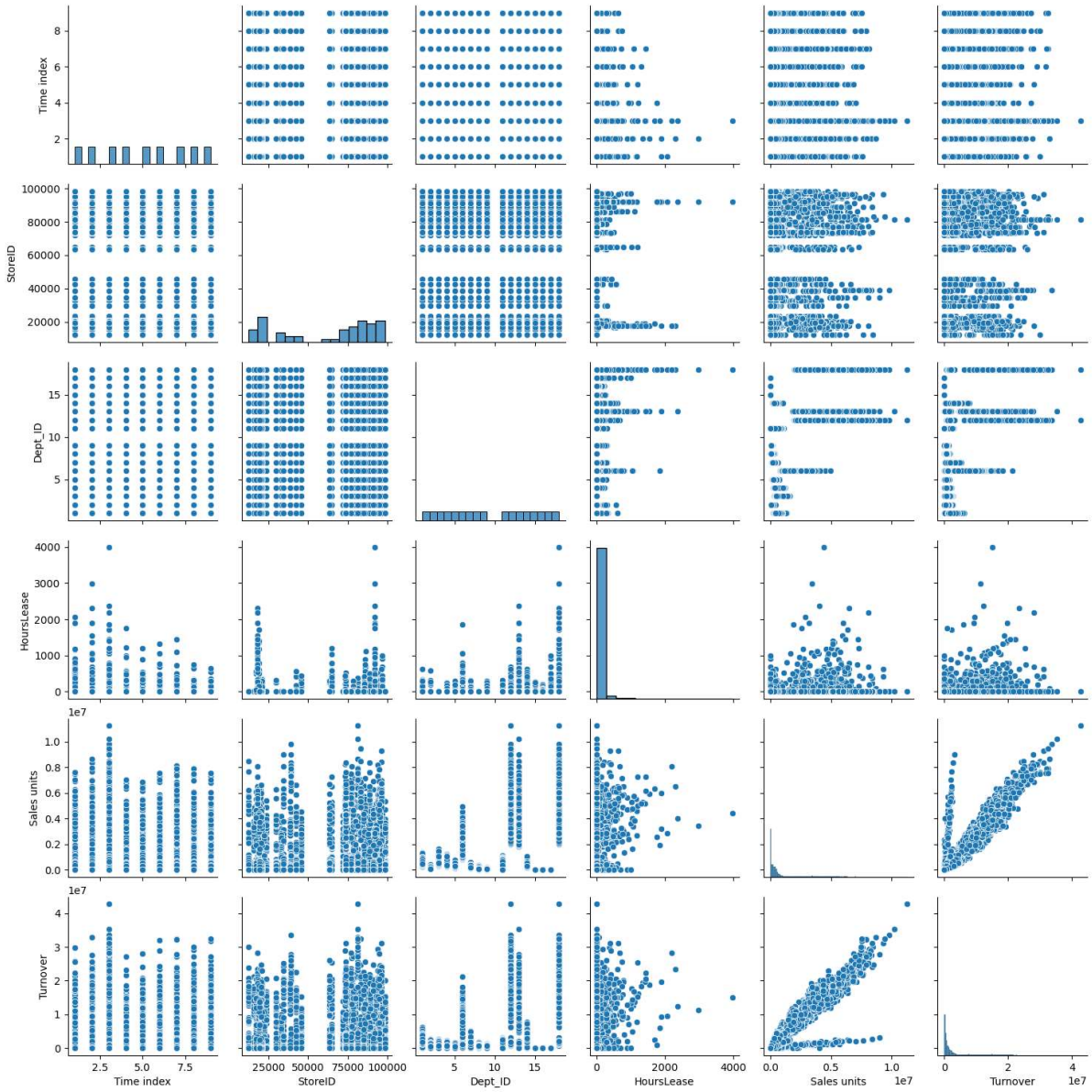
	Time index	StoreID	Dept_ID	HoursLease	Sales units	Turnover
count	7650.000000	7650.000000	7650.000000	7650.000000	7.650000e+03	7.650000e+03
mean	5.000000	61995.220000	9.470588	22.036078	1.076471e+06	3.721393e+06
std	2.582158	29924.581631	5.337429	133.299513	1.728113e+06	6.003380e+06
min	1.000000	12227.000000	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 [9]:
```
dft.columns
```

Out[9]:
```
Index(['MonthYear', 'Time index', 'Country', 'StoreID', 'City', 'Dept_ID',
       'Dept. Name', 'HoursOwn', 'HoursLease', 'Sales units', 'Turnover',
       'Area (m2)', 'Opening hours'],
      dtype='object')
```

In [10]:
```
sns.pairplot(dft)
```

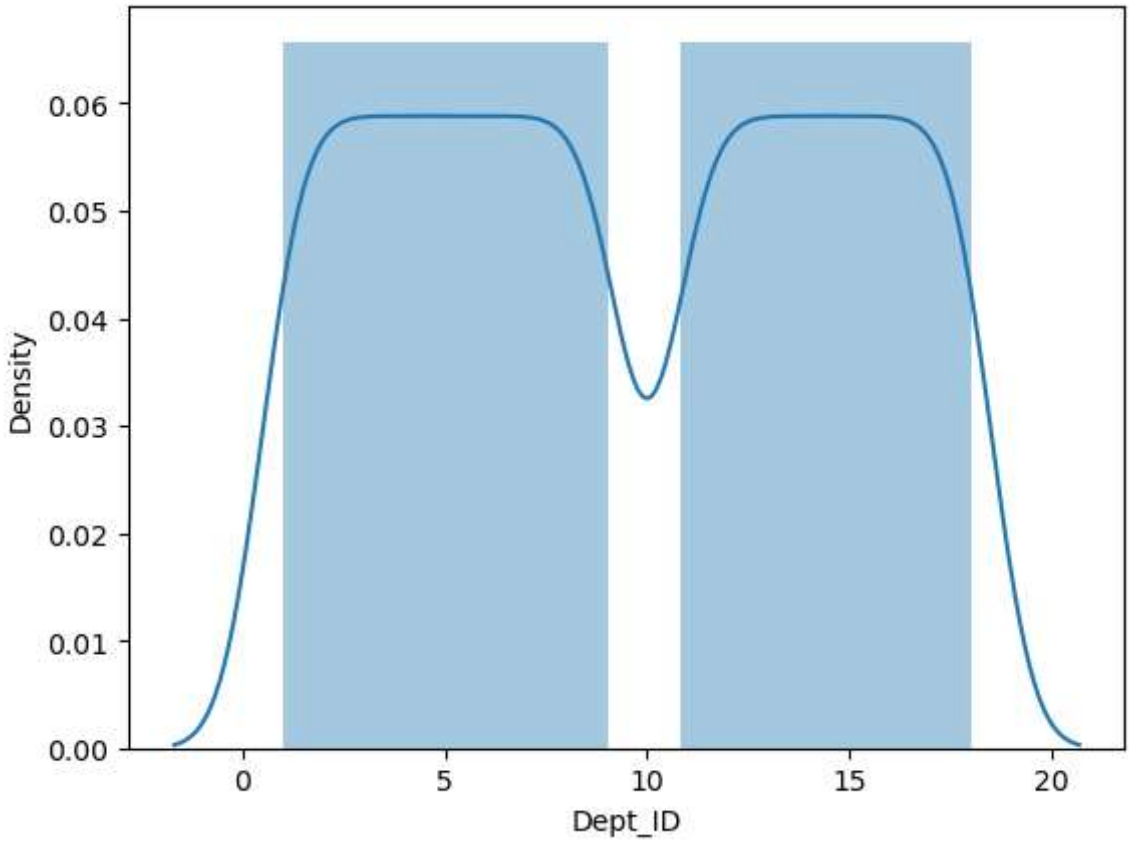
Out[10]:
```
<seaborn.axisgrid.PairGrid at 0x1c721d76b10>
```



In [11]:
```
sns.distplot(dft["Dept_ID"])
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_2176\3941466768.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(dft["Dept_ID"])
```

