Dataset 1

In [134]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge, RidgeCV, Lasso
from sklearn.preprocessing import StandardScaler
```

In [135]:

```
data=pd.read_csv(r"C:\Users\chila\Downloads\Advertising.csv")
data
```

Out[135]:

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	12.0
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9
...	...	...	...	...
195	38.2	3.7	13.8	7.6
196	94.2	4.9	8.1	14.0
197	177.0	9.3	6.4	14.8
198	283.6	42.0	66.2	25.5
199	232.1	8.6	8.7	18.4

200 rows × 4 columns

In [136]:

```
data.head()
```

Out[136]:

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	12.0
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9

In [137]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   TV         200 non-null    float64
 1   Radio      200 non-null    float64
 2   Newspaper  200 non-null    float64
 3   Sales      200 non-null    float64
dtypes: float64(4)
memory usage: 6.4 KB
```

In [138]:

```
data.describe
```

Out[138]:

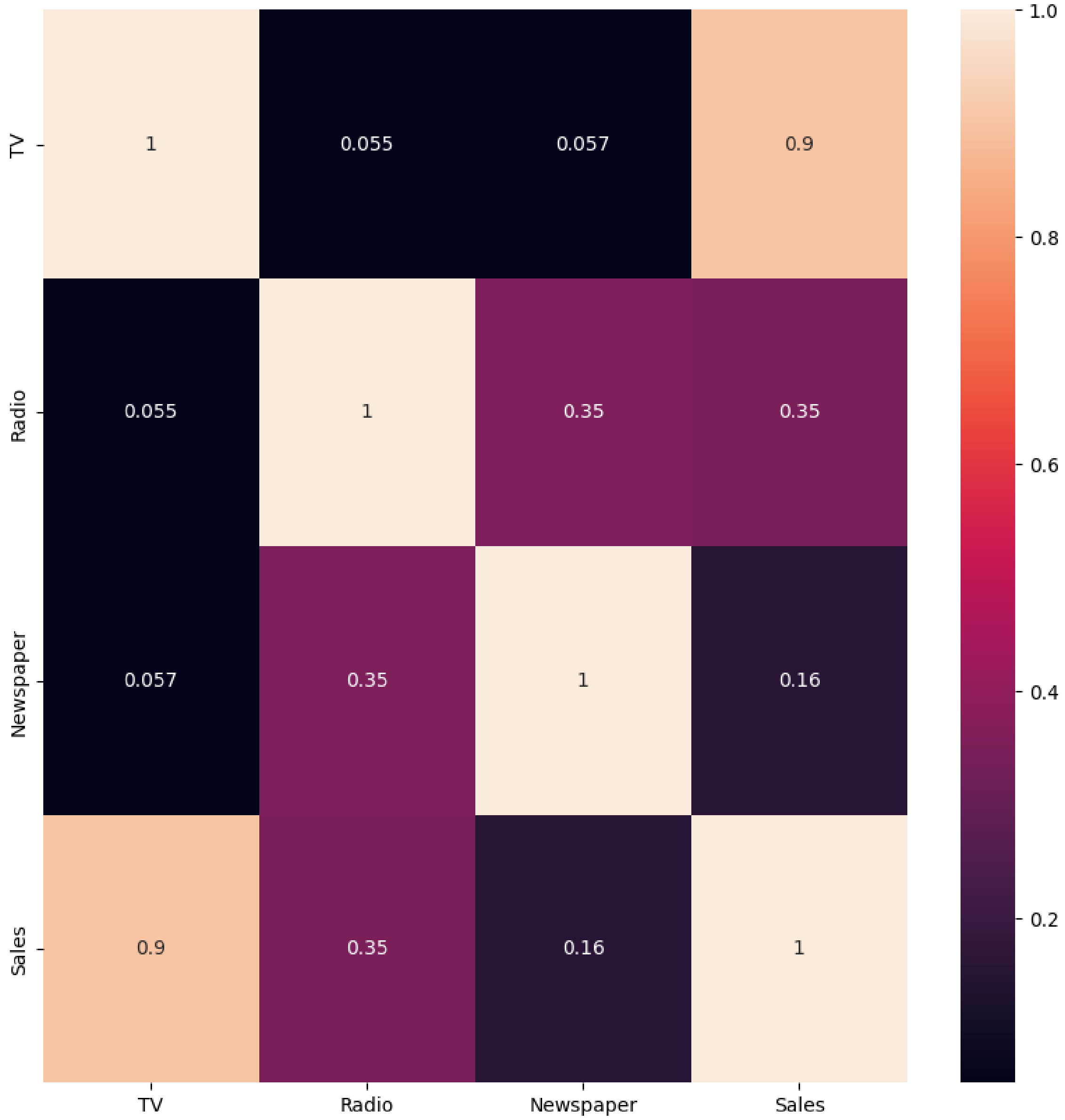
```
<bound method NDFrame.describe of         TV  Radio  Newspaper  Sales
0    230.1   37.8       69.2   22.1
1     44.5   39.3       45.1   10.4
2     17.2   45.9       69.3   12.0
3    151.5   41.3       58.5   16.5
4    180.8   10.8       58.4   17.9
..     ...    ...        ...    ...
195   38.2    3.7       13.8    7.6
196   94.2    4.9        8.1   14.0
197  177.0    9.3        6.4   14.8
198  283.6   42.0       66.2   25.5
199  232.1    8.6        8.7   18.4

[200 rows x 4 columns]>
```

In [139]:

```
plt.figure(figsize = (10, 10))
sns.heatmap(data.corr(), annot = True)
```

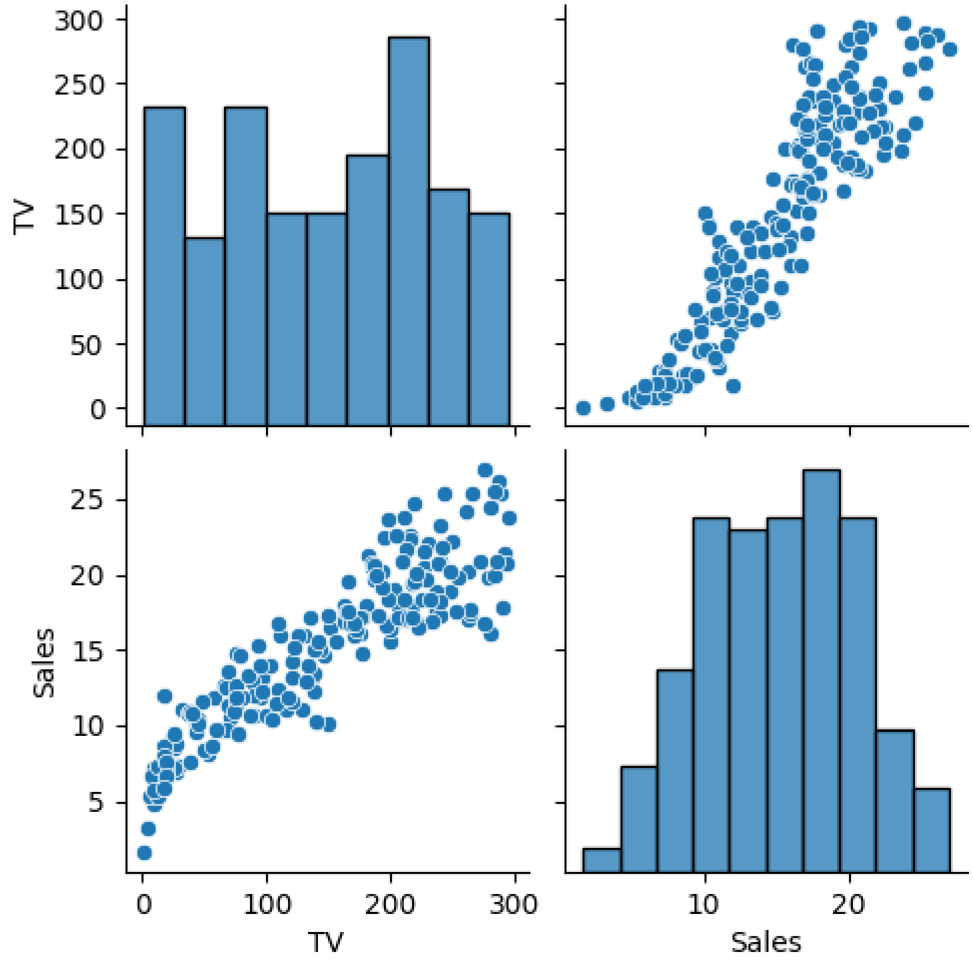
Out[139]:

```
<Axes: >
```



In [140]:

```
data.drop(columns = ["Radio", "Newspaper"], inplace = True)
#pairplot
sns.pairplot(data)
data.Sales = np.log(data.Sales)
```



In [141]:

```
features = data.columns[0:2]
target = data.columns[-1]
#X and y values
x = data[features].values
y = data[target].values
#splot
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=17
print("The dimension of X_train is {}".format(x_train.shape))
print("The dimension of X_test is {}".format(x_test.shape))
#Scale features
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

```
The dimension of X_train is (140, 2)
The dimension of X_test is (60, 2)
```

In [142]:

```
#Model
lr = LinearRegression()
#Fit model
lr.fit(x_train, y_train)
#predict
#prediction = lr.predict(X_test)
#actual
actual = y_test
train_score_lr = lr.score(x_train, y_train)
test_score_lr = lr.score(x_test, y_test)
print("\nLinear Regression Model:\n")
print("The train score for lr model is {}".format(train_score_lr))
print("The test score for lr model is {}".format(test_score_lr))
```

```
Linear Regression Model:

The train score for lr model is 1.0
The test score for lr model is 1.0
```

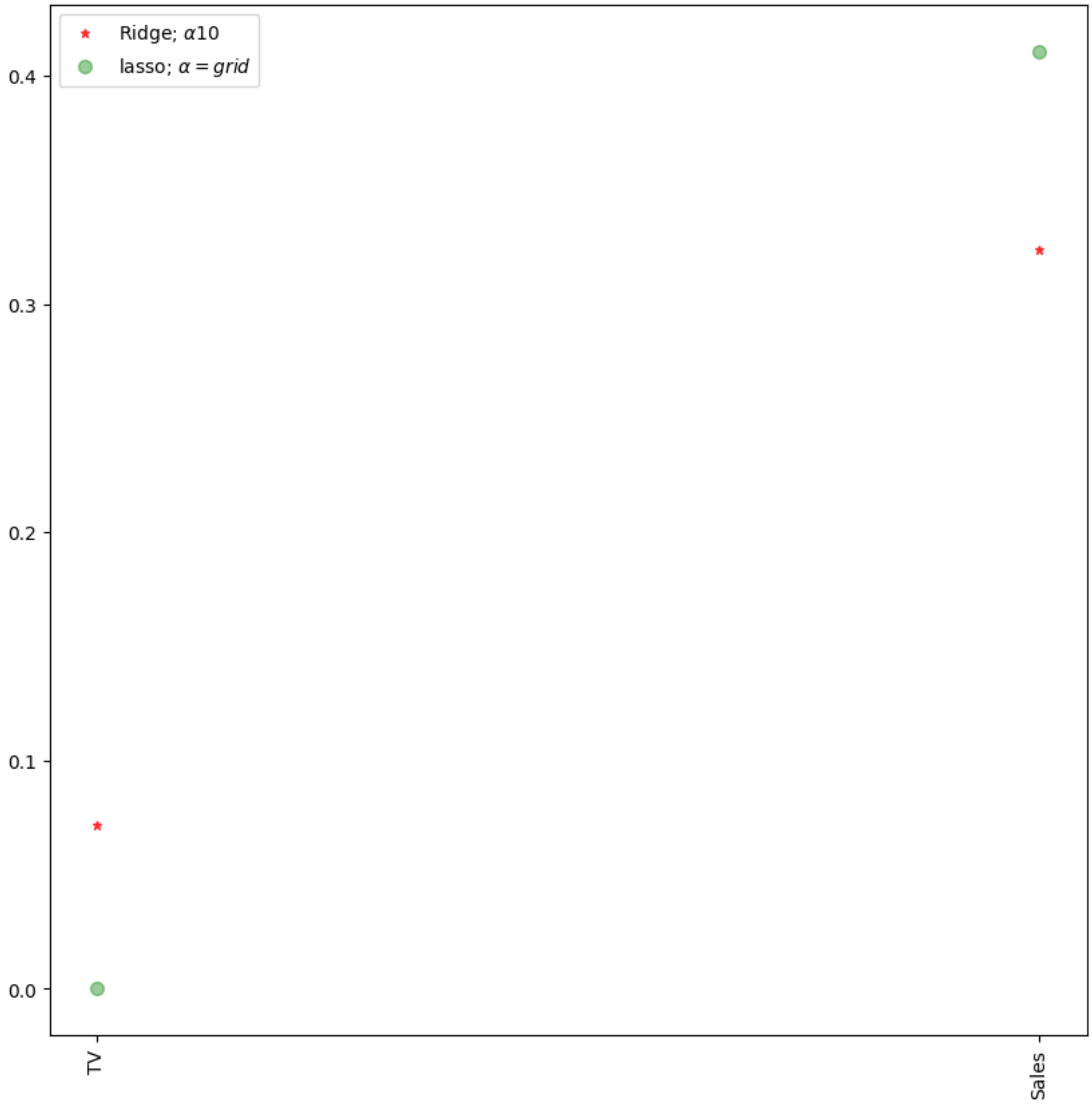
In [143]:

```
#Ridge Regression Model
ridgeReg = Ridge(alpha=10)
ridgeReg.fit(x_train,y_train)
#train and test scorefor ridge regression
train_score_ridge = ridgeReg.score(x_train, y_train)
test_score_ridge = ridgeReg.score(x_test, y_test)
print("\nRidge Model:\n")
print("The train score for ridge model is {}".format(train_score_ridge))
print("The test score for ridge model is {}".format(test_score_ridge))
```

```
Ridge Model:

The train score for ridge model is 0.9902871391941609
The test score for ridge model is 0.984426628514122
```

In [144]:

```
plt.figure(figsize = (10, 10))
plt.plot(features,ridgeReg.coef_,alpha=0.7,linestyle='none',marker='*',markersize=5,colo
plt.plot(features,lr.coef_,alpha=0.4,linestyle='none',marker='o',markersize=7,color='gre
plt.xticks(rotation = 90)
plt.legend()
plt.show()
```



In [145]:

```
#Lasso regression model
print("\nLasso Model: \n")
lasso = Lasso(alpha = 10)
lasso.fit(x_train,y_train)
train_score_ls =lasso.score(x_train,y_train)
test_score_ls =lasso.score(x_test,y_test)
print("The train score for ls model is {}".format(train_score_ls))
print("The test score for ls model is {}".format(test_score_ls))
```

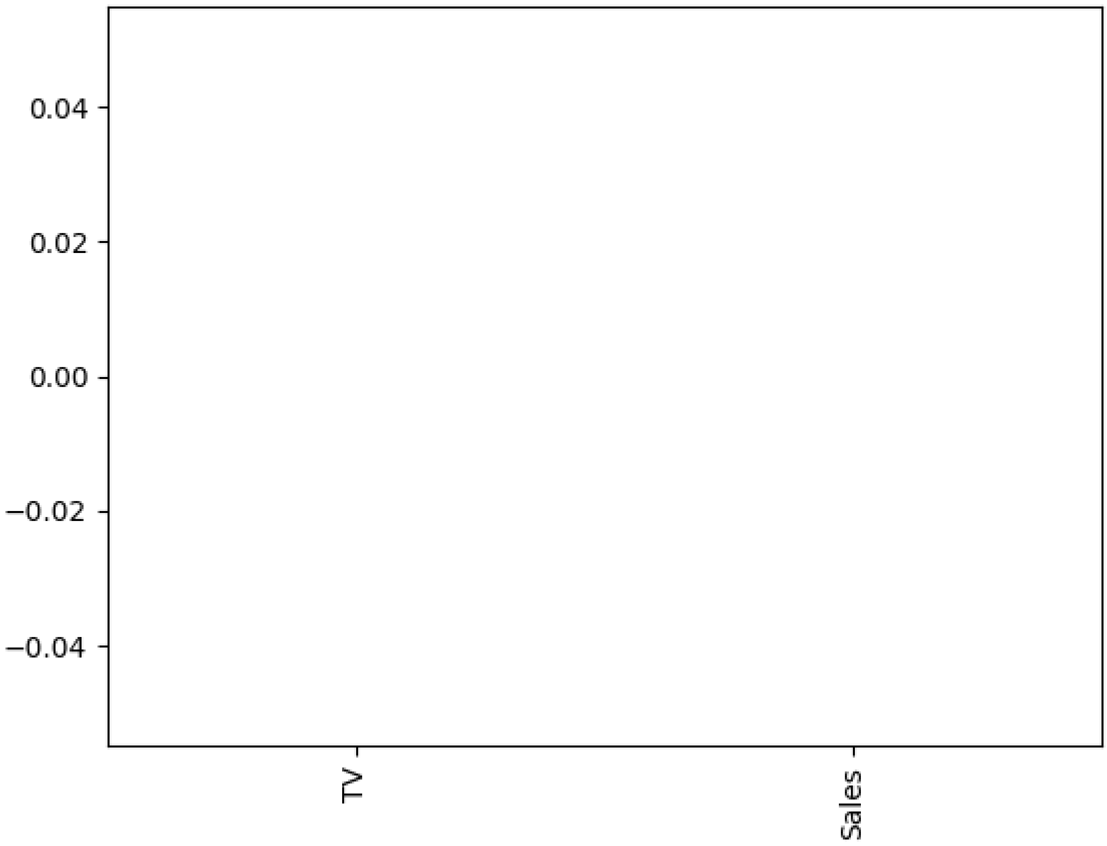
```
Lasso Model: 

The train score for ls model is 0.0
The test score for ls model is -0.0042092253233847465
```

In [146]:

```
pd.Series(lasso.coef_, features).sort_values(ascending = True).plot(kind = "bar")
```

Out[146]:

```
<Axes: >
```



In [147]:

```
#Using the linear CV model
from sklearn.linear_model import LassoCV
#Lasso Cross validation
lasso_cv = LassoCV(alphas = [0.0001, 0.001,0.01, 0.1, 1, 10], random_state=0).fit(x_trai
#score
print(lasso_cv.score(x_train, y_train))
print(lasso_cv.score(x_test, y_test))
```

```
0.9999999343798134
0.9999999152638072
```

ELASTIC NET REGRESSION

In [151]:

```
from sklearn.linear_model import ElasticNet
regr=ElasticNet()
regr.fit(x,y)
print(regr.coef_)
print(regr.intercept_)
```

```
[0.00417976 0.        ]
2.026383919311004
```

In [152]:

```
y_pred_elastic=regr.predict(x_train)
```

In [153]:

```
mean_squared_error=np.mean((y_pred_elastic-y_train)**2)
print("Mean Squared Error on test set",mean_squared_error)
```

```
Mean Squared Error on test set 0.5538818050142158
```

In [154]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge, RidgeCV, Lasso
from sklearn.preprocessing import StandardScaler
```

In [155]:

```
data=pd.read_csv(r"C:\Users\chila\Downloads\fiat500_VehicleSelection_Dataset.csv")
data
```

Out[155]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	
0	1	lounge	51	882	25000	1	44.907242	8.611
1	2	pop	51	1186	32500	1	45.666359	12.241
2	3	sport	74	4658	142228	1	45.503300	11.417
3	4	lounge	51	2739	160000	1	40.633171	17.634
4	5	pop	73	3074	106880	1	41.903221	12.495
...	...	...	...	...	...	...	...	
1533	1534	sport	51	3712	115280	1	45.069679	7.704
1534	1535	lounge	74	3835	112000	1	45.845692	8.666
1535	1536	pop	51	2223	60457	1	45.481541	9.413
1536	1537	lounge	51	2557	80750	1	45.000702	7.682
1537	1538	pop	51	1766	54276	1	40.323410	17.568

1538 rows × 9 columns

In [156]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1538 entries, 0 to 1537
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ID               1538 non-null   int64  
 1   model            1538 non-null   object 
 2   engine_power     1538 non-null   int64  
 3   age_in_days      1538 non-null   int64  
 4   km               1538 non-null   int64  
 5   previous_owners  1538 non-null   int64  
 6   lat              1538 non-null   float64
 7   lon              1538 non-null   float64
 8   price            1538 non-null   int64  
dtypes: float64(2), int64(6), object(1)
memory usage: 108.3+ KB
```

In [157]:

```
data=data[['engine_power','price']]
data.columns=['Eng','pri']
```

In [158]:

```
data.head()
```

Out[158]:

	Eng	pri
0	51	8900
1	51	8800
2	74	4200
3	51	6000
4	73	5700

In [159]:

```
data.tail()
```

Out[159]:

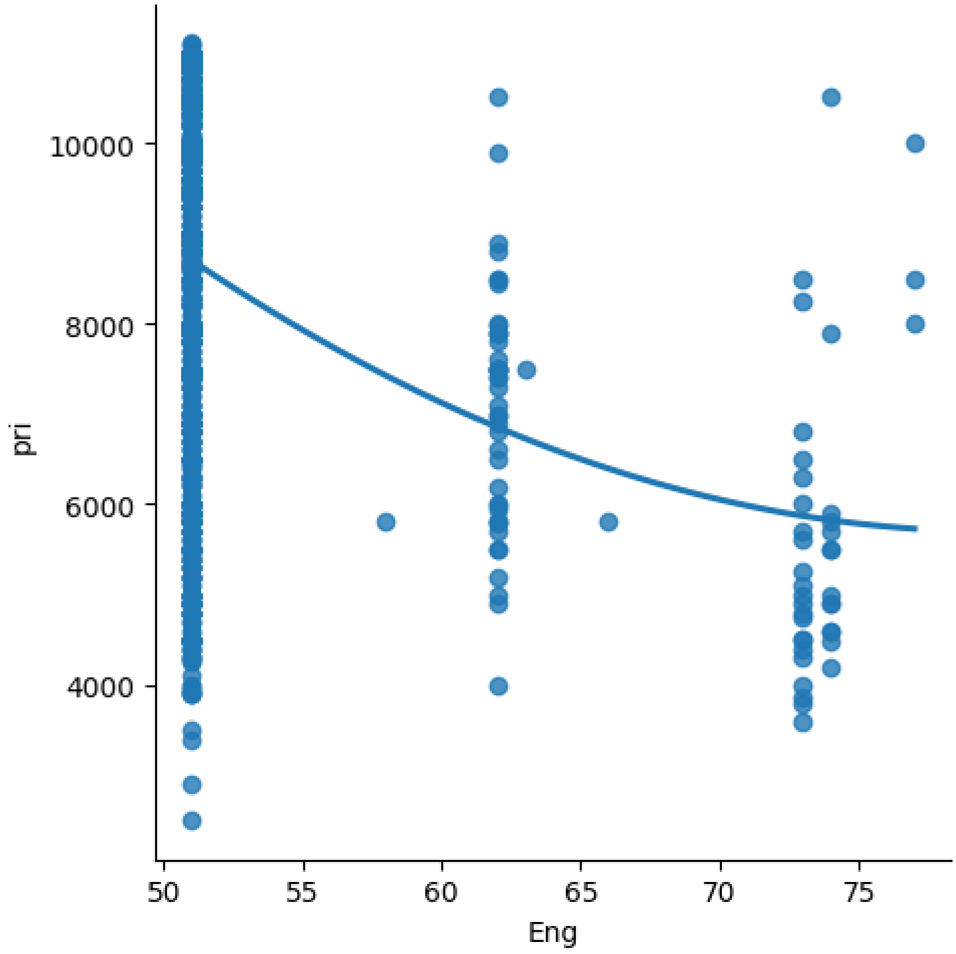
	Eng	pri
1533	51	5200
1534	74	4600
1535	51	7500
1536	51	5990
1537	51	7900

In [160]:

```
sns.lmplot(x='Eng',y='pri',data=data,order=2,ci=None)
```

Out[160]:

```
<seaborn.axisgrid.FacetGrid at 0x260b1488490>
```



In [161]:

```
data.describe()
```

Out[161]:

	Eng	pri
count	1538.000000	1538.000000
mean	51.904421	8576.003901
std	3.988023	1939.958641
min	51.000000	2500.000000
25%	51.000000	7122.500000
50%	51.000000	9000.000000
75%	51.000000	10000.000000
max	77.000000	11100.000000

In [162]:

```
data.fillna(method='ffill')
```

Out[162]:

	Eng	pri
0	51	8900
1	51	8800
2	74	4200
3	51	6000
4	73	5700
...	...	...
1533	51	5200
1534	74	4600
1535	51	7500
1536	51	5990
1537	51	7900

1538 rows × 2 columns

In [163]:

```
x=np.array(data['Eng']).reshape(-1,1)
y=np.array(data['pri']).reshape(-1,1)
```

In [164]:

```
data.dropna(inplace=True)
```

```
C:\Users\chila\AppData\Local\Temp\ipykernel_15008\1368182302.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  data.dropna(inplace=True)
```

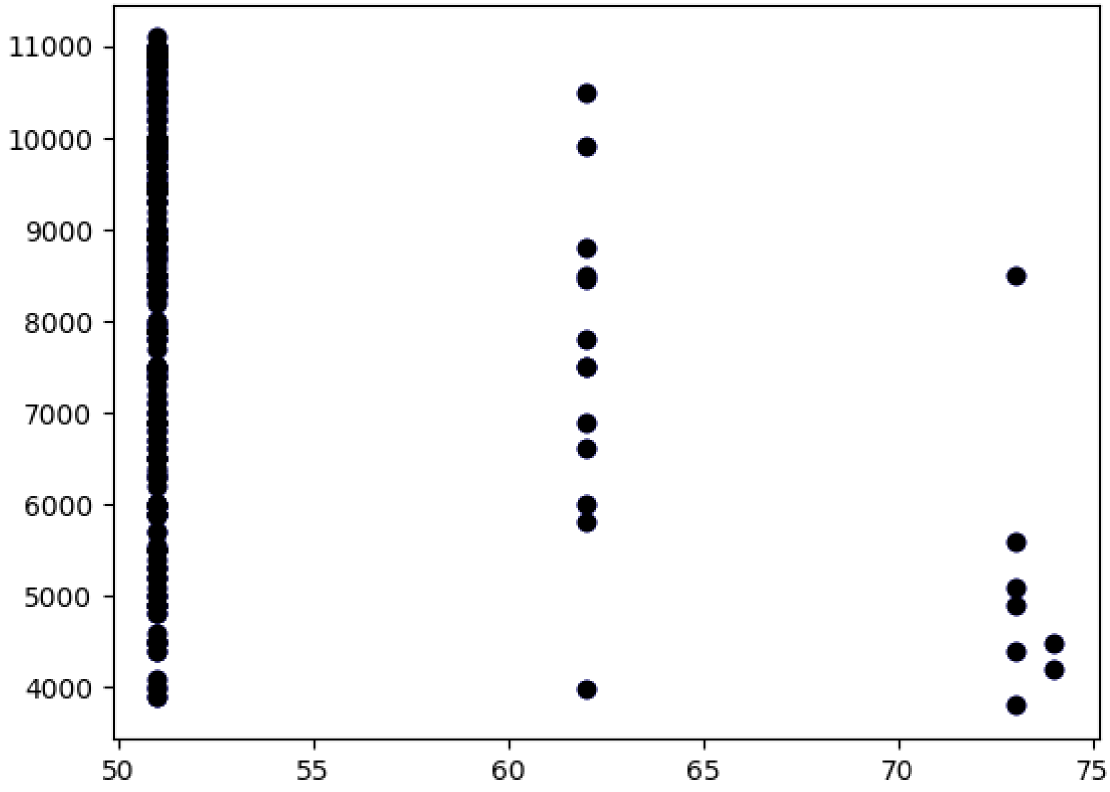
In [165]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25)
#splitting the dataset into training and testing dataset
regr=LinearRegression()
regr.fit(x_train,y_train)
print(regr.score(x_test,y_test))
```

```
0.07262451631405753
```

In [166]:

```
y_pred=regr.predict(x_test)
plt.scatter(x_test,y_test,color='b')
plt.scatter(x_test,y_test,color='k')
plt.show()
```

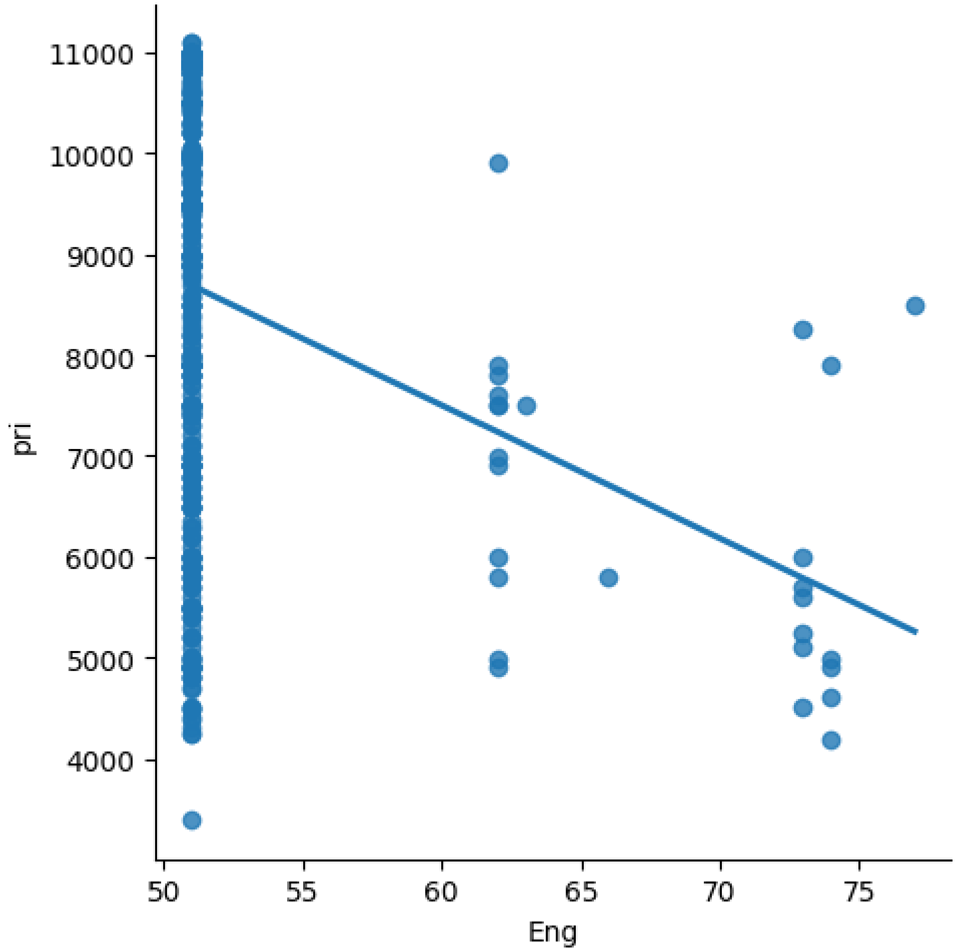


In [167]:

```
df500=data[:][:500]
sns.lmplot(x='Eng',y='pri',data=df500,order=1,ci=None)
```

Out[167]:

```
<seaborn.axisgrid.FacetGrid at 0x260b1886590>
```



Dataset2

In [168]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing, svm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
```

In [169]:

```
df=pd.read_csv(r"C:\Users\chila\Downloads\bottle.csv.zip")
df
```

```
C:\Users\chila\AppData\Local\Temp\ipykernel_15008\2541466974.py:1: DtypeWa
rning: Columns (47,73) have mixed types. Specify dtype option on import or
set low_memory=False.
  df=pd.read_csv(r"C:\Users\chila\Downloads\bottle.csv.zip")
```

Out[169]:

	Cst_Cnt	Btl_Cnt	Sta_ID	Depth_ID	Depthm	T_degC	Salnty	O2ml_L	STheta
0	1	1	054.0 056.0	19-4903CR-HY-060-0930-05400560-0000A-3	0	10.500	33.4400	NaN	25.64900
1	1	2	054.0 056.0	19-4903CR-HY-060-0930-05400560-0008A-3	8	10.460	33.4400	NaN	25.65600
2	1	3	054.0 056.0	19-4903CR-HY-060-0930-05400560-0010A-7	10	10.460	33.4370	NaN	25.65400
3	1	4	054.0 056.0	19-4903CR-HY-060-0930-05400560-0019A-3	19	10.450	33.4200	NaN	25.64300
4	1	5	054.0 056.0	19-4903CR-HY-060-0930-05400560-0020A-7	20	10.450	33.4210	NaN	25.64300
...	...	...	...	...	...	...	...	...	...
864858	34404	864859	093.4 026.4	20-1611SR-MX-310-2239-09340264-0000A-7	0	18.744	33.4083	5.805	23.87055
864859	34404	864860	093.4 026.4	20-1611SR-MX-310-2239-09340264-0002A-3	2	18.744	33.4083	5.805	23.87072
864860	34404	864861	093.4 026.4	20-1611SR-MX-310-2239-09340264-0005A-3	5	18.692	33.4150	5.796	23.88911
864861	34404	864862	093.4 026.4	20-1611SR-MX-310-2239-09340264-	10	18.161	33.4062	5.816	24.01426

In [170]:

```
df.head()
```

Out[170]:

	Cst_Cnt	Btl_Cnt	Sta_ID	Depth_ID	Depthm	T_degC	Salnty	O2ml_L	STheta	O2Sat	.
0	1	1	054.0 056.0	19-4903CR-HY-060-0930-05400560-0000A-3	0	10.50	33.440	NaN	25.649	NaN	.
1	1	2	054.0 056.0	19-4903CR-HY-060-0930-05400560-0008A-3	8	10.46	33.440	NaN	25.656	NaN	.
2	1	3	054.0 056.0	19-4903CR-HY-060-0930-05400560-0010A-7	10	10.46	33.437	NaN	25.654	NaN	.
3	1	4	054.0 056.0	19-4903CR-HY-060-0930-05400560-0019A-3	19	10.45	33.420	NaN	25.643	NaN	.
4	1	5	054.0 056.0	19-4903CR-HY-060-0930-05400560-0020A-7	20	10.45	33.421	NaN	25.643	NaN	.

5 rows × 74 columns

	Cst_Cnt	Btl_Cnt	Sta_ID	Depth_ID	Depthm	T_degC	Salnty	O2ml_L	STheta
864862	34404	864863	093.4 026.4	20-1611SR-MX-310-2239-09340264-0015A-3	15	17.533	33.3880	5.774	24.15297

864863 rows × 74 columns

