Import pandas and numpy

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as mp
import seaborn as sb
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
kcd=pd.read_csv("/home/placement/Downloads/fiat500.csv")
kcd.info
```

Out[2]:
```
<bound method DataFrame.info of         ID   model  engine_power  age_in_days      km  previous_o
wners  \
0        1  lounge            51          882   25000                1
1        2     pop            51         1186   32500                1
2        3   sport            74         4658  142228                1
3        4  lounge            51         2739  160000                1
4        5     pop            73         3074  106880                1
...    ...     ...           ...          ...     ...              ...
1533  1534   sport            51         3712  115280                1
1534  1535  lounge            74         3835  112000                1
1535  1536     pop            51         2223   60457                1
1536  1537  lounge            51         2557   80750                1
1537  1538     pop            51         1766   54276                1

            lat        lon  price
0     44.907242   8.611560   8900
1     45.666359  12.241890   8800
2     45.503300  11.417840   4200
3     40.633171  17.634609   6000
4     41.903221  12.495650   5700
...         ...        ...    ...
1533  45.069679   7.704920   5200
1534  45.845692   8.666870   4600
1535  45.481541   9.413480   7500
1536  45.000702   7.682270   5990
1537  40.323410  17.568270   7900

[1538 rows x 9 columns]>
```

In [3]:
```
kcd
```

Out[3]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon	price
0	1	lounge	51	882	25000	1	44.907242	8.611560	8900
1	2	pop	51	1186	32500	1	45.666359	12.241890	8800
2	3	sport	74	4658	142228	1	45.503300	11.417840	4200
3	4	lounge	51	2739	160000	1	40.633171	17.634609	6000
4	5	pop	73	3074	106880	1	41.903221	12.495650	5700
...	...	...	...	...	...	...	...	...	...
1533	1534	sport	51	3712	115280	1	45.069679	7.704920	5200
1534	1535	lounge	74	3835	112000	1	45.845692	8.666870	4600
1535	1536	pop	51	2223	60457	1	45.481541	9.413480	7500
1536	1537	lounge	51	2557	80750	1	45.000702	7.682270	5990
1537	1538	pop	51	1766	54276	1	40.323410	17.568270	7900

1538 rows × 9 columns

In [4]:
```
a=kcd.groupby(['model']).count()
a
```

Out[4]:

	ID	engine_power	age_in_days	km	previous_owners	lat	lon	price
model								
lounge	1094	1094	1094	1094	1094	1094	1094	1094
pop	358	358	358	358	358	358	358	358