Out[11]: <Axes: xlabel='Dept_ID', ylabel='Density'>

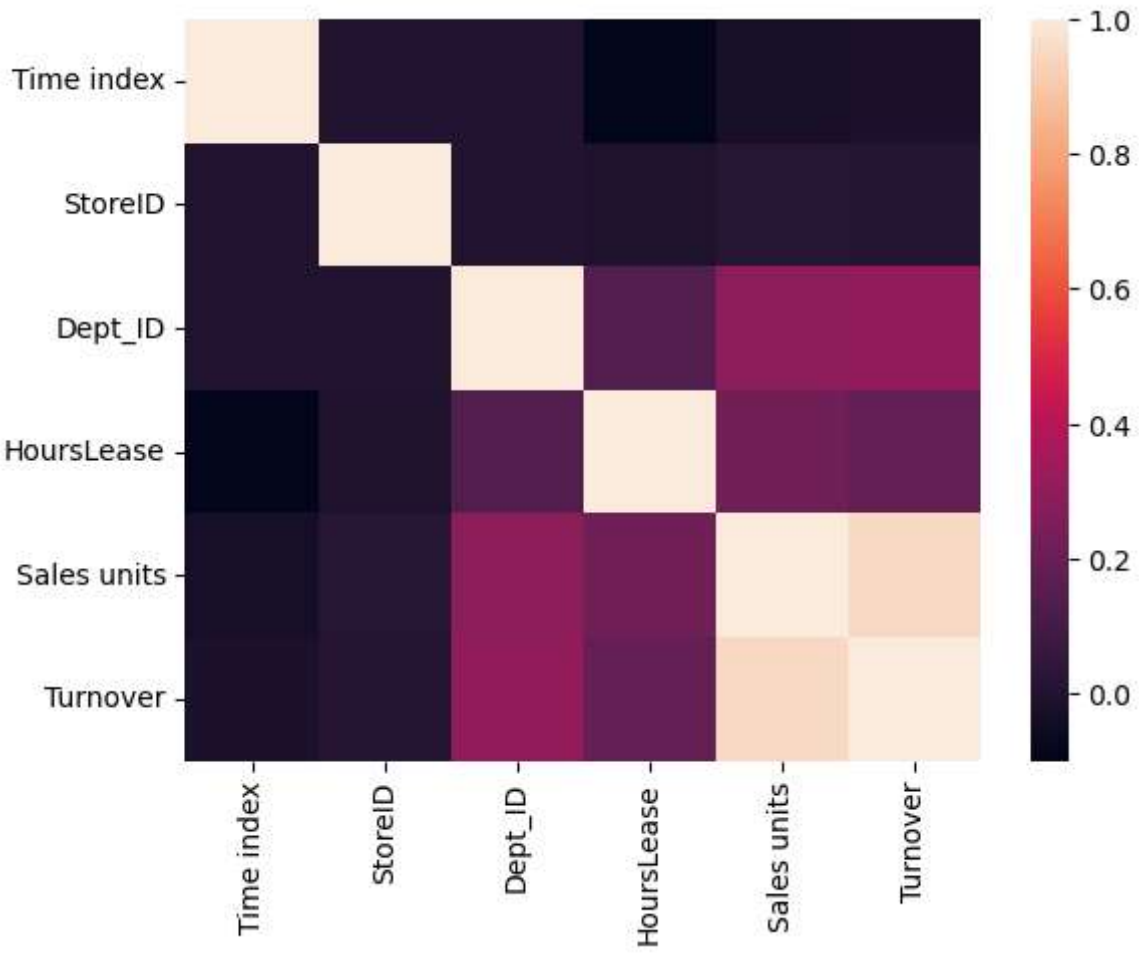


In [12]:
```
df1=dft[['MonthYear', 'Time index', 'Country', 'StoreID', 'City', 'Dept_ID',
       'Dept. Name', 'HoursOwn', 'HoursLease', 'Sales units', 'Turnover',
       'Area (m2)', 'Opening hours']]
```

In [13]:
```
sns.heatmap(df1.corr())
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_2176\781785195.py:1: FutureWarnin
g: The default value of numeric_only in DataFrame.corr is deprecated. In a fu
ture version, it will default to False. Select only valid columns or specify
the value of numeric_only to silence this warning.
  sns.heatmap(df1.corr())
```

Out[13]: <Axes: >



In [14]:
```
x=df1[['Time index', 'StoreID', 'HoursLease', 'Sales units', 'Turnover']]
y=df1['Dept_ID']
```

In [15]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [16]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[16]:
```
▾ LinearRegression

LinearRegression()
```

In [17]:
```
print(lr.intercept_)
```

```
8.083819329695611
```

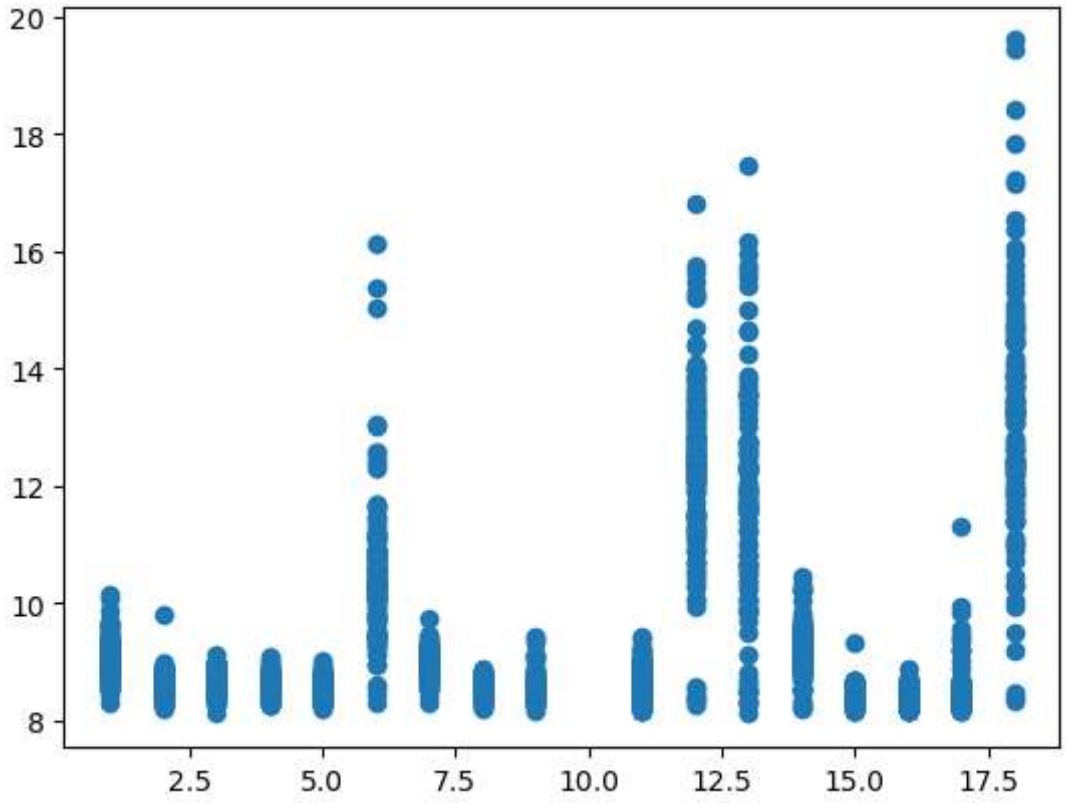
In [18]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[18]:

	Co-efficient
Time index	5.260571e-02
StoreID	1.367194e-06
HoursLease	3.371636e-03
Sales units	-6.930884e-08
Turnover	2.768417e-07

In [19]:
```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[19]:
```
<matplotlib.collections.PathCollection at 0x1c7273374d0>
```



In [20]:
```
print(lr.score(x_test,y_test))
```

```
0.08536024223347483
```

In [21]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [22]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[22]:
```
Ridge(alpha=10)
```

In [23]:
```
rr.score(x_test,y_test)
```

Out[23]:
```
0.08536080018124004
```

In [24]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[24]:
```
Lasso(alpha=10)
```

In [25]:
```
la.score(x_test,y_test)
```

Out[25]:
```
0.08655760424826331
```

In [ ]:

D5

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df=pd.read_csv(r"C:\Users\user\Downloads\7_uber.csv")
df
```

Out[2]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_l
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.
...	...	...	...	...	...	...
199995	42598914	2012-10-28 10:49:00.00000053	3.0	2012-10-28 10:49:00 UTC	-73.987042	40.
199996	16382965	2014-03-14 01:09:00.0000008	7.5	2014-03-14 01:09:00 UTC	-73.984722	40.
199997	27804658	2009-06-29 00:42:00.00000078	30.9	2009-06-29 00:42:00 UTC	-73.986017	40.
199998	20259894	2015-05-20 14:56:25.0000004	14.5	2015-05-20 14:56:25 UTC	-73.997124	40.
199999	11951496	2010-05-15 04:08:00.00000076	14.1	2010-05-15 04:08:00 UTC	-73.984395	40.