```
In [171]:
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 864863 entries, 0 to 864862
Data columns (total 74 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   Cst_Cnt              864863 non-null  int64  
 1   Btl_Cnt              864863 non-null  int64  
 2   Sta_ID               864863 non-null  object 
 3   Depth_ID             864863 non-null  object 
 4   Depthm               864863 non-null  int64  
 5   T_degC               853900 non-null  float64
 6   Salnty               817509 non-null  float64
 7   O2ml_L               696201 non-null  float64
 8   STheta               812174 non-null  float64
 9   O2Sat                661274 non-null  float64
 10  Oxy_µmol/Kg          661268 non-null  float64
 11  BtlNum               118667 non-null  float64
 12  RecInd               864863 non-null  int64  
 13  T_prec               853900 non-null  float64
 14  T_qual               23127 non-null   float64
 15  S_prec               817509 non-null  float64
 16  S_qual               74914 non-null   float64
 17  P_qual               673755 non-null  float64
 18  O_qual               184676 non-null  float64
 19  SThtaq               65823 non-null   float64
 20  O2Satq               217797 non-null  float64
 21  ChlorA               225272 non-null  float64
 22  Chlqua               639166 non-null  float64
 23  Phaeop               225271 non-null  float64
 24  Phaqua               639170 non-null  float64
 25  PO4uM                413317 non-null  float64
 26  PO4q                 451786 non-null  float64
 27  SiO3uM               354091 non-null  float64
 28  SiO3qu               510866 non-null  float64
 29  NO2uM                337576 non-null  float64
 30  NO2q                 529474 non-null  float64
 31  NO3uM                337403 non-null  float64
 32  NO3q                 529933 non-null  float64
 33  NH3uM                64962 non-null   float64
 34  NH3q                 808299 non-null  float64
 35  C14As1               14432 non-null   float64
 36  C14A1p               12760 non-null   float64
 37  C14A1q               848605 non-null  float64
 38  C14As2               14414 non-null   float64
 39  C14A2p               12742 non-null   float64
 40  C14A2q               848623 non-null  float64
 41  DarkAs               22649 non-null   float64
 42  DarkAp               20457 non-null   float64
 43  DarkAq               840440 non-null  float64
 44  MeanAs               22650 non-null   float64
 45  MeanAp               20457 non-null   float64
 46  MeanAq               840439 non-null  float64
 47  IncTim               14437 non-null   object 
 48  LightP               18651 non-null   float64
 49  R_Depth              864863 non-null  float64
 50  R_TEMP               853900 non-null  float64
 51  R_POTEMP             818816 non-null  float64
 52  R_SALINITY           817509 non-null  float64
 53  R_SIGMA              812007 non-null  float64
 54  R_SVA                812092 non-null  float64
 55  R_DYNHT              818206 non-null  float64
 56  R_O2                 696201 non-null  float64
 57  R_O2Sat              666448 non-null  float64
 58  R_SIO3               354099 non-null  float64
 59  R_PO4                413325 non-null  float64
 60  R_NO3                337411 non-null  float64
 61  R_NO2                337584 non-null  float64
 62  R_NH4                64982 non-null   float64
 63  R_CHLA               225276 non-null  float64
 64  R_PHAEO              225275 non-null  float64
 65  R_PRES               864863 non-null  int64
 66  R_SAMP               122006 non-null  float64
 67  DIC1                 1999 non-null    float64
 68  DIC2                 224 non-null     float64
 69  TA1                  2084 non-null    float64
 70  TA2                  234 non-null     float64
 71  pH2                  10 non-null      float64
 72  pH1                  84 non-null      float64
 73  DIC Quality Comment  55 non-null      object
dtypes: float64(65), int64(5), object(4)
memory usage: 488.3+ MB
```

In [172]:

```
df.describe()
```

Out[172]:

	Cst_Cnt	Btl_Cnt	Depthm	T_degC	Salnty	O
count	864863.000000	864863.000000	864863.000000	853900.000000	817509.000000	696201.0
mean	17138.790958	432432.000000	226.831951	10.799677	33.840350	3.3
std	10240.949817	249664.587269	316.050259	4.243825	0.461843	2.0
min	1.000000	1.000000	0.000000	1.440000	28.431000	-0.0
25%	8269.000000	216216.500000	46.000000	7.680000	33.488000	1.3
50%	16848.000000	432432.000000	125.000000	10.060000	33.863000	3.4
75%	26557.000000	648647.500000	300.000000	13.880000	34.196900	5.5
max	34404.000000	864863.000000	5351.000000	31.140000	37.034000	11.1

8 rows × 70 columns

In [173]:

```
df.isna().any()
```

Out[173]:

```
Cst_Cnt                False
Btl_Cnt                False
Sta_ID                 False
Depth_ID               False
Depthm                 False
                       ...  
TA1                     True
TA2                     True
pH2                     True
pH1                     True
DIC Quality Comment     True
Length: 74, dtype: bool
```

In [174]:

```
df.isnull().sum()
```

Out[174]:

```
Cst_Cnt                     0
Btl_Cnt                     0
Sta_ID                      0
Depth_ID                    0
Depthm                      0
                        ...  
TA1                    862779
TA2                    864629
pH2                    864853
pH1                    864779
DIC Quality Comment    864808
Length: 74, dtype: int64
```

In [175]:

```
df=df[['Salnty', 'T_degC']]
df.columns=['Sal', 'Temp']
```

In [176]:

```
df.head(20)
```

Out[176]:

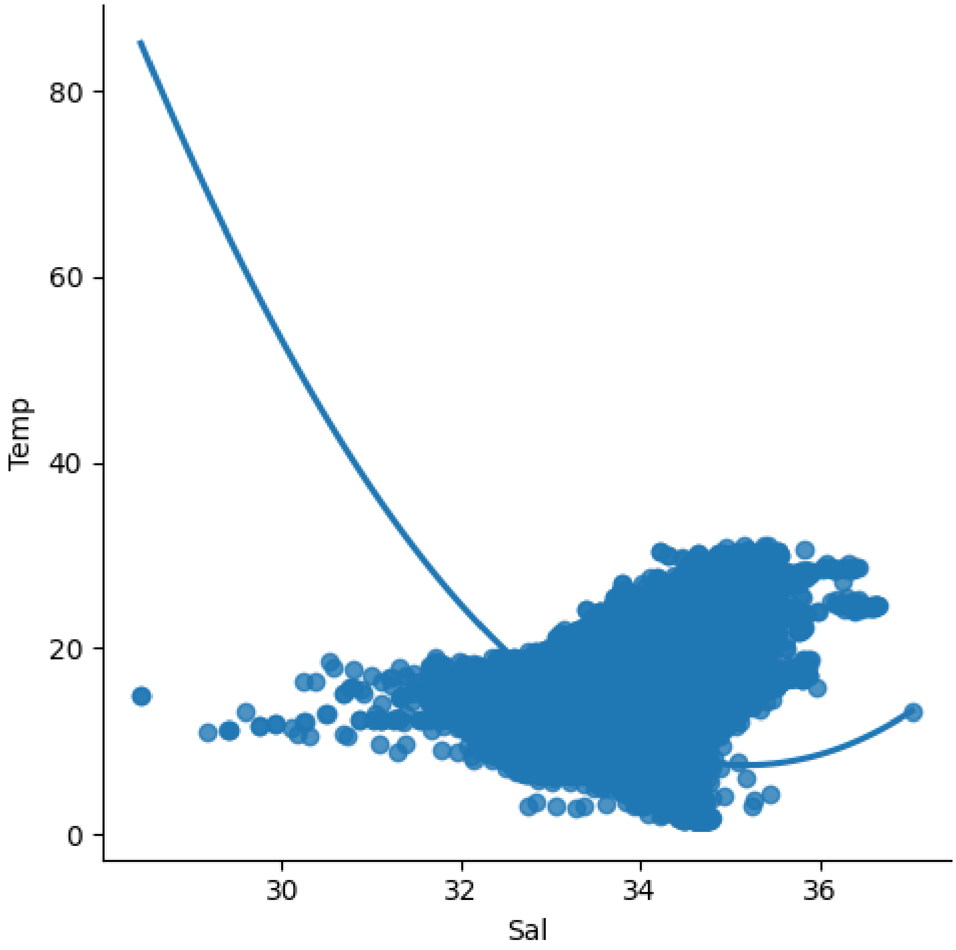
	Sal	Temp
0	33.440	10.50
1	33.440	10.46
2	33.437	10.46
3	33.420	10.45
4	33.421	10.45
5	33.431	10.45
6	33.440	10.45
7	33.424	10.24
8	33.420	10.06
9	33.494	9.86
10	33.510	9.83
11	33.580	9.67
12	33.640	9.50
13	33.689	9.32
14	33.847	8.76
15	33.860	8.71
16	33.876	8.53
17	NaN	8.45
18	33.926	8.26
19	33.980	7.96

In [177]:

```
sns.lmplot(x='Sal',y='Temp',data=df,order=2,ci=None)
```

Out[177]:

```
<seaborn.axisgrid.FacetGrid at 0x260b18867a0>
```



In [178]:

```
df.fillna (method='ffill')
```

Out[178]:

	Sal	Temp
0	33.4400	10.500
1	33.4400	10.460
2	33.4370	10.460
3	33.4200	10.450
4	33.4210	10.450
...	...	...
864858	33.4083	18.744
864859	33.4083	18.744
864860	33.4150	18.692
864861	33.4062	18.161
864862	33.3880	17.533

864863 rows × 2 columns

In [179]:

```
df.fillna(value=0,inplace=True)
```

```
C:\Users\chila\AppData\Local\Temp\ipykernel_15008\709118144.py:1: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df.fillna(value=0,inplace=True)
```

In [180]:

```
x=np.array(df['Sal']).reshape(-1,1)
y=np.array(df[ 'Temp']).reshape(-1,1)
```

In [181]:

```
df.dropna (inplace=True)
```

```
C:\Users\chila\AppData\Local\Temp\ipykernel_15008\1939022369.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df.dropna (inplace=True)
```

In [182]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25)
```

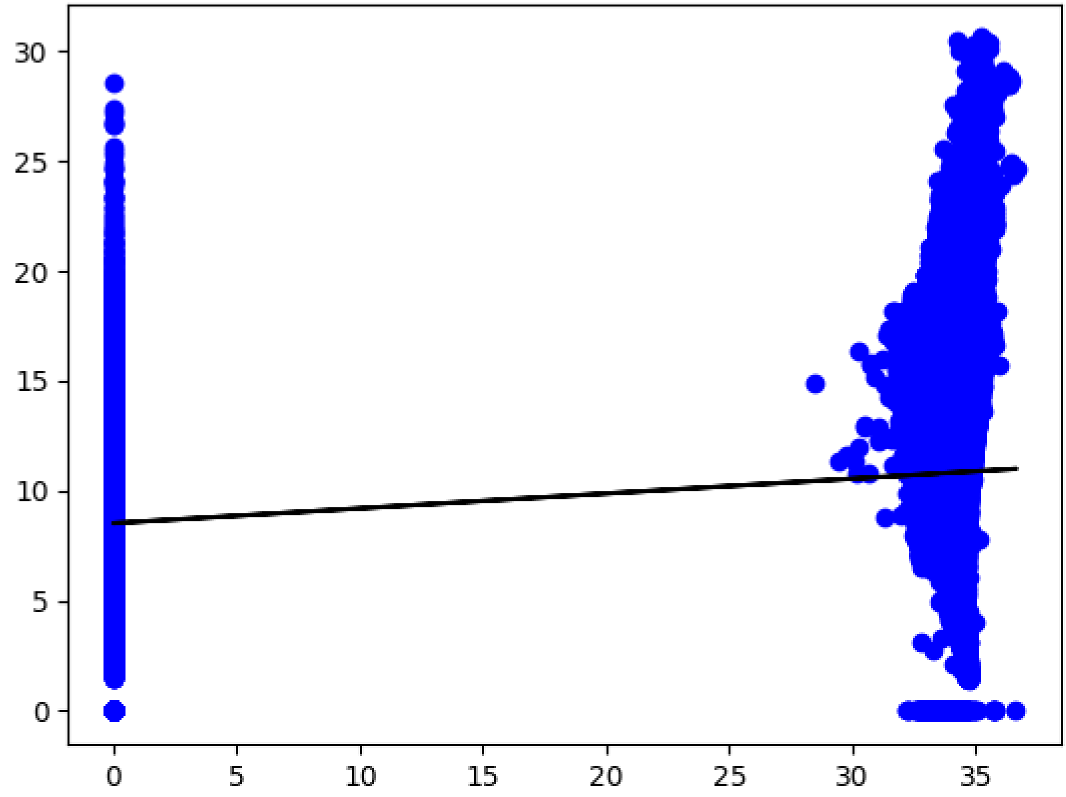
In [183]:

```
regr=LinearRegression()
regr.fit(x_train,y_train)
print(regr.score (x_test,y_test))
```

```
0.014223284614636067
```

In [184]:

```
y_pred=regr.predict(x_test)
plt.scatter(x_test,y_test,color='b')
plt.plot(x_test, y_pred, color='k')
plt.show()
```

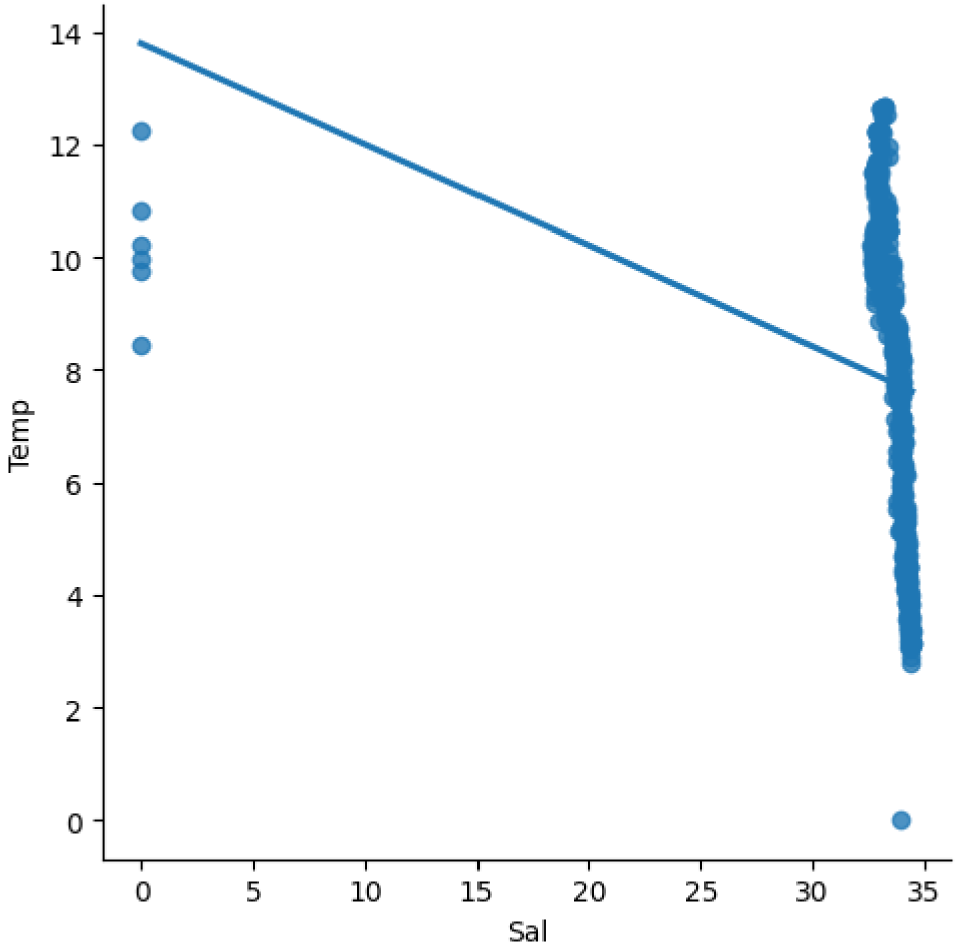


In [185]:

```
df500=df[:] [:500]
sns.lmplot(x='Sal', y= 'Temp', data=df500,order=1, ci=None)
```

Out[185]:

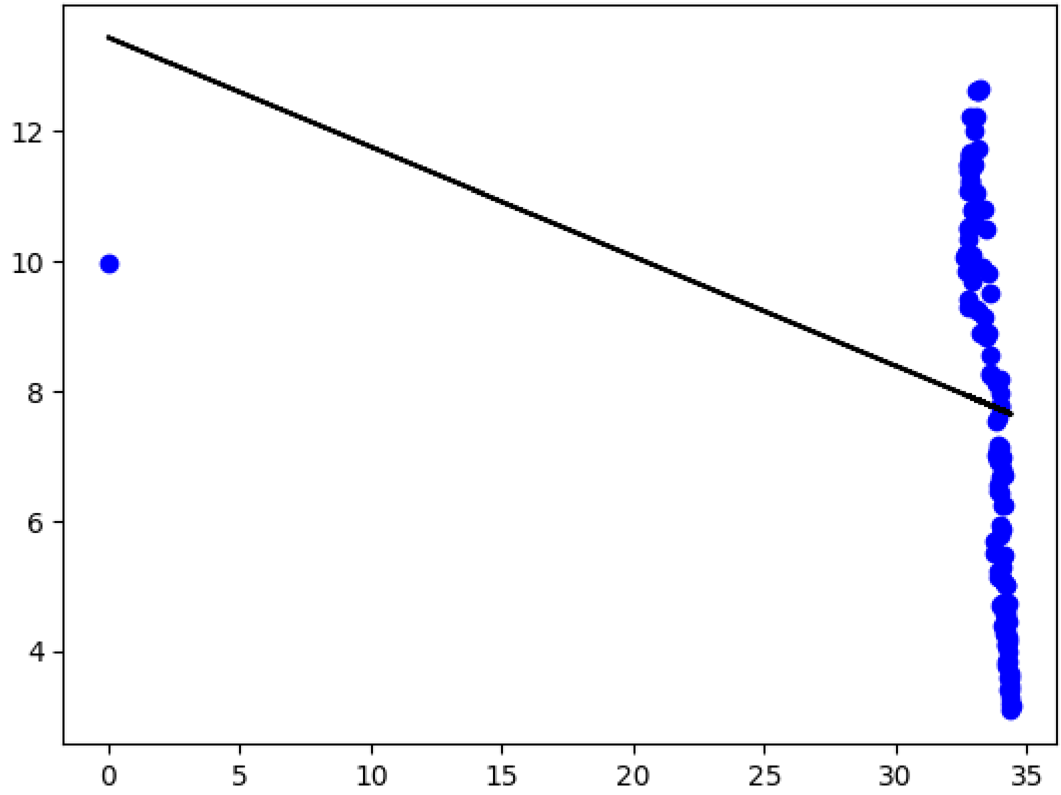
```
<seaborn.axisgrid.FacetGrid at 0x260b1347a90>
```



In [186]:

```
df500.fillna (method='ffill', inplace=True)
x=np.array(df500['Sal']).reshape(-1,1)
y=np.array(df500 [ 'Temp']).reshape(-1,1)
df500.dropna (inplace=True)
x_train,x_test,y_train, y_test=train_test_split(x,y,test_size=0.25)
regr=LinearRegression()
regr.fit(x_train,y_train)
print("Regression: ", regr.score (x_test,y_test))
y_pred=regr.predict(x_test)
plt.scatter(x_test,y_test,color='b')
plt.plot(x_test, y_pred, color='k')
plt.show()
```

```
Regression:  0.053364290956062765
```



In [187]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
model = LinearRegression()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
r2=r2_score(y_test,y_pred)
print("R2 score:",r2)
```

```
R2 score: 0.053364290956062765
```

In [188]:

```
#conclusion: Linear regression is not fit for the model
```

In [189]:

```
features = df.columns[0:2]
target = df.columns[-1]
#X and y values
x = df[features].values
y = df[target].values
#splot
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=17
print("The dimension of X_train is {}".format(x_train.shape))
print("The dimension of X_test is {}".format(x_test.shape))
#Scale features
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

```
The dimension of X_train is (605404, 2)
The dimension of X_test is (259459, 2)
```

Elastic Net

In [190]:

```
from sklearn.linear_model import ElasticNet
regr=ElasticNet()
regr.fit(x,y)
print(regr.coef_)
print(regr.intercept_)
```

```
[0.         0.94934511]
0.5401219631068042
```

In [191]:

```
y_pred_elastic=regr.predict(x_train)
```

In [192]:

```
mean_squared_error=np.mean((y_pred_elastic-y_train)**2)
print("Mean Squared Error on test set",mean_squared_error)
```

```
Mean Squared Error on test set 114.40984808659205
```

Dataset 3

In [210]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge, RidgeCV, Lasso
from sklearn.preprocessing import StandardScaler
```

In [211]:

```
d=pd.read_csv(r"C:\Users\chila\Downloads\fiat500_VehicleSelection_Dataset.csv")
d
```

Out[211]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	
0	1	lounge	51	882	25000	1	44.907242	8.611
1	2	pop	51	1186	32500	1	45.666359	12.241
2	3	sport	74	4658	142228	1	45.503300	11.417
3	4	lounge	51	2739	160000	1	40.633171	17.634
4	5	pop	73	3074	106880	1	41.903221	12.495
...	...	...	...	...	...	...	...	
1533	1534	sport	51	3712	115280	1	45.069679	7.704
1534	1535	lounge	74	3835	112000	1	45.845692	8.666
1535	1536	pop	51	2223	60457	1	45.481541	9.413
1536	1537	lounge	51	2557	80750	1	45.000702	7.682
1537	1538	pop	51	1766	54276	1	40.323410	17.568

1538 rows × 9 columns

In [212]:

```
d.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1538 entries, 0 to 1537
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ID               1538 non-null   int64  
 1   model            1538 non-null   object 
 2   engine_power     1538 non-null   int64  
 3   age_in_days      1538 non-null   int64  
 4   km               1538 non-null   int64  
 5   previous_owners  1538 non-null   int64  
 6   lat              1538 non-null   float64
 7   lon              1538 non-null   float64
 8   price            1538 non-null   int64  
dtypes: float64(2), int64(6), object(1)
memory usage: 108.3+ KB
```

In [214]:

```
d.head()
```

Out[214]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon
0	1	lounge	51	882	25000	1	44.907242	8.611560
1	2	pop	51	1186	32500	1	45.666359	12.241890
2	3	sport	74	4658	142228	1	45.503300	11.417840
3	4	lounge	51	2739	160000	1	40.633171	17.634609
4	5	pop	73	3074	106880	1	41.903221	12.495650

In [215]:

```
d.tail()
```

Out[215]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lo
1533	1534	sport	51	3712	115280	1	45.069679	7.704
1534	1535	lounge	74	3835	112000	1	45.845692	8.666
1535	1536	pop	51	2223	60457	1	45.481541	9.413
1536	1537	lounge	51	2557	80750	1	45.000702	7.682
1537	1538	pop	51	1766	54276	1	40.323410	17.568

In [216]:

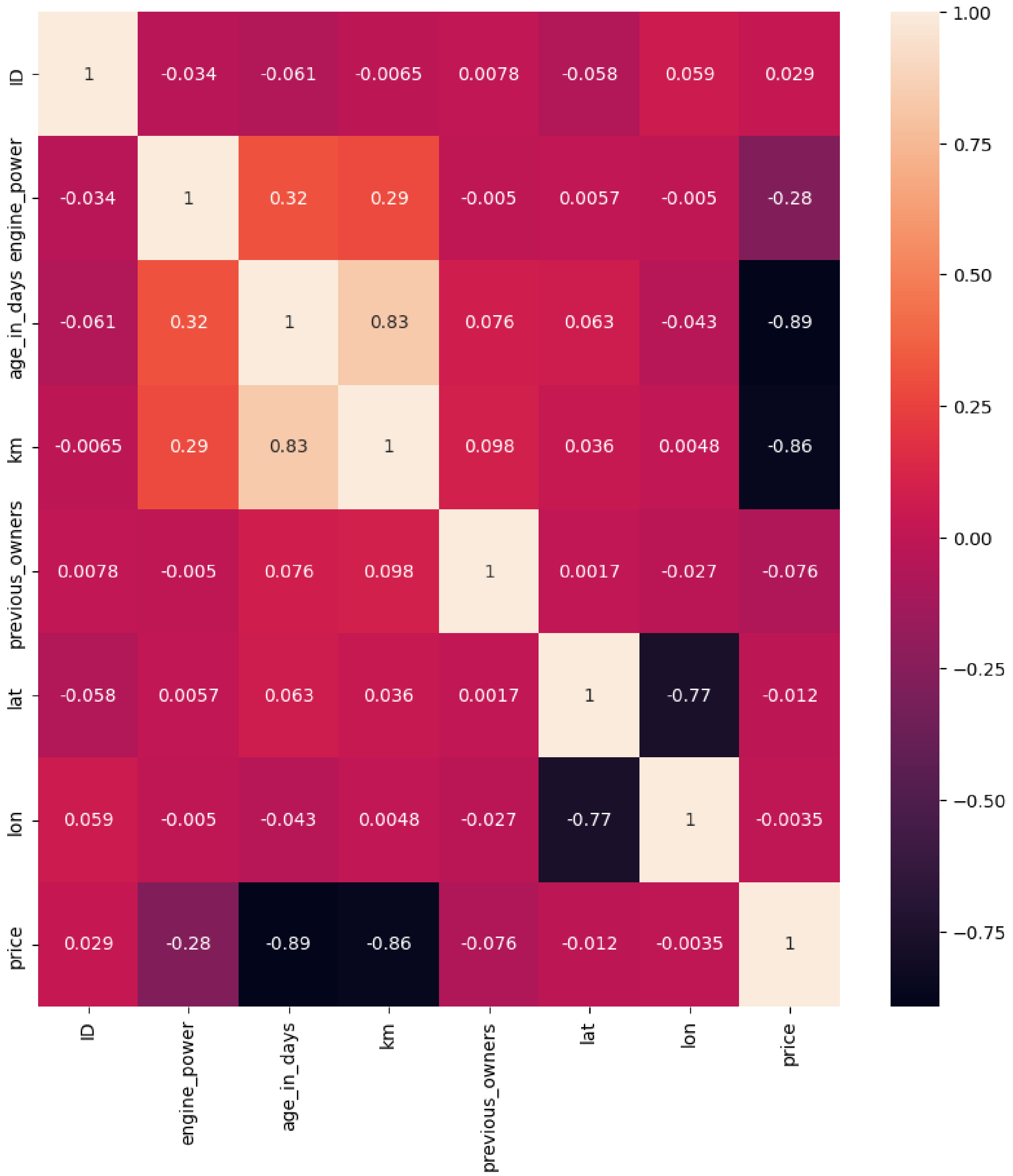
```
d.drop(columns=["model"],inplace=True)
```

In [217]:

```
plt.figure(figsize=(10,10))
sns.heatmap(d.corr(),annot = True)
```

Out[217]:

```
<Axes: >
```



In [219]:

```
features = d.columns[0:2]
target = d.columns[-1]
#X and y values
x = d[features].values
y = d[target].values
#splot
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=17
print("The dimension of X_train is {}".format(x_train.shape))
print("The dimension of X_test is {}".format(x_test.shape))
#Scale features
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

```
The dimension of X_train is (1076, 2)
The dimension of X_test is (462, 2)
```

In [220]:

```
#Model
lr = LinearRegression()
#Fit model
lr.fit(x_train, y_train)
#predict
#prediction = lr.predict(X_test)
#actual
actual = y_test
train_score_lr = lr.score(x_train, y_train)
test_score_lr = lr.score(x_test, y_test)
print("\nLinear Regression Model:\n")
print("The train score for lr model is {}".format(train_score_lr))
print("The test score for lr model is {}".format(test_score_lr))
```

```

Linear Regression Model:

The train score for lr model is 0.07448634159905865
The test score for lr model is 0.07913288661070894
```

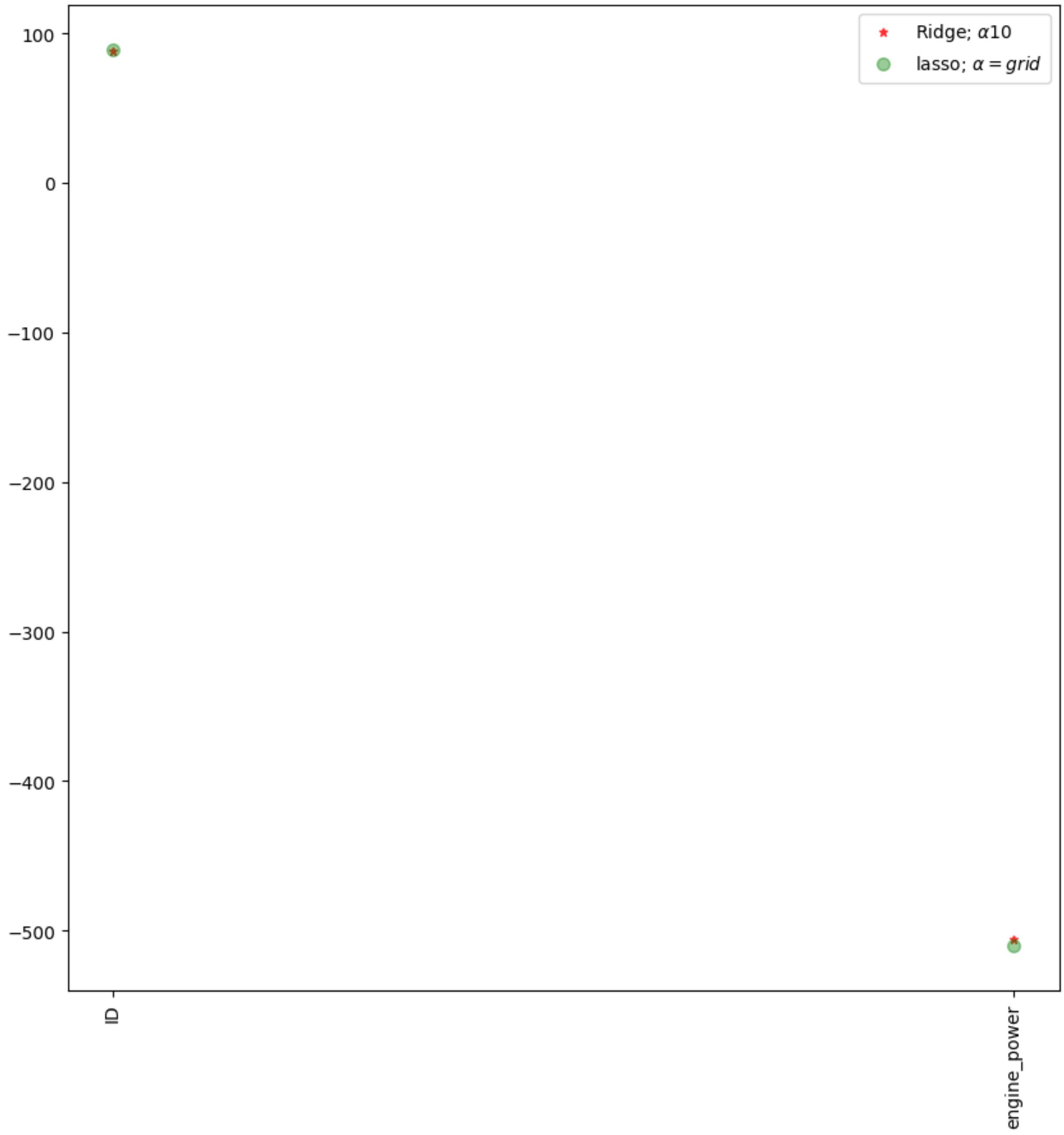
In [221]:

```
#Ridge Regression Model
ridgeReg = Ridge(alpha=10)
ridgeReg.fit(x_train,y_train)
#train and test scorefor ridge regression
train_score_ridge = ridgeReg.score(x_train, y_train)
test_score_ridge = ridgeReg.score(x_test, y_test)
print("\nRidge Model:\n")
print("The train score for ridge model is {}".format(train_score_ridge))
print("The test score for ridge model is {}".format(test_score_ridge))
```

```

Ridge Model:

The train score for ridge model is 0.07448028989896427
The test score for ridge model is 0.07885996726883049
```

In [222]:

```
plt.figure(figsize = (10, 10))
plt.plot(features,ridgeReg.coef_,alpha=0.7,linestyle='none',marker='*',markersize=5,colo
plt.plot(features,lr.coef_,alpha=0.4,linestyle='none',marker='o',markersize=7,color='gre
plt.xticks(rotation = 90)
plt.legend()
plt.show()
```

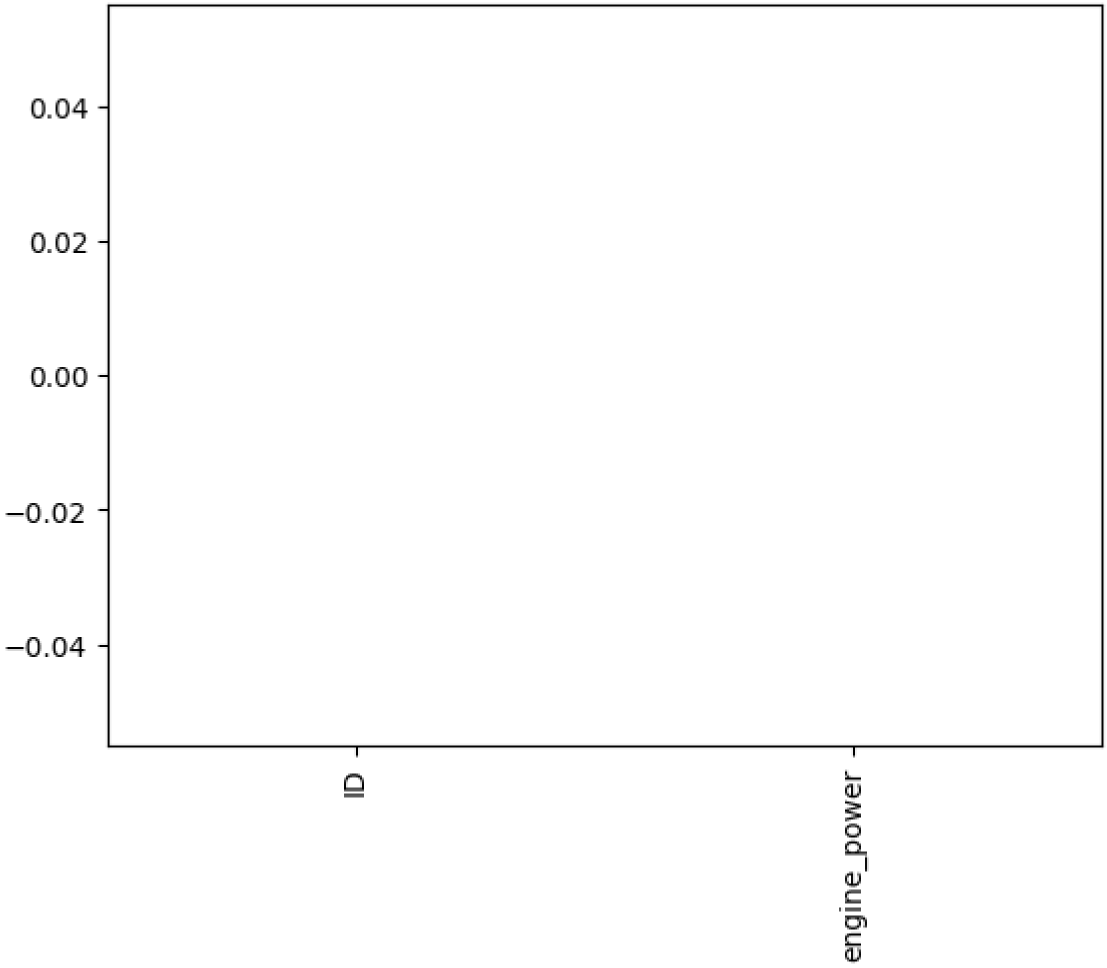


In [223]:

```
pd.Series(lasso.coef_, features).sort_values(ascending = True).plot(kind = "bar")
```

Out[223]:

```
<Axes: >
```



In [224]:

```
from sklearn.linear_model import LassoCV
#Lasso Cross validation
lasso_cv = LassoCV(alphas = [0.0001, 0.001,0.01, 0.1, 1, 10], random_state=0).fit(x_trai
#score
print(lasso_cv.score(x_train, y_train))
print(lasso_cv.score(x_test, y_test))
```

```
0.07448634159905387
0.07913288806451946
```

Elastic net

In [225]:

```
from sklearn.linear_model import ElasticNet
regr=ElasticNet()
regr.fit(x,y)
print(regr.coef_)
print(regr.intercept_)
```

```
[ 8.46751882e-02 -1.30405006e+02]
15279.442735227916
```

In [226]:

```
y_pred_elastic=regr.predict(x_train)
```

In [227]:

```
mse=np.mean((y_pred_elastic-y_train)**2)
print("Mean Squared Error on test set",mse)
```

```
Mean Squared Error on test set 48390222.80186546
```

In [ ]:

```

```