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


from sklearn.linear_model import ElasticNet
from sklearn import metrics


df=pd.read_csv("/content/1_fiat500_VehicleSelection_Dataset - 1_fiat500_VehicleSelection_Dataset (1).csv")
df
```

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon	price	Unnamed: 9	Unnamed: 10
0	1.0	lounge	51.0	882.0	25000.0	1.0	44.907242	8.611559868	8900	NaN	NaN
1	2.0	pop	51.0	1186.0	32500.0	1.0	45.666359	12.24188995	8800	NaN	NaN
2	3.0	sport	74.0	4658.0	142228.0	1.0	45.503300	11.41784	4200	NaN	NaN
3	4.0	lounge	51.0	2739.0	160000.0	1.0	40.633171	17.63460922	6000	NaN	NaN
4	5.0	pop	73.0	3074.0	106880.0	1.0	41.903221	12.49565029	5700	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...
1544	NaN	NaN	NaN	NaN	NaN	NaN	NaN	length	5	NaN	NaN
1545	NaN	NaN	NaN	NaN	NaN	NaN	NaN	concat	lonprice	NaN	NaN
1546	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Null values	NO	NaN	NaN
1547	NaN	NaN	NaN	NaN	NaN	NaN	NaN	find	1	NaN	NaN
1548	NaN	NaN	NaN	NaN	NaN	NaN	NaN	search	1	NaN	NaN

1549 rows × 11 columns

```
df1=df.drop(df.index[1537:],axis=0)
df1=df1.drop(["Unnamed: 9","Unnamed: 10","model"],axis=1)
df1
```

	ID	engine_power	age_in_days	km	previous_owners	lat	lon	price
0	1.0	51.0	882.0	25000.0	1.0	44.907242	8.611559868	8900
1	2.0	51.0	1186.0	32500.0	1.0	45.666359	12.24188995	8800
2	3.0	74.0	4658.0	142228.0	1.0	45.503300	11.41784	4200
3	4.0	51.0	2739.0	160000.0	1.0	40.633171	17.63460922	6000
4	5.0	73.0	3074.0	106880.0	1.0	41.903221	12.49565029	5700
...	...	...	...	...	...	...	...	...
1532	1533.0	51.0	1917.0	52008.0	1.0	45.548000	11.54946995	9900
1533	1534.0	51.0	3712.0	115280.0	1.0	45.069679	7.704919815	5200
1534	1535.0	74.0	3835.0	112000.0	1.0	45.845692	8.666870117	4600
1535	1536.0	51.0	2223.0	60457.0	1.0	45.481541	9.413479805	7500
1536	1537.0	51.0	2557.0	80750.0	1.0	45.000702	7.68227005	5990

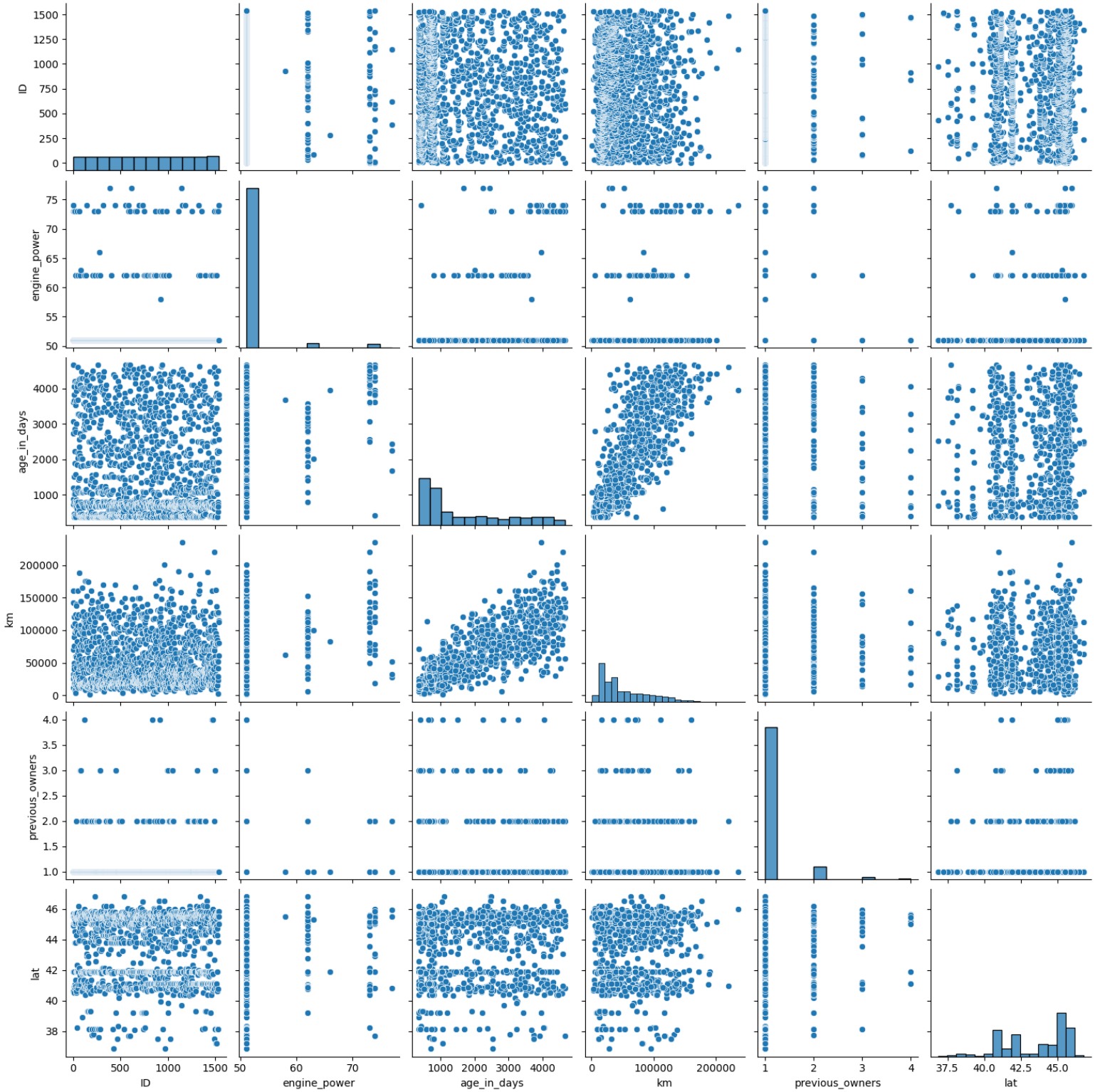
1537 rows × 8 columns

```
df1.describe()
```

	ID	engine_power	age_in_days	km	previous_owners	lat
count	1537.000000	1537.000000	1537.000000	1537.000000	1537.000000	1537.000000
mean	769.000000	51.905010	1650.905660	53395.439167	1.123617	43.543455
std	443.837996	3.989254	1289.938635	40059.858383	0.416546	2.132631
min	1.000000	51.000000	366.000000	1232.000000	1.000000	36.855839
25%	385.000000	51.000000	670.000000	20000.000000	1.000000	41.802990
50%	769.000000	51.000000	1035.000000	39024.000000	1.000000	44.399971
75%	1153.000000	51.000000	2616.000000	79800.000000	1.000000	45.467960
max	1537.000000	77.000000	4658.000000	235000.000000	4.000000	46.795612

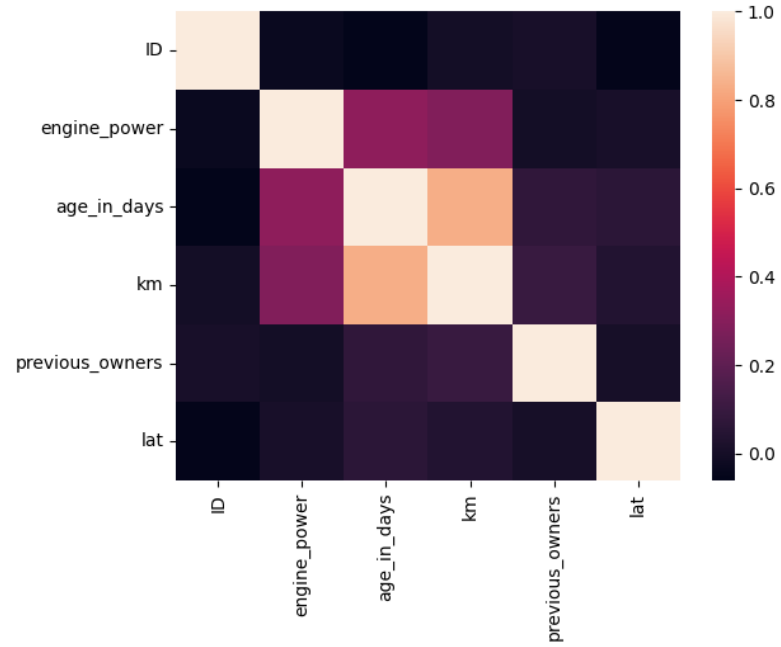
```
sns.pairplot(df1)
```

```
<seaborn.axisgrid.PairGrid at 0x7e498809ea40>
```




```
sns.heatmap(df1.corr())
```

```
<ipython-input-28-3ed1a1a51dc0>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to Fa
  sns.heatmap(df1.corr())
<Axes: >
```




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


y=df1['age_in_days']
x=df1.drop(['age_in_days'],axis=1)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)


model=LinearRegression()
model.fit(x_train,y_train)
model.intercept_
```

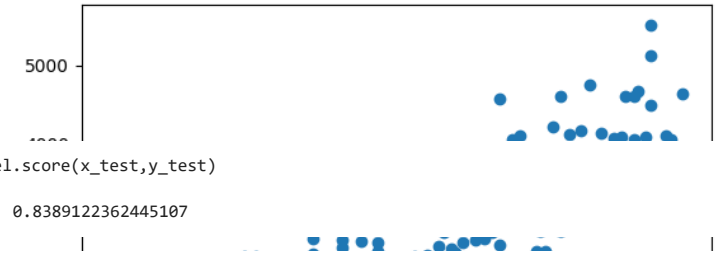
```
3504.2846623018127
```

```
coeff=pd.DataFrame(model.coef_,x.columns,columns=["Coefficient"])
coeff
```

	Coefficient
ID	-0.106036
engine_power	17.402916
km	0.008147
previous_owners	18.257263
lat	17.047097
lon	-13.417091
price	-0.434285