200000 rows × 9 columns

In [3]:
```
df.head()
```

Out[3]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitud
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.73835
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.72822
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.74077
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.79084
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.74408

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 9 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   Unnamed: 0         200000 non-null  int64  
 1   key                200000 non-null  object 
 2   fare_amount        200000 non-null  float64
 3   pickup_datetime    200000 non-null  object 
 4   pickup_longitude   200000 non-null  float64
 5   pickup_latitude    200000 non-null  float64
 6   dropoff_longitude  199999 non-null  float64
 7   dropoff_latitude   199999 non-null  float64
 8   passenger_count    200000 non-null  int64  
dtypes: float64(5), int64(2), object(2)
memory usage: 13.7+ MB
```

In [5]:
```
dff=df.dropna()
```

In [21]:
```
dff.describe()
```

Out[21]:

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff
count	1.999990e+05	199999.000000	199999.000000	199999.000000	199999.000000	19999
mean	2.771248e+07	11.359892	-72.527631	39.935881	-72.525292	3
std	1.601386e+07	9.901760	11.437815	7.720558	13.117408	
min	1.000000e+00	-52.000000	-1340.648410	-74.015515	-3356.666300	-88
25%	1.382534e+07	6.000000	-73.992065	40.734796	-73.991407	4
50%	2.774524e+07	8.500000	-73.981823	40.752592	-73.980093	4
75%	4.155535e+07	12.500000	-73.967154	40.767158	-73.963658	4
max	5.542357e+07	499.000000	57.418457	1644.421482	1153.572603	87

In [22]:
```
dff.columns
```

Out[22]:
```
Index(['Unnamed: 0', 'key', 'fare_amount', 'pickup_datetime',
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'passenger_count'],
      dtype='object')
```

In [23]:
```
dft=dff[['Unnamed: 0', 'fare_amount',
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'passenger_count']]
```

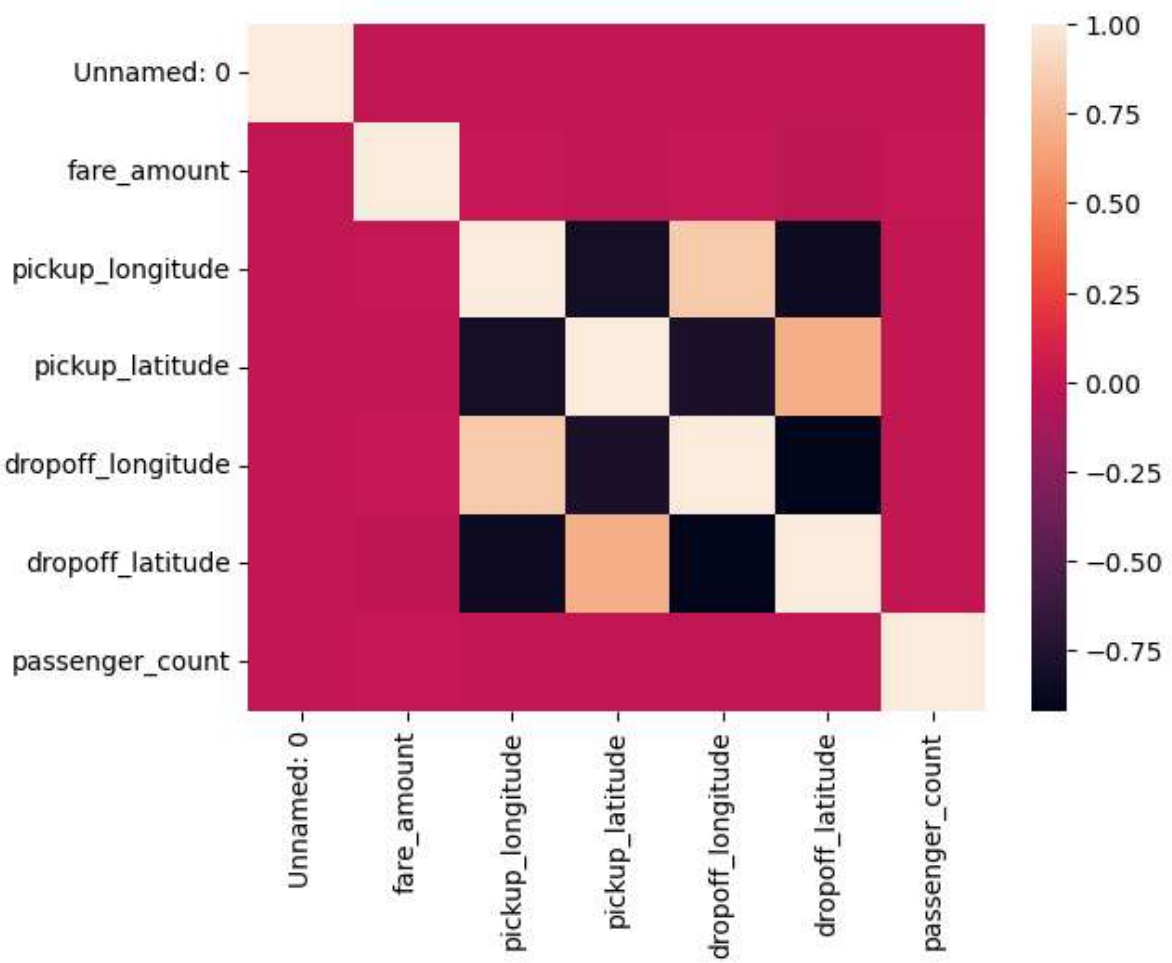
In [ ]:
```
sns.pairplot(dft)
```

In [ ]:
```
sns.distplot(dft["fare_amount"])
```

In [24]:
```
df1=dft[['Unnamed: 0', 'fare_amount',
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'passenger_count']]
```

In [25]:
```
sns.heatmap(df1.corr())
```

Out[25]: <Axes: >



In [39]:
```
x=df1[['Unnamed: 0','pickup_longitude', 'pickup_latitude', 'dropoff_longitude'
       'dropoff_latitude', 'passenger_count']]
y=df1['fare_amount']
```

In [40]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [41]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[41]:
```
▾ LinearRegression

LinearRegression()
```

In [42]:
```
print(lr.intercept_)
```

```
12.157257820553914
```

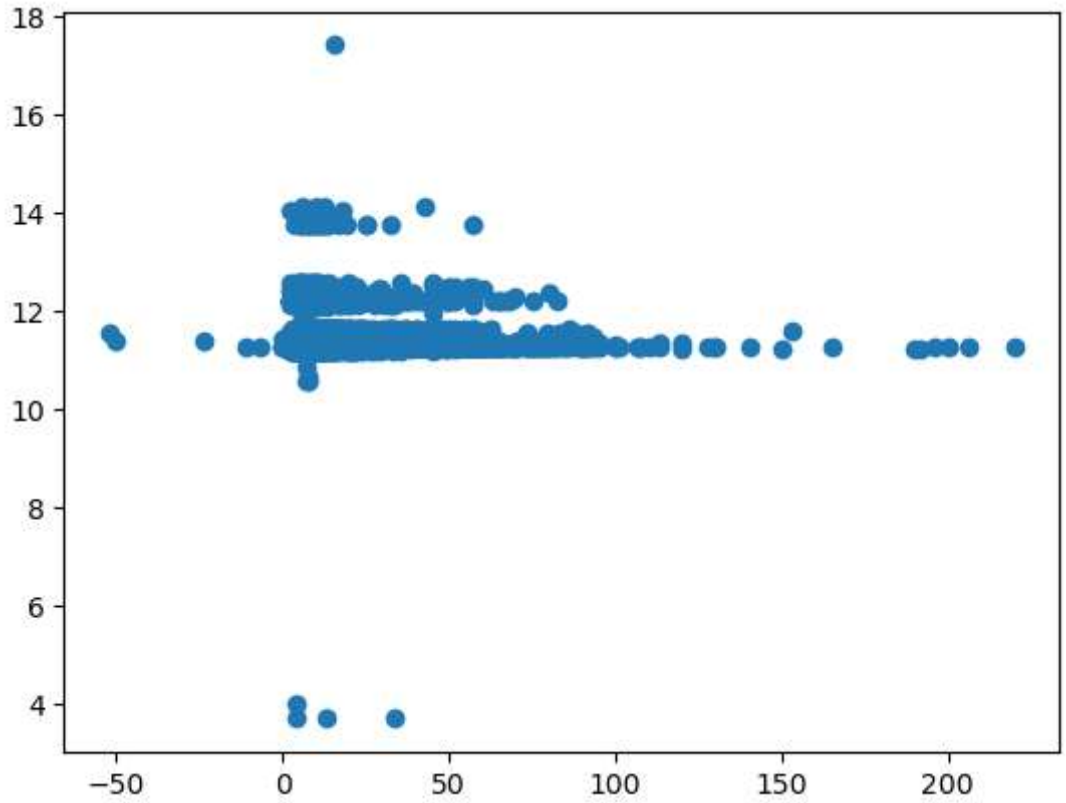
In [43]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[43]:

	Co-efficient
Unnamed: 0	1.778644e-10
pickup_longitude	1.141615e-02
pickup_latitude	-2.969902e-04
dropoff_longitude	-9.176792e-03
dropoff_latitude	-1.937769e-02
passenger_count	7.150869e-02

In [44]:
```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[44]: <matplotlib.collections.PathCollection at 0x21f516dea50>



In [45]:
```
print(lr.score(x_test,y_test))
```

```
-0.00010459028551013105
```

In [46]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [47]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[47]:
```
Ridge(alpha=10)
```

In [48]:
```
rr.score(x_test,y_test)
```

Out[48]:
```
-0.0001045904980259138
```

In [49]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[49]:
```
Lasso(alpha=10)
```

In [50]:
```
la.score(x_test,y_test)
```

Out[50]:
```
-7.285937013001842e-05
```

In [ ]:

D6

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [4]:
```
df=pd.read_csv(r"C:\Users\user\Downloads\8_BreastCancerPrediction.csv")
df
```

Out[4]:

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

569 rows × 33 columns

In [5]:
```
df.head(10)
```

Out[5]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_m
0	842302	M	17.99	10.38	122.80	1001.0	0.11
1	842517	M	20.57	17.77	132.90	1326.0	0.08
2	84300903	M	19.69	21.25	130.00	1203.0	0.10
3	84348301	M	11.42	20.38	77.58	386.1	0.14
4	84358402	M	20.29	14.34	135.10	1297.0	0.10
5	843786	M	12.45	15.70	82.57	477.1	0.12
6	844359	M	18.25	19.98	119.60	1040.0	0.09
7	84458202	M	13.71	20.83	90.20	577.9	0.11
8	844981	M	13.00	21.82	87.50	519.8	0.12
9	84501001	M	12.46	24.04	83.97	475.9	0.11

10 rows × 33 columns

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 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
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

In [5]:
```
dff=df.drop("Unnamed: 32",axis=1)
```

In [8]:
```
dff.describe()
```

Out[8]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.00000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.09636
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.01406
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.05263
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.08637
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.09587
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.10530
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.16340

8 rows × 31 columns

In [6]:
```
dff.columns
```

Out[6]:
```
Index(['id', 'diagnosis', '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', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

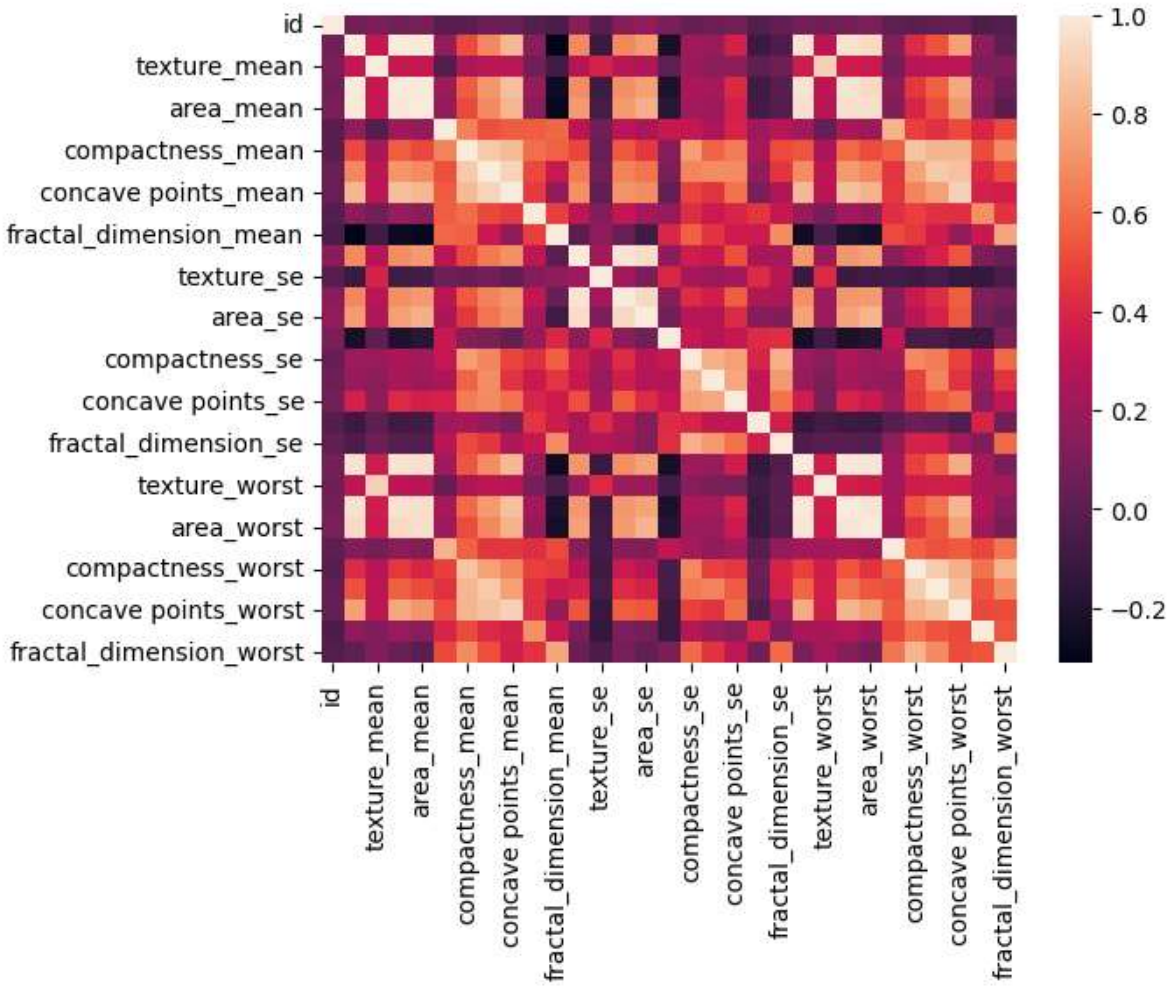
sns.pairplot(dff)

sns.distplot(dff["texture_worst"])

In [13]:
```
df1=df[['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', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst']]
```

In [14]:
```
sns.heatmap(df1.corr())
```

Out[14]: <Axes: >



In [16]:
```
x=df1[['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', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst']]
y=df1["texture_worst"]
```

In [17]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [18]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[18]:
```
LinearRegression
LinearRegression()
```

In [19]:
```
print(lr.intercept_)
```

