Exploratory Data Analysis

In [16]:

```
# Importing Libraries :

# Import pandas library and make it as pd:
import pandas as pd

# Import Numerical Python Library and make it as np :
import numpy as np

# Import pyplot from Matplotlib Library and make it as plt:
import matplotlib.pyplot as plt


# Import Seaborn Library and Make it as SNS:
import seaborn as sns

# Inorder to Supress the Warnings Import FilterWarnings:
from warnings import filterwarnings
filterwarnings('ignore')
```

In [ ]:

```

```

$What - is - Data?$

Data is units of information in structured or unstructured format.

- Collection of relevant tweets
- Records of yield in a farm over a period of time
- Records of stock price every minute
- Records of performance of a sports person

$CLASSIFICATION - OF - DATA$:

1. Time Series Data

2. Cross Sectional Data

3. Pooled Data

4. Panel Data

In [ ]:
```
1
```

In [ ]:
```
1
```

1. Time Series Data

- Time series data is the set of observations on a variable at different time points
- The data may be collected daily, weekly, monthly, or annually

Time-series data

Year	Numbers of cars rented by alamo
2000	10000
2001	8000
2002	11000
2003	5000
2004	12000

	A	B	C
1	Year	Quarter	Sales
2	2012	1	$165,000.00
3		2	$253,000.00
4		3	$316,000.00
5		4	$287,000.00
6	2013	1	$257,000.00
7		2	$308,000.00
8		3	$376,000.00
9		4	$351,000.00

2. Cross Sectional Data

Cross-sectional data are set of observations on two or more variables at the same time point

City	Date	MaxTemperature	Humidity	Wind
NYC	1/1/2015	55	45%	4 mph
NYC	1/1/2014	30	39%	16 mph
NYC	1/1/2013	47	65%	21 mph
SFO	1/1/2015	70	35%	21 mph
SFO	1/1/2014	75	23%	2 mph
SFO	1/1/2013	71	39%	13 mph
Boston	1/1/2015	34	39%	16 mph
Boston	1/1/2014	26	17%	27 mph
Boston	1/1/2013	45	46%	18 mph

3. Pooled Data

Pooled data is combination of both time series data and cross sectional data.

Test questions	Original *n*	Original *M*	Original *SE*	Pooled *M*	Pooled *SE*	χ^2	Asymptotic significant[a]	Exact significant (two-tailed)[b]
Anchor Item Q1	290	0.272	0.026	0.297	0.027	2.828	.985	1.000
Anchor Item Q2	285	0.179	0.023	0.204	0.028	7.142	.712	1.000
Anchor Item Q3	252	0.202	0.025	0.215	0.032	9.938	.446	1.000
Anchor Item Q4	250	0.304	0.029	0.340	0.037	11.377	.329	1.000
Anchor Item Q5	271	0.472	0.030	0.489	0.030	2.488	.991	1.000
Anchor Item Q6	259	0.467	0.031	0.473	0.032	3.517	.967	1.000
Anchor Item Q7	282	0.298	0.027	0.332	0.031	5.493	.856	1.000
Anchor Item Q8	282	0.294	0.027	0.316	0.030	4.749	.907	1.000
Anchor Item Q9	290	0.172	0.022	0.207	0.030	9.886	.451	1.000

[a]Kruskal–Wallis test, $df = 10$.
[b]Binomial distribution used for McNemar.

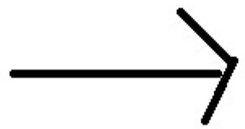
4. Panel Data

- Panel data is type of pooled data
- The same cross-sectional unit is surveyed over time
- Also known as longitudinal or micro-panel data

Person ID	Age	Sex	Year	*Income*
1	27	1	2015	1600
1	28	1	2016	1500
2	42	2	2015	1900
2	43	2	2016	2000
2	44	2	2017	2100
3	34	1	2015	3300

G35

	A	B	C	D
1		FIRM A	FIRM B	FIRM C
2	3/01/1995	0.191439	0.780436	0.329861
3	4/01/1995	-0.04008	0.332088	0.306904
4	5/01/1995	-0.14042	0.681645	0.820281
5	6/01/1995	-5.65741	0.272911	0.637198
6	9/01/1995	5.898009	0.212024	0.173159
7	10/01/1995	-3.1429	0.719781	0.078806
8	11/01/1995	-6.70905	0.230741	0.090504
9	12/01/1995	6.139131	0.201295	0.409763
10	13/01/1995	-3.19934	0.635427	0.561843
11	16/01/1995	3.282443	0.636448	0.43751
12	17/01/1995	-0.05193	0.655553	0.791657
13	18/01/1995	15.7144	0.013025	0.057594
14	19/01/1995	11.65196	0.535783	0.657062
15	20/01/1995	0.220995	0.65229	0.872535
16	23/01/1995	-14.755	0.092412	0.171942
17	24/01/1995	2.882044	0.68886	0.360246
18	25/01/1995	0.186261	0.544323	0.677453
19				

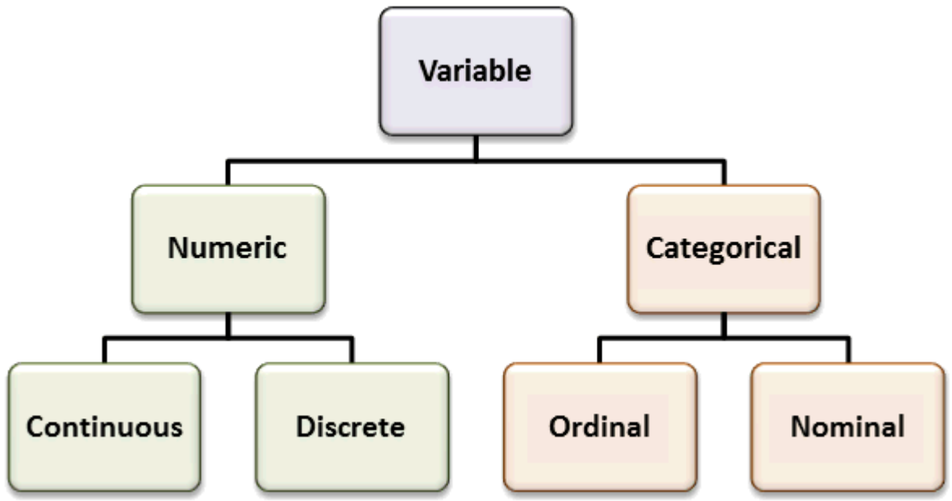


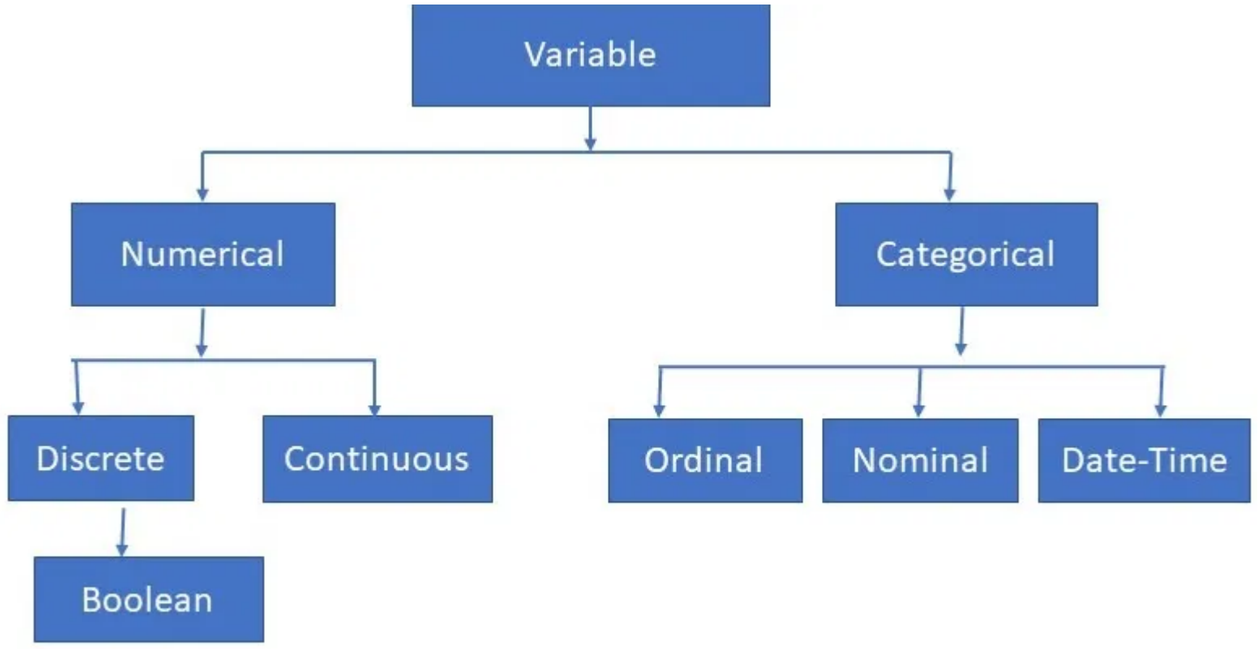
3/01/1995	FIRM A	0.821848	
4/01/1995	FIRM A	-0.04008	
5/01/1995	FIRM A	-0.14042	
6/01/1995	FIRM A	-5.65741	
9/01/1995	FIRM A	5.898009	
10/01/1995	FIRM A	-3.1429	
11/01/1995	FIRM A	-6.70905	
3/01/1995	FIRM B	6.139131	
4/01/1995	FIRM B	-3.19934	
5/01/1995	FIRM B	3.282443	
6/01/1995	FIRM B	-0.05193	
9/01/1995	FIRM B	15.7144	
10/01/1995	FIRM C	11.65196	
11/01/1995	FIRM C	0.220995	
23/01/1995	FIRM C	-14.755	
24/01/1995	FIRM C	2.882044	
25/01/1995	FIRM C	0.186261	

original **panel form I want**

Understand the variable:

To understand the descriptive statistics, we need to first understand the type of each variable or column





Categorical Variable

Nominal Data:

- Nominal data has no order and has two or more than two categories
- It represents discrete units and are used to label variables
- Example: Housing type - Apartment, Bungalow, Penthouse

Binary Data:

- Binary data has no order and has strictly two categories
- Also known as dichotomous variable
- Example: Presence or absence of a diseaseThis

Categorical Variable

Ordinal Data:

- Ordinal data is ordered nominal data
- It represents discrete and ordered
- Example: Education - Primary, Secondary, High School, College and University which are ordered

IMPORTANT:

- At times data may appear to be numeric but is actually categorical
- Consider the adjoining table. The ‘Gender’ column has value 1 and 0
- The numeric values used to represent a categorical variable cannot be used for numerical computation

In [12]:
```
import pandas as pd
```

In [14]:
```
_dict_ = {'Name ' : ['Riya', 'Arya', 'Sam', 'Joe', 'Harry', 'Sarah'],
          'Gender' : [0, 0, 1, 1, 1, 0],
          'Age' : [45, 23, 34, 54, 12, 43]}

DataFrame = pd.DataFrame(_dict_)
DataFrame
```

Out[14]:

	Name	Gender	Age
0	Riya	0	45
1	Arya	0	23
2	Sam	1	34
3	Joe	1	54
4	Harry	1	12
5	Sarah	0	43

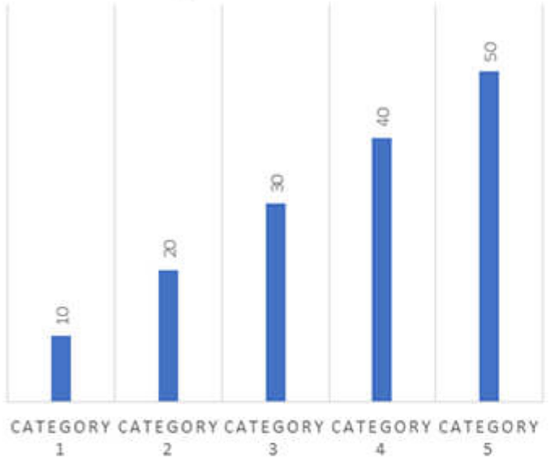
Data Analysis:

- **Descriptive**
- **Inferential**

Descriptive:

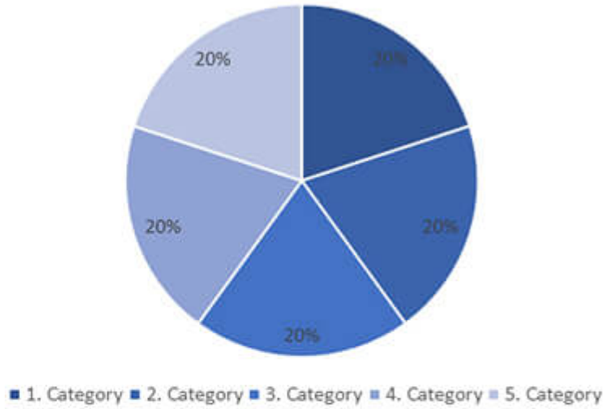
- Descriptive statistics is the term given to the analysis of the data that helps describe, visualize or summarize the data
- Helps exhibit the patterns in the data
- However, descriptive statistics only help in understanding the data and not make conclusions regarding any hypothesis
- We can not generalise the facts, they are confined only to the data at hand (that may be the sample)

Descriptive Statistics



- Describes the characteristics of data
- **Example:** Population, Frequency of the variables

Inferential Statistics

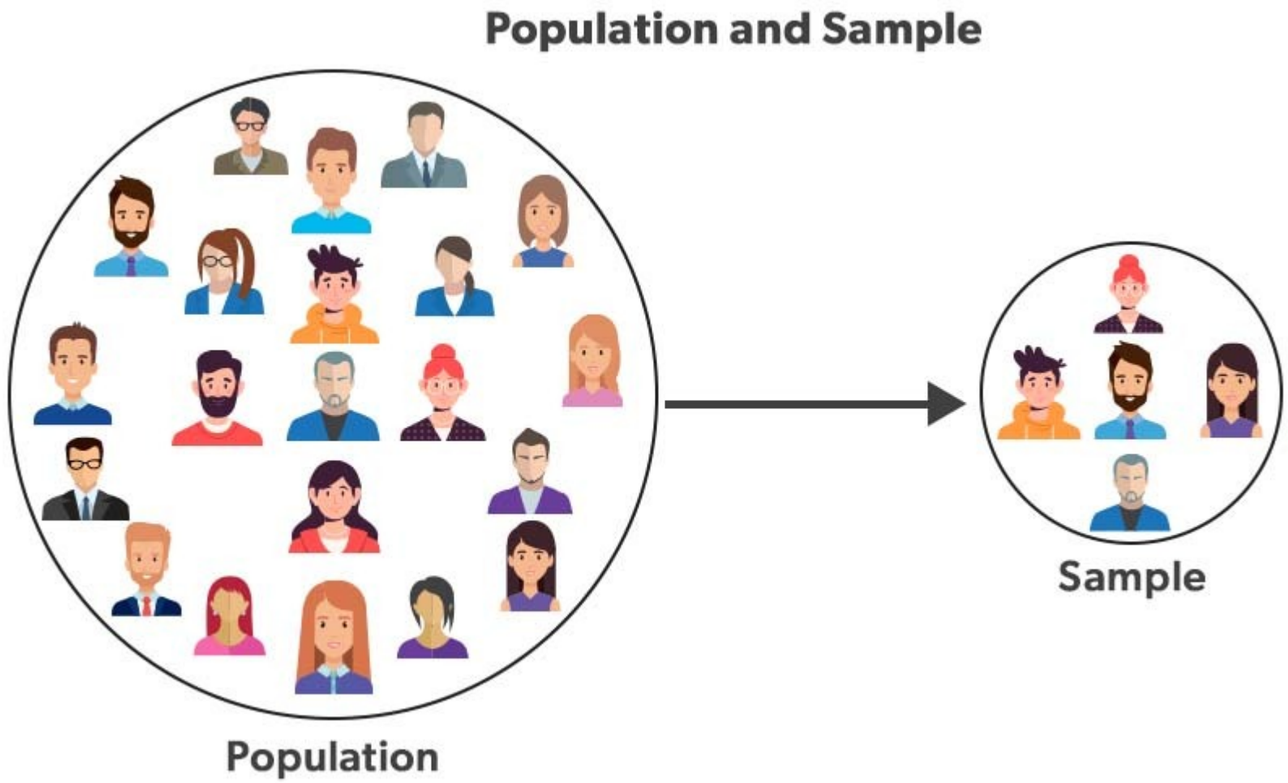


- Studies the sample of the same data
- **Example:** Grade, Percentile

Population & Sample:

● Population is a collection of all the individuals or objects

● Sample is a subset of the population which is a representative of the population



Example:

Suppose a company producing electric bulbs wants to know the average life of a bulb. If the details on average life of all bulbs are available, then this information is regarded as the population. It would be challenging for the company to test each and every bulb produced. In such a scenario, the company would draw a sample from the produced bulbs to test.

Descriptive statistics

The major characteristics we observe in each variable of the data set are:

1. Measures of central tendency
2. Distribution of the data
3. Measures of dispersion

Measures of Central Tendency

● A measure of central tendency is a single value that identifies the central position of the data

● It includes mean, median, mode and partition values

Mean

● Mean is a central tendency of the data

● It is defined as the sum of all observations divided by the number of observations: where N is the total number of observations

● The whole data is spread out around the mean

Mean

Merits:

● Based on all observation

Demerits:

● Affected by extreme observations (outliers)

● Arithmetically it is not possible to obtain the mean of a data with missing values

● Mean can not be calculated for categorical data

Outlier

● An outlier is a value that behaves differently than other observations

● It does not not follow the usual pattern

Obtain the mean

Each column of a DataFrame is a pandas Series.

Let us create a series and find its mean.

In [ ]:
```
1
```

Trimmed mean

● Arithmetic mean obtained by ignoring lowest and highest α% observations is called as the α% - trimmed mean

● Using a trimmed mean, it is possible to eliminate the influence of outliers

● Note the observations are arranged either in increasing or decreasing order

● Also called as truncated mean

In [2]:
```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
```

In [76]:
```
1 prices_missing = pd.Series([1, 1, 4, 56, 7,8 ,90, 34, 67, 4, 76, 32, 8, 45, 35, 87,
2                             np.NaN, 89, 65, 90, 74, 88, 34, 99, 88, 67])
3
4 print('Mean : ', prices_missing.mean())
5 print('Median : ', prices_missing.median())
```

```
Mean :  49.96
Median :  56.0
```

In [14]:
```
1 # Obtain a 20% of trimmed Mean:
2
3 import scipy
4 from scipy import stats
5
6 scipy.stats.trim_mean(prices_missing, proportiontocut = 0.20)
```

Out[14]: 54.0

In [ ]:
```
1
```

Median

● Median is the middle most observation in the data when it is arranged either in ascending or descending order of their values

● It divides the data into two equal parts. Thus, it is a positional average

● Median will be the middle term, if the number of observations is odd

● Median will be average of middle two terms, if number of observations is eve

Obtain the median

Using mean and median

● For a numeric variable, the missing value is replace by the mean if there are no outliers present

- In case if outliers are present the missing values are replaced by the median
- For symmetric data median = mean, so either of the value can be used

In [ ]:
```
1
```

Mode:

- Mode of the data is the value which has the highest frequency
- In other words, it is most repeated observation
- A set of observations having two modes is said to be bimodal
- A set of observations may have more than two modes. Such data is said to be multimodal

Mode Merits:

- It is applicable to both numeric and categorical data
- It is not affected by extreme observations

Demerits:

- It is not based on all observations

In [ ]:
```
1
```

OBTAIN MODE:

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [61]:
```
# Create a Mode:

prices_pens = pd.Series([0, 35, 67, 34, 10, 56, 100, 67,
                         99, 45, 100, 45, 36, 99, 56])

prices_pens.value_counts()
```

Out[61]:
```
67     2
56     2
100    2
99     2
45     2
0      1
35     1
34     1
10     1
36     1
dtype: int64
```

In [62]:
```
# print the Mode :

prices_pens.mode()
```

Out[62]:
```
0     45
1     56
2     67
3     99
4    100
dtype: int64
```

Measure of Dispersion:

- The measure of Dispersion refers to the Variability with the Data.
- Variability is the Measure of how Close or Far the Data from the Central Value.
- Reliability of a Measure of Central Tendency is more dispersion in the Data is less.
- It includes Range, variance, Standard Deviation, Coefficient of Variation and Interquartile Range.

Range:

- Range is the Differernce B/w the Largest and Smallest Observation.
- Range = Xn - X1

- Where Xn is the Largest Value and X1 is the Smallest value.

Merits:

- It is a basic way to measure the Variability

Demerits:

- If Either of the Smallest or the Largest Values turns out to be an Outlier then the Range Value is Unreliable.
- The Other in-Between Observation do not affect the range Data.
- It Doesn't give the Clear picture on the Data.

Obtain the Range:

In [63]:
```
year_establishment = pd.Series((1955, 1990, 1930, 2000))

Range = year_establishment.max() - year_establishment.min()
Range
```

Out[63]:
```
70
```

Variance:

Variance is the arithmetic mean of squares of deviations taken from the mean.

Interpretation: The variance looks at how spread out the observations are from the mean. The higher the variance more the data is spread out.

Obtain the variance:

In [66]:
```
1 prices_pens.var()
```

Out[66]: 1044.6857142857143

The Unit of measurement of variance

- The unit of measurement of the variance is not the same as that of the original data
- Consider an industry producing screws. The diameter of each screw is noted in millimeters
- The variance in the diameter is measured in squared Millimeters
- Thus a usual practice is to take square root of the variance which would have the same unit as the original data
- The value thus obtained is called the standard deviation of the data

Interquartile range

- The interquartile range is defined as the difference between the third and the the first quartile
- It gives the range in which the middle 50% of the data lies
- The value of IQR is used to remove outliers in the data

In [78]:
```
1  # Obtain the First Quartiles:
2  Q1 = prices_missing.quantile(0.25)
3
4  # Obtain the Quantile:
5  Q3 = prices_missing.quantile(0.75)
6
7  IQR = Q3 - Q1
8
9  # Print the IQR:
10 IQR
```

Out[78]: 79.0

Interpretation : The InterQuartile range of prices_missing is 79.0

Z-score

- Every value in the dataset has the z-score
- To find the z-score of a value, subtract the mean from it and divide it by the standard
- The z-scores are extensively used in statistics especially in hypothesis testing
- It is also used to normalize the data, ie is it scales the data to get the mean of the scaled data as 0 and variance as 1

The z-score tells how much is the value away from the mean

```
0 ---->> The corresponding value is same as the mean

< 0 ---->>  The corresponding value is less than the mean

>  0 ---->> The corresponding value is greater than the mean

< -3 or > 3 ---->>  The corresponding value is potential outlier
```

Covariance

- Covariance is a measure of how much two random variables vary together
- Covariance is similar to variance, but variance explains how a single variable varies, and covariance explains how two variables vary together
- It cannot tell whether the value indicates a strong relationship or weak relationship, since the covariance can take any value

In [16]:
```
# Create Imports Series:
imports_raw_material = pd.Series([10, 11, 14, 15, 20, 22, 16, 12, 15, 13])

# Create Export Series:
export_finished_products = pd.Series([11, 12 ,14, 18, 21, 26, 44, 34, 12])

# Calculate the Covariance:
imports_raw_material.cov(export_finished_products)
```

Out[16]: 14.0

Correlation

- Correlation is the extent of linear relationship among numeric variables
- It indicates the extent to which two variables increase or decrease in parallel
- The value of a correlation coefficient ranges between -1 and 1
- A positive correlation exists if variable increase/decrease simultaneously
- A negative correlation exists if one variable increases, while the other variable decreases

In [17]:
```
imports_raw_material.corr(export_finished_products)
```

Out[17]: 0.30909991855425206

In [ ]:

In [ ]:

In [ ]:

HANDS ON PRACTISE WITH mpg Dataset

Let us now do hands on practise with dataset

- We will work on mpg dataset
- mpg stands for miles per gallon
- The data is generally used to predict the miles per gallon of a car

In [7]:
```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
```

In [9]:
```
1 # Import the Mpg Data set:
2 df_mpg = pd.read_csv('auto-mpg.csv')
```

Problem Statement

The data is generally used to predict the miles per gallon of a car

In [10]:
```
1 # Display head of the dataset:
2 # Head() " Displays the First Five rows:
3 df_mpg.head()
```

Out[10]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

In [11]:
```
1 # Tail () " Displays the last Five rows:
2 df_mpg.tail()
```

Out[11]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
393	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup
395	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage
396	28.0	4	120.0	79	2625	18.6	82	1	ford ranger
397	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10

In [12]:
```
# Entire Dataset:

df_mpg
```

Out[12]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup
395	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage
396	28.0	4	120.0	79	2625	18.6	82	1	ford ranger
397	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10

398 rows × 9 columns

Interpretation : **There are (398 Rows and 9 Columns) has been presented in this DataSet.**

Check the Size and Variable Types:

In [27]:
```
# Check the Size of the Data:
df_mpg.shape
```

Out[27]:
```
(398, 9)
```

In [28]:
```
# Check the Variable Type :
df_mpg.dtypes
```

Out[28]:
```
mpg             float64
cylinders         int64
displacement    float64
horsepower       object
weight            int64
acceleration    float64
model year        int64
origin            int64
car name         object
dtype: object
```

In [29]:
```
# Information about the Dataset:
df_mpg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64  
 2   displacement  398 non-null    float64
 3   horsepower    398 non-null    object 
 4   weight        398 non-null    int64  
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64  
 7   origin        398 non-null    int64  
 8   car name      398 non-null    object 
dtypes: float64(3), int64(4), object(2)
memory usage: 28.1+ KB
```

In [13]:
```
# Check for Duplicate Values in the DataSet:

df_mpg.duplicated().sum().sum()
```

Out[13]: 0

In [41]:
```
# Check for Null Values in our DataSet:
df_mpg.isnull().sum().sum()
```

Out[41]: 0

In [20]:
```
# Descriptive Statistics:

df_mpg.describe()
```

Out[20]:

	mpg	cylinders	displacement	weight	acceleration	model year	origin
count	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000
mean	23.514573	5.454774	193.425879	2970.424623	15.568090	76.010050	1.572864
std	7.815984	1.701004	104.269838	846.841774	2.757689	3.697627	0.802055
min	9.000000	3.000000	68.000000	1613.000000	8.000000	70.000000	1.000000
25%	17.500000	4.000000	104.250000	2223.750000	13.825000	73.000000	1.000000
50%	23.000000	4.000000	148.500000	2803.500000	15.500000	76.000000	1.000000
75%	29.000000	8.000000	262.000000	3608.000000	17.175000	79.000000	2.000000
max	46.600000	8.000000	455.000000	5140.000000	24.800000	82.000000	3.000000

In [22]:
```
# Descriptive Stats for Cat Col:

df_mpg.describe(include = 'object')
```

Out[22]:

	horsepower	car name
count	398	398
unique	94	305
top	150	ford pinto
freq	22	6

In [29]:
```
# Skewness for my Entire DataSet:

from warnings import filterwarnings
filterwarnings('ignore')

df_mpg.skew()
```

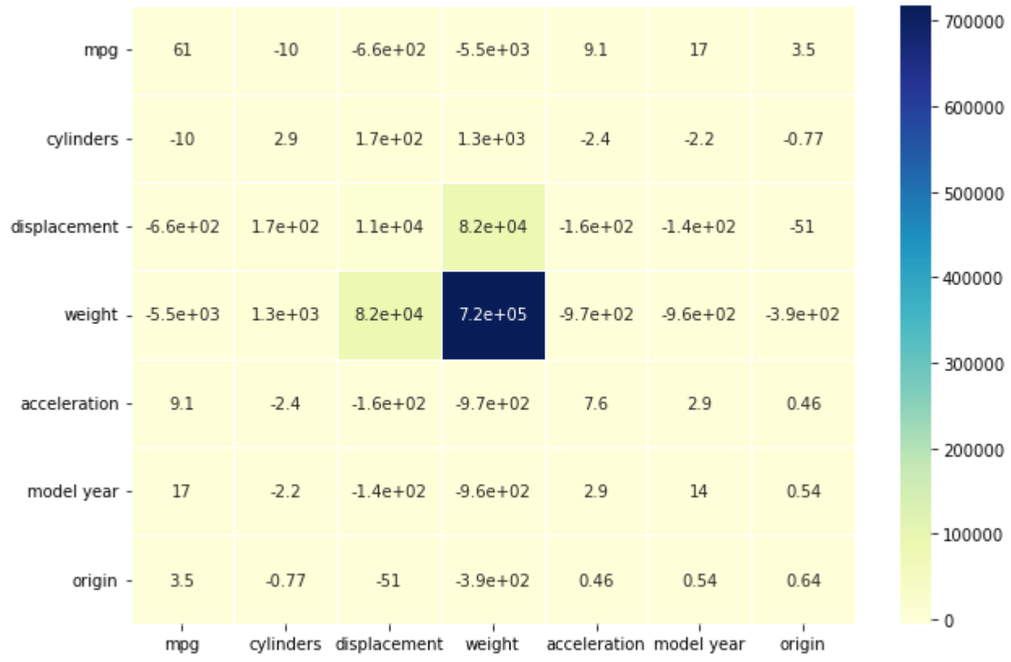
Out[29]:
```
mpg             0.457066
cylinders       0.526922
displacement    0.719645
weight          0.531063
acceleration    0.278777
model year      0.011535
origin          0.923776
dtype: float64
```

In [30]:
```
# Obtain kurt():

df_mpg.kurt()
```

Out[30]:
```
mpg            -0.510781
cylinders      -1.376662
displacement   -0.746597
weight         -0.785529
acceleration    0.419497
model year     -1.181232
origin         -0.817597
dtype: float64
```

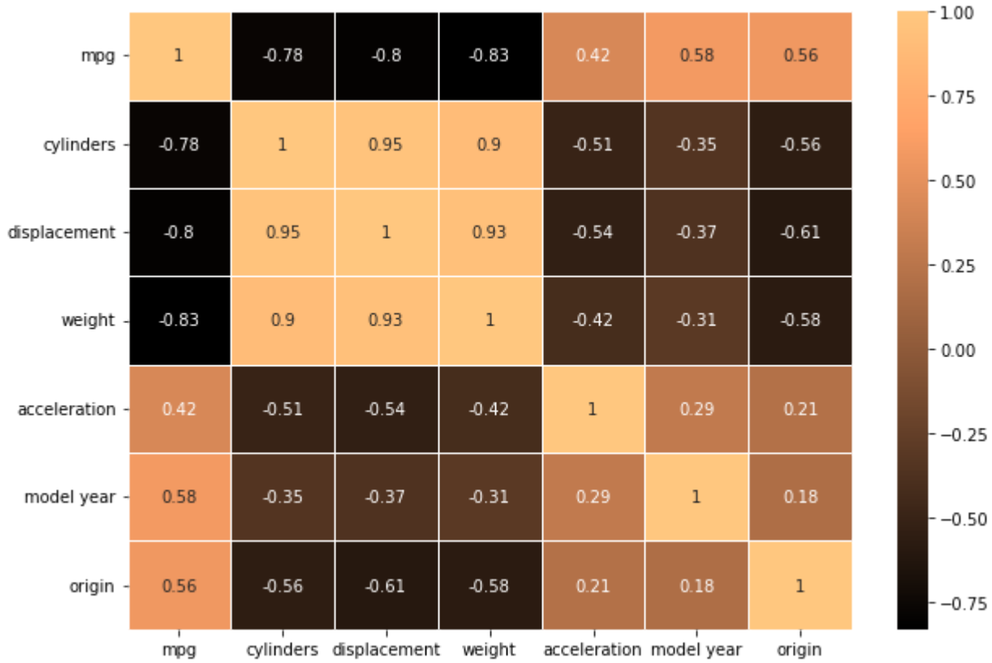
In [33]:

```
# Covariance:

cov = df_mpg.cov()
cov

# Set the plot Size:
fig, ax = plt.subplots(figsize = (10, 7))

# PLot a Heatmap for the Correlation Matrix
# annot : print Values in each Cell
# linewidths : sepcify width of the Line and Specifying the Plot
# vmin : Minimum Value of the Variable
# vmax : Maximum Value of the Variable
# Cmap : colour Code for the Plot
# fmt : set the Decimal place of the Annot
sns.heatmap(cov, annot = True, linewidths = 0.95,
            cmap = 'YlGnBu', fmt = '.2g')

# Display the Plot:
plt.show()
```



In [35]:

```
# Correlation:

corr = df_mpg.corr()
corr

# Set the plot Size:
fig, ax = plt.subplots(figsize = (10, 7))

# PLot a Heatmap for the Correlation Matrix
# annot : print Values in each Cell
# linewidths : sepcify width of the Line and Specifying the Plot
# vmin : Minimum Value of the Variable
# vmax : Maximum Value of the Variable
# Cmap : colour Code for the Plot
# fmt : set the Decimal place of the Annot
sns.heatmap(corr, annot = True, linewidths = 0.95,
            cmap = 'copper', fmt = '.2g')

# Display the Plot:
plt.show()
```



UNIVARIATE ANALYSIS

- Univariate Analysis is the Simplest Form of Statistical Analysis
- It is Study of a single (uni) variable
- The Key Purpose is to Describe the Data using different numerical and Visualization Techniques

1. Histogram
2. Density Plot
3. Box Plot
4. Violin plot
5. Cumulative Distribution

$Univariate--Analysis$:

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
df_car = pd.read_csv('auto-mpg.csv')
df_car.head()
```

Out[3]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

In [6]:
```
# Univariate Analysis:
```

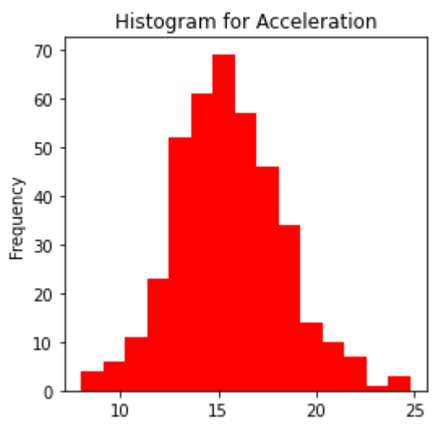
- Considering the Variable 'Acceleration'
- Check the Number of Observations in the Data using the len()

In [7]:
```
acceleration = df_car['acceleration']
```

In [8]:
```
len(acceleration)
```

Out[8]: 398

In [9]:
```
# Bins = 15, Creates 15 class Intervals
plt.hist(acceleration, bins = 15, color = 'r')

# Set the Title:
plt.title('Histogram for Acceleration')

# Set label for y - axis :
plt.ylabel('Frequency')

# Display the Plot:
plt.show()
```



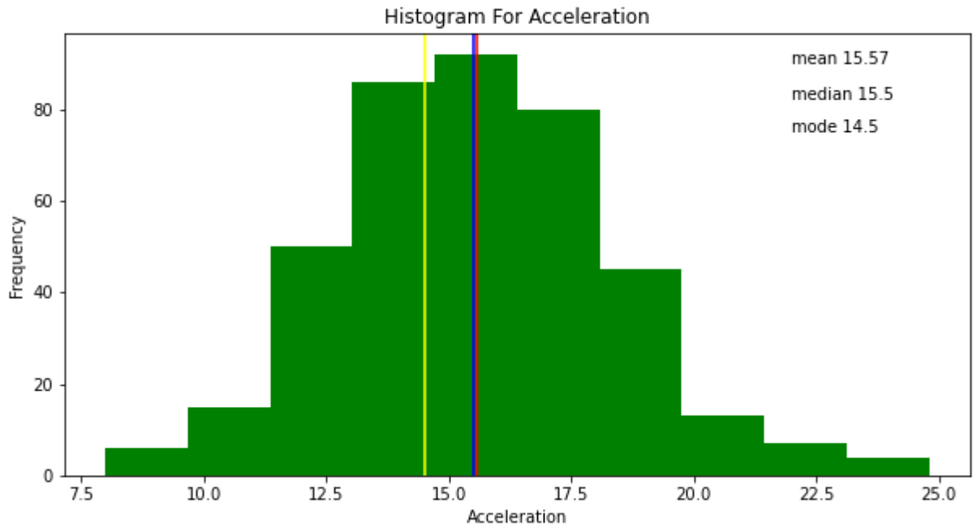
Interpretation : The Histogram divides the Data into Bins, counts the Data points in each bin

It shows the Bins on the x - axis and the Counts on the y - axis

The Variable "acceleration" has `Symmetrical Distribution`

In [10]:

```
# Set the Figure size:
plt.figure(figsize = (10, 5))

# Bins - 10, Creates the class Interval:
plt.hist(acceleration, bins = 10, color = 'g')

# PLot the Lines of the Mean, Median and Mode on my Histogram PLot:
# Specify Different colors for each line along by using the 'Color' Parameter
plt.axvline(acceleration.mean(), color = 'red')
plt.axvline(acceleration.median(), color = 'blue')
plt.axvline(acceleration.mode()[0], color = 'Yellow')


# Add the Values in the Plot:
plt.text(22, 90, 'mean'+' '+ str(round(acceleration.mean(),2)))
plt.text(22, 82, 'median'+' '+ str(round(acceleration.median(),2)))
plt.text(22, 75, 'mode'+' '+ str(round(acceleration.mode()[0],2)))

# Set title:
plt.title('Histogram For Acceleration')

# Set the Label of x- axis :
plt.xlabel('Acceleration')

# Set the Label for y - axis :
plt.ylabel('Frequency')

# Display the Plot:
plt.show()
```



In [2]:

```
# Import the Library:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

What is Density Plot:

- A density Plot is a Smoothened Version of Histogram Estimated from the Data
- The 'kde'(Kernel Density Estimator) is drawn at every individual data point and all of these curves are the added together to create a single smooth density estimation.

In [24]:

```
# Set the Figure Size:
plt.rcParams['figure.figsize'] =[10, 10]

# Distplot() - a plot for kernel Density Estimator:
sns.distplot(acceleration)

# Set the Title:
plt.title('Density plot for Acceleration')


# Display the PLot:
plt.show()

```

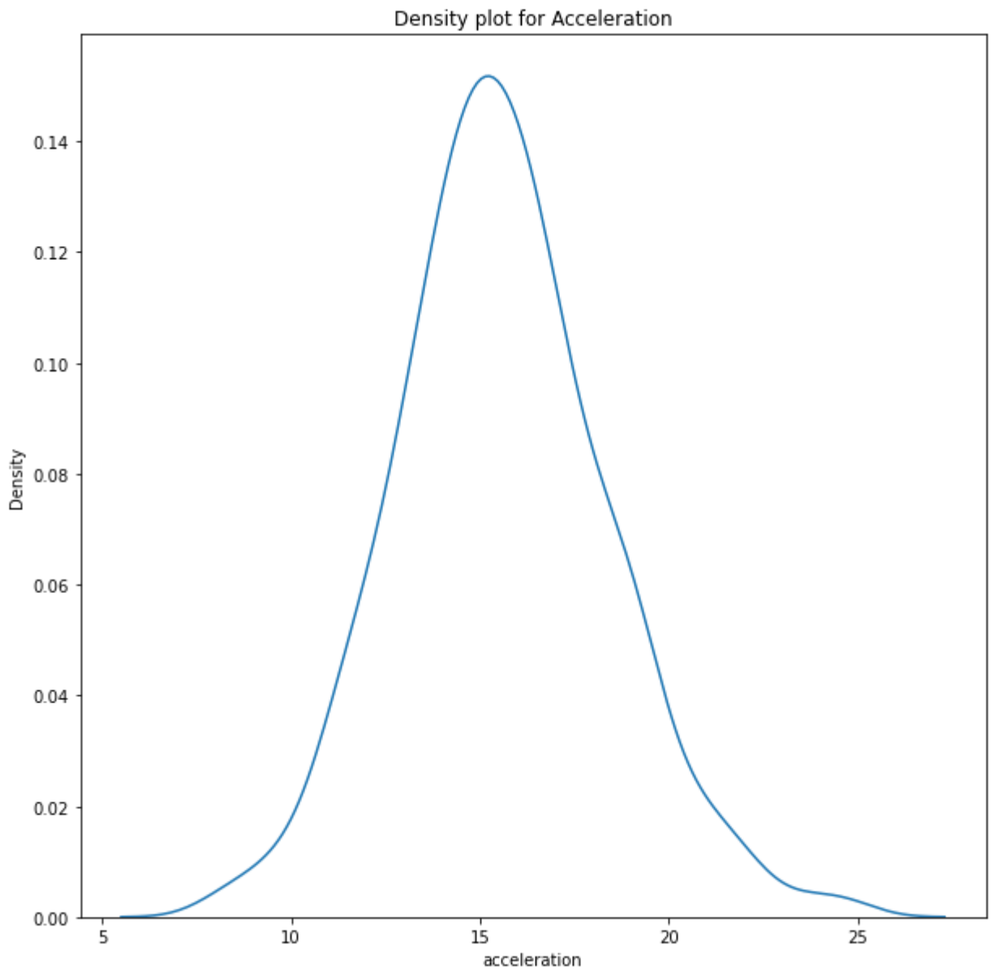
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_47640/296992581.py in <module>
      3
      4 # Distplot() - a plot for kernel Density Estimator:
----> 5 sns.distplot(acceleration)
      6
      7 # Set the Title:

NameError: name 'acceleration' is not defined
```

`Interpretation` : The Dist of a acc is near to normal

In [14]:

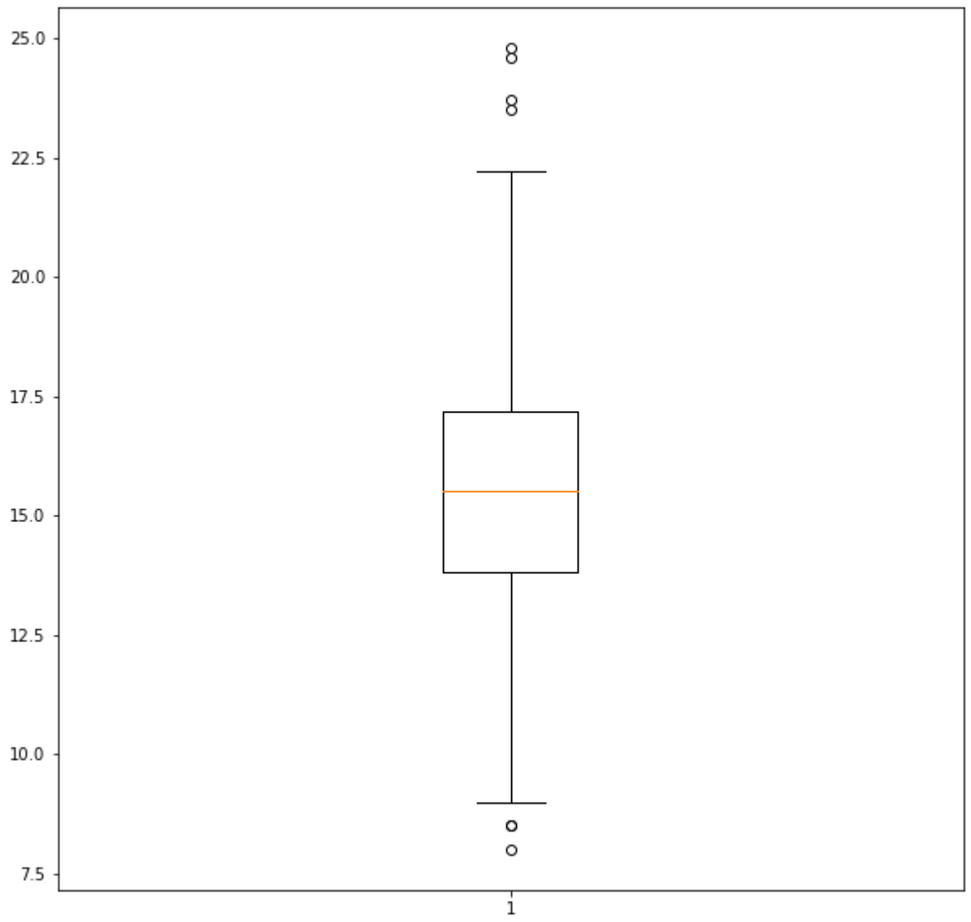
```
# Distplot without Histogram:

# Distplot() - a plot for kernel Density Estimator:
sns.distplot(acceleration, hist = False)

# Set the Title:
plt.title('Density plot for Acceleration')


# Display the PLot:
plt.show()

```

```
C:\Users\bhuva\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```



In [16]:

```
# Box plot:

plt.boxplot(acceleration)

# Display the plot:
plt.show()
```



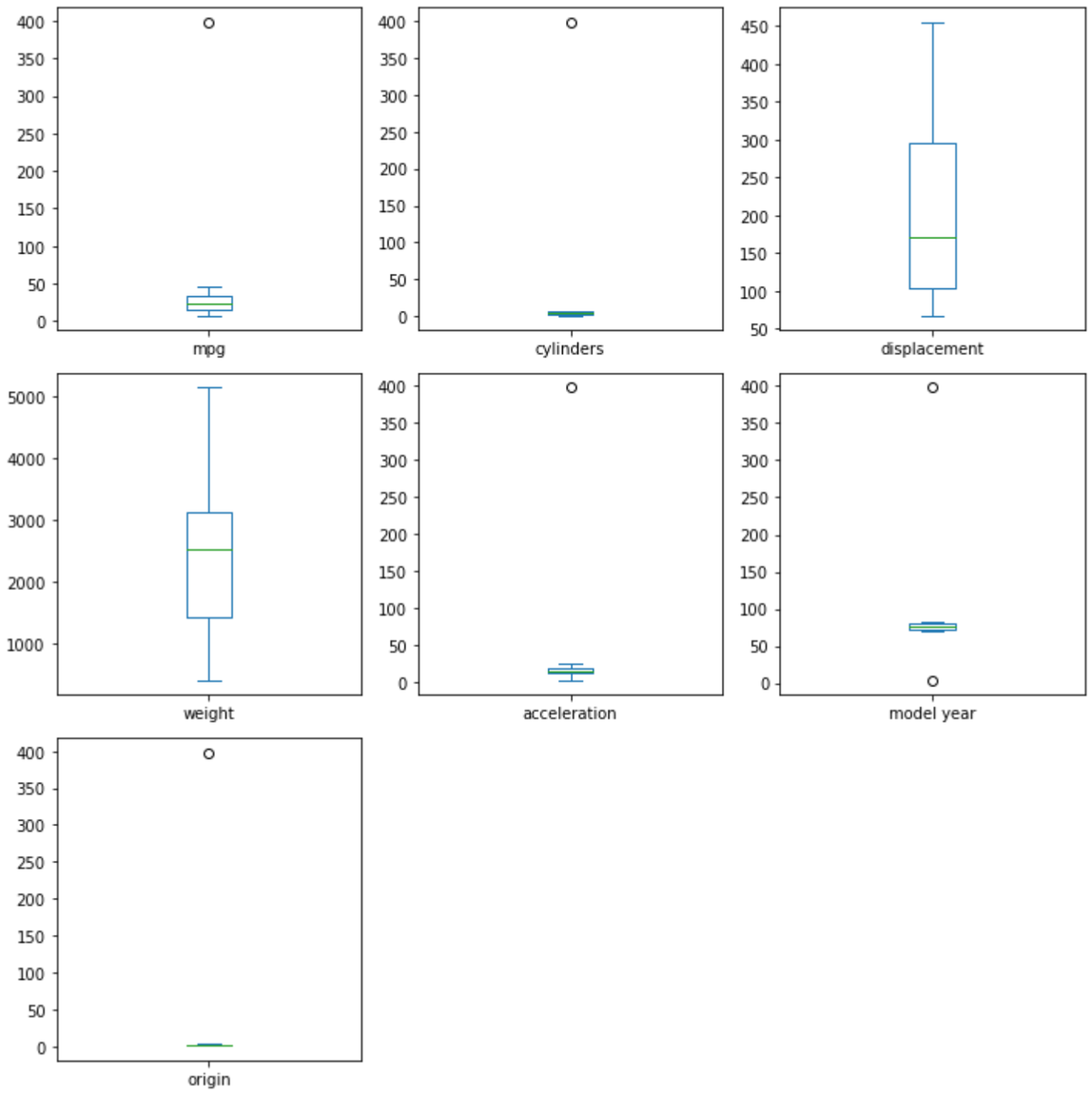
`Interpretation` : The Boxplot of column Acceleration has Outliers in it, thus we can able to remove by using IQR method.

In [20]:

```
# Only Numerical Colms from my Dataset:


df_numerical = df_car.describe(include = [np.number])
```

In [21]:

```
df_numerical.plot(kind = 'box', subplots = True, layout = (3, 3))



# To give the Specified padding from the subplots:
plt.tight_layout()

# Display the Plot:
plt.show()
```



In [ ]:

```

```

In [ ]:
```
1
```

In [ ]:
```
1
```

Distribution of the Plot by Using Distplot:

`Hint` : We can also the plot the Distribution plot by using without Histogram

In [ ]:
```
1
```

In [ ]:
```
1
```

MULTIVARIATE ANALYSIS

- It is the analysis of two or more variables
- Used to study the relationships between the Variables
- Useful in determining the effect of one variable on other variables

1. Scatter plot
2. Correlation Matrix
3. Pair Plot
4. Grouped Box Plot

Handling Non-numeric Datas :

Variable Encoding

- The dataset may contain numerical and / or Categorical Varibales. Most of the algorithmsare designed to work on numeric data.
- There are Several Methods (`label Encoding` , `Dummy Encoding`) to convert the Categorical data into numerical Data.

Types of Encoding

1. N-1 Dummy Encoding
2. One-Hot Encoding
3. Label encoding
5. Oridnal Encoding
6. Frequency Encoding
7. Target Encoding

In [11]:
```
1 df_car.head(5)
```

Out[11]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

In [12]:
```
1 df_car['origin'].value_counts()
```

Out[12]:
```
1    249
3     79
2     70
Name: origin, dtype: int64
```

In [13]:
```
1 df_car['car name'].value_counts()
```

Out[13]:
```
ford pinto             6
toyota corolla         5
amc matador            5
ford maverick          5
chevrolet chevette     4
                      ..
chevrolet monza 2+2    1
ford mustang ii        1
pontiac astro          1
amc pacer              1
chevy s-10             1
Name: car name, Length: 305, dtype: int64
```

In [14]:

```
# Create Dummy Variables for 'Car name' :
# Drop_first = 'True' creates (n - 1) Dummy Variables from Categories
pd.get_dummies(df_car, columns = ['car name'], drop_first = True)
```

Out[14]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name_amc ambassador dpl	car name_amc ambassador sst	...	car name_volvo 145e (sw)	car name_volvo 244dl	car name_volvo 245	car name_volvo 264gl	car name_volvo diesel	car name_vw dasher (diesel)	car name_vw pickup	car name_vw rabbit	na
0	18.0	8	307.0	130	3504	12.0	70	1	0	0	...	0	0	0	0	0	0	0	0	
1	15.0	8	350.0	165	3693	11.5	70	1	0	0	...	0	0	0	0	0	0	0	0	
2	18.0	8	318.0	150	3436	11.0	70	1	0	0	...	0	0	0	0	0	0	0	0	
3	16.0	8	304.0	150	3433	12.0	70	1	0	0	...	0	0	0	0	0	0	0	0	
4	17.0	8	302.0	140	3449	10.5	70	1	0	0	...	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
393	27.0	4	140.0	86	2790	15.6	82	1	0	0	...	0	0	0	0	0	0	0	0	
394	44.0	4	97.0	52	2130	24.6	82	2	0	0	...	0	0	0	0	0	0	1	0	
395	32.0	4	135.0	84	2295	11.6	82	1	0	0	...	0	0	0	0	0	0	0	0	
396	28.0	4	120.0	79	2625	18.6	82	1	0	0	...	0	0	0	0	0	0	0	0	
397	31.0	4	119.0	82	2720	19.4	82	1	0	0	...	0	0	0	0	0	0	0	0	

398 rows × 312 columns

One Hot Encoding

In [15]:

```
# One Hot Encoding:

pd.get_dummies(df_car, columns = ['origin'])
```

Out[15]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	car name	origin_1	origin_2	origin_3
0	18.0	8	307.0	130	3504	12.0	70	chevrolet chevelle malibu	1	0	0
1	15.0	8	350.0	165	3693	11.5	70	buick skylark 320	1	0	0
2	18.0	8	318.0	150	3436	11.0	70	plymouth satellite	1	0	0
3	16.0	8	304.0	150	3433	12.0	70	amc rebel sst	1	0	0
4	17.0	8	302.0	140	3449	10.5	70	ford torino	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	82	ford mustang gl	1	0	0
394	44.0	4	97.0	52	2130	24.6	82	vw pickup	0	1	0
395	32.0	4	135.0	84	2295	11.6	82	dodge rampage	1	0	0
396	28.0	4	120.0	79	2625	18.6	82	ford ranger	1	0	0
397	31.0	4	119.0	82	2720	19.4	82	chevy s-10	1	0	0

398 rows × 11 columns

In [20]:

```
# Sklearn Library:

# Import The OneHotEncoder:

from sklearn.preprocessing import OneHotEncoder

# Creating an Instance of One-Hot-Encoder:
encode = OneHotEncoder()

# fit_transform : Fit to data and Returns a Transformed Version:
# toarray() : Returns the Numpy array
# Columns : Add the Column names

df_encode = pd.DataFrame(encode.fit_transform(df_car[['origin']]).toarray(), columns = ['origin_europe', 'origin_japan', 'origin_usa'])


# Merge with main Data DataFrame ( df_car):
# Axis = 1: it stands for columns
df_encode = pd.concat([df_car, df_encode], axis = 1)


# Print the First 2 Rows of the DataSet:
df_encode
```

Out[20]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name	origin_europe	origin_japan	origin_usa
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu	1.0	0.0	0.0
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320	1.0	0.0	0.0
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite	1.0	0.0	0.0
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst	1.0	0.0	0.0
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino	1.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl	1.0	0.0	0.0
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup	0.0	1.0	0.0
395	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage	1.0	0.0	0.0
396	28.0	4	120.0	79	2625	18.6	82	1	ford ranger	1.0	0.0	0.0
397	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10	1.0	0.0	0.0

398 rows × 12 columns

`Label Encoding`

The Label Encoder Consider as level in a Categorical Variable by `Alphabetical Order`

In [27]:

```
df_car.head(2)
```

Out[27]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320

In [6]:

```
# Example:

# Use Sklearn Library:
from sklearn.preprocessing import LabelEncoder

# Create an Instance:
labelencoder = LabelEncoder()

# fit the encoder:
df_car['encoded_performance_of_my_car'] = labelencoder.fit_transform(df_car.displacement)

# display the Data:
df_car
```

Out[6]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name	encoded_performance_of_my_car
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu	69
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320	72
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite	70
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst	67
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino	66
...	...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl	41
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup	17
395	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage	40
396	28.0	4	120.0	79	2625	18.6	82	1	ford ranger	34
397	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10	33

398 rows × 10 columns

In [ ]:

```
# Interpretation

# Low performance ranges from 17 - 27
# Average Performance ranges from 37 - 43
# Super Performance ranges from 61 - 72
```

In [4]:

```
df_car['encoded_performance_of_my_car'].value_counts()
```

Out[4]:

```
17    21
19    18
72    18
70    17
61    17
      ..
22     1
26     1
37     1
27     1
43     1
Name: encoded_performance_of_my_car, Length: 82, dtype: int64
```

Ordinal Encoding

In [12]:
```
# Import the Ordinal Encoder:
from sklearn.preprocessing import OrdinalEncoder

# Create an Instance:
orderencoding = OrdinalEncoder(categories = [['Average', 'Good', 'Excellent']])

# Fit the Encoder:
df_car['Ordered_performance'] = orderencoding.fit_transform(df_car.mpg.values.reshape(-1, 1))

# Display the Data:
df_car.head(3)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_50112/1775378340.py in <module>
      6
      7 # Fit the Encoder:
----> 8 df_car['Ordered_performance'] = orderencoding.fit_transform(df_car.mpg.values.reshape(-1, 1))
      9
     10 # Display the Data:

~\anaconda3\lib\site-packages\sklearn\base.py in fit_transform(self, X, y, **fit_params)
    865         if y is None:
    866             # fit method of arity 1 (unsupervised transformation)
--> 867             return self.fit(X, **fit_params).transform(X)
    868         else:
    869             # fit method of arity 2 (supervised transformation)

~\anaconda3\lib\site-packages\sklearn\preprocessing\_encoders.py in fit(self, X, y)
   1292
   1293         # `_fit` will only raise an error when `self.handle_unknown="error"`
-> 1294         self._fit(X, handle_unknown=self.handle_unknown, force_all_finite="allow-nan")
   1295
   1296         if self.handle_unknown == "use_encoded_value":

~\anaconda3\lib\site-packages\sklearn\preprocessing\_encoders.py in _fit(self, X, handle_unknown, force_all_finite, return_counts)
    104                     cats = result
    105             else:
--> 106                 cats = np.array(self.categories[i], dtype=Xi.dtype)
    107                 if Xi.dtype.kind not in "OUS":
    108                     sorted_cats = np.sort(cats)

ValueError: could not convert string to float: 'Average'
```

Frequency Encoding

skill Encoded Feature

Python 0.44 python = 0.44 Python 0.44 Python 0.44 Python 0.44 R 0.33 R = 0.33 R 0.33 R 0.33 SQL 0.22 SQl 0.22 SQL = 0.22 SQl 0.22

import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns

In [14]:
```
1 df_car.head()
```

Out[14]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

In [3]:
```
1 df_car.head(1)
```

Out[3]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu

In [5]:
```
1  # Frequency Encoding:
2  # Size of the each Category:
3  encoding = df_car.groupby('displacement').size()
4
5  # Get the Frequency of Each Category:
6  encoding = encoding / len(df_car)
7
8  # encode the Column:
9  df_car['frequency_displacement'] = df_car.displacement.map(encoding)
10
11 # Display the Data:
12 df_car.head()
```

Out[5]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name	frequency_displacement
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu	0.007538
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320	0.045226
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite	0.042714
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst	0.017588
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino	0.027638

Target Encoding

In [3]:
```
df_smoker = pd.DataFrame({
    'Smoker' : ['yes', 'yes', 'no', 'no', 'yes', 'no', 'yes', 'yes', 'no'],
    'Target' : [1, 0, 1, 0, 0,0,  1, 0, 0]
})
df_smoker
```

Out[3]:

	Smoker	Target
0	yes	1
1	yes	0
2	no	1
3	no	0
4	yes	0
5	no	0
6	yes	1
7	yes	0
8	no	0

In [4]:
```
mean = df_smoker.groupby('Smoker')['Target'].mean()
```

In [9]:
```
df_smoker['Smoker_encoded'] = df_smoker.Smoker.map(mean)


df_smoker
```

Out[9]:

	Smoker	Target	Smoker_encoded
0	yes	1	0.40
1	yes	0	0.40
2	no	1	0.25
3	no	0	0.25
4	yes	0	0.40
5	no	0	0.25
6	yes	1	0.40
7	yes	0	0.40
8	no	0	0.25

Feature Scaling:

- Feature Scaling is also Called as Data Normalization.
- It is the technique used to Transform the Data into a common scale. Since the Features have various ranges, it becomes a necessary step in the data preprocessing while using machine learning algorithms.

Methods to Perform Feature Scaling:

1. Z-score Normalization (or) Standardization

- Standardization transforms the Data such that the data has mean 0 and Unit Variance.
- The Procedure involves subracting the mean from Observation and dividing by the Standard Deviation
- (xnew) = x - μ / sigma

In [15]:
```
df_car.head(1)
```

Out[15]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu

In [5]:
```
# Import Standard Scaler:
from sklearn.preprocessing import StandardScaler

# Minimum and maximum Values of 'Weight':
# '\n' - add Space

print('Minimum value Before Transformation : ', df_car.weight.min(), '\n'
     'Maximum value Before Transformation : ', df_car.weight.max(), '\n')

# Insantinate the StandardScaler:
standard_scale = StandardScaler()

# Fit the StandardScaler:
# fit_transform() : returns a Transformed Data
df_car['Scaled_Weight'] = standard_scale.fit_transform(df_car[['weight']])

print('Minimum value after Transformation : ', df_car['Scaled_Weight'].min(), '\n'
     'Maximum value after Transformation : ', df_car['Scaled_Weight'].max(), '\n')
```

```
Minimum value Before Transformation :  1613
Maximum value Before Transformation :  5140

Minimum value after Transformation :  -1.6049434405635041
Maximum value after Transformation :  2.565185359572092
```

Scaled value mean = 0

----------->> IMPORTANT

Scaled value Standard Deviation = 1

In [6]:
```
print('Mean : ', df_car['Scaled_Weight'].mean())
print('\n')
print('Standard Deviation : ', df_car['Scaled_Weight'].std())
```

```
Mean :  -9.902743058842477e-17


Standard Deviation :  1.0012586537392114
```

In [4]:
```
df_car
```

Out[4]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name	Scaled_Weight
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu	0.630870
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320	0.854333
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite	0.550470
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst	0.546923
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino	0.565841
...	...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl	-0.213324
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup	-0.993671
395	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage	-0.798585
396	28.0	4	120.0	79	2625	18.6	82	1	ford ranger	-0.408411
397	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10	-0.296088

398 rows × 10 columns

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

2. Min-Max Normalization

- Performs linear transformation on the original data
- The min-max normalization is given as
- [(xnorm) = X - Xmin / Xmax - Xmin]

In [11]:
```
# IMPORT LIBRARIES:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [12]:
```
1 df_car.head(2)
```

Out[12]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name	min_max_scaled_weight
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu	0.536150
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320	0.589736

In [10]:
```
1  # Import MInMaxScaler:
2  from sklearn.preprocessing import MinMaxScaler
3
4  # Creating an Instance:
5  min_max = MinMaxScaler()
6
7  # Fit the MinMaxScaler:
8  df_car['min_max_scaled_weight'] = min_max.fit_transform(df_car[['weight']])
9
10 # Minimum and Maximum  of Normalized Weight:
11 df_car['min_max_scaled_weight'].min(), df_car['min_max_scaled_weight'].max()
```

Out[10]: (0.0, 1.0)

Interpretation :

The New Data Has been Generated for the Particular Column weight b/w 0 - 1

Data Transformation:

1. Log Transformation

- Reduce the **skewness in the Distribution of the Original Data**
- Makes the Data more interpretable
- The Arithmetic mean of the log-transformed data is the geometric mean of the original Data
- It Converts the Exponential Growth of the linear Growth

In [13]:
```
1 df_car.head(1)
```

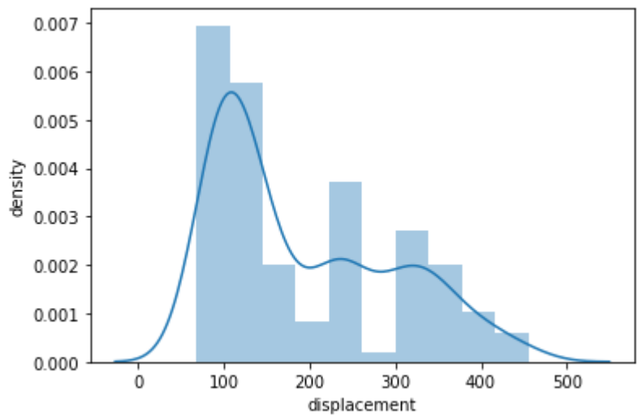
Out[13]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name	min_max_scaled_weight
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu	0.53615

In [14]:

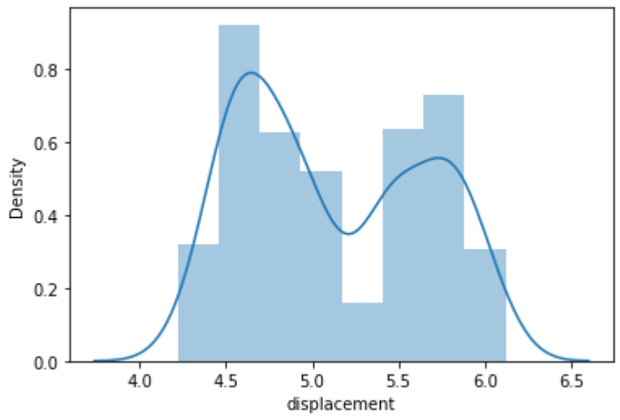
```
# Set the Figure Sizxe:
plt.rcParams['figure.figsize'] = [6, 4]

# Distribution of the Displacement:
sns.distplot(df_car['displacement'])

plt.ylabel('density')

# Coefficient of Skewness:
print('Skewness : ', df_car['displacement'].skew())

# Display the PLot:
plt.show()
```

```
C:\Users\bhuva\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)

Skewness :  0.7196451643005952
```



In [15]:
```
# Apply Natural Log Transformation with (base 'e')
log_displacement = np.log(df_car['displacement'])


# Coefficient of Skewness:
print('Skewness : ', log_displacement.skew())

# Distribution of Log_transformed Variable:
sns.distplot(log_displacement)

# Set label for y - Axis :
plt.ylabel('Density')

# Display the Plot :
plt.show()
```

```
Skewness :  0.22600298495225188

C:\Users\bhuva\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [26]:
```
df_car.shape

```

Out[26]: (398, 10)

In [ ]:
```

```

In [ ]:
```

```

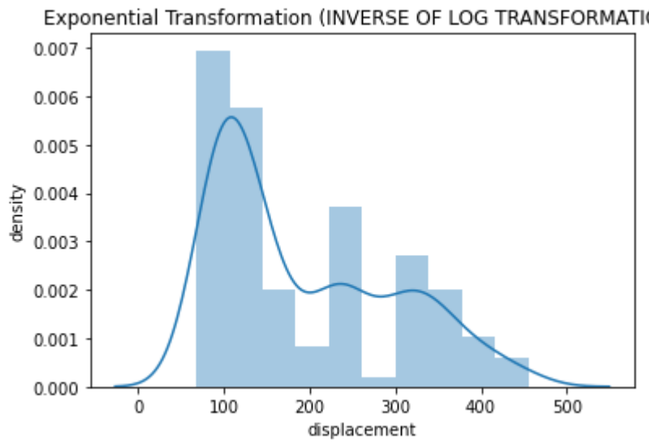
2. Exponential Transformation

- It is the **Inverse transformation**` of the Log Transformation
- It is used to Convert the log Transformed values to their original Units

In [16]:

```
# Anti - LOg of 'Displacement' :
displacement = np.exp(log_displacement)

# Plot The Distribution :
sns.distplot(displacement)

# Set the Label y - axis:
plt.ylabel('density')

# Set the Title:
plt.title('Exponential Transformation (INVERSE OF LOG TRANSFORMATION)')

# Display the Plot:
plt.show()
```

```
C:\Users\bhuva\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please
adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



3. Box-Cox Transformation

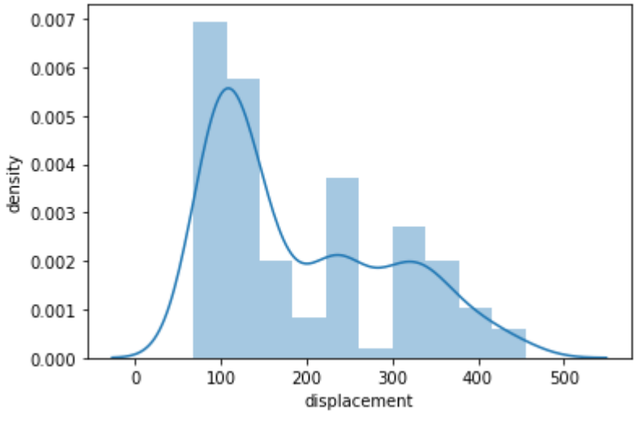
- Generalized version of Log transformation
- Reduce the SKewness of the data making it more Symmetrical
- The Box-Cox transformation for Variable x with Positive Values is defined as:

  x(λ) = x ^ λ - 1 / λ Whereas, λ != 0

In [37]:

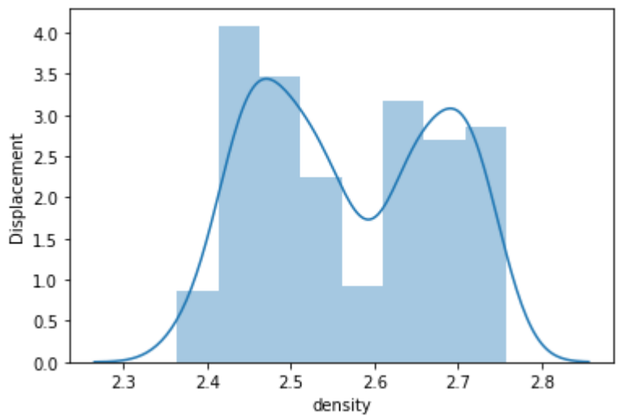
```
# Set the Figure Size:
plt.rcParams['figure.figsize'] = [6, 4]

# Distribution of the Displacement:
sns.distplot(df_car['displacement'])

# Set the Label for y - axis:
plt.ylabel('density')

# Coefficient of the Skewness:
print('Skewnenss : ', df_car['displacement'])

# Display the Plot:
plt.show()
```

```
Skewnenss :  0      307.0
1      350.0
2      318.0
3      304.0
4      302.0
       ...
393    140.0
394     97.0
395    135.0
396    120.0
397    119.0
Name: displacement, Length: 398, dtype: float64
```



In [41]:

```
# Box Cox Transformation:

# Import scipy Library:

import scipy

from scipy import stats

# Scipy.stats.boxcox() returns tuple with transformation:
box_displacement = scipy.stats.boxcox(df_car['displacement'])

# Distribution of Transformed Variable:
# [0] refers to array of the indexing particular Column in Displacement:
sns.distplot(box_displacement[0])

# Set axes label:
plt.ylabel('Displacement')
plt.xlabel('density')

# Display the plot:
plt.show()


```



In [49]:

```
df_car_price = pd.read_csv('CarPrice_Assignment.csv')

# Shape of the Data:
print(df_car_price.shape)


# Print First Five Rows:
df_car_price.head()
```

```
(205, 26)
```

Out[49]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsystem	boreratio	stroke	compressionratio	horsepower	peakrpm	citympg
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000	19
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500	24
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500	18

5 rows × 26 columns

Read the Dataset Data Information Data Type Conversion

Missing Value Treatment

- Replace by Frequency
- Replac by Mean
- Replace by Median
- Repalce by bfill() and ffill()

Outlier Treatment

- z - Score
- Interquartile Range Methodolgy

Feature Engineering

Train Test Split

Supervised Learning:

Linear Regression Algorithms

Logistic Regression Algorithms

Unsupervised Learning:

Clustering

Hyper parameter tuning

Decision Tree

Random Forest

MISSING VALUES :

- Standard Missing Values
- Non Standard Missing Values

In [62]:

```
# Import libraries:


# Importing pandas library and make it as pd
import pandas as pd



import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [63]:

```
df = pd.read_csv('k-circlesales-cleaned.csv')
df.head()
```

Out[63]:

	Unnamed: 0	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales	Profit
0	0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8	OUT049	1999	Medium	Tier 2	Supermarket Type1	3735.1380	11.5
1	1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.3	OUT018	2009	Medium	Tier 2	Supermarket Type2	443.4228	14.3
2	2	FDN15	17.50	Low Fat	0.016760	Meat	141.6	OUT049	1999	Medium	Tier 2	Supermarket Type1	2097.2700	14.5
3	3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.1	OUT010	1998	NaN	NaN	Grocery Store	732.3800	13.6
4	4	NCD19	8.93	Low Fat	0.000000	Household	53.9	OUT013	1987	High	Tier 3	Supermarket Type1	994.7052	14.1

1. Data Summary:

In [64]:
```
# Find the Shape of the Dataset:
df.shape
```

Out[64]: (8523, 14)

In [65]:
```
# Summary of all the Variable in the Dataset:
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 14 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Unnamed: 0                 8523 non-null   int64
 1   Item_Identifier            8523 non-null   object
 2   Item_Weight                7774 non-null   float64
 3   Item_Fat_Content           8523 non-null   object
 4   Item_Visibility            8523 non-null   float64
 5   Item_Type                  8523 non-null   object
 6   Item_MRP                   8523 non-null   float64
 7   Outlet_Identifier          8523 non-null   object
 8   Outlet_Establishment_Year  8523 non-null   int64
 9   Outlet_Size                6113 non-null   object
 10  Outlet_Location_Type       6473 non-null   object
 11  Outlet_Type                8523 non-null   object
 12  Item_Outlet_Sales          8523 non-null   float64
 13  Profit                     8523 non-null   float64
dtypes: float64(5), int64(2), object(7)
memory usage: 932.3+ KB
```

The Statistical Summary of the Dataset for Numerical Variables

In [66]:
```
# Summary of Numerical Variables:

df.describe()

# By default describe(), gives you five point summaryb only for Numerical Columns.
```

Out[66]:

	Unnamed: 0	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales	Profit
count	8523.000000	7774.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000
mean	4261.000000	11.676740	0.066132	140.998838	1997.831867	2181.288914	13.414514
std	2460.522505	5.776851	0.051598	62.258099	8.371760	1706.499616	1.701840
min	0.000000	0.000000	0.000000	31.300000	1985.000000	33.290000	0.100000
25%	2130.500000	7.720000	0.026989	93.800000	1987.000000	834.247400	13.150000
50%	4261.000000	11.800000	0.053931	142.700000	1999.000000	1794.331000	13.900000
75%	6391.500000	16.500000	0.094585	185.650000	2004.000000	3101.296400	14.300000
max	8522.000000	21.350000	0.328391	266.900000	2009.000000	13086.964800	24.000000

The Statistical Summary of the Dataset for Categorical Variables

In [67]:
```
# Get Summary for Cat columns:
df.describe(include = 'object')
```

Out[67]:

	Item_Identifier	Item_Fat_Content	Item_Type	Outlet_Identifier	Outlet_Size	Outlet_Location_Type	Outlet_Type
count	8523	8523	8523	8523	6113	6473	8523
unique	1559	2	16	10	3	8	4
top	FDW13	Low Fat	Fruits and Vegetables	OUT027	Medium	Tier 2	Supermarket Type1
freq	10	5517	1232	935	2793	2793	5577

In [68]:
```
# Check for Null Values:

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

Out[68]:
```
Unnamed: 0                    0
Item_Identifier               0
Item_Weight                 749
Item_Fat_Content              0
Item_Visibility               0
Item_Type                     0
Item_MRP                      0
Outlet_Identifier             0
Outlet_Establishment_Year     0
Outlet_Size                2410
Outlet_Location_Type       2050
Outlet_Type                   0
Item_Outlet_Sales             0
Profit                        0
dtype: int64
```

In [69]:
```
# Total Null Values in the Dataset
# Sum of all the Null values:

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

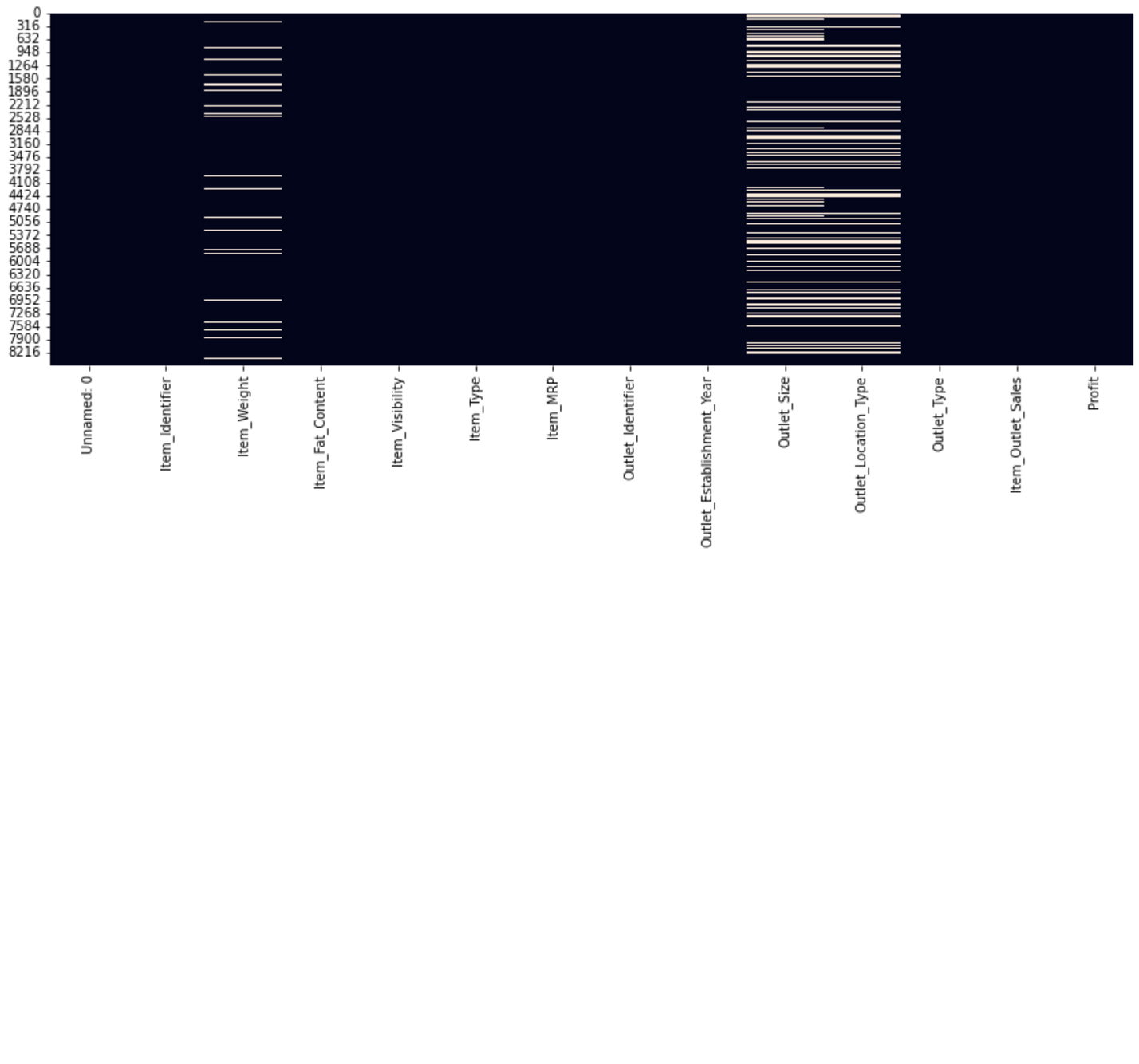
Out[69]:
```
5209
```

In [70]:
```
# Sanity Check whether our dataset has null values:
df.isnull().values.any()
```

Out[70]:
```
True
```

In [71]:

```
# Let us plot the Heatmap to visualize the missing values:



# Set the Figure Size:
plt.rcParams['figure.figsize'] = [15, 5]

# Plot the heatmap:
sns.heatmap(df.isnull(), cbar = False)

# PLot the map:
plt.show()
```



In [72]:
```
# get the count of Missing Values:
missing_values = df.isnull().sum()

# check for missing Values:
total = df.isnull().sum().sort_values(ascending = False)

# Calculate percentage of Null Values:
percent = ((df.isnull().sum() / df.shape[0])*100)

# Sort the Values in descending order
percent = percent.sort_values(ascending = False)

# Concatenate the Total Missing Values and Percentage of Missing Values:
missing_data = pd.concat([total, percent], axis = 1,
                         keys = ['Total', 'percentage'])
# Add the Data types:
missing_data['Type'] = df[missing_data.index].dtypes

# View the Missing Data:
missing_data
```

Out[72]:

	Total	percentage	Type
Outlet_Size	2410	28.276428	object
Outlet_Location_Type	2050	24.052564	object
Item_Weight	749	8.787985	float64
Unnamed: 0	0	0.000000	int64
Item_Identifier	0	0.000000	object
Item_Fat_Content	0	0.000000	object
Item_Visibility	0	0.000000	float64
Item_Type	0	0.000000	object
Item_MRP	0	0.000000	float64
Outlet_Identifier	0	0.000000	object
Outlet_Establishment_Year	0	0.000000	int64
Outlet_Type	0	0.000000	object
Item_Outlet_Sales	0	0.000000	float64
Profit	0	0.000000	float64

In [73]:
```
df.shape
```

Out[73]: (8523, 14)

In [74]:
```
# drop the rows having missing values:
df_sales_drop = df.dropna()

# Display the DataFrame
df_sales_drop.shape
```

Out[74]: (5364, 14)

In [75]:
```
df.shape
```

Out[75]: (8523, 14)

In [76]:
```
# Make a copy of the Dropped DataFrame:
df_Sales2 = df_sales_drop.copy()
```

Let us Consider each Variable Separately for missing value Treatment

1. Outlet_size

In [77]:
```
# check Number of Null Values:
df.Outlet_Size.isnull().sum()
```

Out[77]: 2410

In [78]:
```
df.Outlet_Size.value_counts()
```

Out[78]:
```
Medium    2793
Small     2388
High       932
Name: Outlet_Size, dtype: int64
```

In [79]:
```
# Let's Check the Head of the Column:
df.Outlet_Size
```

Out[79]:
```
0       Medium
1       Medium
2       Medium
3          NaN
4         High
          ...
8518      High
8519       NaN
8520     Small
8521    Medium
8522     Small
Name: Outlet_Size, Length: 8523, dtype: object
```

In [80]:
```
# Obtain the Mode Value:
df.Outlet_Size.mode()
```

Out[80]:
```
0    Medium
dtype: object
```

In [81]:
```
# Replace all the Missing Values with 'Medium':
df.Outlet_Size.replace(np.NaN, 'Medium', inplace = True)
```

In [82]:
```
# Sanity Check for the Missing Values:
df.Outlet_Size.isnull().sum()
```

Out[82]: 0

In [83]:
```
# check for Value Counts:
df.Outlet_Size.value_counts()
```

Out[83]:
```
Medium    5203
Small     2388
High       932
Name: Outlet_Size, dtype: int64
```

2. The Variable Outlet_Location_Type:

In [84]:
```
df.Outlet_Location_Type.isnull().sum()
```

Out[84]: 2050

In [85]:
```
# Check for the Count of Sub-categories:
df.Outlet_Location_Type.value_counts()
```

Out[85]:
```
Tier 2    2793
Tier1     2388
Tier 3     932
?          120
--         109
-           67
na          48
NAN         16
Name: Outlet_Location_Type, dtype: int64
```

In [86]:
```
# replace '?' with NaN
# to_replace : value that will be replaced
# Value : Value to replace values matching 'to replace' with
df.Outlet_Location_Type.replace(to_replace = '?', value = np.NaN, inplace = True)


# replace ' -' with NaN
# to_replace : value that will be replaced
# Value : Value to replace values matching 'to replace' with
df.Outlet_Location_Type.replace(to_replace = 'na', value = np.NaN, inplace = True)


# replace ' -' with NaN
# to_replace : value that will be replaced
# Value : Value to replace values matching 'to replace' with
df.Outlet_Location_Type.replace(to_replace = 'NAN', value = np.NaN, inplace = True)
```

In [87]:
```
# replace '--' with NaN
# to_replace : value that will be replaced
# Value : Value to replace values matching 'to replace' with
df.Outlet_Location_Type.replace(to_replace = '  --', value = np.NaN, inplace = True)
```

In [88]:
```
# replace ' -' with NaN
# to_replace : value that will be replaced
# Value : Value to replace values matching 'to replace' with
df.Outlet_Location_Type.replace(to_replace ='  -', value = np.NaN, inplace = True )
```

In [89]:
```
# Sanity Check:
df.Outlet_Location_Type.value_counts()
```

Out[89]:
```
Tier 2    2793
Tier1     2388
Tier 3     932
Name: Outlet_Location_Type, dtype: int64
```

In [90]:
```
# Sanity Check for Missing VAlues:
df.Outlet_Location_Type.isnull().sum()
```

Out[90]:
```
2410
```

In [91]:
```
df.head(1)
```

Out[91]:

	Unnamed: 0	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales	Profit
0	0	FDA15	9.3	Low Fat	0.016047	Dairy	249.8	OUT049	1999	Medium	Tier 2	Supermarket Type1	3735.138	11.5

In [92]:
```
# Considering the Variable Item_Outlet_sales and Outlet_Location_Type
# Dropna() : drops rows having missing values:

df_impute = df[['Item_Outlet_Sales', 'Outlet_Location_Type']].dropna()


# Group the Data by the Loaction Type:
# Mean() : return the mean of the Values:
df_impute = df_impute.groupby(df_impute['Outlet_Location_Type']).mean()

# Display the DataFrame:
df_impute
```

Out[92]:

	Item_Outlet_Sales
Outlet_Location_Type	
Tier 2	2681.603542
Tier 3	2298.995256
Tier1	1912.149161

In [93]:
```
# Create a variable 'ind' for all the indexes:
index = np.where(df.Outlet_Location_Type.isnull())[0]


# Impute the Values using For Loop:
for ind in index:
    if df.loc[ind, 'Item_Outlet_Sales'] <= 2100:
        df.loc[ind, 'Outlet_Location_Type'] = 'Tier1'
    elif df.loc[ind, 'Item_Outlet_Sales'] >= 2500:
        df.loc[ind, 'Outlet_Location_Type'] = 'Tier2'
    else:
        df.loc[ind, 'Outlet_Location_Type'] = 'Tier3'
```

In [94]:
```
df.Outlet_Location_Type.isnull().sum()
```

Out[94]:
```
0
```

In [95]:
```
df.Item_Weight.describe()
```

Out[95]:
```
count    7774.000000
mean       11.676740
std         5.776851
min         0.000000
25%         7.720000
50%        11.800000
75%        16.500000
max        21.350000
Name: Item_Weight, dtype: float64
```

In [96]:
```
# Replace the Value with NaN
df.Item_Weight.replace(0, np.NaN, inplace = True)
```

In [97]:
```
df.Item_Weight.describe()
```

Out[97]:
```
count    7060.000000
mean       12.857645
std         4.643456
min         4.555000
25%         8.773750
50%        12.600000
75%        16.850000
max        21.350000
Name: Item_Weight, dtype: float64
```

Choice of Making Median or Mean for Numerical Variable

In [98]:

```
# Import Necessary Libraries
from matplotlib import gridspec

# Set the Plot Size:
plt.rcParams['figure.figsize'] = [15, 5]

# Specify the Geometry of the Grid that a Subplot is Placed in:
# Split the PLot into 2 Rows and 3 Columns:
gs = gridspec.GridSpec(2, 3, width_ratios = [.5, .5, .5], height_ratios = [2, 15])


# Step = 1: Specify the Plot Location by calling 'gs' initiative above:
# Step = 2: Write the Plot Text:
# Write the Text in the Plot using text()
# x : Location on x-axis where the Text is to be Written
# y : Location on y-axis where the Text is to be Written
# s : Text to be Written
# Fontsize: set the Font Size
# Step = 3: use.axis('off') to Hide the X and Y axes:

# Plot the 1st Row and 1st column:
a11 = plt.subplot(gs[0, 0])
a11.text(x = 0.1, y = 0.03, s = 'Original Data', fontsize = 15)
a11.axis('off')

# Plot the 1st Row and 2nd Column:
a12 = plt.subplot(gs[0, 1])
a12.text(x = 0.1, y = 0.03, s = 'Data Imputed with Mean', fontsize = 15)
a12.axis('off')

# Plot the 1st Row and 3rd Column:
a12 = plt.subplot(gs[0, 2])
a12.text(x = 0.1, y = 0.03, s = 'Data Imputed with Median', fontsize = 15)
a12.axis('off')

# SUMMARY STATISTICS:
# Step = 1: Specify the Location of the Plot Calling 'gs' Initiative Above in:
# Step = 2: Specify the Text along with Location:
# Step = 3: use.axis('off') to Hide the x and y axes:

# Plot in 2nd Row and 1st Column:
# Original Data:
a12 = plt.subplot(gs[1, 0])
a12.text(0.05, .3, s = str(df['Item_Weight'].describe()), fontsize = 15)
a12.axis('off')

# PLot in 2nd Row and 2nd Column:
# fill the missing Values with Mean
# Obtain the Mean of the Data
mu = df['Item_Weight'].mean()
a12 = plt.subplot(gs[1, 1])
a12.text(0.05, 0.3, s = str(df['Item_Weight'].fillna(mu).describe()), fontsize = 15)
a12.axis('off')


# Fill the Missing Value with Median:
me = df['Item_Weight'].median()
a12 = plt.subplot(gs[1, 2])
a12.text(0.05, 0.3, s = str(df['Item_Weight'].fillna(me).describe()), fontsize = 15)
a12.axis('off')

```

```
# Display the Plot:
plt.show()
```

```
Original Data
count    7060.000000
mean       12.857645
std         4.643456
min         4.555000
25%         8.773750
50%        12.600000
75%        16.850000
max        21.350000
Name: Item_Weight, dtype: float64

Data Imputed with Mean
count    8523.000000
mean       12.857645
std         4.226124
min         4.555000
25%         9.310000
50%        12.857645
75%        16.000000
max        21.350000
Name: Item_Weight, dtype: float64

Data Imputed with Median
count    8523.00000
mean       12.81342
std         4.22724
min         4.55500
25%         9.31000
50%        12.60000
75%        16.00000
max        21.35000
Name: Item_Weight, dtype: float64
```

In [99]:
```
df['Item_Weight'].fillna(me, inplace = True)
```

In [100]:
```
df.Item_Weight.describe()
```

Out[100]:
```
count    8523.00000
mean       12.81342
std         4.22724
min         4.55500
25%         9.31000
50%        12.60000
75%        16.00000
max        21.35000
Name: Item_Weight, dtype: float64
```

In [101]:

```
# Sanity check:


# get the count of Missing Values:
missing_values = df.isnull().sum()

# check for missing Values:
total = df.isnull().sum().sort_values(ascending = False)

# Calculate percentage of Null Values:
percent = ((df.isnull().sum() / df.shape[0])*100)

# Sort the Values in descending order
percent = percent.sort_values(ascending = False)

# Concatenate the Total Missing Values and Percentage of Missing Values:
missing_data = pd.concat([total, percent], axis = 1,
                         keys = ['Total', 'percentage'])
# Add the Data types:
missing_data['Type'] = df[missing_data.index].dtypes

# View the Missing Data:
missing_data
```

Out[101]:

	Total	percentage	Type
Unnamed: 0	0	0.0	int64
Item_Identifier	0	0.0	object
Item_Weight	0	0.0	float64
Item_Fat_Content	0	0.0	object
Item_Visibility	0	0.0	float64
Item_Type	0	0.0	object
Item_MRP	0	0.0	float64
Outlet_Identifier	0	0.0	object
Outlet_Establishment_Year	0	0.0	int64
Outlet_Size	0	0.0	object
Outlet_Location_Type	0	0.0	object
Outlet_Type	0	0.0	object
Item_Outlet_Sales	0	0.0	float64
Profit	0	0.0	float64

OUTLIER DETECTION

- 1. Based on Boxplots

In [102]:
```
# Filtering the Numerical Column from the DataSet:
df_num = df.select_dtypes(include = [np.number])
```

In [103]:
```
df_num.head(2)
```

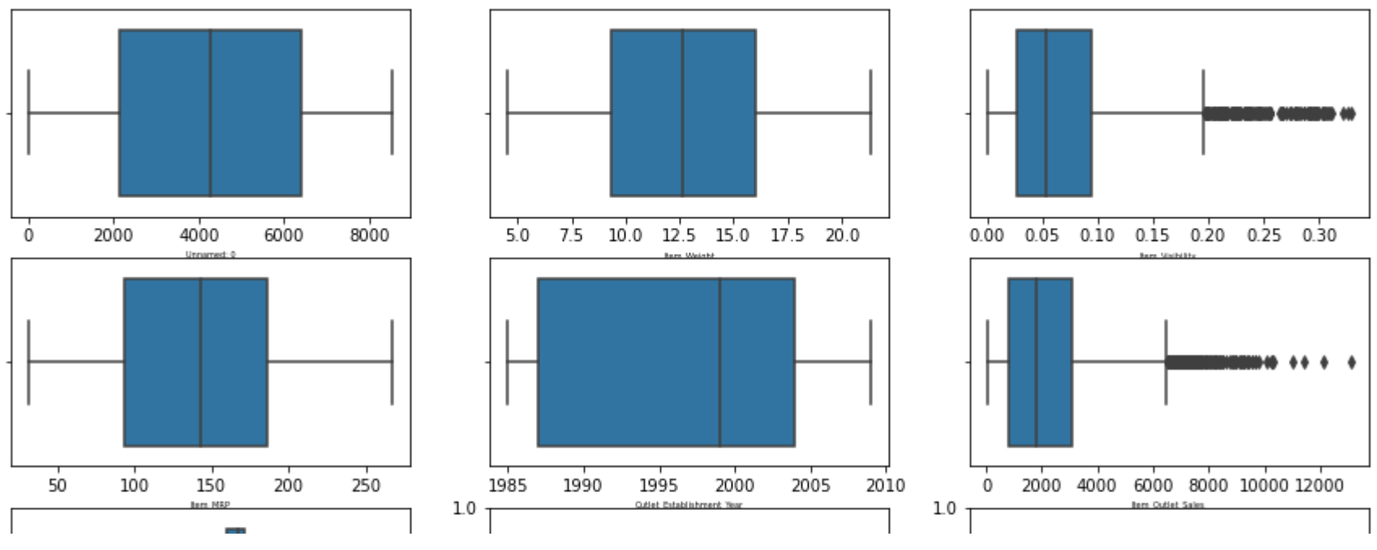
Out[103]:

	Unnamed: 0	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales	Profit
0	0	9.30	0.016047	249.8	1999	3735.1380	11.5
1	1	5.92	0.019278	48.3	2009	443.4228	14.3

In [104]:
```
df_num.columns
```

Out[104]:
```
Index(['Unnamed: 0', 'Item_Weight', 'Item_Visibility', 'Item_MRP',
       'Outlet_Establishment_Year', 'Item_Outlet_Sales', 'Profit'],
      dtype='object')
```

In [105]:
```
# Plot the Boxplot for each Columns:
# Subplots() : plot Subplots:
# Figsize() : set the Figure Size:
fig, ax = plt.subplots(3, 3, figsize = (15, 8))


# Plot the Boxplot using boxplot() from Seaborn Library:
# Z : let the Variable z defines the Boxplot:
# X : data from which the Boxplot is to be Predicted:
# orient : 'h' specifies the Horizontal Boxplot
# Whis : Proportion of the IQR, It past the low and High Quartiles to extend the plot
# ax : specifies the axes object to draw the plot
# Set_xlabel() : set the x - axis Label
# FontSize () : set the Font size of the x - axis
for variable, subplot in zip(df_num.columns, ax.flatten()):
    z = sns.boxplot(x = df_num[variable], orient = 'h', whis = 1.5, ax = subplot)
    z.set_xlabel(variable, fontsize = 5)
```



- 1. Based on IQR Method

In [106]:
```
# Obtain the First Quartiles:
Q1 = df_num.quantile(0.25)

# Obtain the  Quartiles:
Q3 = df_num.quantile(0.75)

# Obtain the IQR:
IQR = Q3 - Q1

# Print the IQR:
IQR
```

Out[106]:
```
Unnamed: 0                 4261.000000
Item_Weight                   6.690000
Item_Visibility               0.067596
Item_MRP                     91.850000
Outlet_Establishment_Year    17.000000
Item_Outlet_Sales          2267.049000
Profit                        1.150000
dtype: float64
```

In [107]:
```
# Filter out the OUtlier Values:
# ~ : Select all the rows which do not satisfy the Condition
# any() : returns whether the element is True over the Column
# axis = 1: Indicates should Select the alternate columns('0' for index Positions)

from warnings import filterwarnings
filterwarnings('ignore')


df_sales_iqr = df[~((df < (Q1 - 1.5 * IQR))| (df > (Q3 + 1.5 * IQR))).any(axis = 1)]
```

In [108]:
```
df_sales_iqr.shape
```

Out[108]: (7515, 14)

In [109]:
```
df.shape
```

Out[109]: (8523, 14)

Based on Z - Score

z - score of a value is the Difference Between that value and the mean, divided by the Standard Deviation. If the `Z - score greater than +3 or less than -3` it indicates an Outlier Value in our Dataset.

In [110]:
```
df.head(1)
```

Out[110]:

	Unnamed: 0	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales	Profit
0	0	FDA15	9.3	Low Fat	0.016047	Dairy	249.8	OUT049	1999	Medium	Tier 2	Supermarket Type1	3735.138	11.5

In [111]:
```
# Import the Library:
import scipy

# from scipy import Stats module:
from scipy import stats

# z - score are defined for each observations in a Variable
# Compute the z - scores using the method zscore from the Scipy Library
z_scores_profit = scipy.stats.zscore(df_num['Profit'])

# display the z - scores:
z_scores_profit
```

Out[111]:
```
0      -1.125033
1       0.520342
2       0.637869
3       0.108998
4       0.402815
          ...
8518    0.402815
8519    0.461578
8520   -2.300301
8521    0.461578
8522    0.696632
Name: Profit, Length: 8523, dtype: float64
```

In [112]:
```
# Print the Rows  where the z - score is less than -3
row_index_less = np.where(z_scores_profit < - 3)

print(row_index_less)
```

```
(array([  41,   50,  144,  217,  320,  324,  406,  425,  432,  457,  546,
        602,  607,  611,  641,  716,  761,  816,  892,  921,  926,  957,
        960,  986, 1066, 1133, 1178, 1212, 1237, 1304, 1409, 1459, 1533,
       1551, 1561, 1582, 1623, 1645, 1649, 1659, 1669, 1715, 1784, 1788,
       1805, 1816, 1835, 1867, 1893, 1900, 1916, 1956, 1974, 1984, 1989,
       2000, 2019, 2060, 2062, 2130, 2201, 2224, 2246, 2289, 2332, 2338,
       2425, 2449, 2477, 2510, 2515, 2526, 2537, 2543, 2572, 2636, 2792,
       2836, 2919, 2925, 2969, 2987, 2992, 2996, 3025, 3035, 3098, 3105,
       3223, 3254, 3296, 3356, 3415, 3416, 3476, 3513, 3619, 3668, 3675,
       3759, 3785, 3849, 3852, 3904, 3917, 3932, 3936, 3990, 4073, 4091,
       4146, 4197, 4213, 4407, 4487, 4493, 4556, 4571, 4705, 4721, 4724,
       4764, 4800, 5049, 5127, 5231, 5274, 5286, 5313, 5336, 5378, 5516,
       5540, 5657, 5658, 5728, 5730, 5750, 5788, 5883, 5891, 5906, 5971,
       5987, 6093, 6137, 6211, 6247, 6271, 6372, 6390, 6419, 6507, 6534,
       6541, 6558, 6606, 6610, 6696, 6759, 6760, 6776, 6778, 6897, 6931,
       6972, 7042, 7180, 7242, 7267, 7304, 7351, 7384, 7432, 7629, 7684,
       7751, 7779, 7797, 7822, 7851, 7893, 7919, 7948, 8074, 8099, 8101,
       8237, 8273, 8366, 8376, 8395, 8429, 8433, 8456], dtype=int64),)
```

`Interpretation` : The Rows Corresponding to the above Dispalyed index are the Outliers for the Data.

In [113]:
```
# Print the rows where z - score is more than 3:
row_index_more = np.where(z_scores_profit > 3)

row_index_more
```

Out[113]:
```
(array([3026, 4386, 5089, 8369], dtype=int64),)
```

In [114]:
```
# Count of Outliers in the Variable representing Profit:


len(row_index_less[0]) + len(row_index_more[0])

```

Out[114]:
```
199
```

There are 199 Outliers in the Data by using z - score identification Methodology

In [115]:
```
# Filter out the Outlier Values:
# ~ : select all the Rows which do not Satisfy the Condition

df_Zscore_profit = df['Profit'][~((z_scores_profit < -3)|(z_scores_profit > 3))]

```

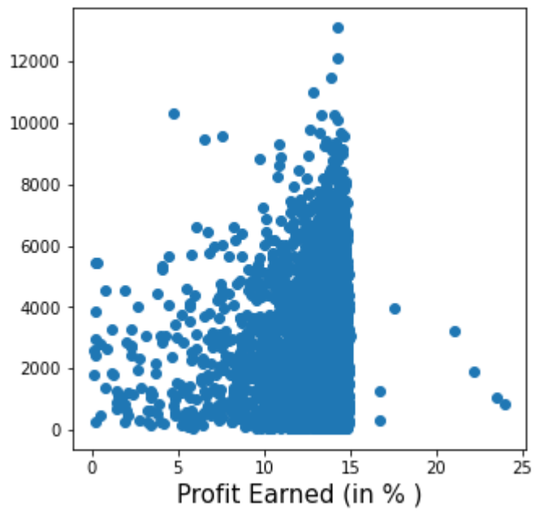
In [116]:
```
# Check for the Shape:
df_Zscore_profit.shape
```

Out[116]:
```
(8324,)
```

In [117]:
```
# Original Dataset Shape:
df.shape
```

Out[117]: (8523, 14)

In [118]:
```
# Initialize the Figure
# FIgsize: set the Plot size:

fig, ax = plt.subplots(figsize = (5, 5))

# Plot the Scatter Plot:
ax.scatter(df['Profit'], df['Item_Outlet_Sales'])


# Set the x -axis label:
# Fontsize : set the font size:
ax.set_xlabel('Profit Earned (in % )', fontsize = 15)

# Set the y - axis
ax.set_ylabel('')
plt.show()
```



In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [124]:
```
df.head(5)
```

Out[124]:

	Unnamed: 0	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales	Profit
0	0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8	OUT049	1999	Medium	Tier 2	Supermarket Type1	3735.1380	11.5
1	1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.3	OUT018	2009	Medium	Tier 2	Supermarket Type2	443.4228	14.3
2	2	FDN15	17.50	Low Fat	0.016760	Meat	141.6	OUT049	1999	Medium	Tier 2	Supermarket Type1	2097.2700	14.5
3	3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.1	OUT010	1998	Medium	Tier1	Grocery Store	732.3800	13.6
4	4	NCD19	8.93	Low Fat	0.000000	Household	53.9	OUT013	1987	High	Tier 3	Supermarket Type1	994.7052	14.1

In [123]:
```
# import the sklearn Library:
import sklearn

# import the train_test_split module from sklearn
from sklearn.model_selection import train_test_split
```

In [125]:
```
# Select the target column:
Y = df['Item_Outlet_Sales']

# select the Independent Column:
# By dropping the Target Column
X = df.drop(['Item_Outlet_Sales'], axis = 1)
```

In [130]:
```
# Let us now split the Dataset into Test and Train
# Test size : The Proportion of the Data to be included in the Testing set.

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.25,
                                                    random_state = 100)
print('X_train : ', x_train.shape)
print('X_test : ', x_test.shape)
print('Y_train : ', y_train.shape)
print('Y_test : ', y_test.shape)
```

```
X_train :  (6392, 13)
X_test :  (2131, 13)
Y_train :  (6392,)
Y_test :  (2131,)
```

In [ ]:
```
1
```

Linear Regression in Machine Learning

- Linear regression is one of the easiest and most popular Machine Learning algorithms.
- It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as sales, salary, age, product price, etc.
- Linear regression algorithm shows a linear relationship between a `dependent (y)` and one or more `independent (y)` variables, hence called as linear regression.
- Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

Types of Linear Regression

Linear regression can be further divided into two types of the algorithm:

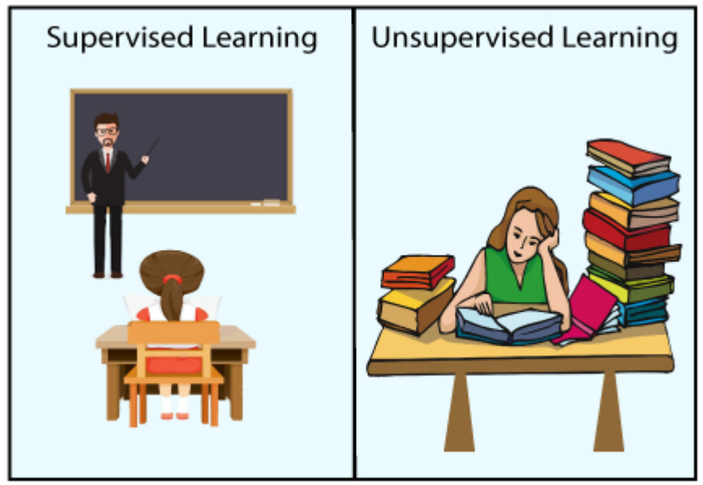
`Simple Linear Regression:`

If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.

`Multiple Linear regression:`

If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

Supervised and Unsupervised Learning



`Supervised Machine Learning:`

Supervised learning is a machine learning method in which models are trained using labeled data. In supervised learning, models need to find the mapping function to map the input variable (X) with the output variable (Y).

Supervised Machine learning Supervised learning needs supervision to train the model, which is similar to as a student learns things in the presence of a teacher. Supervised learning can be used for two types of problems: Classification and Regression.

`Example:` Suppose we have an image of different types of fruits. The task of our supervised learning model is to identify the fruits and classify them accordingly. So to identify the image in supervised learning, we will give the input data as well as output for that, which means we will train the model by the shape, size, color, and taste of each fruit. Once the training is completed, we will test the model by giving the new set of fruit. The model will identify the fruit and predict the output using a suitable algorithm

Supervised learning is the types of machine learning in which machines are trained using well "labelled" training data, and on basis of that data, machines predict the output. The labelled data means some input data is already tagged with the correct output.

In supervised learning, the training data provided to the machines work as the supervisor that teaches the machines to predict the output correctly. It applies the same concept as a student learns in the supervision of the teacher.

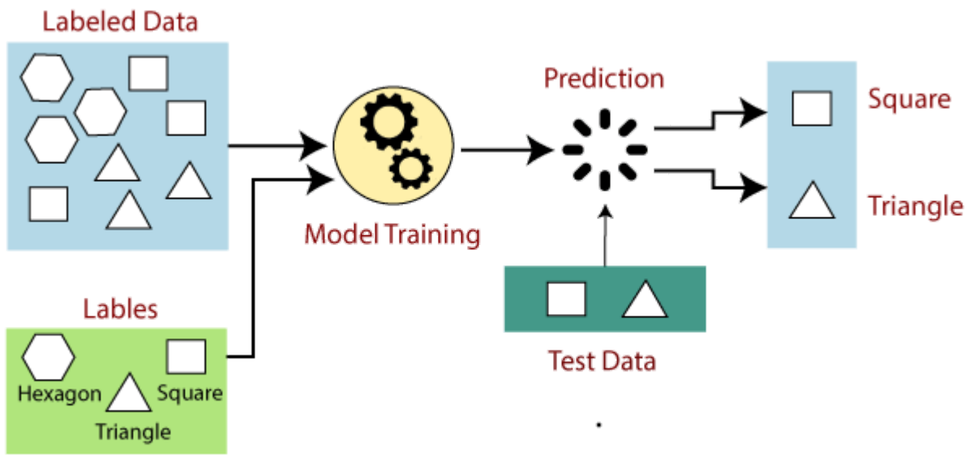
Supervised learning is a process of providing input data as well as correct output data to the machine learning model. The aim of a supervised learning algorithm is to find a mapping function to map the input variable(x) with the output variable(y).

In the real-world, supervised learning can be used for `Risk Assessment, Image classification, Fraud Detection, spam filtering, etc`

How Supervised Learning Works?

In supervised learning, models are trained using labelled dataset, where the model learns about each type of data. Once the training process is completed, the model is tested on the basis of test data (a subset of the training set), and then it predicts the output.

The working of Supervised learning can be easily understood by the below example and diagram



Suppose we have a dataset of different types of shapes which includes square, rectangle, triangle, and Polygon. Now the first step is that we need to train the model for each shape.

If the given shape has four sides, and all the sides are equal, then it will be labelled as a Square.

If the given shape has three sides, then it will be labelled as a triangle.

If the given shape has six equal sides then it will be labelled as hexagon.

Now, after training, we test our model using the test set, and the task of the model is to identify the shape.

The machine is already trained on all types of shapes, and when it finds a new shape, it classifies the shape on the bases of a number of sides, and predicts the output.

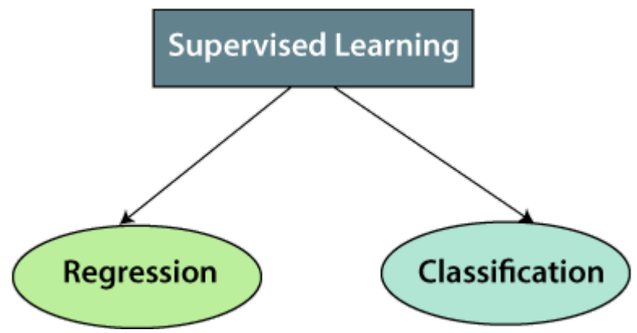
Steps Involved in Supervised Learning:

- First Determine the type of training dataset
- Collect/Gather the labelled training data.
- Split the training dataset into training dataset, test dataset, and validation dataset.
- Determine the input features of the training dataset, which should have enough knowledge so that the model can accurately predict the output.
- Determine the suitable algorithm for the model, such as support vector machine, decision tree, etc.
- Execute the algorithm on the training dataset. Sometimes we need validation sets as the control parameters, which are the subset of training datasets.

- Evaluate the accuracy of the model by providing the test set. If the model predicts the correct output, which means our model is accurate.

Types of supervised Machine learning Algorithms:

Supervised learning can be further divided into two types of problems:



1. Regression

Regression algorithms are used if there is a relationship between the input variable and the output variable. It is used for the prediction of continuous variables, such as Weather forecasting, Market Trends, etc. Below are some popular Regression algorithms which come under supervised learning:

- Linear Regression
- Regression Trees
- Non-Linear Regression
- Bayesian Linear Regression
- Polynomial Regression

2. Classification

Classification algorithms are used when the output variable is categorical, which means there are two classes such as Yes-No, Male-Female, True-false, etc.

Spam Filtering,

- Random Forest
- Decision Trees
- Logistic Regression
- Support vector Machines

Advantages of Supervised learning:

- With the help of supervised learning, the model can predict the output on the basis of prior experiences.

- In supervised learning, we can have an exact idea about the classes of objects.
- Supervised learning model helps us to solve various real-world problems such as fraud detection, spam filtering, etc.

Disadvantages of supervised learning:

- Supervised learning models are not suitable for handling the complex tasks.
- Supervised learning cannot predict the correct output if the test data is different from the training dataset.
- Training required lots of computation times.
- In supervised learning, we need enough knowledge about the classes of object.

Unsupervised Machine Learning

In the previous topic, we learned supervised machine learning in which models are trained using labeled data under the supervision of training data. But there may be many cases in which we do not have labeled data and need to find the hidden patterns from the given dataset. So, to solve such types of cases in machine learning, we need unsupervised learning techniques.

As the name suggests, unsupervised learning is a machine learning technique in which models are not supervised using training dataset. Instead, models itself find the hidden patterns and insights from the given data. It can be compared to learning which takes place in the human brain while learning new things. It can be defined as:

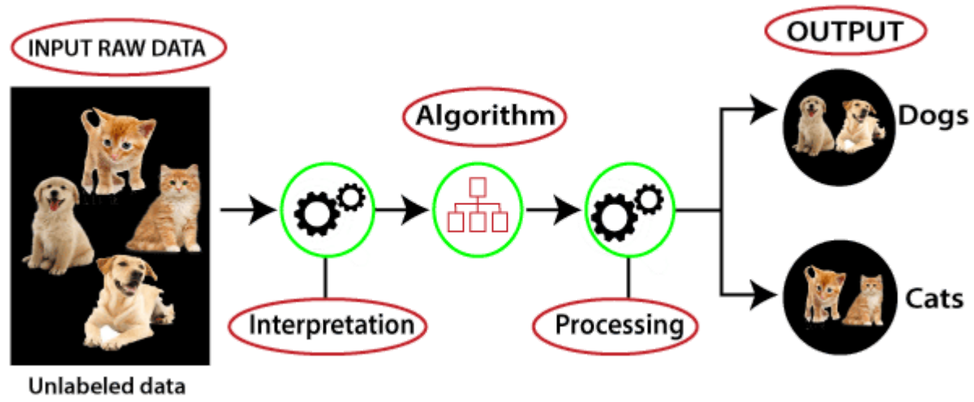
Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data. **The goal of unsupervised learning is to find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format**.

Example: Suppose the unsupervised learning algorithm is given an input dataset containing images of different types of cats and dogs. The algorithm is never trained upon the given dataset, which means it does not have any idea about the features of the dataset. The task of the unsupervised learning algorithm is to identify the image features on their own. Unsupervised learning algorithm will perform this task by clustering the image dataset into the groups according to similarities between images.

Why use Unsupervised Learning?

Below are some main reasons which describe the importance of Unsupervised Learning:

- Unsupervised learning is helpful for finding useful insights from the data.
- Unsupervised learning is much similar as a human learns to think by their own experiences, which makes it closer to the real AI.
- Unsupervised learning works on unlabeled and uncategorized data which make unsupervised learning more important.
- In real-world, we do not always have input data with the corresponding output so to solve such cases, we need unsupervised learning.

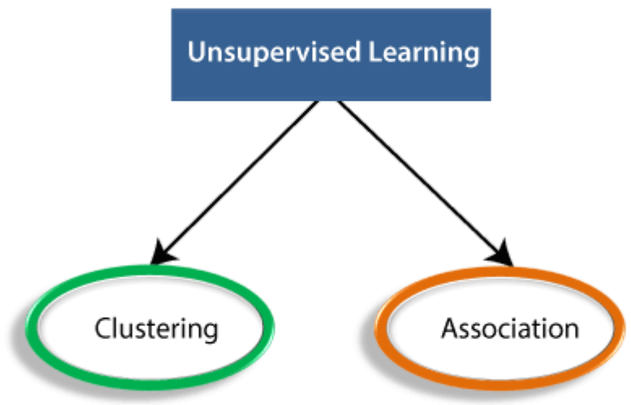


Here, we have taken an unlabeled input data, which means it is not categorized and corresponding outputs are also not given. Now, this unlabeled input data is fed to the machine learning model in order to train it. Firstly, it will interpret the raw data to find the hidden patterns from the data and then will apply suitable algorithms such as k-means clustering, Decision tree, etc.

Once it applies the suitable algorithm, the algorithm divides the data objects into groups according to the similarities and difference between the objects.

`Types of Unsupervised Learning Algorithm:`

The unsupervised learning algorithm can be further categorized into two types of p



Clustering: Clustering is a method of grouping the objects into clusters such that objects with most similarities remains into a group and has less or no similarities with the objects of another group. Cluster analysis finds the commonalities between the data objects and categorizes them as per the presence and absence of those commonalities.

Association: An association rule is an unsupervised learning method which is used for finding the relationships between variables in the large database. It determines the set of items that occurs together in the dataset. Association rule makes marketing strategy more effective. Such as people who buy X item (suppose a bread) are also tend to purchase Y (Butter/Jam) item. A typical example of Association rule is Market Basket Analysis.

`Advantages of Unsupervised Learning`

- Unsupervised learning is used for more complex tasks as compared to supervised learning because, in unsupervised learning, we don't have labeled input data.
- Unsupervised learning is preferable as it is easy to get unlabeled data in comparison to labeled data.

`Disadvantages of Unsupervised Learning`

- Unsupervised learning is intrinsically more difficult than supervised learning as it does not have corresponding output.
- The result of the unsupervised learning algorithm might be less accurate as input data is not labeled, and algorithms do not know the exact output in advance.

In [4]:
```
# Imputation of SLR and OLS Summary:

data = pd.read_excel('Weightexcel.xlsx')
data
```

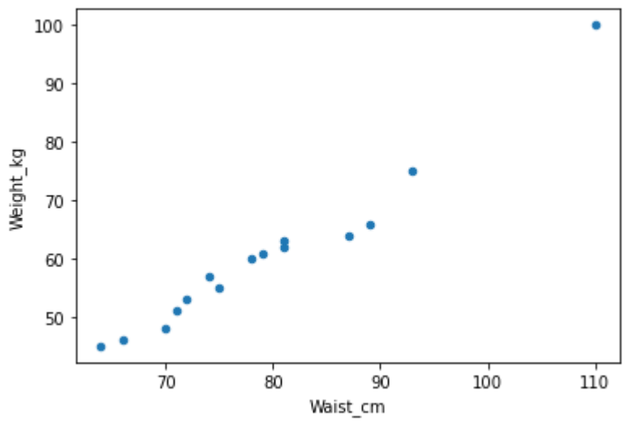
Out[4]:

	Waist_cm	Weight_kg
0	71	51
1	89	66
2	64	45
3	74	57
4	87	64
5	93	75
6	79	61
7	81	62
8	75	55
9	72	53
10	70	48
11	66	46
12	81	63
13	78	60
14	110	100

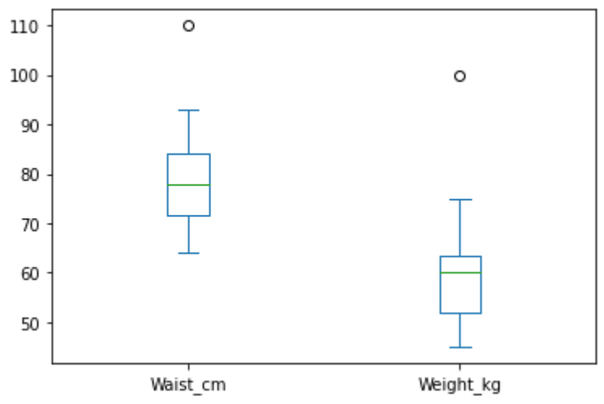
In [5]:
```
data.shape
```

Out[5]: (15, 2)

In [6]:
```
1 data.plot(kind = 'scatter', x = 'Waist_cm', y = 'Weight_kg')
2 plt.show()
```



In [5]:
```
1 data.plot(kind = 'box')
2 plt.show()
```



In [6]:
```
1 data.corr()
```

Out[6]:

	Waist_cm	Weight_kg
Waist_cm	1.000000	0.981216
Weight_kg	0.981216	1.000000

In [13]:
```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
```

In [14]:
```
# Change to the DataFrame variable:
waist = pd.DataFrame(data['Waist_cm'])
weight = pd.DataFrame(data['Weight_kg'])
```

In [15]:
```
waist
```

Out[15]:

	Waist_cm
0	71
1	89
2	64
3	74
4	87
5	93
6	79
7	81
8	75
9	72
10	70
11	66
12	81
13	78
14	110

In [16]:
```
import sklearn

from sklearn.linear_model import LinearRegression
```

In [17]:
```
# Making Instances:
lm = LinearRegression()
model = lm.fit(waist, weight)
```

In [18]:
```
model.coef_
```

Out[18]:
```
array([[1.13470708]])
```

In [19]:
```
model.intercept_
```

Out[19]:
```
array([-29.62009537])
```

In [20]:
```
model.score(waist, weight)
```

Out[20]:
```
0.9627843958606758
```

In [21]:
```
import statsmodels
import statsmodels.api as sm
```

In [22]:
```
# Predict the new value of Weight:
from warnings import filterwarnings
filterwarnings('ignore')


waist_new = np.array([97])
waist_new = waist_new.reshape(-1, 1)
weight_predict = model.predict(waist_new)
weight_predict
```

Out[22]:
```
array([[80.44649183]])
```

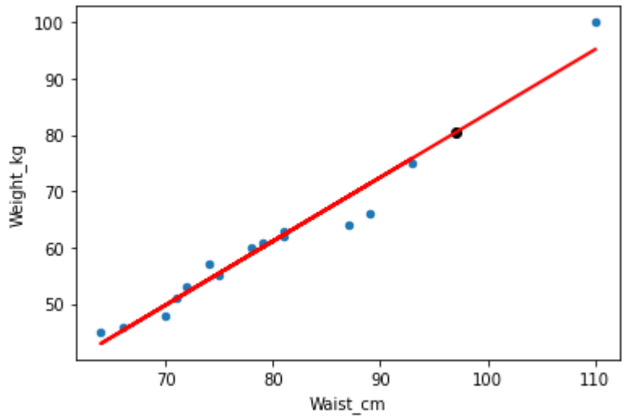
In [21]:
```
# Predict more Values:
X = ([67, 78, 94])
X = pd.DataFrame(X)
Y = model.predict(X)
Y = pd.DataFrame(Y)

df = pd.concat([X, Y], axis = 1, keys = ['Waist_new', 'Weight_predicted'])
df
```

Out[21]:

	Waist_new	Weight_predicted
	0	0
0	67	46.405279
1	78	58.887057
2	94	77.042371

In [27]:

```
# Visual the Result:
data.plot(kind = 'scatter', x = 'Waist_cm', y = 'Weight_kg')


# Plot the REgression line:
plt.plot(waist, model.predict(waist), color = 'r', linewidth = 2)


# Plotting the Predicted Values:
plt.scatter(waist_new, weight_predict, color = 'black')

# Display to Show:
plt.show()
```



In [28]:

```
model = sm.OLS(X, Y)
```

In [29]:

```
fit = model.fit()
```

In [30]:

```
fit.pvalues
```

Out[30]:

```
0    0.002165
dtype: float64
```

In [31]:
```
1 fit.summary()
```

Out[31]:

OLS Regression Results

Dep. Variable:	0	**R-squared (uncentered):**	0.996
Model:	OLS	**Adj. R-squared (uncentered):**	0.994
Method:	Least Squares	**F-statistic:**	460.3
Date:	Sat, 11 Feb 2023	**Prob (F-statistic):**	0.00217
Time:	21:34:05	**Log-Likelihood:**	-9.2545
No. Observations:	3	**AIC:**	20.51
Df Residuals:	2	**BIC:**	19.61
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
0	1.2931	0.060	21.455	0.002	1.034	1.552

Omnibus:	nan	**Durbin-Watson:**	0.981
Prob(Omnibus):	nan	**Jarque-Bera (JB):**	0.306
Skew:	-0.222	**Prob(JB):**	0.858
Kurtosis:	1.500	**Cond. No.**	1.00

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Logistic Regression

In [24]:
```
df = pd.read_csv('Breast_cancer_data.csv')
df
```

Out[24]:

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnosis
0	17.99	10.38	122.80	1001.0	0.11840	0
1	20.57	17.77	132.90	1326.0	0.08474	0
2	19.69	21.25	130.00	1203.0	0.10960	0
3	11.42	20.38	77.58	386.1	0.14250	0
4	20.29	14.34	135.10	1297.0	0.10030	0
...	...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100	0
565	20.13	28.25	131.20	1261.0	0.09780	0
566	16.60	28.08	108.30	858.1	0.08455	0
567	20.60	29.33	140.10	1265.0	0.11780	0
568	7.76	24.54	47.92	181.0	0.05263	1

569 rows × 6 columns

In [25]:
```
df.diagnosis.value_counts()
```

Out[25]:
```
1    357
0    212
Name: diagnosis, dtype: int64
```

In [13]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from warnings import filterwarnings
filterwarnings('ignore')
```

In [14]:
```
from sklearn.datasets import load_breast_cancer

dataset = load_breast_cancer(as_frame = True)
dataset
```

Out[14]:
```
{'data':      mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0          17.99         10.38          122.80     1001.0          0.11840
1          20.57         17.77          132.90     1326.0          0.08474
2          19.69         21.25          130.00     1203.0          0.10960
3          11.42         20.38           77.58      386.1          0.14250
4          20.29         14.34          135.10     1297.0          0.10030
..           ...           ...             ...        ...              ...
564        21.56         22.39          142.00     1479.0          0.11100
565        20.13         28.25          131.20     1261.0          0.09780
566        16.60         28.08          108.30      858.1          0.08455
567        20.60         29.33          140.10     1265.0          0.11780
568         7.76         24.54           47.92      181.0          0.05263

     mean compactness  mean concavity  mean concave points  mean symmetry  \
0             0.27760         0.30010              0.14710         0.2419
1             0.07864         0.08690              0.07017         0.1812
2             0.15990         0.19740              0.12790         0.2069
3             0.28390         0.24140              0.10520         0.2597
4             0.13280         0.19800              0.10430         0.1809
```

In [15]:
```
dataset.data.head()
```

Out[15]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.1189
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.0890
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.0875
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.1730
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.0767

5 rows × 30 columns

In [16]:
```
dataset.target.value_counts()
```

Out[16]:
```
1    357
0    212
Name: target, dtype: int64
```

In [17]:
```
X = dataset['data']
Y = dataset['target']
```

In [18]:
```
# Load Library of Test Train and Split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.25,
                                                    random_state = 100)

print('X_train : ', X_train.shape)
print('X_test : ', X_test.shape)
print('Y_train : ', y_train.shape)
print('Y_test : ', y_test.shape)
```

```
X_train :  (426, 30)
X_test :  (143, 30)
Y_train :  (426,)
Y_test :  (143,)
```

In [19]:
```
from sklearn.preprocessing import StandardScaler

ss_train = StandardScaler()
X_train = ss_train.fit_transform(X_train)

ss_test = StandardScaler()
X_test = ss_test.fit_transform(X_test)
```

In [20]:
```
# Import LOgistic Regression from the Sklearn library:
from sklearn.linear_model import LogisticRegression
```

In [21]:
```
# Create an Instance:
lm = LogisticRegression()
model = lm.fit(X, Y)
```

In [22]:
```
model
```

Out[22]:
```
LogisticRegression()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [23]:
```
predictions = model.predict(X_test)
```

In [24]:

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, predictions)

TN, FP, FN, TP = confusion_matrix(y_test, predictions).ravel()

print('True Positives(TP) = ', TP)
print('False Positives(FP) = ', FP)
print('True Negative(TN) = ', TN)
print('False Negative(FN) = ', FN)
```

```
True Positives(TP) =  53
False Positives(FP) =  33
True Negative(TN) =  23
False Negative(FN) =  34
```

In [28]:

```
accuracy = (TP + TN) / (TP + FP + TN + FN)

print('Accuracy of the Binary Classifier : {:0.3f}'.format(accuracy))
```

```
Accuracy of the Binary Classifier : 0.531
```

In [33]:

```
models = {}

# Logistic Regression:
from  sklearn.linear_model import LogisticRegression
models['Logistic Regression'] = LogisticRegression()

# Support Vector Machines:
from sklearn.svm import LinearSVC
models['Support Vector Machines'] = LinearSVC()


# Decision Tree:
from sklearn.tree import DecisionTreeClassifier
models['Decision Tree'] = DecisionTreeClassifier()

# Random Forest:
from sklearn.ensemble import RandomForestClassifier
models['Random Forest'] = RandomForestClassifier()

# Naive Bayes:
from sklearn.naive_bayes import GaussianNB
models['Naive Bayes'] = GaussianNB()


# k - Nearestv Neighbors:
from sklearn.neighbors import KNeighborsClassifier
models['K- NN Algorithm'] = KNeighborsClassifier()
```

In [37]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score

# creating an instance:
accuracy, precision, recall = {}, {}, {}

for key in models.keys():

    # Fit the Classifier:
    models[key].fit(X_train, y_train)

    # Make Predictions:
    predictions = models[key].predict(X_test)

    # Calculate metrics:
    accuracy[key] = accuracy_score(predictions, y_test)
    precision[key] = precision_score(predictions, y_test)
    recall[key] = recall_score(predictions, y_test)
```

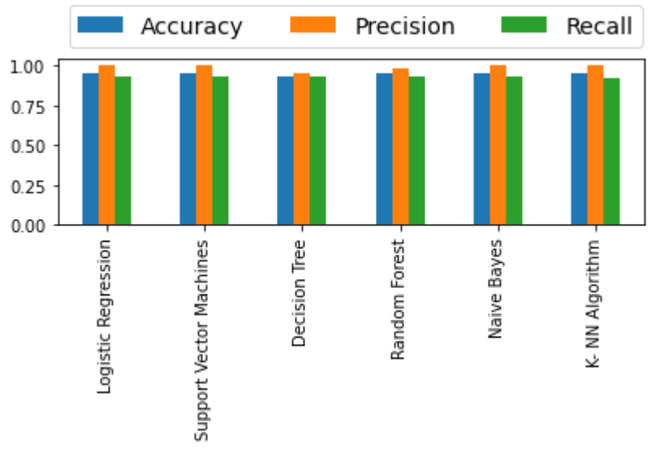
In [39]:
```
import pandas as pd

df_model = pd.DataFrame(index = models.keys(),
                        columns = ['Accuracy', 'Precision', 'Recall'])
df_model['Accuracy'] = accuracy.values()
df_model['Precision'] = precision.values()
df_model['Recall'] = recall.values()

df_model
```

Out[39]:

	Accuracy	Precision	Recall
Logistic Regression	0.958042	1.000000	0.935484
Support Vector Machines	0.958042	1.000000	0.935484
Decision Tree	0.930070	0.954023	0.932584
Random Forest	0.951049	0.988506	0.934783
Naive Bayes	0.958042	1.000000	0.935484
K- NN Algorithm	0.951049	1.000000	0.925532

In [45]:

```
# Visualization of Scores:


ax = df_model.plot.bar()
ax.legend(
    ncol = len(models.keys()),
    bbox_to_anchor = (0, 1),
    loc = 'lower left',
    prop = {'size' : 14}
)

plt.tight_layout()
```

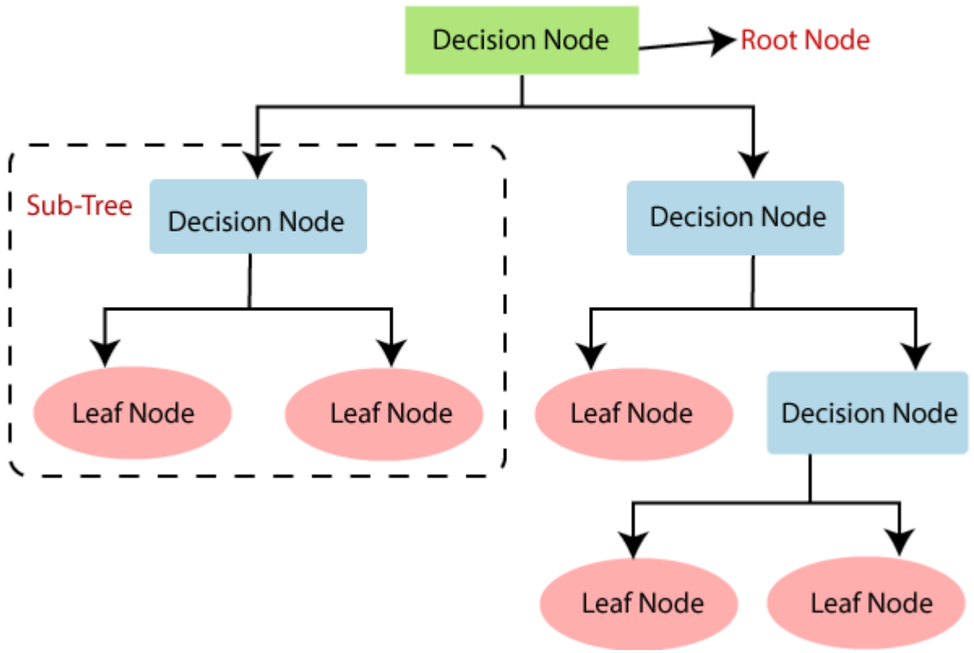


Why use Decision Trees?

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:

Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.

The logic behind the decision tree can be easily understood because it shows a tree-like structure.



`Decision Tree Terminologies`

Root Node: Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.

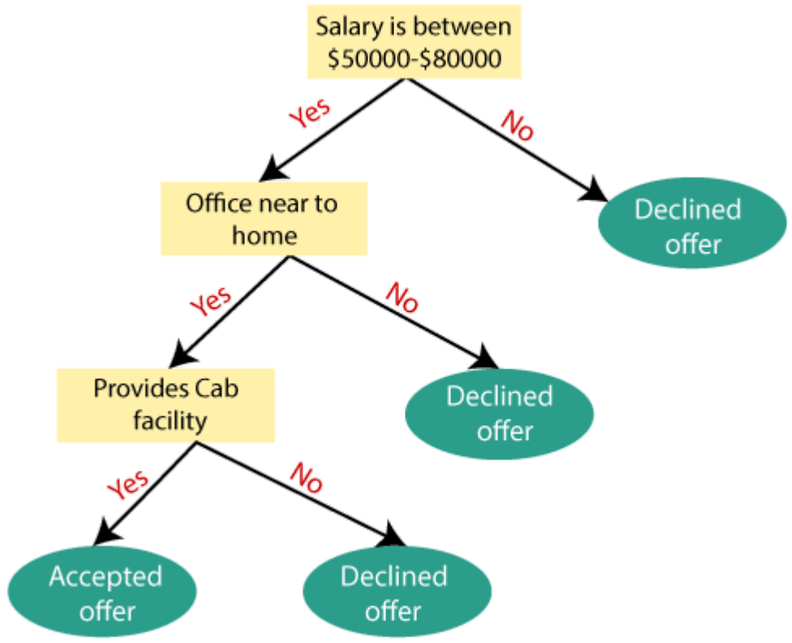
Leaf Node: Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.

Splitting: Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.

Branch/Sub Tree: A tree formed by splitting the tree.

Pruning: Pruning is the process of removing the unwanted branches from the tree. Parent/Child node: The root node of the tree is called the parent node, and other nodes are called the child nodes.

`Example:` Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



`Advantages of the Decision Tree`

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

`Disadvantages of the Decision Tree`

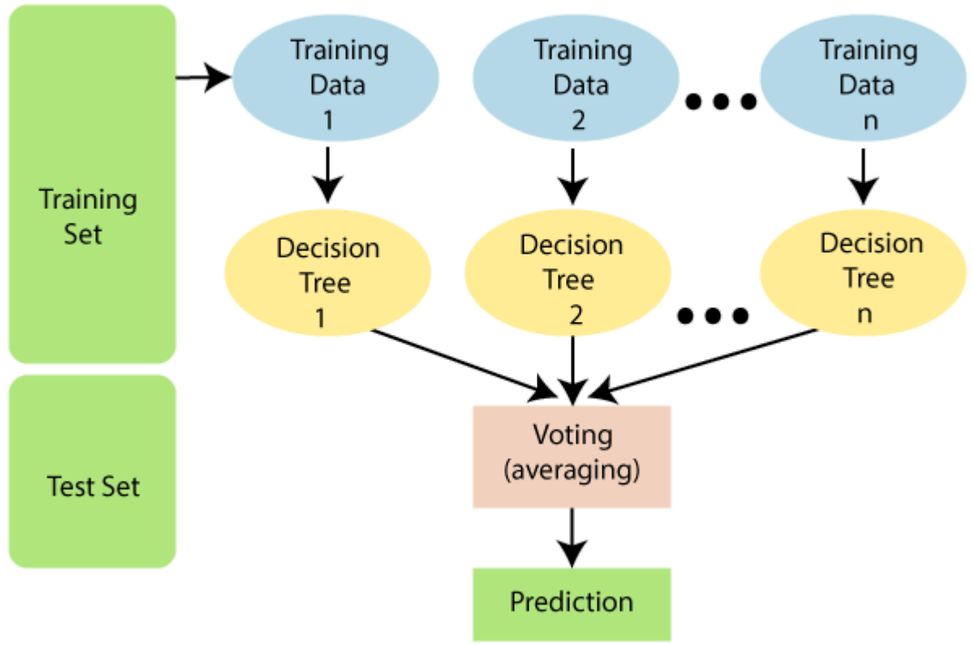
- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the Random Forest algorithm.
- For more class labels, the computational complexity of the decision tree may increase.

`Random Forest Algorithm`

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.



`Applications of Random Forest`

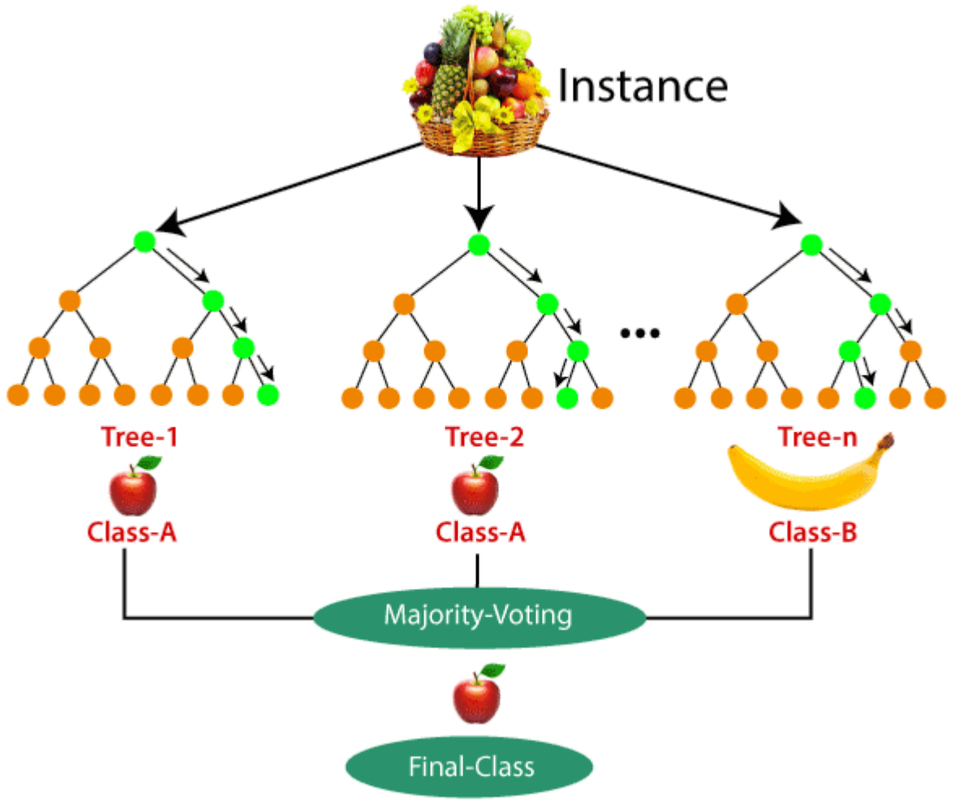
There are mainly four sectors where Random forest mostly used:

Banking: Banking sector mostly uses this algorithm for the identification of loan risk.

Medicine: With the help of this algorithm, disease trends and risks of the disease can be identified.

Land Use: We can identify the areas of similar land use by this algorithm.

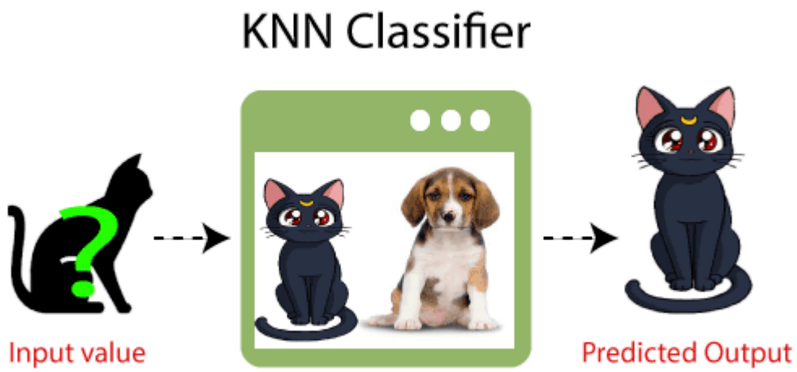
Marketing: Marketing trends can be identified using this algorithm.



K-Nearest Neighbor(KNN) Algorithm for Machine Learning

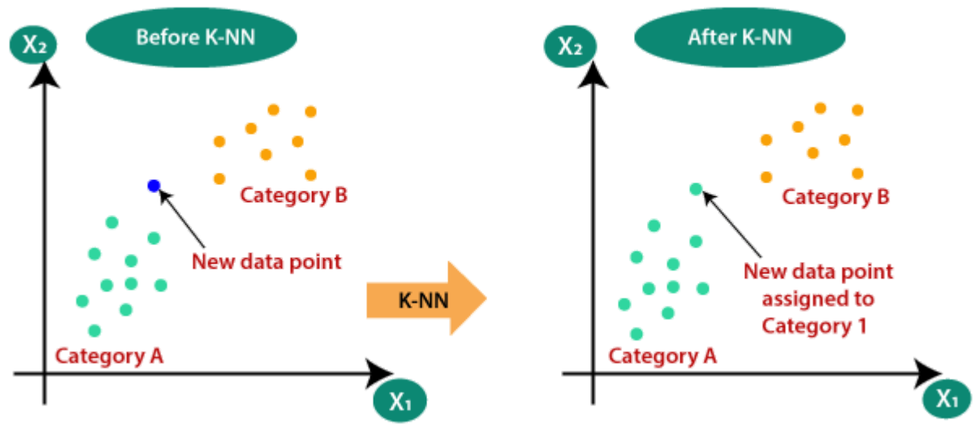
- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
- It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

`Example:` Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.



`Why do we need a K-NN Algorithm?`

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:



`How does K-NN work?` The K-NN working can be explained on the basis of the below algorithm:

Step-1: Select the number K of the neighbors

Step-2: Calculate the Euclidean distance of K number of neighbors

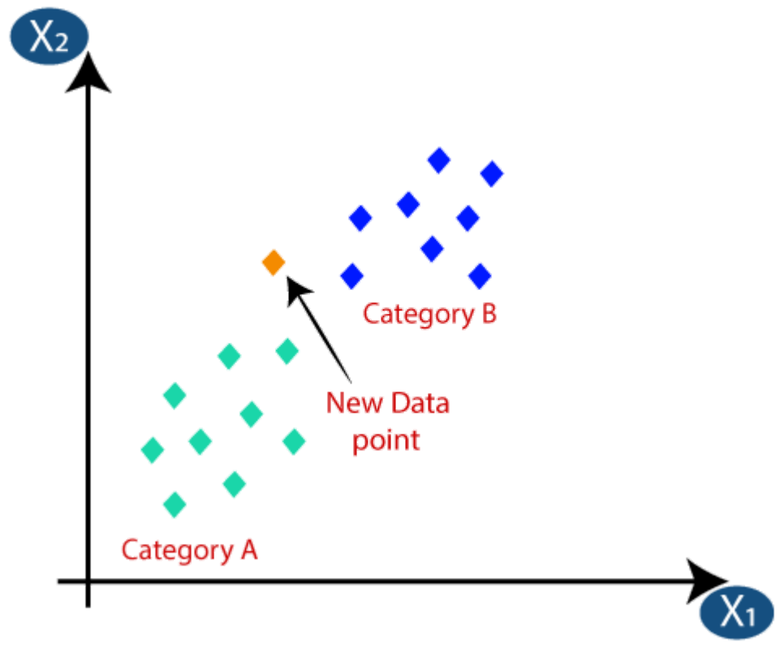
Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.

Step-4: Among these k neighbors, count the number of the data points in each category.

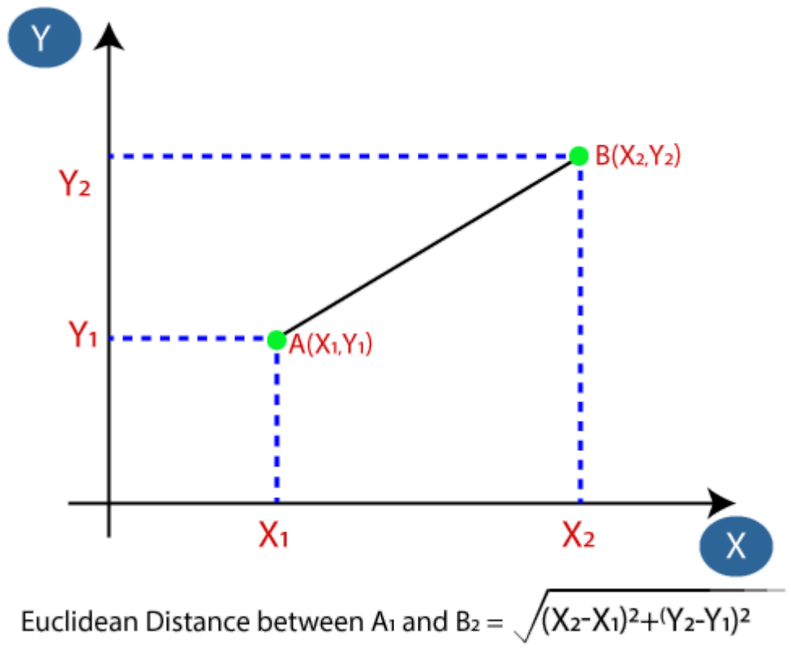
Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.

Step-6: Our model is ready.

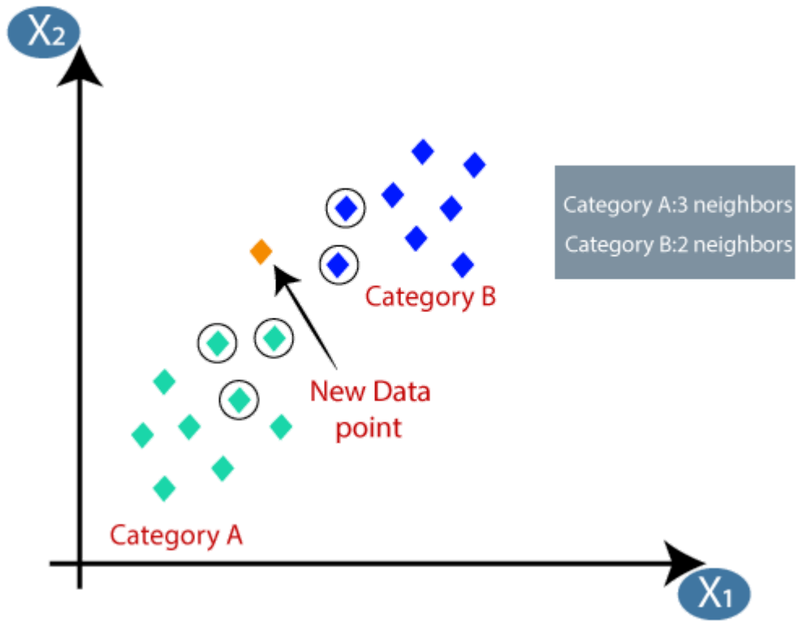
Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the Euclidean distance between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



How to select the value of K in the K-NN Algorithm?

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

Naive Bayes Classifier Algorithm

- Naive Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
- It is mainly used in text classification that includes a high-dimensional training dataset. Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.

- Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

Naïve: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.

Bayes: It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' Theorem:

Bayes' theorem is also known as Bayes' Rule or Bayes' law, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability. The formula for Bayes' theorem is given as:

Naïve Bayes Classifier Algorithm Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Working of Naïve Bayes' Classifier:

Working of Naïve Bayes' Classifier can be understood with the help of the below example:

Suppose we have a dataset of weather conditions and corresponding target variable "Play". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

- Convert the given dataset into frequency tables.
- Generate Likelihood table by finding the probabilities of given features.
- Now, use Bayes theorem to calculate the posterior probability.

Problem: If the weather is sunny, then the Player should play or not?

Solution: To solve this, first consider the below dataset:

Advantages of Naïve Bayes Classifier:

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for text classification problems.

Disadvantages of Naïve Bayes Classifier:

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

Applications of Naïve Bayes Classifier:

- It is used for Credit Scoring.
- It is used in medical data classification.
- It can be used in real-time predictions because Naïve Bayes Classifier is an eager learner.
- It is used in Text classification such as Spam filtering and Sentiment analysis.

In [ ]:

```
1
```