Lasso Ridge Regularization

In [1]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.datasets import fetch_openml
from sklearn.model_selection import cross_val_score
from sklearn import preprocessing
import pickle
```

In [3]:
```
data= pd.read_csv(r"C:\Users\JANHAVI\Downloads\car-mpg.csv")
```

In [4]:
```
data
```

Out[4]:

	mpg	cyl	disp	hp	wt	acc	yr	origin	car_type	car_name
0	18.0	8	307.0	130	3504	12.0	70	1	0	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	0	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	0	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	0	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	0	ford torino
...	...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	82	1	1	ford mustang gl
394	44.0	4	97.0	52	2130	24.6	82	2	1	vw pickup
395	32.0	4	135.0	84	2295	11.6	82	1	1	dodge rampage
396	28.0	4	120.0	79	2625	18.6	82	1	1	ford ranger
397	31.0	4	119.0	82	2720	19.4	82	1	1	chevy s-10

398 rows × 10 columns

In [5]:
```
data = data.drop(['car_name'], axis = 1)
data['origin'] = data['origin'].replace({1: 'america', 2: 'europe', 3: 'asia'})
data = pd.get_dummies(data, columns = ['origin'],dtype=int)
data = data.replace('?',np.nan)
```

In [6]:
```
import numpy as np
import pandas as pd
data = data.apply(pd.to_numeric, errors='ignore')
numeric_cols = data.select_dtypes(include=[np.number]).columns
data[numeric_cols] = data[numeric_cols].apply(lambda x: x.fillna(x.median()))
```

In [7]:
```
data.head()
```

Out[7]:

	mpg	cyl	disp	hp	wt	acc	yr	car_type	origin_america	origin_asia	origin_europe
0	18.0	8	307.0	130.0	3504	12.0	70	0	1	0	0
1	15.0	8	350.0	165.0	3693	11.5	70	0	1	0	0
2	18.0	8	318.0	150.0	3436	11.0	70	0	1	0	0
3	16.0	8	304.0	150.0	3433	12.0	70	0	1	0	0
4	17.0	8	302.0	140.0	3449	10.5	70	0	1	0	0

In [26]:
```
x = data.drop(['mpg'], axis =1)
y = data[['mpg']]
```

In [9]:
```
x_s = preprocessing.scale(x)
x_s = pd.DataFrame(x_s, columns = x.columns)

y_s =  preprocessing.scale(y)
y_s = pd.DataFrame(y_s, columns = y.columns)
```

In [10]:
```
x_s
```

Out[10]:

	cyl	disp	hp	wt	acc	yr	car_type	origin_america	or
0	1.498191	1.090604	0.673118	0.630870	-1.295498	-1.627426	-1.062235	0.773559	-
1	1.498191	1.503514	1.589958	0.854333	-1.477038	-1.627426	-1.062235	0.773559	-
2	1.498191	1.196232	1.197027	0.550470	-1.658577	-1.627426	-1.062235	0.773559	-
3	1.498191	1.061796	1.197027	0.546923	-1.295498	-1.627426	-1.062235	0.773559	-
4	1.498191	1.042591	0.935072	0.565841	-1.840117	-1.627426	-1.062235	0.773559	-
...	...	...	...	...	...	...	...	...	
393	-0.856321	-0.513026	-0.479482	-0.213324	0.011586	1.621983	0.941412	0.773559	-
394	-0.856321	-0.925936	-1.370127	-0.993671	3.279296	1.621983	0.941412	-1.292726	-
395	-0.856321	-0.561039	-0.531873	-0.798585	-1.440730	1.621983	0.941412	0.773559	-
396	-0.856321	-0.705077	-0.662850	-0.408411	1.100822	1.621983	0.941412	0.773559	-
397	-0.856321	-0.714680	-0.584264	-0.296088	1.391285	1.621983	0.941412	0.773559	-

398 rows × 10 columns

In [11]:
```
y_s
```

Out[11]:

	mpg
0	-0.706439
1	-1.090751
2	-0.706439
3	-0.962647
4	-0.834543
...	...
393	0.446497
394	2.624265
395	1.087017
396	0.574601
397	0.958913

398 rows × 1 columns

In [12]:

```
#Split into train, test set

X_train, X_test, y_train,y_test = train_test_split(x_s, y_s, test_size = 0.30, ranc
X_train.shape
```

Out[12]:

```
(278, 10)
```

Simple Linear Model

In [13]:

```
#Fit simple linear model and find coefficients
regression_model = LinearRegression()
regression_model.fit(X_train, y_train)

for idx, col_name in enumerate(X_train.columns):
    print('The coefficient for {} is {}'.format(col_name, regression_model.coef_[0]

intercept = regression_model.intercept_[0]
print('The intercept is {}'.format(intercept))
```

```
The coefficient for cyl is 0.32102238569161323
The coefficient for disp is 0.3248343091848378
The coefficient for hp is -0.22916950059437616
The coefficient for wt is -0.7112101905072294
The coefficient for acc is 0.01471368276419148
The coefficient for yr is 0.37558119495107445
The coefficient for car_type is 0.38147694842331026
The coefficient for origin_america is -0.07472247547584143
The coefficient for origin_asia is 0.044515252035678604
The coefficient for origin_europe is 0.048348549539454194
The intercept is 0.01928411610363977
```

Regularized Ridge Regression

In [14]:

```
#alpha factor here is lambda (penalty term) which helps to reduce the magnitude of
```

```
ridge_model = Ridge(alpha = 0.3)
ridge_model.fit(X_train, y_train)

print('Ridge model coef: {}'.format(ridge_model.coef_))
#As the data has 10 columns hence 10 coefficients appear here
```

```
Ridge model coef: [ 0.31649043  0.31320707 -0.22876025 -0.70109447  0.01295851  0.
37447352
  0.37725608 -0.07423624  0.04441039  0.04784031]
```

Regularization Lasso Regression

In [15]:
```
#alpha factor here is lambda (penalty term) which helps to reduce the magnitude of

lasso_model = Lasso(alpha = 0.1)
lasso_model.fit(X_train, y_train)

print('Lasso model coef: {}'.format(lasso_model.coef_))
#As the data has 10 columns hence 10 coefficients appear here
```

```
Lasso model coef: [-0.         -0.         -0.01690287 -0.51890013  0.          0.
28138241
  0.1278489  -0.01642647  0.          0.        ]
```

Score Comparison

In [16]:
```
#Model score - r^2 or coeff of determinant
#r^2 = 1-(RSS/TSS) = Regression error/TSS


#Simple Linear Model
print(regression_model.score(X_train, y_train))
print(regression_model.score(X_test, y_test))

print('*************************')
#Ridge
print(ridge_model.score(X_train, y_train))
print(ridge_model.score(X_test, y_test))

print('*************************')
#Lasso
print(lasso_model.score(X_train, y_train))
print(lasso_model.score(X_test, y_test))
```

```
0.8343770256960538
0.8513421387780065
*************************
0.8343617931312616
0.8518882171608506
*************************
0.7938010766228453
0.8375229615977084
```

Model Parameter Tunning

In [17]:
```
data_train_test = pd.concat([X_train, y_train], axis =1)
data_train_test.head()
```

Out[17]:

	cyl	disp	hp	wt	acc	yr	car_type	origin_america	or
350	-0.856321	-0.849116	-1.081977	-0.893172	-0.242570	1.351199	0.941412	0.773559	-
59	-0.856321	-0.925936	-1.317736	-0.847061	2.879909	-1.085858	0.941412	-1.292726	-
120	-0.856321	-0.695475	0.201600	-0.121101	-0.024722	-0.815074	0.941412	-1.292726	-
12	1.498191	1.983643	1.197027	0.934732	-2.203196	-1.627426	-1.062235	0.773559	-
349	-0.856321	-0.983552	-0.951000	-1.165111	0.156817	1.351199	0.941412	-1.292726	

In [18]:

```
pip install statsmodels
```

```
Requirement already satisfied: statsmodels in c:\users\janhavi\.conda\lib\site-pac
kages (0.14.5)
Requirement already satisfied: numpy<3,>=1.22.3 in c:\users\janhavi\.conda\lib\sit
e-packages (from statsmodels) (1.24.3)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in c:\users\janhavi\.conda\lib\s
ite-packages (from statsmodels) (1.10.1)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in c:\users\janhavi\.conda\lib
\site-packages (from statsmodels) (2.1.4)
Requirement already satisfied: patsy>=0.5.6 in c:\users\janhavi\.conda\lib\site-pa
ckages (from statsmodels) (1.0.1)
Requirement already satisfied: packaging>=21.3 in c:\users\janhavi\.conda\lib\site
-packages (from statsmodels) (23.0)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\janhavi\.conda\l
ib\site-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\janhavi\.conda\lib\site-pa
ckages (from pandas!=2.1.0,>=1.4->statsmodels) (2022.7)
Requirement already satisfied: tzdata>=2022.1 in c:\users\janhavi\.conda\lib\site-
packages (from pandas!=2.1.0,>=1.4->statsmodels) (2025.2)
Requirement already satisfied: six>=1.5 in c:\users\janhavi\.conda\lib\site-packag
es (from python-dateutil>=2.8.2->pandas!=2.1.0,>=1.4->statsmodels) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [27]:

```python
import statsmodels.formula.api as smf
ols1 = smf.ols(formula = 'mpg ~ cyl+disp+hp+wt+acc+yr+car_type+origin_america+origi
ols1.params
```

Out[27]:

```
Intercept          0.019284
cyl                0.321022
disp               0.324834
hp                -0.229170
wt                -0.711210
acc                0.014714
yr                 0.375581
car_type           0.381477
origin_america    -0.074722
origin_europe      0.048349
origin_asia        0.044515
dtype: float64
```

In [20]:

```python
print(ols1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    mpg   R-squared:                       0.834
Model:                            OLS   Adj. R-squared:                  0.829
Method:                 Least Squares   F-statistic:                     150.0
Date:                Fri, 22 Aug 2025   Prob (F-statistic):           3.12e-99
Time:                        10:36:04   Log-Likelihood:                -146.89
No. Observations:                 278   AIC:                             313.8
Df Residuals:                     268   BIC:                             350.1
Df Model:                           9
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
Intercept          0.0193      0.025      0.765      0.445      -0.030       0.069
cyl                0.3210      0.112      2.856      0.005       0.100       0.542
disp               0.3248      0.128      2.544      0.012       0.073       0.576
hp                -0.2292      0.079     -2.915      0.004      -0.384      -0.074
wt                -0.7112      0.088     -8.118      0.000      -0.884      -0.539
acc                0.0147      0.039      0.373      0.709      -0.063       0.092
yr                 0.3756      0.029     13.088      0.000       0.319       0.432
car_type           0.3815      0.067      5.728      0.000       0.250       0.513
origin_america    -0.0747      0.020     -3.723      0.000      -0.114      -0.035
origin_europe      0.0483      0.021      2.270      0.024       0.006       0.090
origin_asia        0.0445      0.020      2.175      0.031       0.004       0.085
==============================================================================
Omnibus:                       22.678   Durbin-Watson:                   2.105
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               36.139
Skew:                           0.513   Prob(JB):                     1.42e-08
Kurtosis:                       4.438   Cond. No.                     5.73e+15
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly s
pecified.
[2] The smallest eigenvalue is 4.76e-29. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [21]:

```python
#Lets check Sum of Squared Errors (SSE) by predicting value of y for test cases and
mse  = np.mean((regression_model.predict(X_test)-y_test)**2)

# root of mean_sq_error is standard deviation i.e. avg variance between predicted a
import math
rmse = math.sqrt(mse)
print('Root Mean Squared Error: {}'.format(rmse))
```

```
Root Mean Squared Error: 0.3776693425408785
```

In [22]:

```python
import statsmodels
print(statsmodels.__version__)
```

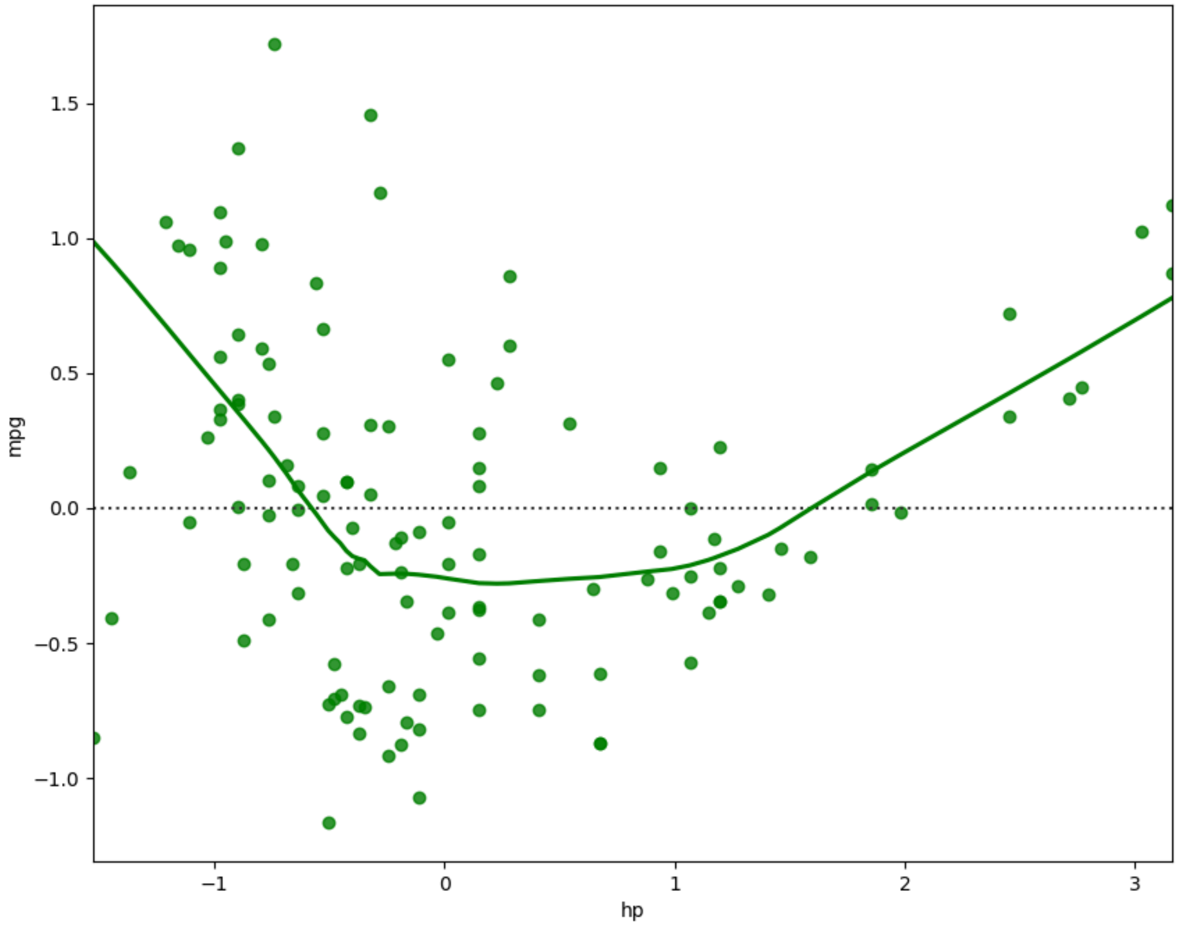
```
0.14.5
```

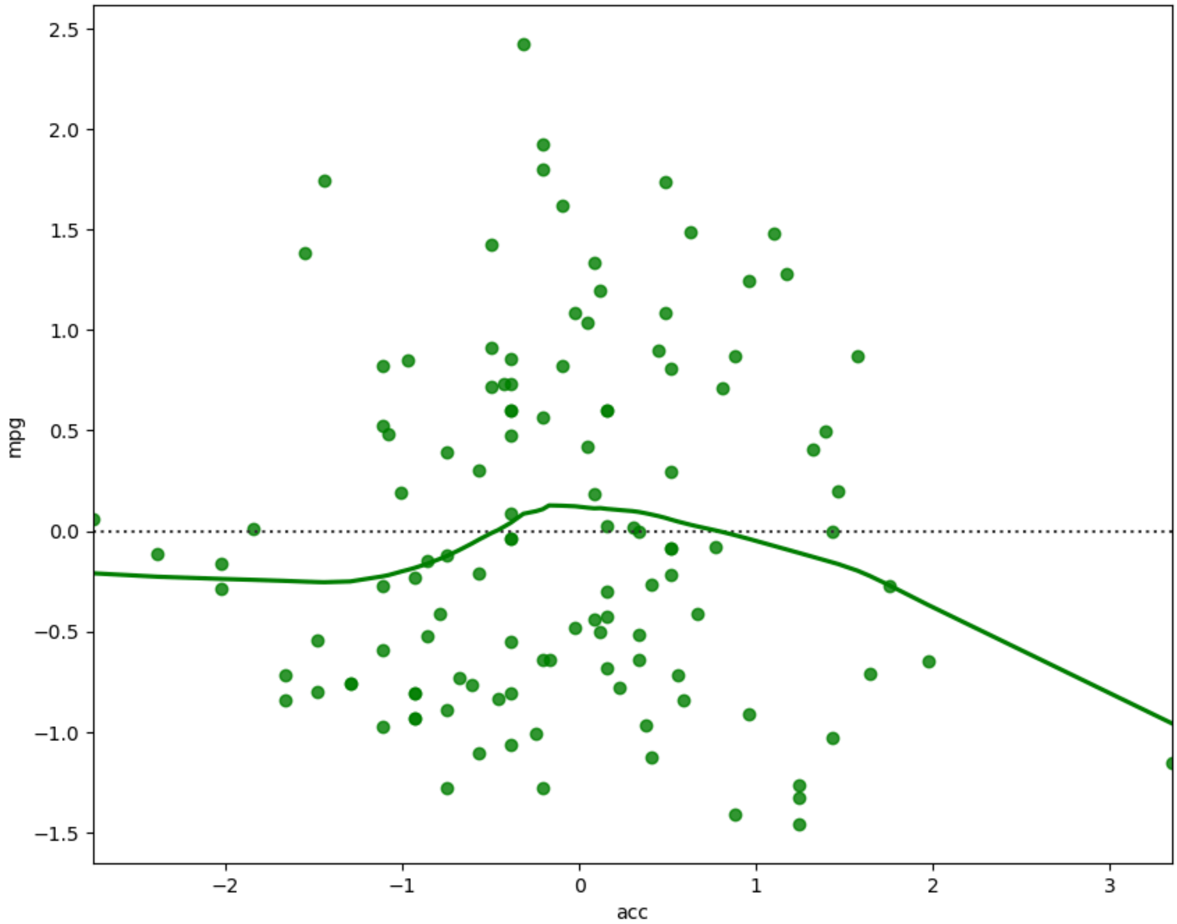
In [23]:

```python
# Is OLS a good model ? Lets check the residuals for some of these predictor.

fig = plt.figure(figsize=(10,8))
sns.residplot(x= X_test['hp'], y= y_test['mpg'], color='green', lowess=True )


fig = plt.figure(figsize=(10,8))
sns.residplot(x= X_test['acc'], y= y_test['mpg'], color='green', lowess=True )
```

Out[23]:

```
<Axes: xlabel='acc', ylabel='mpg'>
```





In [25]:

```
# predict mileage (mpg) for a set of attributes not in the training or test set
y_pred = regression_model.predict(X_test)

# Since this is regression, plot the predicted y value vs actual y values for the t
```

```
# A good model's prediction will be close to actual leading to high R and R2 values
#plt.rcParams['figure.dpi'] = 500
plt.scatter(y_test['mpg'], y_pred)
```

Out[25]: <matplotlib.collections.PathCollection at 0x23d1d4d6c10>