```
6.573534250264856e-08
```

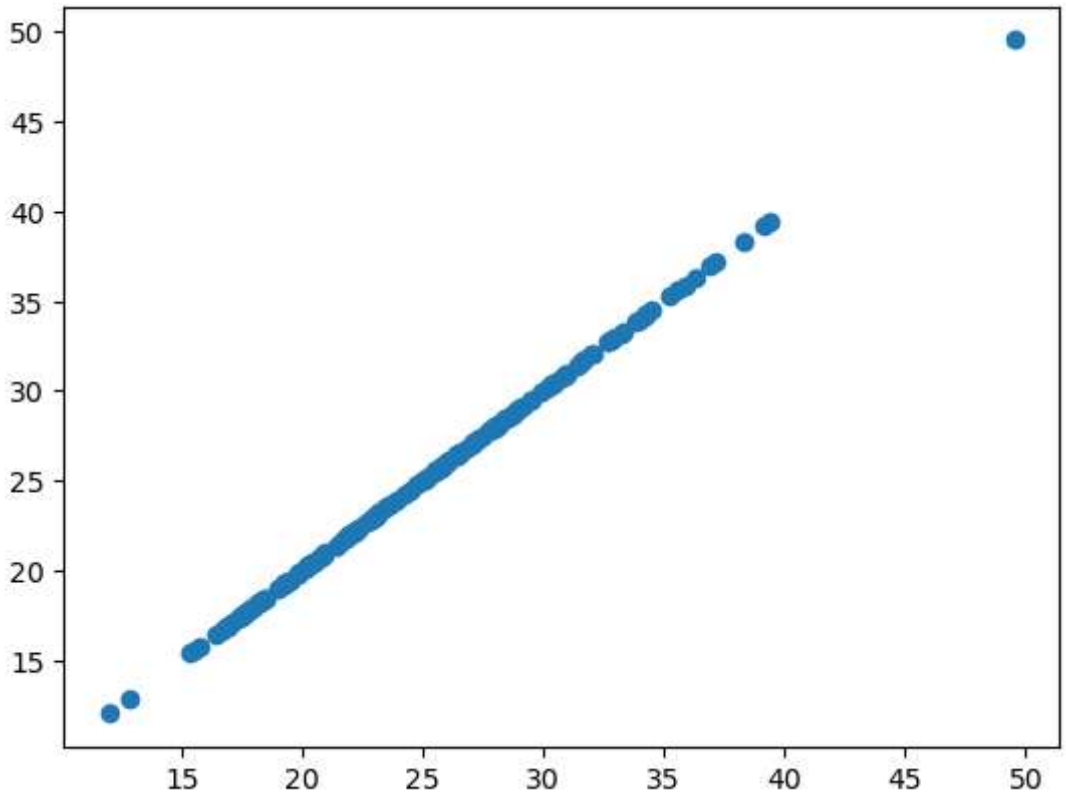
In [20]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[20]:

	Co-efficient
id	2.201906e-17
radius_mean	-8.756362e-08
texture_mean	1.212056e-09
perimeter_mean	1.352952e-08
area_mean	-1.492237e-10
smoothness_mean	-1.107278e-07
compactness_mean	-2.412592e-07
concavity_mean	-1.365733e-08
concave points_mean	2.251890e-07
symmetry_mean	-9.095843e-07
fractal_dimension_mean	2.264502e-07
radius_se	1.569921e-07
texture_se	-9.268980e-10
perimeter_se	-3.156758e-08
area_se	-3.680535e-11
smoothness_se	4.367222e-08
compactness_se	5.591750e-07
concavity_se	1.028080e-06
concave points_se	-2.970168e-09
symmetry_se	1.604074e-08
fractal_dimension_se	-5.090723e-08
radius_worst	-2.104681e-08
texture_worst	1.000000e+00
perimeter_worst	3.495093e-09
area_worst	5.743664e-11
smoothness_worst	-3.323748e-07
compactness_worst	-3.978459e-07
concavity_worst	-9.464521e-08
concave points_worst	5.077955e-07
symmetry_worst	7.018295e-07
fractal_dimension_worst	-7.592206e-08

In [21]:
```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[21]:
```
<matplotlib.collections.PathCollection at 0x202aaa1f950>
```



In [22]:
```
print(lr.score(x_test,y_test))
```

```
0.9999999999999999
```

In [23]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [24]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

```
C:\Users\user\anaconda3\Lib\site-packages\sklearn\linear_model\_ridge.py:216:
LinAlgWarning: Ill-conditioned matrix (rcond=1.39846e-18): result may not be
accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

Out[24]:
```
Ridge
Ridge(alpha=10)
```

In [25]:
```
rr.score(x_test,y_test)
```

Out[25]:
```
0.9999906754449169
```

In [26]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[26]:
```
Lasso(alpha=10)
```

In [27]:
```
la.score(x_test,y_test)
```

Out[27]:
```
0.9168216401120295
```

In [ ]:
```

```

D7

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df=pd.read_csv(r"C:\Users\user\Downloads\11_winequality-red.csv")
df
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alco
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	1
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	1
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	1
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	1
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	1

1599 rows × 12 columns

In [3]:
```
df.head(10)
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	9.4
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	10.0
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	9.5
9	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	10.5