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

```
<matplotlib.collections.PathCollection at 0x7e49d8816ef0>
```




```
model.score(x_test,y_test)
```

```
0.8389122362445107
```



```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
la=Lasso(alpha=10)
la.fit(x_train,y_train)
print(rr.score(x_test,y_test))
la.score(x_test,y_test)
```

```
0.838919765618213
0.8389692135224461
```


```
en=ElasticNet()
en.fit(x_train,y_train)
print(en.coef_)
print(en.intercept_)
print(en.predict(x_test))
print(en.score(x_test,y_test))
```

```
 590.74464813 1018.20022876  730.90148089  910.90022306 1107.99276389
1008.9435207  2081.70834728 3669.93890399  763.45718089  445.12061642
 559.85329026  859.36659278 1541.30339779 2544.68345341 2140.52723213
1354.69204746  363.02422717  572.68800477 2889.66781704  825.27461175
3245.80018085 1926.46970932  847.04076335  324.40274987 1905.34788401
2451.35434477  376.67271407 2001.15592221  504.68887908 2477.49927327
1229.67897296  326.40408748  543.24595385 4628.65364198 4036.73132977
3459.54639087 1840.07918973  584.15439324 1569.66346813  828.36485809
 888.5487282   950.08892726 2098.84500097 1358.78102675  385.39692979
 767.21050275  860.05549171 2948.55480173 2398.56207042  821.8710294
1241.1316723   436.89380771  763.39949348 4035.00417542 2090.54258095
 529.29075286 1291.22669551  591.65765116 3776.42645454  885.21254674
3341.7613384   793.58170336 3063.09846659 2049.20416714  465.71676819
 410.44837122  626.98311086 1067.55344279 2404.35227522  628.62622554
 837.8983847  2092.9023177   545.99599849 3294.47824639 1840.26690181
 850.38599215  748.5286482  1191.53602411   29.15989126 1240.75329363
1023.68682668 1603.93507237  963.2281056  1027.12318796  805.22575964
 399.1560632  3654.72737611 1268.31738298 2488.89696374 2477.62246048
1618.67664358  583.88982542  819.07045885 1316.42236306  358.1827425
 974.01139209 4597.27369058  845.53775217  392.54380535 3514.95059829
2762.57427032  982.02651664  651.63078745 2617.87552292 3688.77442566
 859.26153312 1352.22006447 1391.04393845  394.00414389 2287.15159827
 735.60873689 1771.17192378  598.84592649 1660.99127989  645.97569703
3262.58989756 3498.2676676  1713.43909306 2294.21588805 3011.60070238
4485.32624104 4087.58771104 1270.14510681 2724.7589579  2363.39299276
1336.33242987 1737.13888026  385.78461683  429.64642011  848.586251
 818.03112265  435.31947057 1204.12049283  805.69098656 2850.09415083
 465.3906151   446.16781309  326.55011372  792.20295899 2443.38897653
 498.08719078 1178.70918683 1264.37822848 4075.1129199  2189.06985838
 907.52598445 2130.86348782 1738.82360977 2312.29190737 1889.07525021
1706.02473528  465.01731677 1919.82344202 2191.28339649 3029.70293077
 629.58557267  836.30379169 1630.62847455 2733.44021489  422.56199315
 443.2619007   531.19586791 2973.27024398 1305.40750247 1030.68665365
4145.43631215  627.5449088  2669.67677632 3175.12568811 2500.14340041
 426.04231035  233.36667577 3118.85146587  402.09458822 3912.13214794
 841.51981647 1611.31601238  505.00848064 1010.11498897 2879.7375541
 845.50988768 2347.03820016 2902.64357897 2416.29965481 2050.91450742
 542.51140334  854.4444905  1125.84713003 2101.09411533  272.62211561
4026.20435578 2921.95488412 3885.57165534 1620.05466021 3640.30126326
 164.59666812 3169.17588     580.69649978 1010.00763824 1673.90421754
2501.99076259  518.0590981   781.20560248 1507.84473807 1668.07254755
2338.57126772 3567.43005838  622.67632481 3878.57450336  572.42951018
1724.38198257  294.10449358  589.42305509  994.24835701 1704.85649565
2365.02648207  762.5571244   513.82932114 1014.00005256  841.53044733
 627.84893602  363.38363133  374.26052101  798.25902905  338.59918568
 522.61917675 1550.87950488 1204.6746501  1110.82453746 4139.36706075
 209.91644329 2176.45513931  597.14035055 4661.28994667 2539.16270977
1916.62305615  769.4950909  2042.9465289   618.33174901 2185.43078644
1902.22450741  212.18068512  417.32378987 1214.77379052  386.8809477
2912.39126094 2036.82700115  812.02857078 3371.42910438 2495.64505842
 902.54959878  919.29084181  415.67609297  769.35958274 2224.47859409
1251.70259067 2529.45680248 3181.24430373  235.67380625 4528.44533211
 503.8347572   351.94779442 2328.09630031 1242.18205743  666.19095714
3650.02393018 1468.84689703  403.50773794 4220.35619423 1292.85923858
 779.47264741  960.15846313 2599.22964932 1935.89460715  898.67141929
 658.51126325  543.14395797 1869.11149329 1211.44539048 3589.5384973
5114.86040689  481.56672085  949.01510269 3426.82118835 2779.63635752
2864.75707951 2774.66832133 2437.68076403  453.42298717  495.33115751
 852.68961046  605.44039204 1261.17042495  628.18317377 3947.22047817
```

```
print("MAE",metrics.mean_absolute_error(y_test,prediction))
print("MSE",metrics.mean_squared_error(y_test,prediction))
print("RMSE",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
MAE 402.63429985147536
MSE 279534.92621643865
RMSE 528.710626161834
```

```
daf=pd.read_csv("/content/2_2015.csv")
daf
```

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystopia Residual
0	Switzerland	Western Europe	1	7.587	0.03411	1.39651	1.34951	0.94143	0.66557	0.41978	0.29678	2.51738
1	Iceland	Western Europe	2	7.561	0.04884	1.30232	1.40223	0.94784	0.62877	0.14145	0.43630	2.70201
2	Denmark	Western Europe	3	7.527	0.03328	1.32548	1.36058	0.87464	0.64938	0.48357	0.34139	2.49204
3	Norway	Western Europe	4	7.522	0.03880	1.45900	1.33095	0.88521	0.66973	0.36503	0.34699	2.46531
4	Canada	North America	5	7.427	0.03553	1.32629	1.32261	0.90563	0.63297	0.32957	0.45811	2.45176
...	...	...	...	...	...	...	...	...	...	...	...	...
153	Rwanda	Sub-Saharan Africa	154	3.465	0.03464	0.22208	0.77370	0.42864	0.59201	0.55191	0.22628	0.67042
154	Benin	Sub-Saharan Africa	155	3.340	0.03656	0.28665	0.35386	0.31910	0.48450	0.08010	0.18260	1.63328
155	Syria	Middle East and Northern Africa	156	3.006	0.05015	0.66320	0.47489	0.72193	0.15684	0.18906	0.47179	0.32858

```
daf1=daf.drop(["Country","Region"],axis=1)
daf1.isna().sum()
```

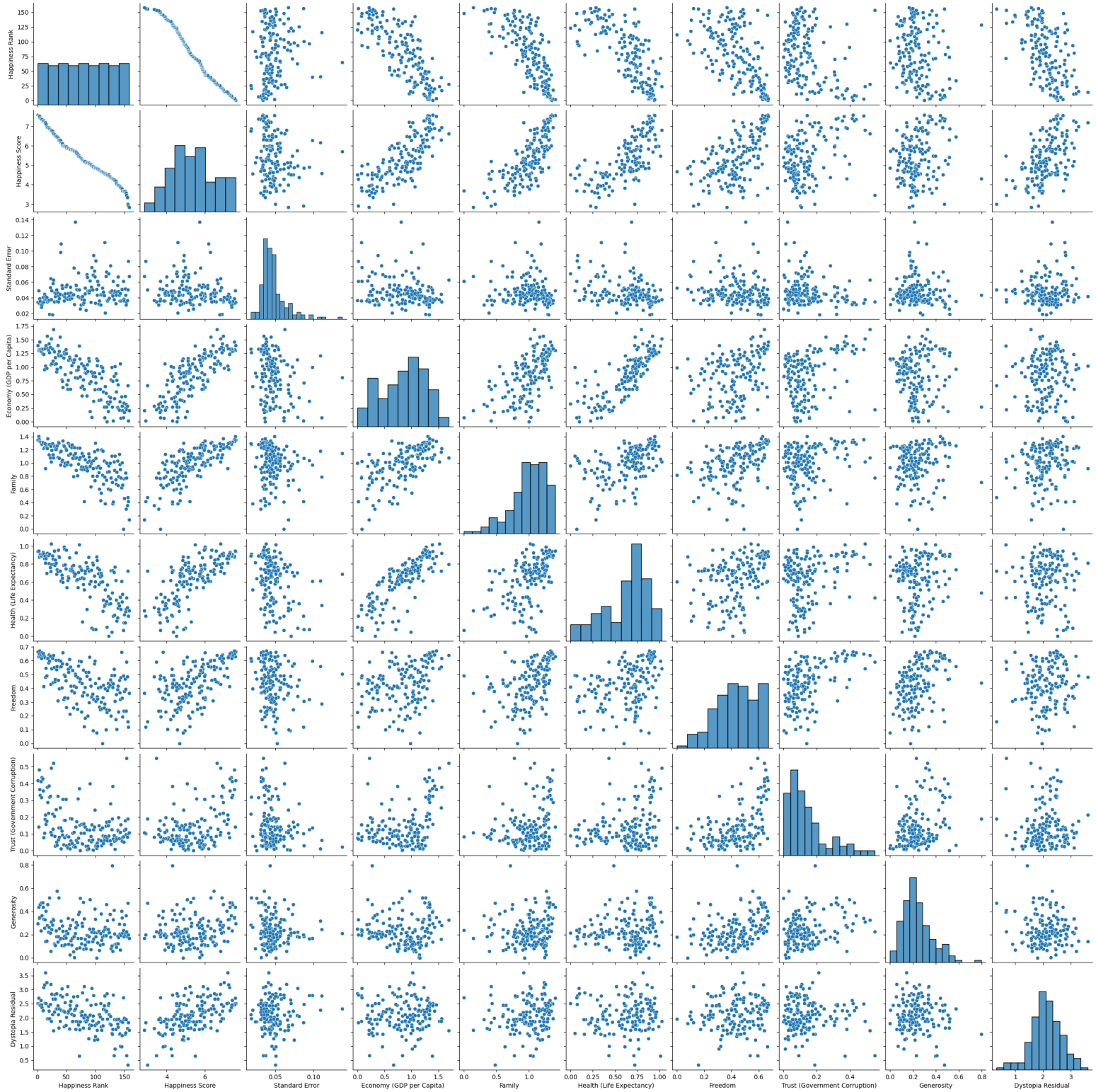
```
Happiness Rank                  0
Happiness Score                 0
Standard Error                  0
Economy (GDP per Capita)        0
Family                          0
Health (Life Expectancy)        0
Freedom                         0
Trust (Government Corruption)   0
Generosity                      0
Dystopia Residual               0
dtype: int64
```

```
daf1.describe()
```

