Import packages and observe dataset

In [1]:

```
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline


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 [3]:

```
data = pd.read_csv(r"C:\Users\chitt\Downloads\car-mpg.csv")
```

In [5]:

```
data
```

Out[5]:

	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 [7]:

```
data.head()
```

Out[7]:

	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 [13]:

```
# Drop 'car_name' if it exists
if 'car_name' in data.columns:
    data = data.drop(['car_name'], axis=1)

# Replace numeric origin values with corresponding labels
if 'origin' in data.columns:
    data['origin'] = data['origin'].replace({1: 'america', 2: 'europe', 3: 'asia

# Apply one-hot encoding to the 'origin' column
if 'origin' in data.columns:
    data = pd.get_dummies(data, columns=['origin'], dtype=int)

# Replace '?' with NaN
data = data.replace('?', np.nan)

# Convert all columns to numeric where possible (non-numeric values will become
data = data.apply(lambda x: pd.to_numeric(x, errors='coerce'))

# Fill NaN values with the median of each column
data = data.apply(lambda x: x.fillna(x.median()), axis=0)
```

In [15]:

```
data.tail()
```

Out[15]:

	mpg	cyl	disp	hp	wt	acc	yr	car_type	origin_america	origin_asia	origir
393	27.0	4	140.0	86.0	2790	15.6	82	1	1	0	
394	44.0	4	97.0	52.0	2130	24.6	82	1	0	0	
395	32.0	4	135.0	84.0	2295	11.6	82	1	1	0	
396	28.0	4	120.0	79.0	2625	18.6	82	1	1	0	
397	31.0	4	119.0	82.0	2720	19.4	82	1	1	0	

In [17]:

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

In [19]:

```
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 [21]:

```
X_s
```

Out[21]:

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

398 rows × 10 columns

In [23]:

```
y_s
```

Out[23]:

	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 [25]:

```
data.shape
```

Out[25]:

```
(398, 11)
```

In [27]:

```
X_train, X_test,y_train,y_test = train_test_split(X_s,y_s, test_size = 0.20,rand
X_train.shape
```

Out[27]:
```
(318, 10)
```

Simple Linear Model

In [29]:
```
#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_

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

```
The coefficient for cyl is 0.24638776053571607
The coefficient for disp is 0.29177092098664514
The coefficient for hp is -0.18081621820393654
The coefficient for wt is -0.6675530609868133
The coefficient for acc is 0.06537309205777078
The coefficient for yr is 0.348177025942672
The coefficient for car_type is 0.3339231253960362
The coefficient for origin_america is -0.08117984631927024
The coefficient for origin_asia is 0.06986098209664919
The coefficient for origin_europe is 0.030003161242288134
The intercept is -0.018006831370923248
```

Regularized Ridge Regression

In [31]:
```
ridge_model = Ridge(alpha = 0.4)
ridge_model.fit(X_train, y_train)

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

```
Ridge model coef: [[ 0.24242411  0.28008024 -0.18071842 -0.65711583  0.06353256
0.34721777
   0.32998816 -0.08077573  0.06989674  0.02945199]]
```

Regularized Lasso Regression

In [35]:
```
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.         -0.07247557 -0.45867691  0.
0.2698134
  0.11341188 -0.04988145  0.          0.        ]
```

Score Comparison

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

```
0.8373422857977738
0.8474768646673948
*************************
0.8373258758714116
0.8471902731156343
*************************
0.8007202116330951
0.8283046020148332
```

```
In [ ]:
```