In [6]:
```
df.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 [7]:
```
df.describe()
```

Out[7]:

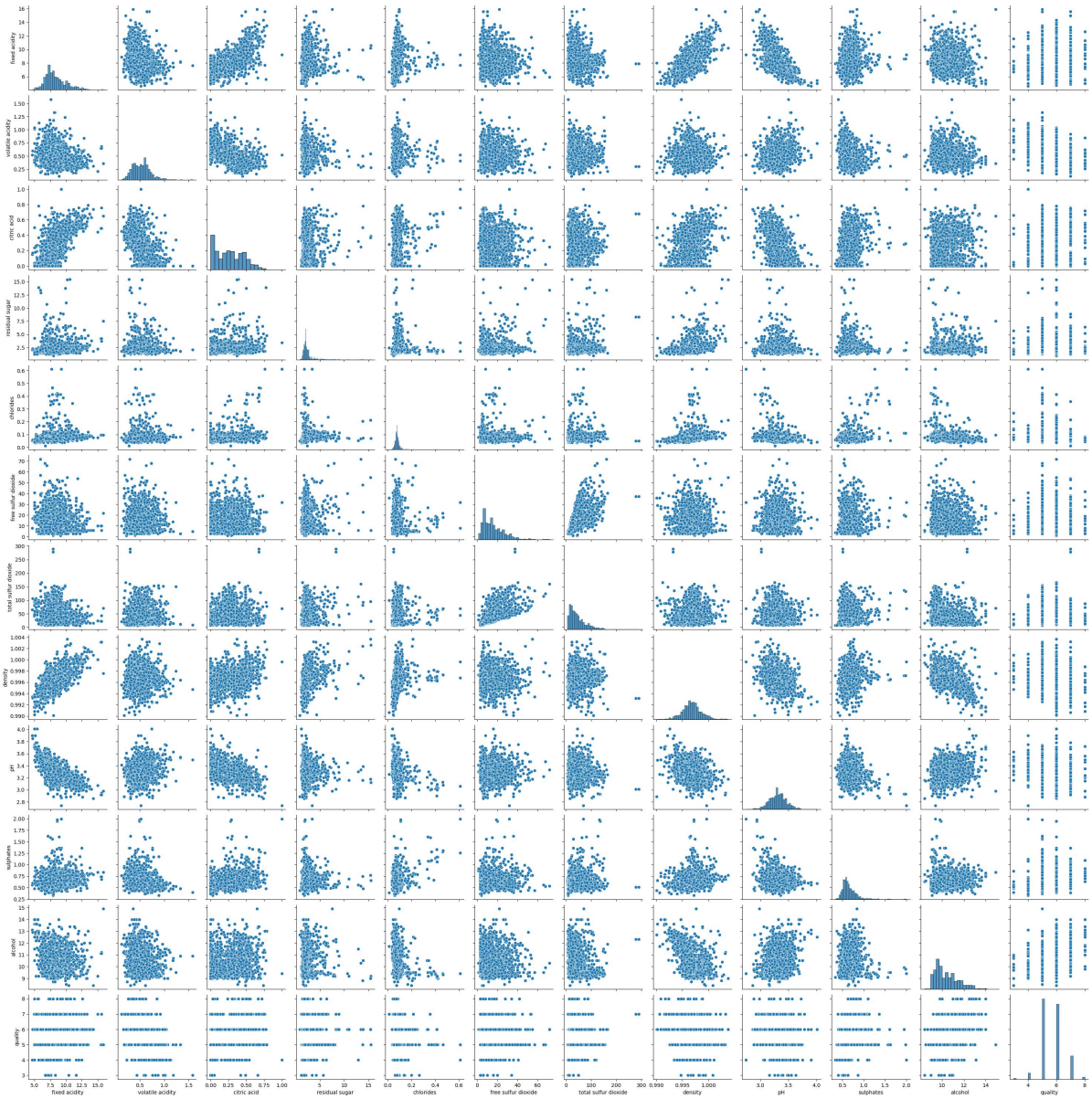
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulf dioxi
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.0000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.4677
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.8953
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.0000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.0000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.0000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.0000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.0000

In [4]:
```
df.columns
```

Out[4]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [5]:
```
sns.pairplot(df)
```

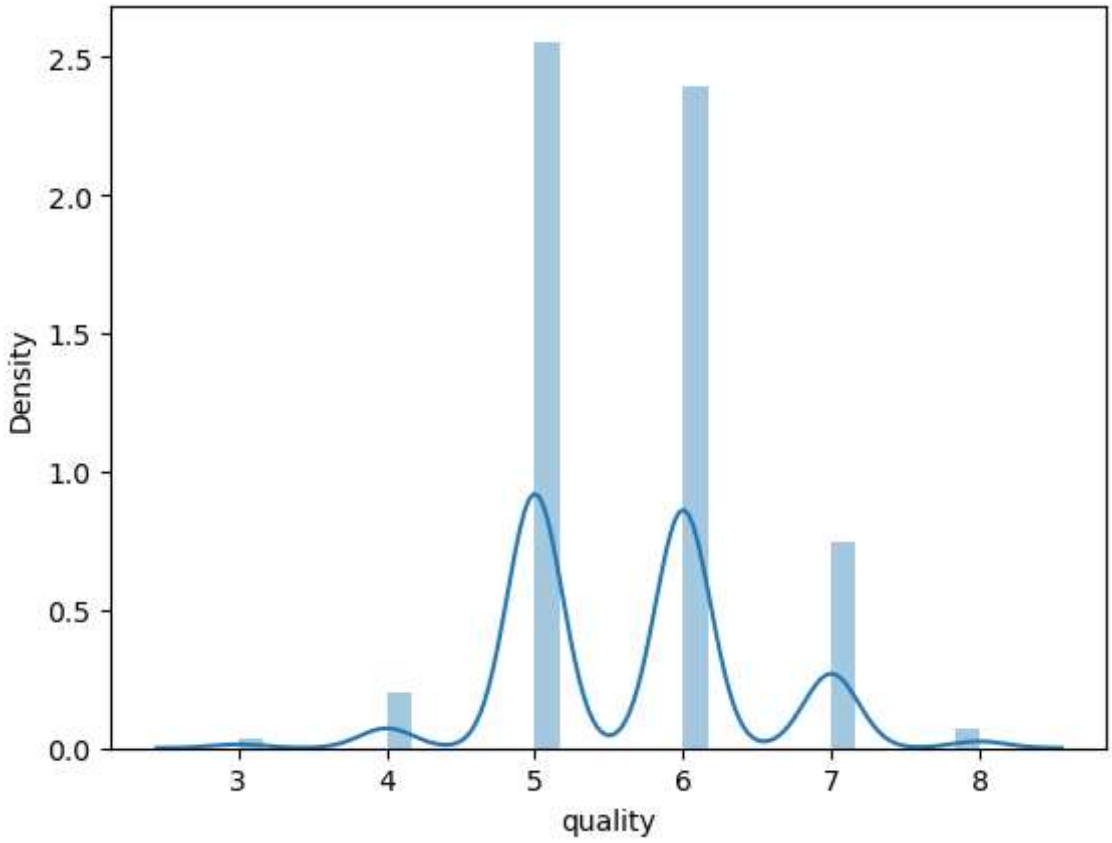
Out[5]:
```
<seaborn.axisgrid.PairGrid at 0x2489f53b6d0>
```



In [6]:
```
sns.distplot(df['quality'])
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_4092\3043670886.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(df['quality'])
```

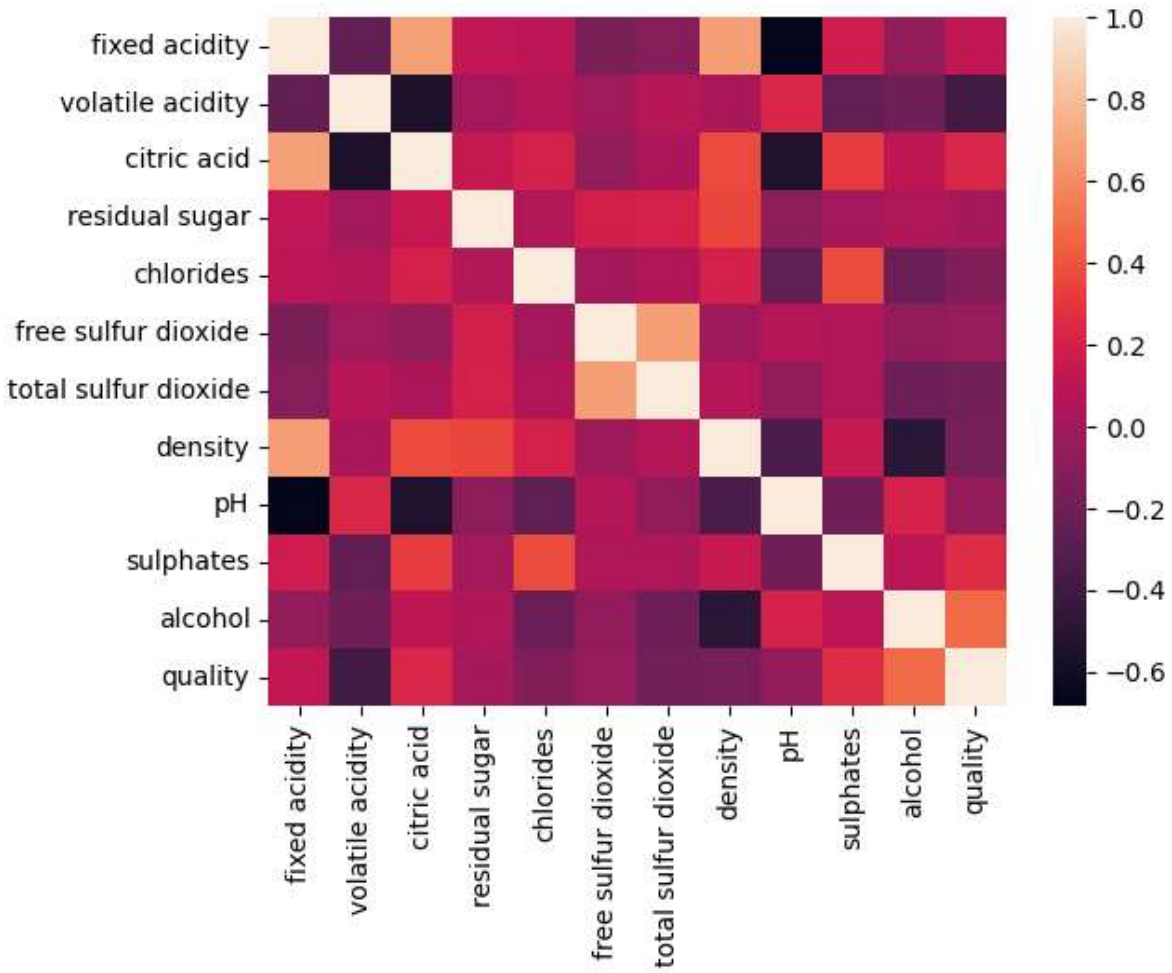
Out[6]: <Axes: xlabel='quality', ylabel='Density'>



In [7]:
```
df1=df[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality']]
```

In [8]:
```
sns.heatmap(df1.corr())
```

Out[8]: <Axes: >



In [10]:
```
x=df1[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol']]
y=df1['quality']
```

In [11]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [12]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[12]:
```
▾ LinearRegression
LinearRegression()
```

In [13]:
```
print(lr.intercept_)
```

