Prediction of Fuel Efficiency of a car

A PROJECT SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE CERTIFICATE OF

Post Graduate Program

In

Data Science, Business Analytics and Big Data

BY

Shwetabh Kumar Gupta Shaz Syyed

UNDER THE GUIDANCE OF

Mr. Vinay Kulkarni

Assistant Professor

Aegis

SCHOOL OF BUSINESS
SCHOOL OF DATA SCIENCE
SCHOOL OF CYBER SECURITY
SCHOOL OF TELECOMMUNICATION

Prediction of Fuel Efficiency of a car

Background

-Purpose

- The automotive industry is extremely competitive. With increasing fuel prices and picky consumers, automobile makers are constantly optimizing their processes to increase fuel efficiency. And if the company could have a reliable estimator for a car's mpg given some known specifications about the vehicle then, then they can beat competitors in the market by both having a more desirable vehicle that is also more efficient, reducing wasted R&D costs and gaining large chunks of the market.

-About the Data

- We have a dataset taken from the StatLib library which is maintained at Carnegie Mellon University. The dataset was used in the 1983 American Statistical Association Exposition. The data concerns city-cycle fuel consumption in miles per gallon.

-Understanding the Variables

- The dataset contains mpg, cylinders, horsepower, weight, acceleration, etc., which should all be self-explanatory.
- Displacement is the volume of the car's engine, usually expressed in litters or cubic centimetres.
- Origin is a discrete value from 1 to 3. This dataset does not describe it beyond that, but for this notebook we assumed 1 to be American-origin vehicle, 2 is European-origin, 3 is Asia/elsewhere.
- Model year is given as a decimal number representing the last two digits of the 4-digit year (eg.
- 1970 is model year = 70).
- Our model has a dimension of (398,9) which includes continuous and categorical Features
- Our model in this dataset will be trained on many different cars with the help of different algorithms, and it should give us a good estimate for our unknown car's mpg.

Overview of the Dataset

- We imported the required libraries in python such as pandas, scipy.stats, matplotlib, sklearn, etc.
- We imported the "auto-mpg.data-original" file and viewed it by head() command under numpy to get a brief overview of columns and data.

```
names = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight','acceleration', 'model_year', 'origin', 'car_name']
df1 = pd.read_table('auto-mpg.data-original', delim_whitespace=True, names=names)
df1.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
0	18.0	8.0	307.0	130.0	3504.0	12.0	70.0	1.0	chevrolet chevelle malibu
1	15.0	8.0	350.0	165.0	3693.0	11.5	70.0	1.0	buick skylark 320
2	18.0	8.0	318.0	150.0	3436.0	11.0	70.0	1.0	plymouth satellite
3	16.0	8.0	304.0	150.0	3433.0	12.0	70.0	1.0	amc rebel sst
4	17.0	8.0	302.0	140.0	3449.0	10.5	70.0	1.0	ford torino

Data Cleaning

- We first checked for Null Value in the Auto Mpg Dataset and found a total of 14 NAN values in Mpg and Horsepower.

```
#null value check
print(len(df1)-df1.count()) #column wise shows the presence of nan value
print(len(df1)-len(df1.dropna())) #gives number of rows contating nan
```

```
mpg             8
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
model_year      0
origin          0
car_name        0
dtype: int64
14
```

```
df1[df1.isnull().any(axis=1)] #showing columns with nans
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
10	NaN	4.0	133.0	115.0	3090.0	17.5	70.0	2.0	citroen ds-21 pallas
11	NaN	8.0	350.0	165.0	4142.0	11.5	70.0	1.0	chevrolet chevelle concours (sw)
12	NaN	8.0	351.0	153.0	4034.0	11.0	70.0	1.0	ford torino (sw)
13	NaN	8.0	383.0	175.0	4166.0	10.5	70.0	1.0	plymouth satellite (sw)
14	NaN	8.0	360.0	175.0	3850.0	11.0	70.0	1.0	amc rebel sst (sw)
17	NaN	8.0	302.0	140.0	3353.0	8.0	70.0	1.0	ford mustang boss 302
38	25.0	4.0	98.0	NaN	2046.0	19.0	71.0	1.0	ford pinto
39	NaN	4.0	97.0	48.0	1978.0	20.0	71.0	2.0	volkswagen super beetle 117
133	21.0	6.0	200.0	NaN	2875.0	17.0	74.0	1.0	ford maverick
337	40.9	4.0	85.0	NaN	1835.0	17.3	80.0	2.0	renault lecar deluxe
343	23.6	4.0	140.0	NaN	2905.0	14.3	80.0	1.0	ford mustang cobra
361	34.5	4.0	100.0	NaN	2320.0	15.8	81.0	2.0	renault 18i
367	NaN	4.0	121.0	110.0	2800.0	15.4	81.0	2.0	saab 900s
382	23.0	4.0	151.0	NaN	3035.0	20.5	82.0	1.0	amc concord dl

- So we removed them with dropna() function and checked again. The dataset was now fine and ready to be explored.

```
#Removing nan's
df1=df1.dropna()
```

```
df1[df1.isnull().any(axis=1)] #checked to see if nan's removed.
```

mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name

```
print(df1.describe())
print(df1.shape)
```

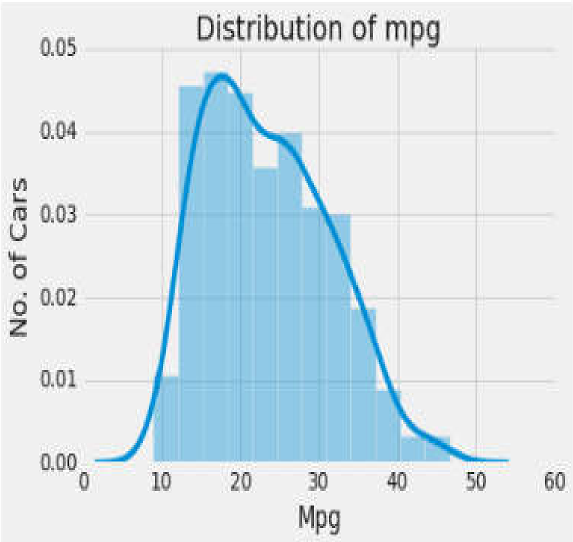
```
              mpg   cylinders  displacement  horsepower       weight  \
count  392.000000  392.000000    392.000000  392.000000   392.000000
mean    23.445918    5.471939    194.411990  104.469388  2977.584184
std      7.805007    1.705783    104.644004   38.491160   849.402560
min      9.000000    3.000000     68.000000   46.000000  1613.000000
25%     17.000000    4.000000    105.000000   75.000000  2225.250000
50%     22.750000    4.000000    151.000000   93.500000  2803.500000
75%     29.000000    8.000000    275.750000  126.000000  3614.750000
max     46.600000    8.000000    455.000000  230.000000  5140.000000

       acceleration  model_year      origin
count    392.000000  392.000000  392.000000
mean      15.541327   75.979592    1.576531
std        2.758864    3.683737    0.805518
min        8.000000   70.000000    1.000000
25%       13.775000   73.000000    1.000000
50%       15.500000   76.000000    1.000000
75%       17.025000   79.000000    2.000000
max       24.800000   82.000000    3.000000
(392, 9)
```

Exploratory Data Analysis

Steps and Conclusion:

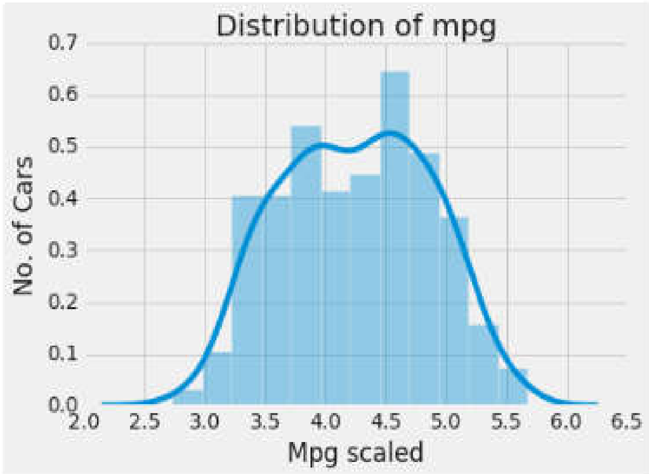
- We first tried visualizing the target variable that is Mpg with distplot. Upon plotting we realized that the plot looked right skewed so we applied box cox Transformation in a view to appeal it closer to normal distribution. We got a low Shapiro-Wikiscore but the plot now tends to be normally distributed

```
#Visualize target variable mpg
plt.style.use("fivethirtyeight")
sns.distplot(df1["mpg"])
plt.xlabel("Mpg")
plt.ylabel("No. of Cars")
plt.title("Distribution of mpg")
sns.despine()
```




```
z=stats.boxcox(df1['mpg'])[0]
plt.style.use("fivethirtyeight")
sns.distplot(z)
plt.xlabel("Mpg scaled")
plt.ylabel("No. of Cars")
plt.title("Distribution of mpg")
sns.despine()
stats.boxcox(df1['mpg'])[1]
stats.shapiro(z)[1]
```

```
0.00013592593313660473
```

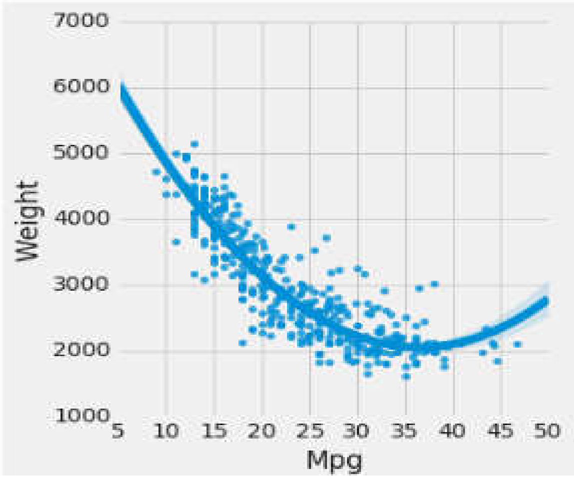


- We try to see the outliers in the target variable by plotting a box plot. We seem to find no outlier and we move ahead.

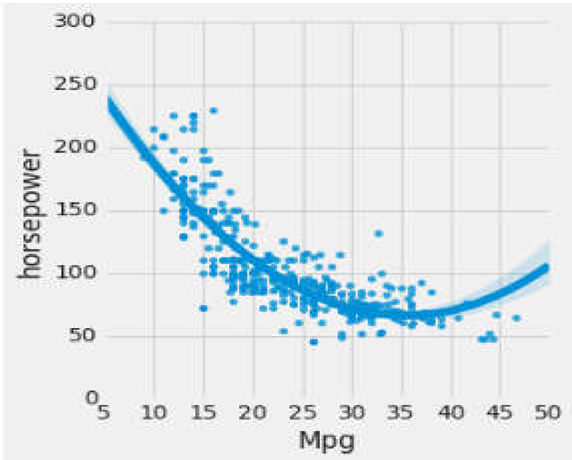
```
#Checking for outliers in the mpg
sns.boxplot(df1["mpg"])
plt.xlabel("Mpg")
plt.title("Box Plot of Mpg")
plt.show()
```

- We now want to see how response variable depends on predictors individually and try to find out some dependency. With the help of lmplot and despine, we are able to find some pattern. We see that displacement, weight and horsepower seem to be inversely related to the target variable mpg whereas scatterplot of year and acceleration seem to be increasing linearly varying with mpg.

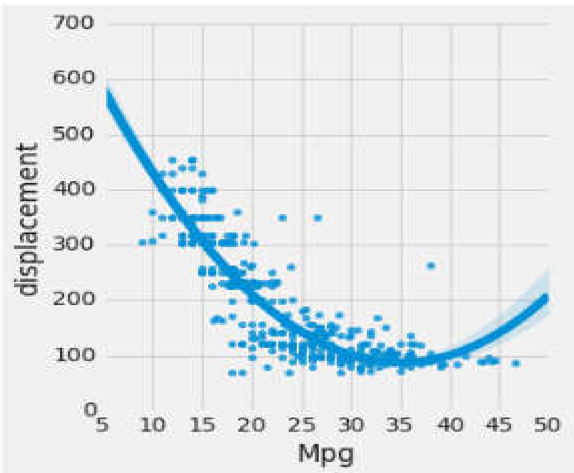
```
sns.lmplot("mpg","weight",df1,order=2)
plt.xlabel("Mpg")
plt.ylabel("Weight")
sns.despine()
```



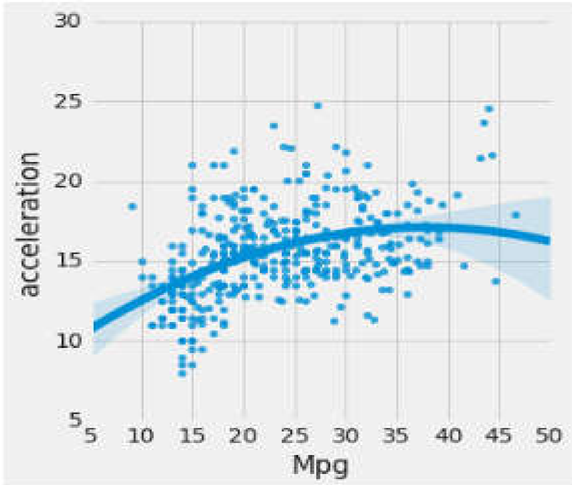

```
sns.lmplot("mpg","horsepower",df1,order=2)
plt.xlabel("Mpg")
plt.ylabel("horsepower")
sns.despine()
```



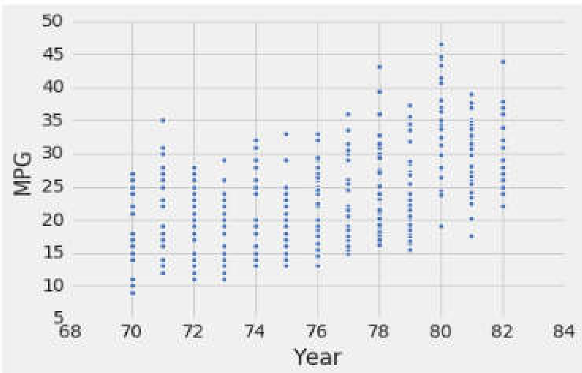

```
sns.lmplot("mpg","displacement",df1,order=2)
plt.xlabel("Mpg")
plt.ylabel("displacement")
sns.despine()
```




```
sns.lmplot("mpg","acceleration",df1,order=2)
plt.xlabel("Mpg")
plt.ylabel("acceleration")
sns.despine()
```

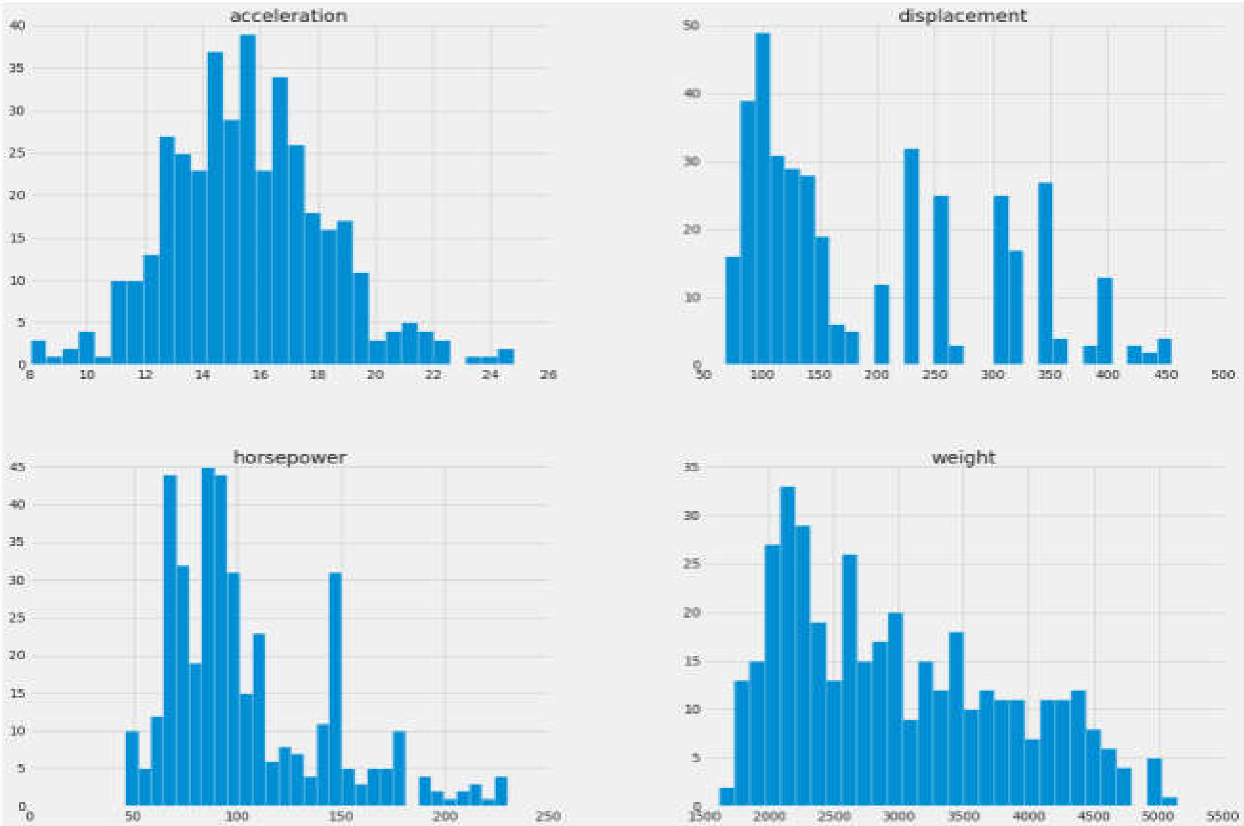



```
sns.scatterplot(df1["model_year"], df1["mpg"])
plt.xlabel("Year")
plt.ylabel("MPG")
sns.despine()
```



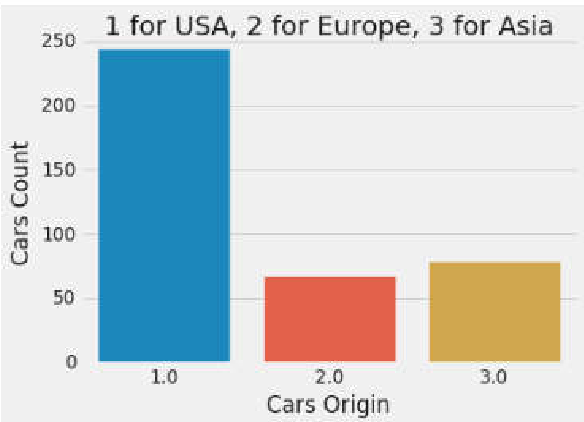
- Now we visualize the distribution of the continuous features like acceleration, weight, displacement, horsepower just like we did for mpg but with histogram. Some histograms are tail heavy which can make it harder for some Machine Learning algorithms to detect patterns. It can be useful to transform these features to make them more normally distributed. Continuous features are distributed on the same scale. However, the scale differs from the multi-valued discrete feature cylinders. Depending on the algorithm, further scaling might be needed.

```
#Distribution of Continuous Features
cont_features = ["displacement", "horsepower", "weight", "acceleration"]
df1[cont_features].hist(bins=30, figsize=(20, 15))
sns.despine()
```

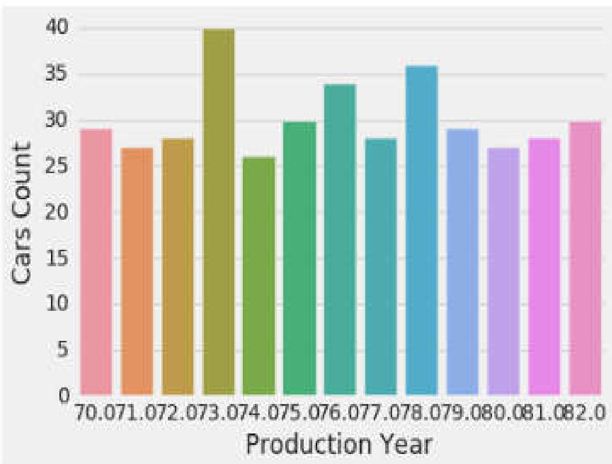


- Now we would want to see the distribution of Categorical Features like origin, year cylinder. We see here realize cylinder seems to play an important role in predicting mpg.

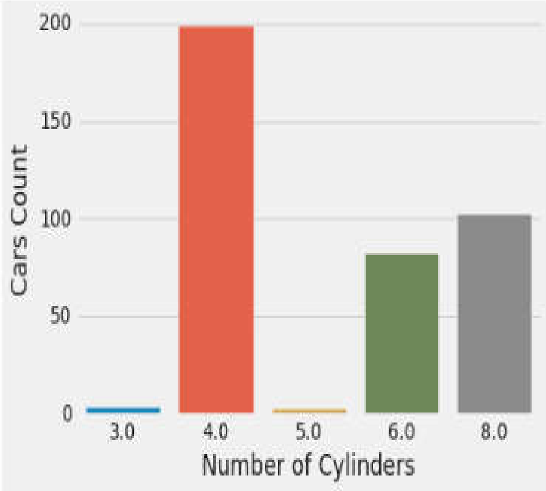
```
#Distribution of Categorical Features

sns.countplot(df1.origin)
plt.ylabel("Cars Count")
plt.xlabel("Cars Origin")
plt.title("1 for USA, 2 for Europe, 3 for Asia")
sns.despine()
```



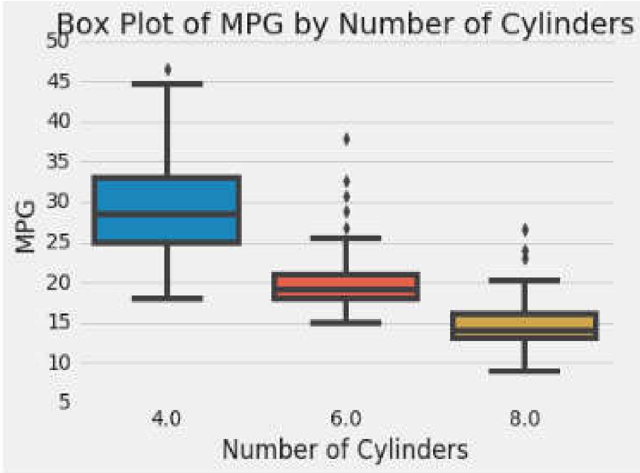

```
sns.countplot(df1.model_year)
plt.ylabel("Cars Count")
plt.xlabel("Production Year")
sns.despine()
```



```
sns.countplot(df1.cylinders)
plt.xlabel("Number of Cylinders")
plt.ylabel("Cars Count")
sns.despine()
```

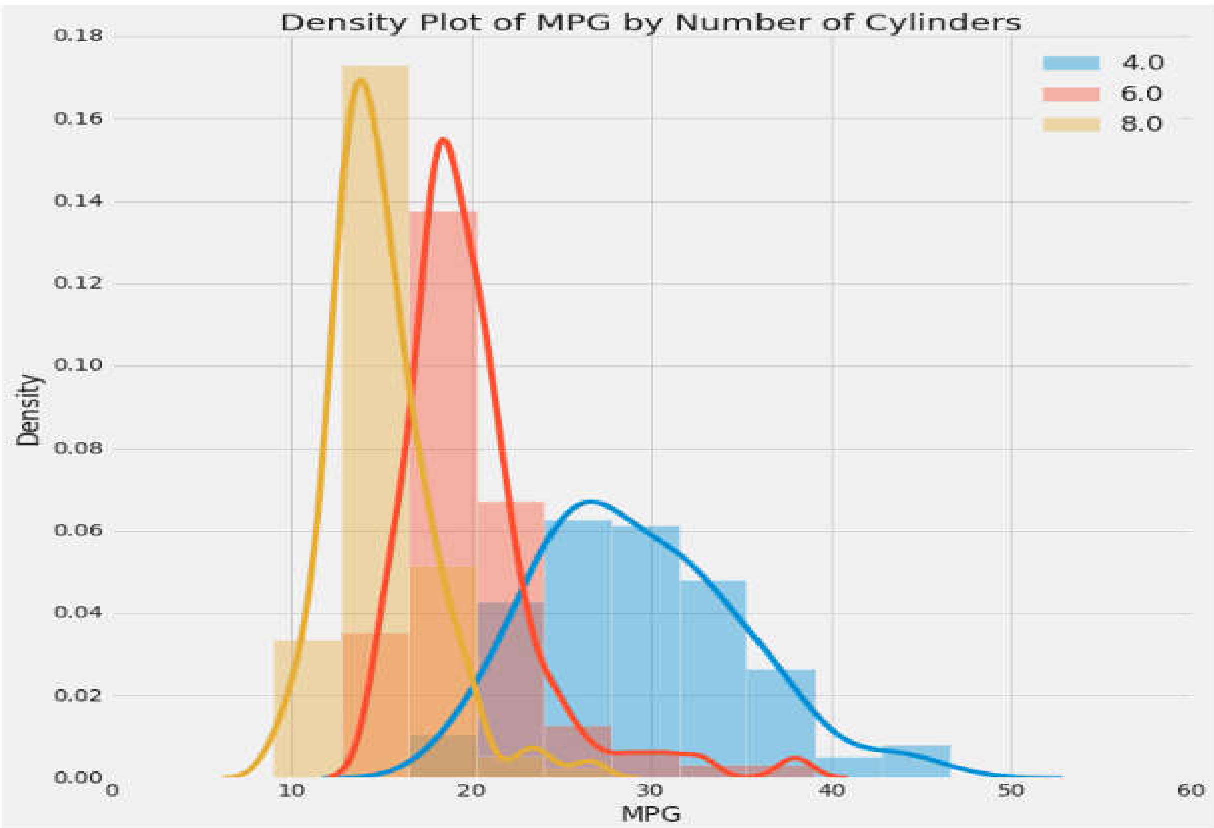



```
df1_cylinders = df1[~df1["cylinders"].isin([3, 5])]
sns.boxplot(x="cylinders", y="mpg", data=df1_cylinders)
plt.xlabel("Number of Cylinders")
plt.ylabel("MPG")
plt.title("Box Plot of MPG by Number of Cylinders")
sns.despine()
```



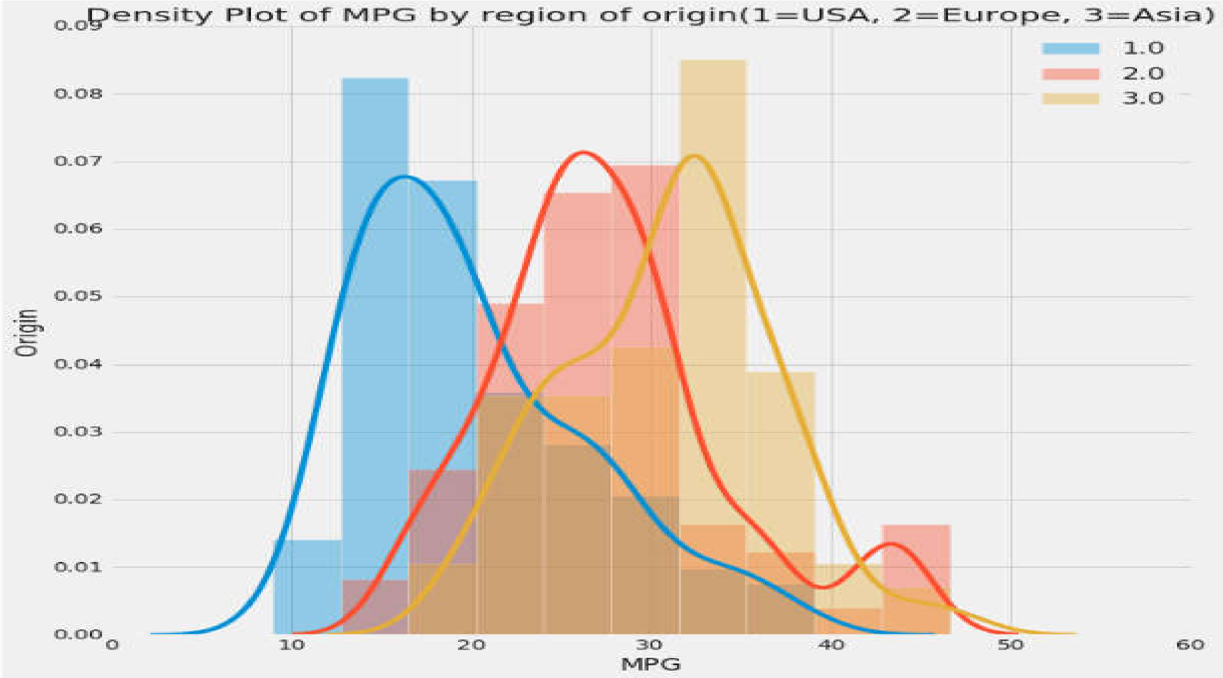
- We try and plot the density plot of mpg by no. of cylinders and we see that a relationship exists between the input features acceleration and year and the target mpg. Mpg seems to increase linearly with these features.

```
_, bins = np.histogram(df1_cylinders["mpg"])
g = sns.FacetGrid(df1_cylinders, hue="cylinders", height=10)
g = g.map(sns.distplot, "mpg", bins=bins)
plt.ylabel("Density")
plt.xlabel("MPG")
plt.title("Density Plot of MPG by Number of Cylinders")
plt.legend()
plt.show()
```



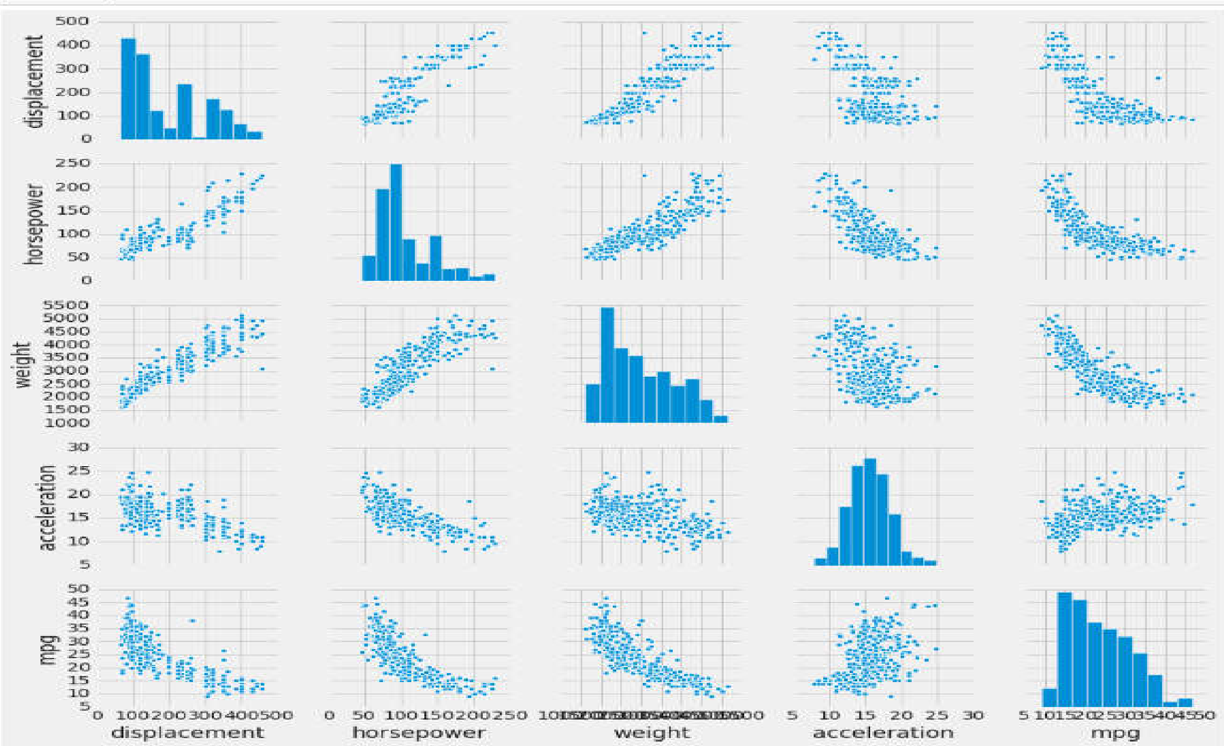
- Also we plot density plot of mpg by region of origin(1 for USA, 2 for Europe and 3 for Asia) and we conclude that a car made in Asia has a higher mgp then a car made in Europe on an average. A car made in Europe has a higher mpg that a car made in USA on average

```
_, bins = np.histogram(df1_cylinders["mpg"])
g = sns.FacetGrid(df1_cylinders, hue="origin", height=10)
g = g.map(sns.distplot, "mpg", bins=bins)
plt.ylabel("Origin")
plt.xlabel("MPG")
plt.title("Density Plot of MPG by region of origin(1=USA, 2=Europe, 3=Asia)")
plt.legend()
plt.show()
```



- After this we try to check feature collinearity via pairplot. Here we see most of the plot seem to be linearly varying

```
#Checking feature Collinearity
continous = ["displacement", "horsepower", "weight", "acceleration", "mpg"]

sns.pairplot(df1[continous].dropna())
plt.show()
```



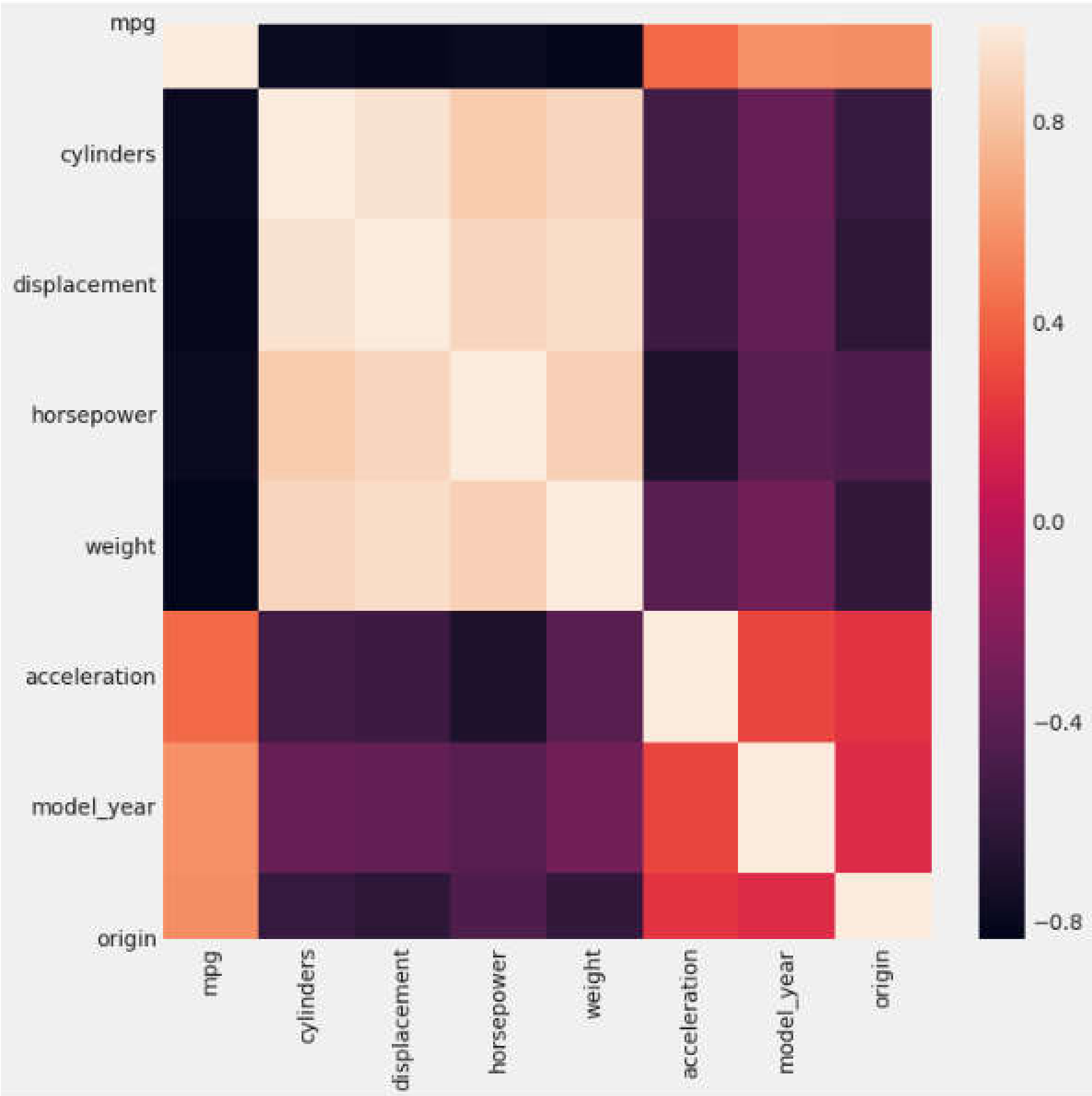
- Now we calculate the Pearson's correlation coefficient with. corr() function and visualize it with the heat map. By this we get a confirmation that cylinder, displacement, weight are highly correlated to these features as well.

```
#Checking for Pearson correlation
df1[continous].corr()
```

	displacement	horsepower	weight	acceleration	mpg
displacement	1.000000	0.897257	0.932994	-0.543800	-0.805127
horsepower	0.897257	1.000000	0.864538	-0.689196	-0.778427
weight	0.932994	0.864538	1.000000	-0.416839	-0.832244
acceleration	-0.543800	-0.689196	-0.416839	1.000000	0.423329
mpg	-0.805127	-0.778427	-0.832244	0.423329	1.000000

```
plt.figure(figsize=(10,10))
sns.heatmap(df1.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x198205afb08>
```



Feature Engineering and Selection

- Previously we saw an inverse relation of mpg with displacement, weight, horsepower so here we take 3 new continuous variable named as inv_displacement, inv_weight, inv_horsepower(we have taken there inverse) and box coz mpg (normalized) data.

Modelling

We here try out with multiple Machine Learning Model and try to create a model which could predict the fuel efficiency of cars and then would select best Model out of them.

1. Multiple Linear Regression

-Here we split the data into train/test sets with a 70/30 ratio and find mean/variance of train set for scaling with the features- ['inv_displacement', 'inv_horsepower', 'inv_weight','acceleration'] and we achieve the R^2 value of 0.7559 and RMSE value of 3.7170

```
#Split data into train/test sets and find mean/variance of train set for scaling

feature = ['inv_displacement', 'inv_horsepower', 'inv_weight','acceleration']
response1=['mpg']
X_train, X_test, y_train, y_test = train_test_split(df2[feature], df2[response1], test_size=0.3, random_state=10)
```

```
regressor = LinearRegression()
regressor.get_params()
regressor.fit(X_train,y_train)
y_predicted = regressor.predict(X_test)
rmse = sqrt(mean_squared_error(y_true=y_test,y_pred=y_predicted))
a=rmse
print("Rmse",rmse)
print("r^2=",regressor.score(X_test, y_test))
```

```
Rmse 3.717009309941545
r^2= 0.7559701022328772
```

2. RidgeCv with Polynomial Features

-Here we scale the X_test, X_train, y_test and y_train and get X_test scaled, X_train scaled, y_test scaled and y_train scaled and then we use the Ridge CV with polynomial features with (0,2,10) as alphas and we make a pipeline and fit the model. With this we achieve the R^2 value of 0.808 and RMSE OF 0.4326.

```
scalerX = preprocessing.StandardScaler().fit(X_train)
scalery = preprocessing.StandardScaler().fit(y_train)

X_train_scaled = scalerX.transform(X_train)
X_test_scaled = scalerX.transform(X_test)
y_train_scaled = scalery.transform(y_train)
y_test_scaled = scalery.transform(y_test)
```

```
# Use RidgeCV with PolynomialFeatures
alphas = np.logspace(0,2,10)

model = make_pipeline(preprocessing.PolynomialFeatures(4, interaction_only=True), \
                      RidgeCV(alphas=alphas))
```

```
# fit model and score it based on r^2 and rmse
model.fit(X_train_scaled, y_train_scaled);
y_pred_scaled = model.predict(X_test_scaled)
print('r^2=',model.score(X_test_scaled, y_test_scaled))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test_scaled, y_pred_scaled)))
```

```
r^2= 0.8081563117485734
Root Mean Squared Error: 0.43624664555350995
```

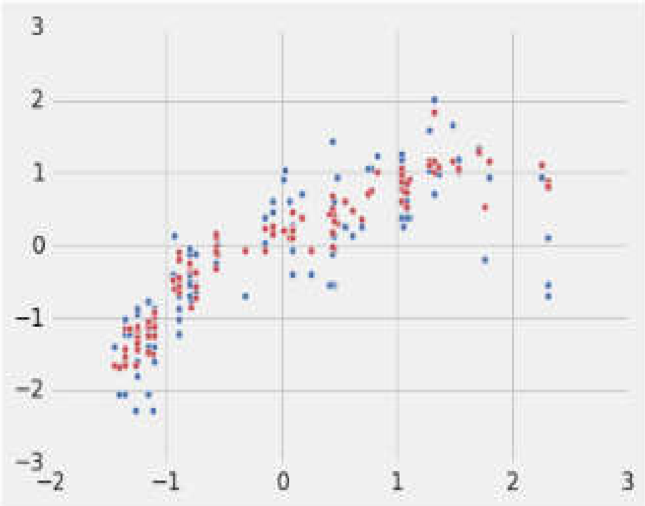
Now we would like to find which features are important so we find the coefficients.

```
# Which predictors were important?
print(model.steps[0][1].get_feature_names(input_features=feature))
print(model.steps[1][1].coef_)
#model.steps[1][1].intercept_
```

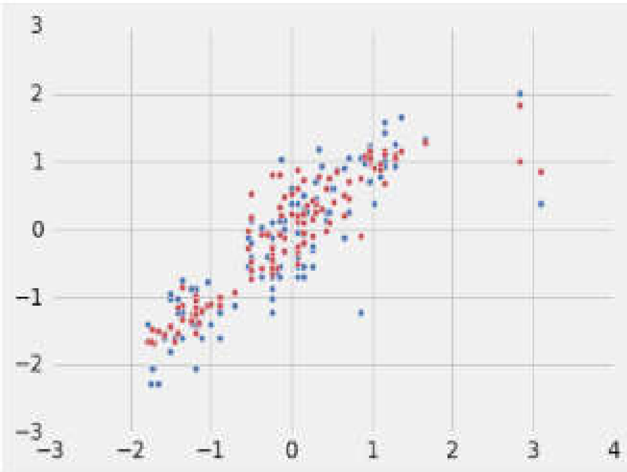
```
['1', 'inv_displacement', 'inv_horsepower', 'inv_weight', 'acceleration', 'inv_displacement inv_horsepower', 'inv_displacement inv_weight', 'inv_displacement acceleration', 'inv_horsepower inv_weight', 'inv_horsepower acceleration', 'inv_weight acceleration', 'inv_displacement inv_horsepower inv_weight', 'inv_displacement inv_horsepower acceleration', 'inv_displacement inv_weight acceleration', 'inv_horsepower inv_weight acceleration', 'inv_displacement inv_horsepower inv_weight acceleration']
[[ 0.          0.24557295  0.37164215  0.34955817 -0.10375852  0.02199679
  -0.09860326 -0.00511496 -0.07716954  0.03615172 -0.01960322 -0.00076355
   0.09447423  0.04731434 -0.13348572 -0.03273329]]
```

-Now we make some plots to visualize accuracy of model predictions along with finding coefficient

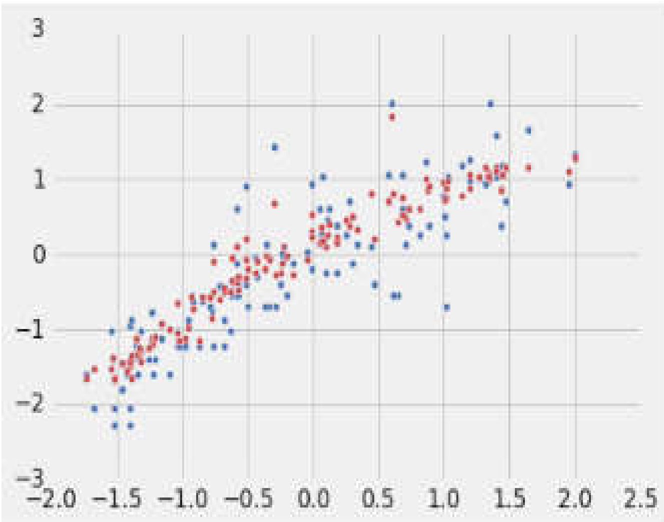
```
col = 0 # inv_displacement
plt.scatter(X_test_scaled[:, col], y_test_scaled);
plt.scatter(X_test_scaled[:, col], y_pred_scaled, c='r')
```



```
col = 1 # inv_horsepower
plt.scatter(X_test_scaled[:, col], y_test_scaled);
plt.scatter(X_test_scaled[:, col], y_pred_scaled, c='r'
```

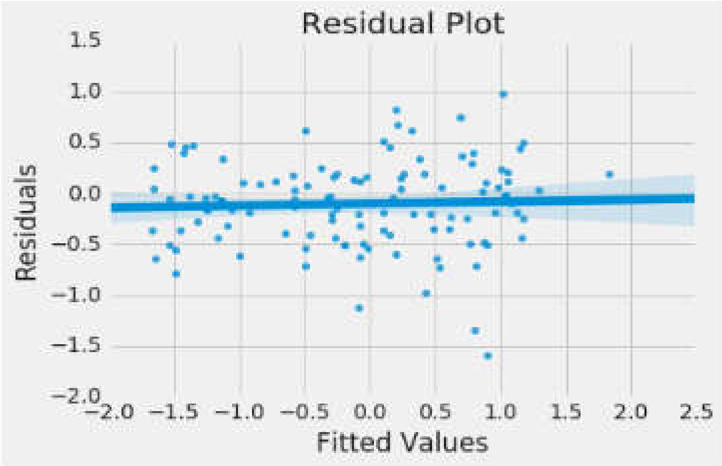


```
col = 2 # inv_weight
plt.scatter(X_test_scaled[:, col], y_test_scaled);
plt.scatter(X_test_scaled[:, col], y_pred_scaled, c='r');
```



And now we plot a residual Plot to see if our assumptions still hold true. And we don't seem to have a pattern in the residual error which seems to be a good sign.

```
ax = sns.regplot(y_pred_scaled[:,0], y_test_scaled[:,0]-y_pred_scaled[:,0]);
ax.set(xlabel='Fitted Values', ylabel='Residuals', title='Residual Plot')
plt.show();
```



3. Gradient Boosting

Here we fit the X_train and Y_train and get the parameters.

```
# gb_regressor = GradientBoostingRegressor(n_estimators=4000)
gb_regressor.fit(X_train,y_train)
```

```
gb_regressor.get_params()
```

```
{'alpha': 0.9,
 'ccp_alpha': 0.0,
 'criterion': 'friedman_mse',
 'init': None,
 'learning_rate': 0.1,
 'loss': 'ls',
 'max_depth': 3,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 4000,
 'n_iter_no_change': None,
 'presort': 'deprecated',
 'random_state': None,
 'subsample': 1.0,
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': 0,
 'warm_start': False}
```

And now we try to predict with X_test with gb.regresssor and we are able to achieve a good R^2 of 0.8068 and an RMSE of 0.2672

```
y_predicted_gbr = gb_regressor.predict(X_test)
rmse_bgr = sqrt(mean_squared_error(y_true=y_test,y_pred=y_predicted_gbr))
print(rmse_bgr)
print(gb_regressor.score(X_test,y_test))
```



```
0.2672773005161249
0.8068403098561533
```

4. Random Forest

At last we try to predict the mpg with one more algorithm that is random forest. We again fit here training data and predict with the testing data. Here in we are able o achieve an R^2 value of 0.649 and RMSE value of 4.6589

```
from sklearn.model_selection import train_test_split
training,test = train_test_split(df3, train_size = 0.7, test_size = 0.3,shuffle=True)
training, valid = train_test_split(training, train_size = 0.7, test_size =0.3,shuffle=True)
training_label = training.pop('mpg')
test_label = test.pop('mpg')
valid_label = valid.pop('mpg')
```

```
from sklearn.ensemble import RandomForestRegressor
rfc = RandomForestRegressor()
rfc.fit(training, training_label) # train the models
rfc_predict = rfc.predict(test) #test the model
```

```
from sklearn.metrics import mean_squared_error
import math
accuracy = dict()
accuracy['RandomForest RMSE'] = math.sqrt(mean_squared_error(test_label,rfc_predict))
print(accuracy)
print("r^2=",rfc.score(test, test_label))
```

```
{'RandomForest RMSE': 4.658593624616988}
r^2= 0.6942682742976263
```

Conclusion

This is the final result of all the models we built.

Algorithms/Results	R^2	RMSE
Multiple Linear Regression	0.759	3.170
RidgeCv with Polynomial Features	0.808	0.4326
Gradient Boosting	0.8068	0.2672
Random Forest	0.649	4.6589

We see that R^2 value seems to be almost similar for RidgeCv with polynomial features and Gradient Boosting we choose to select the Gradient Boosting method due to relatively low RMSE value compared to other models here in and thereby would help us predict the fuel efficiency of the car with high accuracy.