Data Analysis to predict miles per gallon on dataset Auto MPGdataset

Joshitha Mandali
Towson University
jmanda2@students.towson.edu

I. Introduction

In this project, I am trying to analyze and visualize the Auto MPG Dataset in order to predict the most probable mpg dataset. The dataset that has been chosen to perform Exploratory Data Analysis and Data preprocessing for the Assignment-1 is "Auto MPG Dataset". This dataset was downloaded from "UCI Machine Learning Repository" which is center for Machine Learning and Intelligent Systems.

II. Dataset Description

The data in the dataset concerns about city-cycle fuel consumption in miles per gallon, to be predicted in terms of 3 multivalued discrete and 5 continuous attributes and this dataset was originally captured from the StatLib library which was maintained at Carnegie Mellon University.

This dataset was extensively used in the 1983 American Statistical Association Exposition. And it is a slightly modified version of the dataset provided in the StatLib library. For instance, in agreement with use by Ross Quinlan (1993) while predicting the attribute "mpg", 8 of the original instances were removed because they had unknown values for the "mpg" attribute [1]

The attributes are split into the following:

- 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)

III. Exploratory Data Analysis

A) Description of the data:

We need to know the different kinds of data and other statistics of the data loaded before proceeding with further steps. So, excluding NaN values, data.describe() generated results in descriptive statistics that summarizes dispersion, central tendency, shape of dataset distribution.

Loading the dataset:

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib
        import seaborn as sns
        import missingno as msno
        matplotlib.style.use('ggplot')

In [2]: from scipy import stats

In [3]: import matplotlib.pyplot as plt
        import pylab
        import scipy.stats as stats

In [4]: data = pd.read_csv("/Users/joshitha/Downloads/mpgdata.txt")

In [5]: data.head()
Out[5]:
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	modelyear	origin	carname
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	3436.0	10.5	70.0	1.0	ford torino

```
data.describe()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	modelyear	origin
count	122.000000	129.000000	129.00000	128.000000	129.000000	129.000000	129.000000	129.000000
mean	18.532787	6.201550	246.29845	127.500000	3291.232558	14.313953	71.519380	1.387597
std	5.642496	1.864051	123.08279	47.325144	969.184366	3.093005	1.173275	0.676735
min	9.000000	3.000000	68.00000	46.000000	1613.000000	8.000000	70.000000	1.000000
25%	14.000000	4.000000	116.00000	89.500000	2372.000000	12.000000	70.000000	1.000000
50%	18.000000	6.000000	250.00000	112.500000	3288.000000	14.000000	72.000000	1.000000
75%	23.000000	8.000000	350.00000	165.000000	4154.000000	16.000000	73.000000	2.000000
max	35.000000	8.000000	455.00000	230.000000	5140.000000	23.500000	73.000000	3.000000

The data describe generated result index of count, mean, std, min, max, 25%, 50%, 75%.

```
In [7]: data.dtypes

Out[7]: mpg             float64
        cylinders       float64
        displacement    float64
        horsepower      float64
        weight          float64
        acceleration    float64
        modelyear       float64
        origin          float64
        carname          object
        dtype: object
```

Few observations can be made by looking at the data. Like there are missing values across the data frame, which need to be handled. Column “carname” contains many duplicates and needs to be parsed for extracting it in a numerical quantity, it looks like a string (object) right now. Firstly, we drop null or missing values.

Finding out the number of rows and columns or by shape. We can see that there are 129 records and 9 columns

```
In [6]: print(f'Data contain {data.shape[0]} records and {data.shape[1]} columns.')

        Data contain 129 records and 9 columns.
```

B) Handling Missing data:

On dropping the Null or missing values, the data dropped down from 129 records to 121 records

```
data = data.dropna()
```

```
data.shape
```

```
(121, 9)
```

Missing values:

```
In [8]: for col in data.columns:
            pct_missing = data[col].isnull().mean()
            print(f'{col} - {pct_missing :.1%}')

mpg - 5.4%
cylinders - 0.0%
displacement - 0.0%
horsepower - 0.8%
weight - 0.0%
acceleration - 0.0%
modelyear - 0.0%
origin - 0.0%
carname - 0.0%
```

Looking at the percentage of missing values per column

```
In [3]: missing_data = pd.DataFrame({'total_missing': data.isnull().sum(), 'perc_missing': (data.isnull().sum()/82790)*100})
        missing_data
```

Out[3]:

	total_missing	perc_missing
mpg	7	0.008455
cylinders	0	0.000000
displacement	0	0.000000
horsepower	1	0.001208
weight	0	0.000000
acceleration	0	0.000000
modelyear	0	0.000000
origin	0	0.000000
carname	0	0.000000

Column’s “mpg” and “horsepower” have null or missing values as shown above so filling those missing values with some values

```
data['mpg'] = data['mpg'].fillna(10.0)
```

```
data.shape
```

```
(121, 9)
```

```
data['horsepower'] = data['horsepower'].fillna(90.0)
```

```
data.shape
```

```
(121, 9)
```

```
missing_data = pd.DataFrame({'total_missing': data.isnull().
missing_data
```

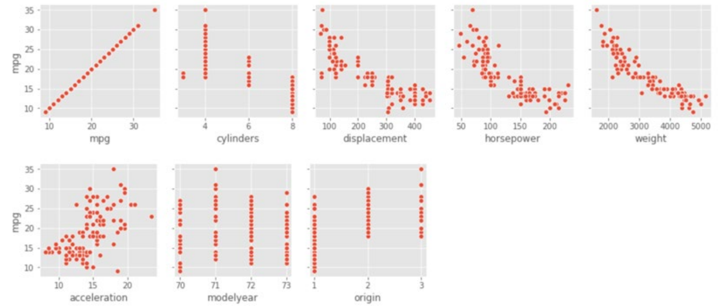
	total_missing	perc_missing
mpg	0	0.0
cylinders	0	0.0
displacement	0	0.0
horsepower	0	0.0
weight	0	0.0
acceleration	0	0.0
modelyear	0	0.0
origin	0	0.0
carname	0	0.0

C) Handling Outliers:

Pairplot of numerical variables:

```
In [16]: df_num = data.select_dtypes(include = ['float64'])
         df_num.head()
```

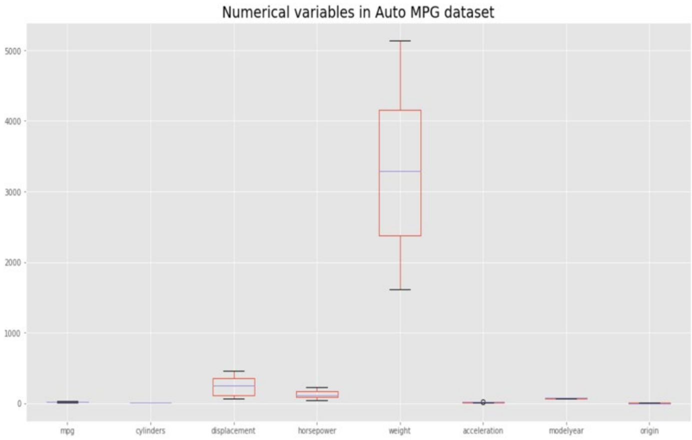
```
for i in range(0, len(df_num.columns), 5):
    sns.pairplot(data=df_num,
                 x_vars=df_num.columns[i:i+5],
                 y_vars=['mpg'])
```



Boxplot of numerical variables:

To graphically depict numerical variables in Auto MPG dataset

```
num_cols = ['mpg','cylinders','displacement','horsepower','weight','acceleration','modelyear','origin']
plt.figure(figsize=(18,9))
data[num_cols].boxplot()
plt.title("Numerical variables in Auto MPG dataset", fontsize=20)
plt.show()
```



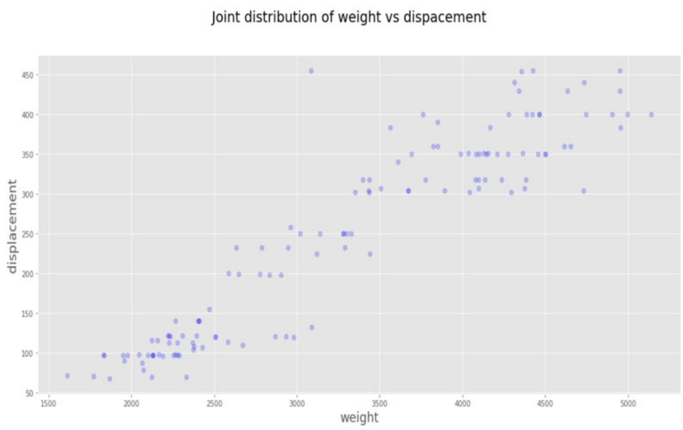
From the boxplot, we can generate top 10 weights of the automobile (we can see that boxplot shows values as outliers, as per the IQR- Inter-Quartile Range).

```
data.sort_values(by=['weight'], ascending=False).head(10)
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	modelyear	origin	carname
51	13.0	8.0	400.0	175.0	5140.0	12.0	71.0	1.0	pontiac safari (sw)
110	11.0	8.0	400.0	150.0	4997.0	14.0	73.0	1.0	"chevrolet impala"
49	12.0	8.0	383.0	180.0	4955.0	11.5	71.0	1.0	dodge monaco (sw)
97	12.0	8.0	429.0	198.0	4952.0	11.5	73.0	1.0	"mercury marquis brougham"
102	12.0	8.0	455.0	225.0	4951.0	11.0	73.0	1.0	"buick electra 225 custom"
111	12.0	8.0	400.0	167.0	4906.0	12.5	73.0	1.0	"ford country"
50	13.0	8.0	400.0	170.0	4746.0	12.0	71.0	1.0	ford country squire (sw)
101	13.0	8.0	440.0	215.0	4735.0	11.0	73.0	1.0	"chrysler new yorker brougham"
34	9.0	8.0	304.0	193.0	4732.0	18.5	70.0	1.0	hi 1200d
112	13.0	8.0	360.0	170.0	4654.0	13.0	73.0	1.0	"plymouth custom suburb"

Joint distribution of weight vs displacement. Visualized the distribution of weight vs displacement to have an understanding about the values

```
plt.figure(figsize=(18,8))
plt.xlabel("weight", fontsize=18)
plt.ylabel("displacement", fontsize=18)
plt.suptitle("Joint distribution of weight vs dispacement", fontsize= 20)
plt.plot(data.weight, data['displacement'], 'bo', alpha=0.2)
plt.show()
```



IV. Data Preprocessing

A) Normalization:

Z-score Normalization:

```
In [42]: zscore_weight = (data.weight-data.weight.mean())/data.weight.std()

In [43]: zscore_weight
Out[43]: 0      0.220545
         1      0.413981
         2      0.150949
         3      0.147879
         4      0.150949
                  ...
         124   -1.454876
         125   -1.157046
         126   -0.723094
         127   -0.430382
         128    0.113081
         Name: weight, Length: 121, dtype: float64
```

Min-Max Normalization:

```
In [44]: mmnorm_weight = (data.weight-data.weight.min())/(data.weight.max()-data.weight.min())

In [45]: mmnorm_weight
Out[45]: 0      0.536150
         1      0.589736
         2      0.516870
         3      0.516019
         4      0.516870
                  ...
         124    0.072016
         125    0.154522
         126    0.274738
         127    0.355826
         128    0.506379
         Name: weight, Length: 121, dtype: float64
```

Scaling variable:

```
df = data
df["mpg"] = df["mpg"]/df["mpg"].max()
df["cylinders"] = df["cylinders"]/df["cylinders"].max()
df['weight'] = df['weight']/df['weight'].max()
df[["mpg","cylinders","weight"]].head()
```

	mpg	cylinders	weight
0	0.514286	1.0	0.681712
1	0.428571	1.0	0.718482
2	0.514286	1.0	0.668482
3	0.457143	1.0	0.667899
4	0.485714	1.0	0.668482

We get an output of normalized “mpg”, “cylinders” “weight” in the range of [0,1]

Feature Scaling:

```
In [73]:
         Y = data.iloc[:,:-1].values
         X = data.iloc[:,1].values

In [74]: # Print the X and Y
         print ('X: %s'%(str(X)))
         print ('-----------------------------------')
         print ('Y: %s'%(str(Y)))
```

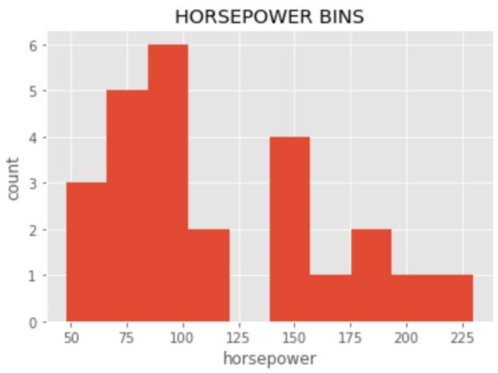
```
X: [0 0 2 2 2 2 2 0 2 2 0 1 0 0 0 2 3 1 2 0 2 2 2 2 0]
-----------------------------------
Y: [[7.69230769e-01 0.00000000e+00 3.07000000e+02 2.00000000e+02
  4.37600000e+03 1.50000000e+01 7.00000000e+01]
 [0.00000000e+00 0.00000000e+00 3.04000000e+02 1.93000000e+02
  4.73200000e+03 1.85000000e+01 7.00000000e+01]
 [           nan 2.00000000e+00 9.70000000e+01 4.80000000e+01
  1.97800000e+03 2.00000000e+01 7.10000000e+01]
 [1.61538462e+01 2.00000000e+00 8.80000000e+01 7.60000000e+01
  2.06500000e+03 1.45000000e+01 7.10000000e+01]
 [1.69230769e+01 2.00000000e+00 7.10000000e+01 6.50000000e+01
  1.77300000e+03 1.90000000e+01 7.10000000e+01]
 [2.00000000e+01 2.00000000e+00 7.20000000e+01 6.90000000e+01
  1.61300000e+03 1.80000000e+01 7.10000000e+01]
 [1.23076923e+01 2.00000000e+00 9.75000000e+01 8.00000000e+01
  2.12600000e+03 1.70000000e+01 7.20000000e+01]
 [6.15384615e+00 0.00000000e+00 3.04000000e+02 1.50000000e+02
  3.67200000e+03 1.15000000e+01 7.20000000e+01]
 [1.46153846e+01 2.00000000e+00 9.80000000e+01 8.00000000e+01
  2.16400000e+03 1.50000000e+01 7.20000000e+01]
 [1.38461538e+01 2.00000000e+00 9.70000000e+01 8.80000000e+01
  2.10000000e+03 1.65000000e+01 7.20000000e+01]
 [3.84615385e+00 0.00000000e+00 3.18000000e+02 1.50000000e+02
  4.23700000e+03 1.45000000e+01 7.30000000e+01]
 [1.07692308e+01 1.00000000e+00 1.98000000e+02 9.50000000e+01
  2.90400000e+03 1.60000000e+01 7.30000000e+01]
 [1.53846154e+00 0.00000000e+00 4.00000000e+02 1.50000000e+02
  4.99700000e+03 1.40000000e+01 7.30000000e+01]
 [3.07692308e+00 0.00000000e+00 3.60000000e+02 1.70000000e+02
  4.65400000e+03 1.30000000e+01 7.30000000e+01]
 [2.30769231e+00 0.00000000e+00 3.50000000e+02 1.80000000e+02
  4.49900000e+03 1.25000000e+01 7.30000000e+01]
```

B) Bining:

Plot the histogram of the attribute "horsepower" in order to see what the distribution of "horsepower" looks like.

```
In [109]: df = data

In [110]: df["horsepower"] = df["horsepower"].astype(int,copy=True)

In [111]: %matplotlib inline
          import matplotlib.pyplot as plt
          plt.hist(df["horsepower"])

          plt.xlabel("horsepower")
          plt.ylabel("count")
          plt.title("HORSEPOWER BINS")
Out[111]: Text(0.5, 1.0, 'HORSEPOWER BINS')
```

```
Out[111]: Text(0.5, 1.0, 'HORSEPOWER BINS')
```



Setting the group names and applying cut function to determine what each value of "horsepower" belongs to

```
In [112]: bins = np.linspace(min(df["horsepower"]),max(df["horsepower"]),4)
          bins
Out[112]: array([ 48.        , 108.66666667, 169.33333333, 230.        ])

In [113]: group_names = ['Low', 'Medium', 'High']

In [114]: df['horsepower-binned'] = pd.cut(df['horsepower'], bins, labels=group_names ,include_lowest = True)
          df[['horsepower','horsepower-binned']].head(20)
```

Out[114]:

	horsepower	horsepower-binned
32	200	High
34	193	High
39	48	Low
59	76	Low
60	65	Low
61	69	Low
65	80	Low
73	150	Medium
90	80	Low
91	88	Low
100	150	Medium
108	95	Low
110	150	Medium
112	170	High
113	180	High

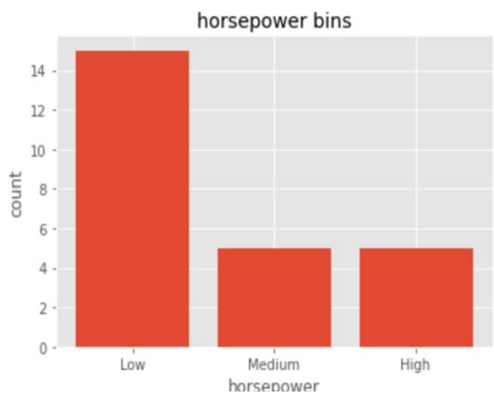
We can now see the number of vehicles in each bin

```
In [115]: df["horsepower-binned"].value_counts()
Out[115]: Low       15
          High       5
          Medium     5
          Name: horsepower-binned, dtype: int64
```

Plotting the distribution of each bin.

```
In [116]: %matplotlib inline
          import matplotlib as plt
          from matplotlib import pyplot
          pyplot.bar(group_names, df["horsepower-binned"].value_counts())

          # set x/y labels and plot title
          plt.pyplot.xlabel("horsepower")
          plt.pyplot.ylabel("count")
          plt.pyplot.title("horsepower bins")
```

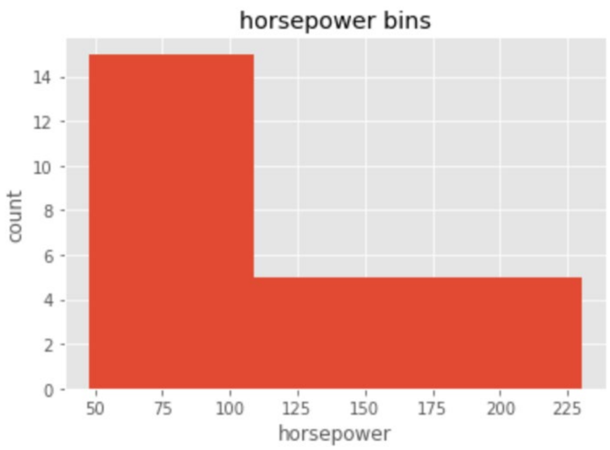
```
Out[116]: Text(0.5, 1.0, 'horsepower bins')
```



Plotting a histogram to visualize the distribution of bins

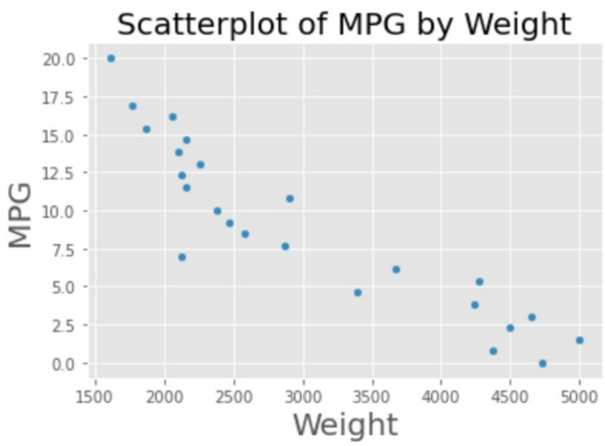
```
In [117]: %matplotlib inline
import matplotlib as plt
from matplotlib import pyplot


# draw historgram of attribute "horsepower" with bins = 3
plt.pyplot.hist(df["horsepower"], bins = 3)

# set x/y labels and plot title
plt.pyplot.xlabel("horsepower")
plt.pyplot.ylabel("count")
plt.pyplot.title("horsepower bins")
```

```
Text(0.5, 1.0, 'horsepower bins')
```




```
ax = data.plot.scatter('weight','mpg')
ax.set_xlabel("Weight",fontsize=20)
ax.set_ylabel("MPG",fontsize=20)
ax.set_title("Scatterplot of MPG by Weight",fontsize=20)
```

```
Text(0.5, 1.0, 'Scatterplot of MPG by Weight')
```



C) Natural Log , Inverse Square rootTransformation

```
In [89]: log_weight = np.log(data.weight)
```

```
In [90]: invsqrt_weight = 1/np.sqrt(data.weight)
```

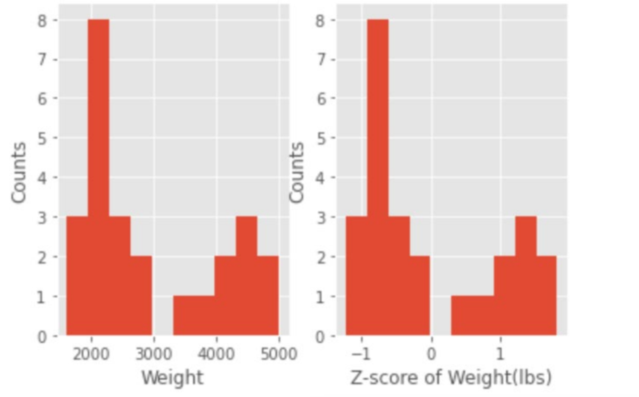
Calculating Skewness:

```
weight_skew =(3*(np.mean(log_weight)-np.median(log_weight)))/np.std(log_weight)
zscore_weight_skew = (3*(np.mean(zscore_weight)-np.median(zscore_weight)))/np.std(zscore_weight)
```

Plotting Side-by-side Histograms of Weight and Z-Score of Weight

```
fig, axarr = plt.subplots(1,2)
data.weight.hist(ax=axarr[0])
zscore_weight.hist(ax=axarr[1])
axarr[0].set_xlabel("Weight")
axarr[0].set_ylabel("Counts")
axarr[1].set_xlabel("Z-score of Weight(lbs)")
axarr[1].set_ylabel("Counts")
```

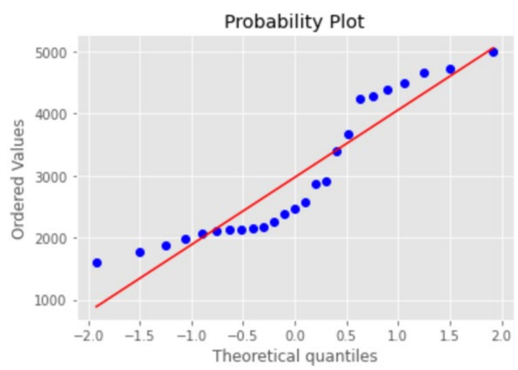
```
Text(0, 0.5, 'Counts')
```



Normal Probability Plot

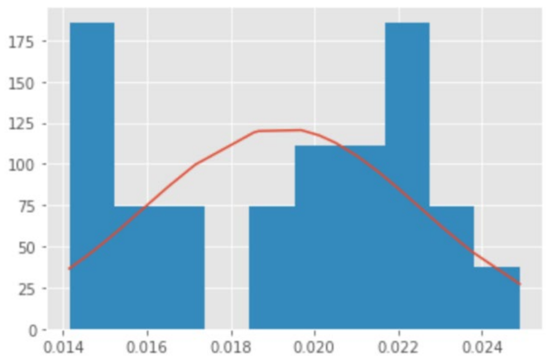
```
plt.ioff()
stats.probplot(data.weight, dist="norm", plot=pylab)
```

```
((array([-1.9213301 , -1.50368257, -1.24941883, -1.0573306 , -0.897955  ,
        -0.75865603, -0.63273432, -0.51614096, -0.40618759, -0.30094875,
        -0.19894932, -0.09898372,  0.        ,  0.09898372,  0.19894932,
         0.30094875,  0.40618759,  0.51614096,  0.63273432,  0.75865603,
         0.897955  ,  1.0573306 ,  1.24941883,  1.50368257,  1.9213301 ]),
  array([1613., 1773., 1867., 1978., 2065., 2100., 2124., 2126., 2158.,
        2164., 2265., 2379., 2472., 2582., 2868., 2904., 3399., 3672.,
        4237., 4278., 4376., 4499., 4654., 4732., 4997.])),
 (1084.5619157839405, 2971.28, 0.9374891135640945))
```



Plotting Histogram with Normal Distribution

```
In [95]: invsqrt_weight_sorted = sorted(invsqrt_weight)
fit = stats.norm.pdf(invsqrt_weight_sorted, np.mean(invsqrt_weight_sorted), np.std(invsqrt_w
pylab.plot(invsqrt_weight_sorted, fit, '-')
pylab.hist(invsqrt_weight,density=True)

Out[95]: (array([186.00027869,  74.40011148,  74.40011148,   0.        ,
          74.40011148, 111.60016721, 111.60016721, 186.00027869,
          74.40011148,  37.20005574]),
 array([0.01414638, 0.01522165, 0.01629691, 0.01737218, 0.01844745,
        0.01952272, 0.02059798, 0.02167325, 0.02274852, 0.02382379,
        0.02489905]),
 <a list of 10 Patch objects>)
```



V. Regression Analysis

A) Simple Linear Regression:

The Data Model that I have used is Simple Linear Regression. Let's understand the relationship between two variables that is a predictor/independent variable(X) and the response/dependent variable(Y). Dependent variable (Y) is the variable that I want to predict

Creating Linear regression object:

```
: from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm

: LinearRegression()
```

Creating a linear function with attribute "mpg" as predictor variable and weight as response variable to check whether mpg would help us predict weight of the car

```
: X = df[['mpg']]
Y = df['weight']
lm.fit(X,Y)

: LinearRegression()
```

Prediction Output: Y= a+bX

```
In [21]: Yhat=lm.predict(X)
Yhat[0:5]

Out[21]: array([0.65455281, 0.74695765, 0.65455281, 0.71615604, 0.68535443])
```

Generating the values of intercept (a) and Slope (b)

```
In [22]: print(lm.intercept_)
print(lm.coef_)

1.2089818556896694
[-1.07805647]
```

From the output we can plug in the actual values, we get as following:

Weight = 1.20 – 1.07 * mpg

Now, creating a linear function with attribute "mpg" as predictor variable and horsepower as response variable to check whether mpg would help us predict weight of the car

```
In [23]: lm1 = LinearRegression()
X = df[['mpg']]
Y = df['horsepower']
lm1.fit(X,Y)

Out[23]: LinearRegression()
```

```
In [24]: Yhat=lm1.predict(X)
Yhat[0:5]

Out[24]: array([130.19038075, 151.08879822, 130.19038075, 144.12265907,
        137.15651991])
```

```
In [25]: print(lm1.intercept_)
print(lm1.coef_)

255.5808855745997
[-243.81487049]
```

From the output we can plug in the actual values, we get as following:

Horsepower = 255.58– 243.81 * mpg

B) Multiple Linear Regression:

Developing a model using "horsepower", "weight", "origin" and "cylinders" as predictor variables

```
In [26]: Z = df[['horsepower', 'weight', 'origin', 'cylinders']]
#Fit the linear model
lm.fit(Z, df['mpg'])

Out[26]: LinearRegression()
```

```
In [27]: print(lm.intercept_)
print(lm.coef_)

0.9229549434670259
[-3.37946779e-04 -5.91197132e-01  3.38343295e-02 -2.73317759e-02]
```

From the output we can plug in the actual values, we get as following

Mpg = 0.9229549434670259 -3.37946779e-04 * horsepower -5.91197132e-01 * weight + 3.38343295e-02 * origin -2.73317759e-02 * cylinders

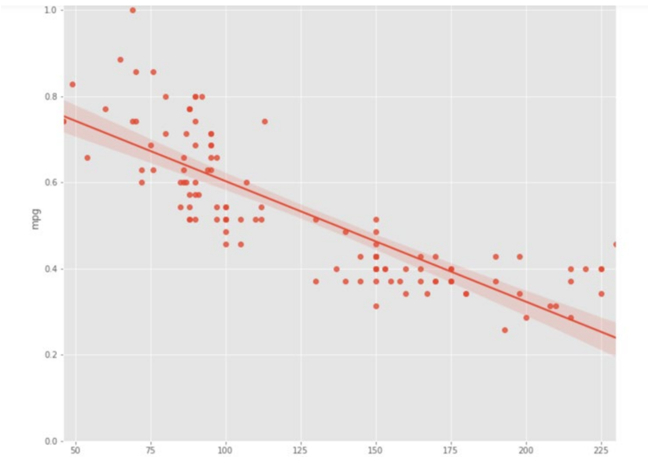
C) Model Evaluation

Visualizing the fit of the model using regression plots

```
In [28]: import seaborn as sns
         %matplotlib inline
```

```
In [29]: width=12
         height=10
         plt.figure(figsize = (width,height))
         sns.regplot(x='horsepower',y='mpg',data=df)
         plt.ylim(0,)
```

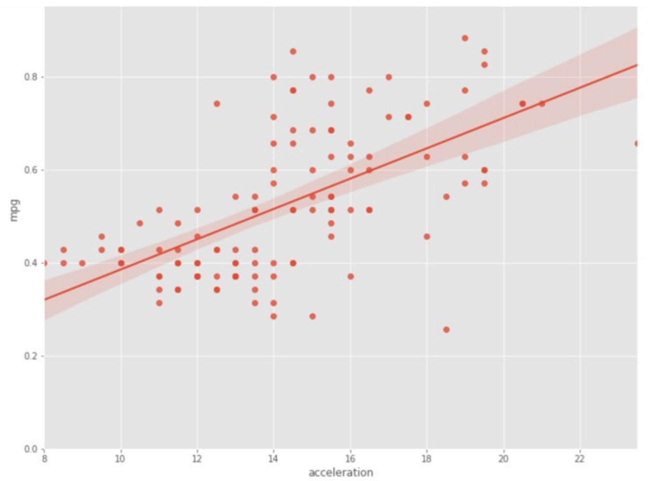
We can see from the plot below that “mpg” is **negatively corelated** to “horsepower”, as the regression slope is negative.



Comparing this plot to the regression plot of “acceleration”

```
In [39]: plt.figure(figsize=(width, height))
         sns.regplot(x="acceleration", y="mpg", data=df)
         plt.ylim(0,)

Out[39]: (0.0, 1.0371428571428571)
```

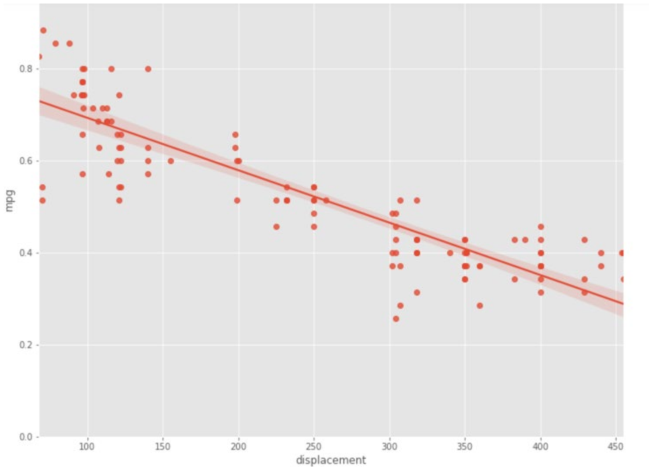
We can see from the plot below that “mpg” is **positively corelated** to “acceleration”, as the regression slope is positive



Comparing this plot to the regression plot of “displacement”

```
: plt.figure(figsize=(width, height))
  sns.regplot(x="displacement", y="mpg", data=df)
  plt.ylim(0,)

: (0.0, 1.0371428571428571)
```

We can see from the plot below that “mpg” is **negatively corelated** to “displacement”, as the regression slope is negative



From the analysis, “acceleration” is strongly corelated with “mpg”. It is approximately **0.60** whereas horsepower and displacement are **-0.82** and **-0.91** respectively

```
df[['horsepower','displacement','acceleration','mpg']].corr()
```

	horsepower	displacement	acceleration	mpg
horsepower	1.000000	0.913575	-0.786826	-0.825720
displacement	0.913575	1.000000	-0.761331	-0.874039
acceleration	-0.786826	-0.761331	1.000000	0.606579
mpg	-0.825720	-0.874039	0.606579	1.000000

VI. Conclusion

In this paper, I have chosen Auto MPG Dataset from UCI Machine Learning Repository and have performed exploratory data analysis, data preprocessing on the dataset and finally based on the exploratory data analysis of the dataset, came up with a prediction question and created a regression model to predict a dependent variable based on a set of dependent variables.

Section V shows the regression analysis where correlation between the attributes has been analyzed. From results it is depicted that “displacement” is strongly corelated with “mpg”

References

[1] http://archive.ics.uci.edu/ml/datasets.php
[2] https://www.kaggle.com/
[3] https://www.edureka.co/blog/exploratory-data-analysis-in-python/