```
1.8371596870274778
```

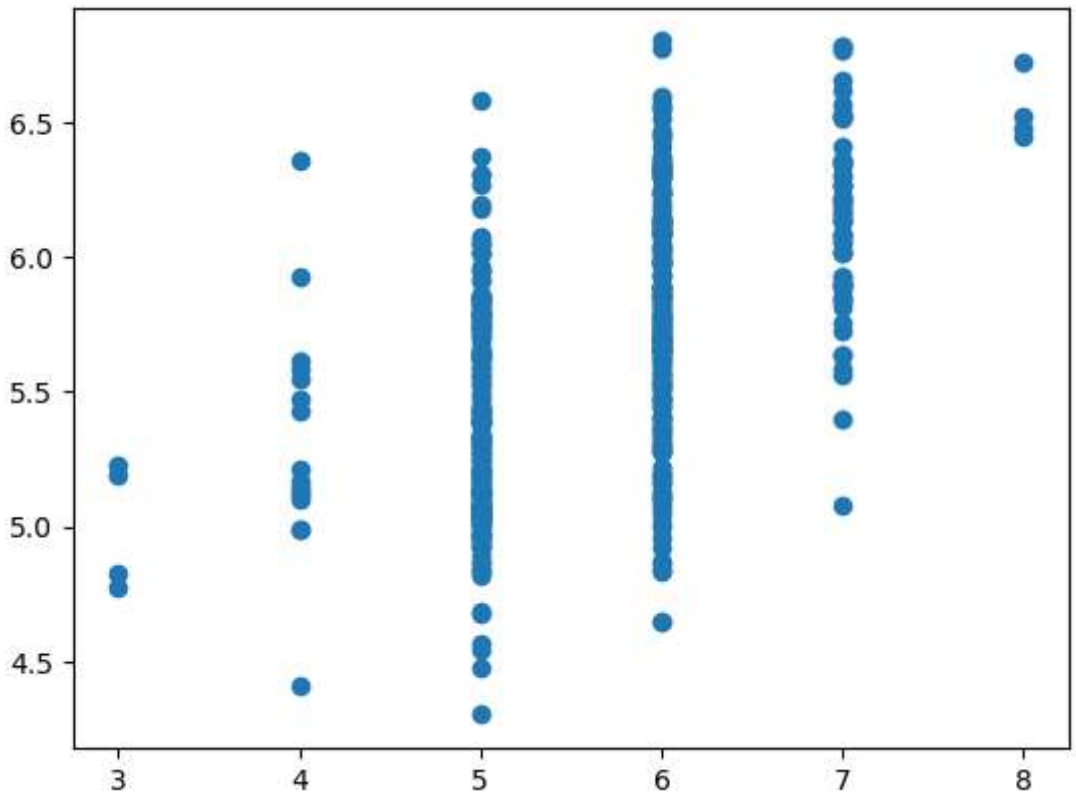
In [14]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[14]:

	Co-efficient
fixed acidity	-0.003015
volatile acidity	-1.193810
citric acid	-0.303974
residual sugar	0.015383
chlorides	-1.856641
free sulfur dioxide	0.004369
total sulfur dioxide	-0.004300
density	3.309743
pH	-0.651472
sulphates	0.939148
alcohol	0.290871

In [15]:
```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[15]:
```
<matplotlib.collections.PathCollection at 0x248b09268d0>
```



In [16]:
```
print(lr.score(x_test,y_test))
```

```
0.30762818465530084
```

In [17]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [18]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[18]:
```
Ridge(alpha=10)
```

In [19]:
```
rr.score(x_test,y_test)
```

Out[19]:
```
0.3098447423349697
```

In [20]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[20]:
```
Lasso(alpha=10)
```

In [21]:
```
la.score(x_test,y_test)
```

Out[21]:
```
-2.2756548888835937e-05
```

In [ ]:

D8

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df=pd.read_csv(r"C:\Users\user\Downloads\12_mobile_prices_2023.csv")
df
```

Out[2]:

	Phone Name	Rating ?/5	Number of Ratings	RAM	ROM/Storage	Back/Rare Camera	Front Camera	Battery	Processor	P
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...	
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	₹
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	
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	
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	
...	...	...	...	...	...	...	...	...	...	
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	MediaTek Helio G70 Processor	₹
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	MediaTek Helio G70 Processor	₹
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	MediaTek Helio G70 Processor	₹
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	MediaTek Helio G90T Processor	₹
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) Processor	₹

1836 rows × 11 columns

In [3]:
```
df.head(10)
```

Out[3]:

	Phone Name	Rating ?/5	Number of Ratings	RAM	ROM/Storage	Back/Rare Camera	Front Camera	Battery	Processor	Price in INR
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...	₹5,649
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	₹11,999
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	₹6,999
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	₹7,749
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	₹6,999
5	POCO M4 5G (Power Black, 64 GB)	4.2	77,128	4 GB RAM	64 GB ROM	50MP + 2MP	8MP Front Camera	5000 mAh	Mediatek Dimensity 700 Processor	₹11,999
6	POCO C55 (Power Black, 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	₹7,749
7	POCO C55 (Forest Green, 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	₹7,749
8	POCO C55 (Cool Blue, 128 GB)	4.1	13,647	6 GB RAM	128 GB ROM	50MP Dual Rear Camera	5MP Front Camera	5000 mAh	Mediatek Helio G85 Processor	₹9,249
9	POCO M4 5G (Yellow, 128 GB)	4.2	40,525	6 GB RAM	128 GB ROM	50MP + 2MP	8MP Front Camera	5000 mAh	Mediatek Dimensity 700 Processor	₹13,999

In [4]:
```
df.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 [5]:
```
dff=df.dropna()
```

In [6]:
```
dff.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1291 entries, 0 to 1835
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Phone Name         1291 non-null   object 
 1   Rating ?/5         1291 non-null   float64
 2   Number of Ratings  1291 non-null   object 
 3   RAM                1291 non-null   object 
 4   ROM/Storage        1291 non-null   object 
 5   Back/Rare Camera   1291 non-null   object 
 6   Front Camera       1291 non-null   object 
 7   Battery            1291 non-null   object 
 8   Processor          1291 non-null   object 
 9   Price in INR       1291 non-null   object 
 10  Date of Scraping   1291 non-null   object 
dtypes: float64(1), object(10)
memory usage: 121.0+ KB
```

In [7]:
```
dff.describe()
```

Out[7]:

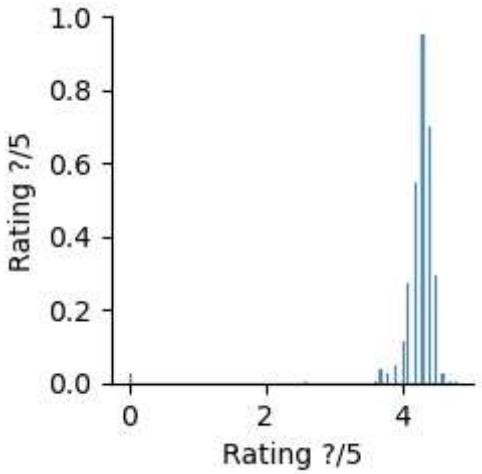
	Rating ?/5
count	1291.000000
mean	4.241208
std	0.427166
min	0.000000
25%	4.200000
50%	4.300000
75%	4.400000
max	4.800000