sport	86	86	86	86	86	86	86	86

In [5]:
```
drop=kcd.drop(['ID','lat','lon'],axis=1)
drop
```

Out[5]:

	model	engine_power	age_in_days	km	previous_owners	price
0	lounge	51	882	25000	1	8900
1	pop	51	1186	32500	1	8800
2	sport	74	4658	142228	1	4200
3	lounge	51	2739	160000	1	6000
4	pop	73	3074	106880	1	5700
...	...	...	...	...	...	...
1533	sport	51	3712	115280	1	5200
1534	lounge	74	3835	112000	1	4600
1535	pop	51	2223	60457	1	7500
1536	lounge	51	2557	80750	1	5990
1537	pop	51	1766	54276	1	7900

1538 rows × 6 columns

In [6]:
```
drop['model']=drop['model'].map({'lounge':1,'pop':2,'sport':3})
drop
```

Out[6]:

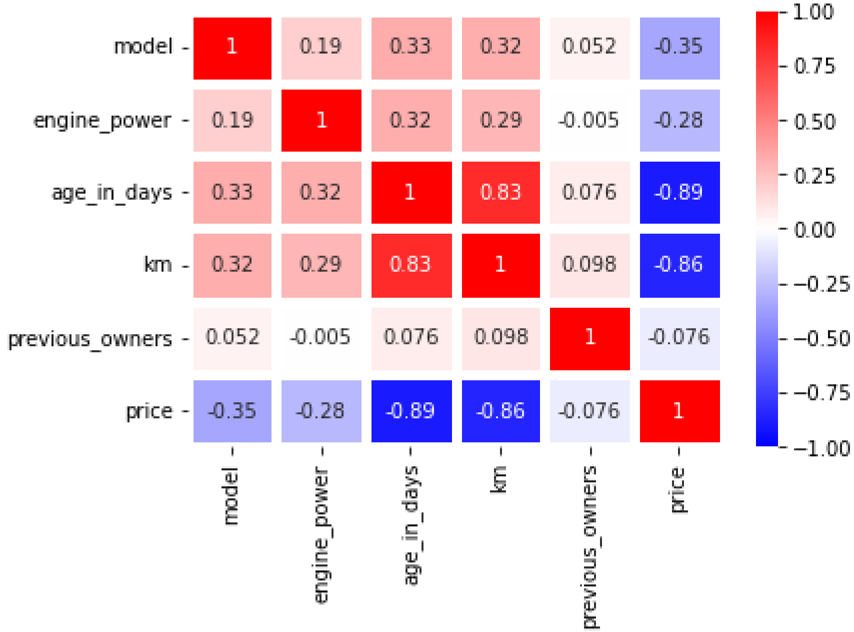
	model	engine_power	age_in_days	km	previous_owners	price
0	1	51	882	25000	1	8900
1	2	51	1186	32500	1	8800
2	3	74	4658	142228	1	4200
3	1	51	2739	160000	1	6000
4	2	73	3074	106880	1	5700
...	...	...	...	...	...	...
1533	3	51	3712	115280	1	5200
1534	1	74	3835	112000	1	4600
1535	2	51	2223	60457	1	7500
1536	1	51	2557	80750	1	5990
1537	2	51	1766	54276	1	7900

1538 rows × 6 columns

In [7]:
```
cor1=drop.corr()
```

In [8]:
```
import seaborn as sb
sb.heatmap(cor1,vmax=1,vmin=-1,annot=True,linewidths=5,cmap='bwr')
```

Out[8]: <Axes: >



In [9]:
```
y=drop['price']
x=drop.drop('price',axis=1)
x
```

Out[9]:

	model	engine_power	age_in_days	km	previous_owners
0	1	51	882	25000	1
1	2	51	1186	32500	1
2	3	74	4658	142228	1
3	1	51	2739	160000	1
4	2	73	3074	106880	1
...	...	...	...	...	...
1533	3	51	3712	115280	1
1534	1	74	3835	112000	1
1535	2	51	2223	60457	1
1536	1	51	2557	80750	1
1537	2	51	1766	54276	1

1538 rows × 5 columns

In [10]:
```
y
```

Out[10]:
```
0       8900
1       8800
2       4200
3       6000
4       5700
        ...
1533    5200
1534    4600
1535    7500
1536    5990
1537    7900
Name: price, Length: 1538, dtype: int64
```

In [11]:

```
!pip3 install scikit-learn
```

```
Requirement already satisfied: scikit-learn in ./.local/lib/python3.8/site-packages (1.2.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in ./.local/lib/python3.8/site-packages (from scikit-learn) (3.1.0)
Requirement already satisfied: scipy>=1.3.2 in ./.local/lib/python3.8/site-packages (from scikit-learn) (1.10.1)
Requirement already satisfied: numpy>=1.17.3 in ./.local/lib/python3.8/site-packages (from scikit-learn) (1.24.3)
Requirement already satisfied: joblib>=1.1.1 in ./.local/lib/python3.8/site-packages (from scikit-learn) (1.2.0)
```

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=0.33,random_state=42)
```

In [13]:

```
x_test.head(10)
```

Out[13]:

	model	engine_power	age_in_days	km	previous_owners
481	2	51	3197	120000	2
76	2	62	2101	103000	1
1502	1	51	670	32473	1
669	1	51	913	29000	1
1409	1	51	762	18800	1
1414	1	51	762	39751	1
1089	1	51	882	33160	1
1507	1	51	701	17324	1
970	1	51	701	29000	1
1198	1	51	1155	38000	1

LinearRegression

In [14]:
```
from sklearn.linear_model import LinearRegression
reg=LinearRegression()
reg.fit(x_train,y_train)
```

Out[14]:
```
▾ LinearRegression
LinearRegression()
```

In [15]:
```
y_pred=reg.predict(x_test)
y_pred
```

Out[15]:
```
array([ 5994.51703157,  7263.58726658,  9841.90754881,  9699.31627673,
       10014.19892635,  9630.58715835,  9649.4499026 , 10092.9819664 ,
        9879.19498711,  9329.19347948, 10407.2964056 ,  7716.91706011,
        7682.89152522,  6673.95810983,  9639.42618839, 10346.53679153,
        9366.53363673,  7707.90063494,  4727.33552438, 10428.17092937,
       10359.87663878, 10364.84674179,  7680.16157493,  9927.58506055,
        7127.7284177 ,  9097.51161986,  4929.31229715,  6940.60225317,
        7794.35120591,  9600.43942019,  7319.85877519,  5224.05298205,
        5559.52039134,  5201.35403287,  8960.11762682,  5659.72968338,
        9915.79926869,  8255.93615893,  6270.40332834,  8556.73835062,
        9749.72882426,  6873.76758364,  8951.72659758, 10301.95669828,
        8674.89268564, 10301.93257222,  9165.73586068,  8846.92420399,
        7044.68964545,  9052.4031418 ,  9390.75738772, 10267.3912561 ,
       10046.90924744,  6855.71260655,  9761.93338967,  9450.05744337,
        9274.98388541, 10416.00474283,  9771.10646661,  7302.96566423,
       10082.61483093,  6996.96553454,  9829.40534825,  7134.21944391,
        6407.26222178,  9971.82132188,  9757.01618446,  8614.84049875,
        8437.92452169,  6489.24658616,  7752.65456507,  6626.60510856,
        8329.88998217, 10412.00324329,  7342.77348105,  8543.63624413,
```

Efficiency

In [16]:
```
from sklearn.metrics import r2_score
r2_score(y_test,y_pred)
```

Out[16]:
```
0.8383895235218546
```

Mean squared error

In [17]:
```
from sklearn.metrics import mean_squared_error as kc
sq=kc(y_test,y_pred)
sq
```

Out[17]:
```
593504.2888137395
```

In [18]:
```
import math as m
dp=m.sqrt(sq)
print(dp)
```

```
770.3922954013361
```

In [19]:
```
results=pd.DataFrame(columns=['price','predicted'])
results['price']=y_test
results['predicted']=y_pred
results=results.reset_index()
results['ID']=results.index
results.head(10)
results.head(10)
```

Out[19]:

	index	price	predicted	ID
0	481	7900	5994.517032	0
1	76	7900	7263.587267	1
2	1502	9400	9841.907549	2
3	669	8500	9699.316277	3
4	1409	9700	10014.198926	4
5	1414	9900	9630.587158	5
6	1089	9900	9649.449903	6
7	1507	9950	10092.981966	7
8	970	10700	9879.194987	8
9	1198	8999	9329.193479	9

In [20]:
```
results['actual price']=results.apply(lambda column:column.price-column.predicted,axis=1)
results
```

Out[20]:

	index	price	predicted	ID	actual price
0	481	7900	5994.517032	0	1905.482968
1	76	7900	7263.587267	1	636.412733
2	1502	9400	9841.907549	2	-441.907549
3	669	8500	9699.316277	3	-1199.316277
4	1409	9700	10014.198926	4	-314.198926
...	...	...	...	...	...
503	291	10900	10007.364639	503	892.635361
504	596	5699	6390.174715	504	-691.174715
505	1489	9500	10079.478928	505	-579.478928
506	1436	6990	8363.337585	506	-1373.337585
507	575	10900	10344.486077	507	555.513923

508 rows × 5 columns

Graph for linear regression

In [21]:
```
'''sb.lineplot(x='ID',y='price',data=results.head(50))
sb.lineplot(x='ID',y='predicted',data=results.head(50))
mp.plot()'''
```

Out[21]:
```
"sb.lineplot(x='ID',y='price',data=results.head(50))\nsb.lineplot(x='ID',y='predicted',data=resul
ts.head(50))\nmp.plot()"
```

Ridge regression

In [22]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge
```

In [23]:
```
alpha=[1e-15,1e-10,1e-8,1e-4,1e-3,1e-2,1,5,10,20,30]
ridge=Ridge()
parameters={'alpha':alpha}
ridge_regressor=GridSearchCV(ridge,parameters)
ridge_regressor.fit(x_train,y_train)
```

Out[23]:
```
▸ GridSearchCV
▸ estimator: Ridge
    ▸ Ridge
```

In [24]:
```
ridge_regressor.best_params_
```

Out[24]:
```
{'alpha': 30}
```

In [25]:
```
ridge=Ridge(alpha=30)
ridge.fit(x_train,y_train)
```

Out[25]:
```
▾ Ridge
Ridge(alpha=30)
```

In [26]:
```
y_pred_ridge=ridge.predict(x_test)
y_pred_ridge
```

```
        9701.97759839,  6265.1567015 ,  7881.36123438,  9500.19796637,
        5025.47380817,  9325.1177875 ,  9953.42729557, 10066.99108051,
        6340.28325743,  9829.7201522 ,  9212.4268255 ,  5354.86017533,
        5519.09597589,  4621.18155819, 10172.08083307,  9997.38747039,
        5314.69298063,  8635.97320822,  7014.22436159, 10164.70409768,
       10162.96208228,  6030.32481479,  9721.22413685,  9643.73908   ,
        9119.42794645,  9151.15935393, 10060.26173637,  9797.55709111,
        7457.40754687,  5207.31722239,  9553.30134771, 10215.40242476,
        5539.21836768, 10641.80922721,  6109.58327259,  9818.42897818,
        9823.93271327,  7957.05365864,  6532.69995519,  9911.89614637,
        8305.02395466,  9090.71359881,  6094.33252933, 10381.83315741,
        6341.430594  ,  8716.47527835,  8354.39216562,  9777.58909907,
        8401.42735884, 10064.05168895,  9976.72869098,  9999.3636296 ,
       10326.61690103,  8528.49212387,  6707.82444589,  9354.63243335,
        6503.27431508, 10324.78127985,  9177.51196649, 10428.42133921,
        9102.45078883,  9925.71907421,  8489.23733274,  9333.40573643,
       10146.38735818,  8393.73837779,  4841.91223122, 10049.07336204,
       10128.07061867, 10561.33720457, 10133.64569557,  4740.81560101,
        7254.59493413,  9652.61398542,  9738.6110774 ,  5626.86021564,
       10172.5372122 ,  5147.1492093 ,  8283.69641236,  7559.36126123
```

Mean_squared error

In [27]:
```
from sklearn.metrics import mean_squared_error#mean_squared error
Ridge_Error=mean_squared_error(y_pred_ridge,y_test)
Ridge_Error
```

Out[27]: 590569.9121697355

Finding the efficieny

In [28]:
```
from sklearn.metrics import r2_score
r2_score(y_test,y_pred_ridge)#finding the efficieny
```

Out[28]: 0.8391885506165899

In [29]:
```
a=drop.loc[drop.model==1]
a
```

Out[29]:

	model	engine_power	age_in_days	km	previous_owners	price
0	1	51	882	25000	1	8900
3	1	51	2739	160000	1	6000
6	1	51	731	11600	1	10750
7	1	51	1521	49076	1	9190
11	1	51	366	17500	1	10990
...	...	...	...	...	...	...
1528	1	51	2861	126000	1	5500
1529	1	51	731	22551	1	9900
1530	1	51	670	29000	1	10800
1534	1	74	3835	112000	1	4600
1536	1	51	2557	80750	1	5990

1094 rows × 6 columns

In [30]:
```
results=pd.DataFrame(columns=['price','predicted'])
results['price']=y_test
results['predicted']=y_pred_ridge
results=results.reset_index()
results['ID']=results.index
results.head(10)
```

Out[30]:

	index	price	predicted	ID
0	481	7900	5987.682984	0
1	76	7900	7272.490419	1
2	1502	9400	9839.847697	2
3	669	8500	9696.775405	3
4	1409	9700	10012.040862	4
5	1414	9900	9628.286853	5
6	1089	9900	9646.945160	6
7	1507	9950	10090.960592	7
8	970	10700	9877.094341	8
9	1198	8999	9326.088982	9

In [31]:
```
results['actual price']=results.apply(lambda column:column.price-column.predicted,axis=1)
results['actual price']=results.apply(lambda column:column.price-column.predicted,axis=1)
results
```

Out[31]:

	index	price	predicted	ID	actual price
0	481	7900	5987.682984	0	1912.317016
1	76	7900	7272.490419	1	627.509581
2	1502	9400	9839.847697	2	-439.847697
3	669	8500	9696.775405	3	-1196.775405
4	1409	9700	10012.040862	4	-312.040862
...	...	...	...	...	...
503	291	10900	10005.311518	503	894.688482
504	596	5699	6400.852430	504	-701.852430
505	1489	9500	10096.776914	505	-596.776914
506	1436	6990	8358.743798	506	-1368.743798
507	575	10900	10343.148204	507	556.851796

508 rows × 5 columns

Graph for ridge regression

In [32]:
```
'''sb.lineplot(x='ID',y='price',data=results.head(50))
sb.lineplot(x='ID',y='predicted',data=results.head(50))
mp.plot()'''
```

Out[32]:
```
"sb.lineplot(x='ID',y='price',data=results.head(50))\nsb.lineplot(x='ID',y='predicted',data=resul
ts.head(50))\nmp.plot()"
```

Elastic regression

In [33]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import ElasticNet
```

In [34]:
```
elastic = ElasticNet()
parameters = {'alpha': [1e-15, 1e-10, 1e-8, 1e-4, 1e-3,1e-2, 1, 5, 10, 20]}
elastic_regressor = GridSearchCV(elastic, parameters)
elastic_regressor.fit(x_train, y_train)
```

Out[34]:
```
▸ GridSearchCV
▸ estimator: ElasticNet
    ▸ ElasticNet
```

In [35]:
```
elastic_regressor.best_params_
```

Out[35]:
```
{'alpha': 0.01}
```

In [36]:
```
elastic = ElasticNet(alpha=30)
elastic.fit(x_train,y_train)
```

Out[36]:
```
▾ ElasticNet
ElasticNet(alpha=30)
```

In [37]:
```
y_pred_elastic=elastic.predict(x_test)
y_pred_elastic
```

```
        9689.93075146, 10320.54884631, 10242.30685083,  7391.14947941,
        9671.97973912,  6185.42897933,  7829.15510381,  9657.48971348,
        4924.49439005,  9283.26099577,  9923.99467256, 10038.29040576,
        6252.66519572,  9800.88978664,  9366.84280568,  5441.09491145,
        5425.48535649,  4711.07356579, 10146.51249424,  9968.23818219,
        5320.9728323 ,  8783.50752611,  6937.56936229, 10329.76624414,
       10135.23559728,  5759.2233148 ,  9687.16305163,  9613.01022669,
        8989.07787636,  9112.72426114, 10032.21748039,  9763.28820058,
        7582.34034345,  5300.98702745,  9709.91435322, 10191.16872762,
        5353.31571914, 10345.86930025,  6022.55471869,  9787.80504014,
        9792.98861668,  8091.12634972,  6448.39849883,  9882.8956354 ,
        8252.07242921,  8777.10169669,  6006.50588925, 10359.70462856,
        6251.36492216,  8864.52845311,  8300.00361731,  9748.07857567,
        8547.44996746, 10035.68770062,  9948.50167427,  9877.83977101,
       10303.77685742,  8477.30212475,  6630.25004181,  9312.63318363,
        6425.08132313, 10301.60818958,  9332.72391804, 10406.2374518 ,
        9059.51448169,  9898.21870992,  8621.52033296,  9489.62246404,
       10311.33144847,  8334.00509691,  4929.03963075, 10019.93598731,
       10292.89665279, 10540.36562619, 10105.37462723,  5014.48195287,
        7179.48918596,  9622.29796477,  9708.84933047,  5530.45668115,
```

Mean_squared error

In [38]:
```
from sklearn.metrics import mean_squared_error#mean_squared error
elastic_Error=mean_squared_error(y_pred_elastic,y_test)
elastic_Error
```

Out[38]: 580642.9647580221

Finding the efficieny

In [39]:
```
from sklearn.metrics import r2_score
r2_score(y_test,y_pred_elastic)#finding the efficieny
```

Out[39]: 0.8418916459967212

In [40]:
```
results=pd.DataFrame(columns=['price','predicted'])
results['price']=y_test
results['predicted']=y_pred_elastic
results=results.reset_index()
results['ID']=results.index
results.head(10)
```

Out[40]:

	index	price	predicted	ID
0	481	7900	6001.991118	0
1	76	7900	7310.025710	1
2	1502	9400	9810.738446	2
3	669	8500	9663.956323	3
4	1409	9700	9982.986019	4
5	1414	9900	9596.758615	5
6	1089	9900	9614.160541	6
7	1507	9950	10063.114198	7
8	970	10700	9847.869524	8
9	1198	8999	9288.104509	9

In [41]:
```
results['actual price']=results.apply(lambda column:column.price-column.predicted,axis=1)
results
```

Out[41]:

	index	price	predicted	ID	actual price
0	481	7900	6001.991118	0	1898.008882
1	76	7900	7310.025710	1	589.974290
2	1502	9400	9810.738446	2	-410.738446
3	669	8500	9663.956323	3	-1163.956323
4	1409	9700	9982.986019	4	-282.986019
...	...	...	...	...	...
503	291	10900	9976.913093	503	923.086907
504	596	5699	6507.813210	504	-808.813210
505	1489	9500	10261.756884	505	-761.756884
506	1436	6990	8307.159518	506	-1317.159518
507	575	10900	10320.770340	507	579.229660

508 rows × 5 columns

Graph for elastic regression

In [42]:
```
'''sb.lineplot(x='ID',y='price',data=results.head(50))
sb.lineplot(x='ID',y='predicted',data=results.head(50))
mp.plot()'''
```

Out[42]:
```
"sb.lineplot(x='ID',y='price',data=results.head(50))\nsb.lineplot(x='ID',y='predicted',data=resul
ts.head(50))\nmp.plot()"
```

In [43]:
```
r2_score(y_test,y_pred)#linear regression efficiency
```

Out[43]:
```
0.8383895235218546
```

In [44]:
```
r2_score(y_test,y_pred_ridge)#ridge regression efficiency
```

Out[44]:
```
0.8391885506165899
```

In [45]:
```
r2_score(y_test,y_pred_elastic)#elastic regression efficiency
```

Out[45]:
```
0.8418916459967212
```

In [ ]: