Predicting Cars Price on base of Properties using EDA

Problem Statement:

We have used Cars dataset with features including make, model, year, engine, and other properties of the car used to predict its price. data link: https://drive.google.com/file/d/1l9wkLD_2IWCF8lELBIND5zGKNZ1O9fiZ/view?usp=sharing

Importing the necessary libraries

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

Load the dataset into dataframe

```
# load the csv file in car data variable
car_data = pd.read_csv("/content/Cars_data_.csv")
```

```
# print the head of the dataframe
car_data.head()
```

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market Category	Ve
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Factory Tuner,Luxury,High-Performance	Co
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Co
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,High-Performance	Co
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Co
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury	Co

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

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

```
<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.

```
# initialise cols_to_drop
cols_to_drop = ["Engine Fuel Type", "Market Category", "Vehicle Style", "Popularity", "Number of Doors", "Vehicl
```

```
print(car_data.columns)
# drop the irrevalent cols and print the head of the dataframe
car_data.drop(cols_to_drop, axis = 1, inplace = True)

# print data head
car_data.head()
```

```
Index(['Make', 'Model', 'Year', 'Engine Fuel Type', 'Engine HP',
       'Engine Cylinders', 'Transmission Type', 'Driven_Wheels',
       'Number of Doors', 'Market Category', 'Vehicle Size', 'Vehicle Style',
       'highway MPG', 'city mpg', 'Popularity', 'MSRP'],
      dtype='object')
```

	Make	Model	Year	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	highway MPG	city 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

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

```
# rename columns
new_names = {'Make': 'Company',
             'Model': 'Model',
             'Engine HP': 'HP',
             'Engine Cylinders': 'Cylinders',
             'Transmission Type': 'Transmission',
             'Driven_Wheels': 'Drive Mode',
             'highway MPG': 'MPG-H',
             'city mpg': 'MPG-C',
             'MSRP': 'Price'
             }
```

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

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

	Company	Model	Year	HP	Cylinders	Transmission	Drive Mode	MPG-H	MPG-C	Price
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.

```
# number of rows before removing duplicated rows
car_data.shape
```

```
(11914, 10)
```

```
# drop the duplicated rows
car_data.drop_duplicates(inplace = True)

# print head of data
car_data.head()
```

	Company	Model	Year	HP	Cylinders	Transmission	Drive Mode	MPG-H	MPG-C	Price
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

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

```
(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

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

	0
Company	0
Model	0
Year	0
HP	69
Cylinders	30
Transmission	0
Drive Mode	0
MPG-H	0
MPG-C	0
Price	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.

```
# drop missing values
car_data.dropna(inplace = True)
```

```
# check number of nan values in each col again
car_data.isnull().sum()
```

	0
Company	0
Model	0
Year	0
HP	0
Cylinders	0
Transmission	0
Drive Mode	0
MPG-H	0
MPG-C	0
Price	0

dtype: int64

```
#Describe statistics of data
car_data.describe()
```

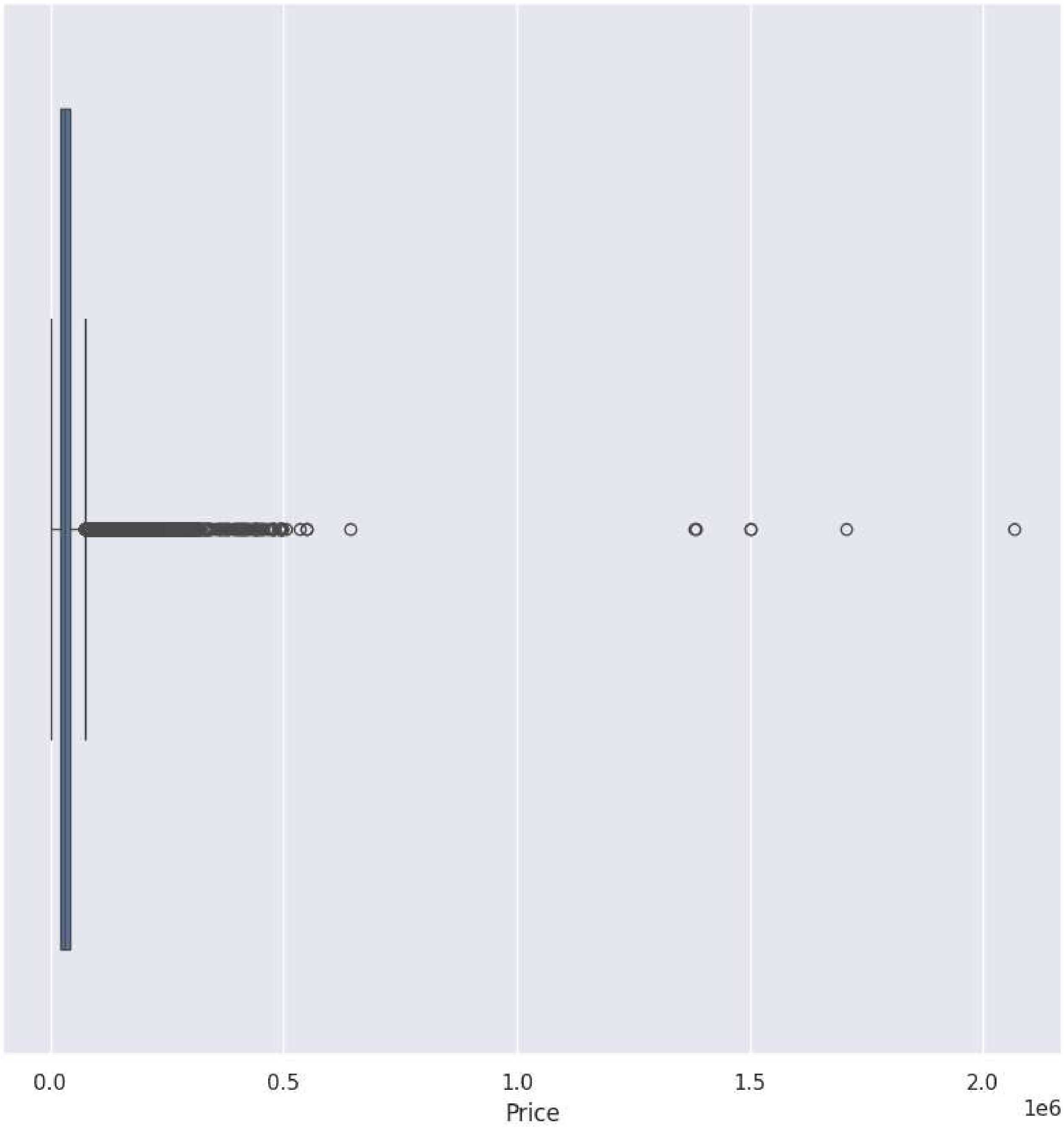
	Year	HP	Cylinders	MPG-H	MPG-C	Price
count	10827.000000	10827.000000	10827.000000	10827.000000	10827.000000	1.082700e+04
mean	2010.896370	254.553062	5.691604	26.308119	19.327607	4.249325e+04
std	7.029534	109.841537	1.768551	7.504652	6.643567	6.229451e+04
min	1990.000000	55.000000	0.000000	12.000000	7.000000	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.

```
# Plot a boxplot for 'Price' column in dataset
plt.figure(figsize = (10, 10))
sns.boxplot(x = car_data['Price'])
```

```
<Axes: xlabel='Price'>
```

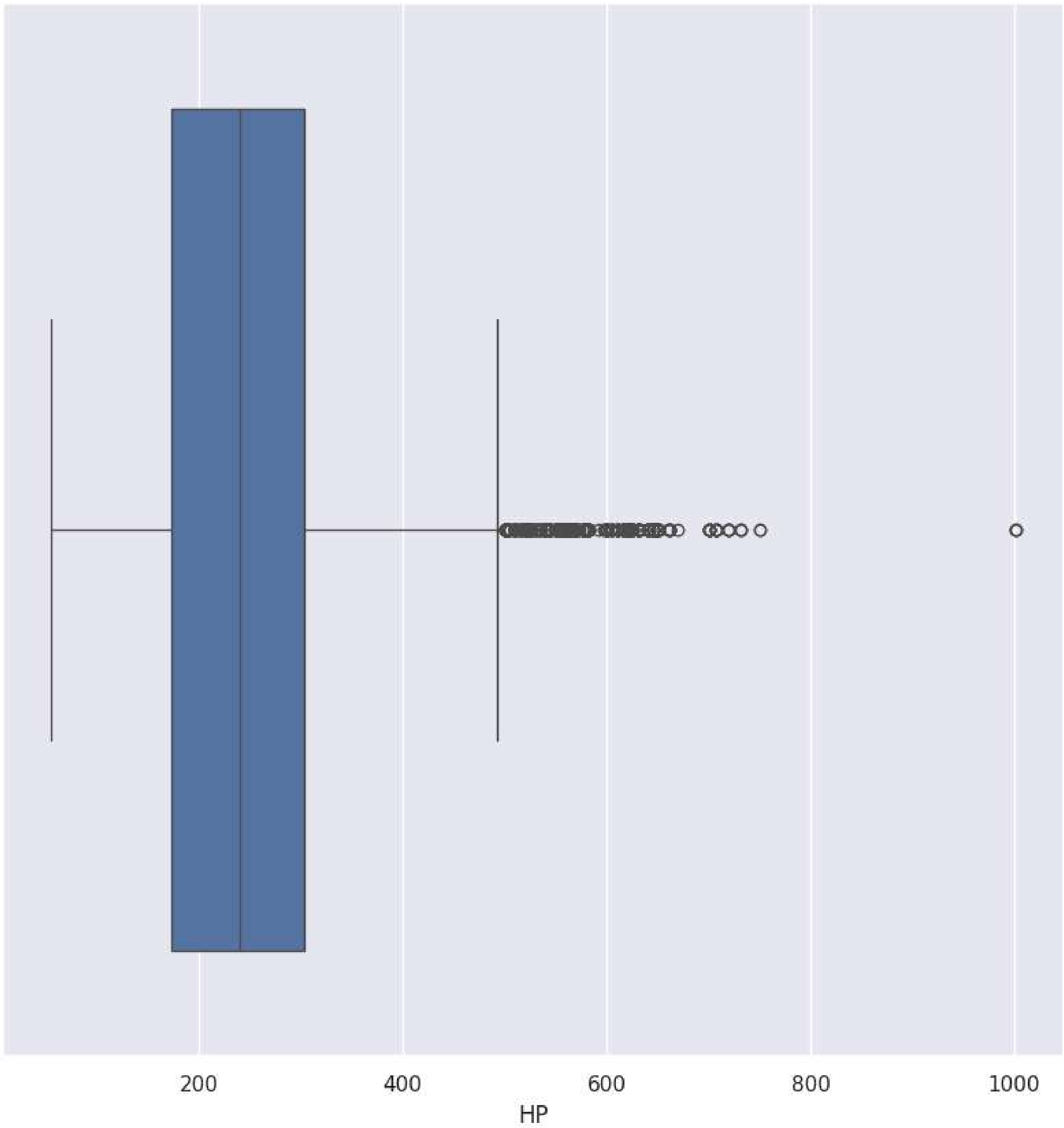


Observation:

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.

```
## PLot a boxplot for 'HP' columns in dataset
plt.figure(figsize = (10, 10))
sns.boxplot(x = car_data['HP'])
```

<Axes: xlabel='HP'>



Observation:

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

Start coding or generate with AI.

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

Hint: Use loc with condition

```
# print all the columns which are of int or float datatype in data
car_data.select_dtypes(include=['int64', 'float64']).columns
```

```
Index(['Year', 'HP', 'Cylinders', 'MPG-H', 'MPG-C', 'Price'], dtype='object')
```

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

```
# save column names of the above output in variable list
clm = car_data.select_dtypes(include=['int64', 'float64']).columns
l = list(clm)
```

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)

```
# define Q1 and Q2
Q1 = car_data['Price'].quantile(0.25)
Q3 = car_data['Price'].quantile(0.75)


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

# # define df2 after removing outliers
car_data2 = car_data[~((car_data['Price'] < (Q1 - 1.5 * IQR)) | (car_data['Price'] > (Q3 + 1.5 * IQR)))]


Q1 = car_data['HP'].quantile(0.25)
Q3 = car_data['HP'].quantile(0.75)

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

# # define df2 after removing outliers
car_data2 = car_data[~((car_data['HP'] < (Q1 - 1.5 * IQR)) | (car_data ['HP']> (Q3 + 1.5 * IQR)))]
```

```
# find the shape of car_data
car_data.shape
```

```
(10827, 10)
```

```
# find the shape of car_data2
car_data2.shape
```

```
(10332, 10)
```

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

for i in car_data2.columns:
```

```
print(car_data[i].value_counts())
```

```
all wheel drive      2281
four wheel drive     1258
Name: count, dtype: int64
MPG-H
24     822
23     758
26     725
22     686
25     685
28     651
27     555
30     499
21     488
19     488
31     488
20     469
29     425
18     345
17     340
33     329
32     292
34     270
16     199
35     199
36     191
37     166
38     130
15     116
40     109
39     107
41      65
42      46
14      37
43      21
46      21
44      21
48      16
45      14
13      13
50      10
47       7
109      6
12       5
53       5
82       3
111      3
354      1
106      1
Name: count, dtype: int64
MPG-C
17     1154
16     1014
15      949
18      938
19      793
20      742
14      603
22      571
21      551
```

Visualising Univariate Distributions

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

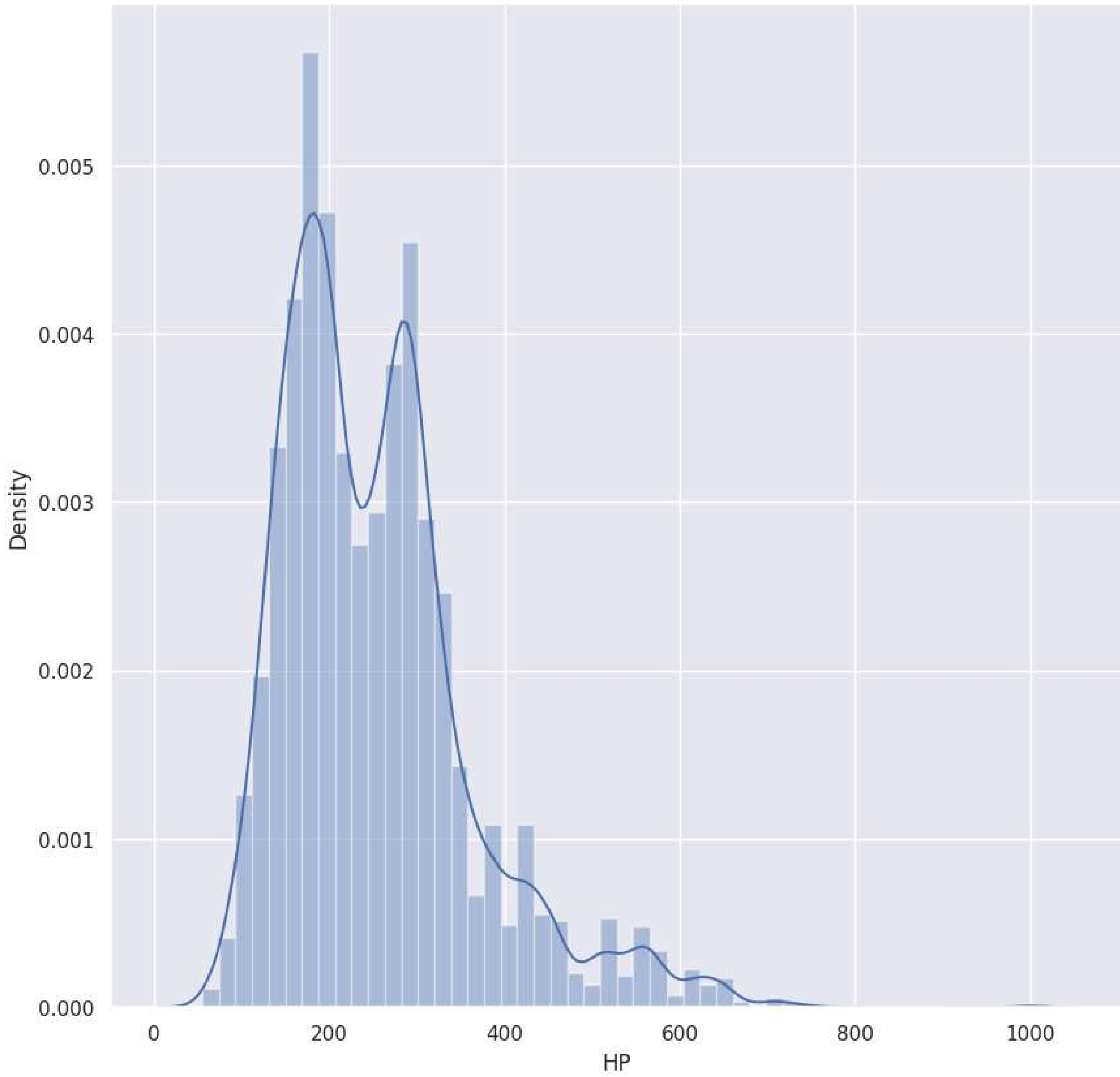
Do you know? you have just now already explored one univariate plot. guess which one? Yeah its box plot.

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`.

```
#ploting distplot for variable HP
plt.figure(figsize = (10, 10))
sns.distplot(car_data['HP'])
```

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



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

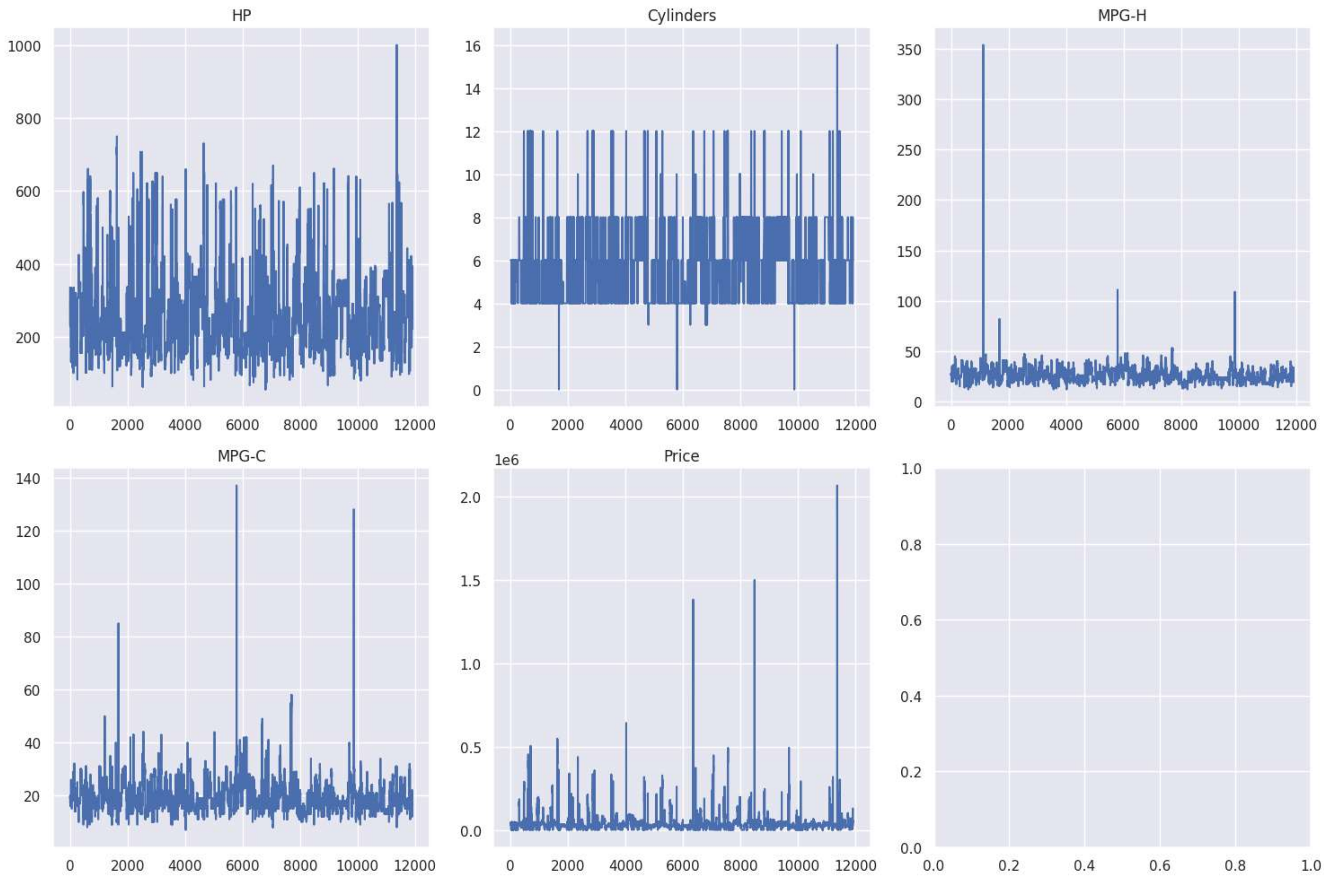
```
# plot all the columns present in list l together using subplot of dimention (2,3)

l = ['HP', 'Cylinders', 'MPG-H', 'MPG-C', 'Price']

fig, axes = plt.subplots(2, 3, figsize=(15, 10))
axes = axes.ravel()

for i, col in enumerate(l):
  axes[i].plot(car_data[col])
  axes[i].set_title(col)

plt.tight_layout()
plt.show()
```



Bar Chart Plots

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

```
plt.figure(figsize = (12,8))
car_data.nlargest(10, 'Price')['Price'].plot(kind='bar')
plt.title('Top 10 Car Prices')
plt.xlabel('Car Model')
plt.ylabel('Price')
```

```
plt.show()
```

Observation:

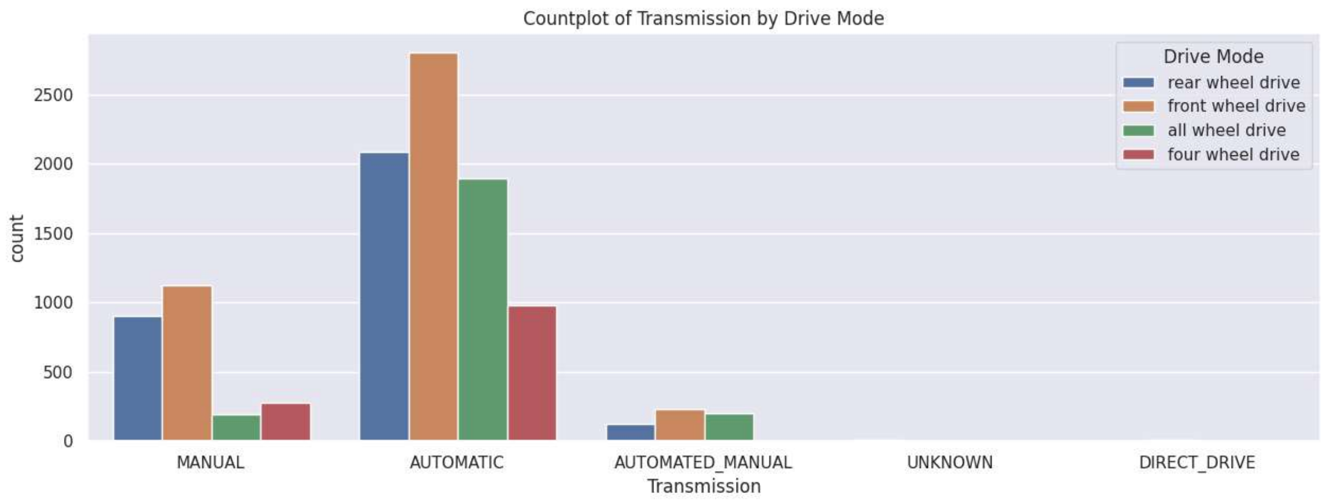
In this plot we can see that we have plot the bar plot with the cars model and nos. of cars.

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

```
plt.figure(figsize=(15,5))
sns.countplot(x='Transmission', hue='Drive Mode', data=car_data)
plt.title('Countplot of Transmission by Drive Mode')
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

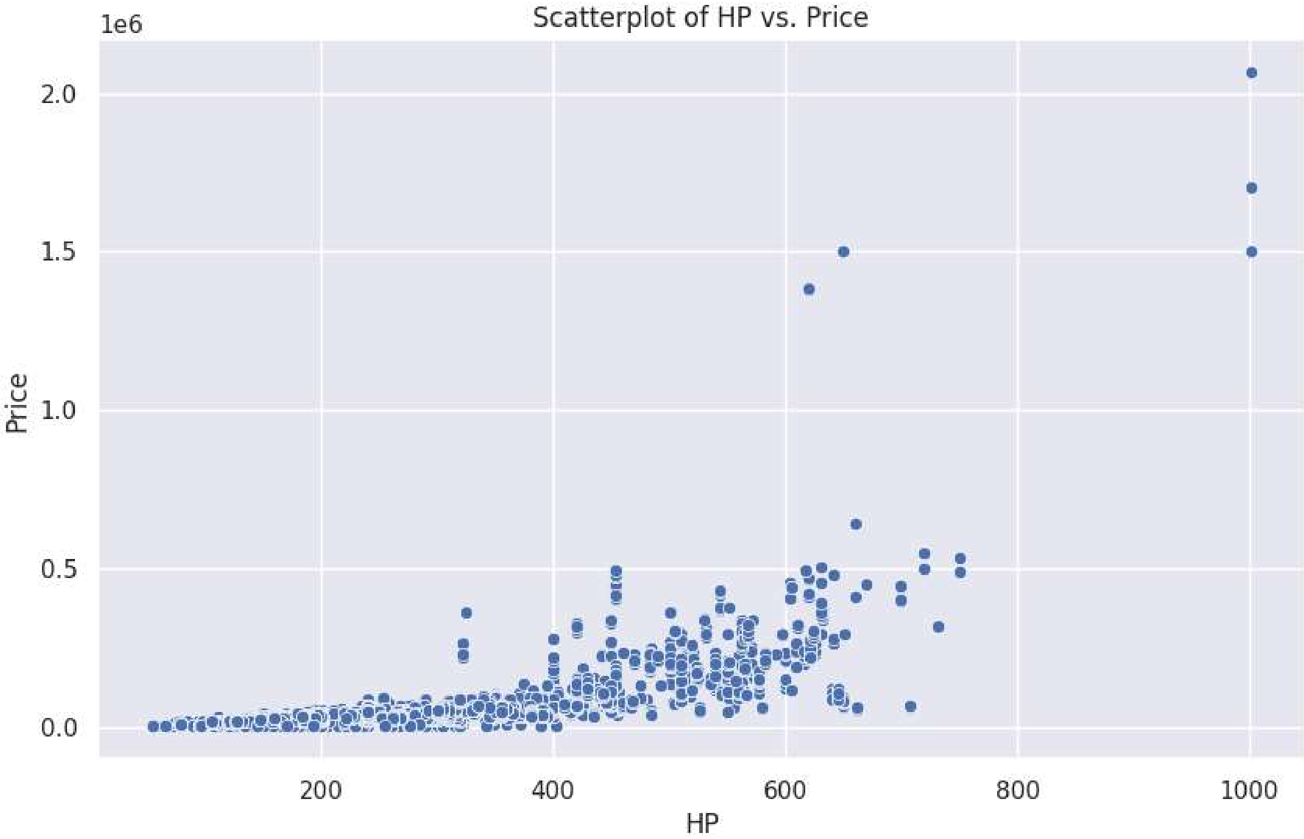
Bivariate distributions are simply two univariate distributions plotted on x and y axes respectively. They help you observe the relationship between the two variables.

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.

```
fig, ax = plt.subplots(figsize=(10,6))
sns.scatterplot(x='HP', y='Price', data=car_data)
plt.title('Scatterplot of HP vs. Price')
plt.show()
```



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.

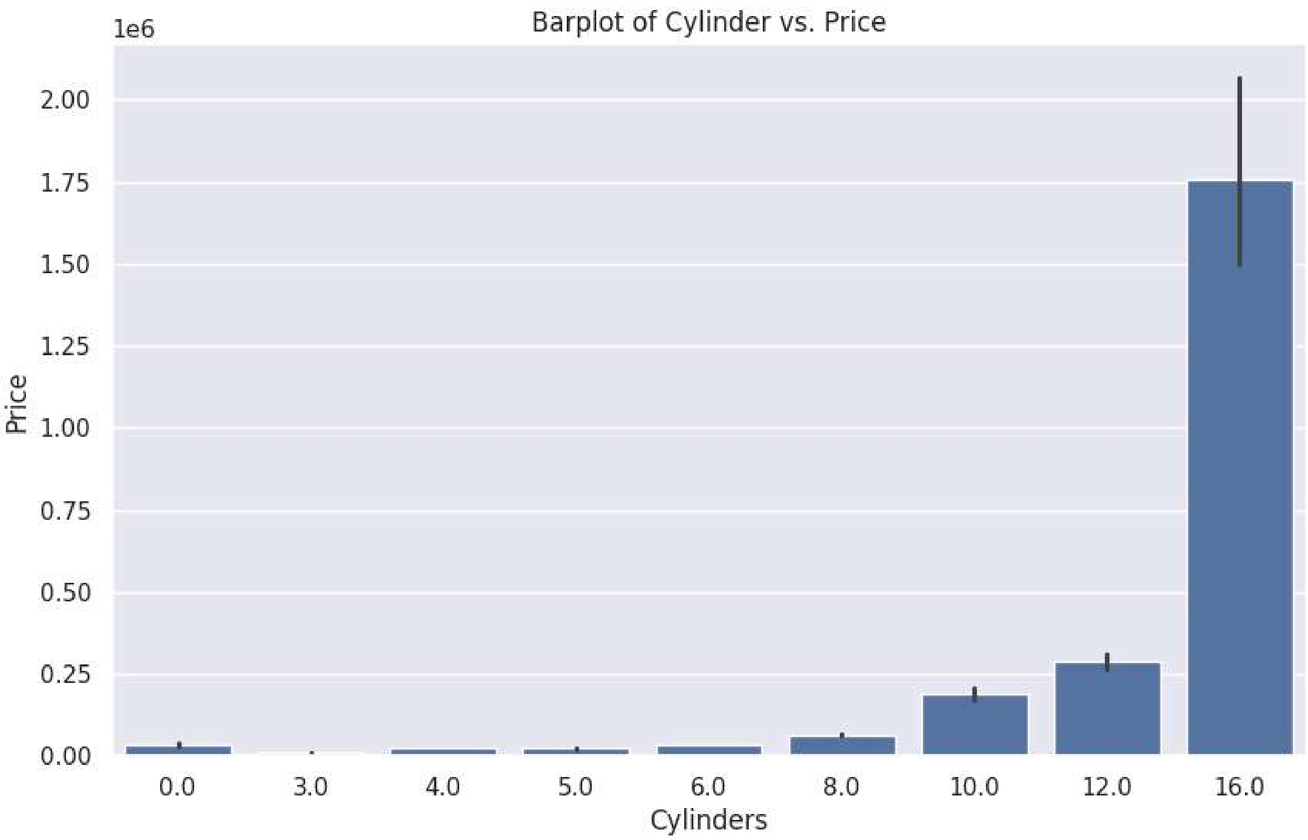
Plotting Aggregated Values across Categories

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.

```
# bar plot with default statistic=mean between Cylinder and Price
plt.figure(figsize=(10,6))
sns.barplot(x='Cylinders', y='Price', data=car_data)
plt.title('Barplot of Cylinder vs. Price')
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.

```
# Plotting categorical variable Transmission across the y-axis
plt.figure(figsize=(10,6))
sns.barplot(x='Transmission', y='Price', data=car_data)
plt.title('Barplot of Transmission vs. Price')
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.

```
#find the correlation of features of the data
corr = car_data[l].corr()
corr
```

	HP	Cylinders	MPG-H	MPG-C	Price
HP	1.000000	0.788007	-0.420281	-0.473551	0.659835
Cylinders	0.788007	1.000000	-0.611576	-0.632407	0.554740
MPG-H	-0.420281	-0.611576	1.000000	0.841229	-0.209150
MPG-C	-0.473551	-0.632407	0.841229	1.000000	-0.234050
Price	0.659835	0.554740	-0.209150	-0.234050	1.000000