In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline

from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import r2_score

data= pd.read_csv(r"C:\Users\santo\OneDrive\Desktop\Data science\Dec2025\Machine Learning\car-mpg.csv")
```

In [3]:
```
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]:
```
data.head(20)
```

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
5	15.0	8	429.0	198	4341	10.0	70	1	0	ford galaxie 500
6	14.0	8	454.0	220	4354	9.0	70	1	0	chevrolet impala
7	14.0	8	440.0	215	4312	8.5	70	1	0	plymouth fury iii
8	14.0	8	455.0	225	4425	10.0	70	1	0	pontiac catalina
9	15.0	8	390.0	190	3850	8.5	70	1	0	amc ambassador dpl
10	15.0	8	383.0	170	3563	10.0	70	1	0	dodge challenger se
11	14.0	8	340.0	160	3609	8.0	70	1	0	plymouth 'cuda 340
12	15.0	8	400.0	150	3761	9.5	70	1	0	chevrolet monte carlo
13	14.0	8	455.0	225	3086	10.0	70	1	0	buick estate wagon (sw)
14	24.0	4	113.0	95	2372	15.0	70	3	1	toyota corona mark ii
15	22.0	6	198.0	95	2833	15.5	70	1	0	plymouth duster
16	18.0	6	199.0	97	2774	15.5	70	1	0	amc hornet
17	21.0	6	200.0	85	2587	16.0	70	1	0	ford maverick
18	27.0	4	97.0	88	2130	14.5	70	3	1	datsun pl510
19	26.0	4	97.0	46	1835	20.5	70	2	1	volkswagen 1131 deluxe sedan

In [12]:
```
import numpy as np
import pandas as pd
```

```
# clean column names
data.columns = data.columns.str.strip().str.lower()

# drop car_name safely
data.drop(columns=['car_name'], errors='ignore', inplace=True)

# encode origin only if present
if 'origin' in data.columns:
    data['origin'] = data['origin'].map({
        1: 'america',
        2: 'europe',
        3: 'asia'
    })
    data = pd.get_dummies(data, columns=['origin'], dtype=int)

# replace missing symbols
data.replace('?', np.nan, inplace=True)
```

In [14]:
```
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())
)
```

```
C:\Users\santo\AppData\Local\Temp\ipykernel_4516\3584547213.py:5: FutureWarning: errors='ignore' is deprecated and will raise i
n a future version. Use to_numeric without passing `errors` and catch exceptions explicitly instead
  data = data.apply(pd.to_numeric, errors='ignore')
```

In [15]:
```
data.head()
```

Out[15]:

	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 [16]:
```
data
```

Out[16]:

	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
...	...	...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86.0	2790	15.6	82	1	1	0	0
394	44.0	4	97.0	52.0	2130	24.6	82	1	0	0	1
395	32.0	4	135.0	84.0	2295	11.6	82	1	1	0	0
396	28.0	4	120.0	79.0	2625	18.6	82	1	1	0	0
397	31.0	4	119.0	82.0	2720	19.4	82	1	1	0	0

398 rows × 11 columns

2. Model building

Here we would like to scale the data as the columns are varied which would result in 1 column dominating the others.

First we divide the data into independent (X) and dependent data (y) then we scale it.

Tip!:

*The reason we don't scale the entire data before and then divide it into train(X) & test(y) is because once you scale the data, the type(data_s) would be numpy.ndarray. It's impossible to divide this data when it's an array. *

Hence we divide type(data) pandas.DataFrame, then proceed to scaling it.

In [17]:
```
x = data.drop(['mpg'],axis =1)
y = data[['mpg']]
```

In [21]:
```
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 [24]:
```
x_s
```

Out[24]:

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

398 rows × 10 columns

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

Out[23]: (398, 11)

In [26]:
```
x_train, x_test, y_train,y_test = train_test_split(x_s, y_s, test_size = 0.30, random_state = 1)
x_train.shape
```

Out[26]: (278, 10)

2. Simple Linear Model

In [27]:
```
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 0.321022385691611
The coefficient for disp is 0.32483430918483897
The coefficient for hp is -0.22916950059437569
The coefficient for wt is -0.7112101905072298
The coefficient for acc is 0.014713682764191237
The coefficient for yr is 0.3755811949510748
The coefficient for car_type is 0.3814769484233099
The coefficient for origin_america is -0.07472247547584178
The coefficient for origin_asia is 0.044515252035677896
The coefficient for origin_europe is 0.04834854953945386
The intercept is 0.019284116103639764
```

2 . b Regularized Ridge Regression

In [28]:
```
ridge_model = Ridge(alpha = 0.3)
ridge_model.fit(x_train, y_train)

print('Ridge model coef:{}'.format(ridge_model.coef_))
```

```
Ridge model coef:[ 0.31649043  0.31320707 -0.22876025 -0.70109447  0.01295851  0.37447352
  0.37725608 -0.07423624  0.04441039  0.04784031]
```

2. c Regularized lasso Regression

In [29]:
```
lasso_model = Lasso(alpha = 0.1)
lasso_model.fit(x_train, y_train)
print('Lassomodel coef:{}'.format(lasso_model.coef_))
```

```
Lassomodel coef:[-0.         -0.         -0.01690287 -0.51890013  0.          0.28138241
  0.1278489  -0.01642647  0.          0.        ]
```

Here we noticed many coefficient are turned to 0 indicating drop of those dimensions from the model

3 . Score Comparison

In [32]:
```
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
0.8513421387780066
**************************
0.8343617931312616
0.8518882171608507
**************************
0.7938010766228453
0.8375229615977083
```

4. Model Parameter Tuning

- r^2 is not a reliable metric as it always increases with addition of more attributes even if the attributes have no influence on the predicted variable. Instead we use adjusted r^2 which removes the statistical chance that improves r^2

(adjusted r^2 = r^2 - fluke)

- Scikit does not provide a facility for adjusted r^2... so we use statsmodel, a library that gives results similar to what you obtain in R language
- This library expects the X and Y to be given in one single dataframe

In [33]:
```
data_train_test = pd.concat([x_train, y_train],axis =1)
data_train_test.head()
```

Out[33]:

	cyl	disp	hp	wt	acc	yr	car_type	origin_america	origin_asia	origin_europe	mpg
350	-0.856321	-0.849116	-1.081977	-0.893172	-0.242570	1.351199	0.941412	0.773559	-0.497643	-0.461968	1.432898
59	-0.856321	-0.925936	-1.317736	-0.847061	2.879909	-1.085858	0.941412	-1.292726	-0.497643	2.164651	-0.065919
120	-0.856321	-0.695475	0.201600	-0.121101	-0.024722	-0.815074	0.941412	-1.292726	-0.497643	2.164651	-0.578335
12	1.498191	1.983643	1.197027	0.934732	-2.203196	-1.627426	-1.062235	0.773559	-0.497643	-0.461968	-1.090751
349	-0.856321	-0.983552	-0.951000	-1.165111	0.156817	1.351199	0.941412	-1.292726	2.009471	-0.461968	1.356035

In [34]:
```
import statsmodels.formula.api as smf
ols1 = smf.ols(formula = 'mpg ~cyl+disp+hp+wt+acc+yr+car_type+origin_america+origin_asia', data= data_train_test).fit()
ols1.params
```

Out[34]:
```
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   -0.136182
origin_asia      -0.006138
dtype: float64
```

In [35]:
```
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:                Thu, 18 Dec 2025   Prob (F-statistic):           3.12e-99
Time:                        11:15:56   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    -0.1362      0.042     -3.227      0.001      -0.219      -0.053
origin_asia       -0.0061      0.034     -0.179      0.858      -0.074       0.061
==============================================================================
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.                         14.9
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [37]:
```
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))
```

