In [1]:
```
#import package and observe dataset

#import numerical libraries
import pandas as pd
import numpy as np

#import graphical plotting libraries
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

#import 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
```

In [2]:
```
data = pd.read_csv(r"C:\Users\soham\OneDrive\Desktop\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 |

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']) data = data.replace('?' ,np.nan) data = data.apply(lambda x: x.fillna(x.median()),axis = 0)

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']) data = data.replace('?', np.nan) data = data.apply(lambda x: x.fillna(x.median()), axis = 0)

In [5]:
```
#Drop car name
#Replace origin into 1,2,3.. dont forget get_dummies
#Replace ? with nan
#Replace all nan with median


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'])
data = data.replace('?', np.nan)
data["hp"] = pd.to_numeric(data["hp"],downcast="float")
data = data.apply(lambda x: x.fillna(x.median()), axis = 0)
```

In [6]:
```
data.head()
```

Out[6]:

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

2.model building

In [8]:
```
x = data.drop(['mpg'], axis = 1) #independent variable
y = data[['mpg']]  #dependent variable
```

In [9]:
```
#scaling the data

x_s = preprocessing.scale(x)
x_s = pd.DataFrame(x_s, columns = x.columns) #converting scaled data into data f

y_s = preprocessing.scale(y)
y_s = pd.DataFrame(y_s, columns = y.columns) #ideally train test data should be 
```

In [10]:
```
#split into train test set

x_train, x_test, y_train, y_test = train_test_split(x_s, y_s, test_size = 0.3, r
x_train.shape
```

Out[10]: (278, 10)

2.a) Simple Linear Model

In [12]:
```
#Fit simple linear model and find coeffients
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_[

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 [14]:
```
#alpha factor here is lambda (penalty term) which helps to reduce the magnitude

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]]
```

2.c) Regularized lasso Regression

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

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.        ]
```

3. Score Comparison

In [66]:
```
#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 Linear Model
print(ridge_model.score(x_train, y_train))
print(ridge_model.score(x_test, y_test))
print('***********************')

#Lasso Linear Model
print(lasso_model.score(x_train, y_train))
print(lasso_model.score(x_test, y_test))
print('***********************')
```

```
0.8343770256960538
0.8513421387780066
***********************
0.8343617931312616
0.8518882171608507
***********************
0.7938010766228453
0.8375229615977083
***********************
```

Polynomial Feature

In [ ]:
```
#poly = PolynomialFeatures(degree = 2, interaction_only = True)

#Fit calculates u and std dev while transform applies the transformation to a pa
#Here fit_transform helps to fit and transform the X_s
#Hence type(X_poly) is numpy.array while type(X_s) is pandas.DataFrame
#X_poly = poly.fit_transform(X_s)
#Similarly capture the coefficients and intercepts of this polynomial feature mo
```

4.Model Parameter Tuning

In [68]:
```
data_train_test = pd.concat([x_train, y_train], axis=1)
data_train_test.head()
```

Out[68]:

| | cyl | disp | hp | wt | acc | yr | car_type | origin_a |
|---|---|---|---|---|---|---|---|---|
| **350** | -0.856321 | -0.849116 | -1.081977 | -0.893172 | -0.242570 | 1.351199 | 0.941412 | 0 |
| **59** | -0.856321 | -0.925936 | -1.317736 | -0.847061 | 2.879909 | -1.085858 | 0.941412 | -1 |
| **120** | -0.856321 | -0.695475 | 0.201600 | -0.121101 | -0.024722 | -0.815074 | 0.941412 | -1 |
| **12** | 1.498191 | 1.983643 | 1.197027 | 0.934732 | -2.203196 | -1.627426 | -1.062235 | 0 |
| **349** | -0.856321 | -0.983552 | -0.951000 | -1.165111 | 0.156817 | 1.351199 | 0.941412 | -1 |

In [76]:
```
import statsmodels.formula.api as smf
ols1 = smf.ols(formula = 'mpg ~ cyl+disp+hp+wt+acc+yr+car_type+origin_america+or
ols1.params
```

Out[76]:
```
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 [78]:
```
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:                Sat, 28 Sep 2024   Prob (F-statistic):           3.12e-99
Time:                        16:50:47   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      0.97
5]
----------------------------------------------------------------------------------
-
Intercept          0.0193      0.025      0.765      0.445      -0.030       0.06
9
cyl                0.3210      0.112      2.856      0.005       0.100       0.54
2
disp               0.3248      0.128      2.544      0.012       0.073       0.57
6
hp                -0.2292      0.079     -2.915      0.004      -0.384      -0.07
4
wt                -0.7112      0.088     -8.118      0.000      -0.884      -0.53
9
acc                0.0147      0.039      0.373      0.709      -0.063       0.09
2
yr                 0.3756      0.029     13.088      0.000       0.319       0.43
2
car_type           0.3815      0.067      5.728      0.000       0.250       0.51
3
origin_america    -0.0747      0.020     -3.723      0.000      -0.114      -0.03
5
origin_europe      0.0483      0.021      2.270      0.024       0.006       0.09
0
origin_asia        0.0445      0.020      2.175      0.031       0.004       0.08
5
==============================================================================
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.                     1.59e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The smallest eigenvalue is 6.14e-30. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [80]:

```
#Lets check Sum of Squared Errors (SSE) by predicting value of y for test cases

mse = np.mean((regression_model.predict(x_test)-y_test)**2)

# root of mean_sq_error is standard deviation i.e. avg variance between predicte

import math
```

```
rmse = math.sqrt(mse)
print('Root Mean Squared Error: {}'.format(rmse))
```

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

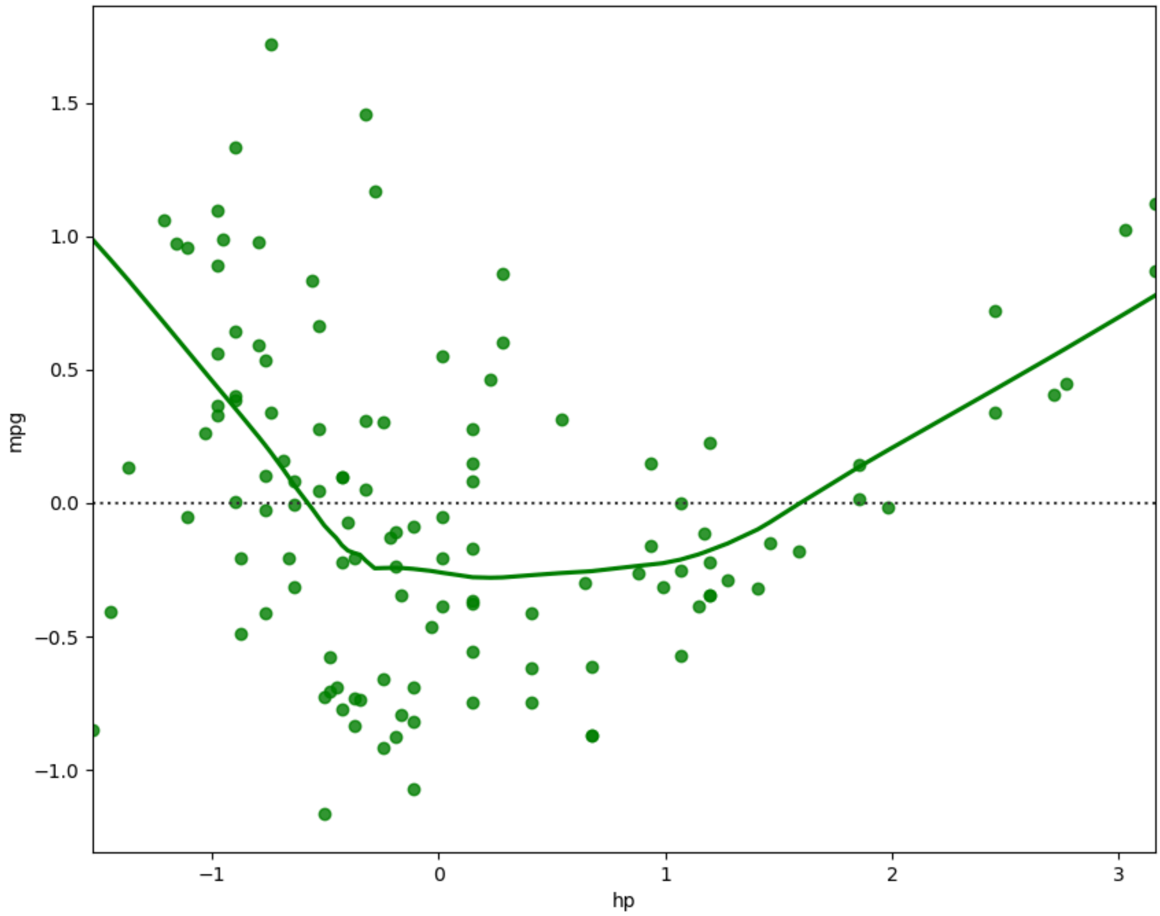
**So there is an avg. mpg difference of 0.37 from real mpg**

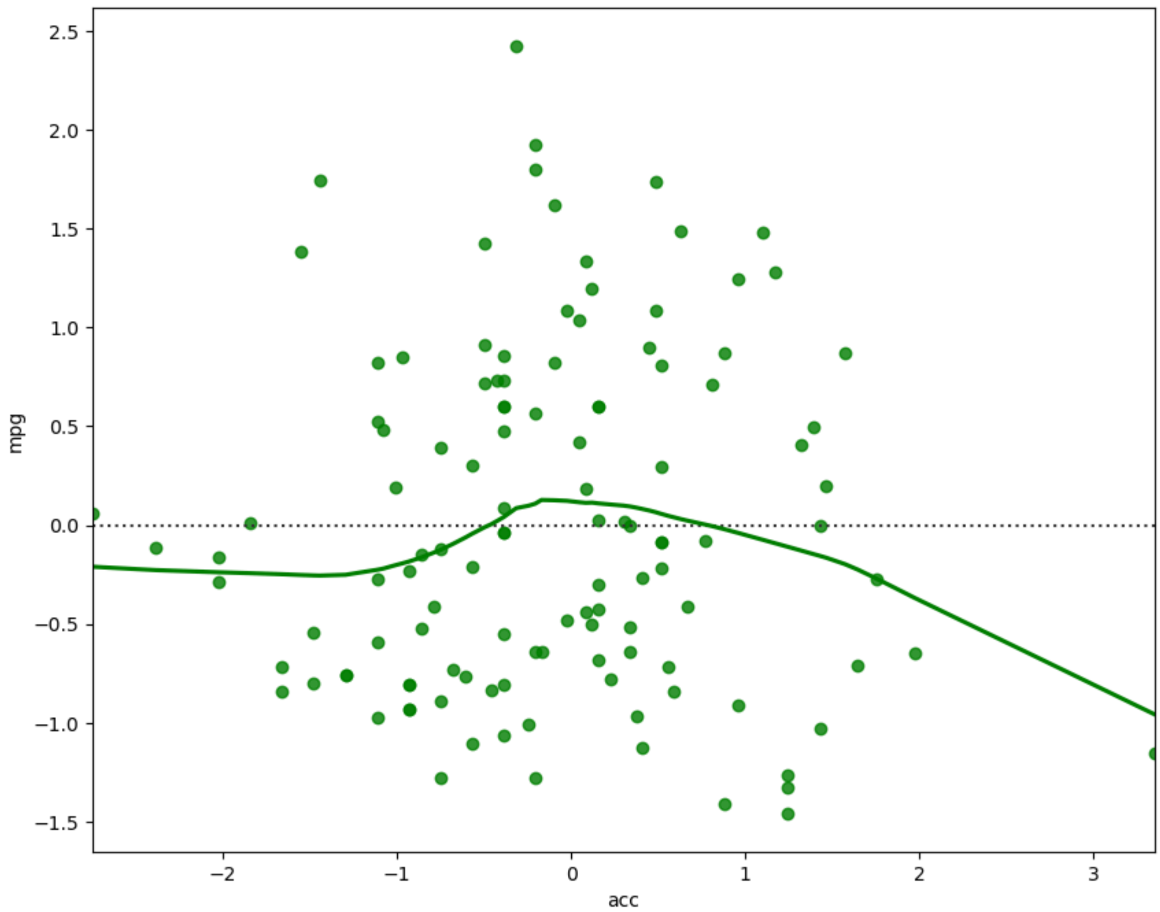
In [88]:
```
# Is OLS a good model ? Lets check the residuals for some of these predictor.

fig = plt.figure(figsize=(10,8))
#sns.residplot(x= x_test['hp'], y= y_test['mpg'],color='green',lowess=True)
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[88]:
```
<Axes: xlabel='acc', ylabel='mpg'>
```

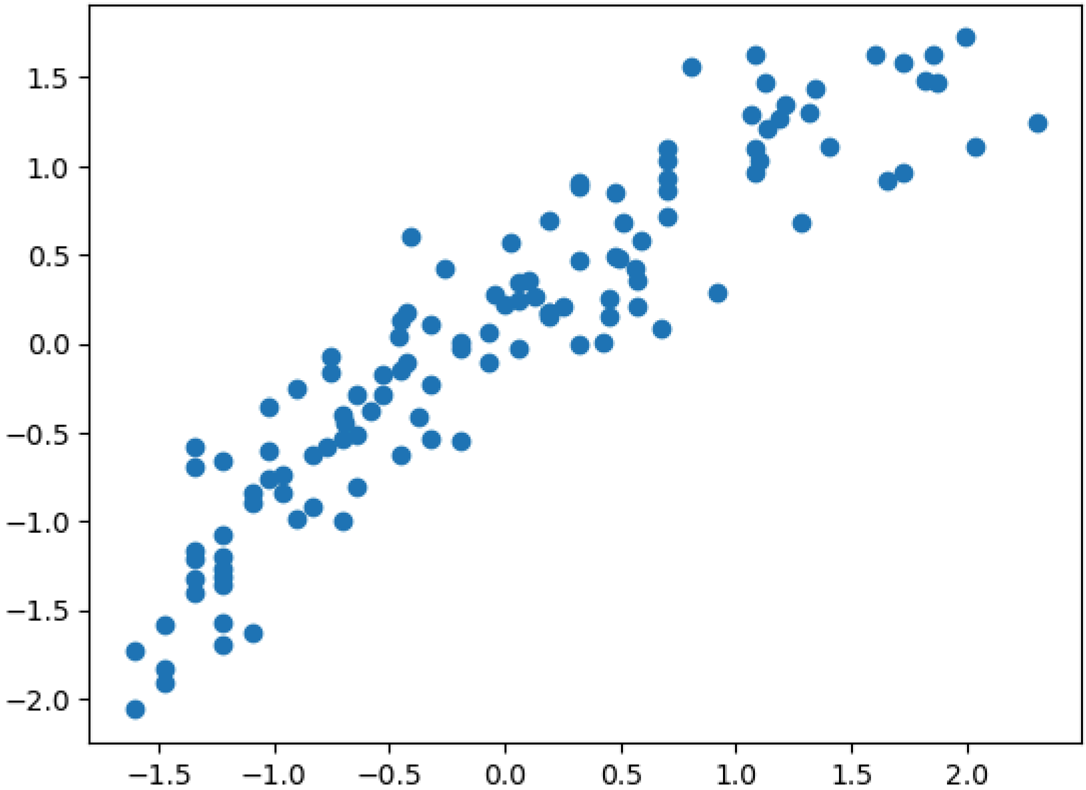





In [92]:
```
# predict mileage (mpg) for a set of attributes not in the training or test set
y_pred = regression_model.predict(x_test)

# Since this is regression, plot the predicted y value vs actual y values for th
# A good model's prediction will be close to actual leading to high R and R2 val
#plt.rcParams['figure.dpi'] = 500
plt.scatter(y_test['mpg'], y_pred)
```

Out[92]: <matplotlib.collections.PathCollection at 0x1f4b8d55670>




In [ ]:
```
# 5. Inference

**Both Ridge & Lasso regularization performs very well on this data, though Ridg

***This kernel is a work in progress.***
```