Lasso and Ridge Regression

--> Lasso regression—also known as L1 regularization—is a form of regularization for linear regression models. Regularization is a statistical method to reduce errors caused by overfitting on training data. The primary goal of LASSO regression is to find a balance between model simplicity and accuracy. It achieves this by adding a penalty term to the traditional linear regression model, which encourages sparse solutions where some coefficients are forced to be exactly zero. This feature makes LASSO particularly useful for feature selection, as it can automatically identify and discard irrelevant or redundant variables.

--> Ridge regression—also known as L2 regularization—is one of several types of regularization for linear regression models. Regularization is a statistical method to reduce errors caused by overfitting on training data. Ridge regression specifically corrects for multicollinearity in regression analysis. This is useful when developing machine learning models that have a large number of parameters, particularly if those parameters also have high weights.

Steps in performing regularization

1. Perform basic EDA
2. Scale data and apply Linear, Ridge & Lasso Regression with Regularization
3. Compare the r^2 score to determine which of the above regression methods gives the highest score
4. Compute Root mean squared error (RMSE) which inturn gives a better score than r^2
5. Finally use a scatter plot to graphically depict the correlation between actual and predicted mpg values

1. Perform Basic EDA

Importing the required libraries

In [1]:
```
# Numerical libraries
import numpy as np # linear algebra
import pandas as pd # Data Processing

# Graphical Libraries
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

# Linear Regression Machine Learning Libraries
from sklearn import preprocessing
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import r2_score
```

Reading the dataset

In [2]:
```
car_data = pd.read_csv(r"C:\Users\vijayram\OneDrive\Desktop\Python_NIT\NIT\Datasets_csv files\car-m
car_data
```

Out[2]:

	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

Data Preprocessing

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

Out[3]:

	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

In [4]:
```

car_data.columns
```

Out[4]:
```
Index(['mpg', 'cyl', 'disp', 'hp', 'wt', 'acc', 'yr', 'origin', 'car_type',
       'car_name'],
      dtype='object')
```

2. Scale data and apply Linear, Ridge & Lasso Regression with Regularization

In [5]:
```
#Drop car name
car_data = car_data.drop(['car_name'], axis = 1)
car_data
```

Out[5]:

	mpg	cyl	disp	hp	wt	acc	yr	origin	car_type
0	18.0	8	307.0	130	3504	12.0	70	1	0
1	15.0	8	350.0	165	3693	11.5	70	1	0
2	18.0	8	318.0	150	3436	11.0	70	1	0
3	16.0	8	304.0	150	3433	12.0	70	1	0
4	17.0	8	302.0	140	3449	10.5	70	1	0
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	82	1	1
394	44.0	4	97.0	52	2130	24.6	82	2	1
395	32.0	4	135.0	84	2295	11.6	82	1	1
396	28.0	4	120.0	79	2625	18.6	82	1	1
397	31.0	4	119.0	82	2720	19.4	82	1	1

398 rows × 9 columns

In [6]:
```
#Replace origin into 1,2,3..
car_data['origin'] = car_data['origin'].replace({1: 'Asia', 2: 'America', 3: 'Europe'})
car_data
```

Out[6]:

	mpg	cyl	disp	hp	wt	acc	yr	origin	car_type
0	18.0	8	307.0	130	3504	12.0	70	Asia	0
1	15.0	8	350.0	165	3693	11.5	70	Asia	0
2	18.0	8	318.0	150	3436	11.0	70	Asia	0
3	16.0	8	304.0	150	3433	12.0	70	Asia	0
4	17.0	8	302.0	140	3449	10.5	70	Asia	0
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	82	Asia	1
394	44.0	4	97.0	52	2130	24.6	82	America	1
395	32.0	4	135.0	84	2295	11.6	82	Asia	1
396	28.0	4	120.0	79	2625	18.6	82	Asia	1
397	31.0	4	119.0	82	2720	19.4	82	Asia	1

398 rows × 9 columns

In [7]:
```
# Get dummies for origin column
car_data = pd.get_dummies(car_data,columns = ['origin'])
car_data
```

Out[7]:

	mpg	cyl	disp	hp	wt	acc	yr	car_type	origin_America	origin_Asia	origin_Europe
0	18.0	8	307.0	130	3504	12.0	70	0	False	True	False
1	15.0	8	350.0	165	3693	11.5	70	0	False	True	False
2	18.0	8	318.0	150	3436	11.0	70	0	False	True	False
3	16.0	8	304.0	150	3433	12.0	70	0	False	True	False
4	17.0	8	302.0	140	3449	10.5	70	0	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	82	1	False	True	False
394	44.0	4	97.0	52	2130	24.6	82	1	True	False	False
395	32.0	4	135.0	84	2295	11.6	82	1	False	True	False
396	28.0	4	120.0	79	2625	18.6	82	1	False	True	False
397	31.0	4	119.0	82	2720	19.4	82	1	False	True	False