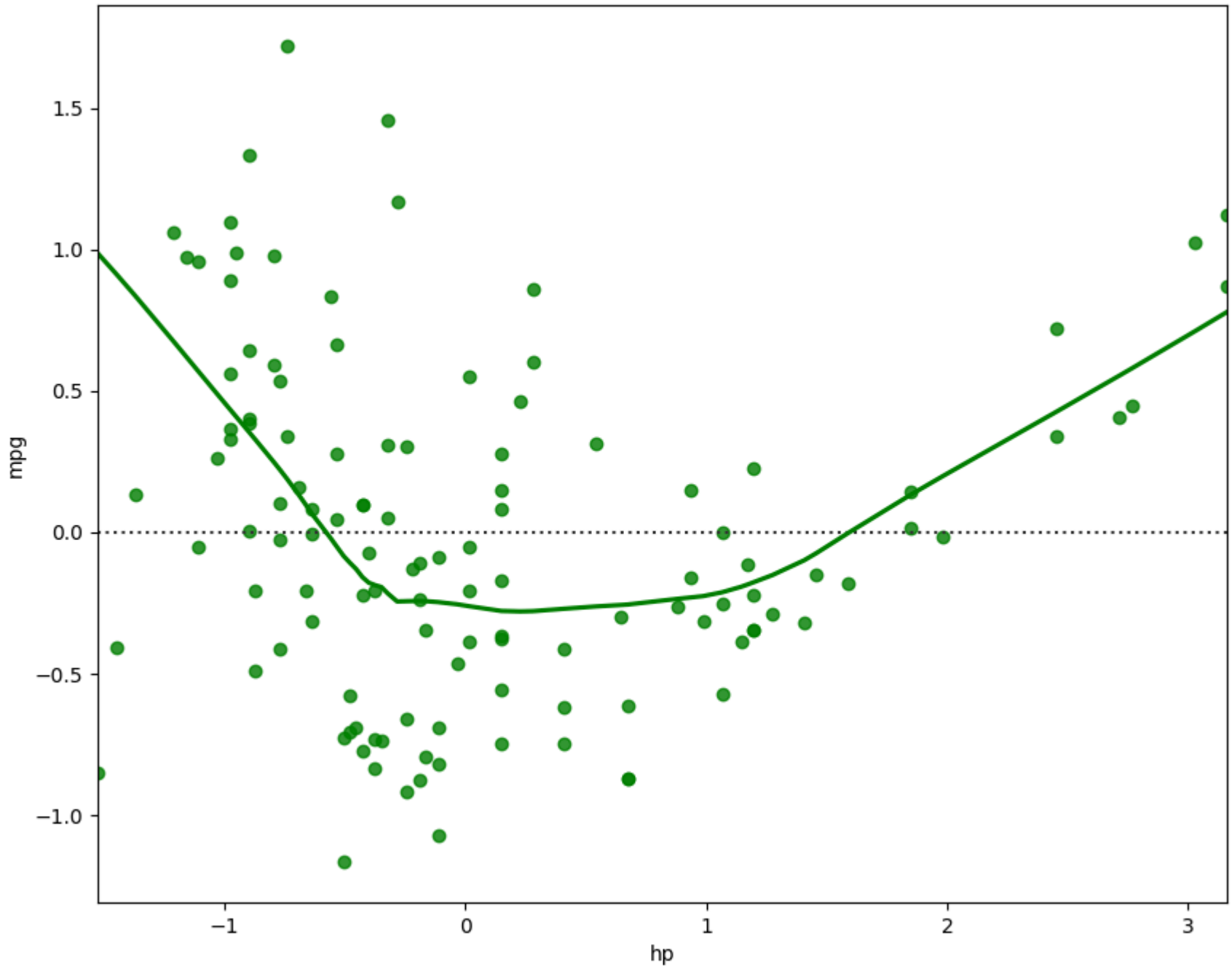
```
Root Mean Squared Error: 0.37766934254087847
```

In [48]:
```
# 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 )
```

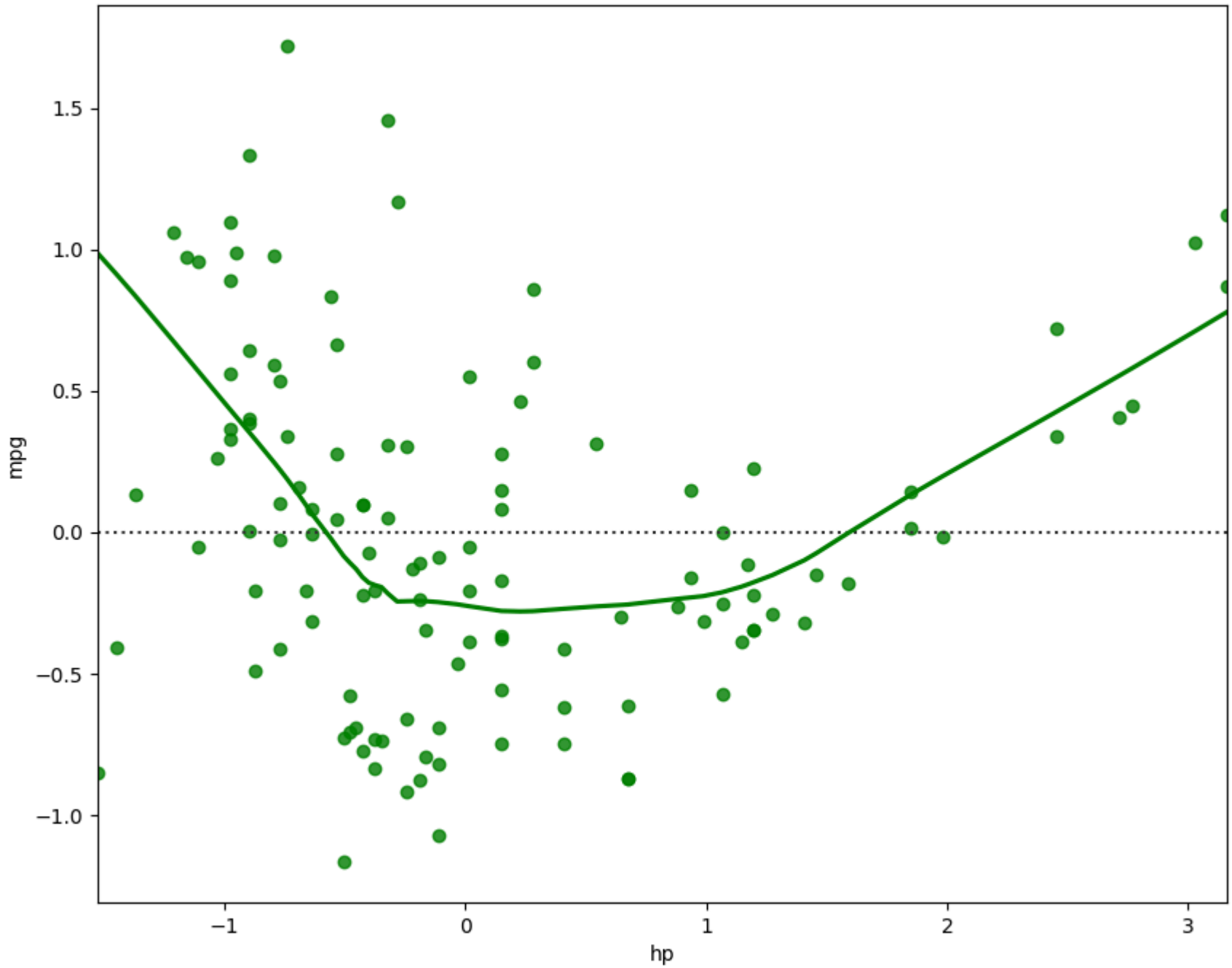
Out[48]:
```
<Axes: xlabel='acc', ylabel='mpg'>
```

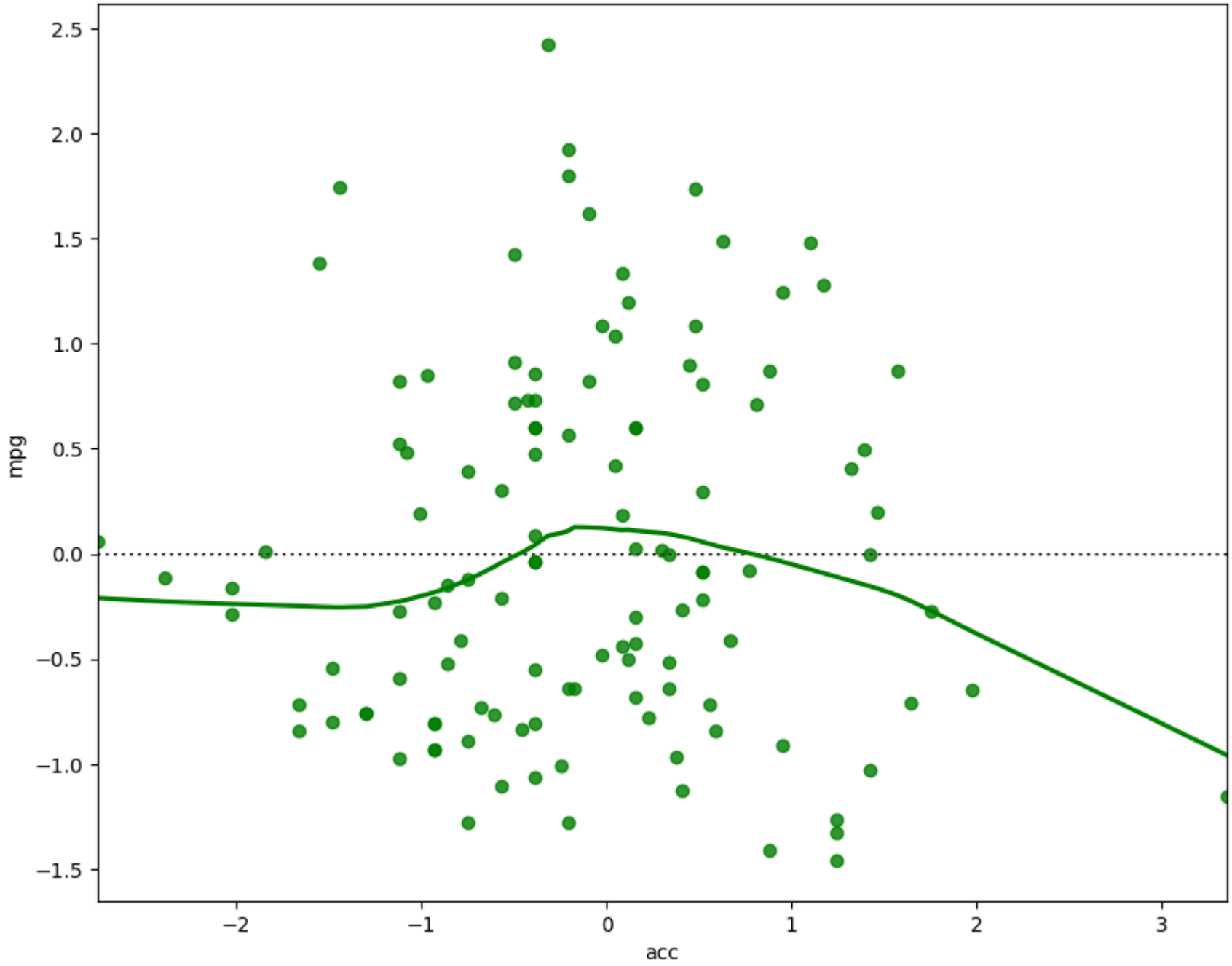
In [49]:
```
plt.show()
```

```
<Figure size 1000x800 with 0 Axes>
```


1.5
1.0
0.5
0.0
−0.5
−1.0
mpg
−1
0
1
2
3
hp

```
<Figure size 1000x800 with 0 Axes>
```

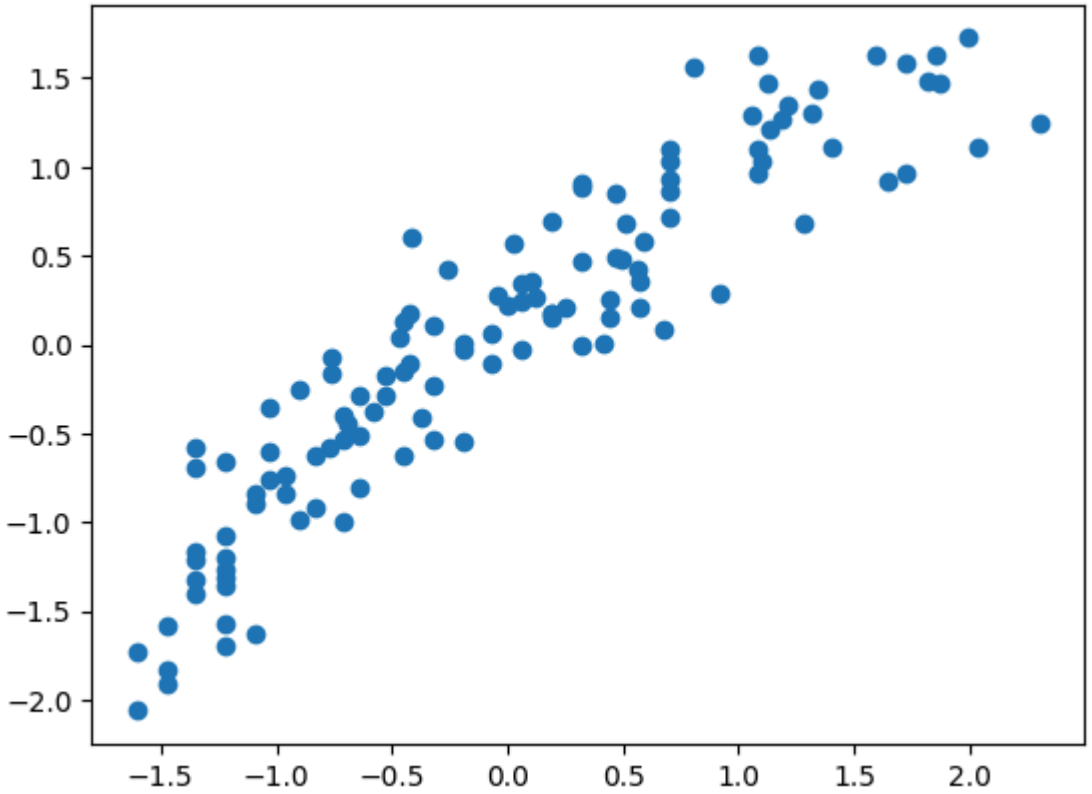

1.5
1.0
0.5
0.0
−0.5
−1.0
mpg
−1
0
1
2
3
hp


2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
mpg
−2
−1
0
1
2
3
acc

In [43]:
```
y_pred = regression_model.predict(x_test)
plt.scatter(y_test['mpg'], y_pred)
```

Out[43]:
```
<matplotlib.collections.PathCollection at 0x1fd99e72210>
```

In [44]:
```
plt.show()
```



5. Inference

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.

This kernel is a work in progress.