In [6]:
```
dff.columns
```

Out[6]:
```
Index(['Phone Name', 'Rating ?/5', 'Number of Ratings', 'RAM', 'ROM/Storage',
       'Back/Rare Camera', 'Front Camera', 'Battery', 'Processor',
       'Price in INR', 'Date of Scraping'],
      dtype='object')
```

In [7]:
```
sns.pairplot(dff)
```

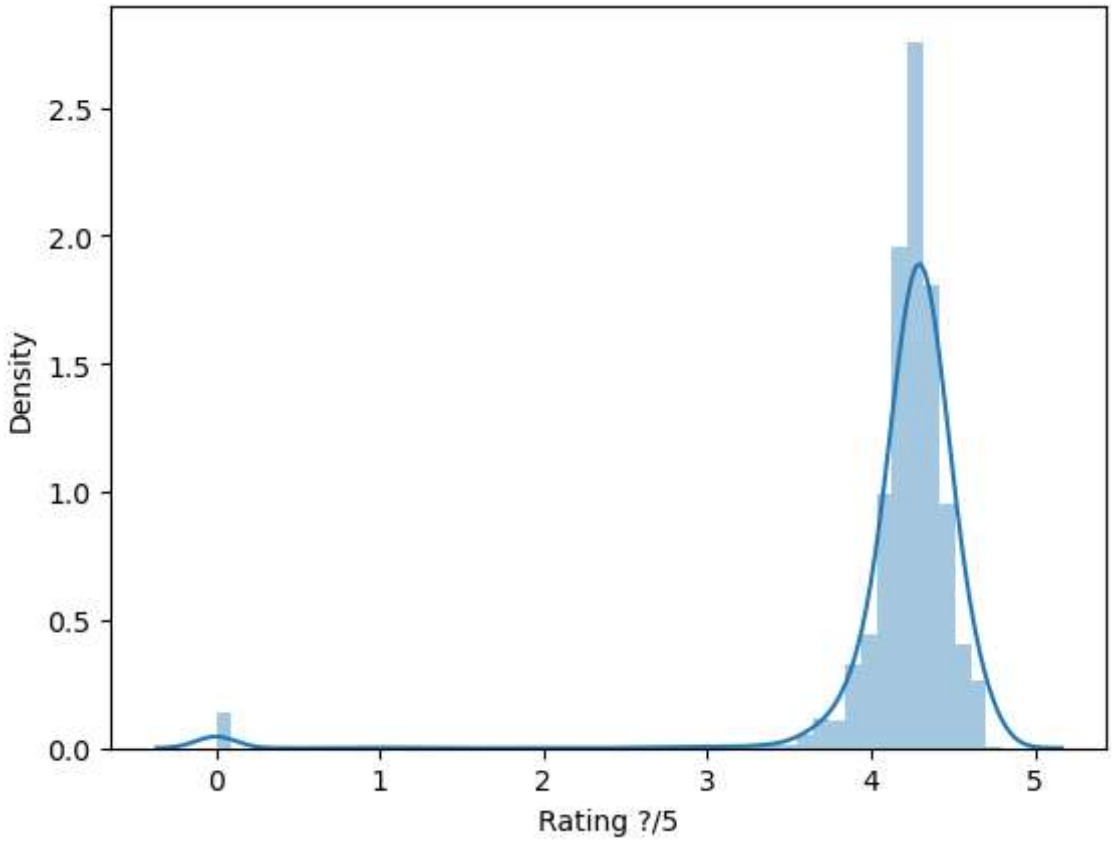
Out[7]:
```
<seaborn.axisgrid.PairGrid at 0x29d331be6d0>
```



In [8]:
```
sns.distplot(df["Rating ?/5"])
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_5412\1803240722.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(df["Rating ?/5"])
```

Out[8]: <Axes: xlabel='Rating ?/5', ylabel='Density'>



In [9]:
```
df1=df[['Phone Name', 'Rating ?/5', 'Number of Ratings', 'RAM', 'ROM/Storage',
       'Back/Rare Camera', 'Front Camera', 'Battery', 'Processor',
       'Price in INR', 'Date of Scraping']]
```

In [10]:
```
sns.heatmap(df1.corr())
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_5412\781785195.py:1: FutureWarnin
g: The default value of numeric_only in DataFrame.corr is deprecated. In a fu
ture version, it will default to False. Select only valid columns or specify
the value of numeric_only to silence this warning.
  sns.heatmap(df1.corr())
```

Out[10]: <Axes: >

In [15]:
```
x=df1[['Rating ?/5']]
y=df1['Rating ?/5']
```

In [16]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [17]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[17]:
```
▾ LinearRegression

LinearRegression()
```

In [18]:
```
print(lr.intercept_)
```

```
-1.7763568394002505e-15
```

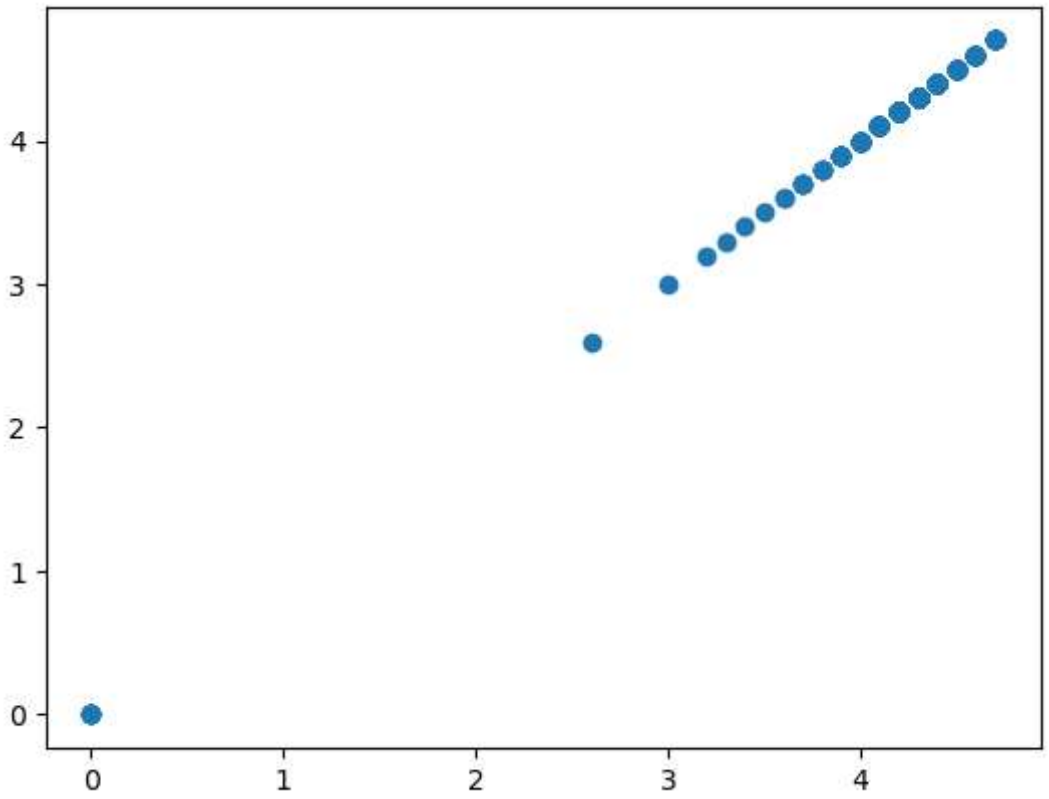
In [19]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[19]:

	Co-efficient
Rating ?/5	1.0

In [20]:
```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[20]: <matplotlib.collections.PathCollection at 0x29d3abdcc90>



In [21]:
```
print(lr.score(x_test,y_test))
```

```
1.0
```

In [22]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [23]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[23]:

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

In [24]:
```
rr.score(x_test,y_test)
```

Out[24]:
```
0.9994242916253057
```

In [25]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[25]:
```
Lasso(alpha=10)
```

In [26]:
```
la.score(x_test,y_test)
```

Out[26]:
```
-0.0024746499792245302
```

In [ ]: