Exploratory Data Analysis

Problem Statement:

We have used Cars dataset from kaggle with features including make, model, year, engine, and other properties of the car used to predict its price.

Importing the necessary libraries

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(color_codes=True)
from scipy import stats
import warnings
warnings.filterwarnings("ignore")
```

Load the dataset into dataframe

In [2]:

```
## load the csv file
data = pd.read_csv('D:\Twilearnass1/Cars_data.csv')
```

In [3]:

```
## print the head of the dataframe

data.head()
```

Out[3]:

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0

Now we observe the each features present in the dataset.

`Make:` The Make feature is the company name of the Car.
`Model:` The Model feature is the model or different version of Car models.
`Year:` The year describes the model has been launched.
`Engine Fuel Type:` It defines the Fuel type of the car model.
`Engine HP:` It's say the Horsepower that refers to the power an engine produces.
`Engine Cylinders:` It define the nos of cylinders in present in the engine.
`Transmission Type:` It is the type of feature that describe about the car transmission type i.e Mannual or automatic.
`Driven_Wheels:` The type of wheel drive.
`No of doors:` It defined nos of doors present in the car.
`Market Category:` This features tells about the type of car or which category the car belongs.
`Vehicle Size:` It's say about the about car size.
`Vehicle Style:` The feature is all about the style that belongs to car.
`highway MPG:` The average a car will get while driving on an open stretch of road without stopping or starting, typically at a higher speed.
`city mpg:` City MPG refers to driving with occasional stopping and braking.
`Popularity:` It can refered to rating of that car or popularity of car.
`MSRP:` The price of that car.

Check the datatypes

In [4]:

```
# Get the datatypes of each columns number of records in each column.
data.info()
#df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11914 entries, 0 to 11913
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Make               11914 non-null  object 
 1   Model              11914 non-null  object 
 2   Year               11914 non-null  int64  
 3   Engine Fuel Type   11911 non-null  object 
 4   Engine HP          11845 non-null  float64
 5   Engine Cylinders   11884 non-null  float64
 6   Transmission Type  11914 non-null  object 
 7   Driven_Wheels      11914 non-null  object 
 8   Number of Doors    11908 non-null  float64
 9   Market Category    8172 non-null   object 
 10  Vehicle Size       11914 non-null  object 
 11  Vehicle Style      11914 non-null  object 
 12  highway MPG        11914 non-null  int64  
 13  city mpg           11914 non-null  int64  
 14  Popularity         11914 non-null  int64  
 15  MSRP               11914 non-null  int64  
dtypes: float64(3), int64(5), object(8)
memory usage: 1.5+ MB
```

Dropping irrevalent columns

If we consider all columns present in the dataset then unneccessary columns will impact on the model's accuracy.
Not all the columns are important to us in the given dataframe, and hence we would drop the columns that are irrevalent to us. It would reflect our model's accucary so we need to drop them. Otherwise it will affect our model.

The list cols_to_drop contains the names of the cols that are irrevalent, drop all these cols from the dataframe.

```
cols_to_drop = ["Engine Fuel Type", "Market Category", "Vehicle Style", "Popularity", "Number of Doors", "Vehicle Size"]
```

These features are not neccessary to obtain the model's accucary. It does not contain any relevant information in the dataset.

In [5]:

```
# initialise cols_to_drop
cols_to_drop = ['Engine Fuel Type','Market Category','Vehicle Style','Popularity','Numbe
```

In [6]:

```
# drop the irrevalent cols and print the head of the datafram
new = data.drop(cols_to_drop,axis=1)

# print df head
new.head()
```

Out[6]:

	Make	Model	Year	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	highway MPG	city mpg	MSF
0	BMW	1 Series M	2011	335.0	6.0	MANUAL	rear wheel drive	26	19	461
1	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	19	406
2	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	20	363
3	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	294
4	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	345

Renaming the columns

Now, Its time for renaming the feature to useful feature name. It will help to use them in model training purpose.

We have already dropped the unneccesary columns, and now we are left with useful columns. One extra thing that we would do is to rename the columns such that the name clearly represents the essence of the column.

The given dict represents (in key value pair) the previous name, and the new name for the dataframe columns

In [7]:

```
# rename cols
rename_cols = {'Make':'NAME','Model':'MODEL','Year':'YEAR','Engine HP':'ENGINE','Engine
               'highway MPG' : 'H_MPG','city mpg':'C_MPG','MSRP':'MSRP'}
```

In [8]:

```
# use a pandas function to rename the current columns
new.rename(columns = rename_cols, inplace=True)
```

In [9]:

```
# Print the head of the dataframe
new.head()
```

Out[9]:

	NAME	MODEL	YEAR	ENGINE	CYLINDER	TYPE	WHEEL	H_MPG	C_MPG	MSRP
0	BMW	1 Series M	2011	335.0	6.0	MANUAL	rear wheel drive	26	19	46135
1	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	19	40650
2	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	20	36350
3	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	29450
4	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	34500

Dropping the duplicate rows

There are many rows in the dataframe which are duplicate, and hence they are just repeating the information. Its better if we remove these rows as they don't add any value to the dataframe.

For given data, we would like to see how many rows were duplicates. For this, we will count the number of rows, remove the dublicated rows, and again count the number of rows.

In [10]:

```
# number of rows before removing duplicated rows
#len(new)
new.shape
```

Out[10]:

```
(11914, 10)
```

In [12]:

```
# drop the duplicated rows
new_n = new.drop_duplicates()

# print head of df
new_n.head()
```

Out[12]:

	NAME	MODEL	YEAR	ENGINE	CYLINDER	TYPE	WHEEL	H_MPG	C_MPG	MSRP
0	BMW	1 Series M	2011	335.0	6.0	MANUAL	rear wheel drive	26	19	46135
1	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	19	40650
2	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	20	36350
3	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	29450
4	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	34500

In [13]:

```
# Count Number of rows after deleting duplicated rows
new_n.shape
```

Out[13]:

```
(10925, 10)
```

Dropping the null or missing values

Missing values are usually represented in the form of Nan or null or None in the dataset.

Finding whether we have null values in the data is by using the isnull() function.

There are many values which are missing, in pandas dataframe these values are reffered to as np.nan. We want to deal with these values beause we can't use nan values to train models. Either we can remove them to apply some strategy to replace them with other values.

To keep things simple we will be dropping nan values

In [14]:

```
# check for nan values in each columns
new_n.isnull().sum()
```

Out[14]:

```
NAME         0
MODEL        0
YEAR         0
ENGINE      69
CYLINDER    30
TYPE         0
WHEEL        0
H_MPG        0
C_MPG        0
MSRP         0
dtype: int64
```

As we can see that the HP and Cylinders have null values of 69 and 30. As these null values will impact on models' accuracy. So to avoid the impact we will drop the these values. As these values are small camparing with dataset that will not impact any major affect on model accuracy so we will drop the values.

In [15]:

```
# drop missing values
new_n = new_new.dropna()
```

In [16]:

```
# Make sure that missing values are removed
# check number of nan values in each col again
new_n.isnull().sum()
```

Out[16]:

```
NAME        0
MODEL       0
YEAR        0
ENGINE      0
CYLINDER    0
TYPE        0
WHEEL       0
H_MPG       0
C_MPG       0
MSRP        0
dtype: int64
```

In [17]:

```
#Describe statistics of df
new_n.describe()
```

Out[17]:

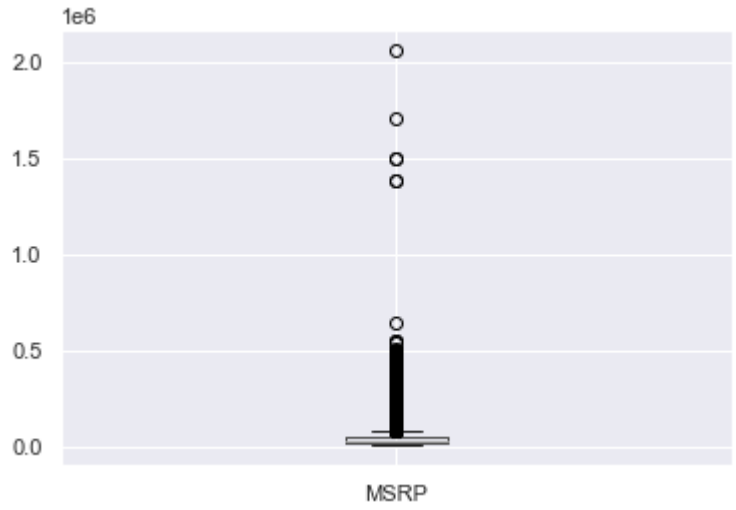
	YEAR	ENGINE	CYLINDER	H_MPG	C_MPG	MSRP
count	10827.000000	10827.000000	10827.000000	10827.000000	10827.000000	1.082700e+04
mean	2010.896370	254.553062	5.691604	26.308119	19.327607	4.249325e+04
std	7.029534	109.841537	1.768551	7.504652	6.643567	6.229451e+04
min	1990.000000	55.000000	0.000000	12.000000	7.000000	2.000000e+03
25%	2007.000000	173.000000	4.000000	22.000000	16.000000	2.197250e+04
50%	2015.000000	240.000000	6.000000	25.000000	18.000000	3.084500e+04
75%	2016.000000	303.000000	6.000000	30.000000	22.000000	4.330000e+04
max	2017.000000	1001.000000	16.000000	354.000000	137.000000	2.065902e+06

Removing outliers

Sometimes a dataset can contain extreme values that are outside the range of what is expected and unlike the other data. These are called outliers and often machine learning modeling and model skill in general can be improved by understanding and even removing these outlier values.

In [18]:

```
## Plot a boxplot for 'Price' column in dataset.

boxplot = new_n.boxplot(column=['MSRP'])
```

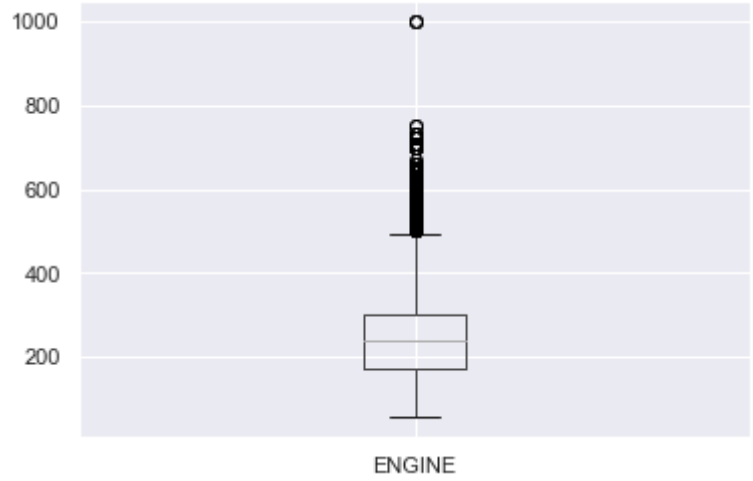


Observation:

Here as you see that we got some values near to 1.5 and 2.0 . So these values are called outliers. Because there are away from the normal values. Now we have detect the outliers of the feature of Price. Similarly we will checking of anothers features.

In [19]:

```
## PLot a boxplot for 'HP' columns in dataset

boxplot1 = new_n.boxplot(column=['ENGINE'])
```



Observation:

Here boxplots show the proper distribution of of 25 percentile and 75 percentile of the feature of HP.

In [ ]:

```

```

print all the columns which are of int or float datatype in df.

Hint: Use loc with condition

In [23]:

```
# print all the columns which are of int or float datatype in df.

r = new_n.select_dtypes(['int','float']).columns
print('\n columns:\n',r)
```

```
 columns:
 Index(['YEAR', 'ENGINE', 'CYLINDER', 'H_MPG', 'C_MPG', 'MSRP'], dtype='ob
ject')
```

Save the column names of the above output in variable list named 'l'

In [24]:

```
# save column names of the above output in variable list
l = list(r)
l
```

Out[24]:

```
['YEAR', 'ENGINE', 'CYLINDER', 'H_MPG', 'C_MPG', 'MSRP']
```

Outliers removal techniques - IQR Method

Here comes cool Fact for you!

IQR is the first quartile subtracted from the third quartile; these quartiles can be clearly seen on a box plot on the data.

- Calculate IQR and give a suitable threshold to remove the outliers and save this new dataframe into df2.

Let us help you to decide threshold: Outliers in this case are defined as the observations that are below (Q1 − 1.5x IQR) or above (Q3 + 1.5x IQR)

In [25]:

```
## define Q1 and Q2
Q1 = np.percentile(new_n['ENGINE'],25)
Q3 = np.percentile(new_n['ENGINE'],75)

# # define IQR (interquantile range)
IQR =  Q3 - Q1
IQR
```

Out[25]:

```
130.0
```

In [ ]:

```
# # define df2 after removing outliers
#df2 =
```

In [ ]:

```
# find the shape of df & df2
```

In [ ]:

```
# find unique values and there counts in each column in df using value counts function.

# for i in df.columns:
#     print ("--------------- %s ---------------" % i)
#     # code here
```

Visualising Univariate Distributions

We will use seaborn library to visualize eye catchy univariate plots.

Do you know? you have just now already explored one univariate plot. guess which one? Yeah its box plot.

Histogram & Density Plots

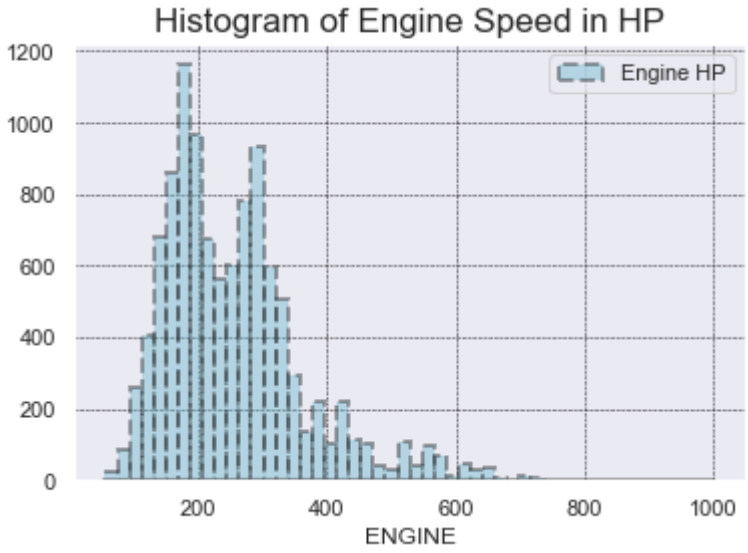
Histograms and density plots show the frequency of a numeric variable along the y-axis, and the value along the x-axis. The `sns.distplot()` function plots a density curve. Notice that this is aesthetically better than vanilla `matplotlib` .

In [ ]:

```
#ploting distplot for variable HP
```

In [26]:

```
sns.distplot(new_n['ENGINE'],  kde = False,
             hist_kws = {'color':'c','edgecolor':'k','linewidth':2,'linestyle':'--'} ,
             label = 'Engine HP')
                                                                                  # HISTOGRA
plt.title("Histogram of Engine Speed in HP", fontsize = 17)                          # TITLE
plt.grid(color = 'k', linestyle = '--', linewidth = 0.5)                          # BACKGROU
plt.legend()
plt.show()
```



Observation:

We plot the Histogram of feature HP with help of distplot in seaborn.
In this graph we can see that there is max values near at 200. similary we have also the 2nd highest value near 400 and so on.
It represents the overall distribution of continuous data variables.

Since seaborn uses matplotlib behind the scenes, the usual matplotlib functions work well with seaborn. For example, you can use subplots to plot multiple univariate distributions.

- Hint: use matplotlib subplot function

Bar Chart Plots

Plot a histogram depicting the make in X axis and number of cars in y axis.

Observation:

In this plot we can see that we have plot the bar plot with the cars model and nos. of cars.

In [27]:

```
new_n.columns
```

Out[27]:

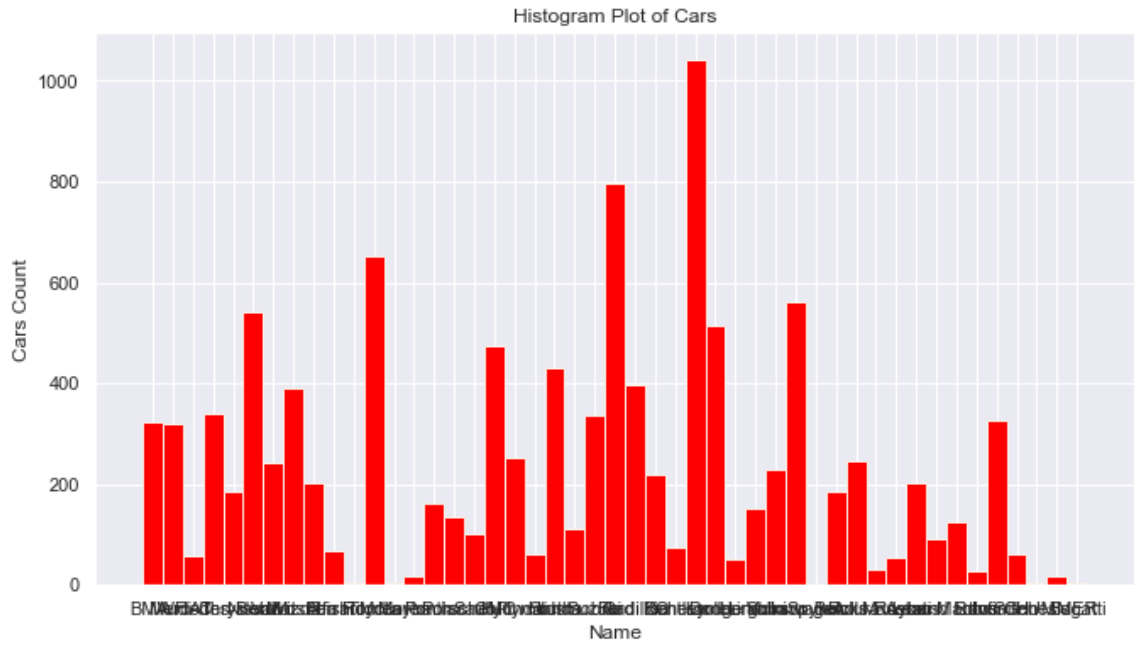
```
Index(['NAME', 'MODEL', 'YEAR', 'ENGINE', 'CYLINDER', 'TYPE', 'WHEEL', 'H_
MPG',
       'C_MPG', 'MSRP'],
      dtype='object')
```

In [30]:

```
plt.figure(figsize = (11,6))

sns.histplot(data = new_n , x = 'NAME', color='red', edgecolor = 'linen', alpha =1, bins

plt.title("Histogram Plot of Cars")
plt.xlabel('Name')
plt.ylabel('Cars Count')

# use nlargest and then .plot to get bar plot like below output
# Plot Title, X & Y label
```

Out[30]:

```
Text(0, 0.5, 'Cars Count')
```



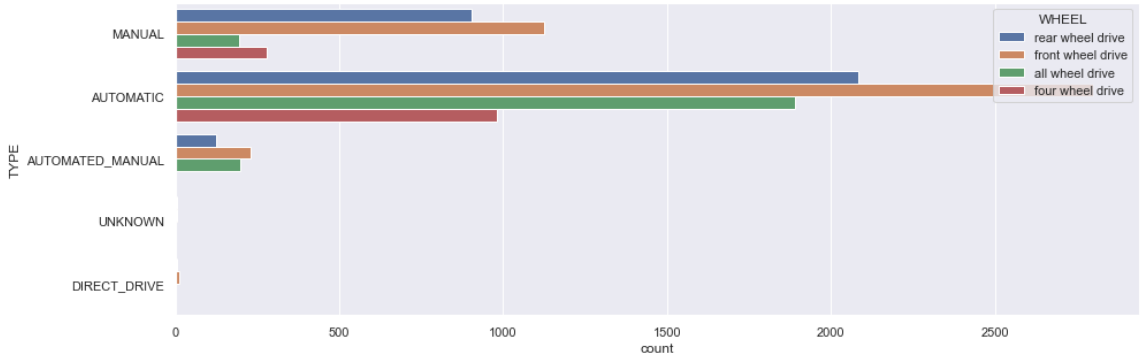
Count Plot

A count plot can be thought of as a histogram across a categorical, instead of quantitative, variable.

Plot a countplot for a variable Transmission vertically with hue as Drive mode

In [31]:

```
plt.figure(figsize=(15,5))

# plot countplot on transmission and drive mode

sns.countplot(y='TYPE',hue='WHEEL',data=new_n)
plt.show()
```



Observation:

In this count plot, We have plot the feature of Transmission with help of hue.
We can see that the the nos of count and the transmission type and automated manual is plotted. Drive mode as been given with help of hue.

Visualising Bivariate Distributions

Bivariate distributions are simply two univariate distributions plotted on x and y axes respectively. They help you observe the relationship between the two variables.

Scatter Plots

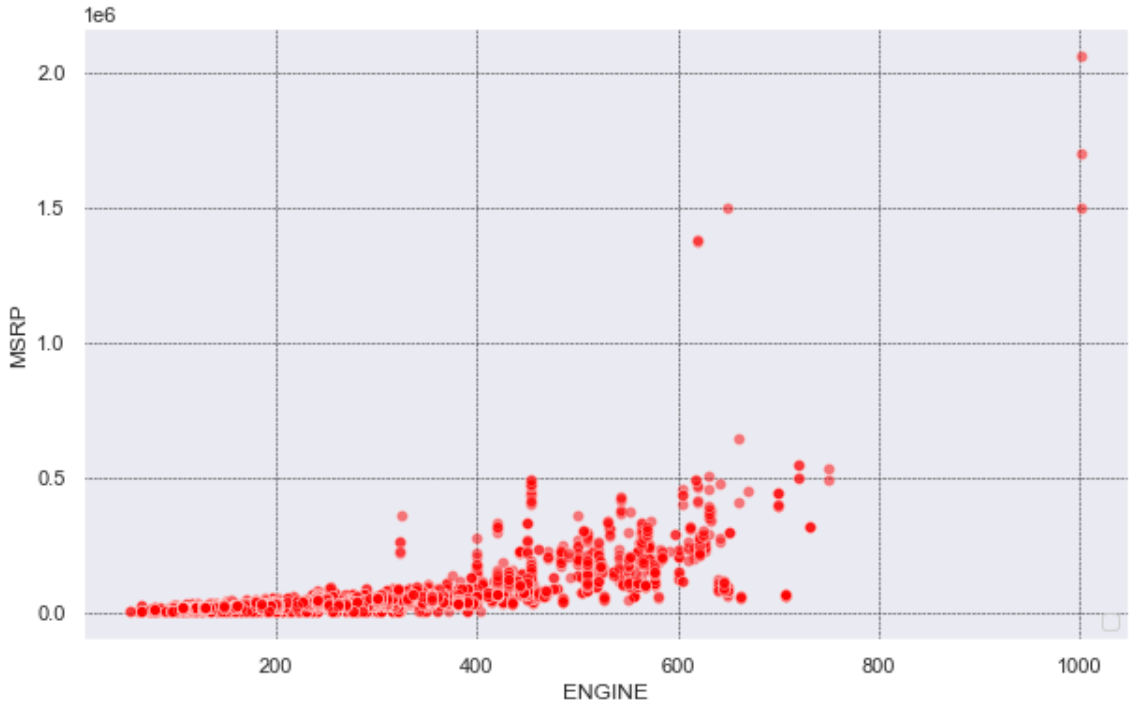
Scatterplots are used to find the correlation between two continuos variables.

Using scatterplot find the correlation between 'HP' and 'Price' column of the data.

In [35]:

```
## Your code here -
fig, ax = plt.subplots(figsize=(10,6))
sns.scatterplot(x = 'ENGINE',y = 'MSRP',data = new_new,sizes = (120,180),color='red',alp
plt.grid(color = 'k', linestyle = '--', linewidth = 0.5)
plt.legend(loc = 4)
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose la
bel start with an underscore are ignored when legend() is called with no a
rgument.
```



Observation:

It is a type of plot or mathematical diagram using Cartesian coordinates to display values for typically two variables for a set of data.
We have plot the scatter plot with x axis as HP and y axis as Price.
The data points between the features should be same either wise it give errors.

Plotting Aggregated Values across Categories

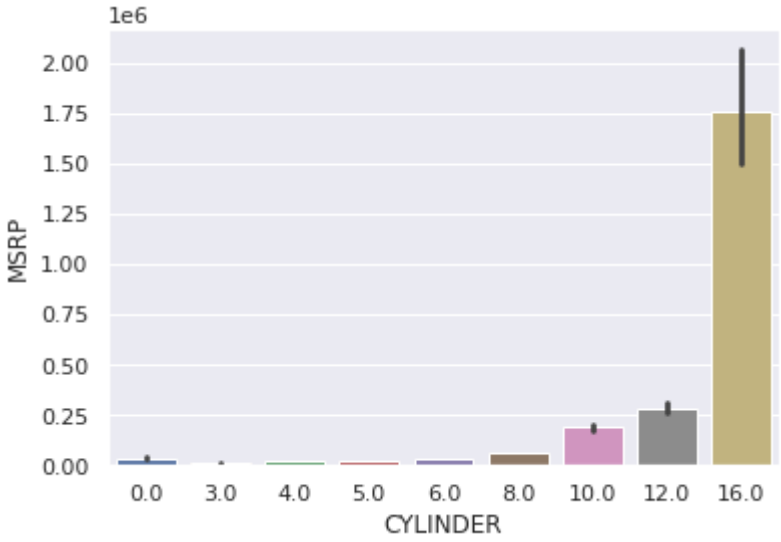
Bar Plots - Mean, Median and Count Plots

Bar plots are used to **display aggregated values** of a variable, rather than entire distributions. This is especially useful when you have a lot of data which is difficult to visualise in a single figure.

For example, say you want to visualise and *compare the Price across Cylinders*. The `sns.barplot()` function can be used to do that.

In [ ]:

```
# bar plot with default statistic=mean between Cylinder and Price

sns.barplot(x='CYLINDER',y='MSRP',data=new_new)
plt.show()
```



Observation:

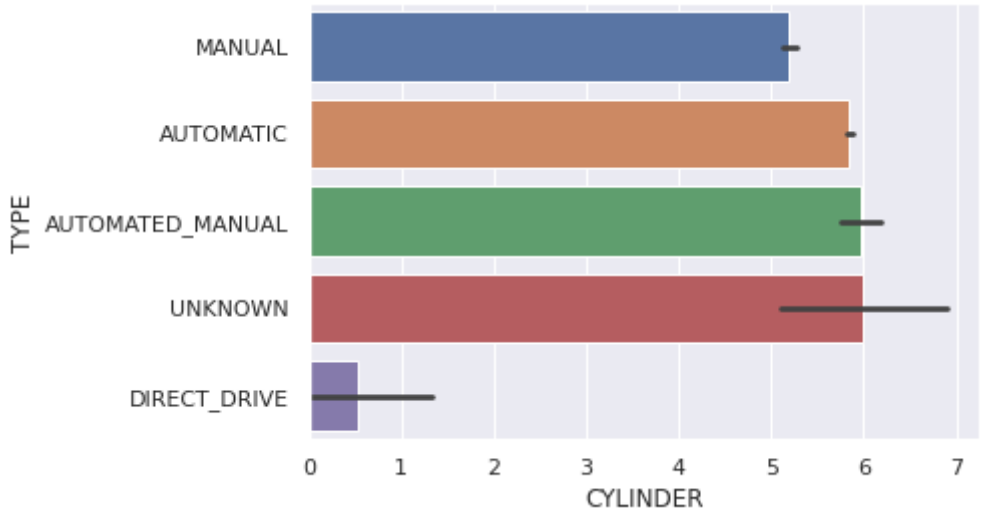
By default, seaborn plots the mean value across categories, though you can plot the count, median, sum etc.
Also, barplot computes and shows the confidence interval of the mean as well.

When you want to visualise having a large number of categories, it is helpful to plot the categories across the y-axis.

Let's now drill down into Transmission sub categories.

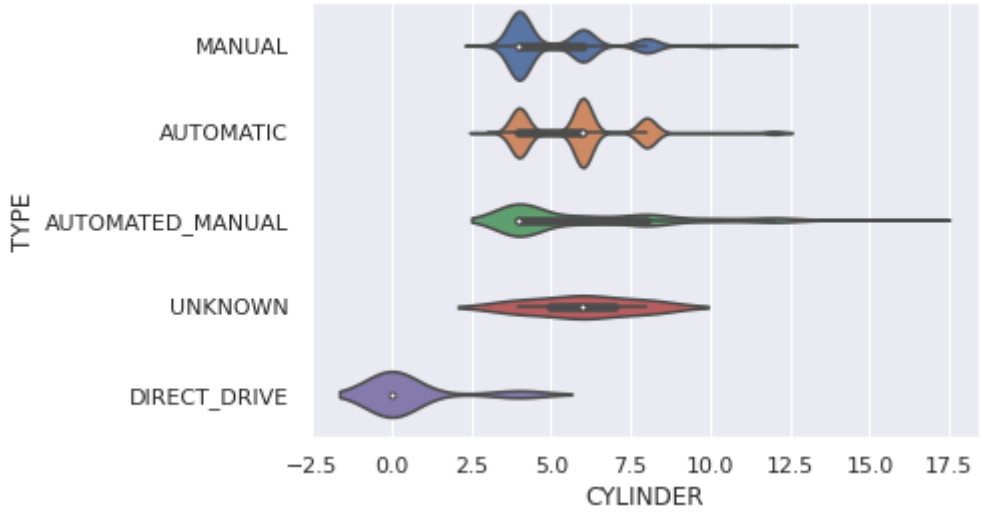
In [ ]:

```
# Plotting categorical variable Transmission across the y-axis

sns.barplot(x='CYLINDER',y='TYPE',data=new_new)
plt.show()
```



In [ ]:

```
sns.violinplot(x='CYLINDER',y='TYPE',data=new_new)
plt.show()
```



These plots looks beutiful isn't it? In Data Analyst life such charts are there unavoidable friend.:)

Multivariate Plots

Heatmaps

A heat map is a two-dimensional representation of information with the help of colors. Heat maps can help the user visualize simple or complex information

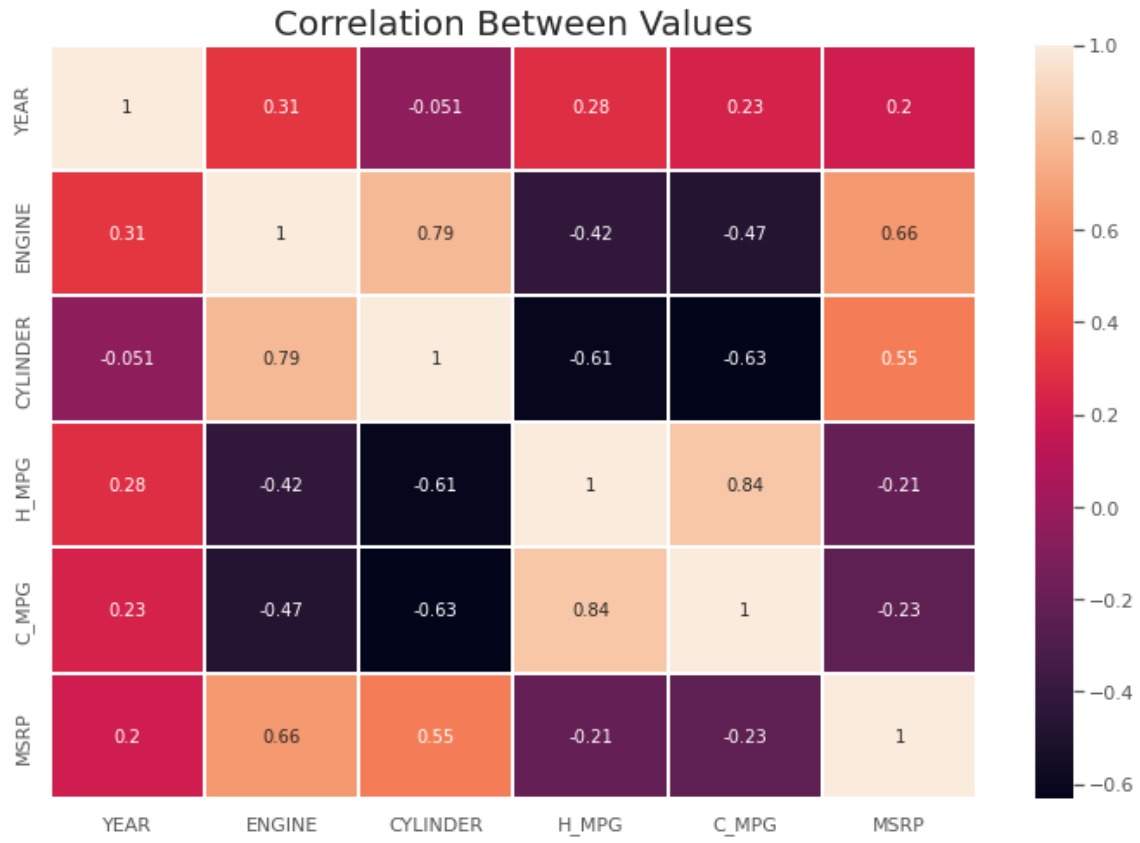
Using heatmaps plot the correlation between the features present in the dataset.

In [ ]:

```
#find the correlation of features of the data
# corr =

# print corr
```

In [ ]:

```
# Using the correlated df, plot the heatmap
# set cmap = 'BrBG', annot = True - to get the same graph as shown below
# set size of graph = (12,8)

plt.figure(figsize = (12,8))                                                    # C

sns.heatmap(new_new.corr(),annot = True, linewidth = 1)

plt.title('Correlation Between Values',fontsize = 20)
plt.show()
```



Observation:

A heatmap contains values representing various shades of the same colour for each value to be plotted. Usually the darker shades of the chart represent higher values than the lighter shade. For a very different value a completely different colour can also be used.

The above heatmap plot shows correlation between various variables in the colored scale of -1 to 1.