	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystopia Residual
count	158.000000	158.000000	158.000000	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	0.143422	0.237296	2.098977
std	45.754363	1.145010	0.017146	0.403121	0.272369	0.247078	0.150693	0.120034	0.126685	0.553550
min	1.000000	2.839000	0.018480	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.328580
25%	40.250000	4.526000	0.037268	0.545808	0.856823	0.439185	0.328330	0.061675	0.150553	1.759410
50%	79.500000	5.232500	0.043940	0.910245	1.029510	0.696705	0.435515	0.107220	0.216130	2.095415
75%	118.750000	6.243750	0.052300	1.158448	1.214405	0.811013	0.549092	0.180255	0.309883	2.462415
max	158.000000	7.587000	0.136930	1.690420	1.402230	1.025250	0.669730	0.551910	0.795880	3.602140

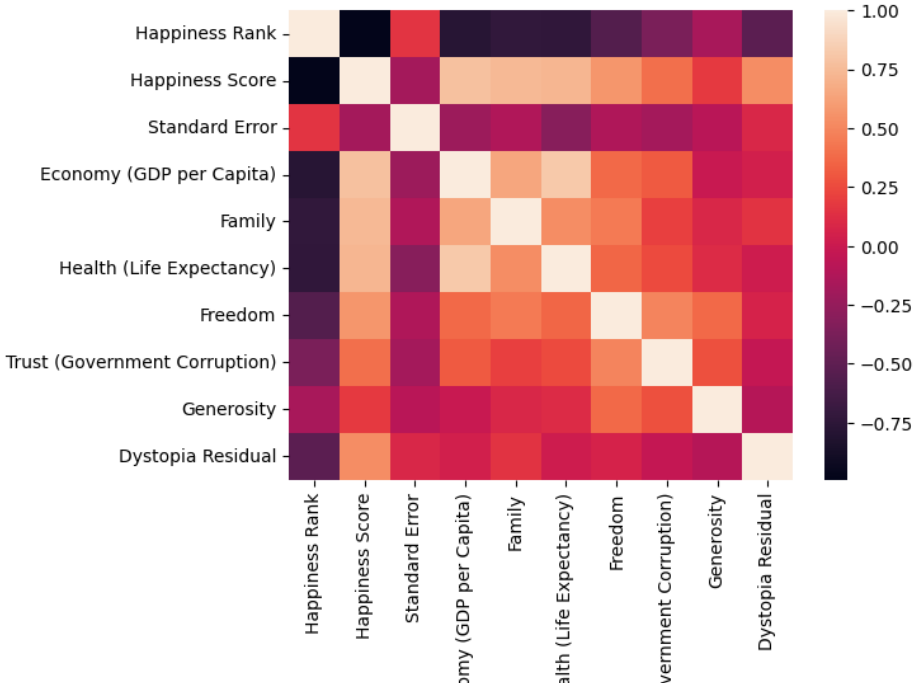
```
sns.pairplot(daf1)
```

<seaborn.axisgrid.PairGrid at 0x7e49d8867820>




```
sns.heatmap(daf1.corr())
```

<Axes: >




```
y=daf1['Standard Error']
x=daf1.drop(['Standard Error'],axis=1)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)



model=LinearRegression()

model.fit(x_train,y_train)

model.intercept_
```

```
0.20869511421123185
```

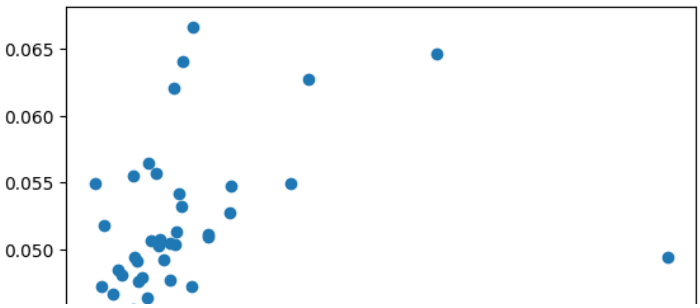
```
coeff=pd.DataFrame(model.coef_,x.columns,columns=["Coefficient"])
coeff
```

	Coefficient
Happiness Rank	-0.000506
Happiness Score	-2.910609
Economy (GDP per Capita)	2.900859
Family	2.894359
Health (Life Expectancy)	2.855783
Freedom	2.884681
Trust (Government Corruption)	2.871231
Generosity	2.888274
Dystopia Residual	2.892031

```
prediction=model.predict(x_test)

plt.scatter(y_test,prediction)
```

```
<matplotlib.collections.PathCollection at 0x7e4980a7d690>
```




```
model.score(x_test,y_test)
```

```
0.05308012173145704
```

```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
la=Lasso(alpha=10)
la.fit(x_train,y_train)
print(rr.score(x_test,y_test))
la.score(x_test,y_test)
```

```
0.06493881163854776
-0.00047954236123848304
```

```
en=ElasticNet()
en.fit(x_train,y_train)
print(en.coef_)
print(en.intercept_)
print(en.predict(x_test))
print(en.score(x_test,y_test))
```

```
[ 0. -0. -0. -0. -0. -0. -0. -0.  0.]
0.04800163636363638
[0.04800164 0.04800164 0.04800164 0.04800164 0.04800164 0.04800164
 0.04800164 0.04800164 0.04800164 0.04800164 0.04800164 0.04800164
 0.04800164 0.04800164 0.04800164 0.04800164 0.04800164 0.04800164
 0.04800164 0.04800164 0.04800164 0.04800164 0.04800164 0.04800164
 0.04800164 0.04800164 0.04800164 0.04800164 0.04800164 0.04800164
 0.04800164 0.04800164 0.04800164 0.04800164 0.04800164 0.04800164
 0.04800164 0.04800164 0.04800164 0.04800164 0.04800164 0.04800164
 0.04800164 0.04800164 0.04800164 0.04800164 0.04800164 0.04800164]
-0.00047954236123848304
```

```
print("MAE",metrics.mean_absolute_error(y_test,prediction))
print("MSE",metrics.mean_squared_error(y_test,prediction))
print("RMSE",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
MAE 0.011162596731313023
MSE 0.0002923271595678132
RMSE 0.017097577593560242
```

```
df=pd.read_csv("/content/3_Fitness-1.csv")
df
```

	Row Labels	Sum of Jan	Sum of Feb	Sum of Mar	Sum of Total Sales
0	A	5.62%	7.73%	6.16%	75
1	B	4.21%	17.27%	19.21%	160
2	C	9.83%	11.60%	5.17%	101
3	D	2.81%	21.91%	7.88%	127
4	E	25.28%	10.57%	11.82%	179
5	F	8.15%	16.24%	18.47%	167
6	G	18.54%	8.76%	17.49%	171
7	H	25.56%	5.93%	13.79%	170
8	Grand Total	100.00%	100.00%	100.00%	1150

```
df=df.drop(['Row Labels'],axis=1)
```

```
df["Sum of Jan"]=df["Sum of Jan"].replace("%","",regex=True).astype(float)
df["Sum of Feb"]=df["Sum of Feb"].replace("%","",regex=True).astype(float)
df["Sum of Mar"]=df["Sum of Mar"].replace("%","",regex=True).astype(float)
```

```
df
```

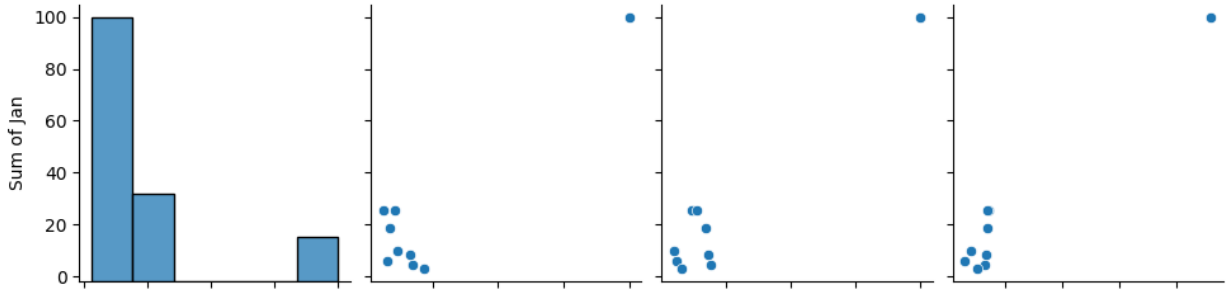
	Sum of Jan	Sum of Feb	Sum of Mar	Sum of Total Sales
0	5.62	7.73	6.16	75
1	4.21	17.27	19.21	160
2	9.83	11.60	5.17	101
3	2.81	21.91	7.88	127
4	25.28	10.57	11.82	179
5	8.15	16.24	18.47	167
6	18.54	8.76	17.49	171
7	25.56	5.93	13.79	170
8	100.00	100.00	100.00	1150

```
df.describe()
```

	Sum of Jan	Sum of Feb	Sum of Mar	Sum of Total Sales
count	9.000000	9.000000	9.000000	9.000000
mean	22.222222	22.223333	22.221111	255.555556
std	30.438329	29.612265	29.640999	337.332963
min	2.810000	5.930000	5.170000	75.000000
25%	5.620000	8.760000	7.880000	127.000000
50%	9.830000	11.600000	13.790000	167.000000
75%	25.280000	17.270000	18.470000	171.000000
max	100.000000	100.000000	100.000000	1150.000000

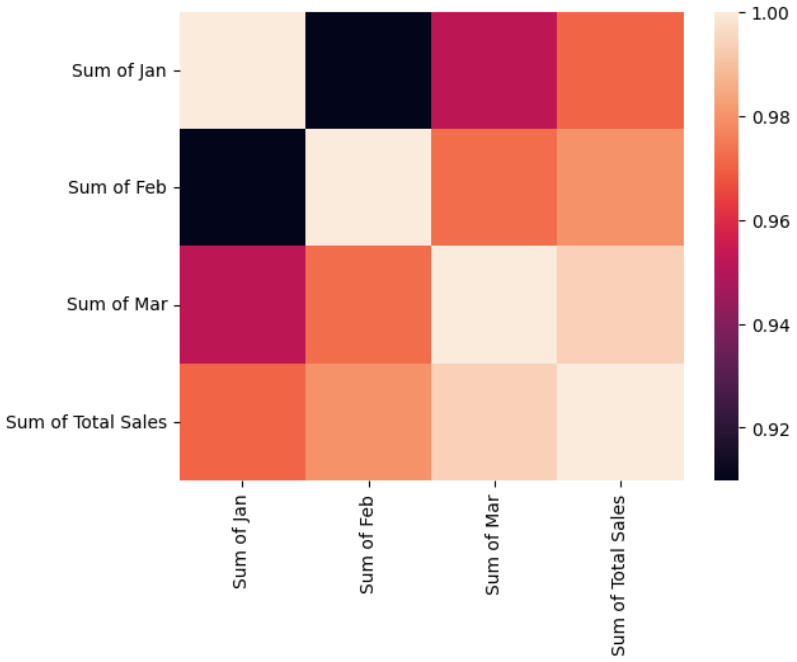
```
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x7e4980add150>
```




```
sns.heatmap(df.corr())
```

```
<Axes: >
```




```
y=df['Sum of Feb']
x=df.drop(['Sum of Feb'],axis=1)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

