Connect, Message, Like,Follow&Share, 100% Free Counselling➔ThankYou

Follow me on Linkedin

Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is an approach to analyze the data using visual techniques. It is used to discover trends, patterns, or to check assumptions with the help of statistical summary and graphical representations. **Exploratory Data Analysis** is a process of examining or understanding the data and extracting insights or main characteristics of the data. EDA is generally classified into two methods, i.e. **graphical** analysis and **non-graphical** analysis. EDA is very essential because it is a good practice to first understand the problem statement and the various relationships between the data features before getting your hands dirty.

Exploratory Data Analysis

- Examine the data distribution
- Handling missing values of the dataset(a most common issue with every dataset)
- Handling the outliers
- Removing duplicate data
- Encoding the categorical variables
- Normalizing and Scaling

To understand the steps involved in EDA, we will use Python as the programming language and Jupyter Notebooks because it’s open-source, and not only it’s an excellent IDE but also very good for visualization and presentation.

Step 1

First, we will import all the python libraries that are required for this, which include **NumPy** for numerical calculations and scientific computing, **Pandas** for handling data, and **Matplotlib** and **Seaborn** for visualization.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pylab import rcParams
rcParams['figure.figsize'] = (12,6)
sns.set()
```

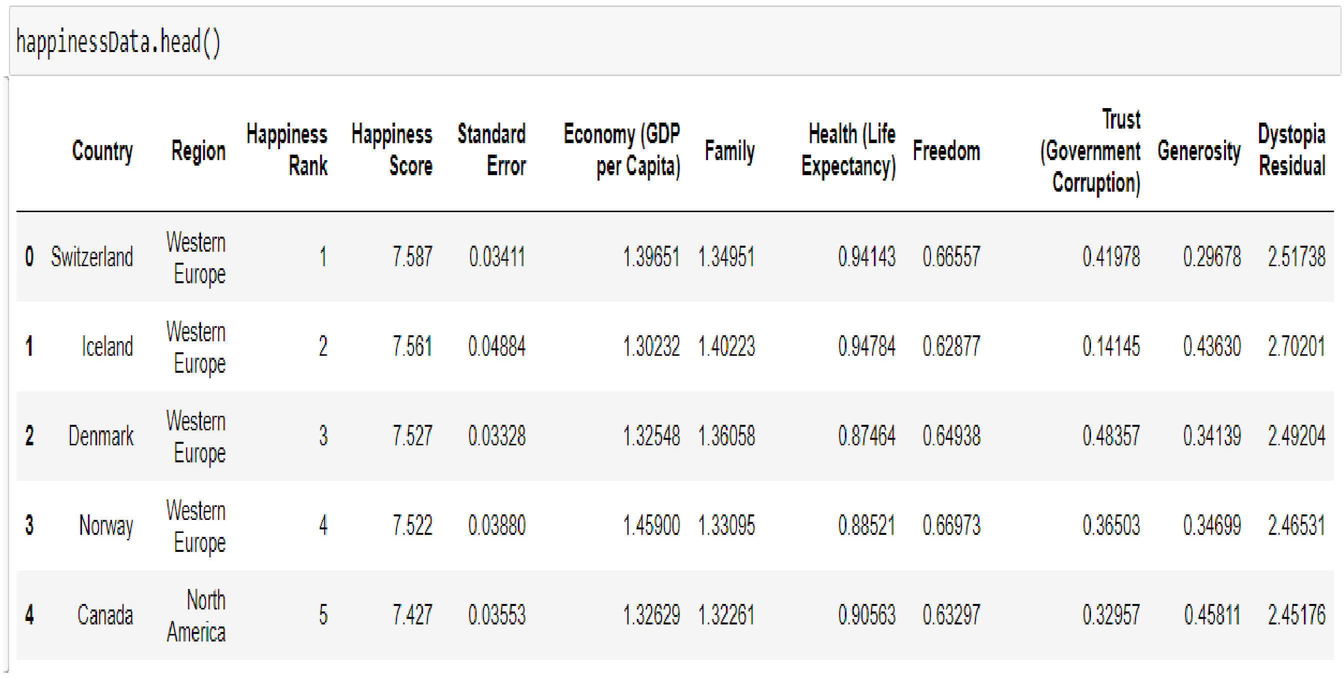
Step 2

Then we will load the data into the **Pandas** data frame. For this analysis, we will use a dataset of “World Happiness Report”, which has the following columns: GDP per Capita, Family, Life Expectancy, Freedom, Generosity, Trust Government Corruption, etc. to describe the extent to which these factors contribute to evaluating the happiness.

```
happinessData = pd.read_csv("C:\\Users\\nkr4n\\Documents\\JupyterNotebooks\\files\\Happiness.csv")
```

Step 3

We can observe the dataset by checking a few of the rows using the **head()** method, which returns the first five records from the dataset.

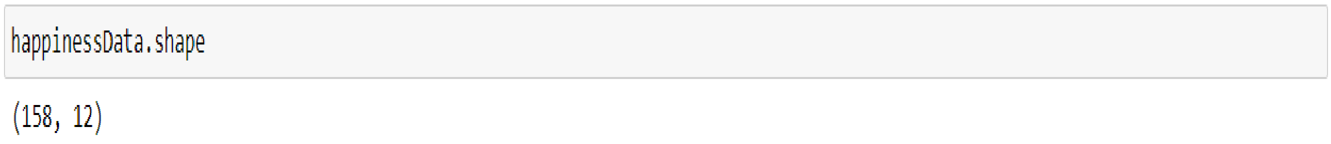


```
happinessData.head()
```

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystopia Residual
0	Switzerland	Western Europe	1	7.587	0.03411	1.39651	1.34951	0.94143	0.66557	0.41978	0.29678	2.51738
1	Iceland	Western Europe	2	7.561	0.04884	1.30232	1.40223	0.94784	0.62877	0.14145	0.43630	2.70201
2	Denmark	Western Europe	3	7.527	0.03328	1.32548	1.36058	0.87464	0.64938	0.48357	0.34139	2.49204
3	Norway	Western Europe	4	7.522	0.03880	1.45900	1.33095	0.88521	0.66973	0.36503	0.34699	2.46531
4	Canada	North America	5	7.427	0.03553	1.32629	1.32261	0.90563	0.63297	0.32957	0.45811	2.45176

Step 4

Using **shape**, we can observe the dimensions of the data.

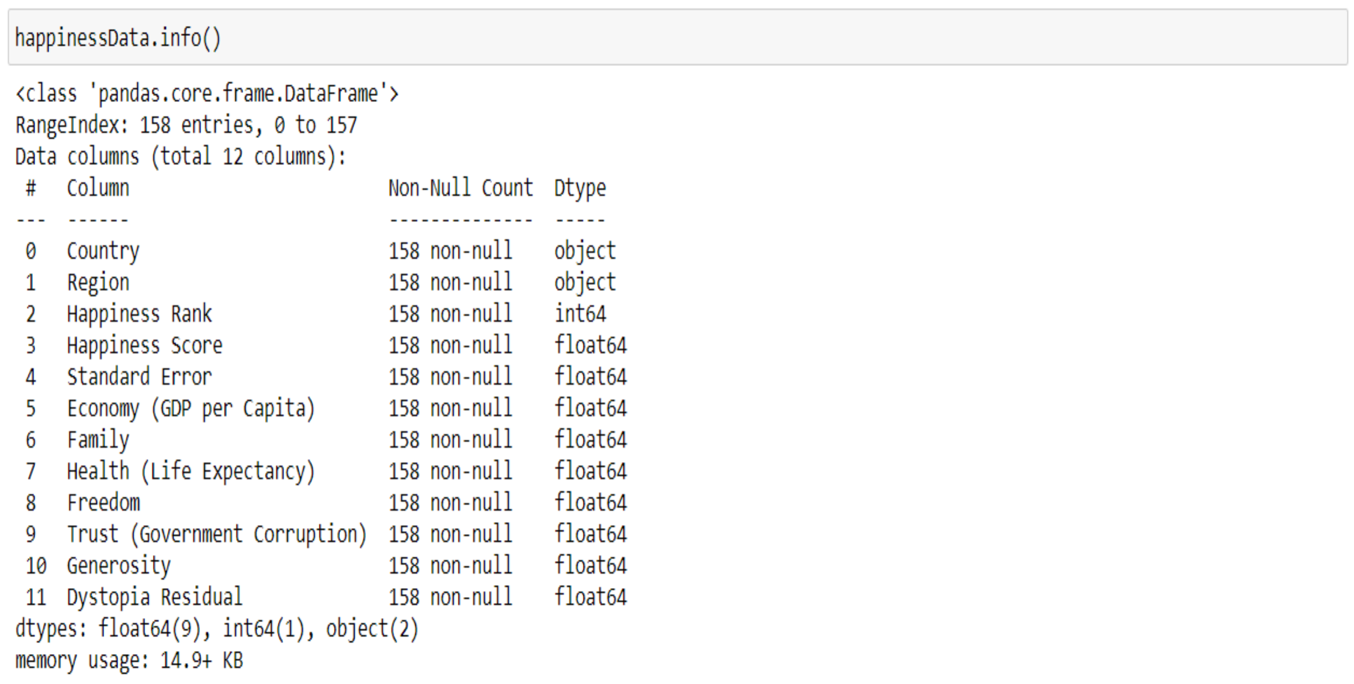


```
happinessData.shape
```

```
(158, 12)
```

Step 5

info() method shows some of the characteristics of the data such as Column Name, No. of non-null values of our columns, Dtype of the data, and Memory Usage.



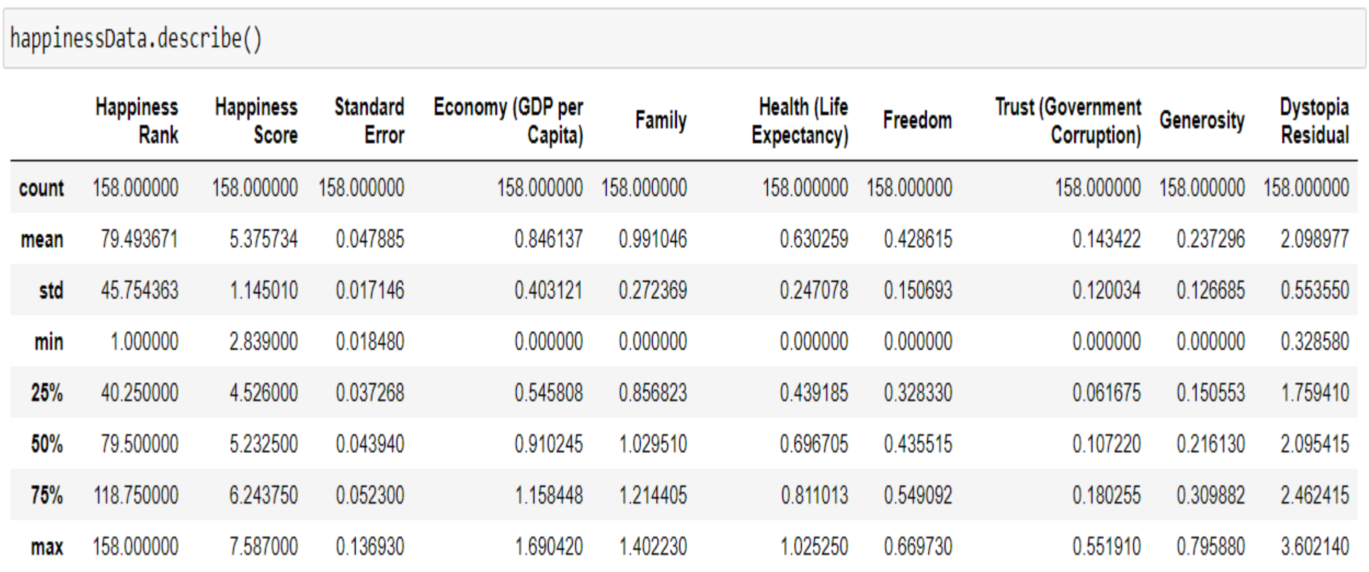
```
happinessData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 158 entries, 0 to 157
Data columns (total 12 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   Country                        158 non-null    object 
 1   Region                         158 non-null    object 
 2   Happiness Rank                 158 non-null    int64  
 3   Happiness Score                158 non-null    float64
 4   Standard Error                 158 non-null    float64
 5   Economy (GDP per Capita)       158 non-null    float64
 6   Family                         158 non-null    float64
 7   Health (Life Expectancy)       158 non-null    float64
 8   Freedom                        158 non-null    float64
 9   Trust (Government Corruption)  158 non-null    float64
 10  Generosity                     158 non-null    float64
 11  Dystopia Residual              158 non-null    float64
dtypes: float64(9), int64(1), object(2)
memory usage: 14.9+ KB
```

From this, we can observe, that the data which we have doesn't have any missing values. We are very lucky in this case, but in real-life scenarios, the data usually has missing values which we need to handle for our model to work accurately. (Note – Later on, I'll show you how to handle the data if it has missing values in it)

Step 6

We will use **describe()** method, which shows basic statistical characteristics of each numerical feature (int64 and float64 types): number of non-missing values, mean, standard deviation, range, median, 0.25, 0.50, 0.75 quartiles.



```
happinessData.describe()
```

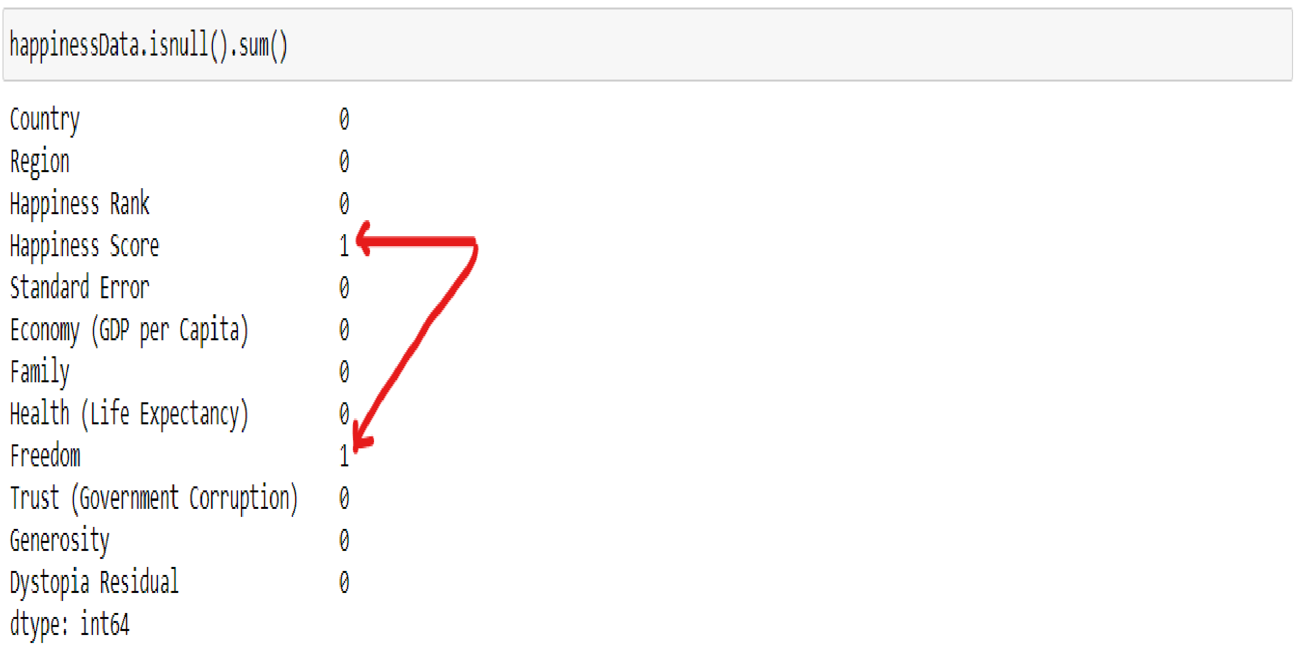
	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystopia Residual
count	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000
mean	79.493671	5.375734	0.047885	0.846137	0.991046	0.630259	0.428615	0.143422	0.237296	2.098977
std	45.754363	1.145010	0.017146	0.403121	0.272369	0.247078	0.150693	0.120034	0.126685	0.553550
min	1.000000	2.839000	0.018480	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.328580
25%	40.250000	4.526000	0.037268	0.545808	0.856823	0.439185	0.328330	0.061675	0.150553	1.759410
50%	79.500000	5.232500	0.043940	0.910245	1.029510	0.696705	0.435515	0.107220	0.216130	2.095415
75%	118.750000	6.243750	0.052300	1.158448	1.214405	0.811013	0.549092	0.180255	0.309882	2.462415
max	158.000000	7.587000	0.136930	1.690420	1.402230	1.025250	0.669730	0.551910	0.795880	3.602140

Step 7

Handling missing values in the dataset. Luckily, this dataset doesn't have any missing values, but the real world is not so naive as our case.

So I have removed a few values intentionally just to depict how to handle this particular case.

We can check if our data contains a null value or not by the following command



```
happinessData.isnull().sum()
```

```
Country                          0
Region                           0
Happiness Rank                   0
Happiness Score                  1
Standard Error                   0
Economy (GDP per Capita)         0
Family                           0
Health (Life Expectancy)         0
Freedom                          1
Trust (Government Corruption)    0
Generosity                       0
Dystopia Residual                0
dtype: int64
```

As we can see that “Happiness Score” and “Freedom” features have 1 missing values each.

So, now we can handle the missing values by using a few techniques, which are

- **Drop the missing values** – If the dataset is huge and missing values are very few then we can directly drop the values because it will not have much impact.
- **Replace with mean values** – We can replace the missing values with mean values, but this is not advisable in case if the data has outliers.
- **Replace with median values** – We can replace the missing values with median values, and it is recommended in case if the data has outliers.
- **Replace with mode values** – We can do this in the case of a Categorical feature.
- **Regression** – It can be used to predict the null value using other details from the dataset.

For our case, we will handle missing values by replacing them with the median value.

```
medianHappinessScore = happinessData['Happiness Score'].median()
medianFreedom = happinessData['Freedom'].median()
```

```
happinessData['Happiness Score'].replace(np.nan,medianHappinessScore,inplace=True)
happinessData['Freedom'].replace(np.nan,medianFreedom,inplace=True)
```

And, now we can again check if the missing values have been handled or not.

```
happinessData.isnull().sum()
```

```
Country                          0
Region                           0
Happiness Rank                   0
Happiness Score                  0
Standard Error                   0
Economy (GDP per Capita)         0
Family                           0
Health (Life Expectancy)         0
Freedom                          0
Trust (Government Corruption)    0
Generosity                       0
Dystopia Residual                0
dtype: int64
```

And, now we can see that our dataset doesn't have any null values now.

Step 8

We can check for duplicate values in our dataset as the presence of duplicate values will hamper the accuracy of our ML model.

```
duplicateValues = happinessData.duplicated()
print(duplicateValues.sum())
happinessData[duplicateValues]
```

```
2
```

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystopia Residual
147	Costa Rica	Latin America and Caribbean	12	7.226	0.04454	0.95578	1.23788	0.86027	0.63376	0.10583	0.25497	3.17728
153	Singapore	Southeastern Asia	24	6.798	0.03780	1.52186	1.02000	1.02525	0.54252	0.49210	0.31105	1.88501

We can remove duplicate values using **drop_duplicates()**

```
happinessData.drop_duplicates(inplace=True)
```

```
duplicatedValues = happinessData.duplicated()
duplicatedValues.sum()
```

```
0
```

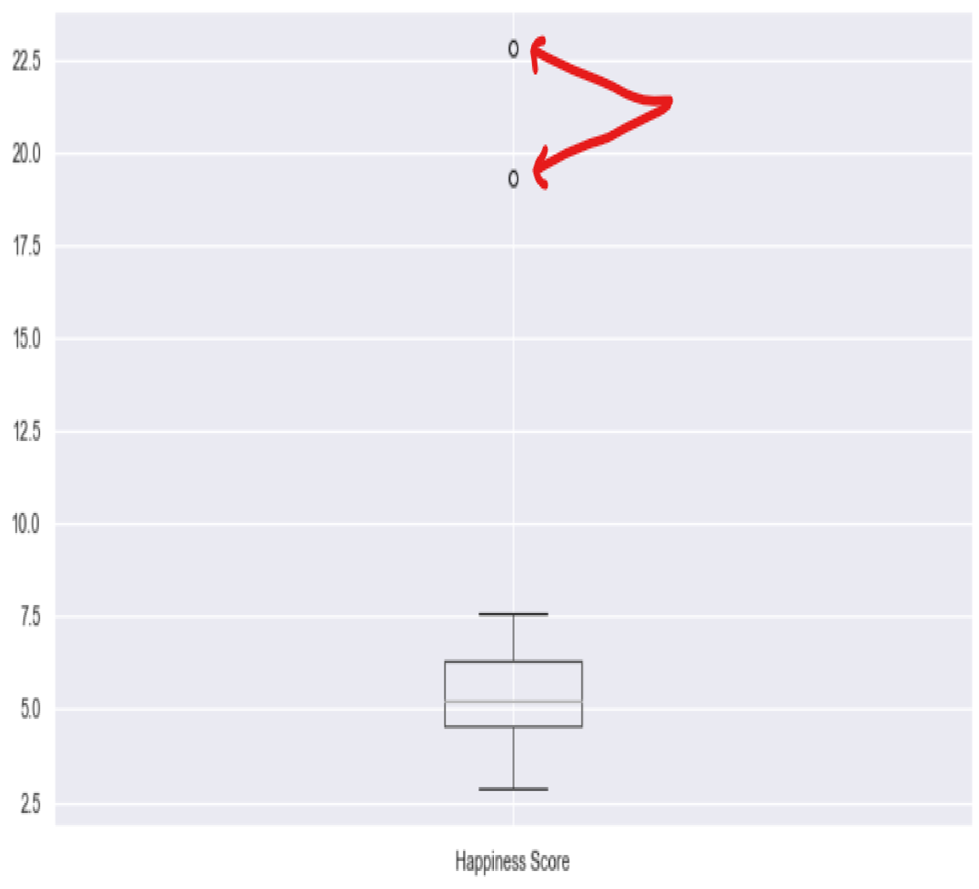
As we can see that the duplicate values are now handled.

Step 9

Handling the **outliers** in the data, i.e. the **extreme values** in the data. We can find the outliers in our data using a Boxplot.

```
happinessData.boxplot(column=['Happiness Score'])
plt.show
```

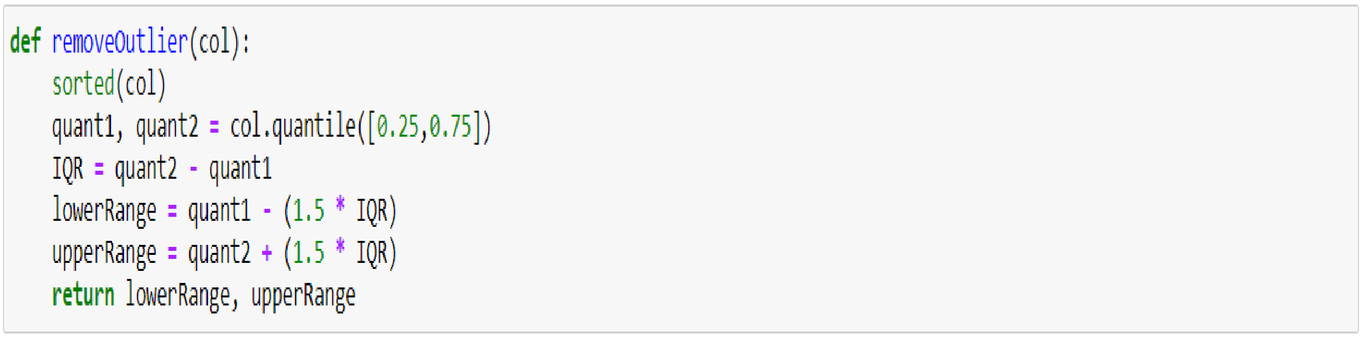
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



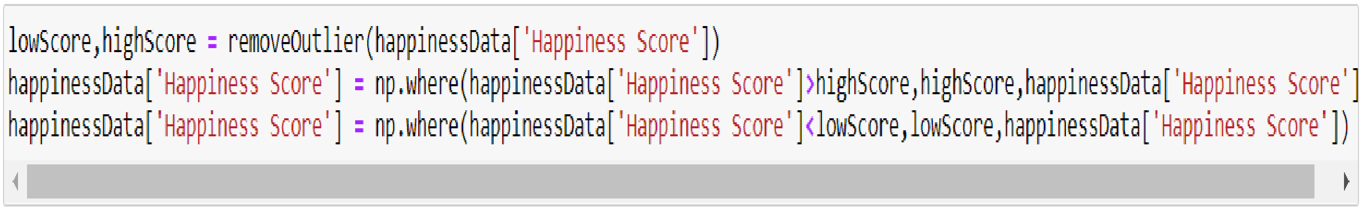
As we can observe from the above boxplot that the normal range of data lies within the block and the outliers are denoted by the small circles in the extreme end of the graph.

So to handle it we can either drop the outlier values or replace the outlier values using **IQR(Interquartile Range Method)**.

IQR is calculated as the difference between the 25th and the 75th percentile of the data. The percentiles can be calculated by sorting the selecting values at specific indices. The IQR is used to identify outliers by defining limits on the sample values that are a factor k of the IQR. The common value for the factor k is the value 1.5.



```python
def removeOutlier(col):
    sorted(col)
    quant1, quant2 = col.quantile([0.25,0.75])
    IQR = quant2 - quant1
    lowerRange = quant1 - (1.5 * IQR)
    upperRange = quant2 + (1.5 * IQR)
    return lowerRange, upperRange
```

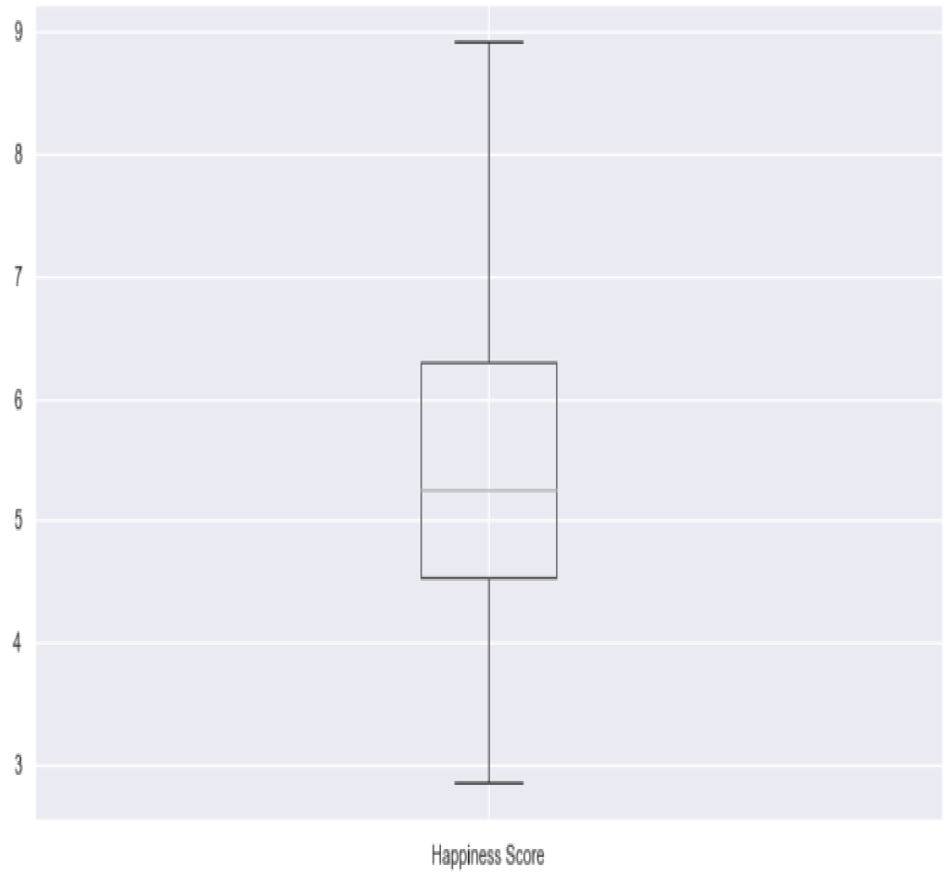


```python
lowScore,highScore = removeOutlier(happinessData['Happiness Score'])
happinessData['Happiness Score'] = np.where(happinessData['Happiness Score']>highScore,highScore,happinessData['Happiness Score']
happinessData['Happiness Score'] = np.where(happinessData['Happiness Score']<lowScore,lowScore,happinessData['Happiness Score'])
```

Now we can again plot the boxplot and check if the outliers have been handled or not.

```
happinessData.boxplot(column=['Happiness Score'])
plt.show
```

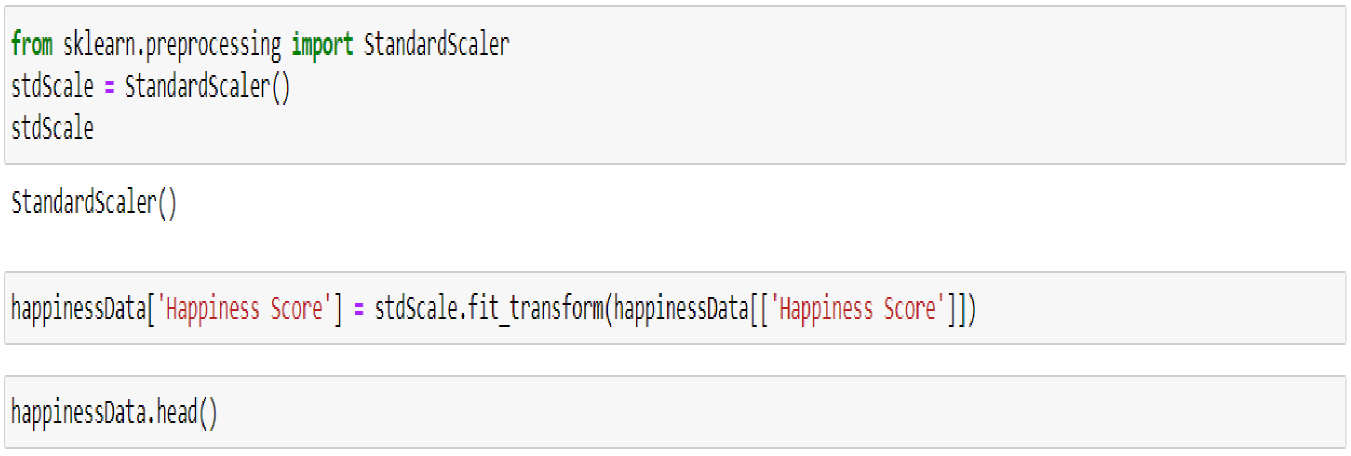
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Finally, we can observe that our data is now free from outliers.

Step 10

Normalizing and Scaling – Data Normalization or **feature scaling** is a process to standardize the range of features of the data as the range may vary a lot. So we can preprocess the data using ML algorithms. So for this, we will use **StandardScaler** for the numerical values, which uses the formula as **x-mean/std deviation**.

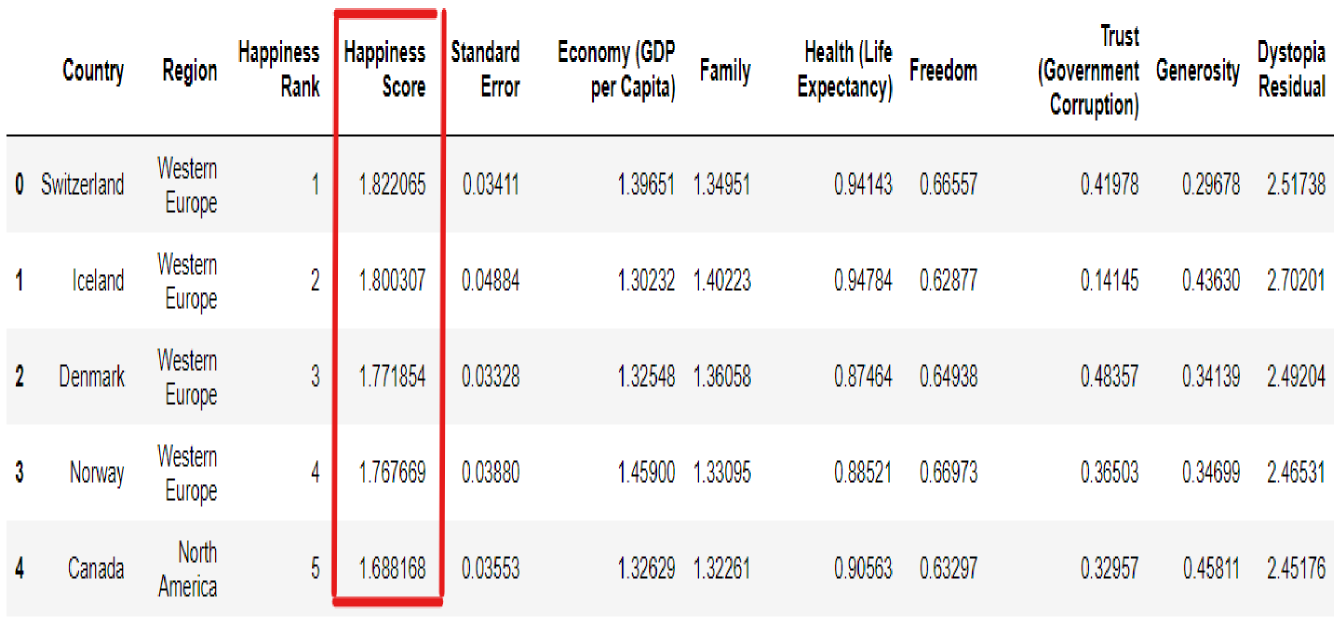


```
from sklearn.preprocessing import StandardScaler
stdScale = StandardScaler()
stdScale
```

```
StandardScaler()
```

```
happinessData['Happiness Score'] = stdScale.fit_transform(happinessData[['Happiness Score']])
```

```
happinessData.head()
```



	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystopia Residual
0	Switzerland	Western Europe	1	1.822065	0.03411	1.39651	1.34951	0.94143	0.66557	0.41978	0.29678	2.51738
1	Iceland	Western Europe	2	1.800307	0.04884	1.30232	1.40223	0.94784	0.62877	0.14145	0.43630	2.70201
2	Denmark	Western Europe	3	1.771854	0.03328	1.32548	1.36058	0.87464	0.64938	0.48357	0.34139	2.49204
3	Norway	Western Europe	4	1.767669	0.03880	1.45900	1.33095	0.88521	0.66973	0.36503	0.34699	2.46531
4	Canada	North America	5	1.688168	0.03553	1.32629	1.32261	0.90563	0.63297	0.32957	0.45811	2.45176

As we can see that the "Happiness Score" column has been normalized.

Step 11

We can find the **pairwise correlation** between the different columns of the data using the **corr()** method. (Note – All non-numeric data type column will be ignored.)

happinessData.corr() is used to find the pairwise correlation of all columns in the data frame. Any 'nan' values are automatically excluded.

The resulting coefficient is a value between -1 and 1 inclusive, where:

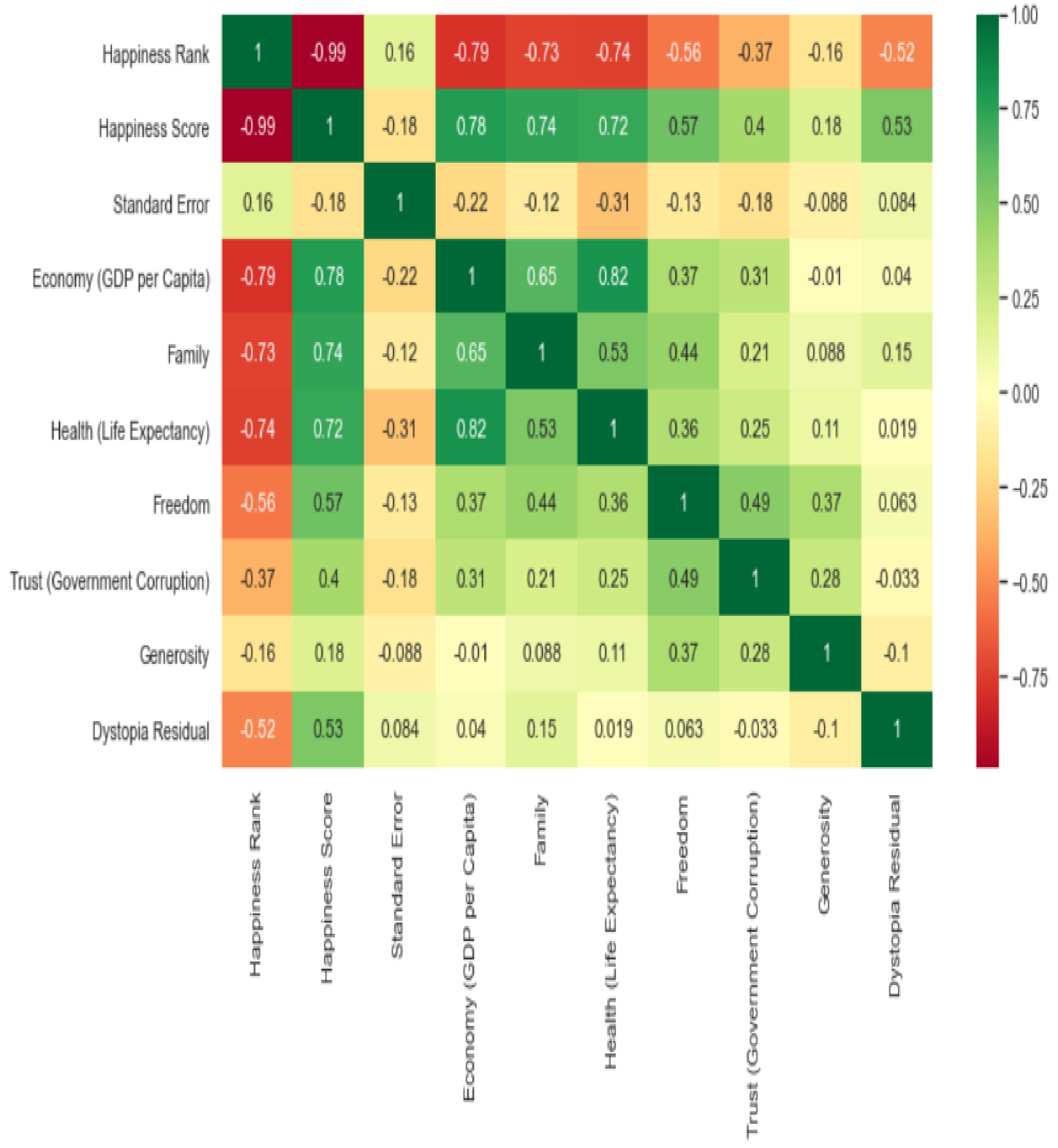
- 1: Total positive linear correlation
- 0: No linear correlation, the two variables most likely do not affect each other
- -1: Total negative linear correlation

Pearson Correlation is the default method of the function “corr”.

Now, we will create a **heatmap** using Seaborn to visualize the correlation between the different columns of our data:

```
sns.heatmap(happinessData.corr(),annot=True,cmap='RdYlGn')
```

```
<AxesSubplot:>
```



	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystopia Residual
Happiness Rank	1	-0.99	0.16	-0.79	-0.73	-0.74	-0.56	-0.37	-0.16	-0.52
Happiness Score	-0.99	1	-0.18	0.78	0.74	0.72	0.57	0.4	0.18	0.53
Standard Error	0.16	-0.18	1	-0.22	-0.12	-0.31	-0.13	-0.18	-0.088	0.084
Economy (GDP per Capita)	-0.79	0.78	-0.22	1	0.65	0.82	0.37	0.31	-0.01	0.04
Family	-0.73	0.74	-0.12	0.65	1	0.53	0.44	0.21	0.088	0.15
Health (Life Expectancy)	-0.74	0.72	-0.31	0.82	0.53	1	0.36	0.25	0.11	0.019
Freedom	-0.56	0.57	-0.13	0.37	0.44	0.36	1	0.49	0.37	0.063
Trust (Government Corruption)	-0.37	0.4	-0.18	0.31	0.21	0.25	0.49	1	0.28	-0.033
Generosity	-0.16	0.18	-0.088	-0.01	0.088	0.11	0.37	0.28	1	-0.1
Dystopia Residual	-0.52	0.53	0.084	0.04	0.15	0.019	0.063	-0.033	-0.1	1

As we can observe from the above heatmap of correlations, there is a high correlation between –

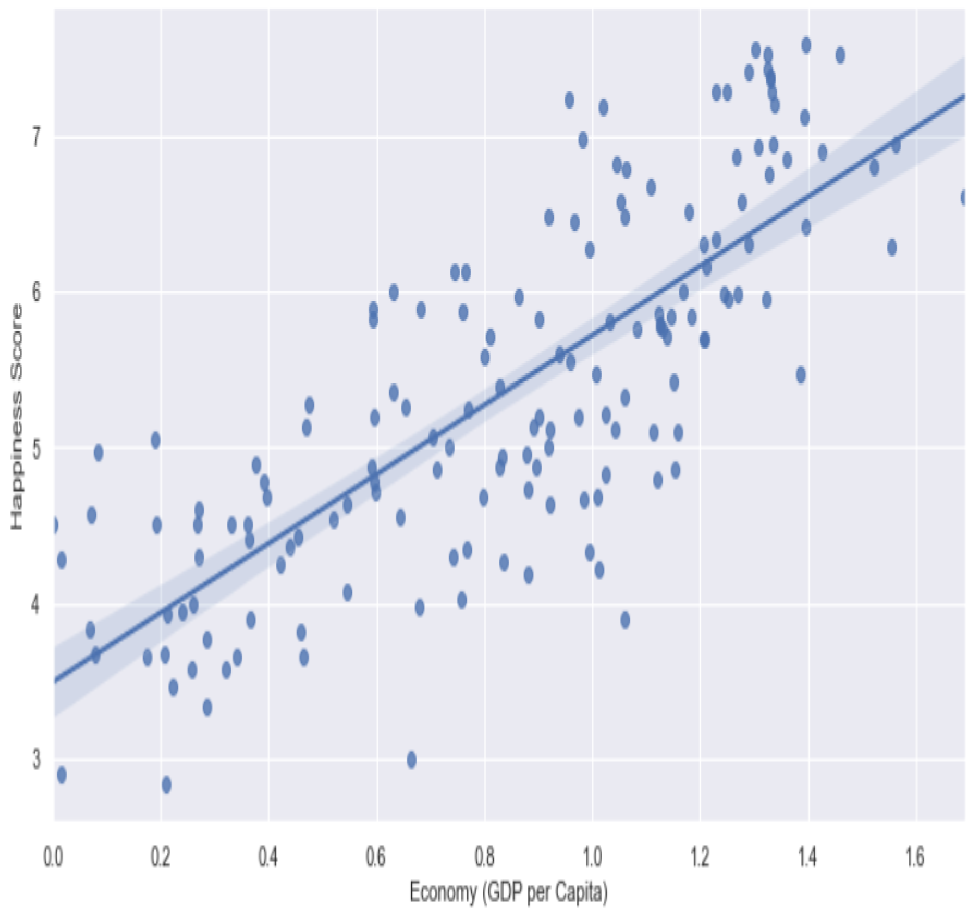
- Happiness Score – Economy (GDP per Capita) = 0.78
- Happiness Score – Family = 0.74
- Happiness Score – Health (Life Expectancy) = 0.72
- Economy (GDP per Capita) – Health (Life Expectancy) = 0.82

Step 12

Now, using Seaborn, we will visualize the relation between Economy (GDP per Capita) and Happiness Score by using a **regression plot**. And as we can see, as the Economy increases, the Happiness Score increases as well as denoting a **positive relation**.

```
sns.regplot(x='Economy (GDP per Capita)',y='Happiness Score',data=happinessData)
```

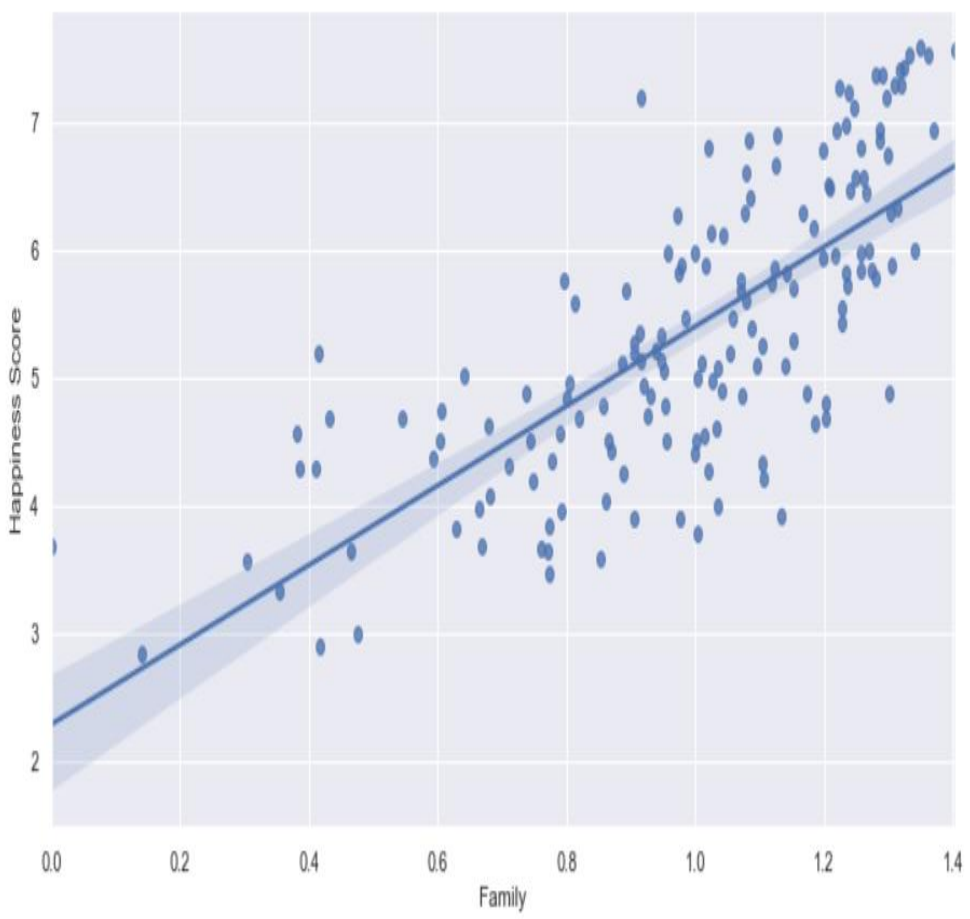
```
<AxesSubplot:xlabel='Economy (GDP per Capita)', ylabel='Happiness Score'>
```



Now, we will visualize the relation between Family and Happiness Score by using a regression plot.

```
sns.regplot(x='Family',y='Happiness Score',data=happinessData)
```

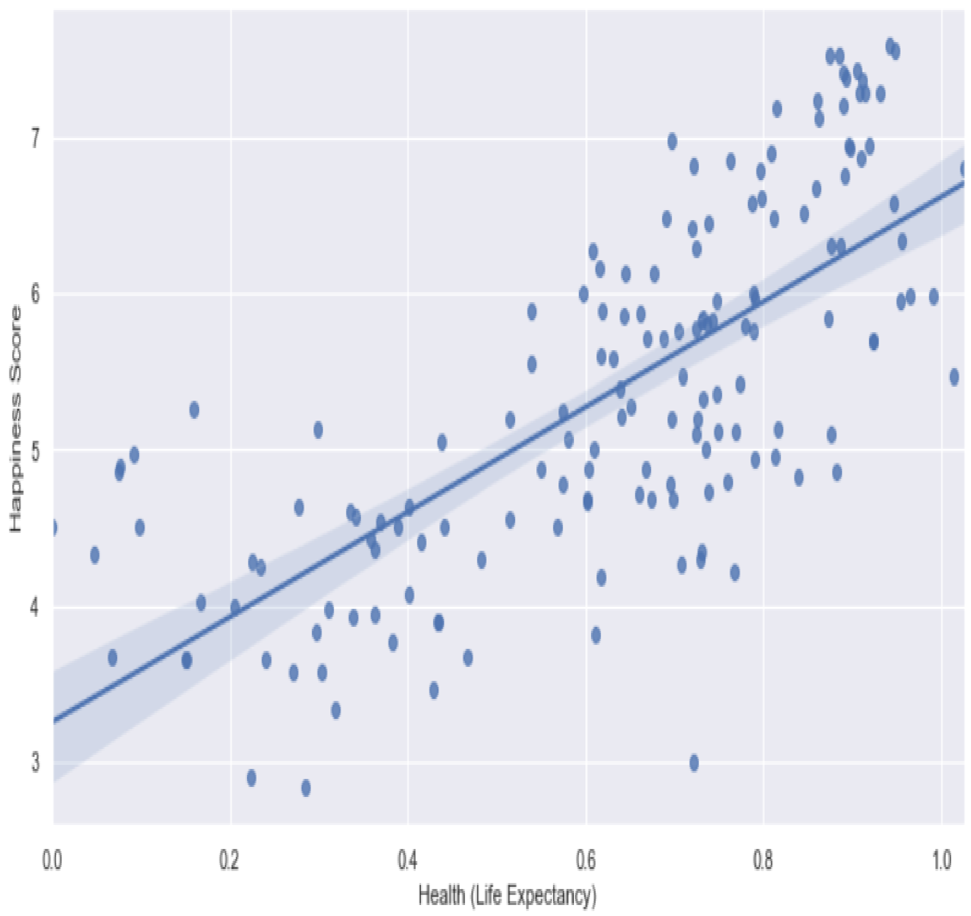
```
<AxesSubplot:xlabel='Family', ylabel='Happiness Score'>
```



Now, we will visualize the relation between Health (Life Expectancy) and Happiness Score by using a regression plot. And as we can see that, as Happiness is dependent on health, i.e. **Good Health is equal to More Happy a person is.**

```
sns.regplot(x='Health (Life Expectancy)',y='Happiness Score',data=happinessData)
```

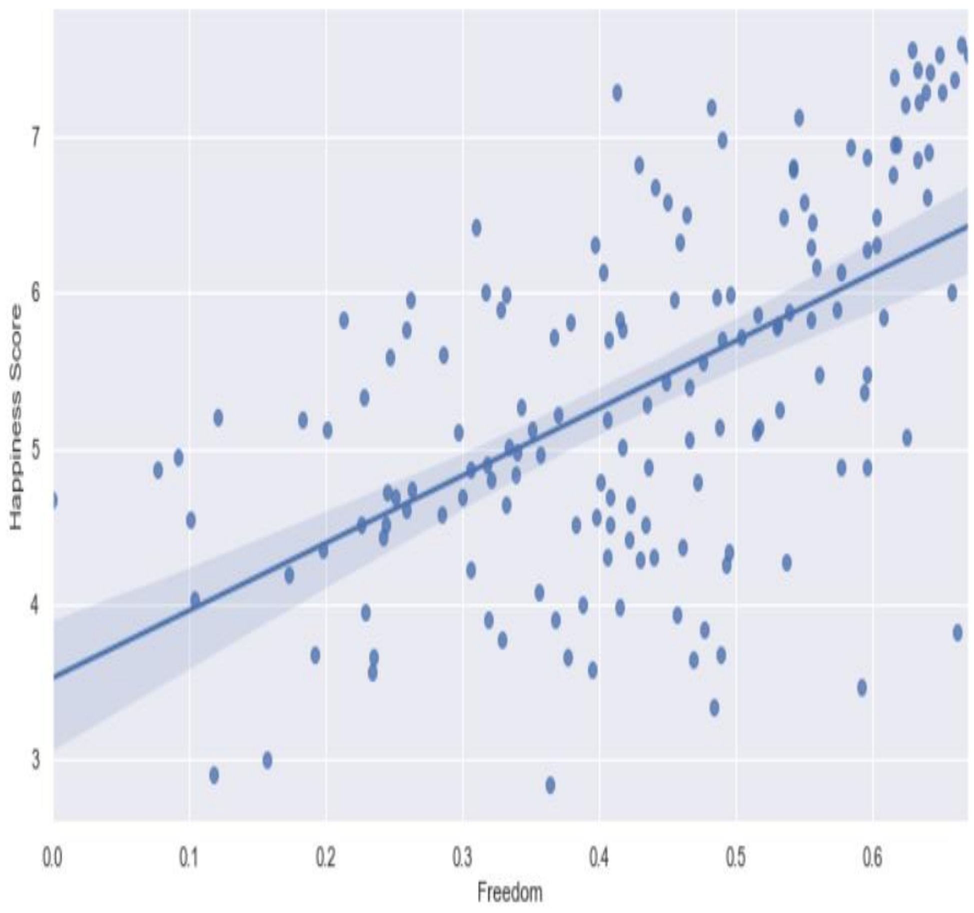
```
<AxesSubplot:xlabel='Health (Life Expectancy)', ylabel='Happiness Score'>
```



Now, we will visualize the relation between Freedom and Happiness Score by using a regression plot. And as we can see that, as the **correlation is less** between these two parameters so the graph is more **scattered** and the **dependency is less between the two.**

```
sns.regplot(x='Freedom',y='Happiness Score',data=happinessData)
```

```
<AxesSubplot:xlabel='Freedom', ylabel='Happiness Score'>
```



I hope we all now have a basic understanding of how to perform Exploratory Data Analysis(EDA).

Hence, the above are the steps that I personally follow for Exploratory Data Analysis, but there are various other plots and commands, which we can use to explore more into the data.

Dataset Used

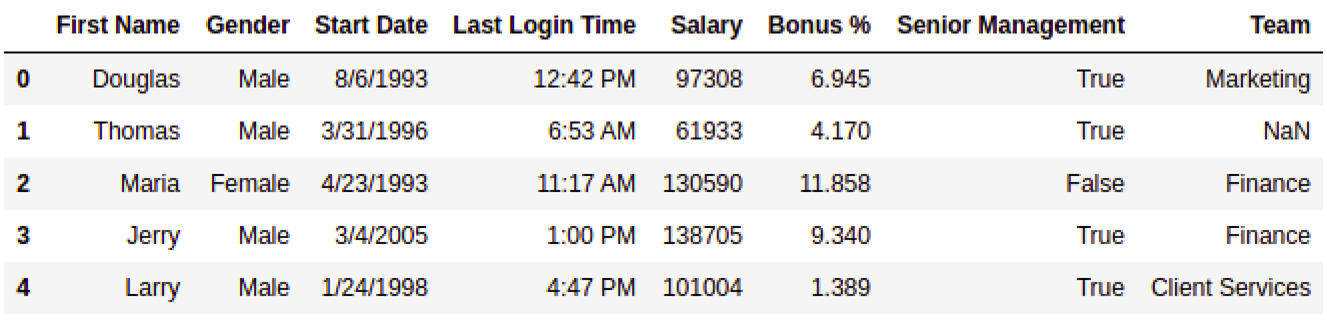
For the simplicity of the article, we will use a single dataset. We will use the employee data for this. It contains 8 columns namely – First Name, Gender, Start Date, Last Login, Salary, Bonus%, Senior Management, and Team.

Dataset Used: Employees.csv
Let's read the dataset using the Pandas module and print the 1st five rows. To print the first five rows we will use the **head()** function.
Example:

- Python3

```
import pandas as pd

import numpy as np

 df = pd.read_csv('employees.csv')

df.head()
```

Output:



	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services

Getting insights about the dataset

Let's see the shape of the data using the shape.

Example:

- Python3

```
df.shape
```

Output:

```
(1000, 8)
```

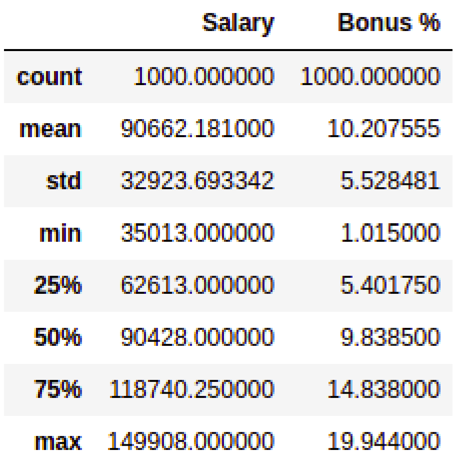
This means that this dataset has 1000 rows and 8 columns.

Let’s get a quick summary of the dataset using the **describe()** method. The describe() function applies basic statistical computations on the dataset like extreme values, count of data points standard deviation, etc. Any missing value or NaN value is automatically skipped. describe() function gives a good picture of the distribution of data.
Example:

- Python3

```
df.describe()
```

Output:



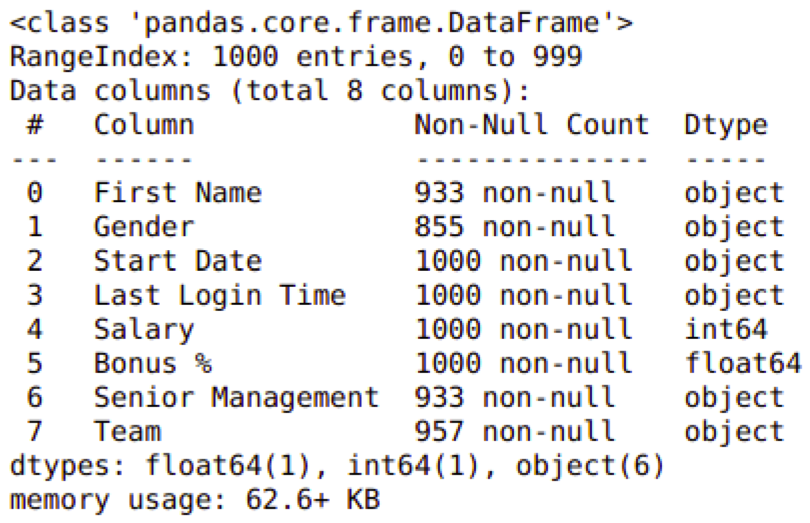
	Salary	Bonus %
count	1000.000000	1000.000000
mean	90662.181000	10.207555
std	32923.693342	5.528481
min	35013.000000	1.015000
25%	62613.000000	5.401750
50%	90428.000000	9.838500
75%	118740.250000	14.838000
max	149908.000000	19.944000

Now, let’s also the columns and their data types. For this, we will use the info() method.

- Python3

```
df.info()
```

Output:



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   First Name         933 non-null    object 
 1   Gender             855 non-null    object 
 2   Start Date         1000 non-null   object 
 3   Last Login Time    1000 non-null   object 
 4   Salary             1000 non-null   int64  
 5   Bonus %            1000 non-null   float64
 6   Senior Management  933 non-null    object 
 7   Team               957 non-null    object 
dtypes: float64(1), int64(1), object(6)
memory usage: 62.6+ KB
```

Till now we have got an idea about the dataset used. Now Let's see if our dataset contains any missing value or not.

Handling Missing Values

You all must be wondering why a dataset will contain any missing value. It can occur when no information is provided for one or more items or for a whole unit. For Example, Suppose different users being surveyed may choose not to share their income, some users may choose not to share the address in this way many datasets went missing. Missing Data is a very big problem in real-life scenarios. Missing Data can also refer to as NA(Not Available) values in pandas. There are several useful functions for detecting, removing, and replacing null values in Pandas DataFrame :

- isnull()
- notnull()
- dropna()
- fillna()
- replace()
- interpolate()

Now let's check if there are any missing values in our dataset or not.

Example:

- Python3

```
df.isnull().sum()
```

Output:

```
First Name           67
Gender              145
Start Date            0
Last Login Time       0
Salary                0
Bonus %               0
Senior Management    67
Team                 43
dtype: int64
```

We can see that every column has a different amount of missing values. Like Gender as 145 missing values and salary has 0. Now for handling these missing values there can be several cases like dropping the rows containing NaN or replacing NaN with either mean, median, mode, or some other value.

Now, let’s try to fill the missing values of gender with the string “No Gender”.

Example:

- Python3

```
df["Gender"].fillna("No Gender", inplace = True)

df.isnull().sum()
```

Output:

```
First Name           67
Gender                0
Start Date            0
Last Login Time       0
Salary                0
Bonus %               0
Senior Management    67
Team                 43
dtype: int64
```

We can see that now there is no null value for the gender column. Now, Let’s fill the senior management with the mode value.

Example:

- Python3

```
mode = df['Senior Management'].mode().values[0]
```

df['Senior Management']= df['Senior Management'].replace(np.nan, mode)

df.isnull().sum()

Output:

```
First Name          67
Gender               0
Start Date           0
Last Login Time      0
Salary               0
Bonus %              0
Senior Management    0
Team                43
dtype: int64
```

Now for the first name and team, we cannot fill the missing values with arbitrary data, so, let’s drop all the rows containing these missing values.

Example:

- Python3

df = df.dropna(axis = 0, how ='any')

print(df.isnull().sum())

df.shape

Output:

```
First Name          0
Gender              0
Start Date          0
Last Login Time     0
Salary              0
Bonus %             0
Senior Management   0
Team                0
dtype: int64

(899, 8)
```

We can see that our dataset is now free of all the missing values and after dropping the data the number of also reduced from 1000 to 899.

Note: For more information, refer Working with Missing Data in Pandas.

After removing the missing data let’s visualize our data.

Data visualization

Data Visualization is the process of analyzing data in the form of graphs or maps, making it a lot easier to understand the trends or patterns in the data. There are various types of visualizations –

- **Univariate analysis:** This type of data consists of only one variable. The analysis of univariate data is thus the simplest form of analysis since the information deals with only one quantity that changes. It does not deal with causes or relationships and the main purpose of the analysis is to describe the data and find patterns that exist within it.
- **Bi-Variate analysis:** This type of data involves two different variables. The analysis of this type of data deals with causes and relationships and the analysis is done to find out the relationship among the two variables.
- **Multi-Variate analysis:** When the data involves three or more variables, it is categorized under multivariate.

Let’s see some commonly used graphs –

Note: *We will use Matplotlib and Seaborn library for the data visualization. If you want to know about these modules refer to the articles –*

- *Matplotlib Tutorial*
- *Python Seaborn Tutorial*

Histogram

It can be used for both uni and bivariate analysis.

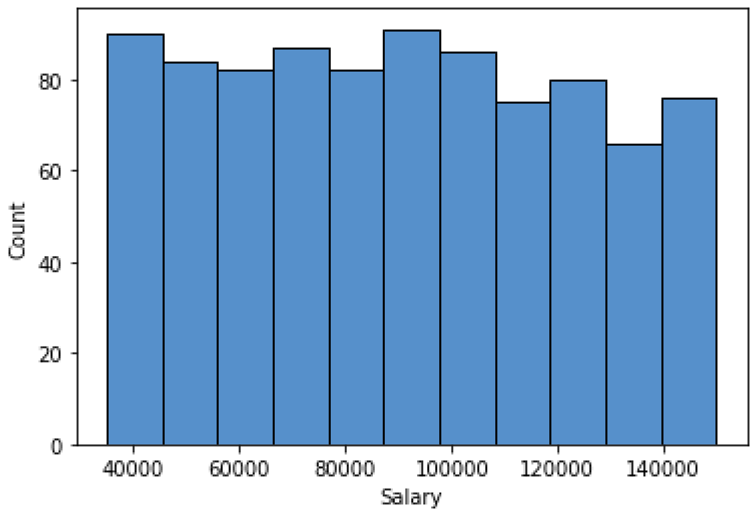
Example:

- Python3

```
# importing packages

import seaborn as sns

import matplotlib.pyplot as plt
```

```
sns.histplot(x='Salary', data=df, )

plt.show()
```

Output:



Boxplot

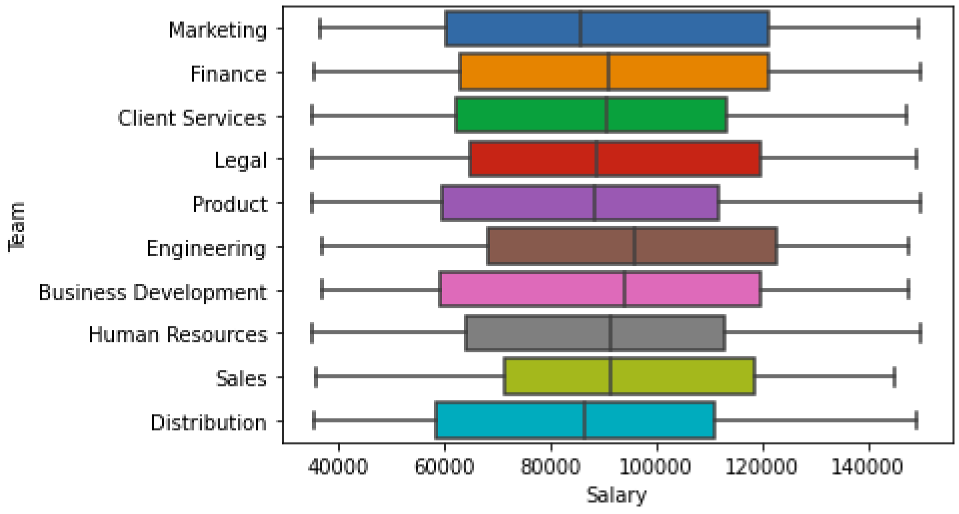
It can also be used for univariate and bivariate analyses.

Example:

- Python3

```
# importing packages

import seaborn as sns

import matplotlib.pyplot as plt

sns.boxplot( x="Salary", y='Team', data=df, )

plt.show()
```

Output:



Scatter Plot

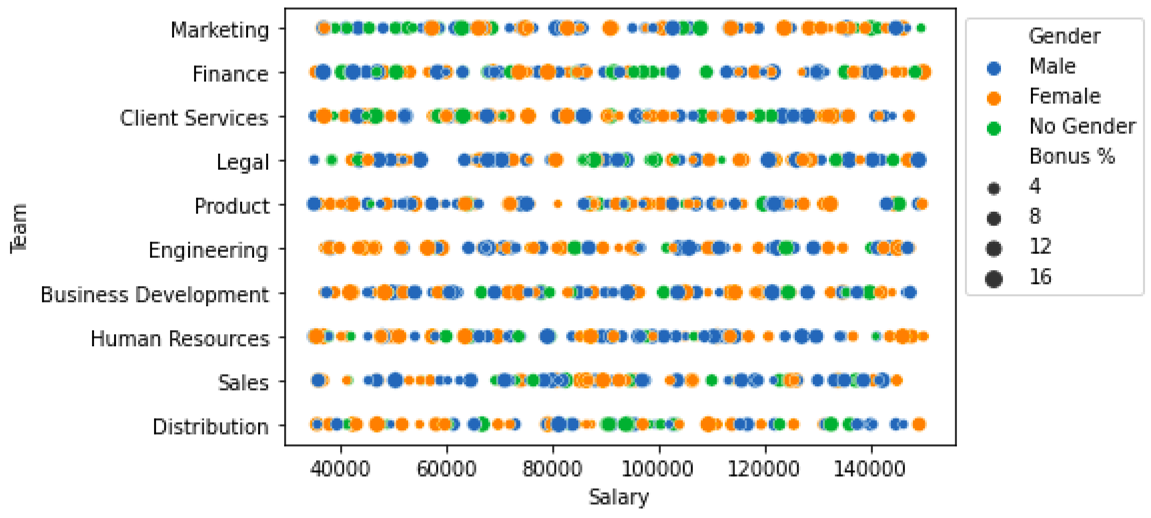
It can be used for bivariate analyses.

Example:

- Python3

```
# importing packages

import seaborn as sns

import matplotlib.pyplot as plt

sns.scatterplot( x="Salary", y='Team', data=df,

                hue='Gender', size='Bonus %')

 # Placing Legend outside the Figure

plt.legend(bbox_to_anchor=(1, 1), loc=2)

plt.show()
```

Output:



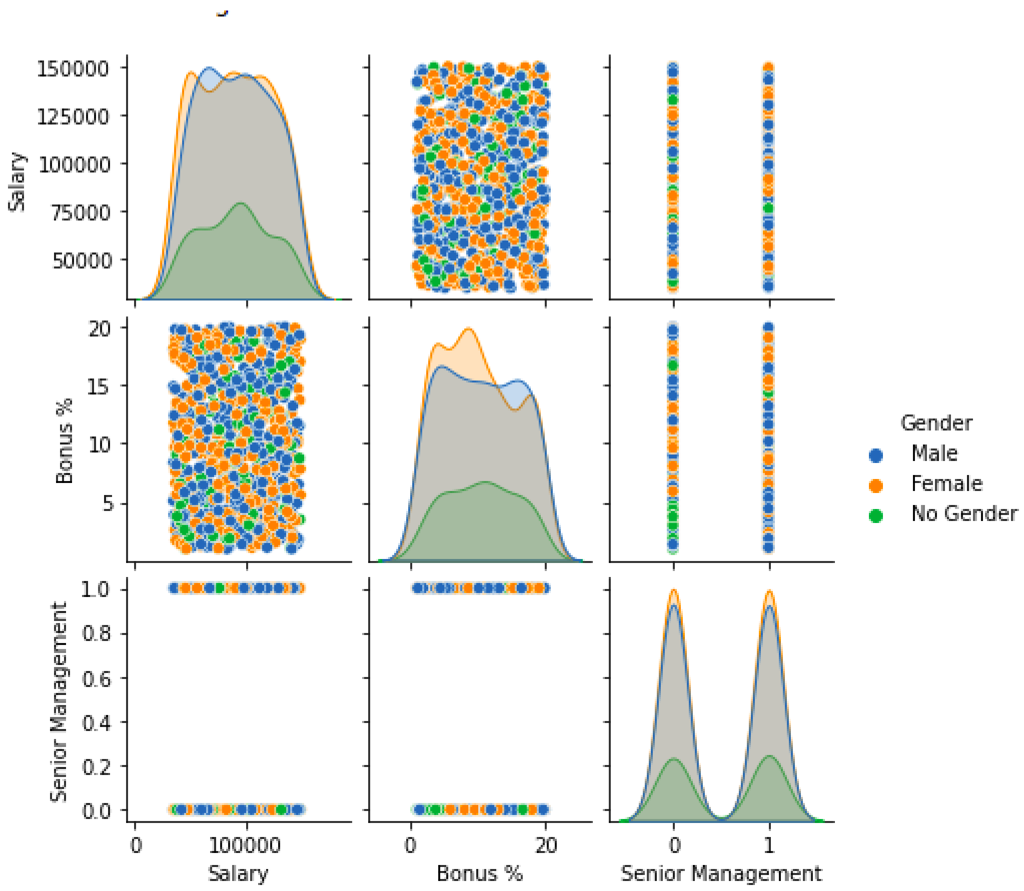
For multivariate analysis, we can the pairplot()method of seaborn module. We can also use it for the multiple pairwise bivariate distributions in a dataset.

Example:

- Python3

```
# importing packages

import seaborn as sns

import matplotlib.pyplot as plt

sns.pairplot(df, hue='Gender', height=2)
```

Output:



Handling Outliers

An Outlier is a data-item/object that deviates significantly from the rest of the (so-called normal)objects. They can be caused by measurement or execution errors. The analysis for outlier detection is referred to as outlier mining. There are many ways to detect the outliers, and the removal process is the data frame same as removing a data item from the panda's dataframe.

Let's consider the iris dataset and let's plot the boxplot for the SepalWidthCm column.

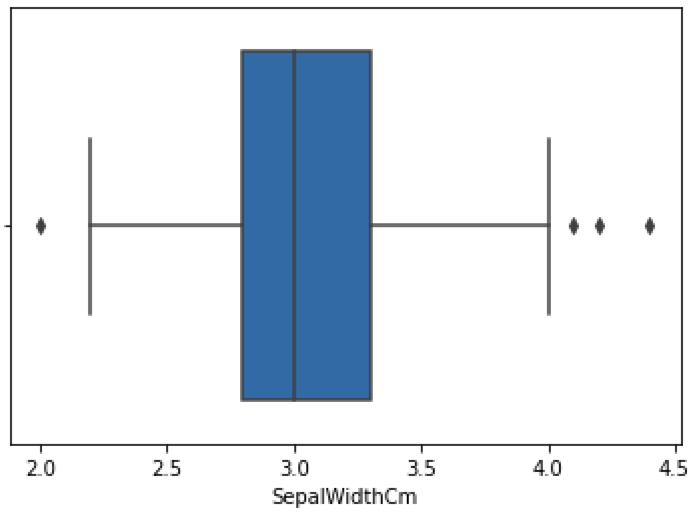
Example:

- Python3

```
# importing packages

import seaborn as sns

import matplotlib.pyplot as plt
```

```
# Load the dataset

df = pd.read_csv('Iris.csv')

sns.boxplot(x='SepalWidthCm', data=df)
```

Output:



In the above graph, the values above 4 and below 2 are acting as outliers.

Removing Outliers

For removing the outlier, one must follow the same process of removing an entry from the dataset using its exact position in the dataset because in all the above methods of detecting the outliers end result is the list of all those data items that satisfy the outlier definition according to the method used.

Example: We will detect the outliers using IQR and then we will remove them. We will also draw the boxplot to see if the outliers are removed or not.

- Python3

```
# Importing

import sklearn

from sklearn.datasets import load_boston
```

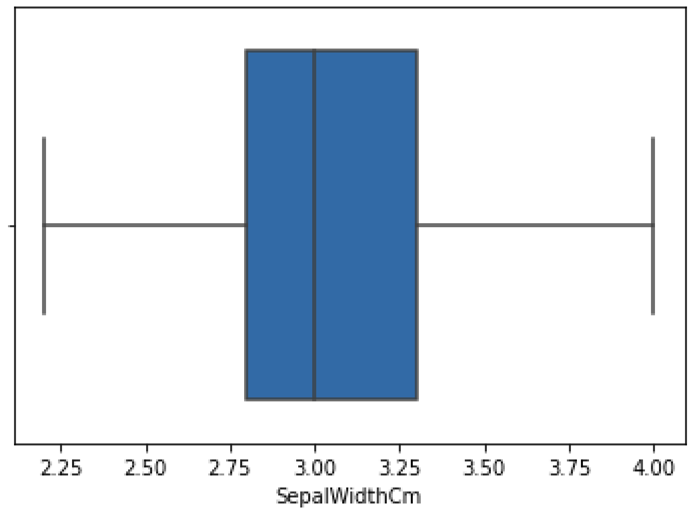
```
import pandas as pd

import seaborn as sns

# Load the dataset

df = pd.read_csv('Iris.csv')

# IQR

Q1 = np.percentile(df['SepalWidthCm'], 25,

                interpolation = 'midpoint')

Q3 = np.percentile(df['SepalWidthCm'], 75,

                interpolation = 'midpoint')

IQR = Q3 - Q1

print("Old Shape: ", df.shape)

# Upper bound

upper = np.where(df['SepalWidthCm'] >= (Q3+1.5*IQR))

# Lower bound

lower = np.where(df['SepalWidthCm'] <= (Q1-1.5*IQR))

# Removing the Outliers

df.drop(upper[0], inplace = True)

df.drop(lower[0], inplace = True)
```

print("New Shape: ", df.shape)

sns.boxplot(x='SepalWidthCm', data=df)

Output:

```
Old Shape:  (150, 6)
New Shape:  (146, 6)

<AxesSubplot:xlabel='SepalWidthCm'>
```



Exploratory Data Analysis or EDA is used to take insights from the data. Data Scientists and Analysts try to find different patterns, relations, and anomalies in the data using some statistical graphs and other visualization techniques. Following things are part of EDA :

1. Get maximum insights from a data set
2. Uncover underlying structure
3. Extract important variables from the dataset
4. Detect outliers and anomalies(if any)
5. Test underlying assumptions
6. Determine the optimal factor settings

Why EDA is important?

The main purpose of EDA is to detect any errors, outliers as well as to understand different patterns in the data. It allows Analysts to understand the data better before making any assumptions. The outcomes of EDA helps businesses to know their customers, expand their business and take decisions accordingly.

How to perform EDA?

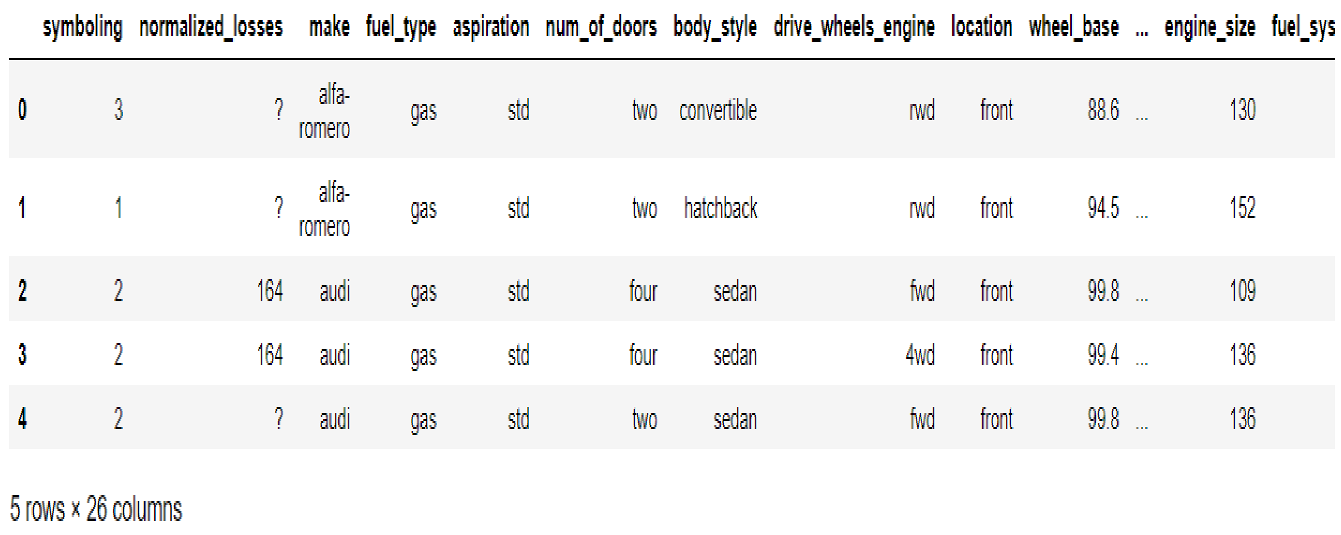
To understand EDA better let us take an example. We will be using Automobile Dataset for analysis.

1. Import libraries and load dataset

Python Code:

We can see that the dataset has 26 attributes and column names are missing. We can also observe that there are '?' at some places which means our data has missing value also. We will fill in column names first.

```
cols=['symboling','normalized_losses','make','fuel_type','aspiration','num_of_doors'
,'body_style','drive_wheels_engine','location','wheel_base','length','width','height','c
urb_weight','engine_type','num_of_cylinders','engine_size','fuel_system','bore','stro
ke','compression_ratio','horsepower','peak_rpm','city_mpg','highway_mpg','price']
auto.columns=cols
auto.head()
```



	symboling	normalized_losses	make	fuel_type	aspiration	num_of_doors	body_style	drive_wheels_engine	location	wheel_base	...	engine_size	fuel_sys
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	
1	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	
2	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	
3	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	
4	2	?	audi	gas	std	two	sedan	fwd	front	99.8	...	136	

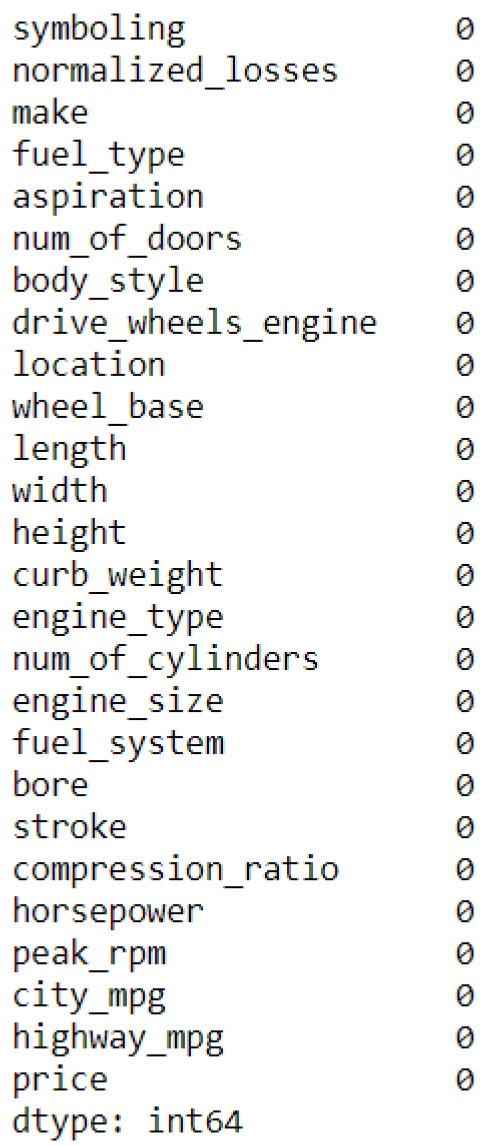
5 rows × 26 columns

Image Source: Jupyter Notebook Screenshot

We got our column names. The price column is our target variable.

2.Check for missing values

auto.isnull().sum()

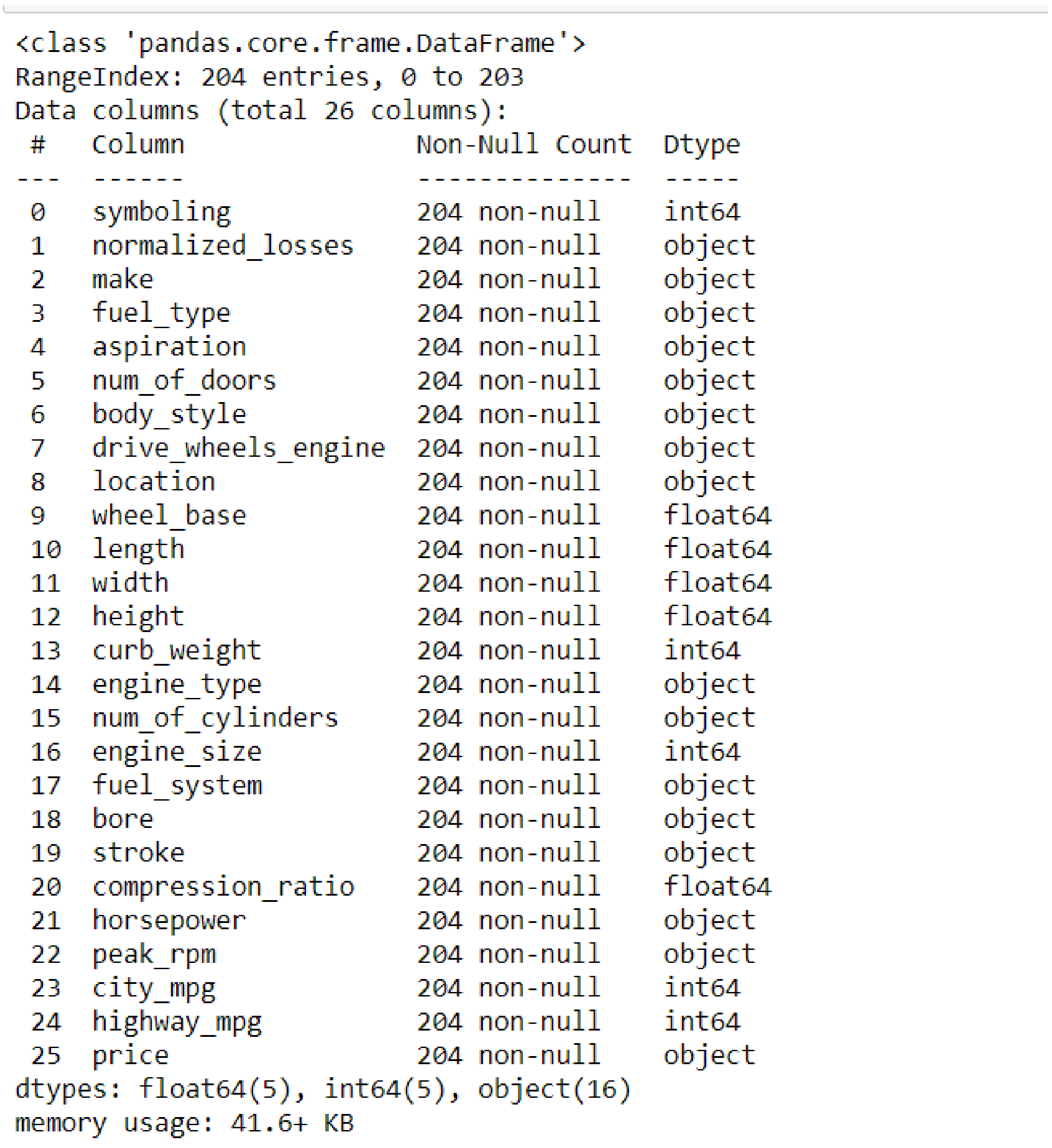


```
symboling             0
normalized_losses     0
make                  0
fuel_type             0
aspiration            0
num_of_doors          0
body_style            0
drive_wheels_engine   0
location              0
wheel_base            0
length                0
width                 0
height                0
curb_weight           0
engine_type           0
num_of_cylinders      0
engine_size           0
fuel_system           0
bore                  0
stroke                0
compression_ratio     0
horsepower            0
peak_rpm              0
city_mpg              0
highway_mpg           0
price                 0
dtype: int64
```

Image Source: Jupyter Notebook Screenshot

It is showing that we don't have any null values in our dataset but we have observed earlier that there were '?' symbols in the dataset, which means that these symbols are in the form of an object. Let us now check the data types of each attribute.

auto.info()

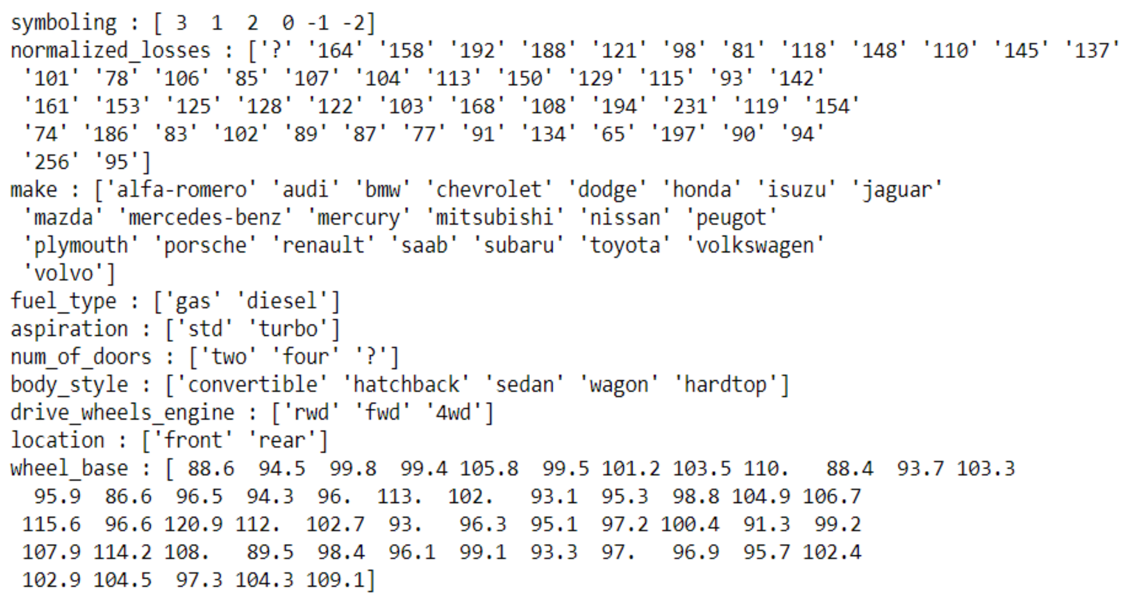


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 204 entries, 0 to 203
Data columns (total 26 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   symboling           204 non-null    int64
 1   normalized_losses   204 non-null    object
 2   make                204 non-null    object
 3   fuel_type           204 non-null    object
 4   aspiration          204 non-null    object
 5   num_of_doors        204 non-null    object
 6   body_style          204 non-null    object
 7   drive_wheels_engine 204 non-null    object
 8   location            204 non-null    object
 9   wheel_base          204 non-null    float64
 10  length              204 non-null    float64
 11  width               204 non-null    float64
 12  height              204 non-null    float64
 13  curb_weight         204 non-null    int64
 14  engine_type         204 non-null    object
 15  num_of_cylinders    204 non-null    object
 16  engine_size         204 non-null    int64
 17  fuel_system         204 non-null    object
 18  bore                204 non-null    object
 19  stroke              204 non-null    object
 20  compression_ratio   204 non-null    float64
 21  horsepower          204 non-null    object
 22  peak_rpm            204 non-null    object
 23  city_mpg            204 non-null    int64
 24  highway_mpg         204 non-null    int64
 25  price               204 non-null    object
dtypes: float64(5), int64(5), object(16)
memory usage: 41.6+ KB
```

Image Source: Jupyter Notebook Screenshot

We can observe that those columns that have symbols are in object form as well as some columns should be of an integer type but are of an object type. Now let us detect which columns have symbols and if there are any other symbols too.

```
#Checking for wrong entries like symbols -,?,#,*,etc.
for col in auto.columns:
    print('{} : {}'.format(col,auto[col].unique()))
```



```
symboling : [ 3  1  2  0 -1 -2]
normalized_losses : ['?' '164' '158' '192' '188' '121' '98' '81' '118' '148' '110' '145' '137'
 '101' '78' '106' '85' '107' '104' '113' '150' '129' '115' '93' '142'
 '161' '153' '125' '128' '122' '103' '168' '108' '194' '231' '119' '154'
 '74' '186' '83' '102' '89' '87' '77' '91' '134' '65' '197' '90' '94'
 '256' '95']
make : ['alfa-romero' 'audi' 'bmw' 'chevrolet' 'dodge' 'honda' 'isuzu' 'jaguar'
 'mazda' 'mercedes-benz' 'mercury' 'mitsubishi' 'nissan' 'peugot'
 'plymouth' 'porsche' 'renault' 'saab' 'subaru' 'toyota' 'volkswagen'
 'volvo']
fuel_type : ['gas' 'diesel']
aspiration : ['std' 'turbo']
num_of_doors : ['two' 'four' '?']
body_style : ['convertible' 'hatchback' 'sedan' 'wagon' 'hardtop']
drive_wheels_engine : ['rwd' 'fwd' '4wd']
location : ['front' 'rear']
wheel_base : [ 88.6  94.5  99.8  99.4 105.8  99.5 101.2 103.5 110.   88.4  93.7 103.3
  95.9  86.6  96.5  94.3  96.  113.  102.   93.1  95.3  98.8 104.9 106.7
 115.6  96.6 120.9 112.  102.7  93.   96.3  95.1  97.2 100.4  91.3  99.2
 107.9 114.2 108.   89.5  98.4  96.1  99.1  93.3  97.   96.9  95.7 102.4
 102.9 104.5  97.3 104.3 109.1]
```

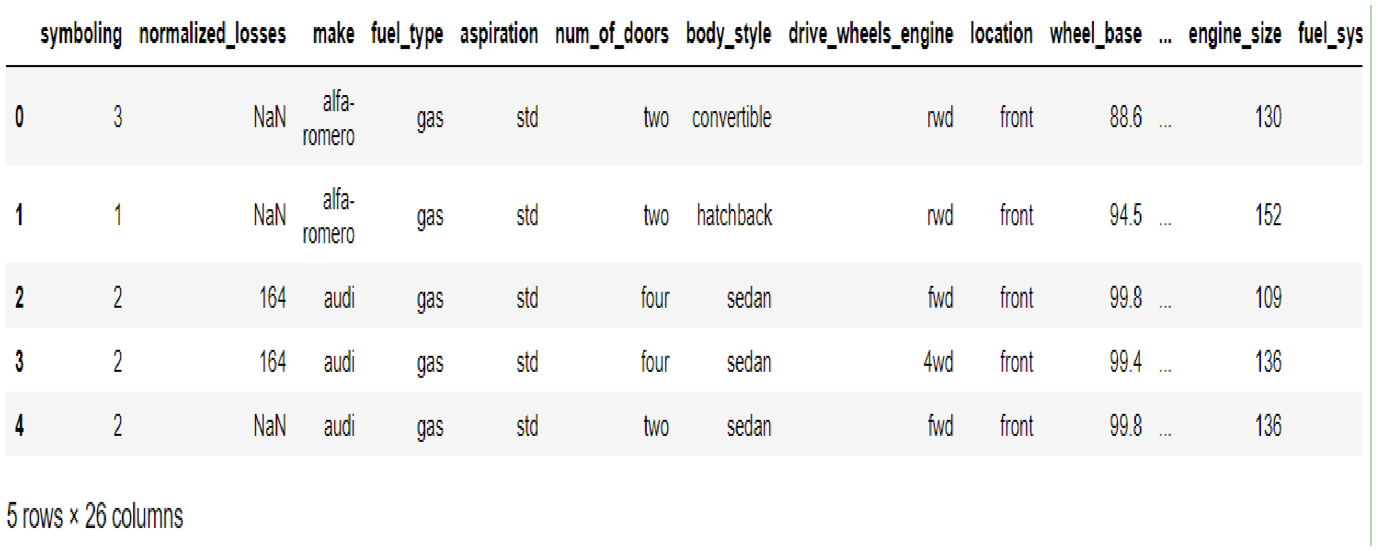
Image Source: Jupyter Notebook Screenshot

There are null values in our dataset in form of '?' only but pandas are not reading them so we will replace them into *np.nan* form.

for col in auto.columns:

    auto[col].replace({'?':np.nan},inplace=True)

auto.head()

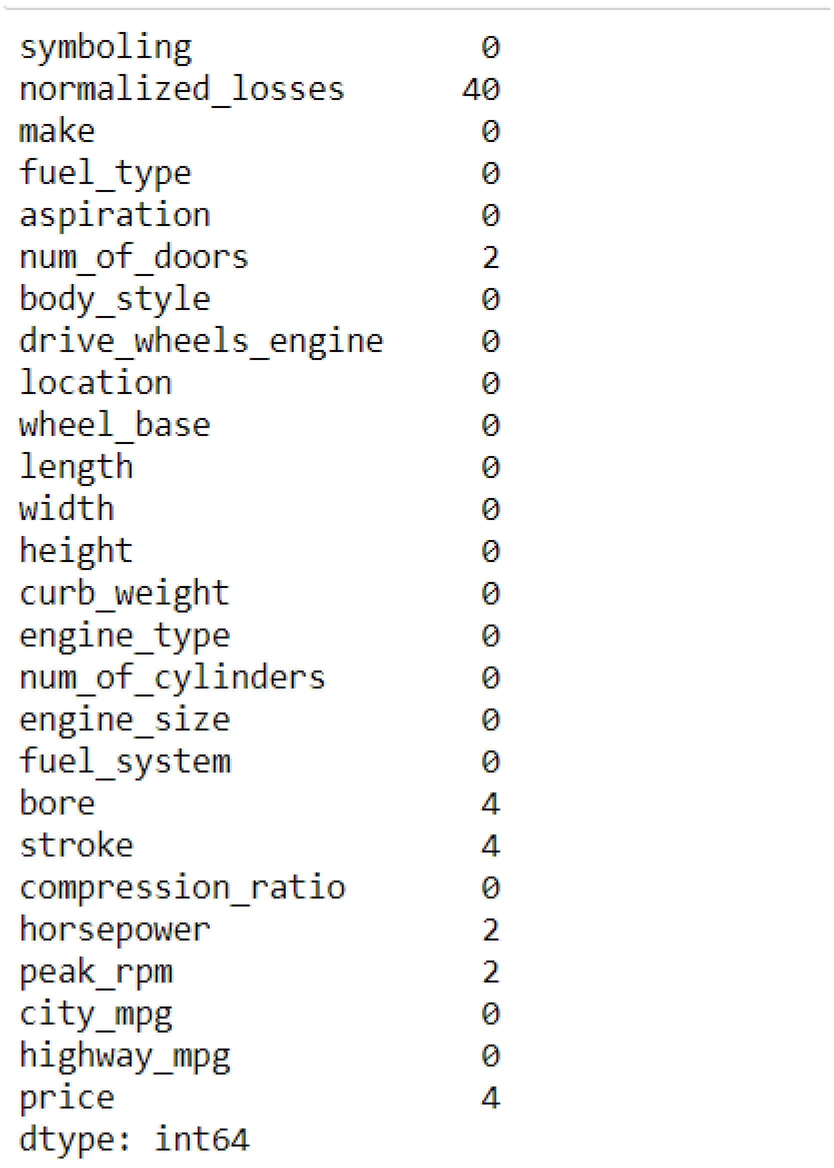


	symboling	normalized_losses	make	fuel_type	aspiration	num_of_doors	body_style	drive_wheels_engine	location	wheel_base	...	engine_size	fuel_sys
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	
1	1	NaN	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	
2	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	
3	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	
4	2	NaN	audi	gas	std	two	sedan	fwd	front	99.8	...	136	

5 rows × 26 columns

Now we can observe that the '?' symbols have been converted into *NaN* form. Let us check for missing values again.

auto.isnull().sum()



```
symboling              0
normalized_losses     40
make                   0
fuel_type              0
aspiration             0
num_of_doors           2
body_style             0
drive_wheels_engine    0
location               0
wheel_base             0
length                 0
width                  0
height                 0
curb_weight            0
engine_type            0
num_of_cylinders       0
engine_size            0
fuel_system            0
bore                   4
stroke                 4
compression_ratio      0
horsepower             2
peak_rpm               2
city_mpg               0
highway_mpg            0
price                  4
dtype: int64
```

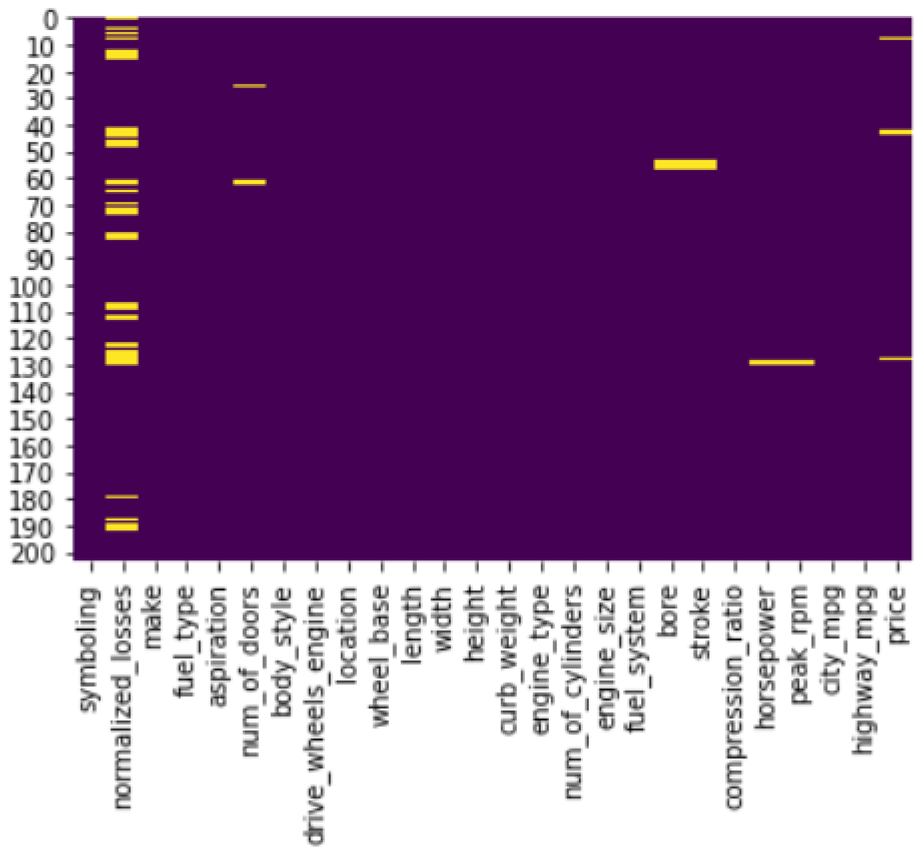
We can observe that now there are missing values in some columns.

3. Visualizing the missing values

With the help of heatmap, we can see the amount of data that is missing from the attribute. With this, we can make decisions whether to drop these missing values or to replace them. Usually dropping the missing values is not advisable but sometimes it may be helpful too.

sns.heatmap(auto.isnull(),cbar=False,cmap='viridis')

```
<AxesSubplot:>
```

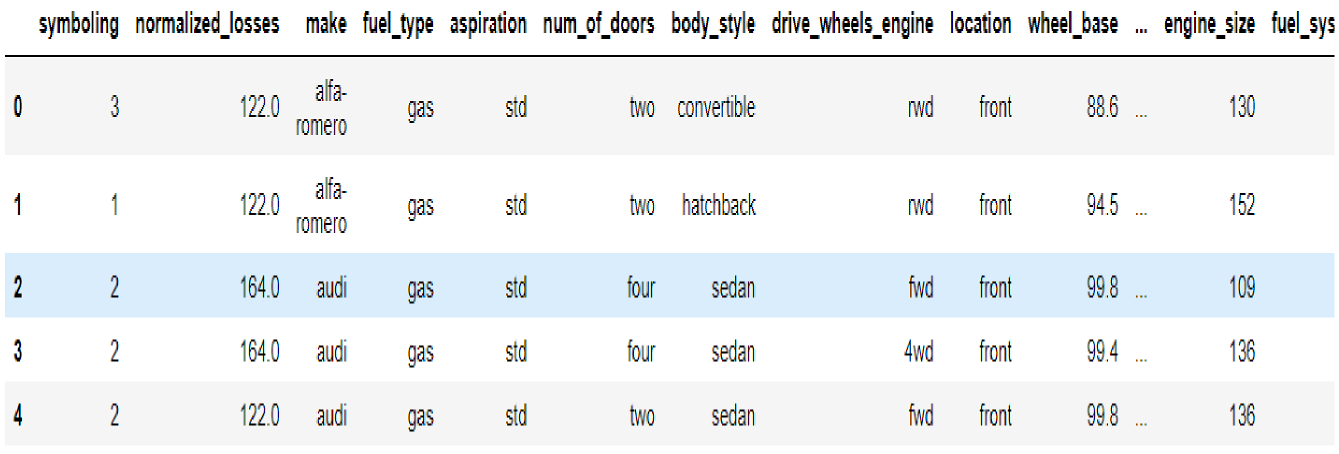


Now observe that there are many missing values in *normalized_losses* while other columns have fewer missing values. We can't drop the *normalized_losses* column as it may be important for our prediction.

4.Replacing the missing values

We will be replacing these missing values with mean because the number of missing values is less(we can use median too).

```
num_col = ['normalized_losses', 'bore',  'stroke', 'horsepower', 'peak_rpm','price']
for col in num_col:
    auto[col]=pd.to_numeric(auto[col])
    auto[col].fillna(auto[col].mean(), inplace=True)
auto.head()
```



	symboling	normalized_losses	make	fuel_type	aspiration	num_of_doors	body_style	drive_wheels_engine	location	wheel_base	...	engine_size	fuel_sys
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	
1	1	122.0	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	
2	2	164.0	audi	gas	std	four	sedan	fwd	front	99.8	...	109	
3	2	164.0	audi	gas	std	four	sedan	4wd	front	99.4	...	136	
4	2	122.0	audi	gas	std	two	sedan	fwd	front	99.8	...	136	

5 rows × 26 columns

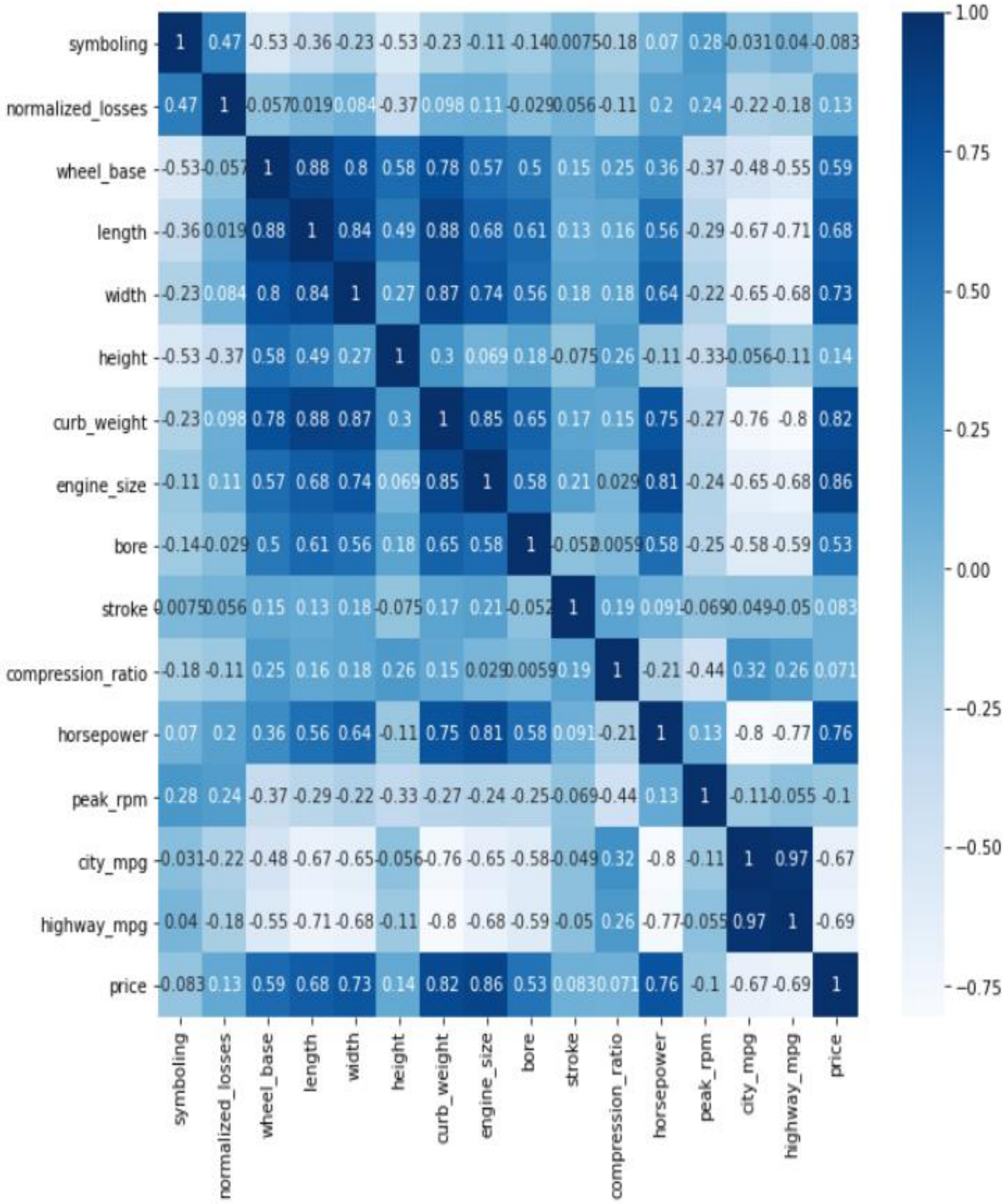
We can observe that now our missing values are replaced with mean.

5. Asking Analytical Questions and Visualizations

This is the most important step in EDA. This step will decide how much can you think as an Analyst. This step varies from person to person in terms of their questioning ability. Try to ask questions related to independent variables and the target variable. For example – how fuel_type will affect the price of the car?

Before this let us check the correlation between different variables, this will give us a roadmap on how to proceed further.

```
plt.figure(figsize=(10,10))
sns.heatmap(auto.corr(),cbar=True,annot=True,cmap='Blues')
```



Positive Correlation

- *Price – wheel_base, length, width, curb_weight, engine_size, bore, horsepower*
- *wheelbase – length, width, height, curb_weight, engine_size, price*
- *horsepower – length, width, curb_weight, engine_size, bore, price*
- *Highway mpg – city mpg*

Negative Correlation

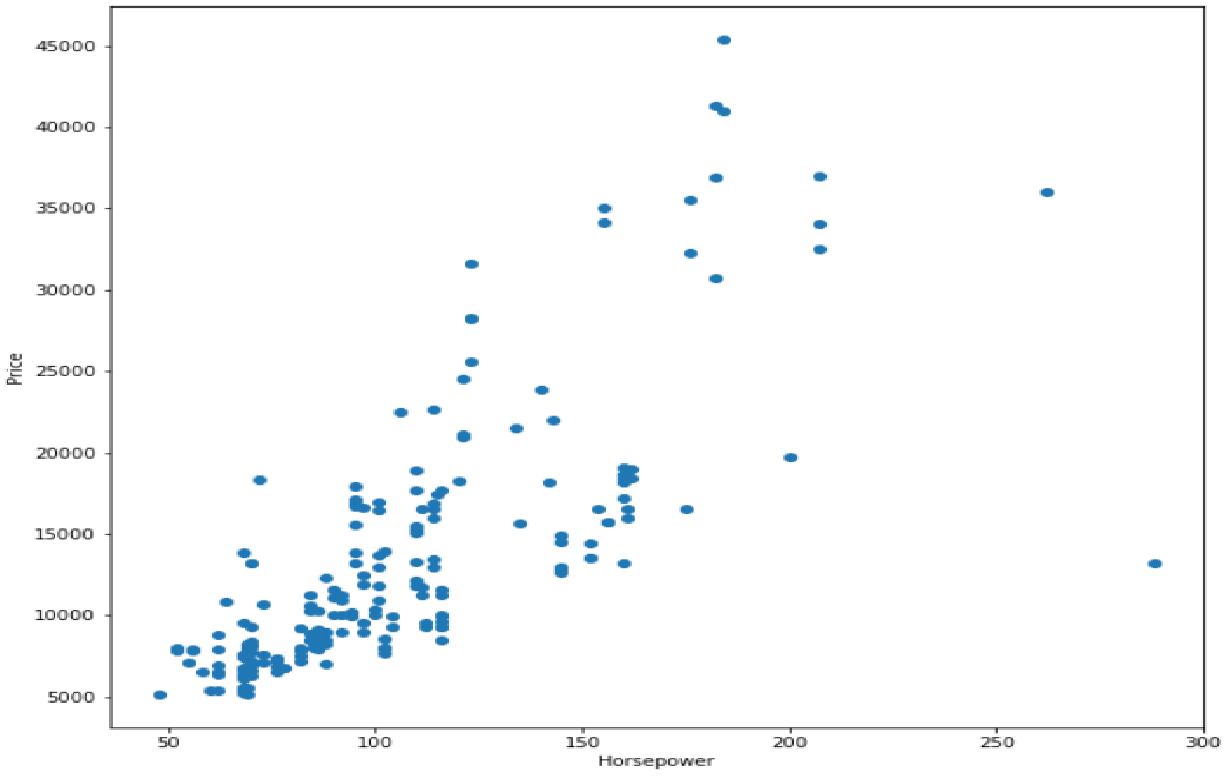
- *Price – highway_mpg, city_mpg*
- *highway_mpg – wheel base, length, width, curb_weight, engine_size, bore, horsepower, price*
- *city – wheel base, length, width, curb_weight, engine_size, bore, horsepower, price*

This heatmap has given us great insights into the data.

Now let us apply domain knowledge and ask the questions which will affect the price of the automobile.

1. How does the horsepower affect the price?

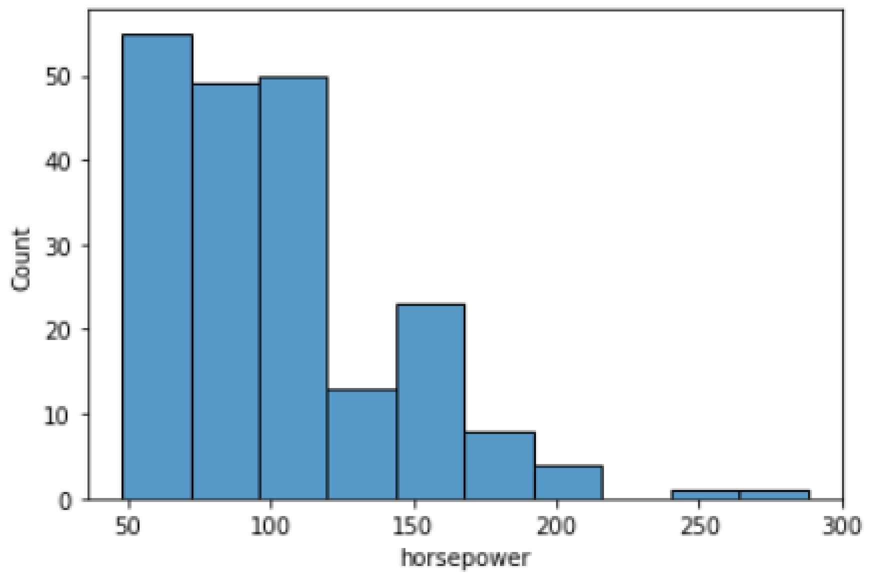
```
plt.figure(figsize=(10,10))
plt.scatter(x='horsepower',y='price',data=auto)
plt.xlabel('Horsepower')
plt.ylabel('Price')
```



We can see that most of the horsepower value lies between 50-150 has price mostly between 5000-25000, there are outliers also(between 200-300).

Let’s see a count between 50-100 i.e univariate analysis of horsepower.

```
sns.histplot(auto.horsepower,bins=10)
```



The average count between 50-100 is 50 and it is positively skewed.

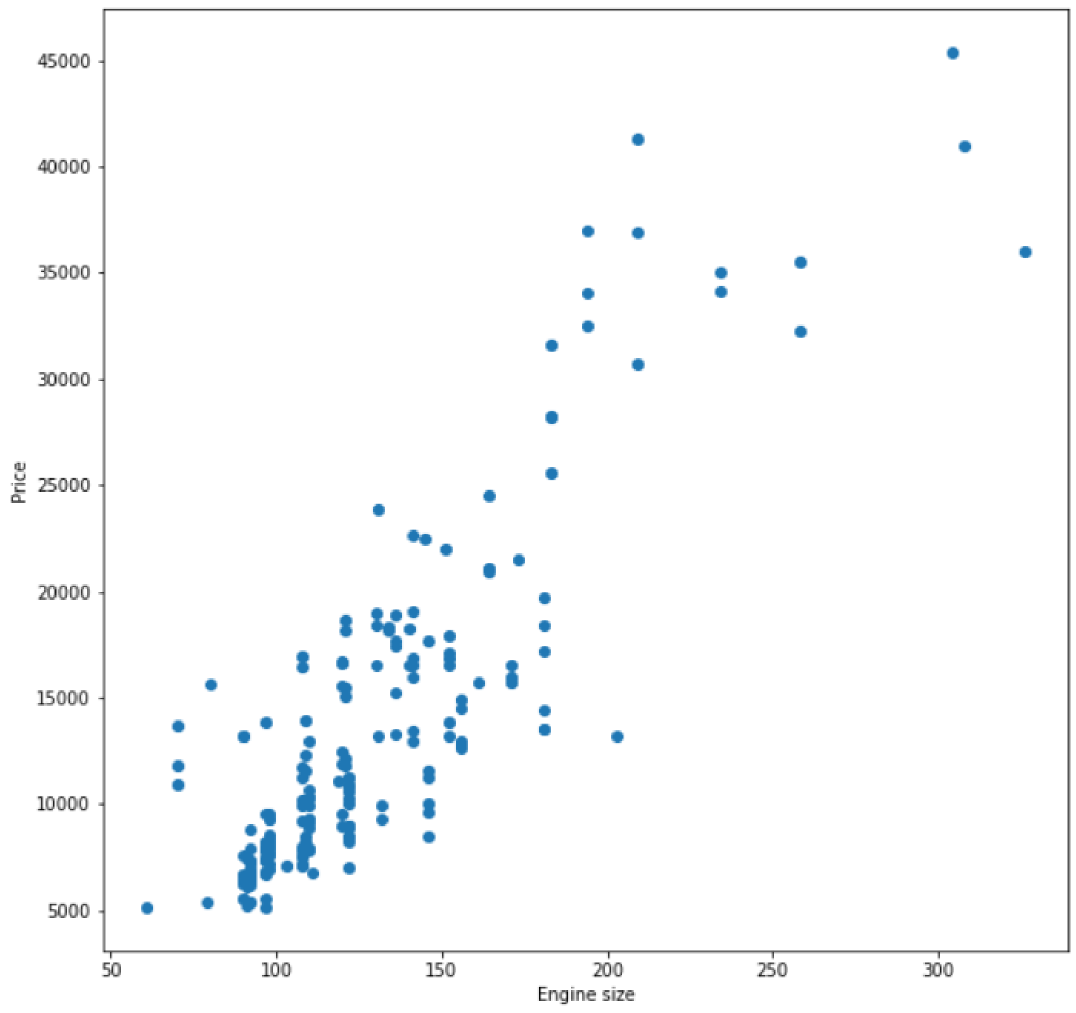
2. What is the relation between engine_size and price?

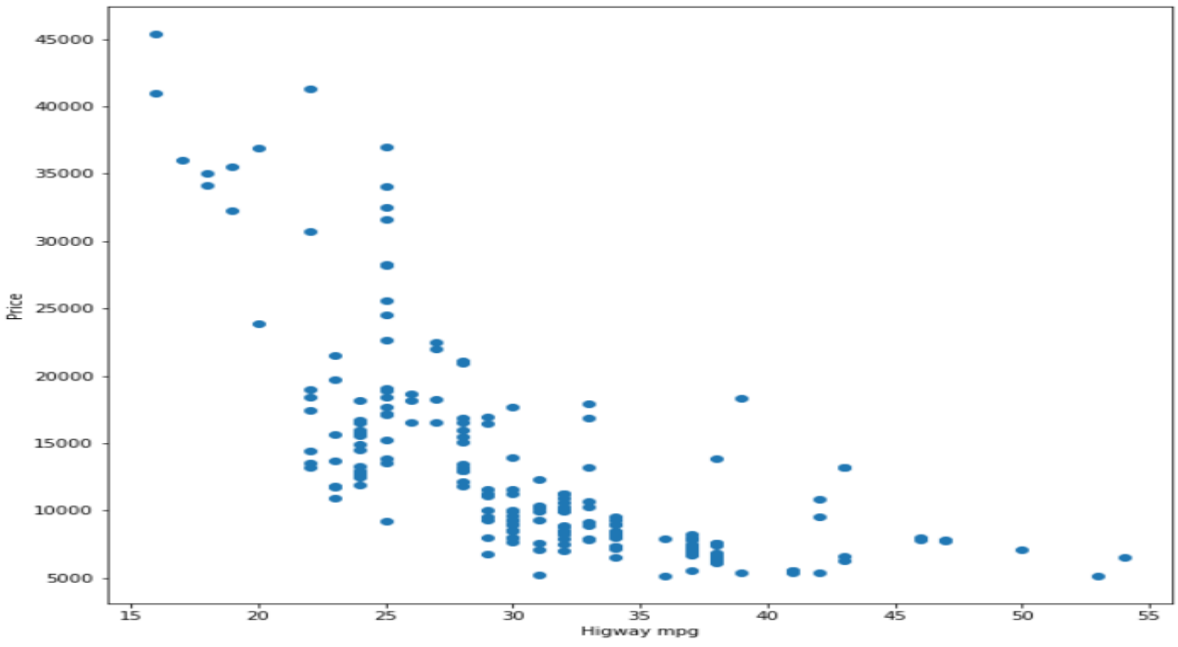
```
plt.figure(figsize=(10,10))
plt.scatter(x='engine_size',y='price',data=auto)
plt.xlabel('Engine size')
plt.ylabel('Price')
```

We can observe that the pattern is similar to horsepower vs price.

3. How does highway_mpg affects price?

```
plt.figure(figsize=(10,10))
plt.scatter(x='highway_mpg',y='price',data=auto)
plt.xlabel('Higway mpg')
plt.ylabel('Price')
```





We can see price decreases with an increase in higway_mpg.

Let us check the number of doors.

#Unique values in num_of_doors

auto.num_of_doors.value_counts()

```
four    114
two      88
Name: num_of_doors, dtype: int64
```

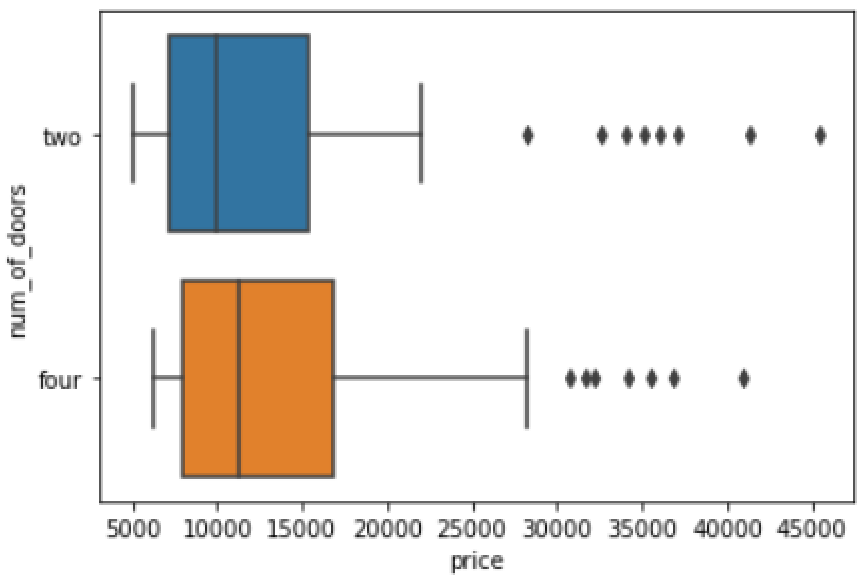
Image Source: Jupyter Notebook Screenshot

4. Relation between no. of doors and price

We will use a boxplot for this analysis.

sns.boxplot(x='price',y='num_of_doors',data=auto)

```
<AxesSubplot:xlabel='price', ylabel='num_of_doors'>
```



With this boxplot, we can conclude that the average price of a vehicle with two doors is 10000, and the average price of a vehicle with four doors is 12000.

With this plot, we have gained enough insights from data and our data is ready to build a model.

Exploratory data analysis is the first and most important phase in any data analysis. EDA is a method or philosophy that aims to uncover the most important and frequently overlooked patterns in a data set. We examine the data and attempt to formulate a hypothesis. Statisticians use it to get a bird eyes view of data and try to make sense of it.

In this EDA series we will cover the following points:

1. Data sourcing
2. Data cleaning
3. Univariate analysis
4. Bi-variate/Multivariate analysis

Data Sourcing

Data mainly arrives from a variety of sources, and your first responsibility as an analyst is to collect it. Because if you know your data batter then it's easy to use for further analysis. So, collect the required data.

Public Data

For research reasons, a substantial amount of data collected by the government or other public entities is made available. These data sets are referred to as public data because they do not require specific permission to view. On the internet, public data is available on a variety of platforms. A large number of data sets are available for direct analysis, while others must be manually extracted and translated into an analysis-ready format.

Private Data

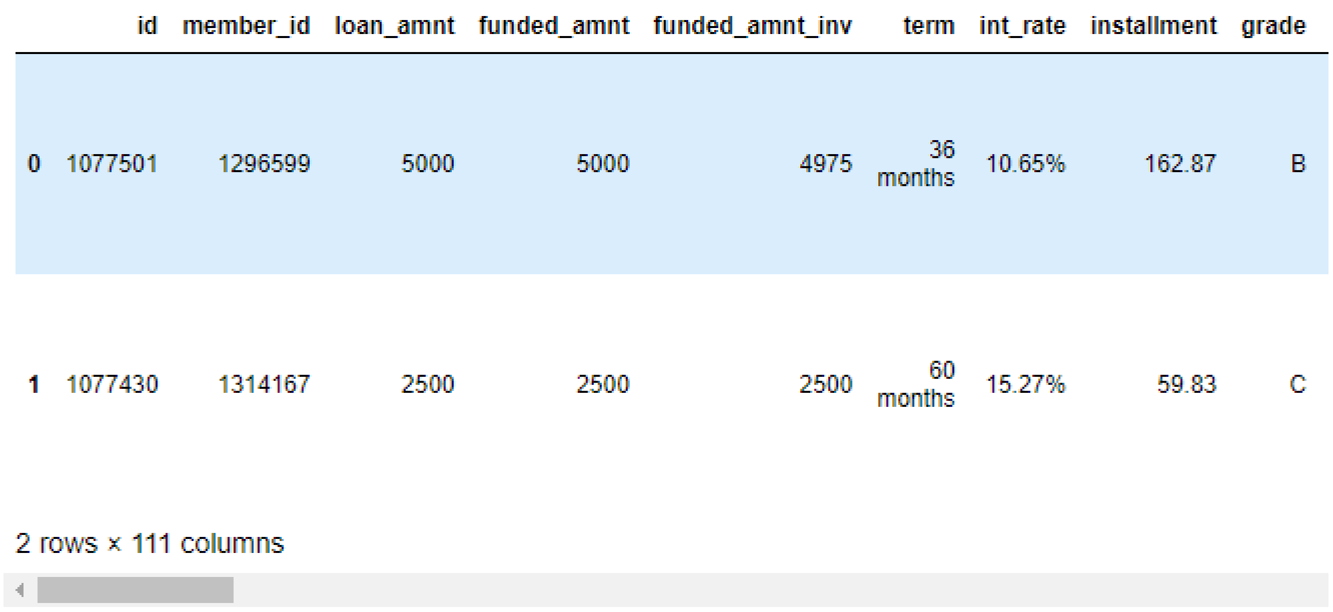
Data that is sensitive to organizations and consequently not available in the public domain is referred to as private data. Banking, telecommunications, retail, and the media are just a few of the major commercial sectors that significantly rely on data to make judgments. Many businesses are looking to use data analytics to help them make important choices. As businesses become more customer-centric, they may use data insights to improve the customer experience while also streamlining their daily operations.

Implementation:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from scipy import stats
import re
import seaborn as sns
print('seaborn versiont:',sns.__version__)
import os
import warnings
warnings.filterwarnings('ignore') # if there are any warning due to version
mismatch, it will be ignored
```

So now we ready to load the data set in Data Frame. here we go.,

```
loan = pd.read_csv('loan.csv',dtype='object')
loan.head(2)
```



	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade
0	1077501	1296599	5000	5000	4975	36 months	10.65%	162.87	B
1	1077430	1314167	2500	2500	2500	60 months	15.27%	59.83	C

2 rows × 111 columns

In the above output, you can see we have many rows (but we print only two rows) and 111 columns.

Now we check Columns List & NA value counts more than 30%

NA_col = loan.isnull().sum()

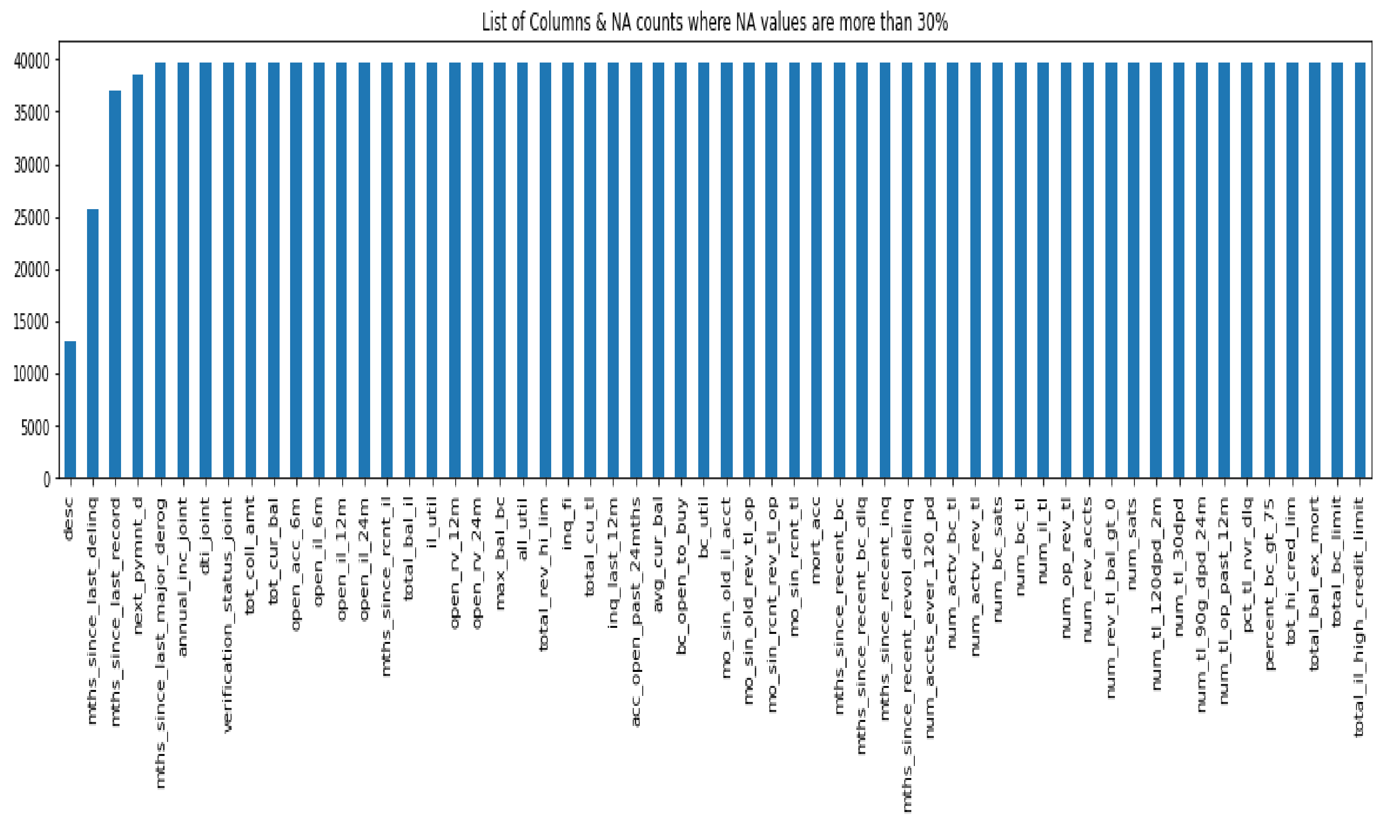
NA_col = NA_col[NA_col.values >(0.2*len(loan))]

plt.figure(figsize=(20,4))

NA_col.plot(kind='bar')

plt.title('**Columns List & NA value counts more than 30%**')

plt.show()



NA_col[NA_col.values >(0.2*len(loan))]

```
desc                              12940
mths_since_last_delinq            25682
mths_since_last_record            36931
next_pymnt_d                      38577
mths_since_last_major_derog       39717
annual_inc_joint                  39717
dti_joint                         39717
verification_status_joint         39717
tot_coll_amt                      39717
tot_cur_bal                       39717
open_acc_6m                       39717
open_il_6m                        39717
open_il_12m                       39717
open_il_24m                       39717
mths_since_rcnt_il                39717
total_bal_il                      39717
il_util                           39717
open_rv_12m                       39717
open_rv_24m                       39717
max_bal_bc                        39717
                                   ...
```

loan.isnull().sum()/len(loan)*100

```
id                               0.0000
member_id                        0.0000
loan_amnt                        0.0000
funded_amnt                      0.0000
funded_amnt_inv                  0.0000
                                 ...
tax_liens                        0.0982
tot_hi_cred_lim                100.0000
total_bal_ex_mort              100.0000
total_bc_limit                 100.0000
total_il_high_credit_limit     100.0000
Length: 111, dtype: float64
```

loan.isnull().sum()

```
id                                0
member_id                         0
loan_amnt                         0
funded_amnt                       0
funded_amnt_inv                   0
                               ...
tax_liens                        39
tot_hi_cred_lim               39717
total_bal_ex_mort             39717
total_bc_limit                39717
total_il_high_credit_limit    39717
Length: 111, dtype: int64
```

<u>Insights Note:</u> Here we clearly see that from the above plot we have 20+ columns in the dataset where all the values are NA.

Because the dataset has 887379 rows and 74 columns, it will be quite tough to go through each column one by one and discover the NA or missing values. So, let’s look for any columns that have more than a specific percentage of missing values, say 30%. We’ll get rid of such columns because it’s impossible to impute missing values for them.

Data Cleaning

When it comes to data, there are many different sorts of quality issues, which is why data cleansing is one of the most time-consuming aspects of data analysis.

Formatting issues (e.g., rows and columns merged), missing data, duplicated rows, spelling discrepancies, and so on could all be present. These difficulties could make data analysis difficult, resulting in inaccuracies or inappropriate results. As a result, these issues must be addressed before data can be analyzed. Data cleansing is frequently done in an unplanned, difficult-to-define manner. ‘single structured process’.

In the next steps, we’ll look at data cleaning:

1. Checklist for Row Corrections and Columns Corrections
2. Checklist for Missing Values Corrections
3. Checklist for Standardizing Values Corrections
4. Checklist for Invalid Values Corrections
5. Checklist for Filter data Corrections

1). Checklist for Row Corrections and Columns Corrections

- **Delete any incorrect rows:** Rows in the header and footer
- **Extra rows should be deleted:** column number, indicators, blank rows, and the page number.
- **If necessary, merge columns to create unique identifiers:** E.g. Combine the state and city into a single full address.
- **For extra data, split the columns:** Split the address into two parts so that the state and city can be examined independently.
- **Name the columns:** If column names are lacking, add them.
- **Consistently rename columns:** Columns with abbreviations and encoding
- **Delete the following columns:** Remove any columns that are no longer needed.
- **Align mismatched columns:** The columns in the dataset may have shifted.

2). Checklist for Missing Values Corrections

- **Set values as missing values:** Identify values that suggest missing data but aren't recognized as such by the software, such as blank strings, "NA," "XX," "666," and so on.
- **Exaggerating is harmful while adding is good:** You should strive to collect as much information as possible from credible external sources, but if you can't, it's best to leave missing data alone rather than embellishing the available rows/columns.
- **Delete rows and columns:** If the number of missing values is insignificant, rows can be erased without affecting the analysis. If the number of missing values is considerable, columns may be eliminated.
- **Fill in partial missing values using your best judgment:** time zone, century, and so on. These are plainly identifiable values.

3). Checklist for Standardizing Values Corrections

- Convert lbs to kgs, miles/hr to km/hr, and so on to ensure that all observations under a variable have a common and consistent unit.
- **If necessary, scale the values:** Ascertain that the observations under a variable are all on the same scale. For better data display, standardize precision, e.g. 5.5211341 kgs to 5.52 kgs.
- **Remove outliers:** High and low numbers that have a disproportionate impact on your analysis' results should be removed.

4). Checklist for Invalid Values Corrections

– Invalid values might appear in numerous forms in a data set.

- Some of the values may be actually invalid; for example, a string “tr8ml” in a variable containing mobile numbers would be meaningless and should be eliminated.
- A height of 12 feet, for example, would be an invalid number in a collection of children’s heights.

– Some inaccurate values, on the other hand, can be fixed.

- For example, a numeric value with a string data type could be changed back to its original numeric type.
- Issues may emerge as a result of a programming language such as Python or R misinterpreting a file’s encoding, resulting in the display of trash characters where valid characters should be. This can be fixed by defining the encoding appropriately or converting the data set to the correct format before importing it.

5). Checklist for Filter data Corrections

- **Data duplication:** Remove identical rows and rows with partly identical columns.
- **Rows to filter:** To retrieve only the rows relevant to the analysis, filter by segment and date period.
- **Columns to filter:** Columns that are significant to the analysis should be chosen.
- **Data in aggregate:** Organize by required keys, then combine the rest.

***** Some points on how to deal with invalid values *****

Tips 1: *Unicode should be encoded correctly:*

- Change the encoding to CP1251 instead of UTF-8 if the data is being read as trash characters.

Tips 2: *Convert data types that aren’t correct:*

- For the convenience of analysis, change the wrong data types to the proper data types.
- For example, if numeric values are saved as strings, frequent data type modifications include string to number: “14,500” to “14500”; string to date: “2021-JUN” to “2021/06”; number to a string: “PIN Code 220-001” to “220001”; and so on.

Tips 3: *Values that are out of range should be corrected as follows:*

- If some of the values are outside the reasonable range, such as temperature less than -263° C, you'll need to make the necessary adjustments. A closer inspection will reveal whether the value can be corrected or if it needs to be eliminated.

Tips 4: *correct values that aren't on the list include:*

- Values that don't belong in a list should be removed. Strings "E" or "F" are invalid values in a data set comprising blood types of individuals, for example, and can be eliminated.

Tips 5: *Correct any errors in the structure:*

- Values that do not adhere to a predefined framework can be eliminated. A pin code of 12 digits, for example, would be an invalid value in a data collection including pin codes of Indian cities and would need to be eliminated. A 12-digit phone number, for example, would be an incorrect value.

Tips 6: *Internal rules must be verified:*

- If there are internal standards, such as the delivery date of a product must be after the order date, they must be correct and consistent.

>>> **Find Check List excel workbook on my GitHub Data_Cleaning _Check_List.xlsx**

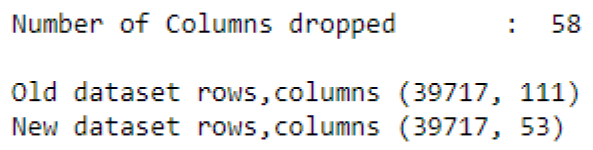
<u>Implementation:</u>

Data Cleaning function for handling nulls and remove Nulls

```
def removeNulls(dataframe, axis =1, percent=0.3):
    df = dataframe.copy()
    ishape = df.shape
    if axis == 0:
        rownames = df.transpose().isnull().sum()
        rownames = list(rownames[rownames.values > percent*len(df)].index)
        df.drop(df.index[rownames],inplace=True)
        print("nNumber of Rows droppedt: ",len(rownames))
    else:
        colnames = (df.isnull().sum()/len(df))
        colnames = list(colnames[colnames.values>=percent].index)
        df.drop(labels = colnames,axis =1,inplace=True)
        print("Number of Columns droppedt: ",len(colnames))
    print("nOld dataset rows,columns",ishape,"nNew dataset
rows,columns",df.shape)
    return df
```

1. Remove columns where NA values are more than or equal to 30%

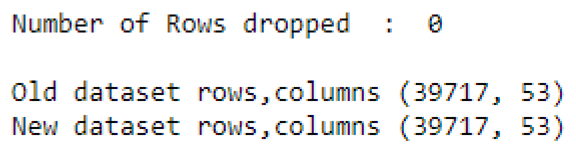
```
loan = removeNulls(loan, axis =1,percent = 0.3)
```



```
Number of Columns dropped       :  58

Old dataset rows,columns (39717, 111)
New dataset rows,columns (39717, 53)
```

2. Remove any rows with NA values greater than or equal to 30%.

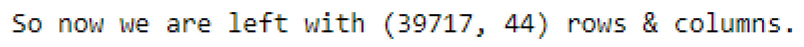
```
loan = removeNulls(loan, axis =1,percent = 0.3)
```



```
Number of Rows dropped  :  0

Old dataset rows,columns (39717, 53)
New dataset rows,columns (39717, 53)
```

3. Remove all columns with only one unique value.

```
unique = loan.nunique()
unique = unique[unique.values == 1]
loan.drop(labels = list(unique.index), axis =1, inplace=True)
print("So now we are left with",loan.shape ,"rows & columns.")
```



```
So now we are left with (39717, 44) rows & columns.
```

4. Employment Term: Replace the value of ‘n/a’ with self-employed.’

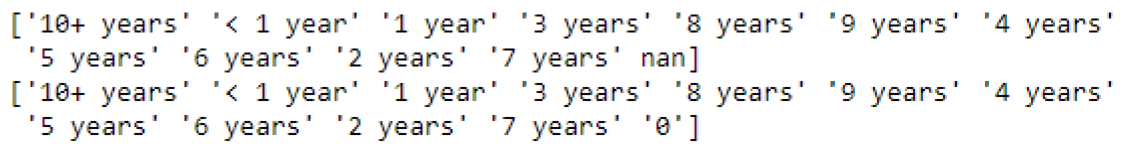
There are some values in emp_term which are **‘n/a’**, we assume that are **‘self-employed’**. Because for ‘self-employed’ applicants, emp_length is ‘Not Applicable’

print(loan.emp_length.unique())

loan.emp_length.fillna('0',inplace=True)

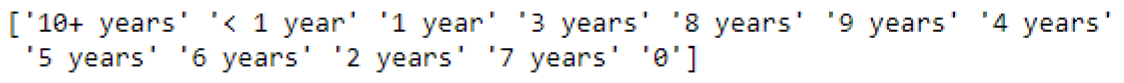
loan.emp_length.replace(['n/a'],'Self-Employed',inplace=True)

print(loan.emp_length.unique())



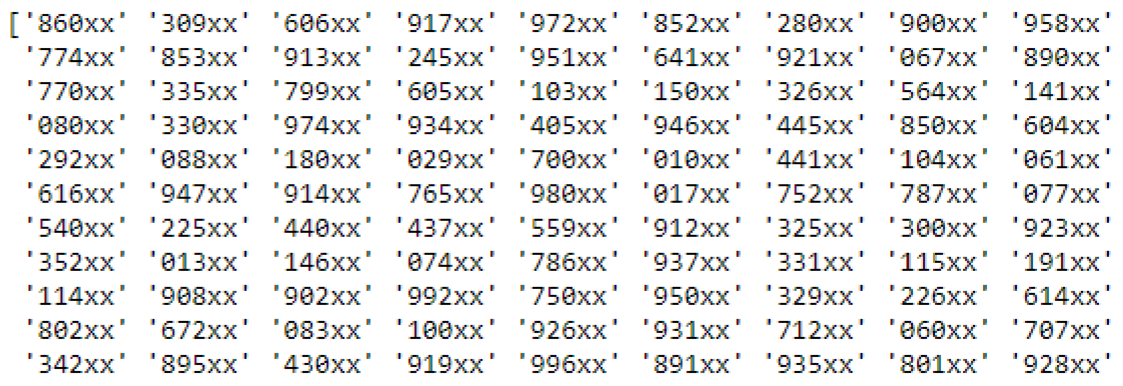
```
['10+ years' '< 1 year' '1 year' '3 years' '8 years' '9 years' '4 years'
 '5 years' '6 years' '2 years' '7 years' nan]
['10+ years' '< 1 year' '1 year' '3 years' '8 years' '9 years' '4 years'
 '5 years' '6 years' '2 years' '7 years' '0']
```

print(loan.emp_length.unique())



```
['10+ years' '< 1 year' '1 year' '3 years' '8 years' '9 years' '4 years'
 '5 years' '6 years' '2 years' '7 years' '0']
```

print(loan.zip_code.unique())



```
['860xx' '309xx' '606xx' '917xx' '972xx' '852xx' '280xx' '900xx' '958xx'
 '774xx' '853xx' '913xx' '245xx' '951xx' '641xx' '921xx' '067xx' '890xx'
 '770xx' '335xx' '799xx' '605xx' '103xx' '150xx' '326xx' '564xx' '141xx'
 '080xx' '330xx' '974xx' '934xx' '405xx' '946xx' '445xx' '850xx' '604xx'
 '292xx' '088xx' '180xx' '029xx' '700xx' '010xx' '441xx' '104xx' '061xx'
 '616xx' '947xx' '914xx' '765xx' '980xx' '017xx' '752xx' '787xx' '077xx'
 '540xx' '225xx' '440xx' '437xx' '559xx' '912xx' '325xx' '300xx' '923xx'
 '352xx' '013xx' '146xx' '074xx' '786xx' '937xx' '331xx' '115xx' '191xx'
 '114xx' '908xx' '902xx' '992xx' '750xx' '950xx' '329xx' '226xx' '614xx'
 '802xx' '672xx' '083xx' '100xx' '926xx' '931xx' '712xx' '060xx' '707xx'
 '342xx' '895xx' '430xx' '919xx' '996xx' '891xx' '935xx' '801xx' '928xx'
```

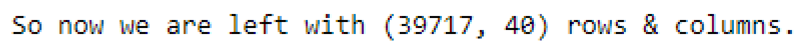
5. Remove any columns that aren’t relevant.

Till now we have removed the columns based on the count & statistics. Now let’s look at each column from a business perspective if that is required or not for our analysis such as Unique IDs, URLs. As the last 2 digits of the zip code are masked ‘xx’, we can remove that.

not_required_columns = ["id","member_id","url","zip_code"]

loan.drop(labels = not_required_columns, axis =1, inplace=True)

```
print("So now we are left with",loan.shape ,"rows & columns.")
```



```
So now we are left with (39717, 40) rows & columns.
```

6. Convert all continuous variables to numeric values.

Cast all continuous variables to numeric so that we can find a correlation between them

```
numeric_columns =
['loan_amnt','funded_amnt','funded_amnt_inv','installment','int_rate','annual_inc','dt
i']
loan[numeric_columns] = loan[numeric_columns].apply(pd.to_numeric)
loan[numeric_columns] = loan[numeric_columns].apply(pd.to_numeric)
# loan rate has diff issue with %

loan.int_rate.unique()

loan['int_rate']=(pd.to_numeric(loan['int_rate'].str.replace(r'%', '')))

loan.int_rate

loan['int_rate_bkp']=loan['int_rate']
```

7. Loan purpose: Remove records with values less than 0.75%.

We will analyze only those categories which contain more than 0.75% of records. Also, we are not aware of what comes under 'Other' we will remove this category as well.

```
(loan.purpose.value_counts()*100)/len(loan)
```

```
debt_consolidation   46.9346
credit_card          12.9164
other                10.0536
home_improvement      7.4930
major_purchase        5.5065
small_business        4.6026
car                   3.9001
wedding               2.3844
medical               1.7448
moving                1.4679
house                 0.9593
vacation              0.9593
educational           0.8183
renewable_energy      0.2593
Name: purpose, dtype: float64
```

loan.purpose.value_counts()

```
debt_consolidation    18641
credit_card            5130
other                  3993
home_improvement       2976
major_purchase         2187
small_business         1828
car                    1549
wedding                 947
medical                 693
moving                  583
house                   381
vacation                381
educational             325
renewable_energy        103
Name: purpose, dtype: int64
```

del_loan_purpose = (loan.purpose.value_counts()*100)/len(loan)

del_loan_purpose = del_loan_purpose[(del_loan_purpose < 0.75) | (del_loan_purpose.index == 'other')]

loan.drop(labels = loan[loan.purpose.isin(del_loan_purpose.index)].index, inplace=True)

print("So now we are left with",loan.shape ,"rows & columns.")

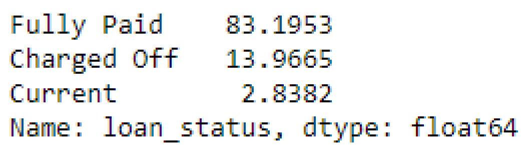
print(loan.purpose.unique())

```
So now we are left with (35621, 41) rows & columns.
['credit_card' 'car' 'small_business' 'wedding' 'debt_consolidation'
 'home_improvement' 'major_purchase' 'medical' 'moving' 'vacation' 'house'
 'educational']
```

8. Loan Status: Remove all records with a value of less than 1.5%.

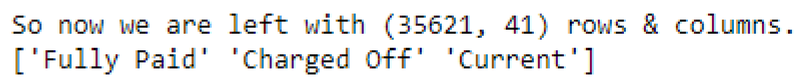
As we can see, Other than [‘Current’, ‘Fully Paid’ & Charged off] other loan_status are not relevant for our analysis.

```
(loan.loan_status.value_counts()*100)/len(loan)
```



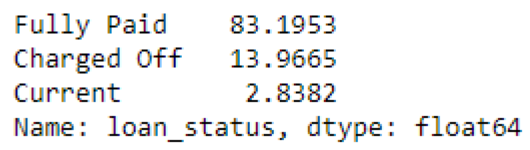
```
Fully Paid     83.1953
Charged Off    13.9665
Current         2.8382
Name: loan_status, dtype: float64
```

```
del_loan_status = (loan.loan_status.value_counts()*100)/len(loan)
del_loan_status = del_loan_status[(del_loan_status < 1.5)]
loan.drop(labels = loan[loan.loan_status.isin(del_loan_status.index)].index,
inplace=True)
print("So now we are left with",loan.shape ,"rows & columns.")
print(loan.loan_status.unique())
```



```
So now we are left with (35621, 41) rows & columns.
['Fully Paid' 'Charged Off' 'Current']
```

```
(loan.loan_status.value_counts()*100)/len(loan)
```



```
Fully Paid     83.1953
Charged Off    13.9665
Current         2.8382
Name: loan_status, dtype: float64
```

Univariate Analysis

This session focuses on analyzing variables one at a time, as the term “uni-variate” implies. Before analyzing numerous variables together, it’s critical to first analyze each one individually.

Continuous Variables

When dealing with continuous variables, it’s important to know the variable’s central tendency and spread. Statistical metrics visualization methods such as Boxplot, Histogram/Distribution Plot, Violin Plot, and others are used to measure these.

Categorical Variables

We’ll utilize a frequency table to study the distribution of categorical variables. Count and Count percent against each category are two metrics that can be used to assess it. As a visualization, a count-plot or a bar chart can be employed.

Implementation:

The univariate function will plot parameter values in graphs.

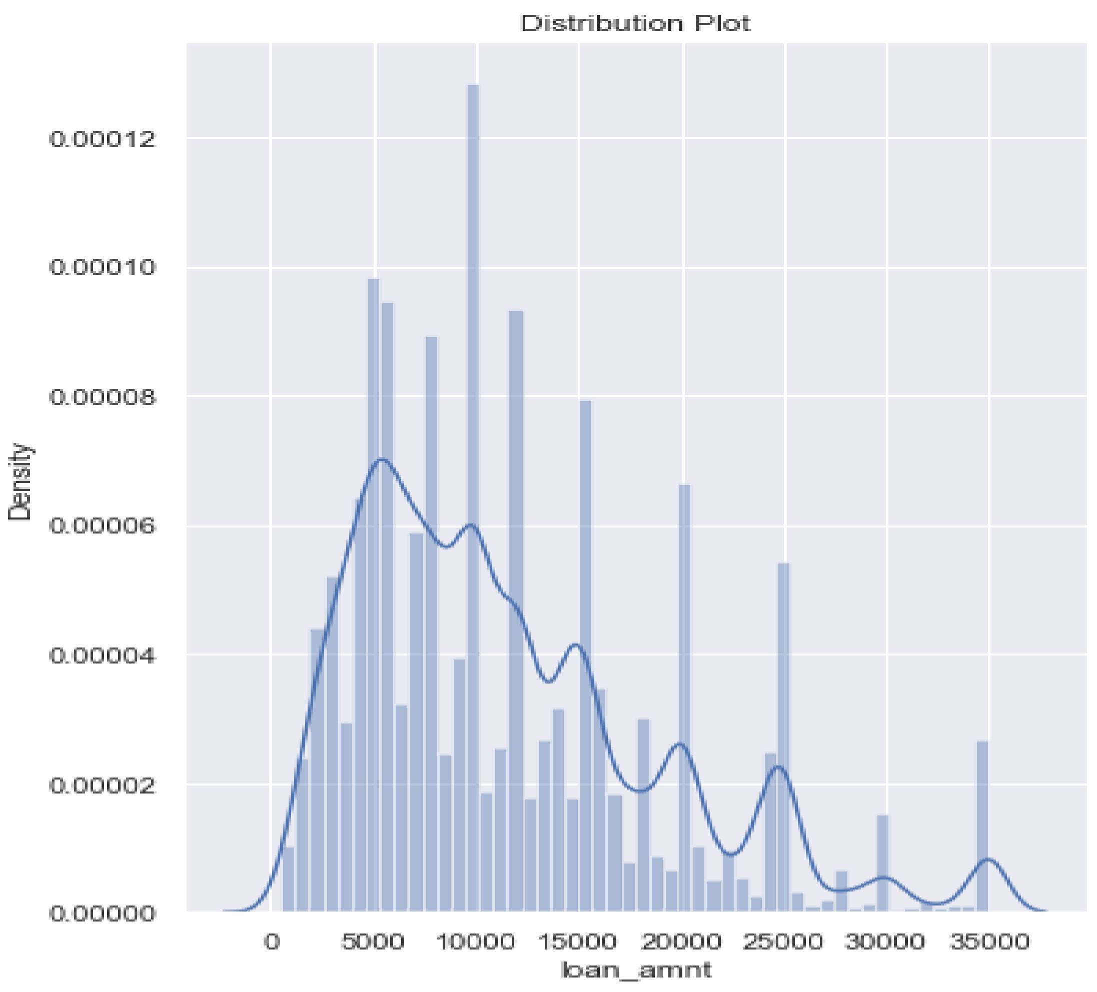
```
def univariate(df,col,vartype,hue =None):
    '''
    Univariate function will plot parameter values in graphs.
    df      : dataframe name
    col     : Column name
    vartype : variable type : continuous or categorical
              Continuous(0)   : Distribution, Violin & Boxplot will be plotted.
              Categorical(1) : Countplot will be plotted.
    hue     : Only applicable in categorical analysis.
    '''
    sns.set(style="darkgrid")
    if vartype == 0:
        fig, ax=plt.subplots(nrows =1,ncols=3,figsize=(20,8))
        ax[0].set_title("Distribution Plot")
        sns.distplot(df[col],ax=ax[0])
        ax[1].set_title("Violin Plot")
        sns.violinplot(data =df, x=col,ax=ax[1], inner="quartile")
        ax[2].set_title("Box Plot")
        sns.boxplot(data =df, x=col,ax=ax[2],orient='v')
    if vartype == 1:
        temp = pd.Series(data = hue)
        fig, ax = plt.subplots()
        width = len(df[col].unique()) + 6 + 4*len(temp.unique())
        fig.set_size_inches(width , 7)
        ax = sns.countplot(data = df, x= col, order=df[col].value_counts().index,hue = 
hue)
        if len(temp.unique()) > 0:
            for p in ax.patches:
                ax.annotate('{:1.1f}%'.format((p.get_height()*100)/float(len(loan))), 
(p.get_x()+0.05, p.get_height()+20))
        else:
            for p in ax.patches:
                ax.annotate(p.get_height(), (p.get_x()+0.32, p.get_height()+20))
        del temp
    else:
        exit
    plt.show()
```

Continuous Variables

Let’s get some insights from loan data

1). Loan Amount

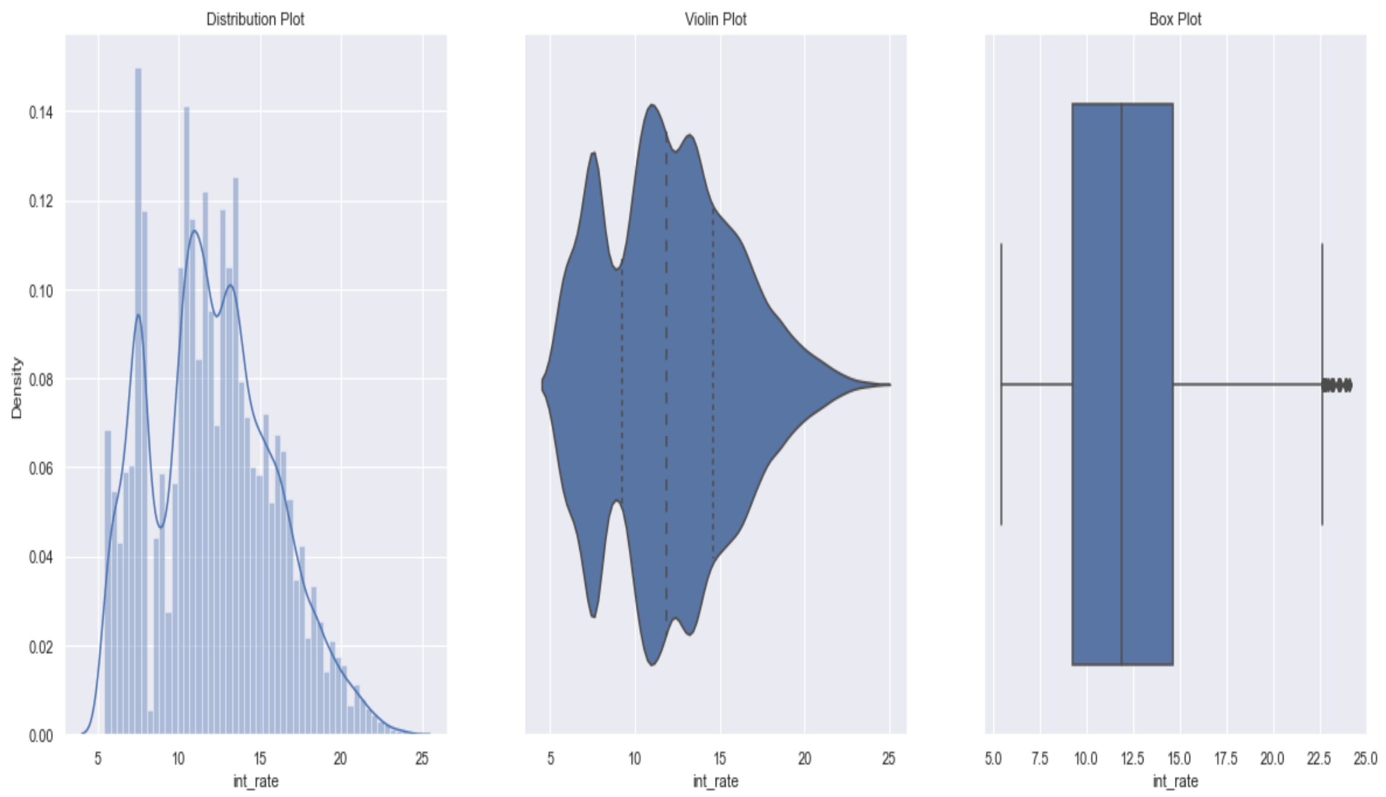
```
univariate(df=loan,col='loan_amnt',vartype=0)
```



Insights: The majority of the loans range from 8000 to 20000 dollars.

2). Interest Rate

```
loan['int_rate'] = loan['int_rate'].replace("%","", regex=True).astype(float)
univariate(df=loan,col='int_rate',vartype=0)
```



Insights: The majority of the loan interest rates are distributed between 10% to 16%.

3). Annual Income

loan["annual_inc"].describe()

```
count      35621
unique      4824
top        60000
freq        1367
Name: annual_inc, dtype: object
```

Remove Outliers (values from 99 to 100%)

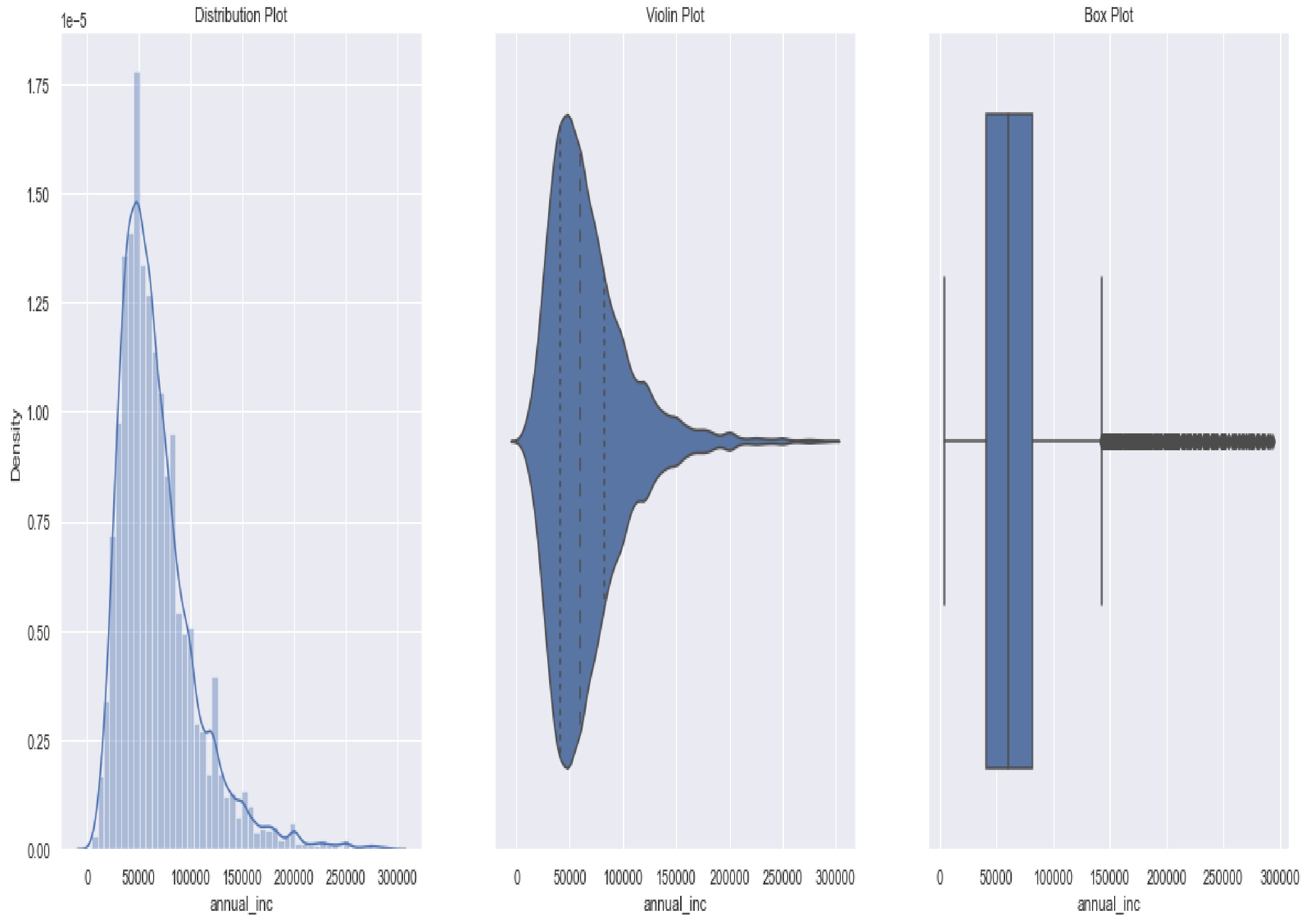
q = loan["annual_inc"].apply(lambda x: float(x)).quantile(0.995)

loan = loan[loan["annual_inc"].apply(lambda x: float(x)) < q]

loan["annual_inc"].describe()

```
count      35621
unique      4824
top        60000
freq        1367
Name: annual_inc, dtype: object
```

```
loan['annual_inc'] = loan['annual_inc'].apply(lambda x: float(x))
univariate(df=loan,col='annual_inc',vartype=0)
```

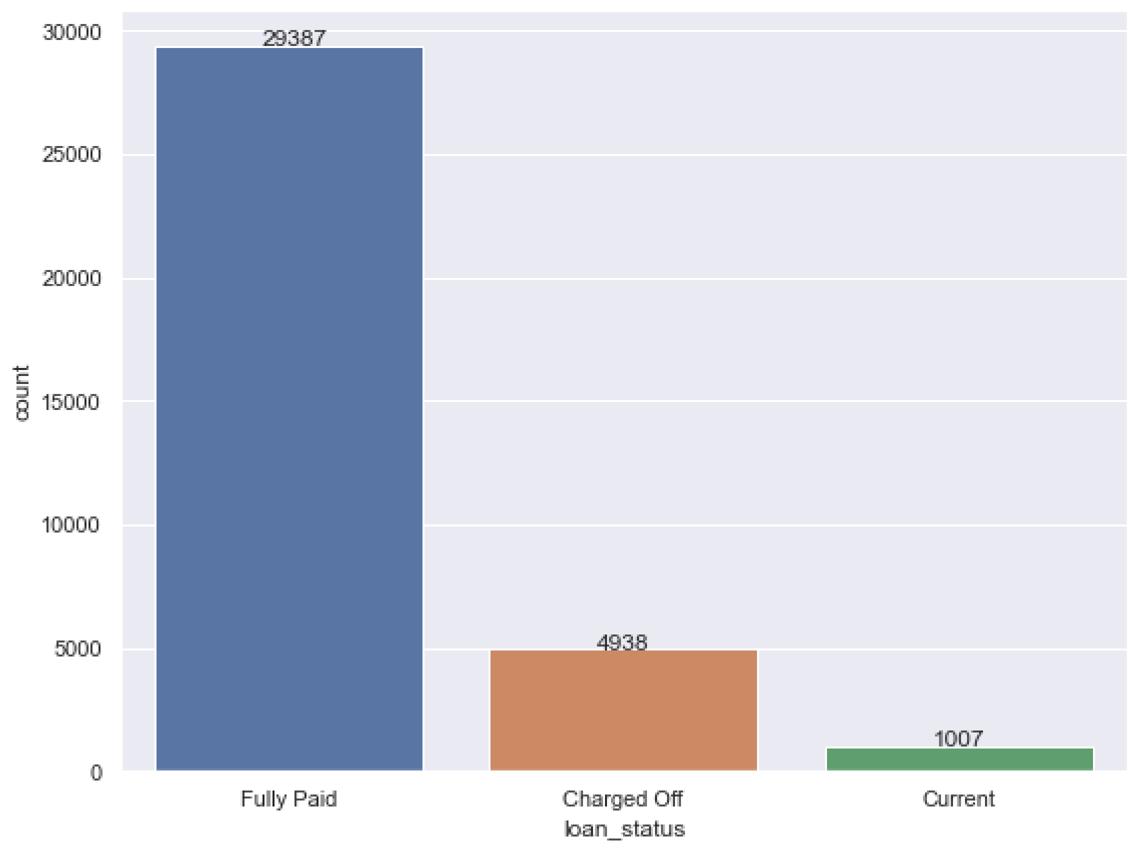


<u>Insights</u>: The majority of the loan applicants earn between 40000 to 90000 USD annually.

Categorical Variables

4). Loan Status

```
univariate(df=loan,col='loan_status',vartype=1)
```



<u>Insights:</u> 5 % Charged off

5). Home Ownership Wise Loan

loan.purpose.unique()

```
array(['credit_card', 'car', 'small_business', 'wedding',
       'debt_consolidation', 'home_improvement', 'major_purchase',
       'medical', 'moving', 'vacation', 'house', 'educational'],
      dtype=object)
```

loan.home_ownership.unique()

```
array(['RENT', 'OWN', 'MORTGAGE', 'OTHER', 'NONE'], dtype=object)
```

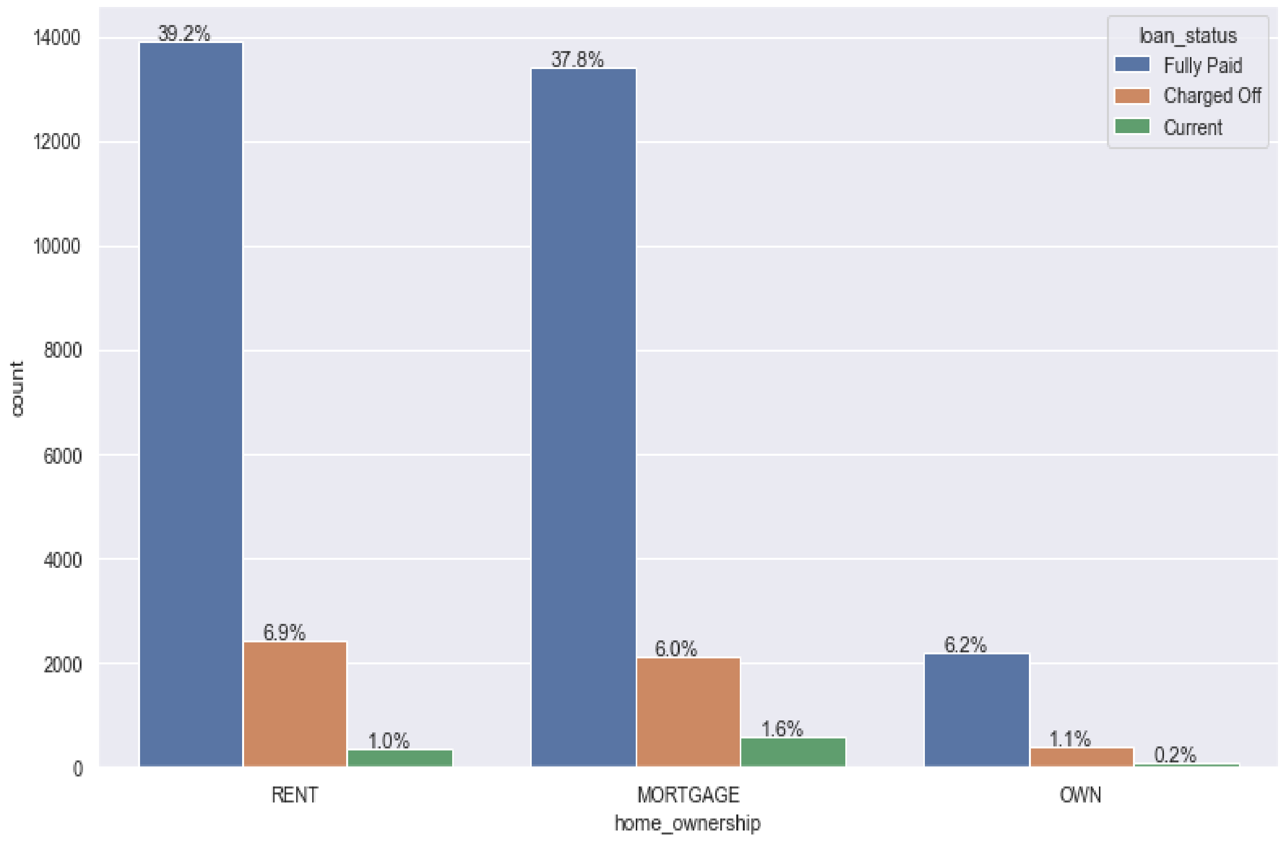
# Remove rows where home_ownership’==’OTHER’, ‘NONE’, ‘ANY’

rem = ['OTHER', 'NONE', 'ANY']

loan.drop(loan[loan['home_ownership'].isin(rem)].index,inplace=True) loan.home_ownership.unique()

```
array(['RENT', 'OWN', 'MORTGAGE'], dtype=object)
```

univariate(df=loan,col='home_ownership',vartype=1,hue='loan_status')



Insights: 40% of applicants live in a rented house, whereas 52% of applicants have a mortgage on their property.

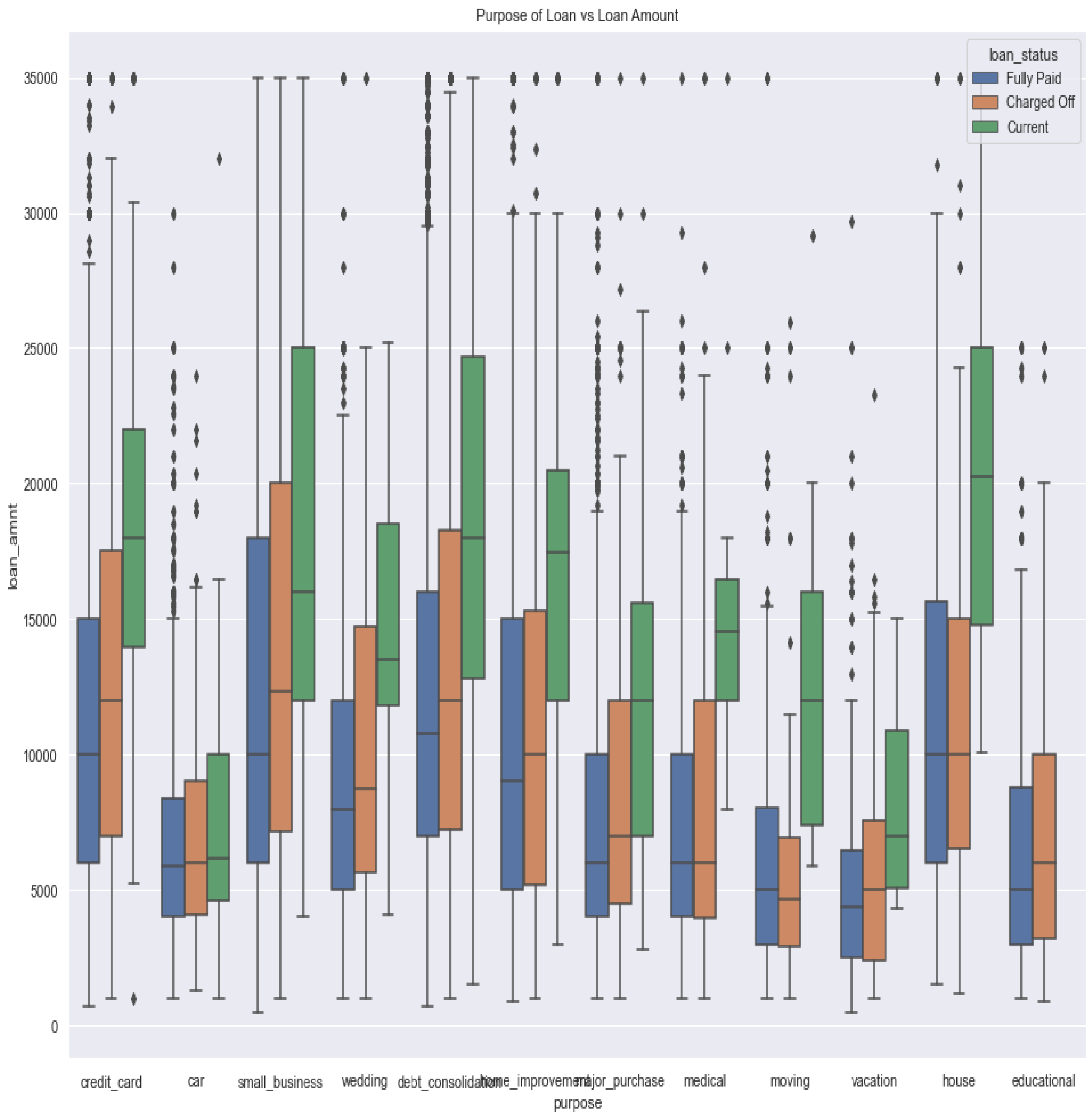
Bi-variate/Multivariate Analysis

The association between two/two or more variables is found using bivariate/multivariate analysis. For every combination of categorical and continuous data, we can perform Bi-variate/Multivariate analysis. Categorical & Categorical, Categorical & Continuous, and Continuous & Continuous are examples of possible combinations.

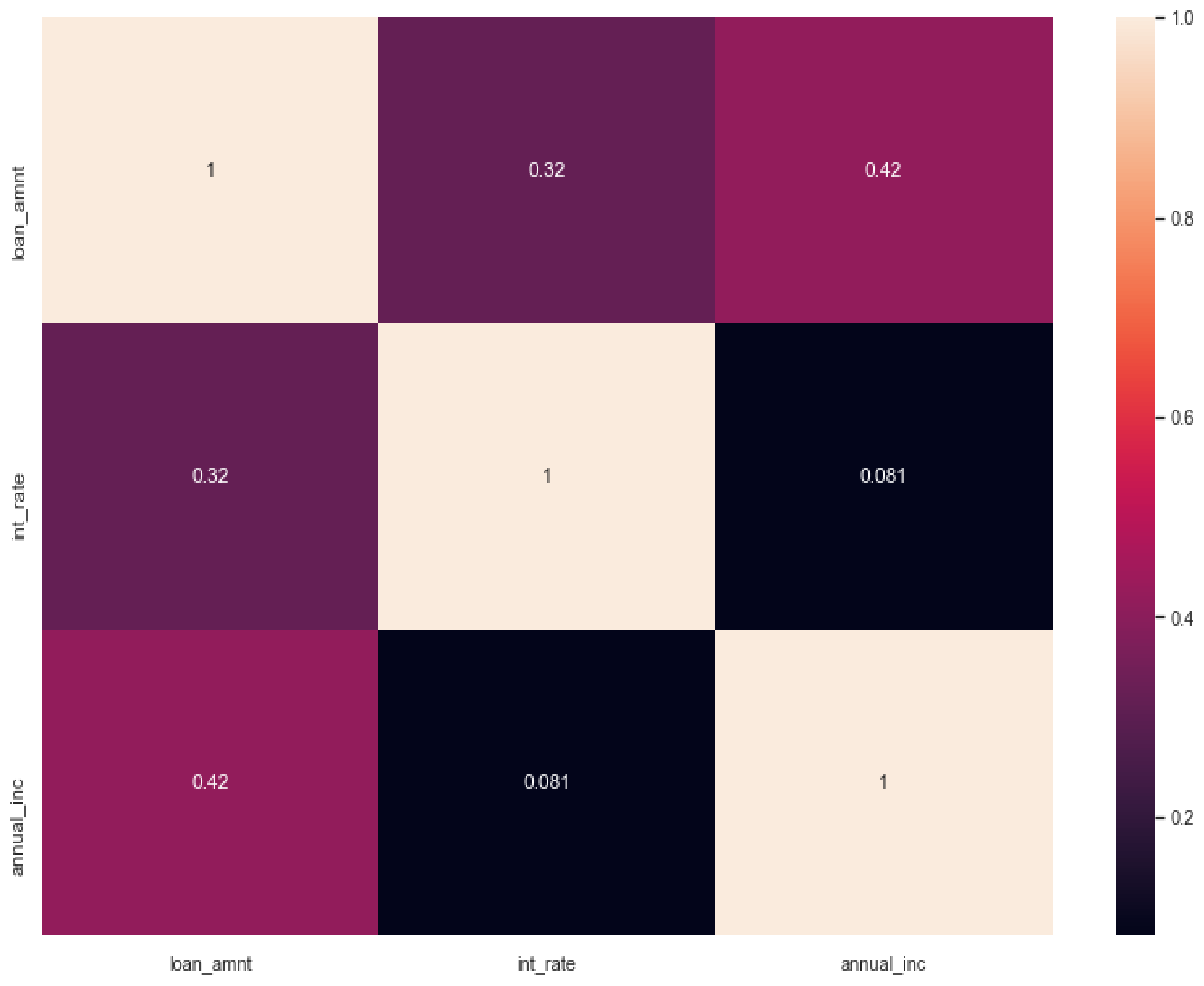
Purpose of Loan / Loan Amount for loan status

plt.figure(figsize=(16,12))

```
loan['loan_amnt'] = loan['loan_amnt'].astype('float')
sns.boxplot(data =loan, x='purpose', y='loan_amnt', hue ='loan_status')
plt.title('Purpose of Loan vs Loan Amount')
plt.show()
```



Heat Map for continuous variables



Insights: The Heat map clearly shows how closely ‘loan amount,’ funded amount, and funded amount inc’ are related. As a result, we can use any one of them for our analysis.

Conclusion

So, here we can see this all different aspect to consider Target Variable is Loan Status And top 5 major variables to consider for loan prediction: Purpose of Loan, Employment Length, Grade, Interest Rate, Term. Now we are ready to Train our model and prediction.

In this article, we learned the most important subjects for any type of analysis so far in this module. The following are the details:

Getting to know the domain Getting to know the data and prepare it for analysis

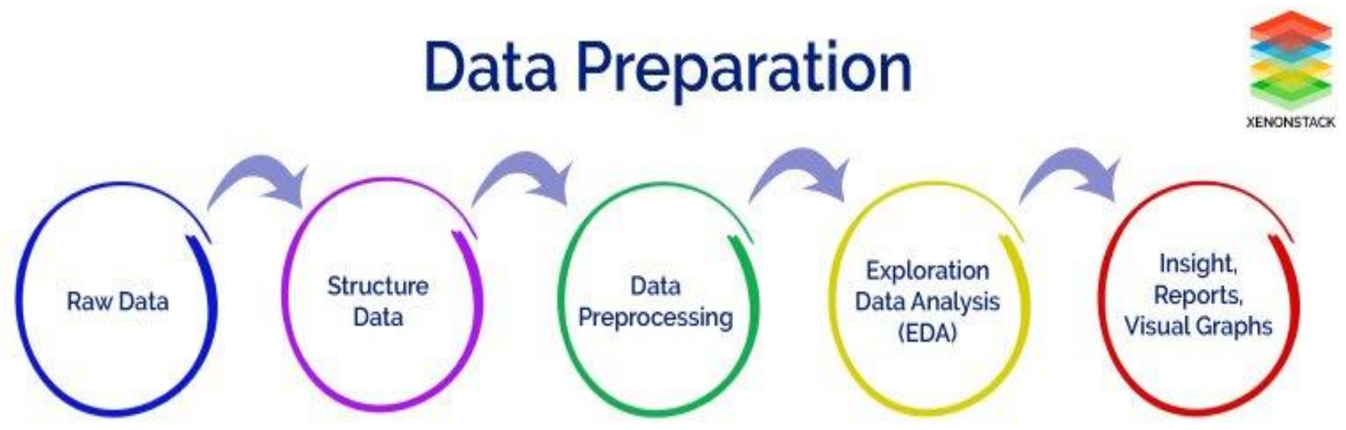
Univariate and segmented uni-variate analysis are two types of univariate analysis.

The bi-variate analysis is a technique for analyzing two variables at the same time.

Creating new metrics based on existing data

The main objective of this article is to cover the steps involved in Data pre-processing, Feature Engineering, and different stages of Exploratory Data Analysis, which is an essential step in any research analysis.

Data pre-processing, Feature Engineering, and EDA are fundamental early steps after data collection. Still, they are not limited to where the data is simply visualized, plotted, and manipulated, without any assumptions, to assess the quality of the data and building models.



Data Pre-processing and Feature Engineering

We spend a lot of time refining our raw data. Data pre-processing and Feature Engineering plays a key role in any data process

Data Pre-processing refers to Data Integration, Data Analysis, Data cleaning, Data Transformation, and Dimension Reduction

Data preprocessing is **the process of cleaning and preparing the raw data to enable feature engineering**

Feature Engineering covers various data engineering techniques such as adding/removing relevant features, handling missing data, encoding the data, handling categorical variables, etc

Feature Engineering is one of the most crucial tasks and **plays a major role in determining the outcome of a model**

Feature engineering involves the creation of features, whereas preprocessing involves cleaning the data.

The Data pre-processing, Feature Engineering, and EDA steps will be carried out in this article using Python.

Import Python Libraries

The first step involved in ML using python is understanding and playing around with our data using libraries. Here is the link to the dataset.

Import all libraries which are required for our analysis, such as Data Loading, Statistical analysis, Visualizations, Data Transformations, Merge and Joins, etc.

Pandas and Numpy have been used for Data Manipulation and numerical Calculations

Matplotlib and Seaborn have been used for Data visualizations.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
#to ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

Reading Dataset

The Pandas library offers a wide range of possibilities for loading data into the pandas DataFrame from files like JSON, .csv, .xlsx, .sql, .pickle, .html, .txt, images etc.

Most of the data are available in a tabular format of CSV files. It is trendy and easy to access. Using the **read_csv()** function, data can be converted to a pandas DataFrame.

In this article, the data to predict **Used car price** is being used as an example. In this dataset, we are trying to analyze the used car's price and how EDA focuses on identifying the factors influencing the car price. We have stored the data in the DataFrame **data.**

```
data = pd.read_csv("used_cars.csv")
```

Analyzing the data

Before we make any inferences, we listen to our data by examining all variables in the data.

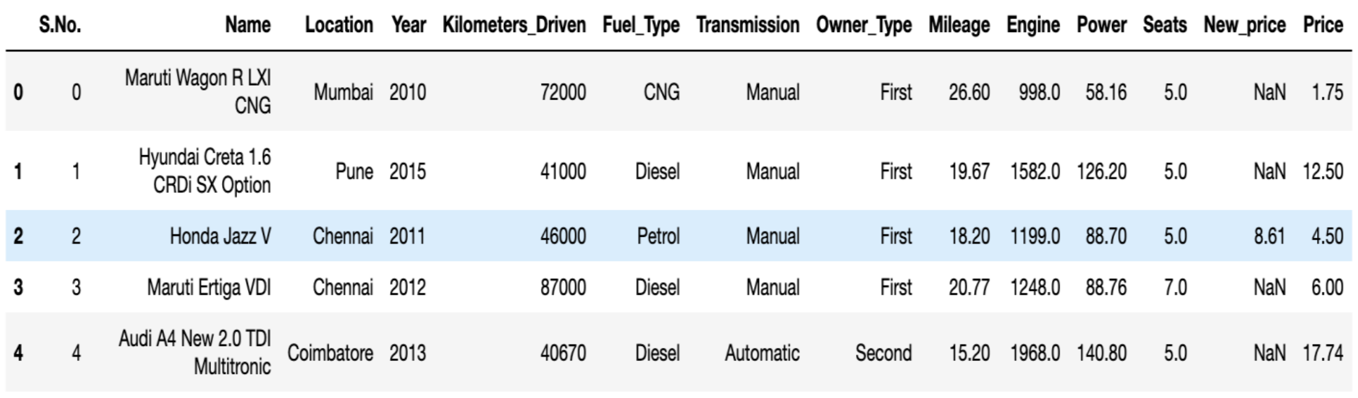
The main goal of data understanding is to gain general insights about the data, which covers the number of rows and columns, values in the data, datatypes, and Missing values in the dataset.

shape – **shape** will display the number of observations(rows) and features(columns) in the dataset

There are 7253 observations and 14 variables in our dataset

head() will display the top 5 observations of the dataset

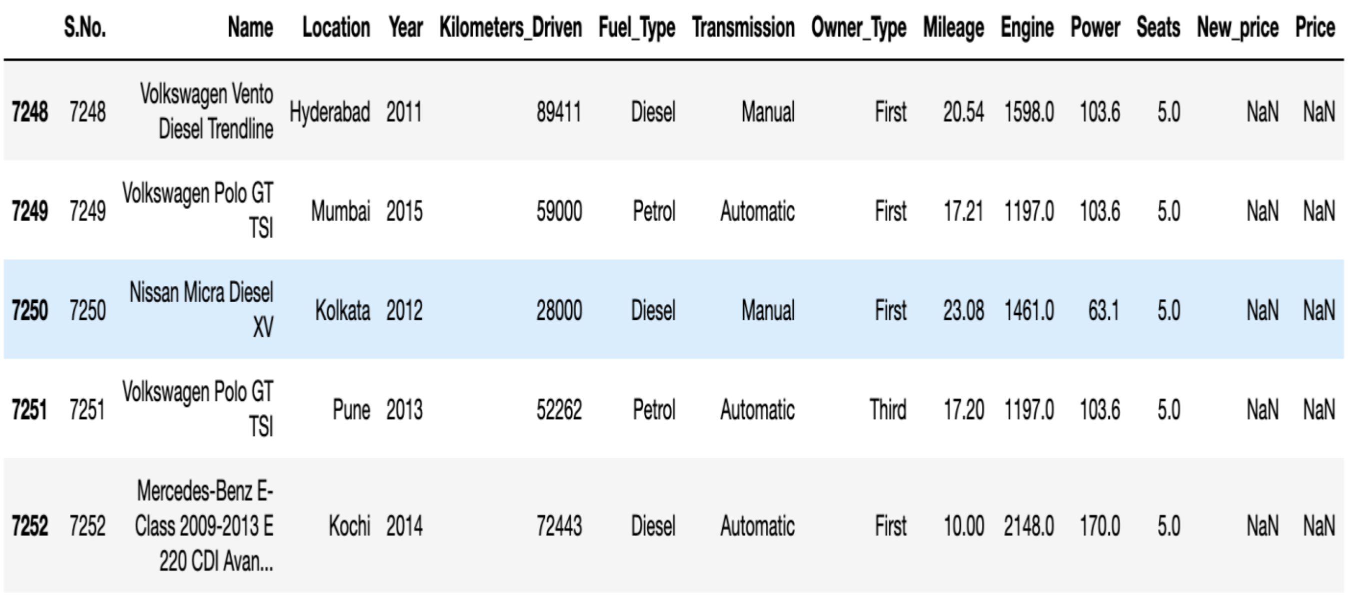
data.head()



	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_price	Price
0	0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.60	998.0	58.16	5.0	NaN	1.75
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67	1582.0	126.20	5.0	NaN	12.50
2	2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.20	1199.0	88.70	5.0	8.61	4.50
3	3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77	1248.0	88.76	7.0	NaN	6.00
4	4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.20	1968.0	140.80	5.0	NaN	17.74

tail() will display the last 5 observations of the dataset

data.tail()



	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_price	Price
7248	7248	Volkswagen Vento Diesel Trendline	Hyderabad	2011	89411	Diesel	Manual	First	20.54	1598.0	103.6	5.0	NaN	NaN
7249	7249	Volkswagen Polo GT TSI	Mumbai	2015	59000	Petrol	Automatic	First	17.21	1197.0	103.6	5.0	NaN	NaN
7250	7250	Nissan Micra Diesel XV	Kolkata	2012	28000	Diesel	Manual	First	23.08	1461.0	63.1	5.0	NaN	NaN
7251	7251	Volkswagen Polo GT TSI	Pune	2013	52262	Petrol	Automatic	Third	17.20	1197.0	103.6	5.0	NaN	NaN
7252	7252	Mercedes-Benz E-Class 2009-2013 E 220 CDI Avan...	Kochi	2014	72443	Diesel	Automatic	First	10.00	2148.0	170.0	5.0	NaN	NaN

info() helps to understand the data type and information about data, including the number of records in each column, data having null or not null, Data type, the memory usage of the dataset

data.info()

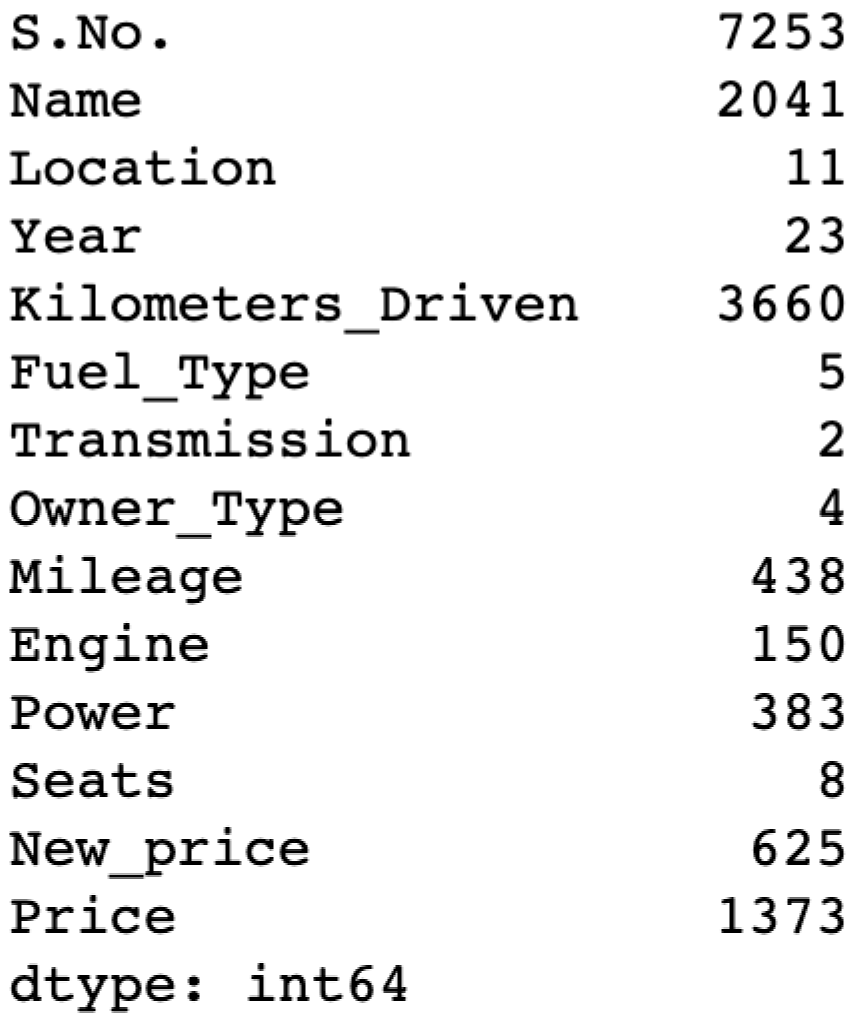
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7253 entries, 0 to 7252
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   S.No.              7253 non-null   int64  
 1   Name               7253 non-null   object 
 2   Location           7253 non-null   object 
 3   Year               7253 non-null   int64  
 4   Kilometers_Driven  7253 non-null   int64  
 5   Fuel_Type          7253 non-null   object 
 6   Transmission       7253 non-null   object 
 7   Owner_Type         7253 non-null   object 
 8   Mileage            7251 non-null   float64
 9   Engine             7207 non-null   float64
 10  Power              7078 non-null   float64
 11  Seats              7200 non-null   float64
 12  New_price          1006 non-null   float64
 13  Price              6019 non-null   float64
dtypes: float64(6), int64(3), object(5)
memory usage: 793.4+ KB
```

data.info() shows the variables Mileage, Engine, Power, Seats, New_Price, and Price have missing values. Numeric variables like Mileage, Power are of datatype as float64 and int64. Categorical variables like Location, Fuel_Type, Transmission, and Owner Type are of object data type

Check for duplication

nunique() based on several unique values in each column and the data description, we can identify the continuous and categorical columns in the data. Duplicated data can be handled or removed based on further analysis

data.nunique()



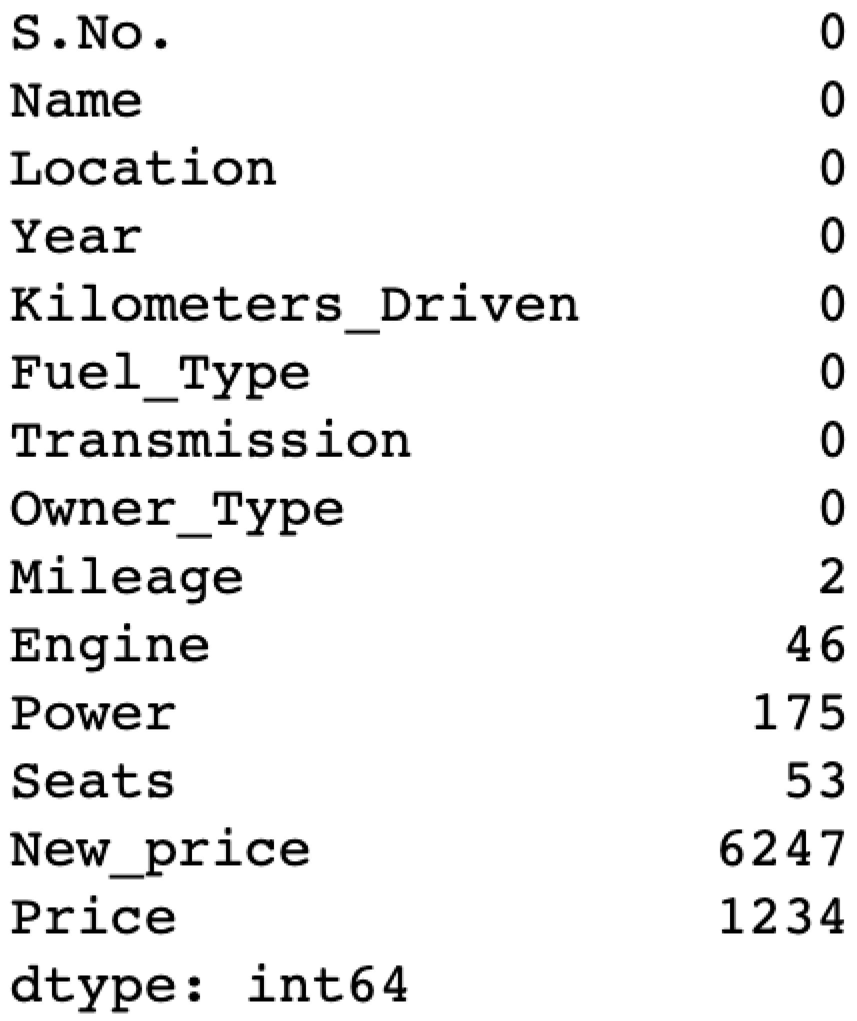
```
S.No.                 7253
Name                  2041
Location                11
Year                    23
Kilometers_Driven     3660
Fuel_Type                5
Transmission             2
Owner_Type               4
Mileage                438
Engine                 150
Power                  383
Seats                    8
New_price              625
Price                 1373
dtype: int64
```

Missing values Calculation

isnull() is widely been in all pre-processing steps to identify null values in the data

In our example, **data.isnull().sum()** is used to get the number of missing records in each column

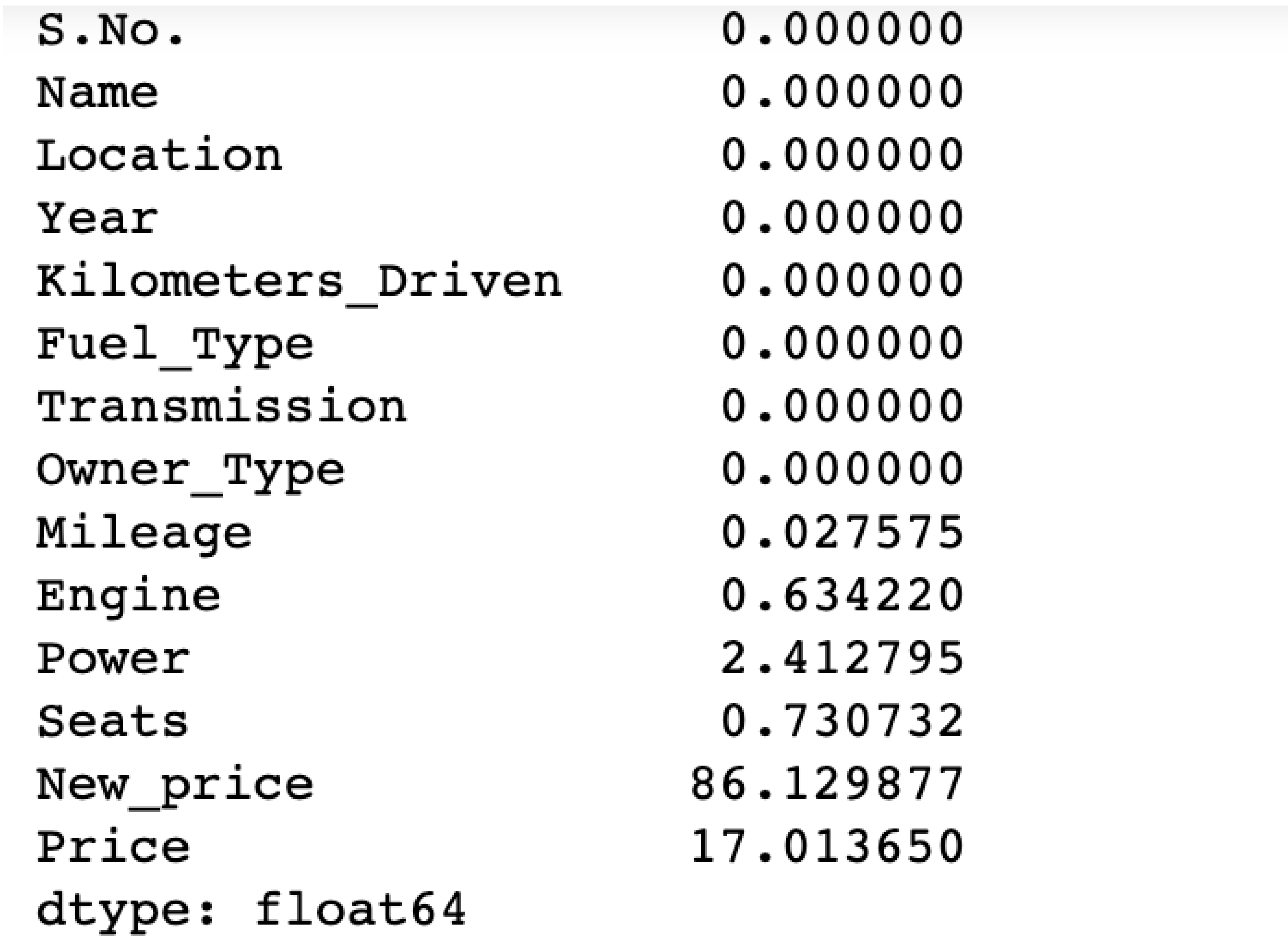
data.isnull().sum()



```
S.No.                   0
Name                    0
Location                0
Year                    0
Kilometers_Driven       0
Fuel_Type               0
Transmission            0
Owner_Type              0
Mileage                 2
Engine                 46
Power                 175
Seats                  53
New_price            6247
Price                1234
dtype: int64
```

The below code helps to calculate the percentage of missing values in each column

(data.isnull().sum()/(len(data)))*100



```
S.No.                  0.000000
Name                   0.000000
Location               0.000000
Year                   0.000000
Kilometers_Driven      0.000000
Fuel_Type              0.000000
Transmission           0.000000
Owner_Type             0.000000
Mileage                0.027575
Engine                 0.634220
Power                  2.412795
Seats                  0.730732
New_price             86.129877
Price                 17.013650
dtype: float64
```

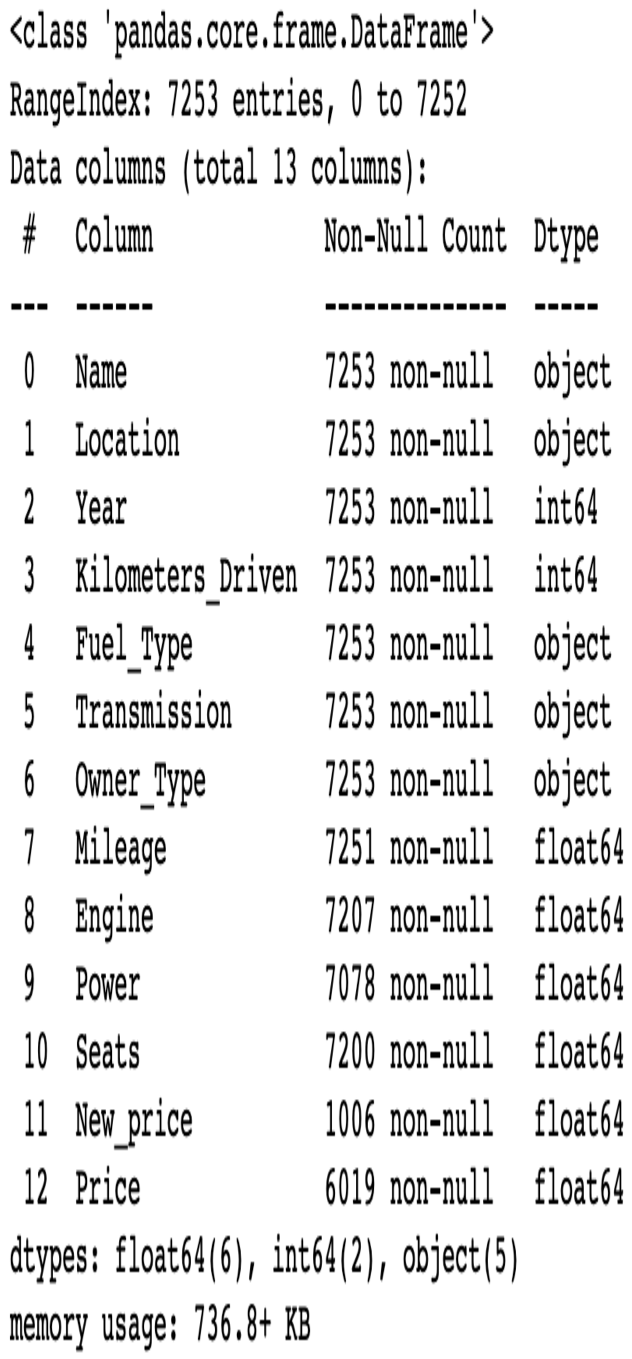
The percentage of missing values for the columns **New_Price** and **Price** is ~86% and ~17%, respectively.

Data Reduction

Some columns or variables can be dropped if they do not add value to our analysis.

In our dataset, the column S.No have only ID values, assuming they don’t have any predictive power to predict the dependent variable.

```
# Remove S.No. column from data
data = data.drop(['S.No.'], axis = 1)
data.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7253 entries, 0 to 7252
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Name               7253 non-null   object
 1   Location           7253 non-null   object
 2   Year               7253 non-null   int64
 3   Kilometers_Driven  7253 non-null   int64
 4   Fuel_Type          7253 non-null   object
 5   Transmission       7253 non-null   object
 6   Owner_Type         7253 non-null   object
 7   Mileage            7251 non-null   float64
 8   Engine             7207 non-null   float64
 9   Power              7078 non-null   float64
 10  Seats              7200 non-null   float64
 11  New_price          1006 non-null   float64
 12  Price              6019 non-null   float64
dtypes: float64(6), int64(2), object(5)
memory usage: 736.8+ KB
```

We start our Feature Engineering as we need to add some columns required for analysis.

Feature Engineering

Feature engineering refers to the process of using domain knowledge to select and transform the most relevant variables from raw data when creating a predictive model using machine learning or statistical modeling. The main goal of Feature engineering is to create meaningful data from raw data.

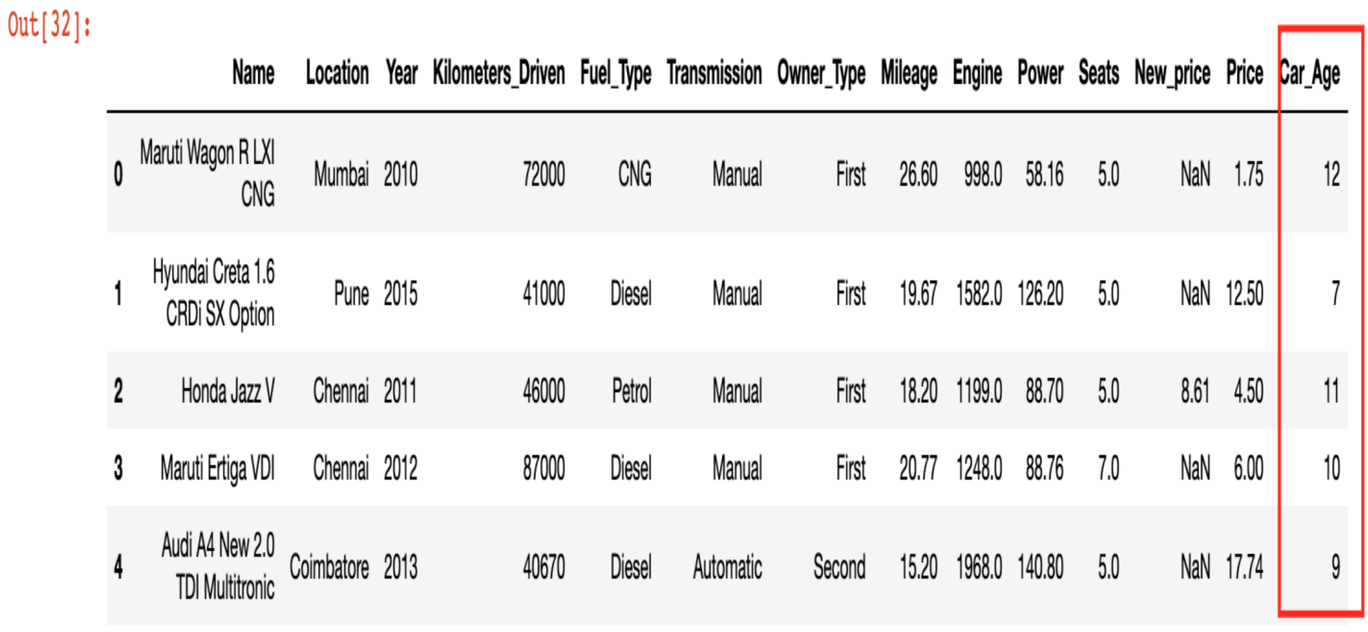
Creating Features

We will play around with the variables Year and Name in our dataset. If we see the sample data, the column “Year” shows the manufacturing year of the car.

It would be difficult to find the car’s age if it is in year format as the Age of the car is a contributing factor to Car Price.

Introducing a new column, “Car_Age” to know the age of the car

```
from datetime import date
date.today().year
data['Car_Age']=date.today().year-data['Year']
data.head()
```

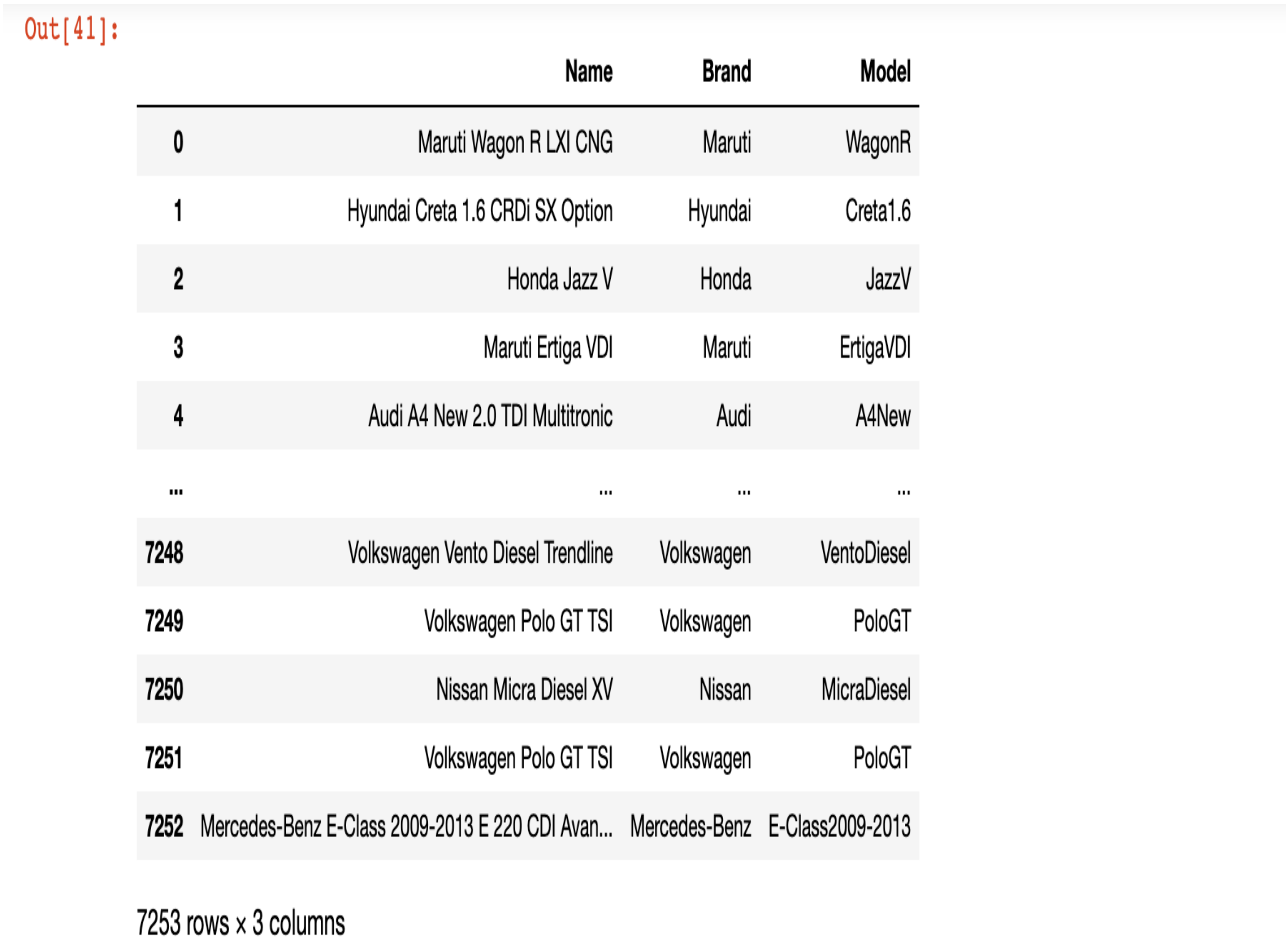


```
Out[32]:
```

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_price	Price	Car_Age
0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.60	998.0	58.16	5.0	NaN	1.75	12
1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67	1582.0	126.20	5.0	NaN	12.50	7
2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.20	1199.0	88.70	5.0	8.61	4.50	11
3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77	1248.0	88.76	7.0	NaN	6.00	10
4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.20	1968.0	140.80	5.0	NaN	17.74	9

Since car names will not be great predictors of the price in our current data. But we can process this column to extract important information using brand and Model names. **Let’s split the name and introduce new variables “Brand” and “Model”**

```
data['Brand'] = data.Name.str.split().str.get(0)
data['Model'] = data.Name.str.split().str.get(1) + data.Name.str.split().str.get(2)
data[['Name','Brand','Model']]
```



Out[41]:

	Name	Brand	Model
0	Maruti Wagon R LXI CNG	Maruti	WagonR
1	Hyundai Creta 1.6 CRDi SX Option	Hyundai	Creta1.6
2	Honda Jazz V	Honda	JazzV
3	Maruti Ertiga VDI	Maruti	ErtigaVDI
4	Audi A4 New 2.0 TDI Multitronic	Audi	A4New
...	...	...	...
7248	Volkswagen Vento Diesel Trendline	Volkswagen	VentoDiesel
7249	Volkswagen Polo GT TSI	Volkswagen	PoloGT
7250	Nissan Micra Diesel XV	Nissan	MicraDiesel
7251	Volkswagen Polo GT TSI	Volkswagen	PoloGT
7252	Mercedes-Benz E-Class 2009-2013 E 220 CDI Avan...	Mercedes-Benz	E-Class2009-2013

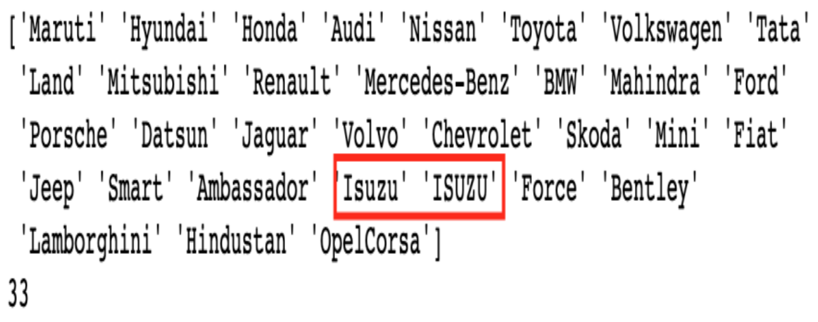
7253 rows × 3 columns

Data Cleaning/Wrangling

Some names of the variables are not relevant and not easy to understand. Some data may have data entry errors, and some variables may need data type conversion. We need to fix this issue in the data.

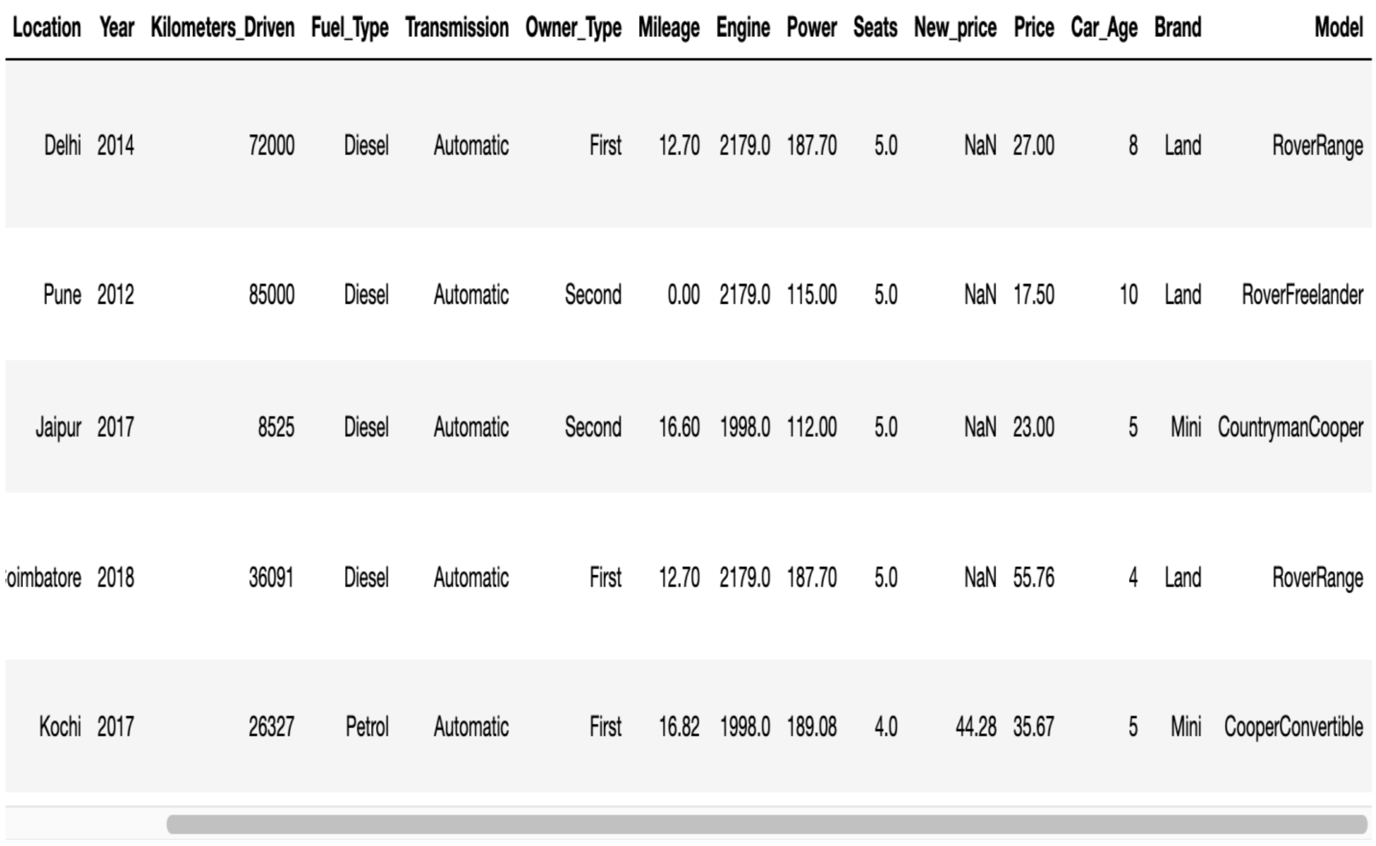
In the example, **The brand name ‘Isuzu’ ‘ISUZU’ and ‘Mini’ and ‘Land’ looks incorrect. This needs to be corrected**

```
print(data.Brand.unique())
print(data.Brand.nunique())
```



```
['Maruti' 'Hyundai' 'Honda' 'Audi' 'Nissan' 'Toyota' 'Volkswagen' 'Tata'
 'Land' 'Mitsubishi' 'Renault' 'Mercedes-Benz' 'BMW' 'Mahindra' 'Ford'
 'Porsche' 'Datsun' 'Jaguar' 'Volvo' 'Chevrolet' 'Skoda' 'Mini' 'Fiat'
 'Jeep' 'Smart' 'Ambassador' 'Isuzu' 'ISUZU' 'Force' 'Bentley'
 'Lamborghini' 'Hindustan' 'OpelCorsa']
33
```

```
searchfor = ['Isuzu' ,'ISUZU','Mini','Land']
data[data.Brand.str.contains('|'.join(searchfor))].head(5)
```



Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_price	Price	Car_Age	Brand	Model
Delhi	2014	72000	Diesel	Automatic	First	12.70	2179.0	187.70	5.0	NaN	27.00	8	Land	RoverRange
Pune	2012	85000	Diesel	Automatic	Second	0.00	2179.0	115.00	5.0	NaN	17.50	10	Land	RoverFreelander
Jaipur	2017	8525	Diesel	Automatic	Second	16.60	1998.0	112.00	5.0	NaN	23.00	5	Mini	CountrymanCooper
oimbatore	2018	36091	Diesel	Automatic	First	12.70	2179.0	187.70	5.0	NaN	55.76	4	Land	RoverRange
Kochi	2017	26327	Petrol	Automatic	First	16.82	1998.0	189.08	4.0	44.28	35.67	5	Mini	CooperConvertible

```
data["Brand"].replace({"ISUZU": "Isuzu", "Mini": "Mini Cooper","Land":"Land
Rover"}, inplace=True)
```

We have done the fundamental data analysis, Featuring, and data clean-up. Let’s move to the EDA process

Voila!! Our Data is ready to perform EDA.

EDA Exploratory Data Analysis

Exploratory Data Analysis refers to the crucial process of performing initial investigations on data to discover patterns to check assumptions with the help of summary statistics and graphical representations.

- EDA can be leveraged to check for outliers, patterns, and trends in the given data.
- EDA helps to find meaningful patterns in data.
- EDA provides in-depth insights into the data sets to solve our business problems.
- EDA gives a clue to impute missing values in the dataset

Statistics Summary

The information gives a quick and simple description of the data.

Can include Count, Mean, Standard Deviation, median, mode, minimum value, maximum value, range, standard deviation, etc.

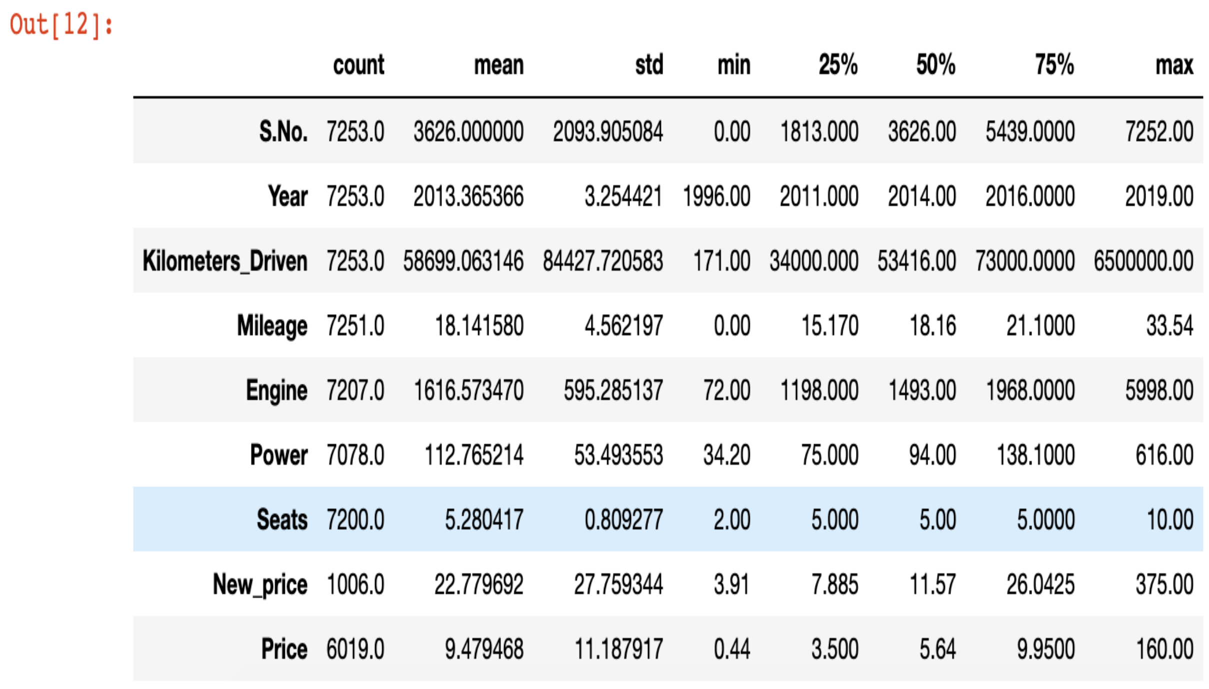
Statistics summary gives a high-level idea to identify whether the data has any outliers, data entry error, distribution of data such as the data is normally distributed or left/right skewed

In python, this can be achieved using describe()

describe() function gives all statistics summary of data

describe()– Provide a statistics summary of data belonging to numerical datatype such as int, float

data.describe().T


Out[12]:

	count	mean	std	min	25%	50%	75%	max
S.No.	7253.0	3626.000000	2093.905084	0.00	1813.000	3626.00	5439.0000	7252.00
Year	7253.0	2013.365366	3.254421	1996.00	2011.000	2014.00	2016.0000	2019.00
Kilometers_Driven	7253.0	58699.063146	84427.720583	171.00	34000.000	53416.00	73000.0000	6500000.00
Mileage	7251.0	18.141580	4.562197	0.00	15.170	18.16	21.1000	33.54
Engine	7207.0	1616.573470	595.285137	72.00	1198.000	1493.00	1968.0000	5998.00
Power	7078.0	112.765214	53.493553	34.20	75.000	94.00	138.1000	616.00
Seats	7200.0	5.280417	0.809277	2.00	5.000	5.00	5.0000	10.00
New_price	1006.0	22.779692	27.759344	3.91	7.885	11.57	26.0425	375.00
Price	6019.0	9.479468	11.187917	0.44	3.500	5.64	9.9500	160.00

From the statistics summary, we can infer the below findings :

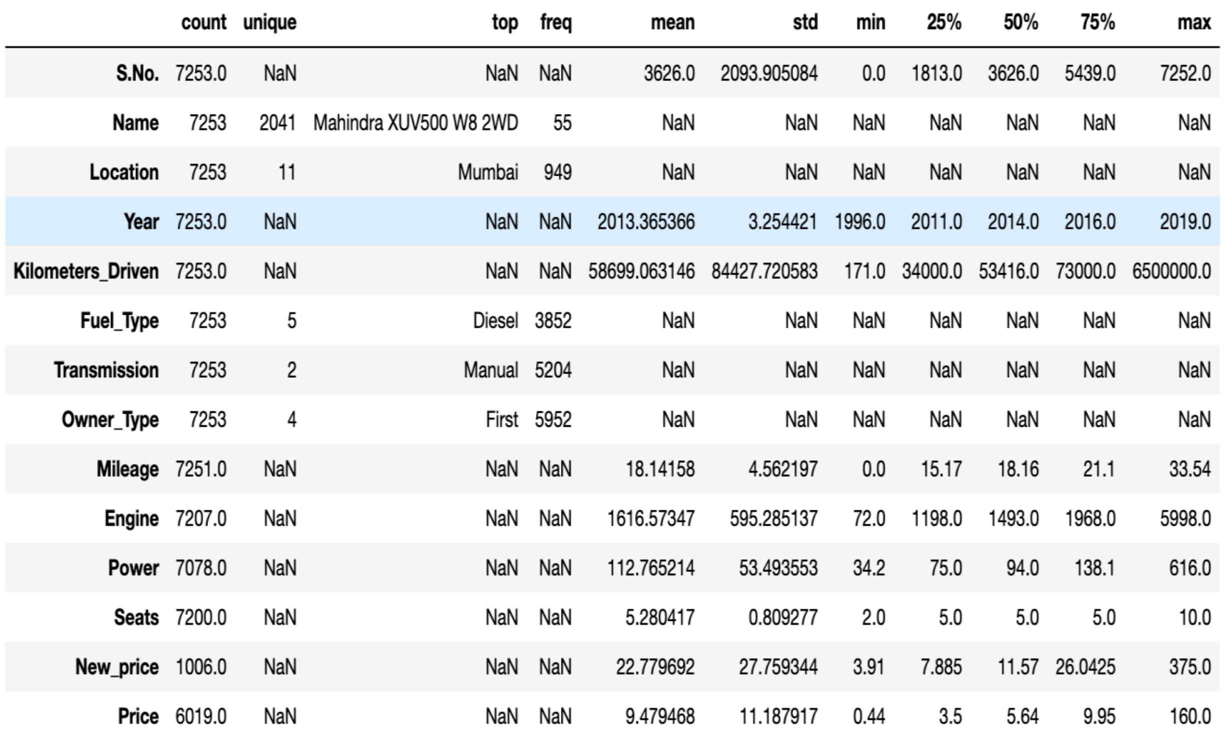
- Years range from 1996- 2019 and has a high in a range which shows used cars contain both latest models and old model cars.
- On average of Kilometers-driven in Used cars are ~58k KM. The range shows a huge difference between min and max as max values show 650000 KM shows the evidence of an outlier. This record can be removed.
- Min value of Mileage shows 0 cars won’t be sold with 0 mileage. This sounds like a data entry issue.
- It looks like Engine and Power have outliers, and the data is right-skewed.
- The average number of seats in a car is 5. car seat is an important feature in price contribution.
- The max price of a used car is 160k which is quite weird, such a high price for used cars. There may be an outlier or data entry issue.

describe(include=’all’) provides a statistics summary of all data, include object,

category etc

data.describe(include='all').T

Out[13]:



	count	unique	top	freq	mean	std	min	25%	50%	75%	max
S.No.	7253.0	NaN	NaN	NaN	3626.0	2093.905084	0.0	1813.0	3626.0	5439.0	7252.0
Name	7253	2041	Mahindra XUV500 W8 2WD	55	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Location	7253	11	Mumbai	949	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Year	7253.0	NaN	NaN	NaN	2013.365366	3.254421	1996.0	2011.0	2014.0	2016.0	2019.0
Kilometers_Driven	7253.0	NaN	NaN	NaN	58699.063146	84427.720583	171.0	34000.0	53416.0	73000.0	6500000.0
Fuel_Type	7253	5	Diesel	3852	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Transmission	7253	2	Manual	5204	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Owner_Type	7253	4	First	5952	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Mileage	7251.0	NaN	NaN	NaN	18.14158	4.562197	0.0	15.17	18.16	21.1	33.54
Engine	7207.0	NaN	NaN	NaN	1616.57347	595.285137	72.0	1198.0	1493.0	1968.0	5998.0
Power	7078.0	NaN	NaN	NaN	112.765214	53.493553	34.2	75.0	94.0	138.1	616.0
Seats	7200.0	NaN	NaN	NaN	5.280417	0.809277	2.0	5.0	5.0	5.0	10.0
New_price	1006.0	NaN	NaN	NaN	22.779692	27.759344	3.91	7.885	11.57	26.0425	375.0
Price	6019.0	NaN	NaN	NaN	9.479468	11.187917	0.44	3.5	5.64	9.95	160.0

Before we do EDA, lets separate Numerical and categorical variables for easy analysis

```
cat_cols=data.select_dtypes(include=['object']).columns
num_cols = data.select_dtypes(include=np.number).columns.tolist()
print("Categorical Variables:")
print(cat_cols)
print("Numerical Variables:")
print(num_cols)
```

```
Categorical Variables:
Index(['Name', 'Location', 'Fuel_Type', 'Transmission', 'Owner_Type', 'Brand',
       'Model'],
      dtype='object')
Numerical Variables:
['Year', 'Kilometers_Driven', 'Mileage', 'Engine', 'Power', 'Seats', 'New_price', 'Price', 'Car_Age']
```

EDA Univariate Analysis

Analyzing/visualizing the dataset by taking one variable at a time:

Data visualization is essential; we must decide what charts to plot to better understand the data. In this article, we visualize our data using Matplotlib and Seaborn libraries.

Matplotlib is a Python 2D plotting library used to draw basic charts we use Matplotlib.

Seaborn is also a python library built on top of Matplotlib that uses short lines of code to create and style statistical plots from Pandas and Numpy

Univariate analysis can be done for both Categorical and Numerical variables.

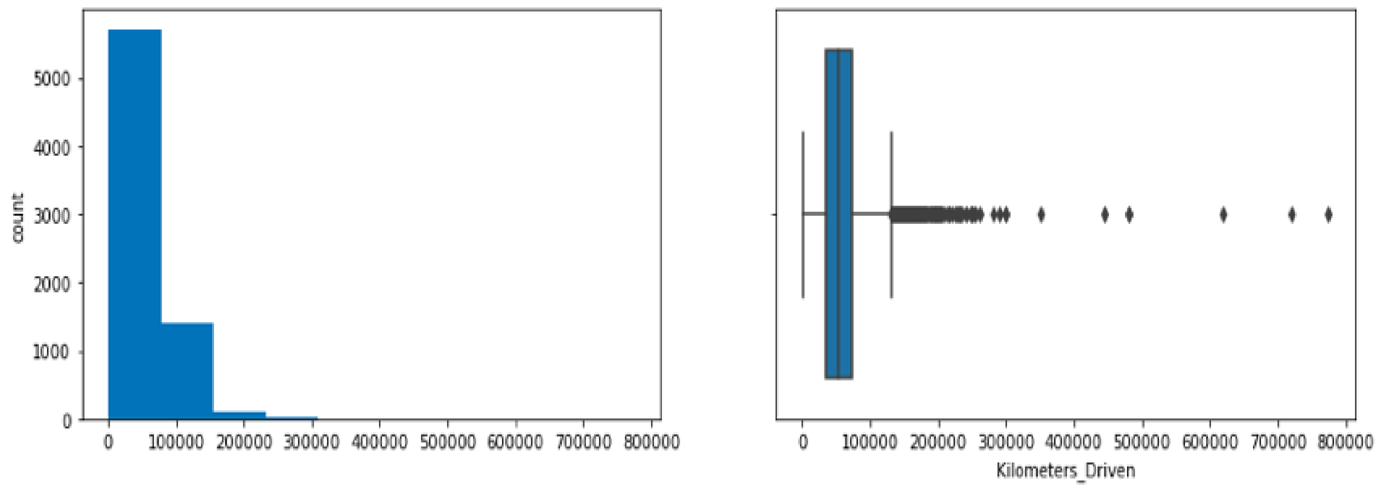
Categorical variables can be visualized using a Count plot, Bar Chart, Pie Plot, etc.

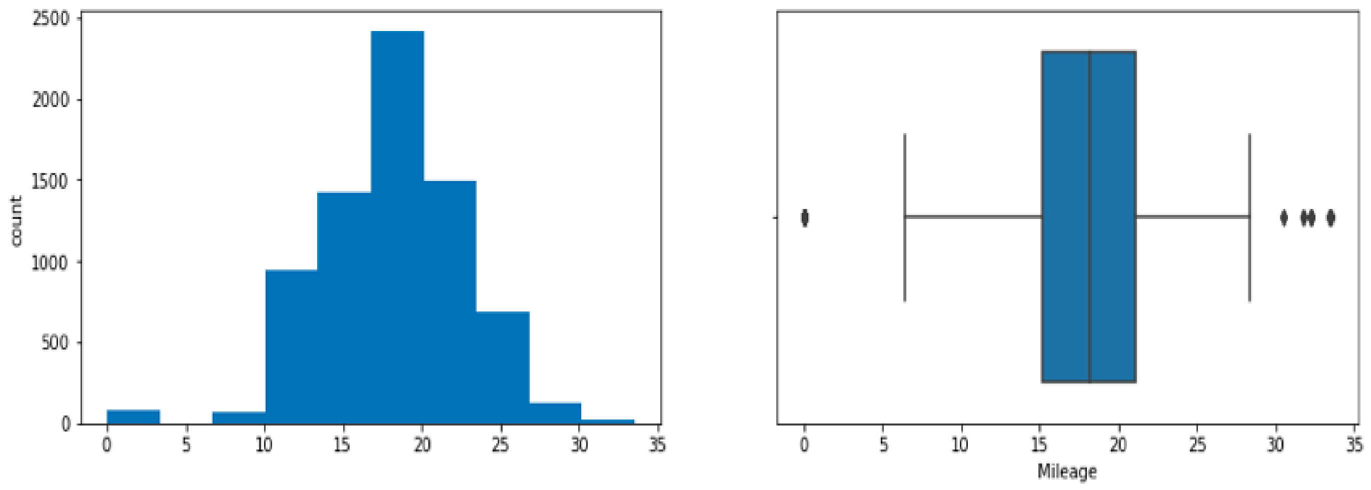
Numerical Variables can be visualized using Histogram, Box Plot, Density Plot, etc.

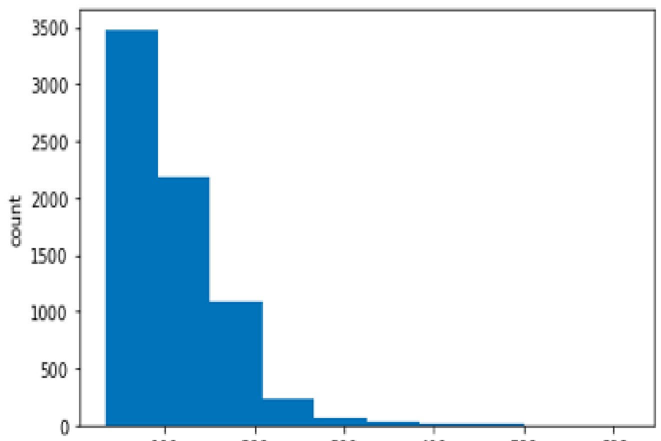
In our example, we have done a Univariate analysis using Histogram and Box Plot for continuous Variables.

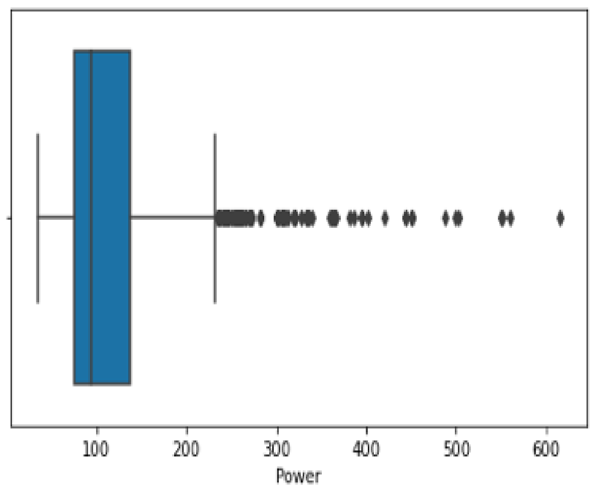
In the below fig, a histogram and box plot is used to show the pattern of the variables, as some variables have skewness and outliers.

```
for col in num_cols:
    print(col)
    print('Skew :', round(data[col].skew(), 2))
    plt.figure(figsize = (15, 4))
    plt.subplot(1, 2, 1)
    data[col].hist(grid=False)
    plt.ylabel('count')
    plt.subplot(1, 2, 2)
    sns.boxplot(x=data[col])
    plt.show()
```









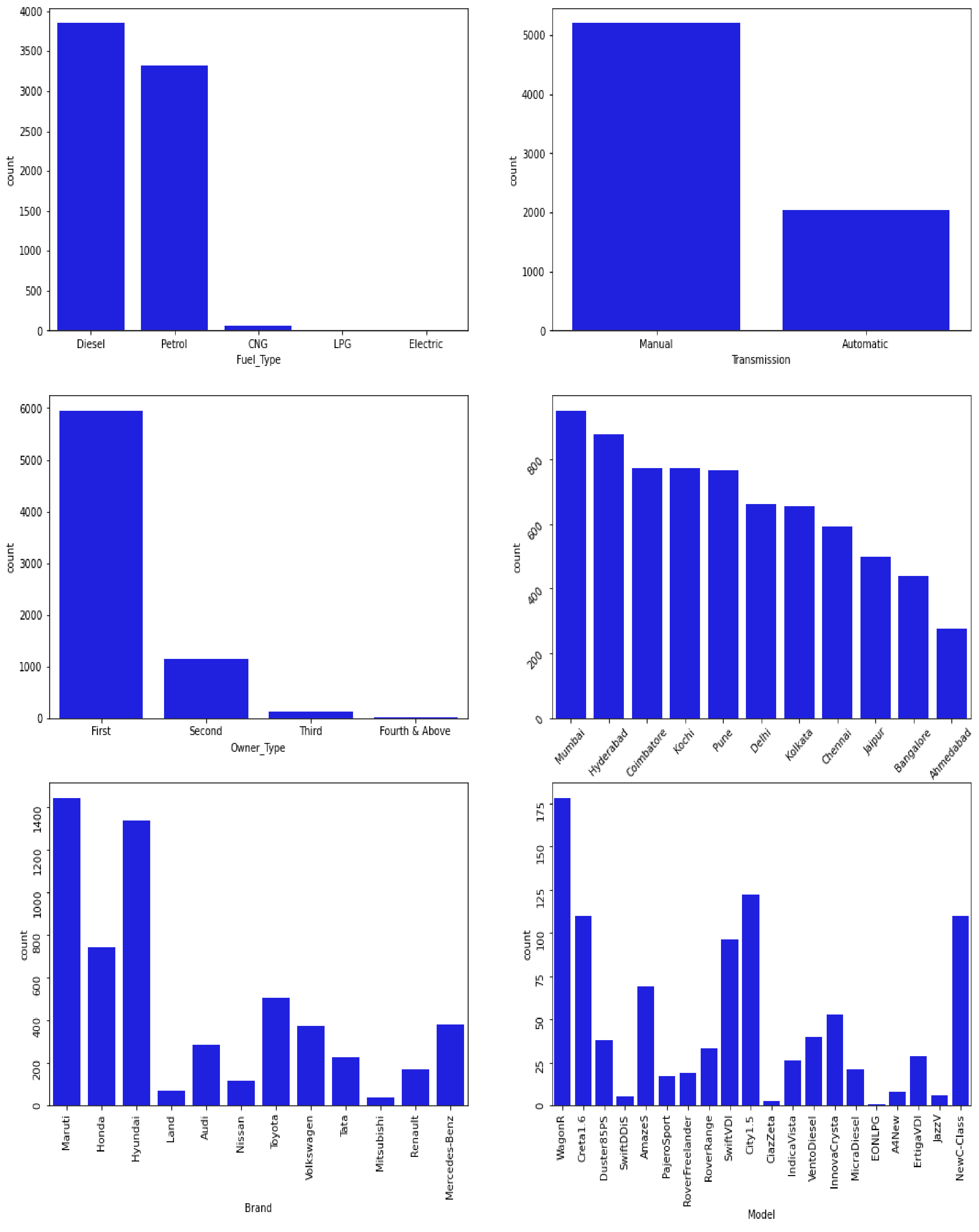
Price and Kilometers Driven are right skewed for this data to be transformed, and all outliers will be handled during imputation

categorical variables are being visualized using a count plot. Categorical variables provide the pattern of factors influencing car price

```
fig, axes = plt.subplots(3, 2, figsize = (18, 18))
fig.suptitle('Bar plot for all categorical variables in the dataset')
sns.countplot(ax = axes[0, 0], x = 'Fuel_Type', data = data, color = 'blue',
        order = data['Fuel_Type'].value_counts().index);
sns.countplot(ax = axes[0, 1], x = 'Transmission', data = data, color = 'blue',
        order = data['Transmission'].value_counts().index);
sns.countplot(ax = axes[1, 0], x = 'Owner_Type', data = data, color = 'blue',
        order = data['Owner_Type'].value_counts().index);
sns.countplot(ax = axes[1, 1], x = 'Location', data = data, color = 'blue',
        order = data['Location'].value_counts().index);
sns.countplot(ax = axes[2, 0], x = 'Brand', data = data, color = 'blue',
        order = data['Brand'].head(20).value_counts().index);
sns.countplot(ax = axes[2, 1], x = 'Model', data = data, color = 'blue',
        order = data['Model'].head(20).value_counts().index);
axes[1][1].tick_params(labelrotation=45);
```

```
axes[2][0].tick_params(labelrotation=90);
axes[2][1].tick_params(labelrotation=90);
```

Count plot for all categorical variables in the dataset



From the count plot, we can have below observations

- Mumbai has the highest number of cars available for purchase, followed by Hyderabad and Coimbatore
- ~53% of cars have fuel type as Diesel this shows diesel cars provide higher performance
- ~72% of cars have manual transmission
- ~82 % of cars are First owned cars. This shows most of the buyers prefer to purchase first-owner cars
- ~20% of cars belong to the brand Maruti followed by 19% of cars belonging to Hyundai
- WagonR ranks first among all models which are available for purchase

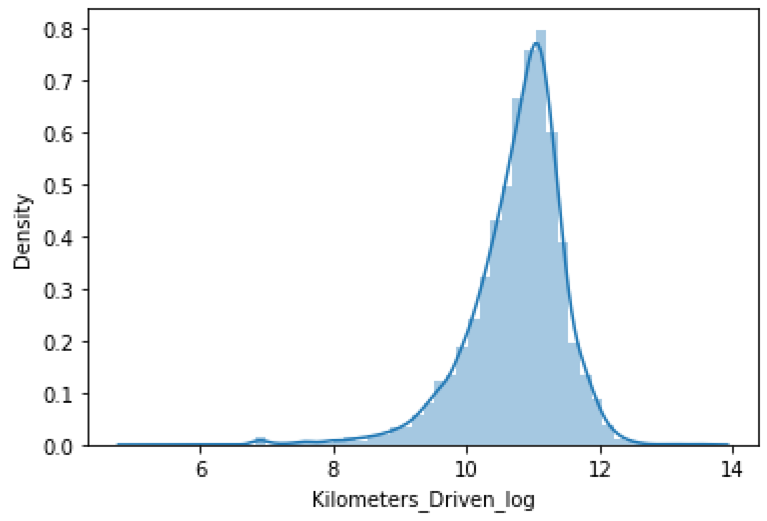
Data Transformation

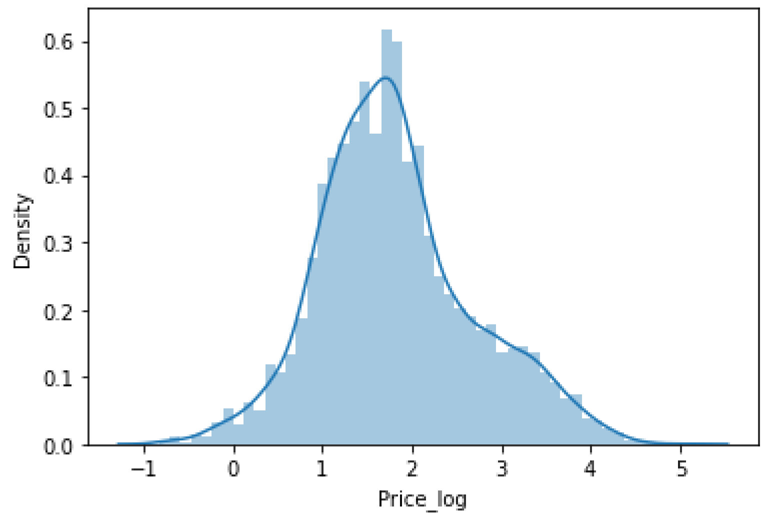
Before we proceed to Bi-variate Analysis, Univariate analysis demonstrated the data pattern as some variables to be transformed.

Price and Kilometer-Driven variables are highly skewed and on a larger scale. Let’s do log transformation.

Log transformation can help in normalization, so this variable can maintain standard scale with other variables:

```
# Function for log transformation of the column
def log_transform(data,col):
    for colname in col:
        if (data[colname] == 1.0).all():
            data[colname + '_log'] = np.log(data[colname]+1)
        else:
            data[colname + '_log'] = np.log(data[colname])
    data.info()
log_transform(data,['Kilometers_Driven','Price'])
#Log transformation of the feature 'Kilometers_Driven'
sns.distplot(data["Kilometers_Driven_log"], axlabel="Kilometers_Driven_log");
```





EDA Bivariate Analysis

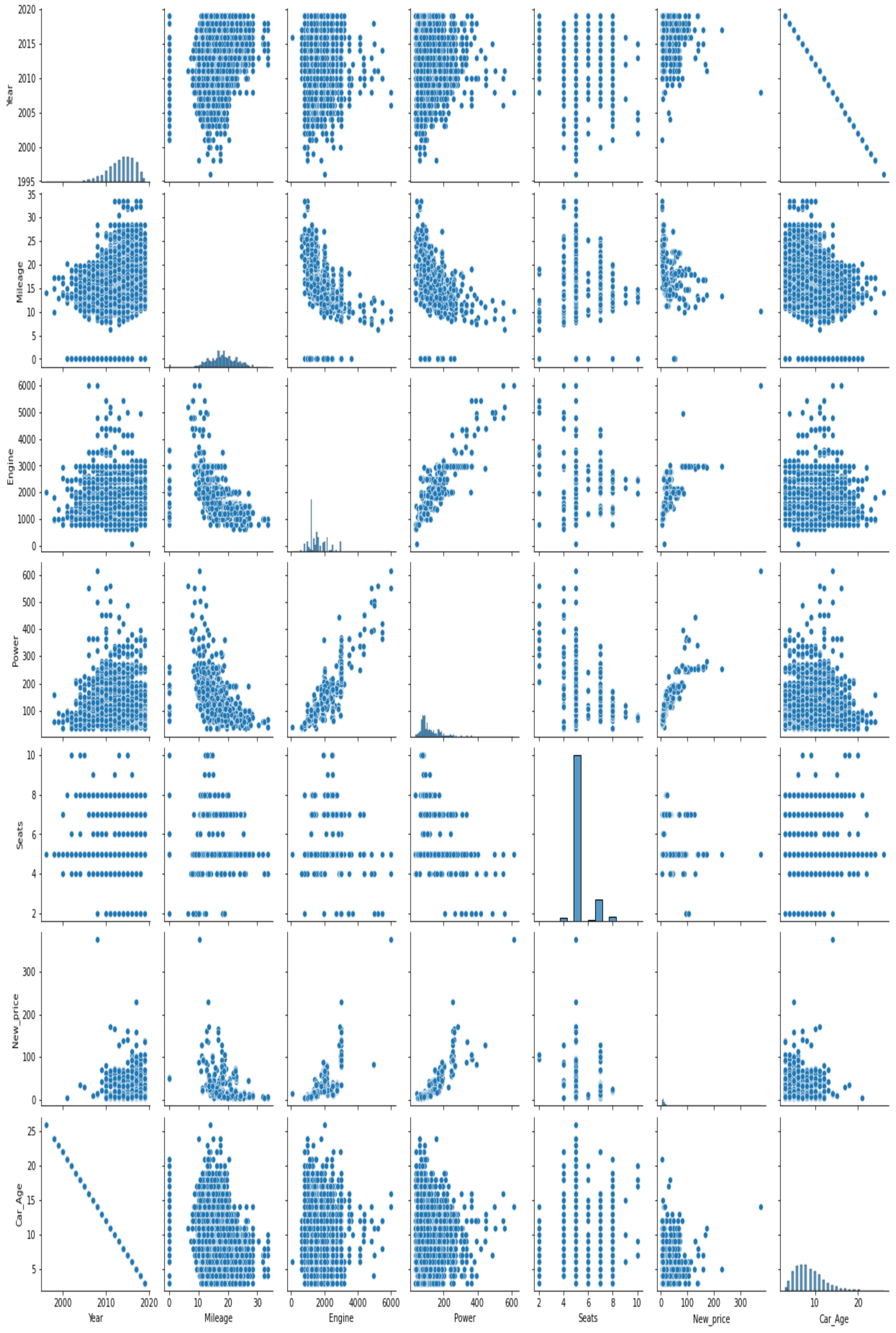
Now, let’s move ahead with bivariate analysis. Bivariate Analysis helps to understand how variables are related to each other and the relationship between dependent and independent variables present in the dataset.

For Numerical variables, Pair plots and Scatter plots are widely been used to do Bivariate Analysis.

A Stacked bar chart can be used for categorical variables if the output variable is a classifier. Bar plots can be used if the output variable is continuous

In our example, a pair plot has been used to show the relationship between two Categorical variables.

```
plt.figure(figsize=(13,17))
sns.pairplot(data=data.drop(['Kilometers_Driven','Price'],axis=1))
plt.show()
```


Year
Mileage
Engine
Power
Seats
New_price
Car_Age

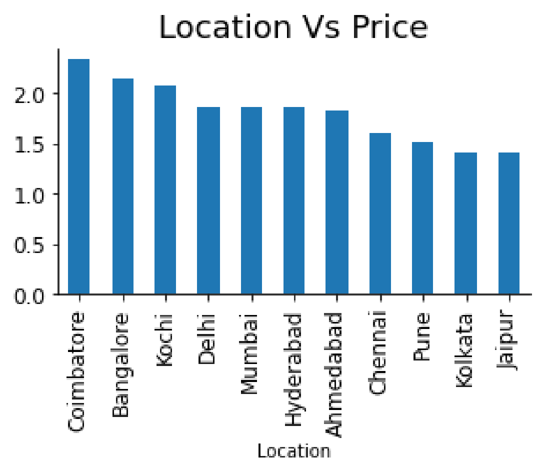
Pair Plot provides below insights:

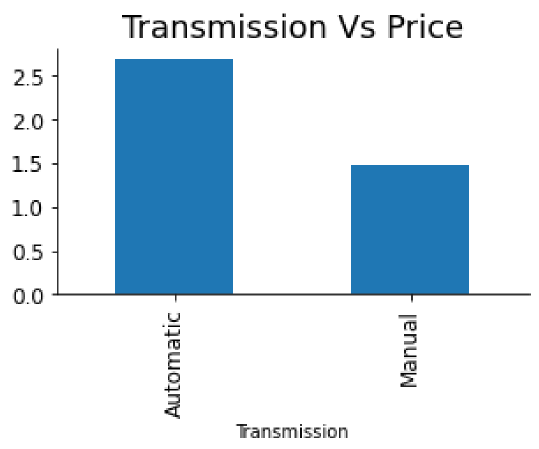
- The variable Year has a positive correlation with price and mileage
- A year has a Negative correlation with kilometers-Driven
- Mileage is negatively correlated with Power
- As power increases, mileage decreases
- Car with recent make is higher at prices. As the age of the car increases price decreases
- Engine and Power increase, and the price of the car increases

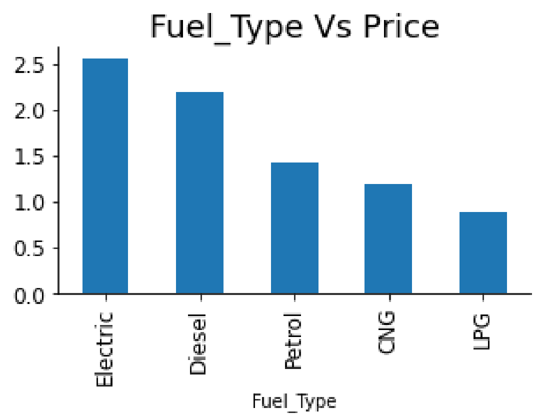
A bar plot can be used to **show the relationship between Categorical variables and continuous variables**

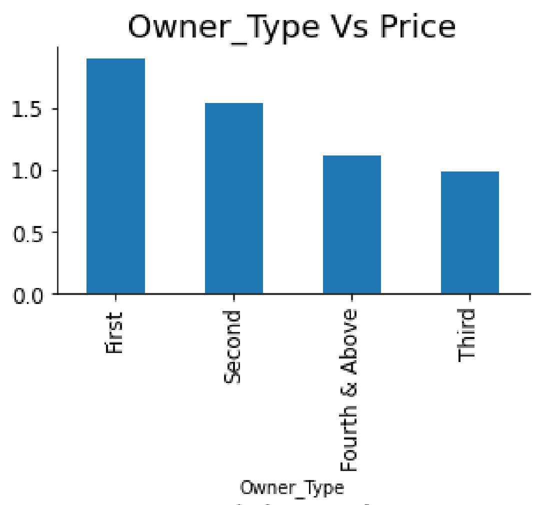
```
fig, axarr = plt.subplots(4, 2, figsize=(12, 18))

data.groupby('Location')['Price_log'].mean().sort_values(ascending=False).plot.bar(ax=axarr[0][0], fontsize=12)

axarr[0][0].set_title("Location Vs Price", fontsize=18)

data.groupby('Transmission')['Price_log'].mean().sort_values(ascending=False).plot.bar(ax=axarr[0][1], fontsize=12)

axarr[0][1].set_title("Transmission Vs Price", fontsize=18)

data.groupby('Fuel_Type')['Price_log'].mean().sort_values(ascending=False).plot.bar(ax=axarr[1][0], fontsize=12)

axarr[1][0].set_title("Fuel_Type Vs Price", fontsize=18)

data.groupby('Owner_Type')['Price_log'].mean().sort_values(ascending=False).plot.bar(ax=axarr[1][1], fontsize=12)

axarr[1][1].set_title("Owner_Type Vs Price", fontsize=18)

data.groupby('Brand')['Price_log'].mean().sort_values(ascending=False).head(10).plot.bar(ax=axarr[2][0], fontsize=12)

axarr[2][0].set_title("Brand Vs Price", fontsize=18)

data.groupby('Model')['Price_log'].mean().sort_values(ascending=False).head(10).plot.bar(ax=axarr[2][1], fontsize=12)

axarr[2][1].set_title("Model Vs Price", fontsize=18)

data.groupby('Seats')['Price_log'].mean().sort_values(ascending=False).plot.bar(ax=axarr[3][0], fontsize=12)

axarr[3][0].set_title("Seats Vs Price", fontsize=18)
```

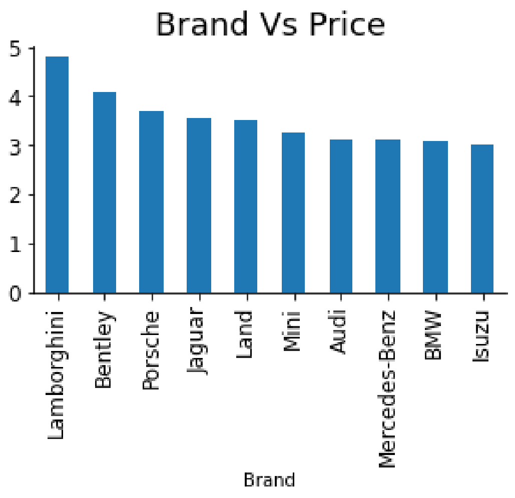
```
data.groupby('Car_Age')['Price_log'].mean().sort_values(ascending=False).plot.bar
(ax=axarr[3][1], fontsize=12)

axarr[3][1].set_title("Car_Age Vs Price", fontsize=18)

plt.subplots_adjust(hspace=1.0)

plt.subplots_adjust(wspace=.5)

sns.despine()
```

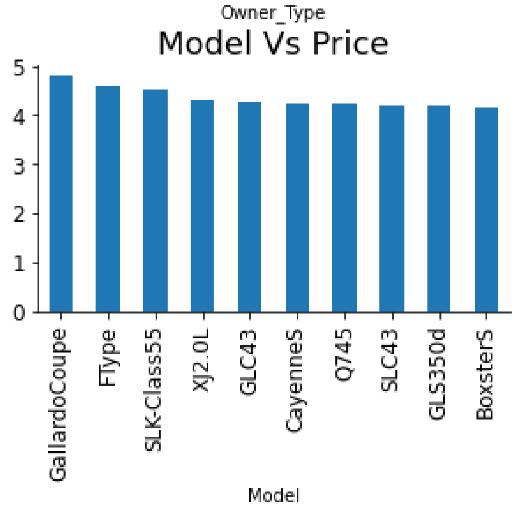

Location Vs Price
0.0
0.5
1.0
1.5
2.0
Coimbatore
Bangalore
Kochi
Delhi
Mumbai
Hyderabad
Ahmedabad
Chennai
Pune
Kolkata
Jaipur
Location

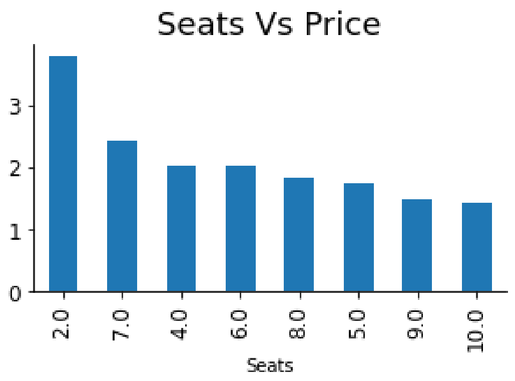

Transmission Vs Price
0.0
0.5
1.0
1.5
2.0
2.5
Automatic
Manual
Transmission

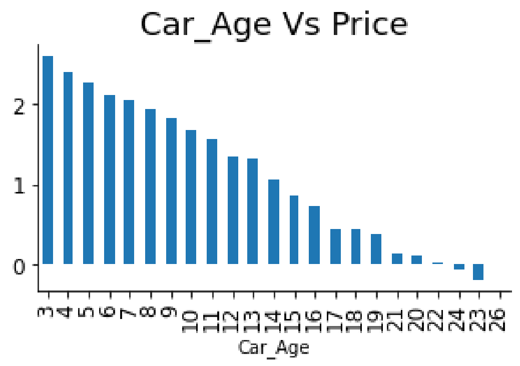

Fuel_Type Vs Price
0.0
0.5
1.0
1.5
2.0
2.5
Electric
Diesel
Petrol
CNG
LPG
Fuel_Type


Owner_Type Vs Price
0.0
0.5
1.0
1.5
First
Second
Fourth & Above
Third
Owner_Type


Brand Vs Price
0
1
2
3
4
5
Lamborghini
Bentley
Porsche
Jaguar
Land
Mini
Audi
Mercedes-Benz
BMW
Isuzu
Brand


Model Vs Price
0
1
2
3
4
5
GallardoCoupe
FType
SLK-Class55
XJ2.0L
GLC43
CayenneS
Q745
SLC43
GLS350d
BoxsterS
Model


Seats Vs Price
0
1
2
3
2.0
7.0
4.0
6.0
8.0
5.0
9.0
10.0
Seats


Car_Age Vs Price
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
21
20
22
24
23
26
Car_Age

Observations

- The price of cars is high in Coimbatore and less price in Kolkata and Jaipur
- Automatic cars have more price than manual cars.
- Diesel and Electric cars have almost the same price, which is maximum, and LPG cars have the lowest price
- First-owner cars are higher in price, followed by a second
- The third owner's price is lesser than the Fourth and above
- Lamborghini brand is the highest in price
- Gallardocoupe Model is the highest in price
- 2 Seater has the highest price followed by 7 Seater
- The latest model cars are high in price

EDA Multivariate Analysis

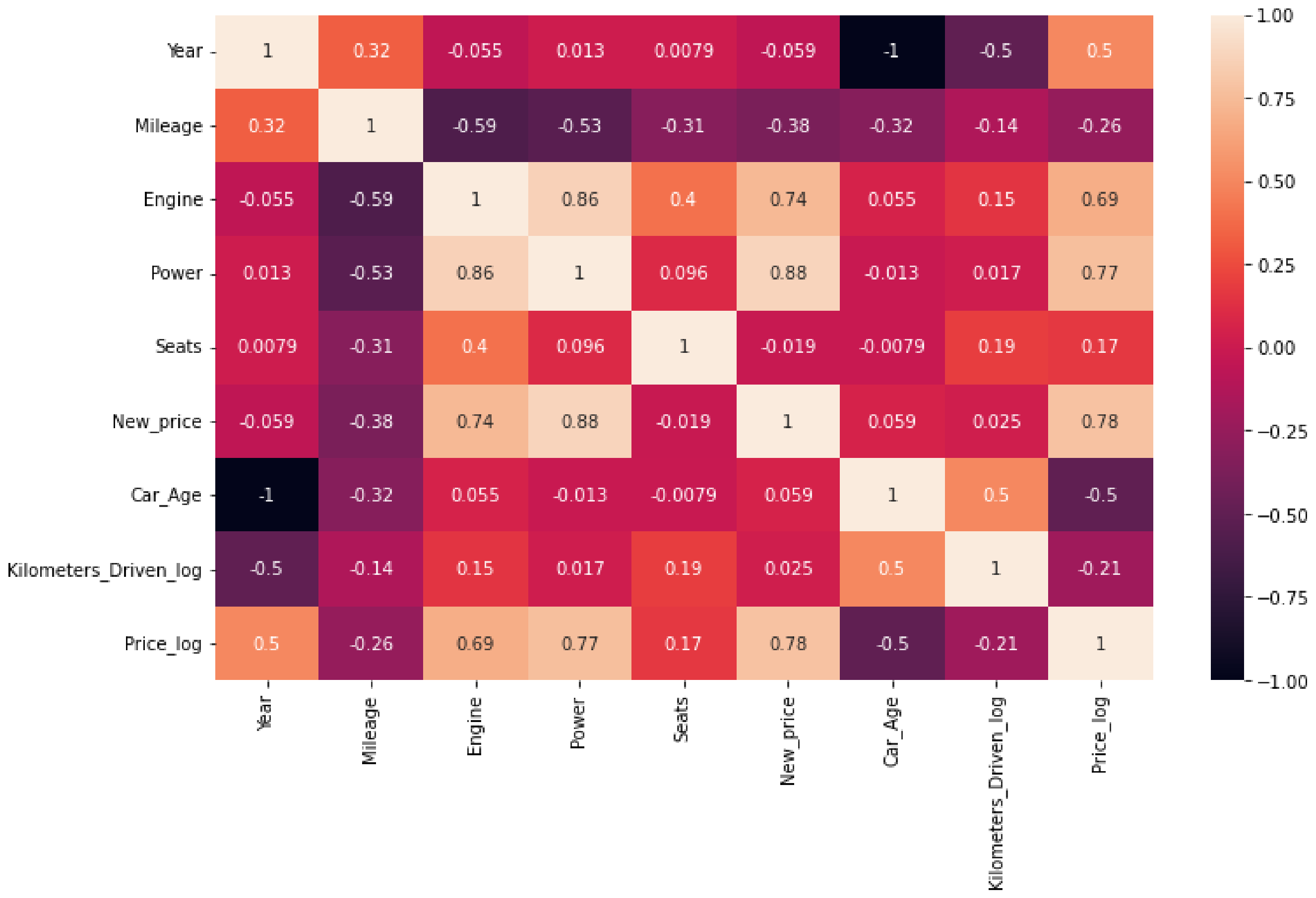
As the name suggests, Multivariate analysis looks at more than two variables. Multivariate analysis is one of the most useful methods to determine relationships and analyze patterns for any dataset.

A heat map is widely been used for Multivariate Analysis

Heat Map gives the correlation between the variables, whether it has a positive or negative correlation.

In our example heat map shows the correlation between the variables.

```
plt.figure(figsize=(12, 7))
sns.heatmap(data.drop(['Kilometers_Driven','Price'],axis=1).corr(), annot = True,
vmin = -1, vmax = 1)
plt.show()
```



From the Heat map, we can infer the following:

- The engine has a strong positive correlation to Power 0.86
- Price has a positive correlation to Engine 0.69 as well Power 0.77
- Mileage has correlated to Engine, Power, and Price negatively
- Price is moderately positive in correlation to year.
- Kilometer driven has a negative correlation to year not much impact on the price
- Car age has a negative correlation with Price
- car Age is positively correlated to Kilometers-Driven as the Age of the car increases; then the kilometer will also increase of car has a negative correlation with Mileage this makes sense

Impute Missing values

Missing data arise in almost all statistical analyses. There are many ways to impute missing values; we can impute the missing values by their **Mean**, **median**, most frequent, or zero values and use advanced imputation algorithms like **KNN**, **Regularization,** etc.

We cannot impute the data with a simple Mean/Median. We must need business knowledge or common insights about the data. If we have domain knowledge, it will add value to the imputation. Some data can be imputed on assumptions.

In our dataset, we have found there are missing values for many columns like Mileage, Power, and Seats.

We observed earlier some observations have zero Mileage. This looks like a data entry issue. We could fix this by filling null values with zero and then the mean value of Mileage since Mean and Median values are nearly the same for this variable chosen Mean to impute the values.

```
data.loc[data["Mileage"]==0.0,'Mileage']=np.nan
data.Mileage.isnull().sum()
data['Mileage'].fillna(value=np.mean(data['Mileage']),inplace=True)
```

Similarly, imputation for Seats. As we mentioned earlier, we need to know common insights about the data.

Let’s assume some cars brand and Models have features like Engine, Mileage, Power, and Number of seats that are nearly the same. Let’s impute those missing values with the existing data:

```
data.Seats.isnull().sum()
data['Seats'].fillna(value=np.nan,inplace=True)
data['Seats']=data.groupby(['Model','Brand'])['Seats'].apply(lambda
x:x.fillna(x.median()))
data['Engine']=data.groupby(['Brand','Model'])['Engine'].apply(lambda
x:x.fillna(x.median()))
data['Power']=data.groupby(['Brand','Model'])['Power'].apply(lambda
x:x.fillna(x.median()))
```

In general, there are no defined or perfect rules for imputing missing values in a dataset. Each method can perform better for some datasets but may perform even worse. Only practice and experiments give the knowledge which works better.