398 rows × 11 columns

In [8]:
```
#Replace ? with nan
car_data = car_data.replace('?', np.nan)
car_data
```

Out[8]:

	mpg	cyl	disp	hp	wt	acc	yr	car_type	origin_America	origin_Asia	origin_Europe
0	18.0	8	307.0	130	3504	12.0	70	0	False	True	False
1	15.0	8	350.0	165	3693	11.5	70	0	False	True	False
2	18.0	8	318.0	150	3436	11.0	70	0	False	True	False
3	16.0	8	304.0	150	3433	12.0	70	0	False	True	False
4	17.0	8	302.0	140	3449	10.5	70	0	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	82	1	False	True	False
394	44.0	4	97.0	52	2130	24.6	82	1	True	False	False
395	32.0	4	135.0	84	2295	11.6	82	1	False	True	False
396	28.0	4	120.0	79	2625	18.6	82	1	False	True	False
397	31.0	4	119.0	82	2720	19.4	82	1	False	True	False

398 rows × 11 columns

In [9]:
```
#Replace all nan with median
car_data = car_data.apply(lambda x: x.fillna(x.median()), axis = 0)
car_data
```

Out[9]:

	mpg	cyl	disp	hp	wt	acc	yr	car_type	origin_America	origin_Asia	origin_Europe
0	18.0	8	307.0	130	3504	12.0	70	0	False	True	False
1	15.0	8	350.0	165	3693	11.5	70	0	False	True	False
2	18.0	8	318.0	150	3436	11.0	70	0	False	True	False
3	16.0	8	304.0	150	3433	12.0	70	0	False	True	False
4	17.0	8	302.0	140	3449	10.5	70	0	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	82	1	False	True	False
394	44.0	4	97.0	52	2130	24.6	82	1	True	False	False
395	32.0	4	135.0	84	2295	11.6	82	1	False	True	False
396	28.0	4	120.0	79	2625	18.6	82	1	False	True	False
397	31.0	4	119.0	82	2720	19.4	82	1	False	True	False

398 rows × 11 columns

In [10]:
```
car_data.head()
```

Out[10]:

	mpg	cyl	disp	hp	wt	acc	yr	car_type	origin_America	origin_Asia	origin_Europe
0	18.0	8	307.0	130	3504	12.0	70	0	False	True	False
1	15.0	8	350.0	165	3693	11.5	70	0	False	True	False
2	18.0	8	318.0	150	3436	11.0	70	0	False	True	False
3	16.0	8	304.0	150	3433	12.0	70	0	False	True	False
4	17.0	8	302.0	140	3449	10.5	70	0	False	True	False

Model Building

In [15]:
```
# Seperating independent and dependent variables
```

In [13]:
```
X = car_data.drop(['mpg'], axis = 1) # independent variable
X
```

Out[13]:

	cyl	disp	hp	wt	acc	yr	car_type	origin_America	origin_Asia	origin_Europe
0	8	307.0	130	3504	12.0	70	0	False	True	False
1	8	350.0	165	3693	11.5	70	0	False	True	False
2	8	318.0	150	3436	11.0	70	0	False	True	False
3	8	304.0	150	3433	12.0	70	0	False	True	False
4	8	302.0	140	3449	10.5	70	0	False	True	False
...	...	...	...	...	...	...	...	...	...	...
393	4	140.0	86	2790	15.6	82	1	False	True	False
394	4	97.0	52	2130	24.6	82	1	True	False	False
395	4	135.0	84	2295	11.6	82	1	False	True	False
396	4	120.0	79	2625	18.6	82	1	False	True	False
397	4	119.0	82	2720	19.4	82	1	False	True	False

398 rows × 10 columns

In [14]:
```
y = car_data[['mpg']] #dependent variable
y
```

Out[14]:

	mpg
0	18.0
1	15.0
2	18.0
3	16.0
4	17.0
...	...
393	27.0
394	44.0
395	32.0
396	28.0
397	31.0

398 rows × 1 columns

In [ ]:
```
#Scaling the data
```

In [16]:
```
X_s = preprocessing.scale(X)
X_s = pd.DataFrame(X_s, columns = X.columns) #converting scaled data into dataframe
X_s
```