```
model=LinearRegression()
model.fit(x_train,y_train)
model.intercept_
```

```
0.000884519407065909
```

```
coeff=pd.DataFrame(model.coef_,x.columns,columns=["Coefficient"])

coeff
```

	Coefficient
Sum of Jan	-0.917801
Sum of Mar	-1.045991
Sum of Total Sales	0.257720

```
prediction=model.predict(x_test)

plt.scatter(y_test,prediction)
```

```
<matplotlib.collections.PathCollection at 0x7e4982c3e3e0>
```


```
model.score(x_test,y_test)
```

```
0.999997913484332
```



```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
la=Lasso(alpha=10)
la.fit(x_train,y_train)
print(rr.score(x_test,y_test))
la.score(x_test,y_test)
```

```
0.9402123802432005
0.05236674613338477
```

```
en=ElasticNet()
en.fit(x_train,y_train)
print(en.coef_)
print(en.intercept_)
print(en.predict(x_test))
print(en.score(x_test,y_test))
```

```
[-0.87580947 -0.84477199  0.23630945]
0.24507651167191113
[ 9.64142316 18.139361   10.41880018]
0.9613001308069328
```

```
print("MAE",metrics.mean_absolute_error(y_test,prediction))
print("MSE",metrics.mean_squared_error(y_test,prediction))
print("RMSE",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
MAE 0.004045932284792404
MSE 2.7956110626971713e-05
RMSE 0.005287353839773892
```

```
df=pd.read_csv("/content/6_Salesworkload1.csv")
df
```

	MonthYear	Time index	Country	StoreID	City	Dept_ID	Dept. Name	HoursOwn	HoursLease	Sales units	Turnover	Customer	Area (m2)	Opening hours
0	10.2016	1.0	United Kingdom	88253.0	London (I)	1.0	Dry	3184.764	0.0	398560.0	1226244.0	NaN	953.04	Type A
1	10.2016	1.0	United Kingdom	88253.0	London (I)	2.0	Frozen	1582.941	0.0	82725.0	387810.0	NaN	720.48	Type A
2	10.2016	1.0	United Kingdom	88253.0	London (I)	3.0	other	47.205	0.0	438400.0	654657.0	NaN	966.72	Type A
3	10.2016	1.0	United Kingdom	88253.0	London (I)	4.0	Fish	1623.852	0.0	309425.0	499434.0	NaN	1053.36	Type A
4	10.2016	1.0	United Kingdom	88253.0	London (I)	5.0	Fruits & Vegetables	1759.173	0.0	165515.0	329397.0	NaN	1053.36	Type A
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7653	06.2017	9.0	Sweden	29650.0	Gothenburg	12.0	Checkout	6322.323	0.0	3886530.0	14538825.0	NaN	#NV	Type A
7654	06.2017	9.0	Sweden	29650.0	Gothenburg	16.0	Customer Services	4270.479	0.0	245.0	0.0	NaN	#NV	Type A

```
df.isna().sum()
```

```
MonthYear          0
Time index         8
```

```
Country             8
StoreID             8
City                8
Dept_ID             8
Dept. Name          8
HoursOwn            8
HoursLease          8
Sales units         8
Turnover            8
Customer         7658
Area (m2)           8
Opening hours       8
dtype: int64
```

```
df1=df.drop(["Customer","Country","Dept. Name","Opening hours","City"],axis=1)
df1=df1.dropna()
```

```
val=df["HoursOwn"]=="?"
print(df.index[val])
```

```
Int64Index([2966, 5889], dtype='int64')
```

```
val=["#NV"]
df1["Area (m2)"].isin(val).sum()
df1=df1.drop([2966,5889],axis=0)
```

```
df1=df1.drop(["Area (m2)"],axis=1)
df1
```

	MonthYear	Time index	StoreID	Dept_ID	HoursOwn	HoursLease	Sales units	Turnover
0	10.2016	1.0	88253.0	1.0	3184.764	0.0	398560.0	1226244.0
1	10.2016	1.0	88253.0	2.0	1582.941	0.0	82725.0	387810.0
2	10.2016	1.0	88253.0	3.0	47.205	0.0	438400.0	654657.0
3	10.2016	1.0	88253.0	4.0	1623.852	0.0	309425.0	499434.0
4	10.2016	1.0	88253.0	5.0	1759.173	0.0	165515.0	329397.0
...	...	...	...	...	...	...	...	...
7653	06.2017	9.0	29650.0	12.0	6322.323	0.0	3886530.0	14538825.0
7654	06.2017	9.0	29650.0	16.0	4270.479	0.0	245.0	0.0
7655	06.2017	9.0	29650.0	11.0	0	0.0	0.0	0.0
7656	06.2017	9.0	29650.0	17.0	2224.929	0.0	245.0	0.0
7657	06.2017	9.0	29650.0	18.0	39652.2	0.0	3886530.0	15056214.0

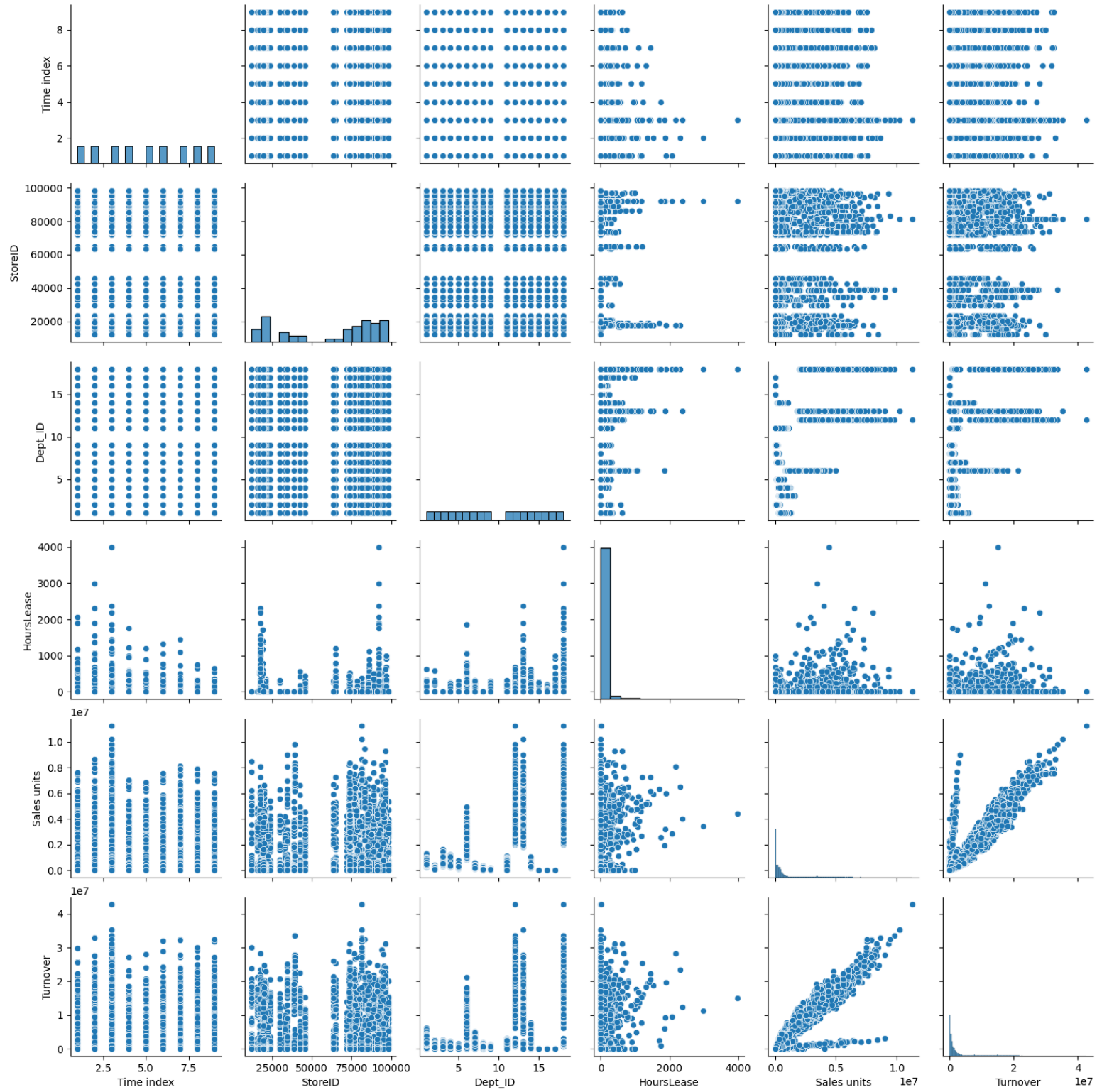
7648 rows × 8 columns

```
df1.describe()
```

	Time index	StoreID	Dept_ID	HoursLease	Sales units	Turnover
count	7648.000000	7648.000000	7648.000000	7648.000000	7.648000e+03	7.648000e+03
mean	4.999869	61999.574268	9.472019	22.041841	1.076492e+06	3.721465e+06
std	2.582369	29923.753974	5.337296	133.316467	1.728290e+06	6.004067e+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	5.455375e+04	2.724480e+05
50%	5.000000	76852.000000	9.000000	0.000000	2.932300e+05	9.315390e+05
75%	7.000000	87703.000000	14.000000	0.000000	9.164325e+05	3.259014e+06
max	9.000000	98422.000000	18.000000	3984.000000	1.124296e+07	4.271739e+07

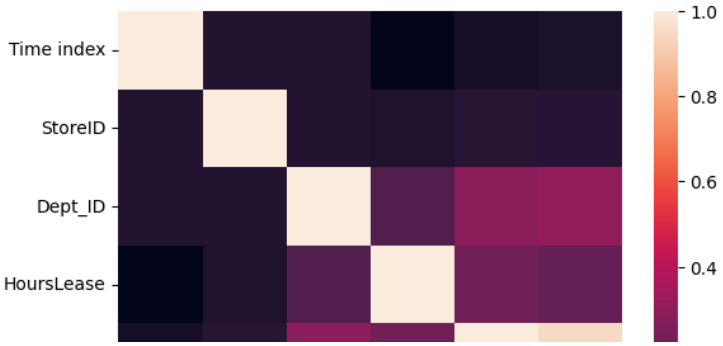
```
sns.pairplot(df1)
```

<seaborn.axisgrid.PairGrid at 0x7e4980a35390>




```
sns.heatmap(df1.corr())
```

```
<ipython-input-73-3ed1a1a51dc0>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to Fa
  sns.heatmap(df1.corr())
<Axes: >
```




```
y=df1['Sales units']
x=df1.drop(['Sales units'],axis=1)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

```
model=LinearRegression()

model.fit(x_train,y_train)

model.intercept_
```

```
69137.55605234369
```

```
coeff=pd.DataFrame(model.coef_,x.columns,columns=["Coefficient"])

coeff
```

	Coefficient
MonthYear	4225.147057
Time index	-2567.754620
StoreID	0.258279
Dept_ID	-8512.478788
HoursOwn	14.732798
HoursLease	382.902951
Turnover	0.251230

```
prediction=model.predict(x_test)

plt.scatter(y_test,prediction)
```

```
<matplotlib.collections.PathCollection at 0x7e497eebeb90>
```

```
model.score(x_test,y_test)
```

```
0.9081427355613942
```

```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
la=Lasso(alpha=10)
la.fit(x_train,y_train)
print(rr.score(x_test,y_test))
la.score(x_test,y_test)
```

```
0.9081426789374274
0.9081426748268175
```

```
en=ElasticNet()
en.fit(x_train,y_train)
print(en.coef_)
print(en.intercept_)
print(en.predict(x_test))
print(en.score(x_test,y_test))
```

```
[ 4.12007713e+03 -2.46452375e+03  2.59038360e-01 -8.32957456e+03
  1.46896957e+01  3.83219842e+02  2.51247837e-01]
67759.61973803444
[4928359.32647805   49857.028568    207377.85324442 ...  693348.04390199
 2703516.69358338   98064.10168945]
0.9081276162049374
```

```
print("MAE",metrics.mean_absolute_error(y_test,prediction))
print("MSE",metrics.mean_squared_error(y_test,prediction))
print("RMSE",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
MAE 270978.733507945
MSE 285057361750.53467
RMSE 533907.6341002577
```

```
df=pd.read_csv("/content/7_uber.csv")
df
```

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	1.0
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	1.0
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	1.0
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	3.0
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	5.0
...	...	...	...	...	...	...	...	...	...
143005	37519461	2013-03-22 11:40:54.0000002	7.5	2013-03-22 11:40:54 UTC	-73.987670	40.733403	-73.976627	40.749377	1.0
143006	26885153	2011-09-04 03:29:38.0000002	6.5	2011-09-04 03:29:38 UTC	-74.002574	40.750052	-73.984594	40.747018	4.0

```
df1=df.drop(["Unnamed: 0","key","pickup_datetime"],axis=1)
df1
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	7.5	-73.999817	40.738354	-73.999512	40.723217	1.0
1	7.7	-73.994355	40.728225	-73.994710	40.750325	1.0
2	12.9	-74.005043	40.740770	-73.962565	40.772647	1.0

```
df1=df1.dropna()
df1.isna().sum()
```

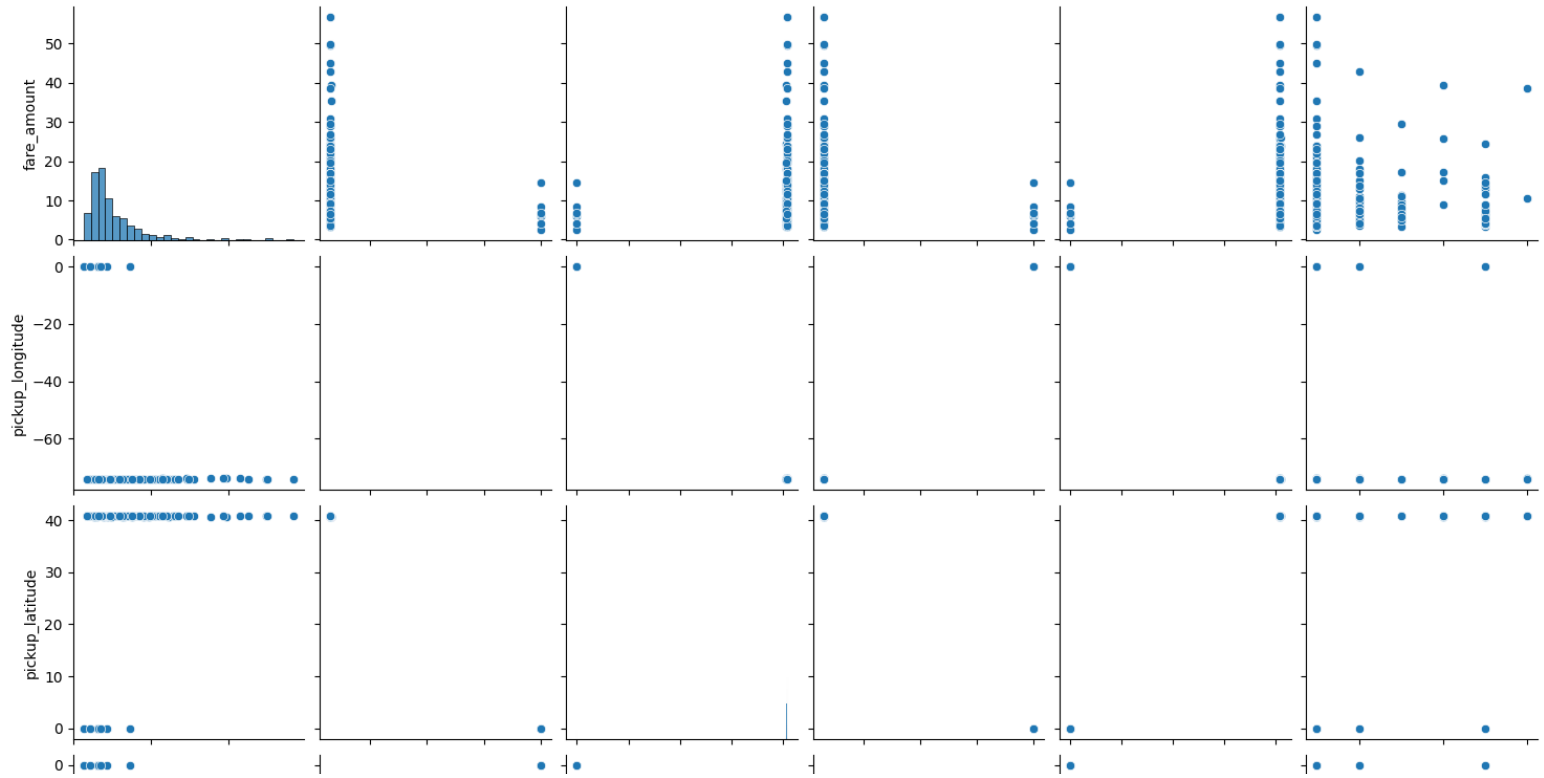
```
fare_amount          0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
dtype: int64
```

```
df1.describe()
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	143008.000000	143008.000000	143008.000000	143008.000000	143008.000000	143008.000000
mean	11.362505	-72.520476	39.940429	-72.529612	39.921274	1.681773
std	9.841352	11.416592	8.297475	14.060949	7.081246	1.412157
min	-52.000000	-1340.648410	-74.015515	-3356.666300	-881.985513	0.000000
25%	6.000000	-73.992052	40.734877	-73.991392	40.733830	1.000000
50%	8.500000	-73.981810	40.752657	-73.980074	40.753029	1.000000
75%	12.500000	-73.967094	40.767130	-73.963690	40.768037	2.000000
max	350.000000	57.418457	1644.421482	1153.572603	872.697628	208.000000

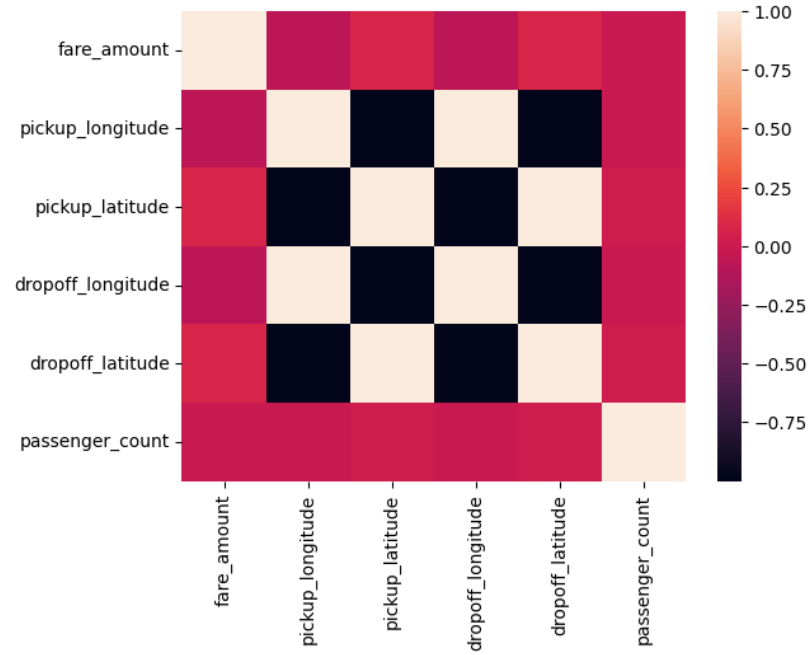
```
sns.pairplot(df1.iloc[0:300,:])
```

```
<seaborn.axisgrid.PairGrid at 0x7e497ed81570>
```




```
sns.heatmap(df1.iloc[0:300,:].corr())
```

```
<Axes: >
```




```
y=df1['passenger_count']
x=df1.drop(['passenger_count'],axis=1)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)


model=LinearRegression()
model.fit(x_train,y_train)
model.intercept_
```

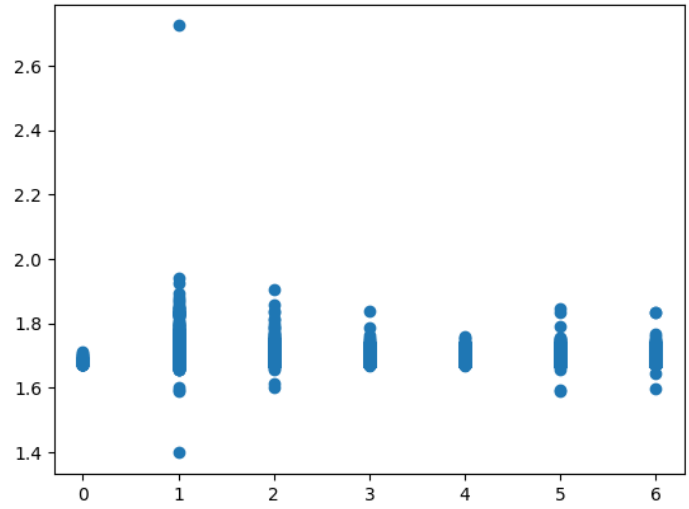
```
1.6850307288627389
```

```
coeff=pd.DataFrame(model.coef_,x.columns,columns=["Coefficient"])
coeff
```

	Coefficient
fare_amount	0.001239
pickup_longitude	0.000005
pickup_latitude	-0.000768

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

```
<matplotlib.collections.PathCollection at 0x7e4988046800>
```




```
print(model.score(x_test,y_test))
print(model.score(x_train,y_train))
```

```
-5.6341340375576365e-05
8.694258944519362e-05
```

```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
la=Lasso(alpha=10)
la.fit(x_train,y_train)
print(rr.score(x_test,y_test))
la.score(x_test,y_test)
```

```
-5.633875001587185e-05
-1.8709026949892404e-07
```

```
en=ElasticNet()
en.fit(x_train,y_train)
print(en.coef_)
print(en.intercept_)
print(en.predict(x_test))
print(en.score(x_test,y_test))
```

```
[ 0.  0. -0.  0. -0.]
1.6816043154687579
[1.68160432 1.68160432 1.68160432 ... 1.68160432 1.68160432 1.68160432]
-1.8709026949892404e-07
```

```
print("MAE",metrics.mean_absolute_error(y_test,prediction))
print("MSE",metrics.mean_squared_error(y_test,prediction))
print("RMSE",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
MAE 0.9540573841116016
MSE 1.6908867161123073
RMSE 1.3003409999351352
```

```
df=pd.read_csv("/content/8_BreastCancerPrediction (1).csv")
df
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_wor
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	...	17
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	...	23
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	...	25
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	...	26
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	...	16
...	...	...	...	...	...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	...	26
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	...	38
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	...	34

```
df.isna().sum()
```

```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
Unnamed: 32                569
dtype: int64
```

```
df1=df.drop(["Unnamed: 32"],axis=1)
df1["diagnosis"]= df1["diagnosis"].replace("M",1,regex=True)
df1["diagnosis"]= df1["diagnosis"].replace("B",0,regex=True)
df1
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_wors
0	842302	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	...	25.38
1	842517	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	...	24.99
2	84300903	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	...	23.57
3	84348301	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	...	14.91
4	84358402	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	...	22.54
...	...	...	...	...	...	...	...	...	...	...	...	
564	926424	1	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	...	25.45
565	926682	1	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	...	23.69
566	926954	1	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	...	18.98
567	927241	1	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	...	25.74
568	92751	0	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	...	9.45

569 rows × 32 columns

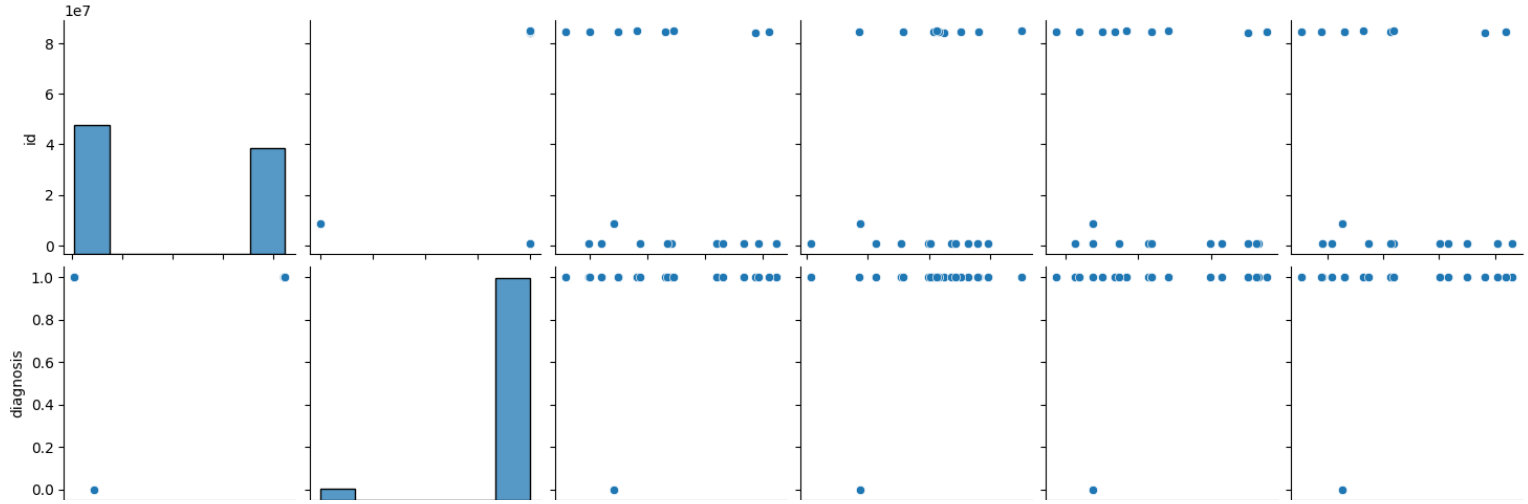
```
df1.describe()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	rad
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	5
mean	3.037183e+07	0.372583	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	...	
std	1.250206e+08	0.483918	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	...	
min	8.670000e+03	0.000000	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	...	
25%	8.692180e+05	0.000000	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	...	
50%	9.060240e+05	0.000000	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	...	
75%	8.813129e+06	1.000000	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	...	
max	9.113205e+08	1.000000	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	...	

8 rows × 32 columns

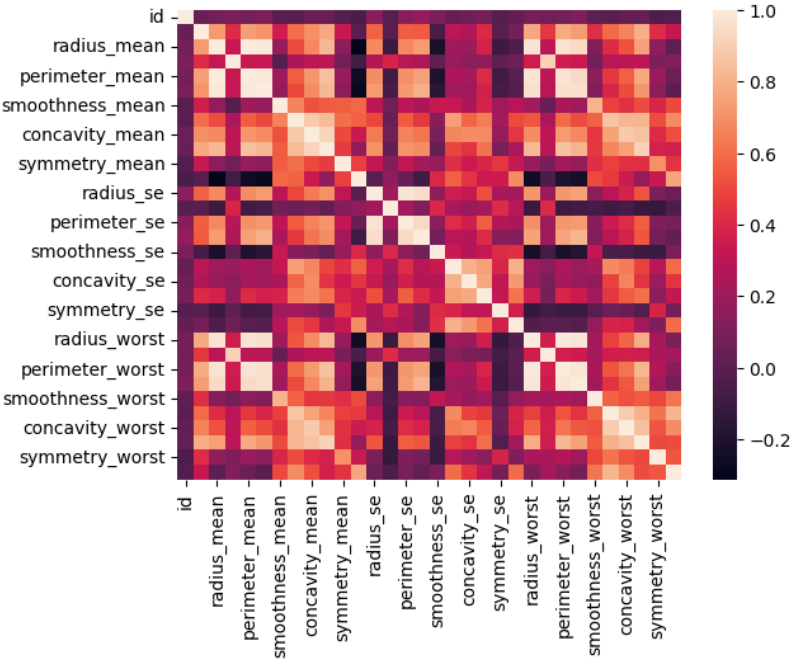
```
sns.pairplot(df1.iloc[:20,:6])
```

```
<seaborn.axisgrid.PairGrid at 0x7e496e1b06d0>
```




```
sns.heatmap(df1.corr())
```

```
<Axes: >
```




```
y=df1['area_mean']
x=df1.drop(['area_mean'],axis=1)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

```
model=LinearRegression()
model.fit(x_train,y_train)
model.intercept_
```

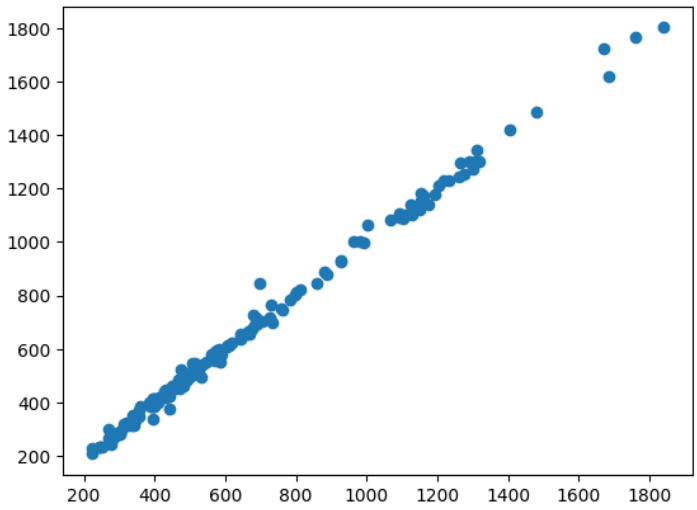
```
-294.134817764369
```

```
coeff=pd.DataFrame(model.coef_,x.columns,columns=["Coefficient"])
coeff
```

	Coefficient
id	-1.151153e-09
diagnosis	1.418979e+00
radius_mean	1.319054e+02
texture_mean	-2.957450e-01
perimeter_mean	-1.319026e+00
smoothness_mean	-3.979334e+02
compactness_mean	-2.214939e+02
concavity_mean	1.716836e+02
concave points_mean	6.539401e+01
symmetry_mean	-2.739111e+02
fractal_dimension_mean	7.113034e+02
radius_se	8.723714e+01
texture_se	-8.037749e+00
perimeter_se	2.610314e+01
area_se	-9.887156e-01
smoothness_se	9.772016e+02
compactness_se	-2.538105e+02
concavity_se	2.435259e+02
concave points_se	-3.083206e+03
symmetry_se	-5.847288e+02
fractal_dimension_se	-1.683049e+02
radius_worst	-6.568110e+01

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

```
<matplotlib.collections.PathCollection at 0x7e496cf0f040>
```




```
print(model.score(x_train,y_train))
```

```
0.9973075472045887
```

```
model.score(x_test,y_test)
```

```
0.9957958264043871
```

```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
la=Lasso(alpha=10)
la.fit(x_train,y_train)
print(rr.score(x_test,y_test))
la.score(x_test,y_test)
```

```
0.9951759177934637
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix (rcond=9.41635e-19): result may not be accura
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
0.9937483667695229
```

```
en=ElasticNet()
en.fit(x_train,y_train)
print(en.coef_)
print(en.intercept_)
print(en.predict(x_test))
print(en.score(x_test,y_test))
```

```
[-7.13025304e-09 -6.45011997e-02  3.72504625e+00 -4.23948402e-02
  1.43391442e+01 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
 -2.34119131e-01  2.87691582e+00  3.79813663e-01  0.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
 -0.00000000e+00 -5.99515191e+00  4.62334425e-02 -6.20137958e+00
  3.71267841e-01 -0.00000000e+00 -0.00000000e+00 -3.34765584e-02
 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00]
-304.2423029909248
[ 455.62354846  842.37292551 1040.16426613  569.16882648  237.0589155
  446.16567582  973.98169675  448.65759779  793.85036557  249.8135488
  490.04763667  536.48463008 1235.75978613  504.83821836  577.14651209
 1420.43945958 1140.75056185  488.24457729  566.96680325  319.29676617
 1777.21336024 1385.38692484  381.8300608   415.82837057  383.22827368
  354.08316457  482.29235276 1130.38911084 1150.1980071   268.50179767
 1204.67738579 1032.96035778  498.67087348  314.00928729  302.45029253
  975.57252027  341.97087794  651.20652488  868.37546215 1151.38831641
  443.00525255 1759.32720006  357.10311472 1595.02926284 1287.48740724
  699.53308855  224.45521042  270.24499472  656.23877569  395.30593947
  500.35235789 1242.51375744  505.51446186  512.57631099  405.82086029
  249.23101356  709.31646951 1228.99231933  534.83133619  500.23207659
  934.49182327  595.04844005  186.88872092  286.63902656  610.22861894
  671.71299829  507.72900513  341.58305951  612.5528986   529.060107
  422.07016029  336.30528867  371.71623154  398.74717847  354.21425812
  598.10117189  424.66698112  644.81202371  442.82777309  475.64048297
 1692.43027048  670.33094983 1104.04709286  516.42574193  463.467609
  805.6653376  1093.06678799  685.3373316   762.12587585  761.66298777
  413.51682206 1059.30445643  390.43315004  392.49702116  695.3265583
 1243.9618513   589.23636032  767.8170504   732.78419668  549.69193508
  858.35087081  576.68688495  429.56073318  505.19949133  569.25103918
  402.48844818 1284.59399907  499.58152427  627.96258175  717.10841749
  469.68391168  474.9253471   570.37523345  303.82303768  741.91022201
  310.86983466  483.83087087  502.08511685 1165.05406742  404.33861954
  596.10530461  249.19592274  656.04128691  612.3850033   665.61744603
  438.79592139  440.42203604  794.95185078  168.54327921  500.17942856
  459.29095122 1169.18721134 1230.98925979 1254.00523559  404.631554
  439.38935339  331.26734429  900.02111523  214.64320564  423.69716069
  581.40380878  729.47190366  564.4812261   536.06916332 1082.00336619
  629.52030788  410.1922487  1327.57657276 1265.00344467 1123.50273017
  247.88884114  472.68451503  395.22126814  662.89174424  390.58402254
  414.66263612  502.8809179   428.71350316  447.29085269  389.25521423
  530.67820519  532.35380783  356.21663691 1464.4057551  1105.88871537
  477.58443904  541.49906068  421.88604524  922.62776123  514.95898047
  521.34654965]
0.9940817207473047
```

```
print("MAE",metrics.mean_absolute_error(y_test,prediction))
print("MSE",metrics.mean_squared_error(y_test,prediction))
print("RMSE",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
MAE 14.701800061638659
MSE 491.50708638143993
RMSE 22.1699590974237
```

```
df=pd.read_csv("/content/11_winequality-red.csv")
df
```

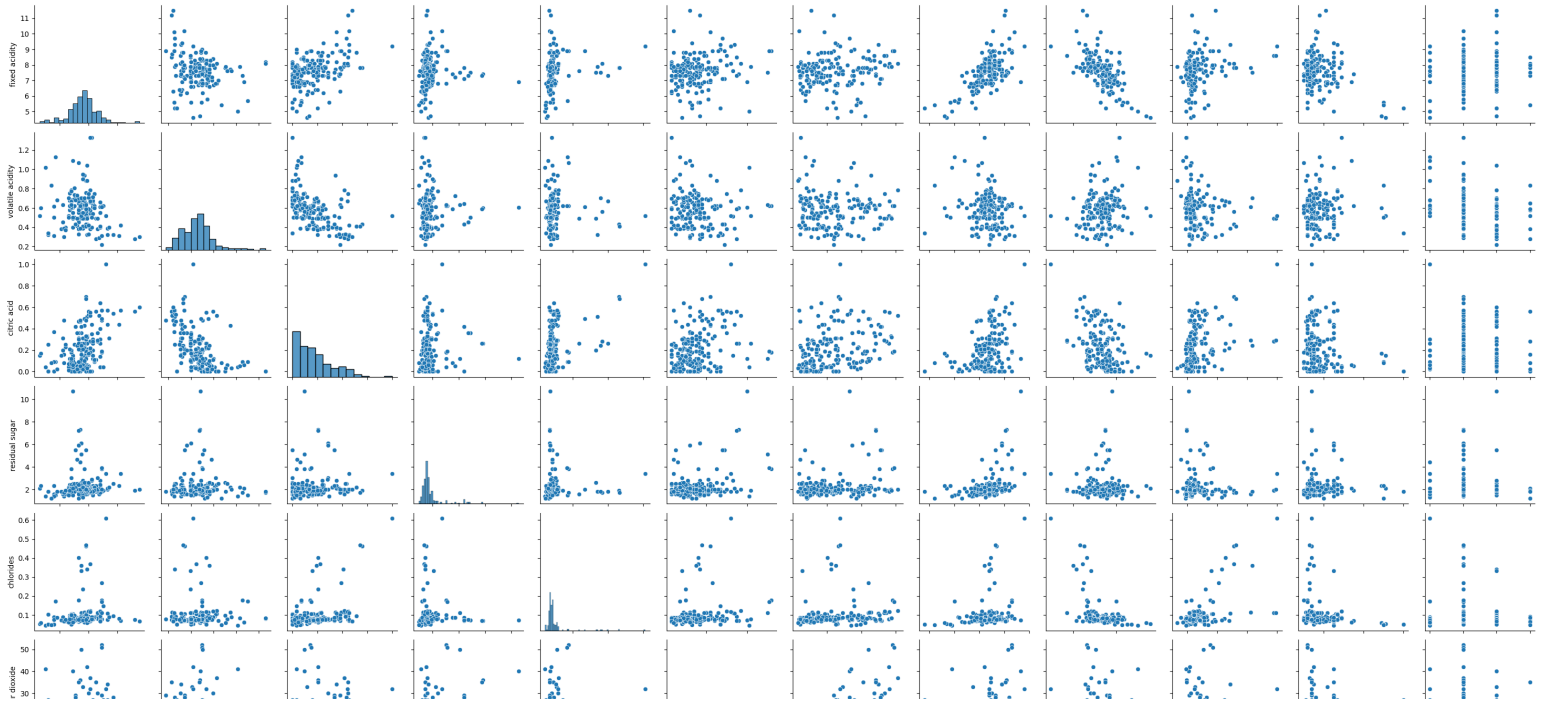
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5

```
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
```

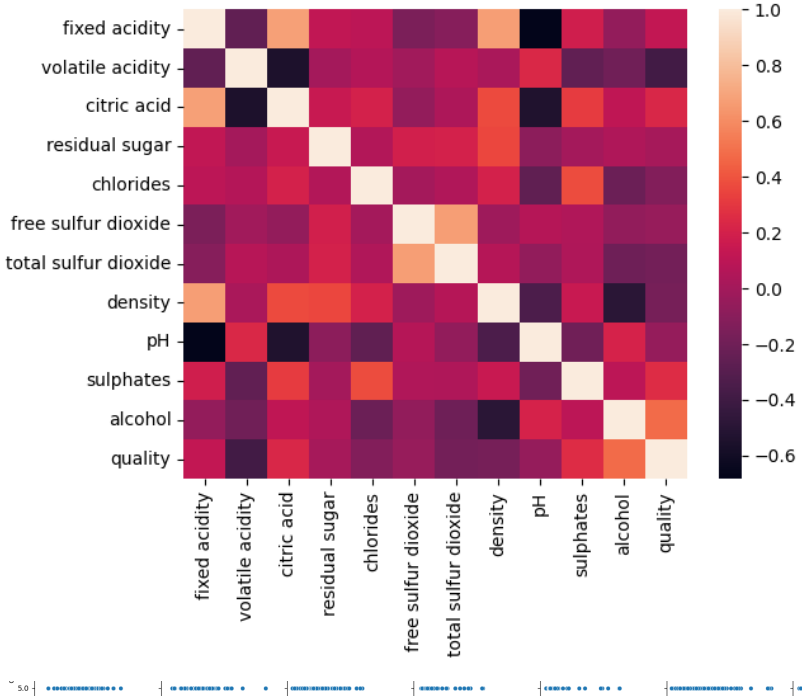
```
sns.pairplot(df.iloc[:200,:])
```

```
<seaborn.axisgrid.PairGrid at 0x7e496cf759f0>
```



```
sns.heatmap(df.corr())
```

```
<Axes: >
```




```
y=df['density']
x=df.drop(['density'],axis=1)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)


model=LinearRegression()
model.fit(x_train,y_train)
model.intercept_
```

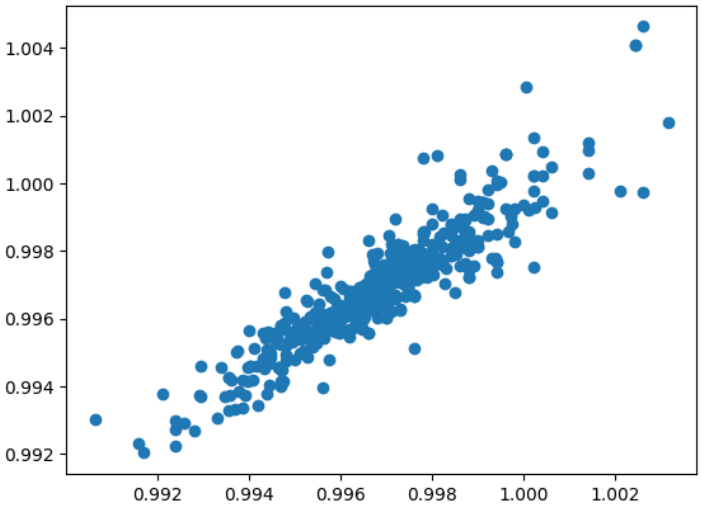
```
0.9796432210106216
```

```
coeff=pd.DataFrame(model.coef_,x.columns,columns=["Coefficient"])
coeff
```

	Coefficient
fixed acidity	0.000943
volatile acidity	0.000484
citric acid	-0.000057
residual sugar	0.000435
chlorides	0.001597
free sulfur dioxide	-0.000006
total sulfur dioxide	0.000002

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

```
<matplotlib.collections.PathCollection at 0x7e4966e47fa0>
```




```
model.score(x_test,y_test)
```

```
0.8389205017022511
```

```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
la=Lasso(alpha=10)
la.fit(x_train,y_train)
print(rr.score(x_test,y_test))
la.score(x_test,y_test)
```

```
0.8247661621273629
-0.002940885054244058
```

```
en=ElasticNet()
en.fit(x_train,y_train)
print(en.coef_)
print(en.intercept_)
print(en.predict(x_test))
print(en.score(x_test,y_test))
```

```
 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
```

```
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
0.99671621 0.99671621 0.99671621 0.99671621 0.99671621 0.99671621
```

```
print("MAE",metrics.mean_absolute_error(y_test,prediction))
print("MSE",metrics.mean_squared_error(y_test,prediction))
print("RMSE",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
MAE 0.0005568867174490188
MSE 5.642509718200992e-07
RMSE 0.0007511664075423629
```

```
df=pd.read_csv("/content/13_placement.csv")
df
```

	cgpa	placement_exam_marks	placed
0	7.19	26.0	1
1	7.46	38.0	1
2	7.54	40.0	1
3	6.42	8.0	1
4	7.23	17.0	0
...	...	...	...
995	8.87	44.0	1
996	9.12	65.0	1
997	4.89	34.0	0
998	8.62	46.0	1
999	4.90	10.0	1

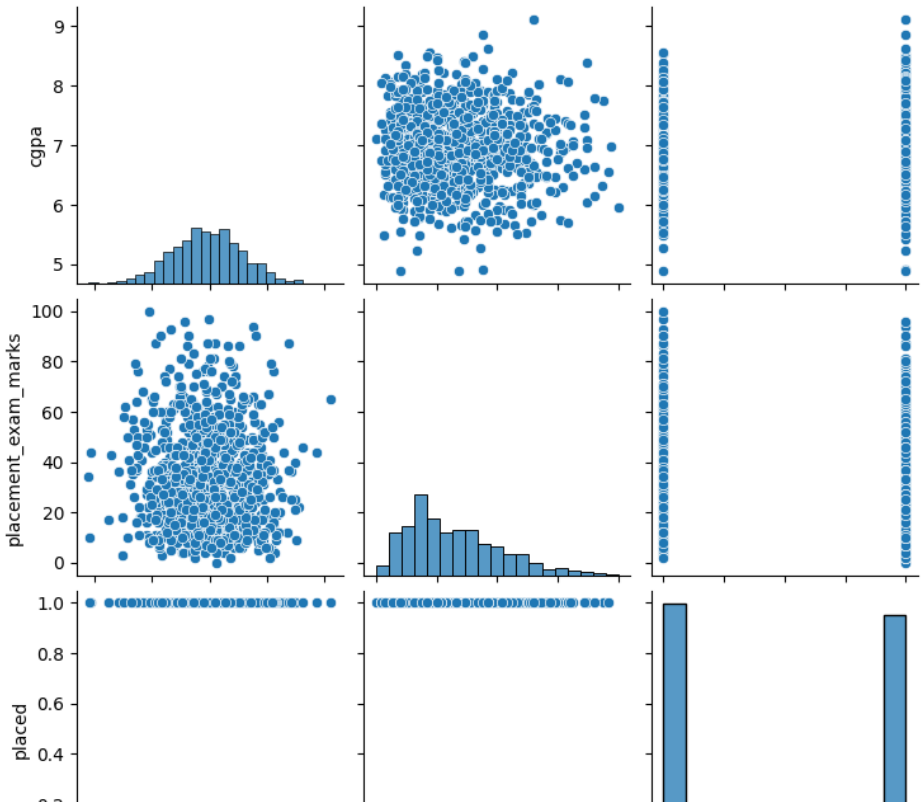
1000 rows × 3 columns

```
df.isna().sum()
```

```
cgpa                    0
placement_exam_marks    0
placed                  0
dtype: int64
```

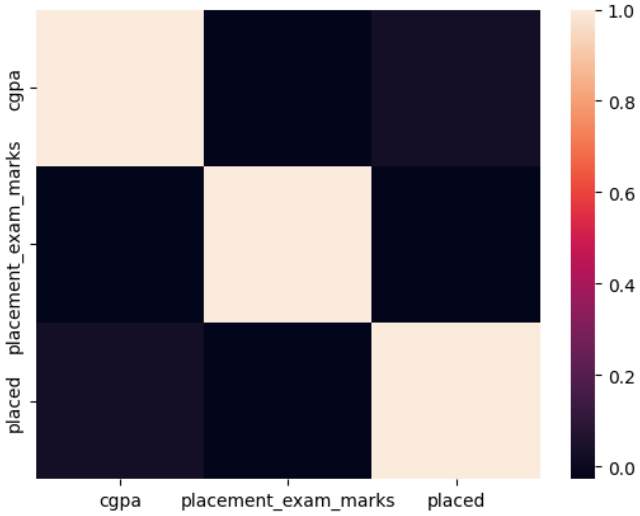
```
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x7e496a6183a0>
```




```
sns.heatmap(df.corr())
```

```
<Axes: >
```




```
y=df['cgpa']
x=df.drop(['cgpa'],axis=1)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)



model=LinearRegression()
model.fit(x_train,y_train)
model.intercept_
```

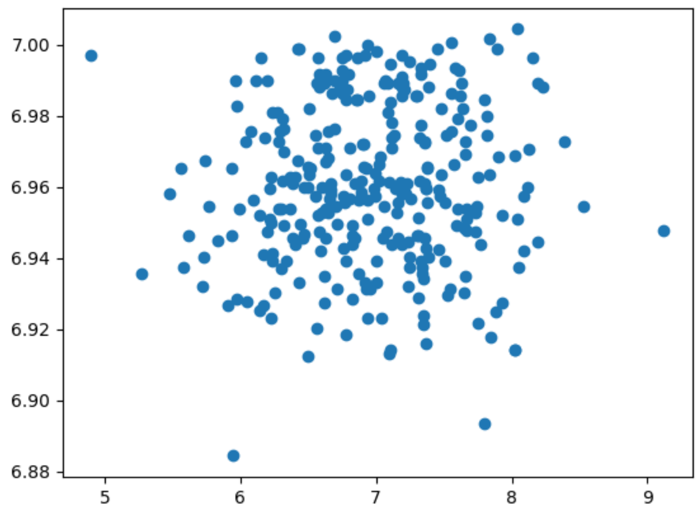
```
6.974328872291244
```

```
coeff=pd.DataFrame(model.coef_,x.columns,columns=["Coefficient"])
coeff
```

	Coefficient
placement_exam_marks	-0.000900
placed	0.031792

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

```
<matplotlib.collections.PathCollection at 0x7e49645beef0>
```




```
print(model.score(x_test,y_test))
print(model.score(x_train,y_train))
```

```
0.001206620602813957
0.0015608392341649457
```

```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
la=Lasso(alpha=10)
la.fit(x_train,y_train)
print(rr.score(x_test,y_test))
la.score(x_test,y_test)
```

```
0.0011979588331737512
-2.022352495467672e-06
```

```
en=ElasticNet()
en.fit(x_train,y_train)
print(en.coef_)
print(en.intercept_)
print(en.predict(x_test))
print(en.score(x_test,y_test))
```

```
[-0.  0.]
6.960971428571429
[6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
```

```
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143
 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143 6.96097143]
-2.022352495467672e-06
```

```
print("MAE",metrics.mean_absolute_error(y_test,prediction))
print("MSE",metrics.mean_squared_error(y_test,prediction))
print("RMSE",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
MAE 0.5083755055141442
MSE 0.3958183360150628
RMSE 0.6291409508330091
```

```
df10=pd.read_csv("/content/5_Instagram data.csv")
df10
```

	Impressions	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profile Visits	Follows
0	3920	2586	1028	619	56	98	9	5	162	35	2
1	5394	2727	1838	1174	78	194	7	14	224	48	10
2	4021	2085	1188	0	533	41	11	1	131	62	12
3	4528	2700	621	932	73	172	10	7	213	23	8
4	2518	1704	255	279	37	96	5	4	123	8	0
...	...	...	...	...	...	...	...	...	...	...	...
114	13700	5185	3041	5352	77	573	2	38	373	73	80
115	5731	1923	1368	2266	65	135	4	1	148	20	18
116	4139	1133	1538	1367	33	36	0	1	92	34	10
117	32695	11815	3147	17414	170	1095	2	75	549	148	214
118	36919	13473	4176	16444	2547	653	5	26	443	611	228

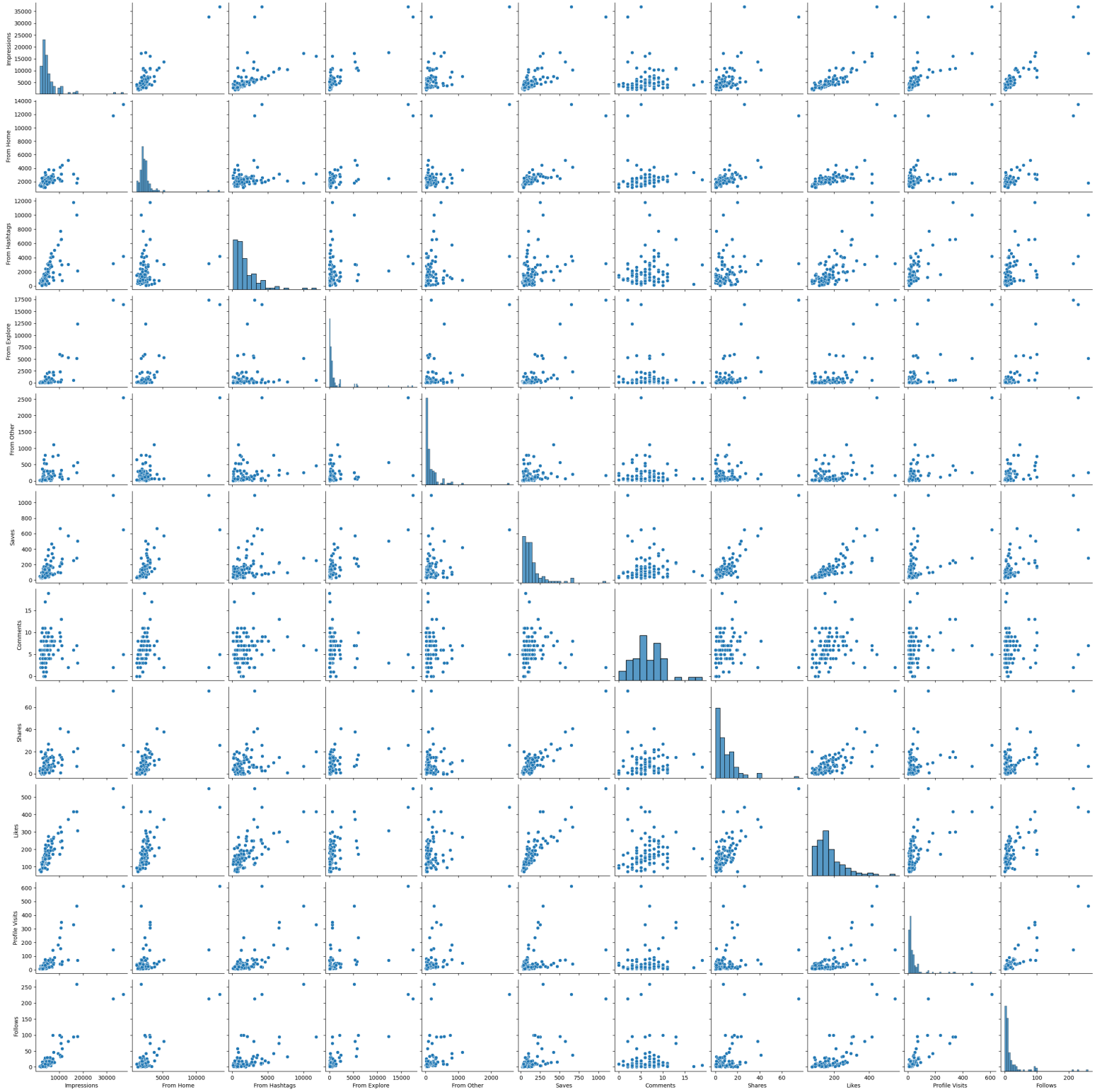
119 rows × 11 columns

```
df10.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119 entries, 0 to 118
Data columns (total 11 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
dtypes: int64(11)
memory usage: 10.4 KB
```

```
sns.pairplot(df10)
```

<seaborn.axisgrid.PairGrid at 0x78cc802d4b80>




```
sns.heatmap(df10.corr())
```