In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
```

In [3]:
```
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import r2_score
```

In [4]:
```
data = pd.read_csv(r"C:\Users\DELL\Desktop\FSDS\ML\22nd- l1, l2, scaling\lasso,
data.head()
```

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

In [5]:
```
data = data.drop(['car_name'], axis=1)
```

In [6]:
```
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 [7]:
```
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\DELL\AppData\Local\Temp\ipykernel_5376\202028991.py:1: FutureWarning: er
rors='ignore' is deprecated and will raise in a future version. Use to_numeric wi
thout passing `errors` and catch exceptions explicitly instead
  data=data.apply(pd.to_numeric, errors='ignore')
```

In [8]:
```
data.head()
```

Out[8]:

	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	1	0	
1	15.0	8	350.0	165.0	3693	11.5	70	0	1	0	
2	18.0	8	318.0	150.0	3436	11.0	70	0	1	0	
3	16.0	8	304.0	150.0	3433	12.0	70	0	1	0	
4	17.0	8	302.0	140.0	3449	10.5	70	0	1	0	

Model Buliding

In [9]:
```
X=data.drop(['mpg'],axis=1)
y=data[['mpg']]
```

In [10]:
```
X_s=preprocessing.scale(X)
X_columns=X.columns
X_s=pd.DataFrame(X_s, columns= X_columns)
y_s=preprocessing.scale(y)
y_columns=y.columns
y_s=pd.DataFrame(y_s, columns= y_columns)
```

In [13]:
```
X_train, X_test, y_train,y_test = train_test_split(X_s, y_s, test_size = 0.30, r
X_train.shape
```

Out[13]:
```
(278, 10)
```

Simple Linear Regression

In [14]:
```
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
```

Regularized Ridge Regression

In [15]:
```
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]]
```

Regularized Lasso Regression

In [16]:
```
lasso_model = Lasso(alpha = 0.1)
lasso_model.fit(X_train, y_train)

print('Lasso model coef: {}'.format(lasso_model.coef_))
```

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

Score Comparision

In [17]:
```
#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
0.8513421387780066
*************************
0.8343617931312616
0.8518882171608506
*************************
0.7938010766228453
0.8375229615977083
```

Model Parameter Tuning

In [18]:
```
data_train_test = pd.concat([X_train, y_train], axis =1)
data_train_test.head()
```

Out[18]:

	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 [19]:
```
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[19]:
```
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]:
```
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, 21 Aug 2025   Prob (F-statistic):           3.12e-99
Time:                        15:53:29   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.
```

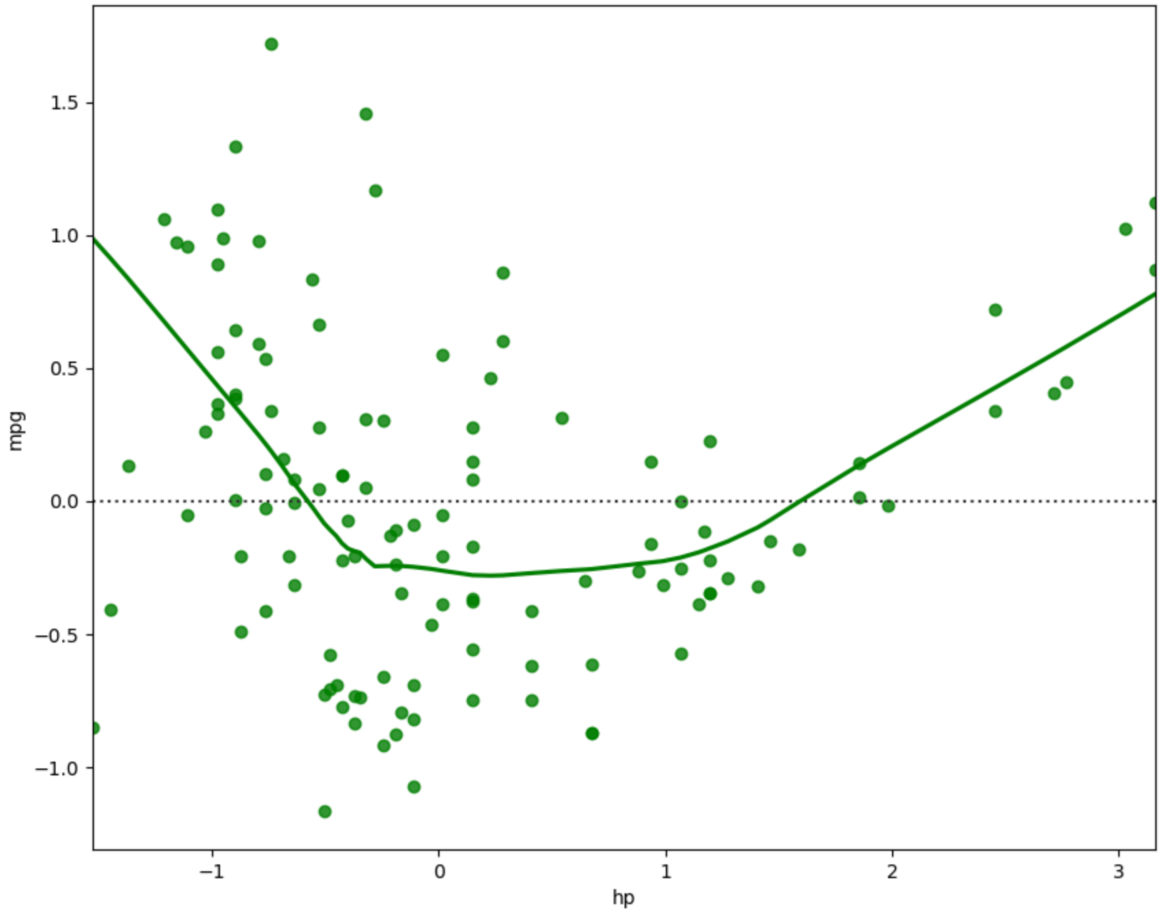
In [21]:

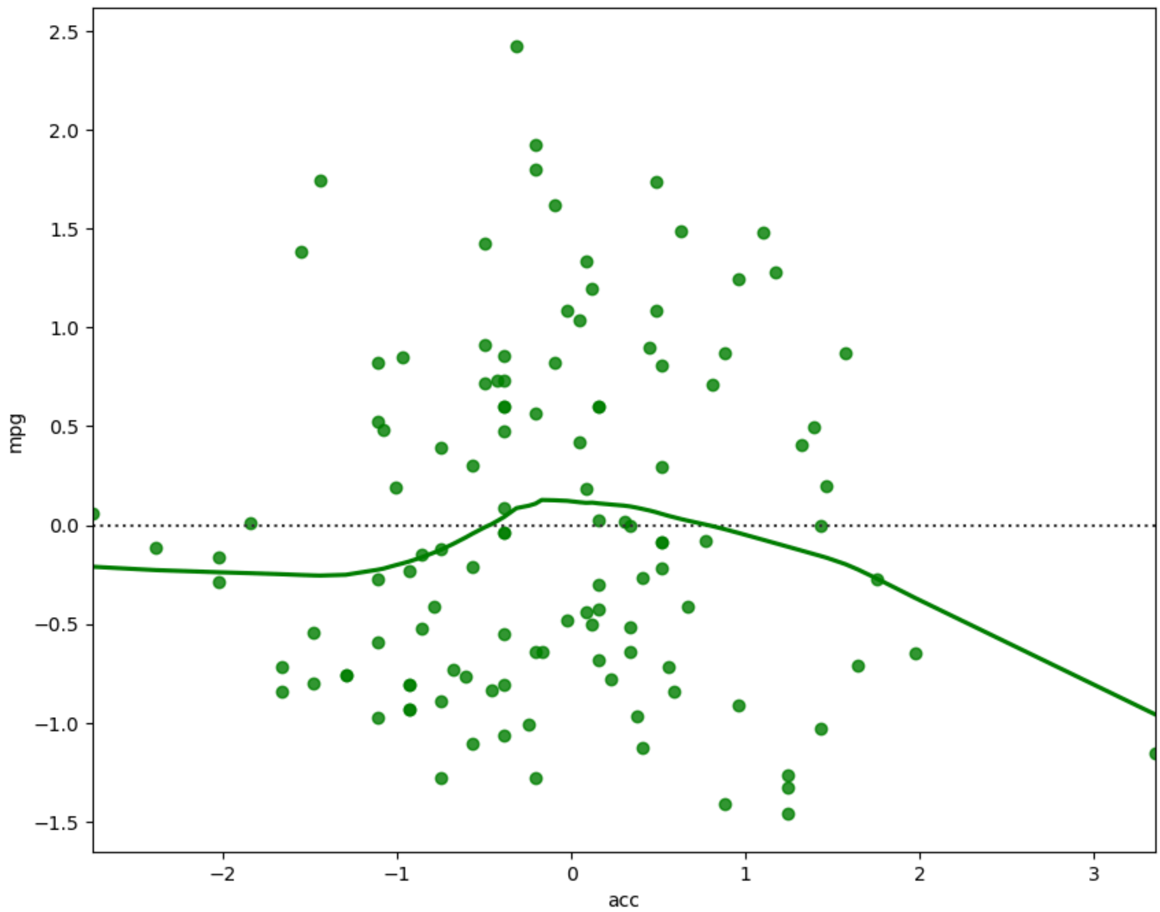
```
mse = np.mean((regression_model.predict(X_test)-y_test)**2)
import math
rmse = math.sqrt(mse)
print('Root Mean Squared Error: {}'.format(rmse))
```

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

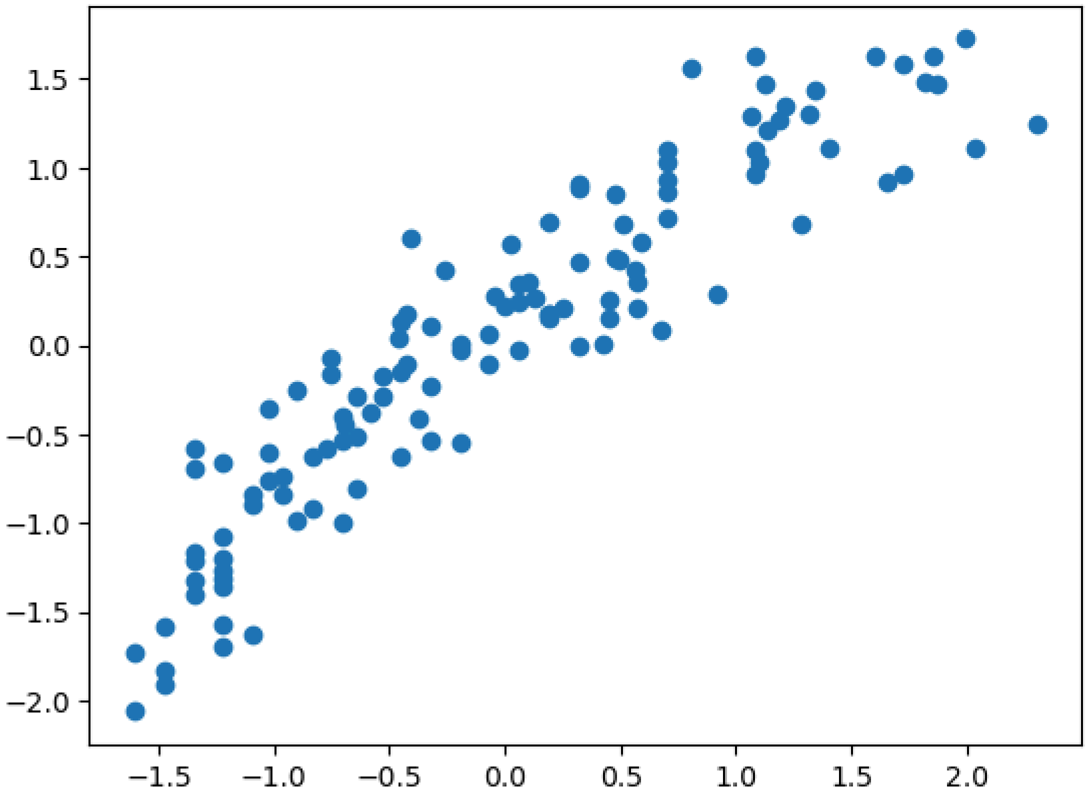
In [25]:
```
fig = plt.figure(figsize=(10,8))
sns.residplot(x= X_test['hp'], y= y_test['mpg'], color='green', lowess=True )
plt.show()

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





In [26]:
```
y_pred = regression_model.predict(X_test)
plt.scatter(y_test['mpg'], y_pred)
plt.show()
```



In [ ]:

In [ ]:

```

```