Out[16]:

	cyl	disp	hp	wt	acc	yr	car_type	origin_America	origin_Asia	origin_Europe
0	1.498191	1.090604	0.673118	0.630870	-1.295498	-1.627426	-1.062235	-0.461968	0.773559	-0.497643
1	1.498191	1.503514	1.589958	0.854333	-1.477038	-1.627426	-1.062235	-0.461968	0.773559	-0.497643
2	1.498191	1.196232	1.197027	0.550470	-1.658577	-1.627426	-1.062235	-0.461968	0.773559	-0.497643
3	1.498191	1.061796	1.197027	0.546923	-1.295498	-1.627426	-1.062235	-0.461968	0.773559	-0.497643
4	1.498191	1.042591	0.935072	0.565841	-1.840117	-1.627426	-1.062235	-0.461968	0.773559	-0.497643
...	...	...	...	...	...	...	...	...	...	...
393	-0.856321	-0.513026	-0.479482	-0.213324	0.011586	1.621983	0.941412	-0.461968	0.773559	-0.497643
394	-0.856321	-0.925936	-1.370127	-0.993671	3.279296	1.621983	0.941412	2.164651	-1.292726	-0.497643
395	-0.856321	-0.561039	-0.531873	-0.798585	-1.440730	1.621983	0.941412	-0.461968	0.773559	-0.497643
396	-0.856321	-0.705077	-0.662850	-0.408411	1.100822	1.621983	0.941412	-0.461968	0.773559	-0.497643
397	-0.856321	-0.714680	-0.584264	-0.296088	1.391285	1.621983	0.941412	-0.461968	0.773559	-0.497643

398 rows × 10 columns

In [17]:
```
y_s = preprocessing.scale(y)
y_s = pd.DataFrame(y_s, columns = y.columns) #converting scaled data into dataframe
y_s
```

Out[17]:

	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 [22]:
```
#Split the data into train, test

X_train, X_test, y_train,y_test = train_test_split(X_s, y_s, test_size = 0.30, random_state =0)
X_train.shape
```

Out[22]: (278, 10)

In [23]:
```
y_train.shape
```

Out[23]: (278, 1)

Simple Linear Model

In [24]:
```
#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][idx]))

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

```
The coefficient for cyl is 0.24744479758946716
The coefficient for disp is 0.28838215446098725
The coefficient for hp is -0.1899034268715289
The coefficient for wt is -0.6732229065111779
The coefficient for acc is 0.06754501540688176
The coefficient for yr is 0.3446364072117274
The coefficient for car_type is 0.3149149154003765
The coefficient for origin_America is 0.031283357351475125
The coefficient for origin_Asia is -0.07682943694882903
The coefficient for origin_Europe is 0.06336048896619992
The intercept is -0.019500467624017432
```

Regularized Ridge Regression

In [25]:
```
#alpha factor here is lambda (penalty term) which helps to reduce the magnitude of coeff

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: [[ 0.24424435  0.27853222 -0.18980689 -0.66458446  0.06588077  0.34396213
   0.31169746  0.03080065 -0.07642734  0.06333336]]
```

Regularized Lasso Regression

In [26]:
```
#alpha factor here is lambda (penalty term) which helps to reduce the magnitude of coeff

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.         -0.06203044 -0.48363379  0.          0.27163751
  0.09620861  0.         -0.03490256  0.        ]
```

3. Compare the r^2 score to determine which of the above regression methods gives the highest score

Score Comparision

In [29]:
```
#Model score - r^2 or coeff of determinant
#r^2 = 1-(RSS/TSS) = Regression error/TSS

print('Simple Linear Model')
print(regression_model.score(X_train, y_train))
print(regression_model.score(X_test, y_test))
print()
print('Ridge Regression Model')
print(ridge_model.score(X_train, y_train))
print(ridge_model.score(X_test, y_test))
print()
print('Lasso Regression Model')
print(lasso_model.score(X_train, y_train))
print(lasso_model.score(X_test, y_test))
```

```
Simple Linear Model
0.836163800114943
0.8439452810748137

Ridge Regression Model
0.8361520170844985
0.8437853815947186

Lasso Regression Model
0.7994535676270828
0.81026554865651
```

Model Parameter Tuning

In [30]:
```
data_train_test = pd.concat([X_train, y_train], axis =1)
data_train_test.head()
```

Out[30]:

	cyl	disp	hp	wt	acc	yr	car_type	origin_America	origin_Asia	origin_Europe
230	1.498191	1.503514	1.720935	1.412400	-1.513346	0.268063	-1.062235	-0.461968	0.773559	-0.497643
357	-0.856321	-0.714680	-0.112746	-0.420234	-0.278877	1.351199	0.941412	-0.461968	-1.292726	2.009471
140	1.498191	1.061796	1.197027	1.521175	-0.024722	-0.544290	-1.062235	-0.461968	0.773559	-0.497643
22	-0.856321	-0.858718	-0.243723	-0.703997	0.701436	-1.627426	0.941412	2.164651	-1.292726	-0.497643
250	1.498191	1.196232	0.935072	0.903991	-0.859804	0.538847	-1.062235	-0.461968	0.773559	-0.497643

In [32]:
```
't statsmodels.formula.api as smf
= smf.ols(formula = 'mpg ~ cyl+disp+hp+wt+acc+yr+car_type+origin_America+origin_Europe+origin_Asia'
params
```

Out[32]:
```
Intercept        -0.019500
cyl               0.247445
disp              0.288382
hp               -0.189903
wt               -0.673223
acc               0.067545
yr                0.344636
car_type          0.314915
origin_America    0.031283
origin_Europe     0.063360
origin_Asia      -0.076829
dtype: float64
```

In [33]:
```
print(ols1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    mpg   R-squared:                       0.836
Model:                            OLS   Adj. R-squared:                  0.831
Method:                 Least Squares   F-statistic:                     152.0
Date:                Tue, 18 Jun 2024   Prob (F-statistic):          7.34e-100
Time:                        16:37:10   Log-Likelihood:                -139.91
No. Observations:                 278   AIC:                             299.8
Df Residuals:                     268   BIC:                             336.1
Df Model:                           9
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
Intercept         -0.0195      0.025     -0.796      0.427      -0.068       0.029
cyl                0.2474      0.102      2.419      0.016       0.046       0.449
disp               0.2884      0.118      2.435      0.016       0.055       0.522
hp                -0.1899      0.077     -2.451      0.015      -0.342      -0.037
wt                -0.6732      0.080     -8.427      0.000      -0.831      -0.516
acc                0.0675      0.040      1.693      0.092      -0.011       0.146
yr                 0.3446      0.028     12.324      0.000       0.290       0.400
car_type           0.3149      0.064      4.889      0.000       0.188       0.442
origin_America     0.0313      0.020      1.550      0.122      -0.008       0.071
origin_Europe      0.0634      0.020      3.148      0.002       0.024       0.103
origin_Asia       -0.0768      0.020     -3.858      0.000      -0.116      -0.038
==============================================================================
Omnibus:                       18.593   Durbin-Watson:                   1.806
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               26.354
Skew:                           0.471   Prob(JB):                     1.89e-06
Kurtosis:                       4.178   Cond. No.                     5.68e+15
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 4.88e-29. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

4. Compute Root mean squared error (RMSE) which inturn gives a better score than r^2

In [34]:
```
#Lets check Sum of Squared Errors (SSE) by predicting value of y for test cases and subtracting fro
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 and actual
import math
rmse = math.sqrt(mse)
print('Root Mean Squared Error: {}'.format(rmse))
```

```
Root Mean Squared Error: 0.4045366587848482
```

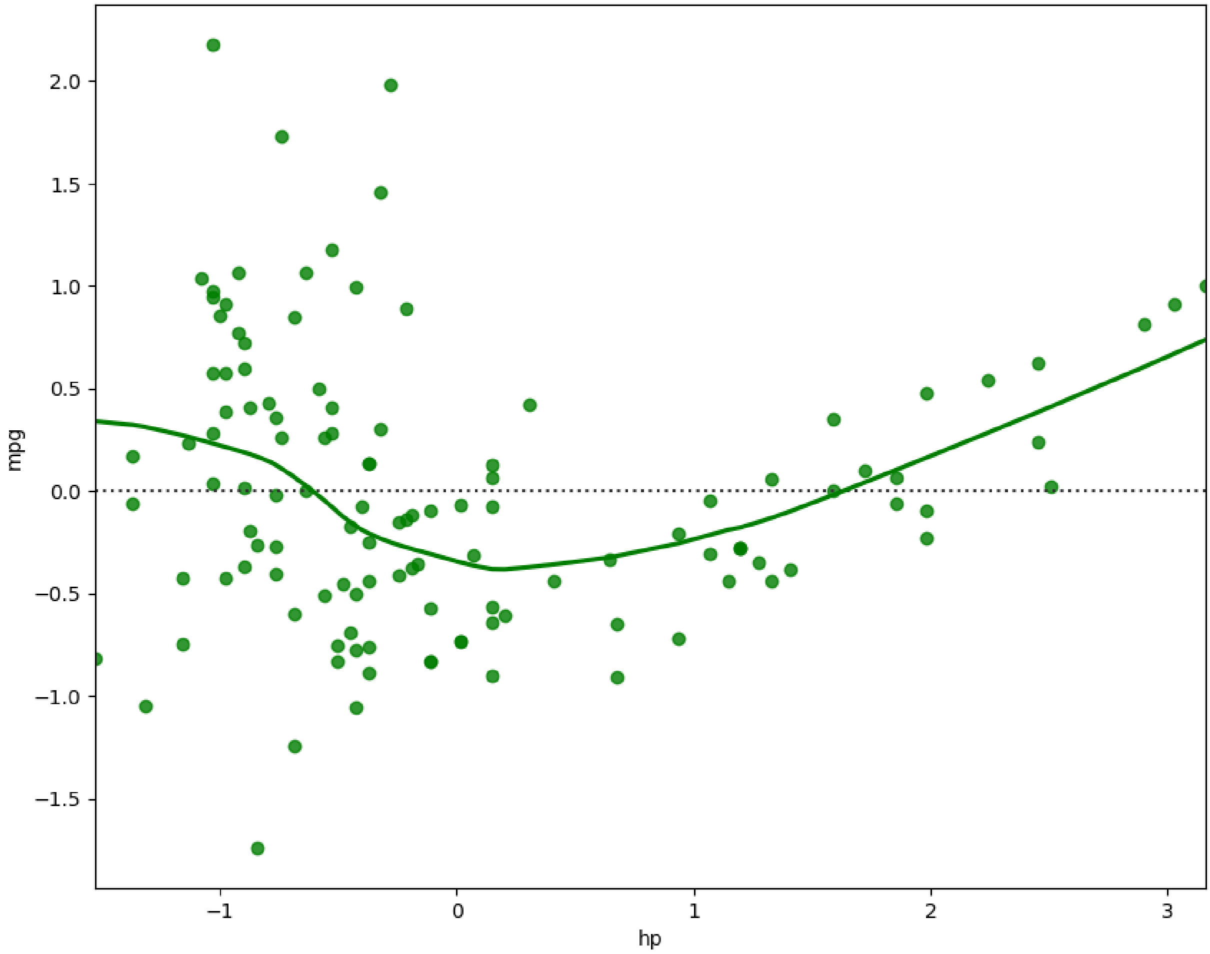
So there is an avg. mpg difference of 0.37 from real mpg

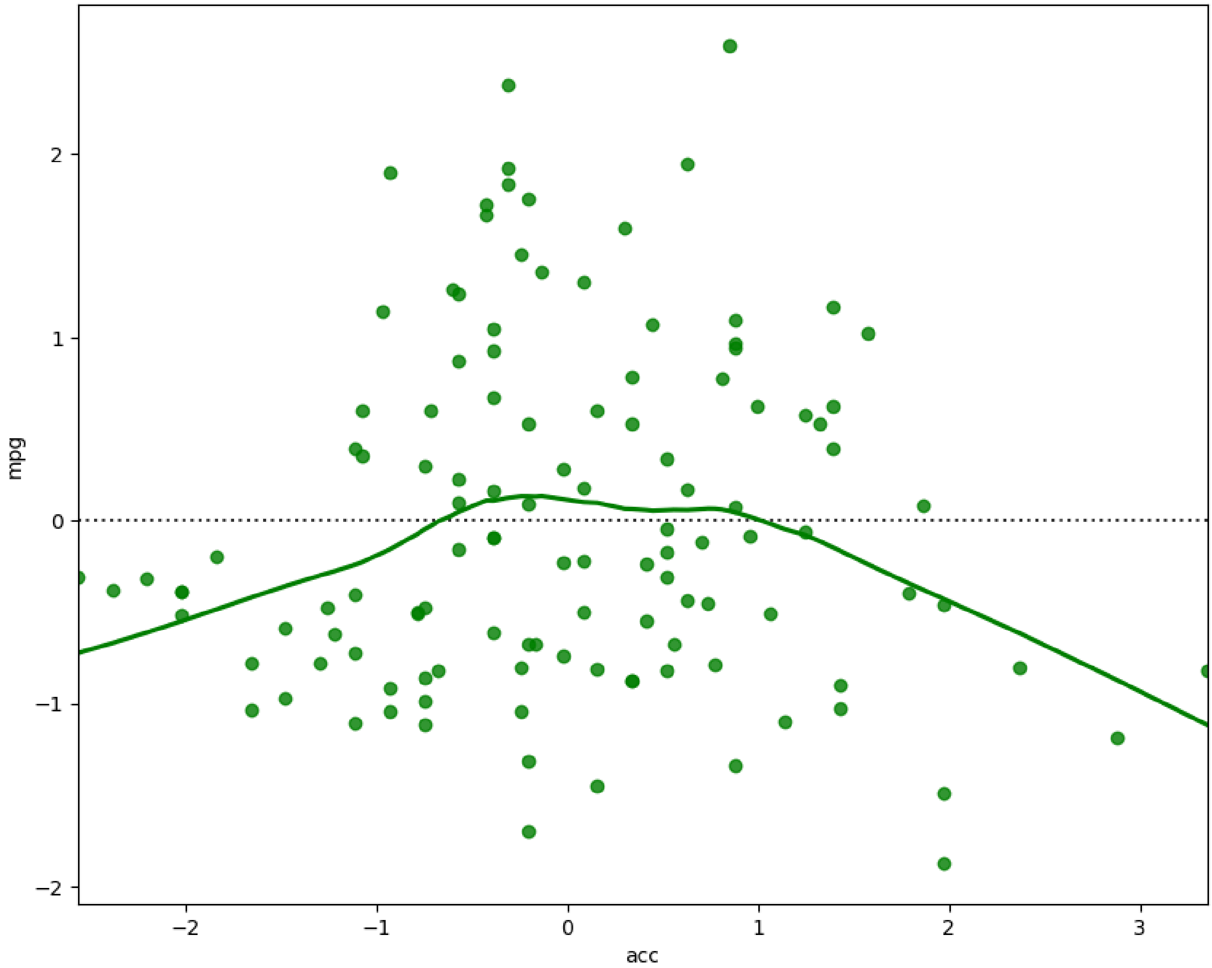
In [35]:
```
# Is OLS a good model ? Lets check the residuals for some of these predictor.

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[35]: <Axes: xlabel='acc', ylabel='mpg'>



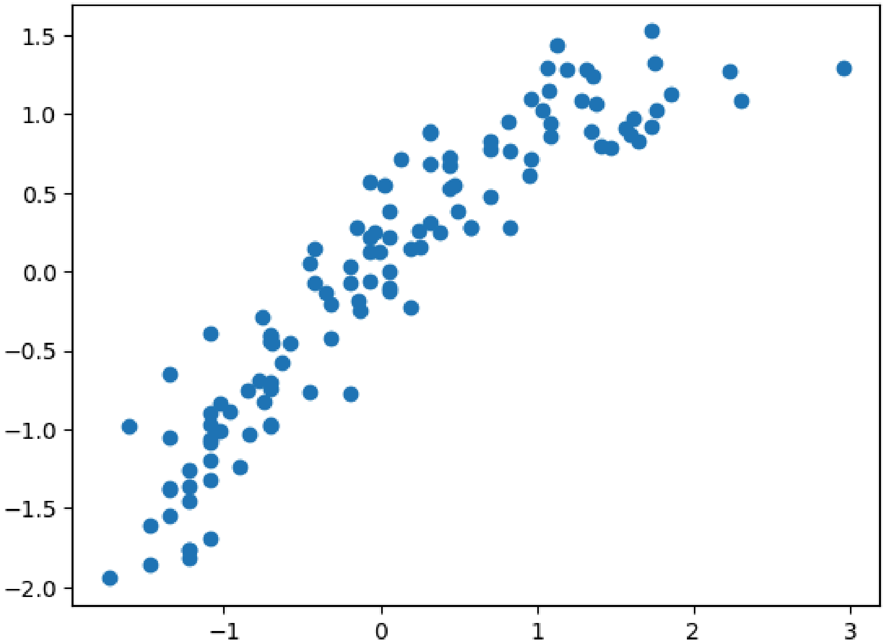


5. Finally use a scatter plot to graphically depict the correlation between actual and predicted mpg values

In [36]:
```
# 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 test data
# 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[36]: <matplotlib.collections.PathCollection at 0x2551a55b310>



Both Ridge & Lasso regularization performs very well on this data. Though Ridge gives a better score, the above scatter plot depicts the correlation between the actual and predicted mpg values

In [ ]: