# Welcome to exploration and analysis of the auto mpg data set.

Welcome to this ipython notebook created for exploration and analysis of the Auto- MPG data-set from UCI Machine Learning Library. The data-set is fairly standard on kaggle but can be accessed separately from the UCI Machine Learning Repository along with many other interesting data-sets. Check http://archive.ics.uci.edu/ml/index.php for more.

This notebook aims primarily to demonstrate use of pandas and seaborn for exploration and visualization of the data-set along with use of scikit learn library to build regression models to predict the Miles Per Gallon(MPG) using the factors provided in the data-set

## So what is the auto-mpg data set?

The following description can be found on the UCI Repository page for the data-set (http://archive.ics.uci.edu/ml/datasets/Auto+MPG)

- This dataset is a slightly modified version of the dataset provided in the StatLib library. In line with the use by Ross Quinlan (1993) in predicting the attribute "mpg", 8 of the original instances were removed because they had unknown values for the "mpg" attribute. The original dataset is available in the file "auto-mpg.data-original".

"The data concerns city-cycle fuel consumption in miles per gallon, to be predicted in terms of 3 multivalued discrete and 5 continuous attributes." (Quinlan, 1993)

But for now we will treat this as a expedition to discover unknown knowledge in the unchartered lands of this dataset.

Let's first ready the equipment we will need for this expedition

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import norm
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
# Input data files are available in the "../input/" directory on kaggle.
from subprocess import check_output
print(check_output(["ls", "../input"]).decode("utf8"))
```

```
auto-mpg.csv
```

We have imported all the packages and libraries we will be using for the initial exploration of data. This notebook will be split into two major sections, majorly:

- Exploration and Visualization using pandas and seaborn packages
- Building evaluating and tuning different regression models using scikit learn package

# Part 1: So let's begin this exploratory journey into the data-set to reveal its hidden secrets!!

In order to begin this exciting journey into the unchartered lands of the auto-mpg data set we first need to know the location of this unexplored land. For our dataset, this location is '../input/auto-mpg.csv' So lets tell python to take us to this place. Since we are the first explorers here, we will call this place data..because we like data..

```
data = pd.read_csv('../input/auto-mpg.csv',index_col='car name')
```

Let's have a look at data

```
print(data.head())
print(data.index)
print(data.columns)
```

```
                           mpg  cylinders  displacement horsepower  weight  \
car name
chevrolet chevelle malibu  18.0          8         307.0        130    3504
buick skylark 320          15.0          8         350.0        165    3693
plymouth satellite         18.0          8         318.0        150    3436
amc rebel sst              16.0          8         304.0        150    3433
ford torino                17.0          8         302.0        140    3449
```

```
                           acceleration  model year  origin
car name
chevrolet chevelle malibu          12.0          70       1
buick skylark 320                  11.5          70       1
plymouth satellite                 11.0          70       1
amc rebel sst                      12.0          70       1
ford torino                        10.5          70       1
Index(['chevrolet chevelle malibu', 'buick skylark 320', 'plymouth satellite',
       'amc rebel sst', 'ford torino', 'ford galaxie 500', 'chevrolet impala',
       'plymouth fury iii', 'pontiac catalina', 'amc ambassador dpl',
       ...
       'chrysler lebaron medallion', 'ford granada l', 'toyota celica gt',
       'dodge charger 2.2', 'chevrolet camaro', 'ford mustang gl', 'vw pickup',
       'dodge rampage', 'ford ranger', 'chevy s-10'],
      dtype='object', name='car name', length=398)
Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'model year', 'origin'],
      dtype='object')
```

So there it is..lots of numbers. We can see that the dataset has the following columns (with their type):

- **mpg**: continuous
- **cylinders**: multi-valued discrete
- **displacement**: continuous
- **horsepower**: continuous
- **weight**: continuous
- **acceleration**: continuous
- **model year**: multi-valued discrete
- **origin**: multi-valued discrete
- **car name**: string (unique for each instance)

```
data.shape
```

```
(398, 8)
```

```
data.isnull().any()
```

```
mpg             False
cylinders       False
displacement    False
horsepower      False
weight          False
acceleration    False
model year      False
origin          False
dtype: bool
```

Nothing seems to be missing

```
data.dtypes
```

```
mpg             float64
cylinders         int64
displacement    float64
horsepower       object
weight            int64
acceleration    float64
model year        int64
origin            int64
dtype: object
```

But then, why is horsepower an object and not a float, the values we saw above were clearly numbers Lets try converting the column using astype()

```
Let's look at the unique elements of horsepower to look for discrepancies
```

```
data.horsepower.unique()
```

```
array(['130', '165', '150', '140', '198', '220', '215', '225', '190',
       '170', '160', '95', '97', '85', '88', '46', '87', '90', '113',
       '200', '210', '193', '?', '100', '105', '175', '153', '180', '110',
       '72', '86', '70', '76', '65', '69', '60', '80', '54', '208', '155',
       '112', '92', '145', '137', '158', '167', '94', '107', '230', '49',
       '75', '91', '122', '67', '83', '78', '52', '61', '93', '148',
       '129', '96', '71', '98', '115', '53', '81', '79', '120', '152',
       '102', '108', '68', '58', '149', '89', '63', '48', '66', '139',
       '103', '125', '133', '138', '135', '142', '77', '62', '132', '84',
       '64', '74', '116', '82'], dtype=object)
```

When we print out all the unique values in horsepower, we find that there is '?' which was used as a placeholder for missing values. Lest remove these entries.

```
data = data[data.horsepower != '?']
```

```
print('?' in data.horsepower)
```

```
False
```

```
data.shape
```

```
(392, 8)
```

```
data.dtypes
```

```
mpg             float64
cylinders         int64
displacement    float64
horsepower       object
weight            int64
acceleration    float64
model year        int64
origin            int64
dtype: object
```

So we see all entries with '?' as place holder for data are removed. However, we the horsepower data is still an object type and not float. That is because pandas coerced the entire column as object when we imported the data set due to '?', so lest change that data

```
data.horsepower = data.horsepower.astype('float')
data.dtypes
```

```
mpg             float64
cylinders         int64
displacement    float64
horsepower      float64
weight            int64
acceleration    float64
model year        int64
origin            int64
dtype: object
```

Now everything looks in order so lets continue, let's describe the dataset

```
data.describe()
```

## Step 1: Let's look at mpg

mean 23.445918 5.471939 194.411990 104.469388 2977.584184 15.541327 75.979592 1.576531

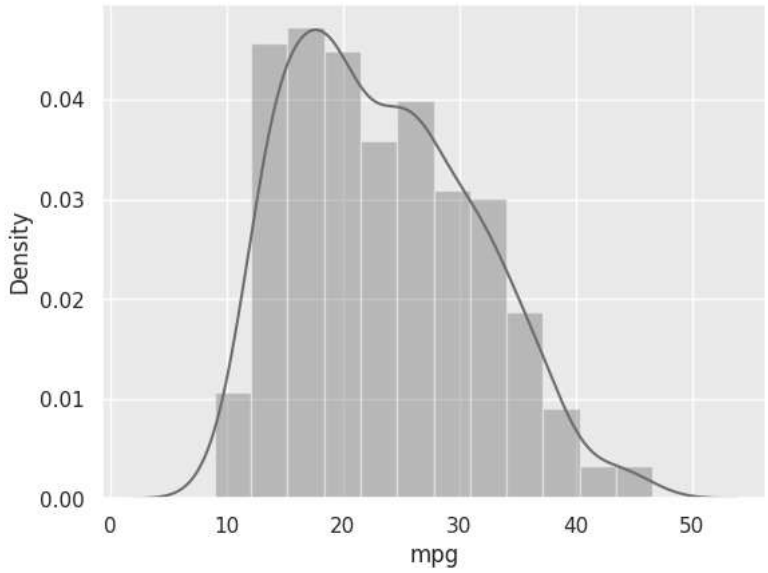
```
data.mpg.describe()
```

```
count    392.000000
mean      23.445918
std        7.805007
min        9.000000
25%       17.000000
50%       22.750000
75%       29.000000
max       46.600000
Name: mpg, dtype: float64
```

So the minimum value is 9 and maximum is 46, but on average it is 23.44 with a variation of 7.8

```
sns.distplot(data['mpg'])
```

```
<Axes: xlabel='mpg', ylabel='Density'>
```




```
print("Skewness: %f" % data['mpg'].skew())
print("Kurtosis: %f" % data['mpg'].kurt())
```

```
Skewness: 0.457092
Kurtosis: -0.515993
```

Using our seaborn tool we can look at mpg:

- Slight of 0,.45
- Kurtosis of -0.51

## Lets visualise some relationships between these data points, but before we do, we need to scale them to same the same range of [0,1]

In order to do so, lets define a function scale

```
def scale(a):
    b = (a-a.min())/(a.max()-a.min())
    return b
```

```
data_scale = data.copy()
```

```
data_scale ['displacement'] = scale(data_scale['displacement'])
data_scale['horsepower'] = scale(data_scale['horsepower'])
data_scale ['acceleration'] = scale(data_scale['acceleration'])
data_scale ['weight'] = scale(data_scale['weight'])
data_scale['mpg'] = scale(data_scale['mpg'])
```

```
data_scale.head()
```

| car name | mpg | cylinders | displacement | horsepower | weight | acceleration | model year | origin |
|---|---|---|---|---|---|---|---|---|
| **chevrolet chevelle malibu** | 0.239362 | 8 | 0.617571 | 0.456522 | 0.536150 | 0.238095 | 70 | 1 |
| **buick skylark 320** | 0.159574 | 8 | 0.728682 | 0.646739 | 0.589736 | 0.208333 | 70 | 1 |
| **plymouth satellite** | 0.239362 | 8 | 0.645995 | 0.565217 | 0.516870 | 0.178571 | 70 | 1 |
| **amc rebel sst** | 0.186170 | 8 | 0.609819 | 0.565217 | 0.516019 | 0.238095 | 70 | 1 |
| **ford torino** | 0.212766 | 8 | 0.604651 | 0.510870 | 0.520556 | 0.148810 | 70 | 1 |

All our data is now scaled to the same range of [0,1]. This will help us visualize data better. We used a copy of the original data-set for this as we will use the data-set later when we build regression models.

```
data['Country_code'] = data.origin.replace([1,2,3],['USA','Europe','Japan'])
data_scale['Country_code'] = data.origin.replace([1,2,3],['USA','Europe','Japan'])
```

```
data_scale.head()
```

| car name | mpg | cylinders | displacement | horsepower | weight | acceleration | model year | origin | Country_code |
|---|---|---|---|---|---|---|---|---|---|
| **chevrolet chevelle malibu** | 0.239362 | 8 | 0.617571 | 0.456522 | 0.536150 | 0.238095 | 70 | 1 | USA |
| **buick skylark 320** | 0.159574 | 8 | 0.728682 | 0.646739 | 0.589736 | 0.208333 | 70 | 1 | USA |
| **plymouth satellite** | 0.239362 | 8 | 0.645995 | 0.565217 | 0.516870 | 0.178571 | 70 | 1 | USA |
| **amc rebel sst** | 0.186170 | 8 | 0.609819 | 0.565217 | 0.516019 | 0.238095 | 70 | 1 | USA |
| **ford torino** | 0.212766 | 8 | 0.604651 | 0.510870 | 0.520556 | 0.148810 | 70 | 1 | USA |

Lets look at MPG's relation to categories

```
var = 'Country_code'
data_plt = pd.concat([data_scale['mpg'], data_scale[var]], axis=1)
f, ax = plt.subplots(figsize=(8, 6))
fig = sns.boxplot(x=var, y="mpg", data=data_plt)
fig.axis(ymin=0, ymax=1)
plt.axhline(data_scale.mpg.mean(),color='r',linestyle='dashed',linewidth=2)
```

```
<matplotlib.lines.Line2D at 0x7be9eb654850>
```

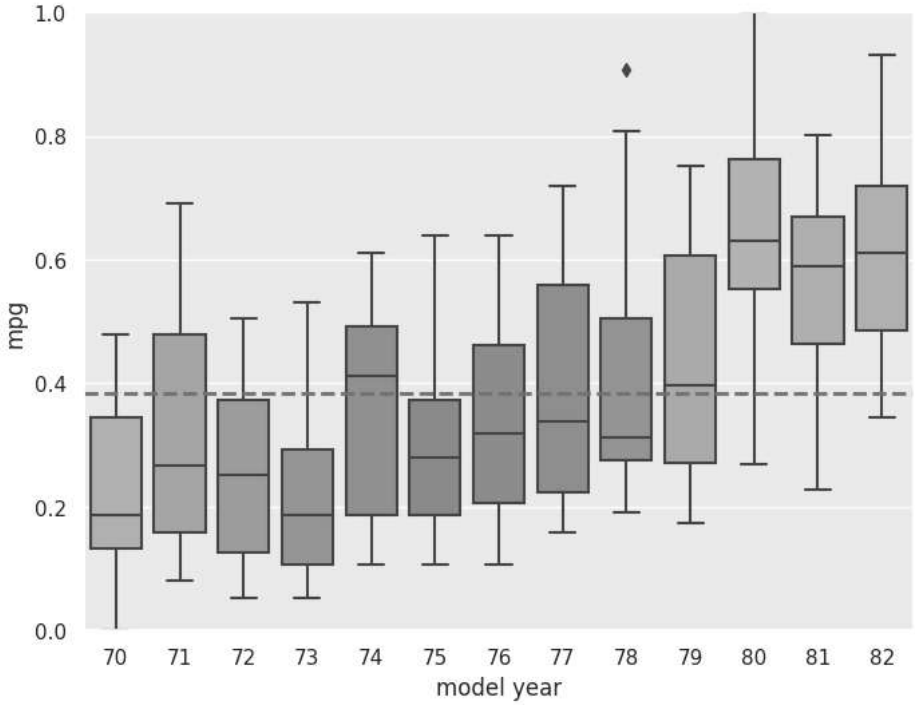

The red line marks the average of the set. From the above plot we can observe:

- Majority of the cars from USA (almost 75%) have MPG below global average.
- Majority of the cars from Japan and Europe have MPG above global average.

## Let's look at the year wise distribution of MPG

```
var = 'model year'
data_plt = pd.concat([data_scale['mpg'], data_scale[var]], axis=1)
f, ax = plt.subplots(figsize=(8, 6))
fig = sns.boxplot(x=var, y="mpg", data=data_plt)
fig.axis(ymin=0, ymax=1)
plt.axhline(data_scale.mpg.mean(),color='r',linestyle='dashed',linewidth=2)
```

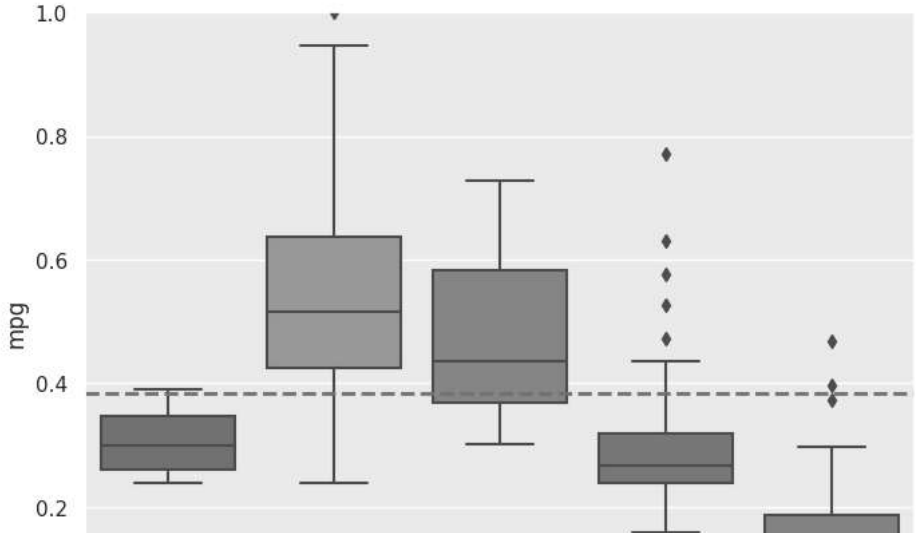
```
<matplotlib.lines.Line2D at 0x7be9ea68b6d0>
```




## And MPG distribution for cylinders

```
var = 'cylinders'
data_plt = pd.concat([data_scale['mpg'], data_scale[var]], axis=1)
f, ax = plt.subplots(figsize=(8, 6))
fig = sns.boxplot(x=var, y="mpg", data=data_plt)
fig.axis(ymin=0, ymax=1)
plt.axhline(data_scale.mpg.mean(),color='r',linestyle='dashed',linewidth=2)
```

```
<matplotlib.lines.Line2D at 0x7be9f26ca590>
```



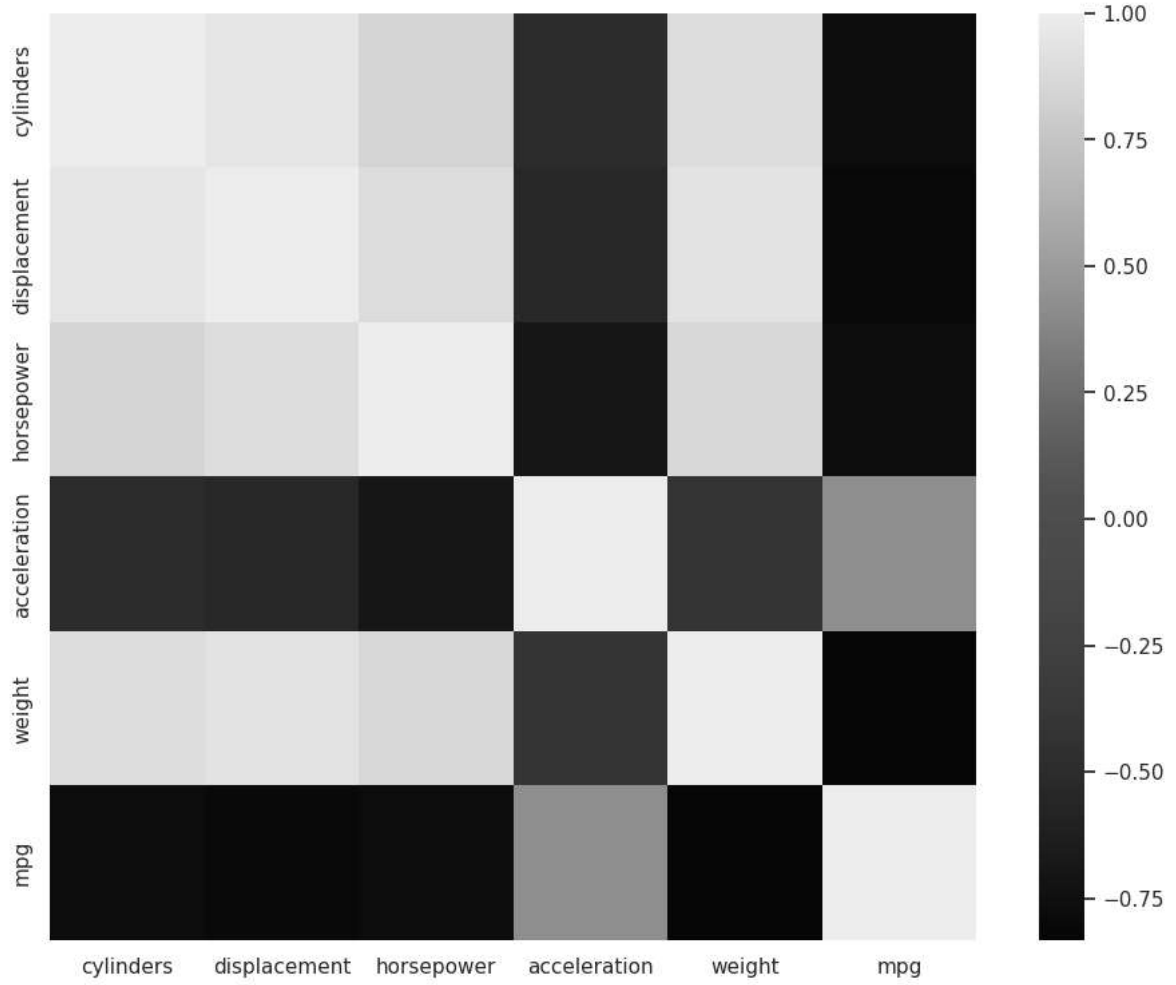

## Now that we have looked at the distribution of



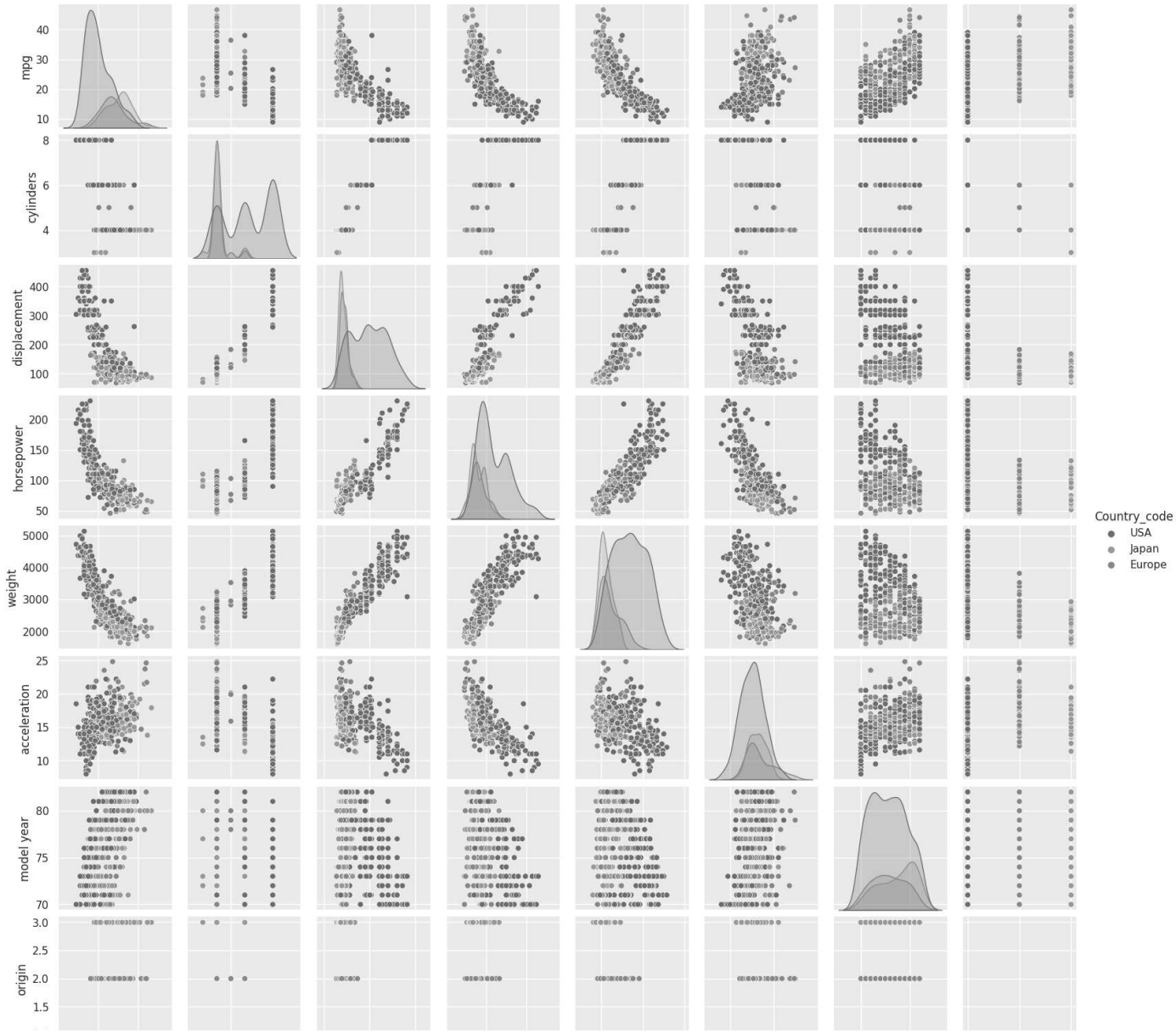
```
numeric_data = data.select_dtypes(include='number')  # Select numeric columns only
corrmat = numeric_data.corr()
#corrmat = data.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, square=True);


# Now, you can calculate the correlation matrix for numeric columns.
```


```
factors = ['cylinders','displacement','horsepower','acceleration','weight','mpg']
corrmat = data[factors].corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, square=True);
```




```
#scatterplot
sns.set()
sns.pairplot(data, size = 2.0,hue ='Country_code')
plt.show()
```




## So far we have seen the data

So far, we have seen the data to get a feel for it, we saw the spread of the desired variable MPG along the various discrete variables, namely, Origin, Year of Manufacturing or Model and Cylinders.
Now lets extract an additional discrete variable company name and add it to this data. We will use regular expressions and str.extract() function of pandas data-frame to make this new column

```
data.index
```

```
Index(['chevrolet chevelle malibu', 'buick skylark 320', 'plymouth satellite',
       'amc rebel sst', 'ford torino', 'ford galaxie 500', 'chevrolet impala',
       'plymouth fury iii', 'pontiac catalina', 'amc ambassador dpl',
       ...
       'chrysler lebaron medallion', 'ford granada l', 'toyota celica gt',
       'dodge charger 2.2', 'chevrolet camaro', 'ford mustang gl', 'vw pickup',
       'dodge rampage', 'ford ranger', 'chevy s-10'],
      dtype='object', name='car name', length=392)
```

## As we can see the index of the data frame contains model name along with the company name. Now lets use regular expressions to quickly extract the company names. As we can see the index is in format 'COMPANY_NAME - SPACE -MODEL - SPACE -VARIANT' and so regular expressions will make it an easy task.

```
data[data.index.str.contains('subaru')].index.str.replace('(.*)', 'subaru dl')
```

```
Index(['subaru', 'subaru dl', 'subaru dl', 'subaru'], dtype='object', name='car name')
```

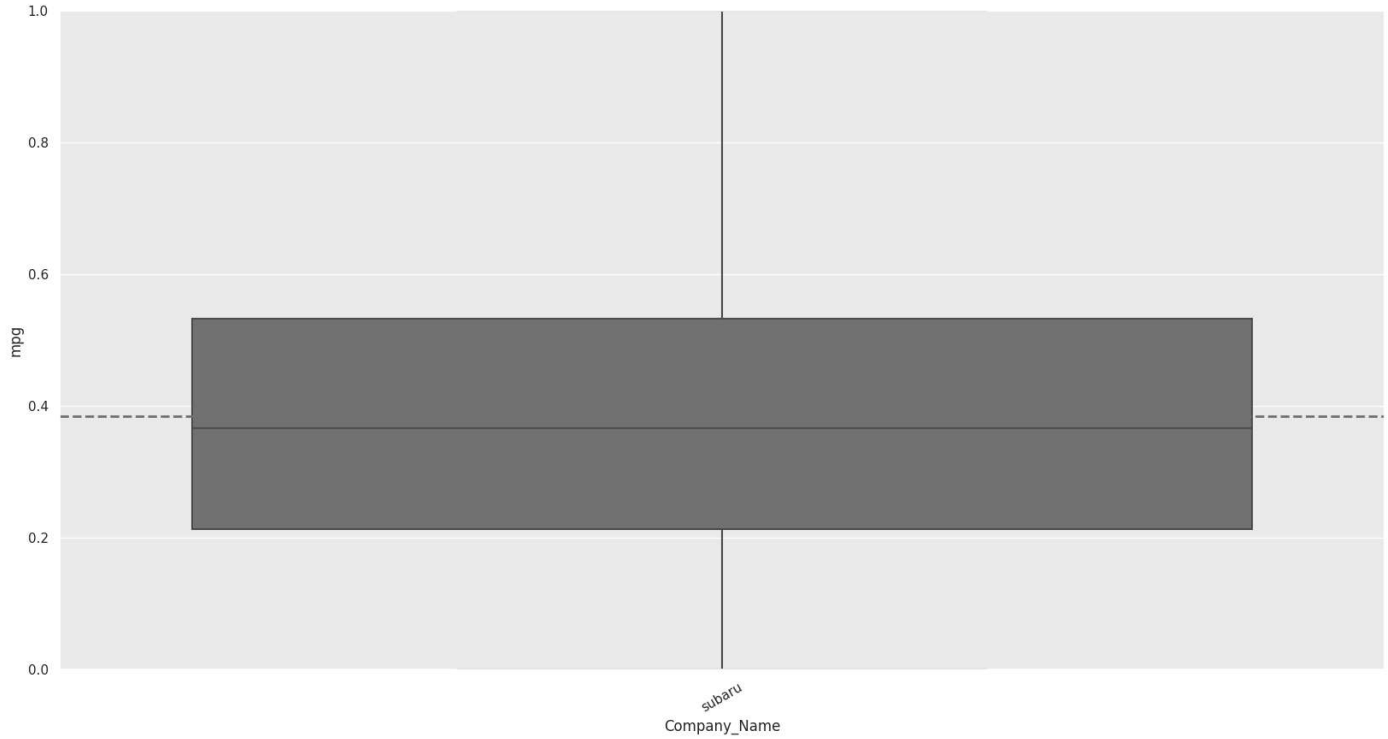
```
data['Company_Name'] = data.index.str.extract('(^.*?)\s')
```

That does it, almost, we can see NaN so some text was not extracted, this may be due to difference in formatting. We ca also see that some companies are named differently and also some spelling mistakes, lets correct these.

```
data['Company_Name'] = data['Company_Name'].replace(['volkswagen','vokswagen','vw'],'VW')
data['Company_Name'] = data['Company_Name'].replace('maxda','mazda')
data['Company_Name'] = data['Company_Name'].replace('toyouta','toyota')
data['Company_Name'] = data['Company_Name'].replace('mercedes','mercedes-benz')
data['Company_Name'] = data['Company_Name'].replace('nissan','datsun')
data['Company_Name'] = data['Company_Name'].replace('capri','ford')
data['Company_Name'] = data['Company_Name'].replace(['chevroelt','chevy'],'chevrolet')
data['Company_Name'].fillna(value = 'subaru',inplace=True)  ## Strin methords will not work on null values so we use fillna()
```

```
var = 'Company_Name'
data_plt = pd.concat([data_scale['mpg'], data[var]], axis=1)
f, ax = plt.subplots(figsize=(20,10))
fig = sns.boxplot(x=var, y="mpg", data=data_plt)
fig.set_xticklabels(ax.get_xticklabels(),rotation=30)
fig.axis(ymin=0, ymax=1)
plt.axhline(data_scale.mpg.mean(),color='r',linestyle='dashed',linewidth=2)
```

```
<matplotlib.lines.Line2D at 0x7be9e2693790>
```



```
data.Company_Name.isnull().any()
```

```
False
```

## Lets look at some extremes

```
var='mpg'
data[data[var]== data[var].min()]
```

| car name | mpg | cylinders | displacement | horsepower | weight | acceleration | model year | origin | Country_code | Company_Name |
|---|---|---|---|---|---|---|---|---|---|---|
| **hi 1200d** | 9.0 | 8 | 304.0 | 193.0 | 4732 | 18.5 | 70 | 1 | USA | subaru |

```
data[data[var]== data[var].max()]
```

| car name | mpg | cylinders | displacement | horsepower | weight | acceleration | model year | origin | Country_code | Company_Name |
|---|---|---|---|---|---|---|---|---|---|---|
| **mazda glc** | 46.6 | 4 | 86.0 | 65.0 | 2110 | 17.9 | 80 | 3 | Japan | subaru |

```
var='displacement'
data[data[var]== data[var].min()]
```

| car name | mpg | cylinders | displacement | horsepower | weight | acceleration | model year | origin | Country_code | Company_Name |
|---|---|---|---|---|---|---|---|---|---|---|
| **fiat 128** | 29.0 | 4 | 68.0 | 49.0 | 1867 | 19.5 | 73 | 2 | Europe | subaru |

```
data[data[var]== data[var].max()]
```

| car name | mpg | cylinders | displacement | horsepower | weight | acceleration | model year | origin | Country_code | Company_Name |
|---|---|---|---|---|---|---|---|---|---|---|
| **pontiac catalina** | 14.0 | 8 | 455.0 | 225.0 | 4425 | 10.0 | 70 | 1 | USA | subaru |
| **buick estate wagon (sw)** | 14.0 | 8 | 455.0 | 225.0 | 3086 | 10.0 | 70 | 1 | USA | subaru |
| **buick electra 225 custom** | 12.0 | 8 | 455.0 | 225.0 | 4951 | 11.0 | 73 | 1 | USA | subaru |

```
var='horsepower'
data[data[var]== data[var].min()]
```

| car name | mpg | cylinders | displacement | horsepower | weight | acceleration | model year | origin | Country_code | Company_Name |
|---|---|---|---|---|---|---|---|---|---|---|
| **volkswagen 1131 deluxe sedan** | 26.0 | 4 | 97.0 | 46.0 | 1835 | 20.5 | 70 | 2 | Europe | subaru |

```
data[data[var]== data[var].max()]
```

| car name | mpg | cylinders | displacement | horsepower | weight | acceleration | model year | origin | Country_code | Company_Name |
|---|---|---|---|---|---|---|---|---|---|---|
| **pontiac grand prix** | 16.0 | 8 | 400.0 | 230.0 | 4278 | 9.5 | 73 | 1 | USA | subaru |

```
var='weight'
data[data[var]== data[var].min()]
```

```
data[data[var]== data[var].max()]
```

| car name | mpg | cylinders | displacement | horsepower | weight | acceleration | model year | origin | Country_code | Company_Name |
|---|---|---|---|---|---|---|---|---|---|---|
| **pontiac safari (sw)** | 13.0 | 8 | 400.0 | 175.0 | 5140 | 12.0 | 71 | 1 | USA | subaru |

```
var='acceleration'
data[data[var]== data[var].min()]
```

| car name | mpg | cylinders | displacement | horsepower | weight | acceleration | model year | origin | Country_code | Company_Name |
|---|---|---|---|---|---|---|---|---|---|---|
| **plymouth 'cuda 340** | 14.0 | 8 | 340.0 | 160.0 | 3609 | 8.0 | 70 | 1 | USA | subaru |

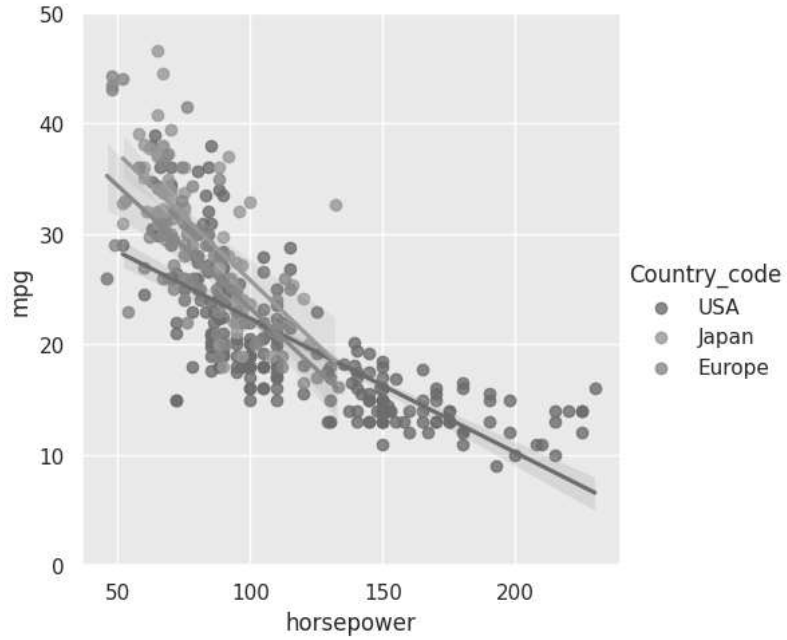
```
data[data[var]== data[var].max()]
```

| car name | mpg | cylinders | displacement | horsepower | weight | acceleration | model year | origin | Country_code | Company_Name |
|---|---|---|---|---|---|---|---|---|---|---|
| **peugeot 504** | 27.2 | 4 | 141.0 | 71.0 | 3190 | 24.8 | 79 | 2 | Europe | subaru |

Now that we have looked at the distribution of the data along discrete variables and we saw some scatter-plots using the seaborn pairplot. Now let's try to find some logical causation for variations in mpg. We will use the lmplot() function of seaborn with scatter set as true. This will help us in understanding the trends in these relations. We can later verify what we see with ate correlation heat map to find if the conclusions drawn are correct. We prefer lmplot() over regplot() for its ability to plot categorical data better. We will split the regressions for different origin countries.

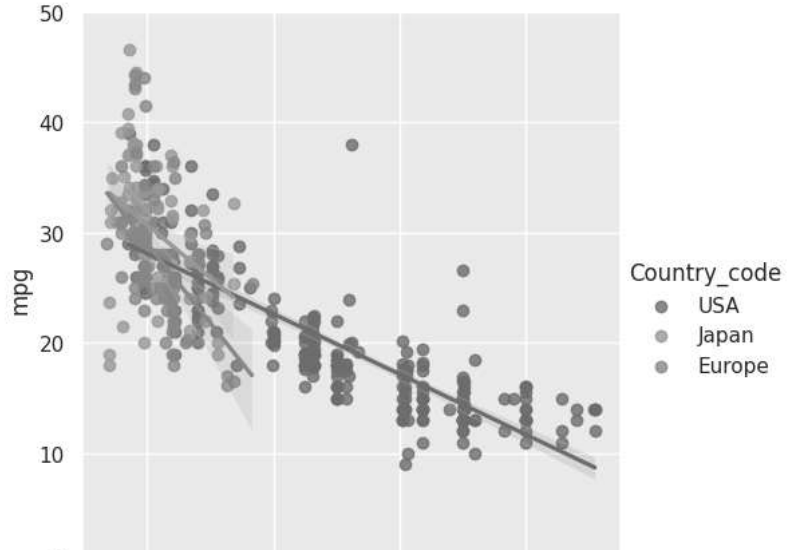
```
var = 'horsepower'
plot = sns.lmplot(x=var, y='mpg', data=data, hue='Country_code')
plot.set(ylim=(0, 50))
```

```
<seaborn.axisgrid.FacetGrid at 0x7be9e27ab490>
```



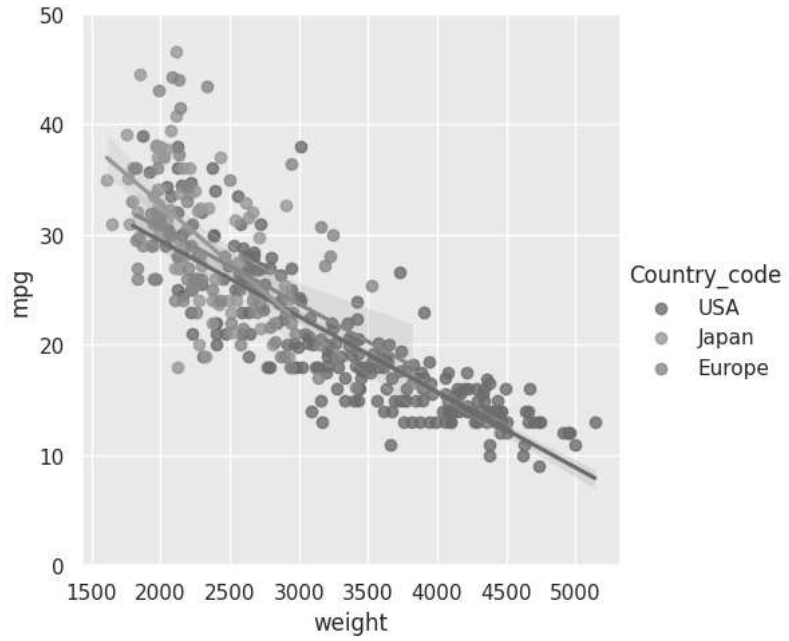

```
var = 'displacement'
plot = sns.lmplot(x=var, y='mpg', data=data, hue='Country_code')
plot.set(ylim = (0,50))
```

```
<seaborn.axisgrid.FacetGrid at 0x7be9e262baf0>
```



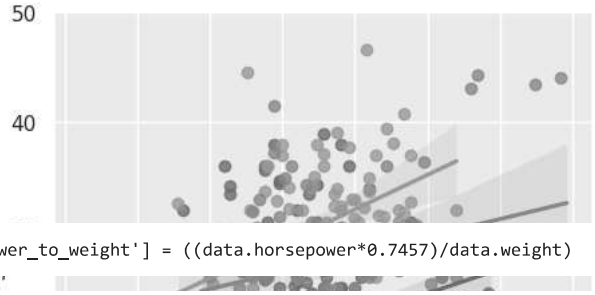

```
var = 'weight'
plot = sns.lmplot(x=var, y='mpg', data=data, hue='Country_code')
plot.set(ylim = (0,50))
```

```
<seaborn.axisgrid.FacetGrid at 0x7be9e251ae90>
```




```
var = 'acceleration'
plot = sns.lmplot(x=var, y='mpg', data=data, hue='Country_code')
plot.set(ylim = (0,50))
```

```
<seaborn.axisgrid.FacetGrid at 0x7be9ec2d8a00>
```




```
data['Power_to_weight'] = ((data.horsepower*0.7457)/data.weight)
```

```
data.sort_values(by='Power_to_weight',ascending=False ).head()
```

| car name | mpg | cylinders | displacement | horsepower | weight | acceleration | model year | origin | Country_code | Company_Name | Power_to_weight |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **buick estate wagon (sw)** | 14.0 | 8 | 455.0 | 225.0 | 3086 | 10.0 | 70 | 1 | USA | subaru | 0.054369 |
| **pontiac grand prix** | 16.0 | 8 | 400.0 | 230.0 | 4278 | 9.5 | 73 | 1 | USA | subaru | 0.040091 |
| **pontiac catalina** | 14.0 | 8 | 455.0 | 225.0 | 4425 | 10.0 | 70 | 1 | USA | subaru | 0.037917 |

## Our journey so far:

So far, we have a looked at our data using various pandas methods and visualized it using seaborn package. We looked at

### MPGs relation with discrete variables

- MPG distribution over given years if manufacturing
  - MPG distribution by country of origin
  - MPG distribution by number of cylinders

### MPGs relation to other continuous variables:

- Pair wise scatter plot of all variables in data.

### Correlation

- We looked at the correlation heat map of all columns in our data

## Lets look at some regression models:

Now that we know what our data looks like, lets use some machine learning models to predict the value of MPG given the values of the factors. We will use pythons scikit learn to train test and tune various regression models on our data and compare the results. We shall use the following regression models:-

- Linear Regression
- GBM Regression

```
data.head()
```

| car name | mpg | cylinders | displacement | horsepower | weight | acceleration | model year | origin | Country_code | Company_Name | Power_to_weight |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **chevrolet chevelle malibu** | 18.0 | 8 | 307.0 | 130.0 | 3504 | 12.0 | 70 | 1 | USA | subaru | 0.027666 |
| **buick skylark 320** | 15.0 | 8 | 350.0 | 165.0 | 3693 | 11.5 | 70 | 1 | USA | subaru | 0.033317 |
| **plymouth satellite** | 18.0 | 8 | 318.0 | 150.0 | 3436 | 11.0 | 70 | 1 | USA | subaru | 0.032554 |

```
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold



factors = ['cylinders','displacement','horsepower','acceleration','weight','origin','model year']
X = pd.DataFrame(data[factors].copy())
y = data['mpg'].copy()


X = StandardScaler().fit_transform(X)


X_train,X_test,y_train,y_test=train_test_split(X,y,test_size = 0.33,random_state=324)
X_train.shape[0] == y_train.shape[0]
```

```
True
```

```
regressor = LinearRegression()


regressor.get_params()
```

```
{'copy_X': True, 'fit_intercept': True, 'n_jobs': None, 'positive': False}
```

```
regressor.fit(X_train,y_train)
```

```
LinearRegression
LinearRegression()
```

```
y_predicted = regressor.predict(X_test)


rmse = sqrt(mean_squared_error(y_true=y_test,y_pred=y_predicted))
rmse
```

```
3.4867296149015616
```

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

```
GradientBoostingRegressor
GradientBoostingRegressor(n_estimators=4000)
```

```
gb_regressor.get_params()
```

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

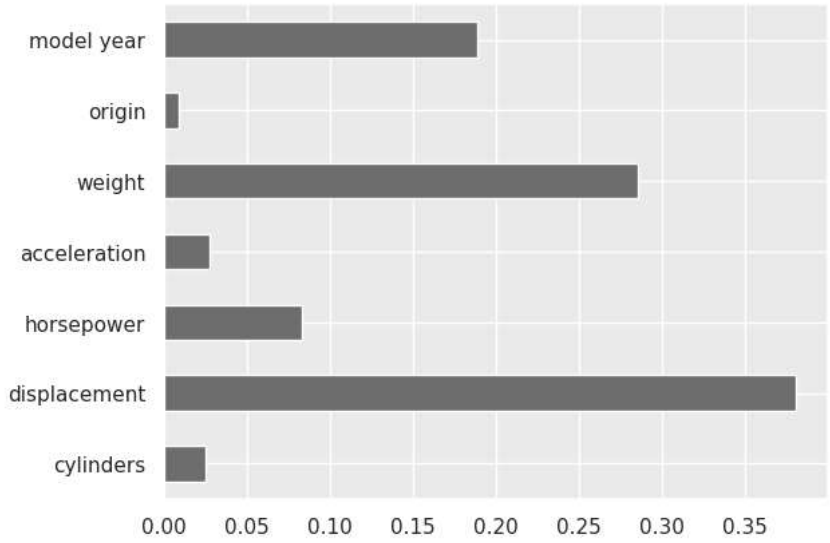
```
y_predicted_gbr = gb_regressor.predict(X_test)


rmse_bgr = sqrt(mean_squared_error(y_true=y_test,y_pred=y_predicted_gbr))
rmse_bgr
```

```
2.66187522614335
```

```
fi= pd.Series(gb_regressor.feature_importances_,index=factors)
fi.plot.barh()
```

```
<Axes: >
```




Good, so our initial models work well, but these metrics were performed on test set and cannot be used for tuning the model, as that will cause bleeding of test data into training data, hence, we will use K-Fold to create Cross Validation sets and use grid search to tune the model.

```
from sklearn.decomposition import PCA


pca = PCA(n_components=2)


pca.fit(data[factors])
```

```
PCA
PCA(n_components=2)
```

```
pca.explained_variance_ratio_
```

```
array([0.99756151, 0.0020628 ])
```

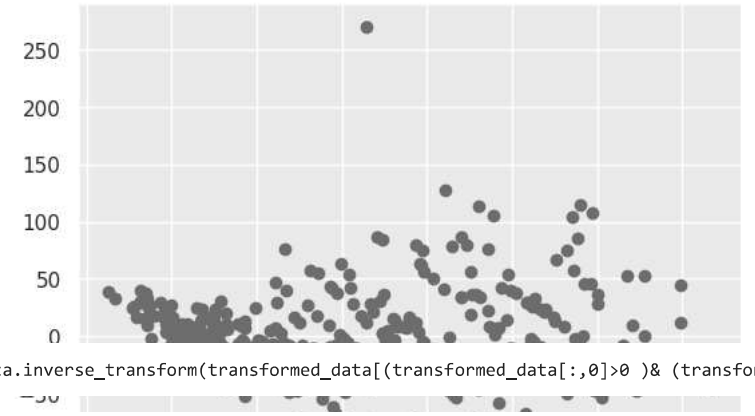
```
pca1 = pca.components_[0]
pca2 = pca.components_[1]


transformed_data = pca.transform(data[factors])


  pc1 = transformed_data[:,0]
  pc2 = transformed_data[:,1]


plt.scatter(pc1,pc2)
```

```
<matplotlib.collections.PathCollection at 0x7be9e2a83820>
```




```
c = pca.inverse_transform(transformed_data[(transformed_data[:,0]>0 )& (transformed_data[:,1]>250)])
```

```
factors
```

```
['cylinders',
 'displacement',
 'horsepower',
 'acceleration',
 'weight',
 'origin',
 'model year']
```

```
c
```

```
array([[9.32016159e+00, 4.65727261e+02, 1.90441442e+02, 5.95699243e+00,
        3.08611199e+03, 6.23550659e-01, 6.93571097e+01]])
```

```
data[(data['model year'] == 70 )&( data.displacement>400)]
```

| car name | mpg | cylinders | displacement | horsepower | weight | acceleration | model year | origin | Country_code | Company_Name | Power_to_weight |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ford galaxie 500 | 15.0 | 8 | 429.0 | 198.0 | 4341 | 10.0 | 70 | 1 | USA | subaru | 0.034013 |
| chevrolet impala | 14.0 | 8 | 454.0 | 220.0 | 4354 | 9.0 | 70 | 1 | USA | subaru | 0.037679 |
| plymouth fury iii | 14.0 | 8 | 440.0 | 215.0 | 4312 | 8.5 | 70 | 1 | USA | subaru | 0.037181 |

***The exceptionally far away point seems to be the Buick estate wagon. This seems logical as the weight data given in the data set seems to be incorrect. The weight for the vehicle is given to be 3086 lbs, however, on research it can be found that the car weight is 4727-4775 lbs. These values are based on internet search***

- **Now we use K-fold to create a new K-fold object called 'cv_sets' that contains index values for training and cross validation and use these sets in GridSearchCV to tune our model so that it does not over fit or under fit the data**
- **We will also define a dictionary called 'params' with the hyper-parameters we want to tune**
- **Lastly we define 'grid' which is a GridSearchCV object which we will provide the parameters to tune and the K folds of data created by using the Kfold in sklearn.model_selection**

```
cv_sets = KFold(n_splits=10, shuffle= True,random_state=100)
params = {'n_estimators' : list(range(40,61)),
          'max_depth' : list(range(1,10)),
          'learning_rate' : [0.1,0.2,0.3] }
grid = GridSearchCV(gb_regressor, params,cv=cv_sets,n_jobs=4)
```

```
grid = grid.fit(X_train, y_train)
```

```
grid.best_estimator_
```

```
gb_regressor_t = grid.best_estimator_
```

```
gb_regressor_t.fit(X_train,y_train)
```

```
GradientBoostingRegressor(learning_rate=0.3, max_depth=2, n_estimators=45)
```

```
y_predicted_gbr_t = gb_regressor_t.predict(X_test)
```

```
rmse = sqrt(mean_squared_error(y_true=y_test,y_pred=y_predicted_gbr_t))
rmse
```

```
2.6894745792140493
```

```
data.duplicated().any()
```

```
False
```