# Assignment No. 1

**Aim:** Data Wrangling: I Perform the following operations using Python on any open source dataset (e.g., data.csv)

1. Import all the required Python Libraries.

2. Locate an open source data from the web (e.g., https://www.kaggle.com). Provide a clear description of the data and its source (i.e., URL of the web site).

3. Load the Dataset into pandas dataframe.

4. Data Preprocessing: check for missing values in the data using pandas isnull(), describe() function to get some initial statistics. Provide variable descriptions. Types of variables etc. Check the dimensions of the data frame.

5. Data Formatting and Data Normalization: Summarize the types of variables by checking the data types (i.e., character, numeric, integer, factor, and logical) of the variables in the data set. If variables are not in the correct data type, apply proper type conversions.

6. Turn categorical variables into quantitative variables in Python. In addition to the codes and outputs, explain every operation that you do in the above steps and explain everything that you do to import/read/scrape the data set.

**Data Wrangling in Python**
Data Wrangling is the process of gathering, collecting, and transforming Raw data into another format for better understanding, decision-making, accessing, and analysis in less time. Data Wrangling is also known as Data Munging.

**Importance Of Data Wrangling**
Data Wrangling is a very important step. The below example will explain its importance as :

Books selling Website want to show top-selling books of different domains, according to user preference. For example, a new user search for motivational books, then they want to show those motivational books which sell the most or having a high rating, etc.

But on their website, there are plenty of raw data from different users. Here the concept of Data Munging or Data Wrangling is used. As we know Data is not Wrangled by System. This process is done by Data Scientists. So, the data Scientist will wrangle data in such a way that they will sort that motivational books that are sold more or have high ratings or user buy this book with these package of Books, etc. On the basis of that, the new user will make choice. This will explain the importance of Data wrangling.

**Data Wrangling in Python** Data Wrangling is a crucial topic for Data Science and Data Analysis. Pandas Framework of Python is used for Data Wrangling. Pandas is

an open-source
library specifically developed for Data Analysis and Data Science. The process like data sorting or filtration, Data grouping, etc.
Data wrangling in python deals with the below functionalities:

1. **Data exploration:** In this process, the data is studied, analyzed and understood by visualizing representations of data.
2. **Dealing with missing values:** Most of the datasets having a vast amount of data contain missing values of *NaN, they are needed to be taken* care of by replacing them with mean, mode, the most frequent value of the column or simply by dropping the row having a *NaN* value.
3. **Reshaping data:** In this process, data is manipulated according to the requirements, where new data can be added or pre-existing data can be modified.
4. **Filtering data:** Some times datasets are comprised of unwanted rows or columns which are required to be removed or filtered
5. **Other:** After dealing with the raw dataset with the above functionalities we get an efficient dataset as per our requirements and then it can be used for a required purpose like data analyzing, machine learning, data visualization, model training etc.

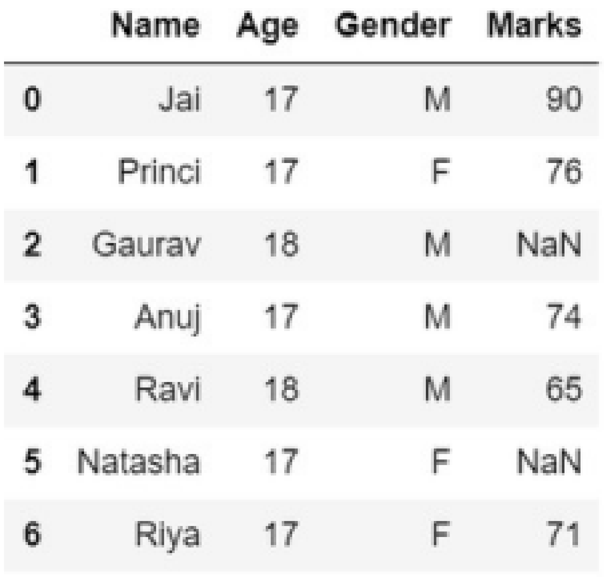
**Below is an example which implements the above functionalities on a raw dataset:**  **Data exploration**, here we assign the data, and then we visualize the data in a tabular format.

Python3

```
# Import pandas package

import pandas as pd



# Assign data

data = {'Name': ['Jai', 'Princi', 'Gaurav',

                        'Anuj', 'Ravi', 'Natasha', 'Riya'],

            'Age': [17, 17, 18, 17, 18, 17, 17],

            'Gender': ['M', 'F', 'M', 'M', 'M', 'F', 'F'],

            'Marks': [90, 76, 'NaN', 74, 65, 'NaN', 71]}
# Convert into DataFrame
```

```
df = pd.DataFrame(data)



# Display data

df
```

**Output:**



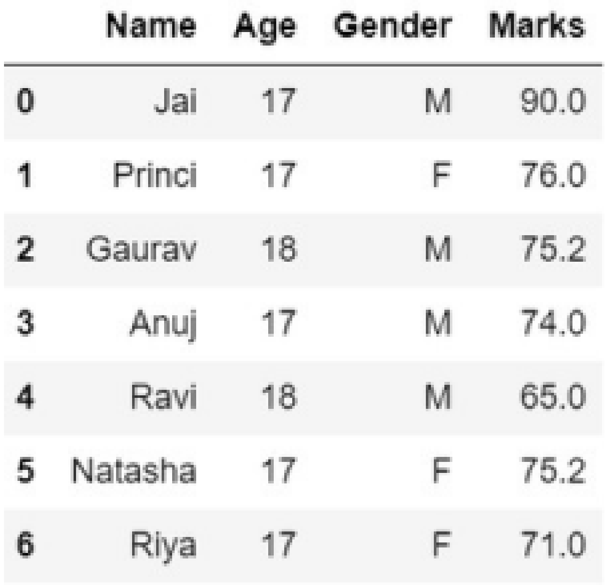
| | Name | Age | Gender | Marks |
|---|---|---|---|---|
| **0** | Jai | 17 | M | 90 |
| **1** | Princi | 17 | F | 76 |
| **2** | Gaurav | 18 | M | NaN |
| **3** | Anuj | 17 | M | 74 |
| **4** | Ravi | 18 | M | 65 |
| **5** | Natasha | 17 | F | NaN |
| **6** | Riya | 17 | F | 71 |

- **Dealing with missing values**, as we can see from the previous output, there are *NaN* values present in the *MARKS* column which are going to be taken care of by replacing them with the column mean.

Python3

```
# Compute average

c = avg = 0

for ele in df['Marks']:
        if str(ele).isnumeric():

                c += 1

                avg += ele
avg /= c



# Replace missing values

df = df.replace(to_replace="NaN",

                        value=avg)



# Display data

df
```

**Output:**



```
   Name     Age  Gender  Marks
0  Jai      17   M       90.0
1  Princi   17   F       76.0
2  Gaurav   18   M       75.2
3  Anuj     17   M       74.0
4  Ravi     18   M       65.0
5  Natasha  17   F       75.2
6  Riya     17   F       71.0
```

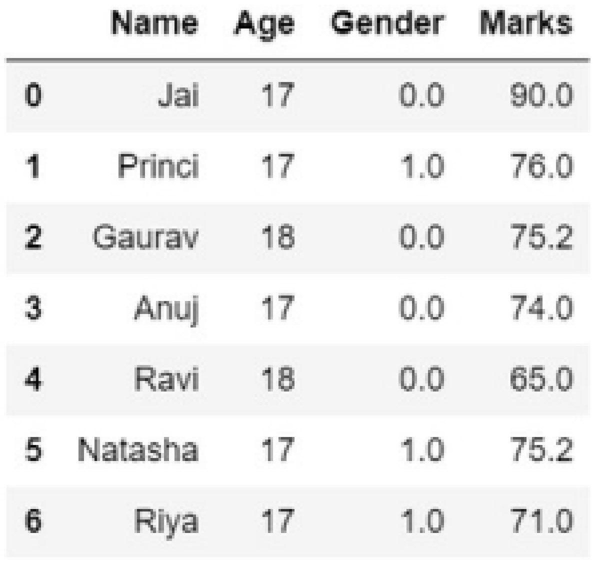
 **Reshaping data**, in the *GENDER* column, we can reshape the data by categorizing them into different numbers.

Python3

```
# Categorize gender

df['Gender'] = df['Gender'].map({'M': 0,

                                           'F': 1, }).astype(float)
```

```
 # Display data

df
```

**Output:**



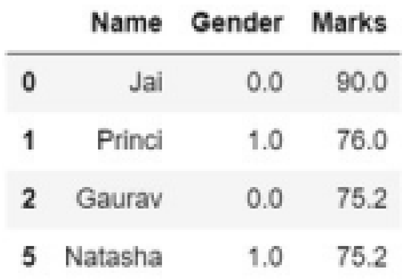
|   | Name | Age | Gender | Marks |
|---|---|---|---|---|
| 0 | Jai | 17 | 0.0 | 90.0 |
| 1 | Princi | 17 | 1.0 | 76.0 |
| 2 | Gaurav | 18 | 0.0 | 75.2 |
| 3 | Anuj | 17 | 0.0 | 74.0 |
| 4 | Ravi | 18 | 0.0 | 65.0 |
| 5 | Natasha | 17 | 1.0 | 75.2 |
| 6 | Riya | 17 | 1.0 | 71.0 |

 **Filtering data**, suppose there is a requirement for the details regarding name, gender, marks  of the top-scoring students. Here we need to remove some unwanted data.

Python3

```
# Filter top scoring students

df = df[df['Marks'] >= 75]
# Remove age row

df = df.drop(['Age'], axis=1)

 # Display data

df
```

**Output:**



|   | Name | Gender | Marks |
|---|---|---|---|
| 0 | Jai | 0.0 | 90.0 |
| 1 | Princi | 1.0 | 76.0 |
| 2 | Gaurav | 0.0 | 75.2 |
| 5 | Natasha | 1.0 | 75.2 |

Hence, we have finally obtained an efficient dataset which can be further used for various purposes.

Now that we know the basics of data wrangling. Below we will discuss various operations using which we can perform data wrangling:

**Wrangling Data Using Merge Operation:**
Merge operation is used to merge raw data and into the desired format.

**Syntax:**
pd.merge( data_frame1,data_frame2, on="field ")

Here the field is the name of the column which is similar on both data-frame.

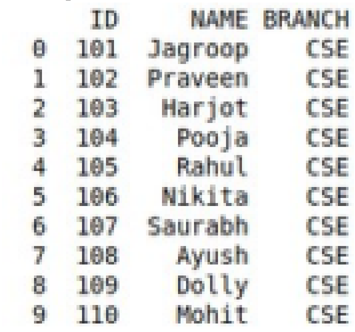
For example: Suppose that a Teacher has two types of Data, first type of Data consist of Details of Students and Second type of Data Consist of Pending Fees Status which is taken from Account Office. So The Teacher will use merge operation here in order to merge the data and provide it meaning. So that teacher will analyze it easily and it also reduces time and effort of Teacher from Manual Merging.

**FIRST TYPE OF DATA:**

Python3

```
# import module
import pandas as pd


 # creating DataFrame for Student Details details =


pd.DataFrame({


         'ID': [101, 102, 103, 104, 105, 106,


                  107, 108, 109, 110],


         'NAME': ['Jagroop', 'Praveen', 'Harjot', 'Pooja', 'Rahul',


                               'Nikita',
                        'Saurabh', 'Ayush', 'Dolly', "Mohit"],

           'BRANCH': ['CSE', 'CSE', 'CSE', 'CSE', 'CSE', 'CSE', 'CSE',

                          'CSE', 'CSE', 'CSE']})



# printing details

print(details)
```

**Output:**



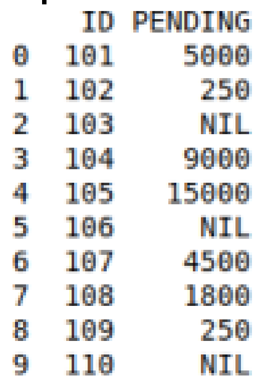
```
    ID     NAME BRANCH
0  101  Jagroop    CSE
1  102  Praveen    CSE
2  103   Harjot    CSE
3  104    Pooja    CSE
4  105    Rahul    CSE
5  106   Nikita    CSE
6  107  Saurabh    CSE
7  108    Ayush    CSE
8  109    Dolly    CSE
9  110    Mohit    CSE
```

**SECOND TYPE OF DATA**

Python3

```
# Import module

import pandas as pd

# Creating Dataframe for Fees_Status

fees_status = pd.DataFrame(

        {'ID': [101, 102, 103, 104, 105,

                    106, 107, 108, 109, 110],

          'PENDING': ['5000', '250', 'NIL',

                          '9000', '15000', 'NIL',
                                    '4500', '1800', '250', 'NIL']})

# Printing fees_status

print(fees_status)
```

**Output:**



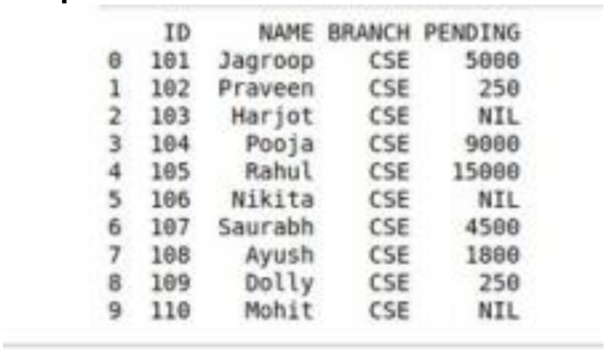
```
     ID PENDING
0   101    5000
1   102     250
2   103     NIL
3   104    9000
4   105   15000
5   106     NIL
6   107    4500
7   108    1800
8   109     250
9   110     NIL
```

**WRANGLING DATA USING MERGE OPERATION:**

Python3

```
# Import module

import pandas as pd

 # Creating Dataframe

details = pd.DataFrame({

     'ID': [101, 102, 103, 104, 105,

               106, 107, 108, 109, 110],

     'NAME': ['Jagroop', 'Praveen', 'Harjot', 'Pooja', 'Rahul',

                         'Nikita',

                 'Saurabh', 'Ayush', 'Dolly', "Mohit"],

     'BRANCH': ['CSE', 'CSE', 'CSE', 'CSE', 'CSE', 'CSE', 'CSE',

                   'CSE', 'CSE', 'CSE']})
 # Creating Dataframe

fees_status = pd.DataFrame(

        {'ID': [101, 102, 103, 104, 105,

                    106, 107, 108, 109, 110],

          'PENDING': ['5000', '250', 'NIL',

                        '9000', '15000', 'NIL',

                        '4500', '1800', '250', 'NIL']})
# Merging Dataframe

print(pd.merge(details, fees_status, on='ID'))
```

**Output:**



```
    ID     NAME BRANCH PENDING
0  101  Jagroop    CSE    5000
1  102  Praveen    CSE     250
2  103   Harjot    CSE     NIL
3  104    Pooja    CSE    9000
4  105    Rahul    CSE   15000
5  106   Nikita    CSE     NIL
6  107  Saurabh    CSE    4500
7  108    Ayush    CSE    1800
8  109    Dolly    CSE     250
9  110    Mohit    CSE     NIL
```

**Wrangling Data using Grouping Method**

The grouping method in Data analysis is used to provide results in terms of various groups taken  out from Large Data. This method of pandas is used to group the outset of data from the large  data set.

Example: There is a Car Selling company and this company have different Brands of various Car Manufacturing Company like Maruti, Toyota, Mahindra, Ford, etc. and have data where different cars are sold in different years. So the Company wants to wrangle only that data where cars are  sold during the year 2010. For this problem, we use another Wrangling technique that  is *groupby()* method.

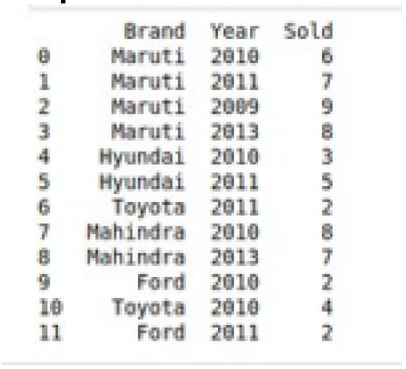
**CARS SELLING DATA:**

Python3

```
# Import module
```

```
import pandas as pd

# Creating Data

car_selling_data = {'Brand': ['Maruti', 'Maruti', 'Maruti', 'Maruti', 'Hyundai',
                              'Hyundai',
                                          'Toyota', 'Mahindra', 'Mahindra',
                                          'Ford', 'Toyota', 'Ford'],
                    'Year': [2010, 2011, 2009, 2013,
                             2010, 2011, 2011, 2010,
                             2013, 2010, 2010, 2011],
                    'Sold': [6, 7, 9, 8, 3, 5,
                             2, 8, 7, 2, 4, 2]}

 # Creating Dataframe of car_selling_data

df = pd.DataFrame(car_selling_data)

 # printing Dataframe

print(df)
```

**Output:**



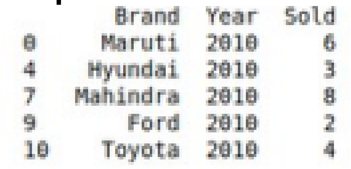
```
       Brand  Year  Sold
0     Maruti  2010     6
1     Maruti  2011     7
2     Maruti  2009     9
3     Maruti  2013     8
4    Hyundai  2010     3
5    Hyundai  2011     5
6     Toyota  2011     2
7   Mahindra  2010     8
8   Mahindra  2013     7
9       Ford  2010     2
10    Toyota  2010     4
11      Ford  2011     2
```

**DATA OF THE YEAR 2010:**

Python3

```
# Import module

import pandas as pd

# Creating Data

car_selling_data = {'Brand': ['Maruti', 'Maruti', 'Maruti', 'Maruti', 'Hyundai', 'Hyundai',

                                                        'Toyota', 'Mahindra', 'Mahindra',

                                                        'Ford', 'Toyota', 'Ford'],

                             'Year': [2010, 2011, 2009, 2013,

                                                 2010, 2011, 2011, 2010,

                                                 2013, 2010, 2010, 2011],

                             'Sold': [6, 7, 9, 8, 3, 5,

                                                 2, 8, 7, 2, 4, 2]}

 # Creating Dataframe for Provided Data

df = pd.DataFrame(car_selling_data)

 # Group the data when year = 2010

grouped = df.groupby('Year')

print(grouped.get_group(2010))
```

**Output:**



```
       Brand  Year  Sold
0     Maruti  2010     6
4    Hyundai  2010     3
7   Mahindra  2010     8
9       Ford  2010     2
10    Toyota  2010     4
```

**Wrangling data by removing Duplication:**
Pandas *duplicates()* method helps us to remove duplicate values from Large Data. An important  part of Data Wrangling is removing Duplicate values from the large data set.

**Syntax:**
DataFrame.duplicated(subset=None, keep='first')

Here subset is the column value where we want to remove Duplicate value.

In *keep*, we have 3 options :
- if *keep ='first'* then the first value is marked as original rest all values if occur will be removed as it is considered as duplicate.
- if *keep='last'* then the last value is marked as original rest all above same values will be removed as it is considered as duplicate values.
- if *keep ='false'* the all the values which occur more than once will be removed as all considered as a duplicate value.

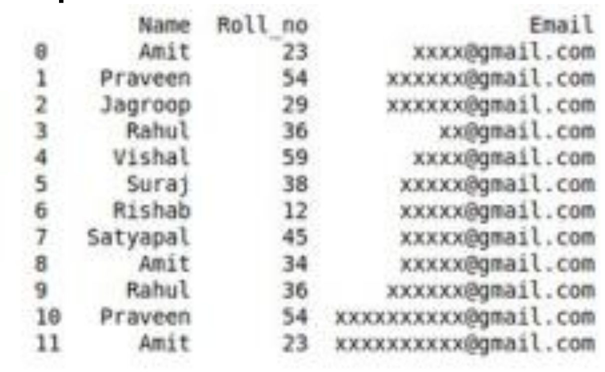
For example, A University will organize the event. In order to participate Students have to fill their details in the online form so that they will contact them. It may be possible that a student will fill the form multiple time. It may cause difficulty for the event organizer if a single student will fill multiple entries. The Data that the organizers will get can be Easily Wrangles by removing duplicate values.

**DETAILS STUDENTS DATA WHO WANT TO PARTICIPATE IN THE EVENT:**

Python3

```
# Import module

import pandas as pd

# Initializing Data

student_data = {'Name': ['Amit', 'Praveen', 'Jagroop',

                         'Rahul', 'Vishal', 'Suraj',

                         'Rishab', 'Satyapal', 'Amit',

                         'Rahul', 'Praveen', 'Amit'],

                'Roll_no': [23, 54, 29, 36, 59, 38,

                            12, 45, 34, 36, 54, 23],
                'Email': ['xxxx@gmail.com', 'xxxxxx@gmail.com',

                          'xxxxxx@gmail.com', 'xx@gmail.com',

                          'xxxx@gmail.com', 'xxxxx@gmail.com',
                                        'xxxxx@gmail.com', 'xxxxx@gmail.com',

                                        'xxxxx@gmail.com', 'xxxxxx@gmail.com',

                                          'xxxxxxxxxx@gmail.com', 'xxxxxxxxxx@gmail.com']}

 # Creating Dataframe of Data

df = pd.DataFrame(student_data)

 # Printing Dataframe

print(df)
```

**Output:**



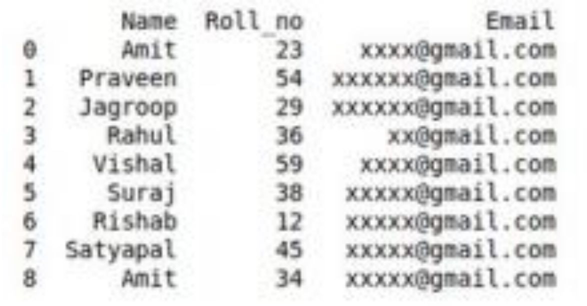
```
        Name  Roll_no                 Email
0       Amit       23        xxxx@gmail.com
1    Praveen       54      xxxxxx@gmail.com
2    Jagroop       29      xxxxxx@gmail.com
3      Rahul       36          xx@gmail.com
4     Vishal       59        xxxx@gmail.com
5      Suraj       38       xxxxx@gmail.com
6     Rishab       12       xxxxx@gmail.com
7   Satyapal       45       xxxxx@gmail.com
8       Amit       34       xxxxx@gmail.com
9      Rahul       36      xxxxxx@gmail.com
10   Praveen       54  xxxxxxxxxx@gmail.com
11      Amit       23  xxxxxxxxxx@gmail.com
```

**DATA WRANGLED BY REMOVING DUPLICATE ENTRIES:**

Python3

```
# import module
import pandas as pd

# initializing Data

student_data = {'Name': ['Amit', 'Praveen', 'Jagroop',

                                         'Rahul', 'Vishal', 'Suraj',

                                         'Rishab', 'Satyapal', 'Amit',

                                         'Rahul', 'Praveen', 'Amit'],
                    'Roll_no': [23, 54, 29, 36, 59, 38,

                                12, 45, 34, 36, 54, 23],

                           'Email': ['xxxx@gmail.com', 'xxxxxx@gmail.com',

                                'xxxxxx@gmail.com', 'xx@gmail.com',

                                'xxxx@gmail.com', 'xxxxx@gmail.com',

                                'xxxxx@gmail.com', 'xxxxx@gmail.com',

                                'xxxxx@gmail.com', 'xxxxxx@gmail.com',

                                  'xxxxxxxxxx@gmail.com', 'xxxxxxxxxx@gmail.com']}

 # creating dataframe

df = pd.DataFrame(student_data)

 # Here df.duplicated() list duplicate Entries in ROllno. # So that ~(NOT) is placed in

order to get non duplicate values. non_duplicate = df[~df.duplicated('Roll_no')]

 # printing non-duplicate values

print(non_duplicate)
```

**Output:**



```
       Name  Roll_no             Email
0      Amit       23    xxxx@gmail.com
1   Praveen       54  xxxxxx@gmail.com
2   Jagroop       29  xxxxxx@gmail.com
3     Rahul       36      xx@gmail.com
4    Vishal       59    xxxx@gmail.com
5     Suraj       38   xxxxx@gmail.com
6    Rishab       12   xxxxx@gmail.com
7  Satyapal       45   xxxxx@gmail.com
8      Amit       34   xxxxx@gmail.com
```

**Conclusion:** Hence we have thourouly studied how to perform the following operations using  Python on any open source dataset (e.g., data.csv)

1. Import all the required Python Libraries.

2. Locate an open source data from the web (e.g., https://www.kaggle.com). Provide a clear  description of the data and its source (i.e., URL of the web site).

3. Load the Dataset into pandas data frame.

4. Data Preprocessing: check for missing values in the data using pandas is null(), describe()  function to get some initial statistics. Provide variable descriptions. Types of variables etc. Check the dimensions of the data frame.

5. Data Formatting and Data Normalization: Summarize the types of variables by checking the  data types (i.e., character, numeric, integer, factor, and logical) of the variables in the data set. If  variables are not in the correct data type, apply proper type conversions.

6. Turn categorical variables into quantitative variables in Python. In addition to the codes and  outputs, explain every operation that you do in the above steps and explain everything that you  do to import/read/scrape the data set.

# Group A Assignment 2

**Title:** Create an "Academic performance" dataset of students and perform the given operations using Python.

**Objective:**

1. Scan all variables for missing values and inconsistencies. If there are missing values and/or inconsistencies, use any of the suitable techniques to deal with them.

2. Scan all numeric variables for outliers. If there are outliers, use any of the suitable techniques to deal with them.
3. Apply data transformations on at least one of the variables. The purpose of this transformation should be one of the following reasons: to change the scale for better understanding of the variable, to convert a non-linear relation into a linear one, or to decrease the skewness and convert the distribution into a normal distribution.

**Theory:**

**#Reading the dataset in a dataframe using Pandas**

```
df = pd.read_csv('https://raw.githubusercontent.com/basilatawneh/Students
Academic-Performance-Dataset-xAPI-Edu-Data-/master/xAPI-Edu-Data.csv')
```

**#describe the given data**

```
print(df. describe())
```

**#Display first 10 rows of data**

```
print(df.head(10))
```

**#Missing values In Pandas missing data is represented by two values:**

**None**: None is a Python singleton object that is often used for missing data in Python code.
**NaN**:NaN (an acronym for Not a Number), is a special floating-point value recognized by all systems

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

**# identify missing items**

```
print(df.isnull())
```

**#outlier data items :-**

An outlier is an observation of a data point that lies an abnormal distance from other values in a given population. (odd man out)

- Like in the following data point (Age)
    - 18,22,45,67,89,**125,**30

It is an abnormal observation during the Data Analysis stage, that data point lies far away from other values.

- List of Animals
    - cat, fox, rabbit, **fish**

**Methods**

1. **Using Box Plot**
   It captures the summary of the data effectively and efficiently with only a simple box and whiskers.

```
# Box Plot
import seaborn as sns
sns.boxplot(df_boston['DIS'])
```

2. **Using ScatterPlot**
   It is used when you have paired numerical data, or when your dependent variable has multiple values for each reading independent variable, or when trying to determine the relationship between the two variables.

```
# Scatter plot
fig, ax = plt.subplots(figsize = (18,10))
ax.scatter(df_boston['INDUS'], df_boston['TAX'])

# x-axis label
ax.set_xlabel('(Proportion non-retail business acres)/(town)')

# y-axis label
ax.set_ylabel('(Full-value property-tax rate)/( $10,000)')
plt.show()
```

3. **Z-score**
   It is also called a standard score. This value/score helps to understand that how far is the data point from the mean. And after setting up a threshold value one can utilize z score values of data points to define the outliers.
   *Zscore = (data_point -mean) / std. deviation*

```
# Z score
from scipy import stats
import numpy as np

z = np.abs(stats.zscore(df_boston['DIS']))
print(z)
```

Now to define an outlier threshold value is chosen which is generally 3.0. As 99.7%

of the data points lie between +/- 3 standard deviation (using Gaussian Distribution approach).

```
threshold = 3
# Position of the outlier
print(np.where(z > 3))
```

**4. IQR (Inter Quartile Range)**

Inter Quartile Range approach to finding the outliers is the most commonly used and most trusted approach used in the research field.

*IQR = Quartile3 – Quartile1*

```
# IQR
Q1 = np.percentile(df_boston['DIS'], 25,
interpolation = 'midpoint')

Q3 = np.percentile(df_boston['DIS'], 75,
interpolation = 'midpoint')
IQR = Q3 - Q1
```

To define the outlier base value is defined above and below datasets normal range namely Upper and Lower bounds, define the upper and the lower bound (1.5*IQR value is considered) :

*upper = Q3 +1.5*IQR*

*lower = Q1 – 1.5*IQR*

```
# Above Upper bound
upper = df_boston['DIS'] >= (Q3+1.5*IQR)

print("Upper bound:",upper)
print(np.where(upper))

# Below Lower bound
lower = df_boston['DIS'] <= (Q1-1.5*IQR)
print("Lower bound:", lower)
print(np.where(lower))
```

**Removing the outliers**

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

*dataframe.drop( row_index, inplace = True*

The above code can be used to drop a row from the dataset given the row_indexes to be dropped. Inplace =True is used to tell python to make the required change in the original dataset. row_index can be only one value or list of values or NumPy array but it must be one dimensional.

**To change the scale for better understanding of the variable**

Consider,

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline



df = pd.DataFrame({
  'Income': [15000, 1800, 120000, 10000],
  'Age': [25, 18, 42, 51],
  'Department': ['HR','Legal','Marketing','Management']
 })
```

Before directly applying any feature transformation or scaling technique, we need to remember the categorical column: Department and first deal with it. This is because we cannot scale non-numeric values.

For that, we 1st create a copy of our dataframe and store the numerical feature names in a list, and their values as well:

```
df_scaled = df.copy()
col_names = ['Income', 'Age']
features = df_scaled[col_names]
```

**MinMax Scaler**

It just scales all the data between 0 and 1. The formula for calculating the scaled value is

$$x\_scaled = (x – x\_min)/(x\_max – x\_min)$$

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df_scaled[col_names] = scaler.fit_transform(features.values)
```

You can see how the values were scaled. The minimum value among the columns became 0, and the maximum value was changed to 1, with other values in between. However, suppose we don't want the income or age to have values like 0. Let us take the range to be (5, 10) from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(5, 10))

df_scaled[col_names] = scaler.fit_transform(features.values)

df_scaled

**Standard Scaler**

For each feature, the Standard Scaler scales the values such that the mean is 0 and the standard deviation is 1(or the variance).

x_scaled = x – mean/std_dev

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

df_scaled[col_names] = scaler.fit_transform(features.values)

df_scaled

**MaxAbsScaler**

In simplest terms, the MaxAbs scaler takes the absolute maximum value of each column and divides each value in the column by the maximum value.

Thus, it first takes the absolute value of each value in the column and then takes the maximum value out of those. This operation scales the data between the range [-1, 1].

df["Balance"] = [100.0, -263.0, 2000.0, -5.0]

from sklearn.preprocessing import MaxAbsScaler

scaler = MaxAbsScaler()

df_scaled[col_names] = scaler.fit_transform(features.values)

df_scaled

**Robust Scaler**

The Robust Scaler, as the name suggests is not sensitive to outliers. This scaler

1. removes the median from the data
2. scales the data by the InterQuartile Range(IQR)

**x_scaled = (x – Q1)/(Q3 – Q1)**

from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()
df_scaled[col_names] = scaler.fit_transform(features.values)

```
df_scaled
```

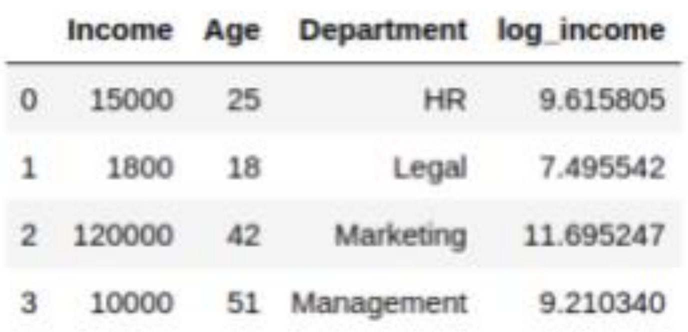
**To decrease the skewness and convert the distribution into a normal distribution**

**Log Transform**

It is primarily used to convert a skewed distribution to a normal distribution/less-skewed distribution. In this transform, we take the log of the values in a column and use these values as the column instead.

```
df['log_income'] = np.log(df['Income'])
# We created a new column to store the log values
```

This is how the dataframe looks like:



| | Income | Age | Department | log_income |
|---|---|---|---|---|
| 0 | 15000 | 25 | HR | 9.615805 |
| 1 | 1800 | 18 | Legal | 7.495542 |
| 2 | 120000 | 42 | Marketing | 11.695247 |
| 3 | 10000 | 51 | Management | 9.210340 |

```
df['log_income'].plot.hist(bins = 5)
```

**Conclusion:** Thus we scan all variables for missing values & inconsistencies, outliers and applied suitable technique to deal with them. We also applied data transformation on variables.

# Group A Assignment 3

**Title:**
Provide summary statistics (mean, median, minimum, maximum, standard deviation) for a dataset (age, income etc.) with numeric variables grouped by one of the qualitative (categorical) variable.

**Objective:**

1. If your categorical variable is age groups and quantitative variable is income, then provide summary statistics of income grouped bythe age groups.
2. Create a list that contains a numeric value for each response to the categorical variable. 3. Display some basic statistical details like percentile, mean, standard deviation etc. of the species of 'Iris-setosa', 'Iris-versicolor' and 'Iris- versicolor' of iris.csv dataset.

**Theory:**
**What is Statistics?**

✓ Statistics is a branch of mathematics that deals with collecting, analyzing, interpreting, and visualizing empirical data.

✓ Descriptive statistics and inferential statistics are the two major areas of statistics.

✓ Descriptive statistics are for describing the properties of sample and population data (what has happened).

✓ Inferential statistics use those properties to test hypotheses, reach conclusions, and make predictions (what can you expect).

**Use of Statistics in Data Science**

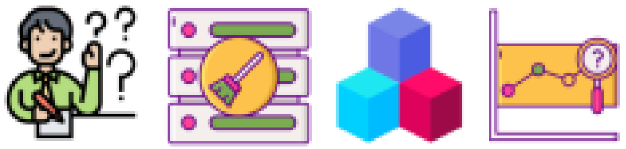
✓ Asking questions about the data

✓ Cleaning and preprocessing the data

✓ Selecting the right features

✓ Model evaluation

✓ Model prediction



**Mean**

✓ The arithmetic mean of a given data is the sum of all observations divided by the number of observations.

✓ For example, a cricketer's scores in five ODI matches are as follows: 12, 34, 45, 50, 24. To find his average score in a match, we calculate the arithmetic mean of data using the mean formula:

$$\text{Mean} = \frac{\textit{Sum of terms}}{\textit{Number of terms}}$$

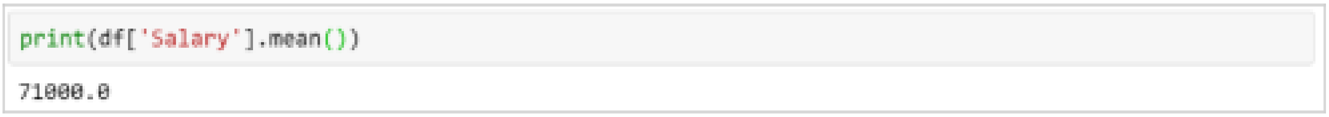
✓ Mean = Sum of all observations/Number of observations

Mean = (12 + 34 + 45 + 50 + 24)/5

Mean = 165/5 = 33

Mean is denoted by x̄ (pronounced as x bar).

To find the mean or the average salary of the employees, you can use the mean() functions in Python.



```
print(df['Salary'].mean())
```

```
71000.0
```

**Mode**

✓ The Mode refers to the most frequently occurring value in your data.

✓ You find the frequency of occurrence of each number and the number with the highest frequency is your mode. If there are no recurring numbers, then there is no mode in the data.

✓ Using the mode, you can find the most commonly occurring point in your data. This is helpful when you have to find the central tendency of categorical values, like the flavor of the most popular chip sold by a brand. You cannot find the average based on the orders; instead, you choose the chip flavor with the highest orders.

✓ Usually, you can count the most frequently occurring values and get your mean. But this only works when the values are discrete. Now, again take the example of class marks.

✓ Example: Take the following marks of students :

Marks = 35, 40, 45, 49, 34, 47, 39, 25, 19, 35, 28, 48

Over here, the value 35 occurs the most frequently and hence is the mode.

✓ But what if the values are categorical? In that case, you must use the formula below:

$$\text{Mode} = l + \left(\frac{f1 - f0}{2f1 - f0 - f2}\right) \times h$$

Where,

l = lower limit of modal class

h = lower limit of preceding modal class

f1 = frequency of modal class

f0 = frequency of class preceding modal class

f2 = frequency of class succeeding modal class

The modal class is simply the class with the highest frequency. Consider the range of frequencies given for the marks obtained by students in a class:

| Marks | 10-20 | 20-30 | 30-40 | 40-50 |
|---|---|---|---|---|
| Number of Students | 1 | 3 | 5 | 4 |

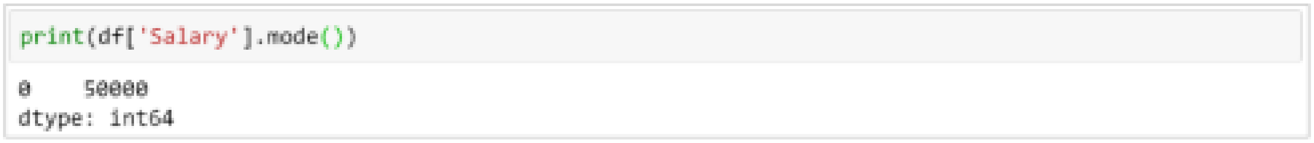
In this case, you can see that class 30-40 has the highest frequency, hence it is the modal class. The remaining values are as follows: l = 30, h = 20, f1 = 5, f0 = 3, f2 = 4

In that case, the mode becomes :

$$\text{Mode} = 30 + \left(\frac{5-3}{2*5-3-4}\right) \times 20$$

$$= 43.33$$

Hence, the mark which occurs most frequently is 43.33. The mode of salary from the salary data frame can be calculated as:



```
print(df['Salary'].mode())
```

```
0    50000
dtype: int64
```

**Median**

✓ Median refers to the middle value of a data. To find the median, you first sort the data in either ascending or descending order or then find the numerical value present in the middle of your data.

✓ It can be used to figure out the point around which the data is centered. It divides the data into two halves and has the same number of data points above and below.

✓ The median is especially useful when the data is skewed data. That is, it has high data distribution towards one side. In this case, the average wouldn't give you a fair mid-value but would lean more towards the higher values. In this case, you can use the middle data point as the central point instead.

✓ Consider n terms X_1, X_2, X_3,………… X_n. The basic formula for the median is by dividing the total number of observations by 2. This works fine when you have an odd number of terms because you will have one middle term and the same number of terms above and below. For an even number of terms, consider the two middle terms and find their average.

$$\text{Median} = \frac{n+1}{2} \text{ th term }, \; n = \text{odd}$$

$$\left\{ \frac{n}{2} \text{ th term} + \frac{n}{2} + 1 \text{ th term} \right\} / 2 \quad , n = \text{even}$$

Example: Consider following are students marks

Marks = 35, 40, 45, 49, 34, 47, 39, 25, 19, 35, 28, 48

To find the middle term, you first have to sort the data or arrange the data in ascending or descending order. This ensures that consecutive terms are next to each other.

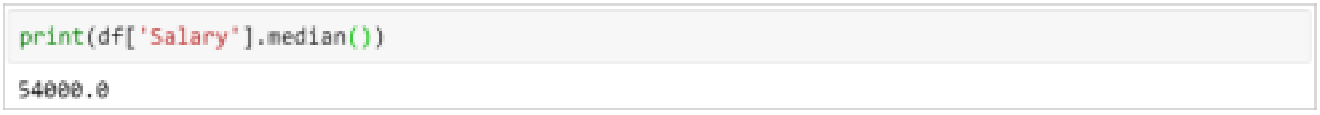
Sorted Marks = 19, 25, 28, 30, 34, 35, 39, 40, 45, 47, 48, 49

You can see that we have 12 data points, so use the median formula for even numbers.

$$\text{Median} = \left\{\left(\frac{12}{2} \text{ th term}\right) + \left(\frac{12}{2} + 1 \text{ th term}\right)\right\} / 2$$
$$= (6^{th} + 7^{th})/2 = (35 + 39)/2$$
$$= 37$$

So, the middle term in the range of marks is 37. This means that the other marks lie in a frequency range of around 37.

The median() function in Python can help you find the median value of a column. From the salary data frame, you can find the median salary as:



```
print(df['Salary'].median())
```

```
54000.0
```

**Standard Deviation and Variance**

✓ Deviation just means how far from the normal

✓ The Standard Deviation is a measure of how spread out numbers are.

✓ Its symbol is σ (the greek letter sigma)
✓ The formula is easy: it is the square root of the Variance.

$$\sigma^2 = \frac{\sum_{i=1}^{n} (x_i - \mu)^2}{n}$$

✓ The Variance is defined as: The average of the squared differences from the Mean.

✓ To calculate the variance follow these steps:

1. Work out the Mean (the simple average of the numbers)
2. Then for each number: subtract the Mean and square the result (the squared difference).
3. Then work out the average of those squared differences

✓ Variance is used to measure the variability in the data from the mean.

To calculate the Variance, take each difference, square it, and then average the result: Variance =

$$\begin{aligned}\sigma^2 &= \frac{206^2 + 76^2 + (-224)^2 + 36^2 + (-94)^2}{5}\\ &= \frac{42436 + 5776 + 50176 + 1296 + 8836}{5}\\ &= \frac{108520}{5}\\ &= 21704\end{aligned}$$

So the Variance is 21,704

And the Standard Deviation is just the square root of Variance, so:

Standard Deviation =

$$\begin{aligned}\sigma &= \sqrt{21704}\\ &= 147.32\ldots\\ &= \mathbf{147}\ \textit{(to the nearest mm)}\end{aligned}$$

**Finding statistical information of the iris flower dataset**

Iris flower has three species - Setosa, Versicolor and Virginica

**Load the dataset**

```
url =
'https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv'
col_name = ['sepal-length','sepal-width','petal-length','petal-width','class']
df = pd.read_csv(url, names = col_name)
```

**Display some information of the data**

```
dataset.shape
dataset.head()
dataset.info()
```

**Display statistical information of the data**

```
dataset.describe()
```

Output:

**groupby()**

A groupby operation involves some combination of splitting the object, applying a function, and combining the results. This can be used to group large amounts of data and compute operations on these groups.

**Example:**

```
df = pd.DataFrame({'Animal': ['Falcon', 'Falcon',
... 'Parrot', 'Parrot'],
... 'Max Speed': [380., 370., 24., 26.]})
>>> df
 Animal Max Speed
0 Falcon 380.0
 1 Falcon 370.0
2 Parrot 24.0
3 Parrot 26.0
>>> df.groupby(['Animal']).mean()
 Max Speed
Animal
Falcon 375.0
Parrot 25.0
```

**Conclusion:** Thus, we performed summary statistics (mean, median, minimum, maximum, standard deviation) for a dataset with numeric variables grouped by the qualitative (categorical) variable.

# Group A Assignment 4

**Title:**
Create a Linear Regression Model using Python/R to predict home prices using Boston Housing Dataset (https://www.kaggle.com/c/boston-housing).

**Objective:**

1. To predict the value of prices of the house using the given features.

**Theory:**

**What is linear Regression?**

Linear Regression is the supervised Machine Learning model in which the model finds the best fit linear line between the independent and dependent variable i.e it finds the linear relationship between the dependent and independent variable.

**Linear Regression on Boston Housing Dataset**

The Housing dataset which contains information about different houses in Boston. This data was originally a part of UCI Machine Learning Repository and has been removed now. We can also access this data from the scikit-learn library. There are 506 samples and 14 feature variables in this dataset. The objective is to predict the value of prices of the house using the given features.

The problem that we are going to solve here is that given a set of features that describe a house in Boston, our machine learning model must predict the house price. To train our machine learning model with boston housing data, we will be using scikit-learn's boston dataset.

In this dataset, each row describes a boston town or suburb. There are 506 rows and 14 attributes (features) with a target column (MEDV).

Now, split the data into independent variables (X's) and dependent variable (Y)data sets. The data will be stored in df_x for the dependent variables and df_y for the dependent variable.

```
#Transform the data set into a data frame
#NOTE: boston.data = the data we want,
# boston.feature_names = the column names of the data
# boston.target = Our target variable or the price of the houses
df_x = pd.DataFrame(boston.data, columns = boston.feature_names)
df_y = pd.DataFrame(boston.target)
```

Initialize the Linear Regression model, split the data into 67% training and 33% testing data, and then train the model with the training data set that contains the independent variables.

```
#Initialize the linear regression model
reg = linear_model.LinearRegression()#Split the data into 67% training and 33% testing
data #NOTE: We have to split the dependent variables (x) and the target or independent
variable  (y)
x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, test_size=0.33,
random_state=42)#Train our model with the training data
reg.fit(x_train, y_train)
```

Get the estimated coefficients for the linear regression model

```
print(reg.coef_)
```

Now that we are done training the linear regression model and looking at the coefficients that describe the linear function, let’s print the model's predictions (what it thinks the values will be for houses) on the test data.

```
#print our price predictions on our test data
y_pred = reg.predict(x_test)
print(y_pred)
```

We want to know what was the actual values for that test data set, so I will print those values to the screen, but first I will print at least one row from the model's prediction, just to make it a little easier to compare data.

```
#Print the the prediction for the third row of our test data actual price =
13.6 y_pred[2]#print the actual price of houses from the testing data set
y_test[0]
```

To check the model's performance/accuracy I will use a metric called mean squared error (MSE). This measurement is simple to implement and easy to understand. The MSE is a measure of the quality of an estimator — it is always non-negative, and values closer to zero indicate a better fit. Usually you want to evaluate your model with other metrics as well to truly get an idea of how well your model performs. I’ll do this two different ways, one  using numpy and the other using sklearn.metrics.

```
# Two different ways to check model performance/accuracy using,
# mean squared error which tells you how close a regression line is to a set of points.
# 1. Mean squared error by numpy
print(np.mean((y_pred-y_test)**2))

# 2. Mean squared error by sklearn
# Resource: https://stackoverflow.com/questions/42453875/precision-score-and-accuracy
score-showing-value-error?rq=1
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_pred))
```

**Conclusion:** Thus, we predicted the value of prices of the house using the given features.

**Group A Assignment 5**

**Title:**

Implement logistic regression using Python/R to perform classification on Social_Network_Ads.csv dataset.

**Objective:**

Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.

**Theory:**

**What is logistic Regression?**

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression). Logistic regression is basically a supervised classification algorithm. In a classification problem, the target variable(or output), y, can take only discrete values for a given set of features(or inputs), X.

● **Dependent variable:** The target variable in a logistic regression model, which we are trying to predict.

● **Independent variables:** The input characteristics or predictor factors applied to the dependent variable's predictions.

● **Logistic function:** The formula used to represent how the independent and dependent variables relate to one another. The logistic function transforms the input variables into a probability value between 0 and 1, which represents the likelihood of the dependent variable being 1 or 0.

● **Odds: T**he proportion of an event's chances of happening to its chances of not happening. The chances are used in logistic regression to model the connection between the independent and dependent variables.

● **Log-odds:** The logistic regression model's calculation is made simpler by using the logarithm of the odds.

● **Coefficient:** The logistic regression model's estimated parameters, which show how the independent and dependent variables relate to one another.

● **Intercept:** A constant term in the logistic regression model, which represents the log-odds when all independent variables are equal to zero.

● **Maximum likelihood estimation:** The method used to estimate the coefficients of the logistic regression model, which maximizes the likelihood of observing the data given the model. ● **Confusion matrix:** A table that lists the number of true positive, true negative, false positive, and

false negative predictions made by a logistic regression model is used to assess the model’s performance.

**Logistic regression on Social_Network_Ads.csv**

We will be taking data from social network ads which tell us whether a person will purchase the ad or not based on the features such as age and salary.

First, we will import all the libraries:

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd
```

Now we will import the dataset and select only age and salary as the features

```
dataset = pd.read_csv('Social_Network_Ads.csv')


dataset.head()
```

Now we will perform splitting for training and testing. We will take 75% of the data for training, and test on the remaining data

```
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 = 0)
```

Next, we scale the features to avoid variation and let the features follow a normal distribution

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)
```

The preprocessing part is over. It is time to fit the model

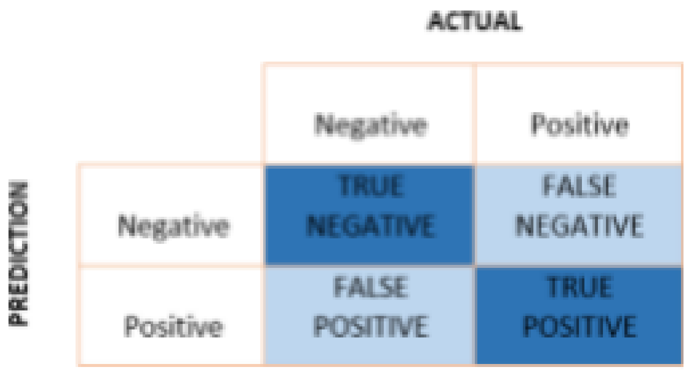
```
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(random_state = 0)

classifier.fit(X_train, y_train)
```

We fitted the model on training data. We will predict the labels of test

data. `y_pred = classifier.predict(X_test)`

**What is a confusion matrix?**

It is a matrix of size 2×2 for binary classification with actual values on one axis and predicted on another.



**Precision**

Out of all the positive predicted, what percentage is truly positive.

$$Precision = \frac{TP}{TP + FP}$$

The precision value lies between 0 and 1.

**Recall**

Out of the total positive, what percentage are predicted positive. It is the same as TPR (true positive rate).

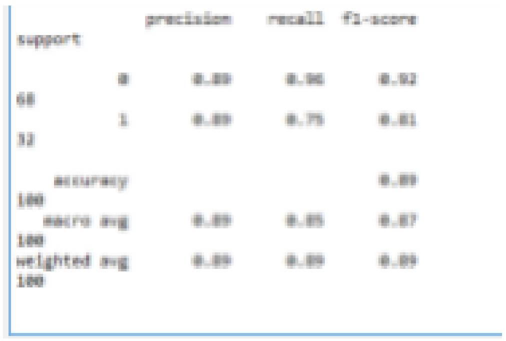
$$Recall = \frac{TP}{TP + FN}$$

The prediction is over. Now we will evaluate the performance of our

model. From sklearn.metrics import confusion_matrix,classification_report

cm = confusion_matrix(y_test, y_pred)

cl_report=classification_report(y_test,y_pred)



```
              precision    recall  f1-score
support

           0       0.89      0.96      0.92
68
           1       0.89      0.75      0.81
32

    accuracy                           0.89
100
   macro avg       0.89      0.85      0.87
100
weighted avg       0.89      0.89      0.89
100
```

**Conclusion:** Thus, we implemented the logistic regression on social network adv dataset and identified the accuracy, precision and recall.

# Assignment 6

## Title: Data Analytics III

## Aim

1. Implement Simple Naïve Bayes classification algorithm using Python/R on iris.csv dataset. 2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.

## Objective:

**Students are able to learn:**

1. How to calculate the probabilities required by the Naive Bayes algorithm. 2. How to implement the Naive Bayes algorithm from scratch.
**3.** How to apply Naive Bayes to a real-world predictive modeling problem.

**Software/Hardware Requirements:** Python/R, on iris.csv dataset, Linux Operating System.

## Theory:

**Contents for Theory:**

**1. Concepts used in Naïve Bayes classifier**

**2. Naive Bayes Example**

**3. Confusion Matrix Evaluation Metrics**

**-----------------------------------------------------------------------------------------------------**

**------ 1. Concepts used in Naïve Bayes classifier**

● Naïve Bayes Classifier can be used for Classification of categorical data. ○ Let there be a 'j' number of classes. C={1,2,….j}

○ Let, input observation is specified by 'P' features. Therefore input observation x is given , x = {F1,F2,…..Fp}

○ The Naïve Bayes classifier depends on Bayes' rule from probability theory. ● Prior probabilities: Probabilities which are calculated for some event based on no other information are called Prior probabilities.

For example, P(A), P(B), P(C) are prior probabilities because while calculating P(A), occurrences of event B or C are not concerned i.e. no information about occurrence of any other event is used.

**Conditional Probabilities:**

$$P\left(\frac{A}{B}\right) = \frac{P(A\cap B)}{P(B)} \quad if\ P(B) \neq 0 \qquad \ldots\ldots\ (1)$$

$$P\left(\frac{B}{A}\right) = \frac{P(B\cap A)}{P(A)} \qquad \ldots\ldots\ldots\ (2)$$

From equation (1) and (2) ,

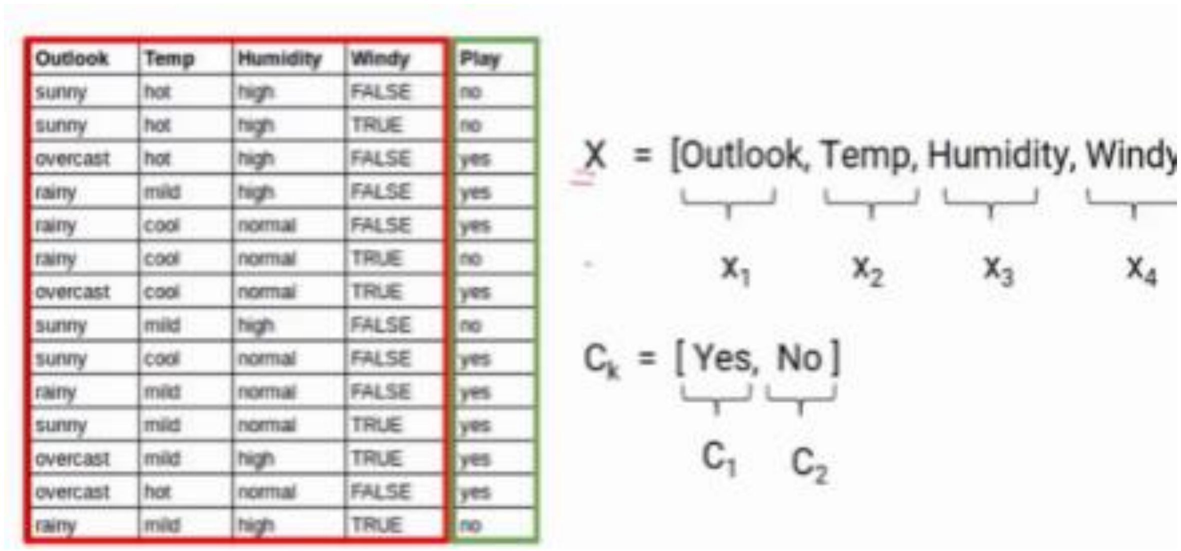
$$P(A\cap B) = P\left(\frac{A}{B}\right).P(B) = P\left(\frac{B}{A}\right).P(A)$$

$$\therefore \qquad P\left(\frac{A}{B}\right) = \frac{P\left(\frac{B}{A}\right).P(A)}{P(B)}$$

Is called the Bayes Rule.

## 2. Example of Naive Bayes

We have a dataset with some features Outlook, Temp, Humidity, and Windy, and the target here is to predict whether a person or team will play tennis or not.



| Outlook | Temp | Humidity | Windy | Play |
|---|---|---|---|---|
| sunny | hot | high | FALSE | no |
| sunny | hot | high | TRUE | no |
| overcast | hot | high | FALSE | yes |
| rainy | mild | high | FALSE | yes |
| rainy | cool | normal | FALSE | yes |
| rainy | cool | normal | TRUE | no |
| overcast | cool | normal | TRUE | yes |
| sunny | mild | high | FALSE | no |
| sunny | cool | normal | FALSE | yes |
| rainy | mild | normal | FALSE | yes |
| sunny | mild | normal | TRUE | yes |
| overcast | mild | high | TRUE | yes |
| overcast | hot | normal | FALSE | yes |
| rainy | mild | high | TRUE | no |

$$X = [\underbrace{Outlook}_{x_1}, \underbrace{Temp}_{x_2}, \underbrace{Humidity}_{x_3}, \underbrace{Windy}_{x_4}$$

$$C_k = [\underbrace{Yes}_{C_1}, \underbrace{No}_{C_2}]$$

### Conditional Probability

Here, we are predicting the probability of class1 and class2 based on the given condition. If I try to write the same formula in terms of classes and features, we will get the following equation

$$P(C_k \mid X) = \frac{P(X \mid C_k) * P(C_k)}{P(X)}$$

Now we have two classes and four features, so if we write this formula for class C1, it will be something like this.

$$P(C_1 \mid x_1 \cap x_2 \cap x_3 \cap x_4) = \frac{P(x_1 \cap x_2 \cap x_3 \cap x_4 \mid C_1) * P(C_1)}{P(x_1 \cap x_2 \cap x_3 \cap x_4)}$$

Here, we replaced Ck with C1 and X with the intersection of X1, X2, X3, X4. You might have a question, It’s because we are taking the situation when all these features are present at the same time.

The Naive Bayes algorithm assumes that all the features are independent of each other or in other words all the features are unrelated. With that assumption, we can further simplify the above formula and write it in this form

$$\boxed{P(C_1 \mid x_1 \cap x_2 \cap x_3 \cap x_4) = \frac{P(x_1 \mid C_1) * P(x_2 \mid C_1) * P(x_3 \mid C_1) * P(x_4 \mid C_1) * P(C_1)}{P(x_1) * P(x_2) * P(x_3) * P(x_4)}}$$

This is the final equation of the Naive Bayes and we have to calculate the probability of both C1 and C2.For this particular example.

| Outlook | Temp | Humidity | Windy | Play |
|---|---|---|---|---|
| Rainy | Cool | High | True | ? |

$$P(Yes \mid X) = P(Rainy \mid Yes) \times P(Cool \mid Yes) \times P(High \mid Yes) \times P(True \mid Yes) \times P(Yes)$$

$$P(Yes \mid X) = 2/9 \times 3/9 \times 3/9 \times 3/9 \times 9/14 = \boxed{0.00529} \longrightarrow 0.2 = \frac{0.00529}{0.02057 + 0.00529}$$

$$P(No \mid X) = P(Rainy \mid No) \times P(Cool \mid No) \times P(High \mid No) \times P(True \mid No) \times P(No)$$

$$P(No \mid X) = 3/5 \times 1/5 \times 4/5 \times 3/5 \times 5/14 = \boxed{0.02057} \longrightarrow 0.8 = \frac{0.02057}{0.02057 + 0.00529}$$

P (N0 | Today) > P (Yes | Today) So, the prediction that golf would be played is ‘No’.

For this implementation we will use the Iris Flower Species Dataset.

The Iris Flower Dataset involves predicting the flower species given measurements of iris flowers.

It is a multiclass classification problem. The number of observations for each class is balanced. There are 150 observations with 4 input variables and 1 output variable. The variable names are as follows:

· Sepal length in cm.
· Sepal width in cm.
· Petal length in cm.
· Petal width in cm.
· Class

**Algorithm (Iris Dataset)**:

**Step 1: Import libraries and create alias for Pandas, Numpy and Matplotlib Step 2: Import the Iris dataset by calling URL.**

**Step 3: Initialize the data frame**

**Step 4: Perform Data Preprocessing**

● Convert Categorical to Numerical Values if applicable ● Check for Null Value
● Divide the dataset into Independent(X) and Dependent(Y)variables.
● Split the dataset into training and testing datasets ● Scale the Features if necessary.

we will define feature (X) and target (y) variables, and split the dataset into training and testing sets.

```
X = df.iloc[:,:4].values

y = df['species'].values
```

**Step 5: Use Naive Bayes algorithm( Train the Machine ) to Create Model** `# import the class`

```
 from sklearn.naive_bayes import GaussianNB
 gaussian = GaussianNB()
gaussian.fit(X_train, y_train)
```

**Step 6: Predict the y_pred for all values of train_x and test_x**

```
Y_pred = gaussian.predict(X_test)
```

**Step 7:Evaluate the performance of Model for train_y and test_y**

```
 accuracy = accuracy_score(y_test,Y_pred)


precision =precision_score(y_test,
Y_pred,average='micro') recall =
recall_score(y_test, Y_pred,average='micro')
```

**Step 8: Calculate the required evaluation parameters**

```
from sklearn.metrics import
precision_score,confusion_matrix,accuracy_score,recall_
s core cm = confusion_matrix(y_test, Y_pred)
```

## Confusion Matrix Definition

A confusion matrix is used to judge the performance of a classifier on the test dataset for which we already know the actual values. Confusion matrix is also termed as Error matrix. It consists of a count of correct and incorrect values broken down by each class. It not only tells us the error made by classifier but also tells us what type of error the classifier made. So, we can say that a confusion matrix is a performance measurement technique of a classifier model where output can be two classes or more. It is a table with four different groups of true and predicted values.

### Terminologies in Confusion Matrix

The confusion matrix shows us how our classifier gets confused while predicting. In a confusion matrix we have four important terms which are:

1. **True Positive (TP)-** Both actual and predicted values are Positive.
2. **True Negative (TN)-** Both actual and predicted values are Negative.
3. **False Positive (FP)-** The actual value is negative but we predicted it as positive.
4. **False Negative (FN)-** The actual value is positive but we predicted it as negative.

## Performance Metrics

Confusion matrix not only used for finding the errors in prediction but is also useful to find some important performance metrics like Accuracy, Recall, Precision, F-measure. We will discuss these terms one by one.

### Accuracy

As the name suggests, the value of this metric suggests the accuracy of our classifier in predicting results.

It is defined as:

Accuracy = (TP + TN) / (TP + TN + FP + FN)
A 99% accuracy can be good, average, poor or dreadful depending upon the problem.

### Precision

Precision is the measure of all actual positives out of all predicted positive values. It

is defined as:

Precision = TP / (TP + FP)

### Recall

Recall is the measure of positive values that are predicted correctly out of all actual positive values. It is defined as:

Recall = TP / (TP + FN)
High Value of Recall specifies that the class is correctly known (because of a small number of False Negative).

## F-measure

It is hard to compare classification models which have low precision and high recall or vice versa. So, for comparing the two classifier models we use F-measure. F-score helps to find the metrics of Recall and Precision in the same interval. Harmonic Mean is used instead of Arithmetic Mean.

F-measure is defined as:

F-measure = 2 * Recall * Precision / (Recall + Precision)
The F-Measure is always closer to the Precision or Recall, whichever has a smaller value

**Conclusion:** In this assignment, we covered how Naïve Bayes theorem used to solve classification problem for iris flower dataset and what is confusion matrix, its need, and how to derive it in Python and R.

# DS&BDL
# Assignment 7

## Title: Text Analytics

## Aim

1. Extract Sample document and apply following document pre-processing methods: Tokenization, POS Tagging, stop words removal, Stemming and Lemmatization. 2. Create representation of document by calculating Term Frequency and Inverse Document Frequency..

## Objective:

**Students are able to learn:**

1. Text Analytics and NLP.
2. Document Pre-Processing Methods.
3. Text Classification using TF-IDF

**Software/Hardware Requirements:** 64-bit Open source Linux, Python,

NLTK **Theory:**

Text Analytics has lots of applications in today's online world. By analyzing tweets on Twitter, we can find trending news and peoples reaction on a particular event. Amazon can understand user feedback or review on the specific product. BookMyShow can discover people's opinion about the movie. Youtube can also analyze and understand peoples viewpoints on a video.

In this assignment we are going to take sample document and will apply following pre-processing methods.

## 1. Text analysis Operations using nltk.

NLTK is a powerful Python package that provides a set of diverse natural languages algorithms. It is free, opensource, easy to use, large community, and well documented. NLTK consists of the most common algorithms such as tokenizing, part-of-speech tagging, stemming, sentiment analysis, topic segmentation, and named entity recognition. NLTK helps the computer to analysis, preprocess, and understand the written text.

```
!pip install nltk
Requirement already satisfied: nltk in /home/northout/anaconda2/lib/python2.7/site-packages
Requirement already satisfied: six in /home/northout/anaconda2/lib/python2.7/site-packages
(from nltk)
 [33mYou are using pip version 9.0.1, however version 10.0.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command. [0m

#Loading NLTK
import nltk
```

## 2. Tokenization

Tokenization is the first step in text analytics. The process of breaking down a text paragraph into smaller chunks such as words or sentence is called Tokenization. Token is a single entity that is building blocks for sentence or paragraph.

### Sentence Tokenization

Sentence tokenizer breaks text paragraph into sentences.

```
from nltk.tokenize import sent_tokenize
text=""" """
tokenized_text=sent_tokenize(text)
print(tokenized_text)
```

### Word Tokenization

Word tokenizer breaks text paragraph into words.

```
from nltk.tokenize import word_tokenize
tokenized_word=word_tokenize(text)
print(tokenized_word)
```

## 3. Stop words

Stopwords considered as noise in the text. Text may contain stop words such as is, am, are, this, a, an, the, etc.
In NLTK for removing stopwords, you need to create a list of stopwords and filter out your list of tokens from these words.

```
from nltk.corpus import stopwords
stop_words=set(stopwords.words("english"))
print(stop_words)
```

**Removing Stop Words**

```
filtered_sent=[]
for w in tokenized_sent:
 if w not in stop_words:
 filtered_sent.append(w)
print("Tokenized Sentence:",tokenized_sent)
print("Filterd Sentence:",filtered_sent)
```

## 4. Stemming and Lemmatization.

### Stemming

Stemming is a process of linguistic normalization, which reduces words to their word root word or chops off the derivational affixes. For example, connection, connected, connecting word reduce to a common word "connect".

```
# Stemming

from nltk.stem import PorterStemmer

from nltk.tokenize import sent_tokenize, word_tokenize
ps = PorterStemmer()

stemmed_words=[]

for w in filtered_sent:

 stemmed_words.append(ps.stem(w))
```

```
print("Filtered Sentence:",filtered_sent)

print("Stemmed Sentence:",stemmed_words)
```

**Lemmatization**

Lemmatization reduces words to their base word, which is linguistically correct lemmas. It transforms root word with the use of vocabulary and morphological analysis. Lemmatization is usually more sophisticated than stemming. Stemmer works on an individual word without knowledge of the context. For example, The word "better" has "good" as its lemma. This thing will miss by stemming because it requires a dictionary look-up.

```
#Lexicon Normalization

#performing stemming and Lemmatization

from nltk.stem.wordnet import WordNetLemmatizer

lem = WordNetLemmatizer()

from nltk.stem.porter import PorterStemmer

stem = PorterStemmer()

word = "flying"

print("Lemmatized Word:",lem.lemmatize(word,"v"))

print("Stemmed Word:",stem.stem(word))
```

## 5. POS Tagging

The primary target of Part-of-Speech(POS) tagging is to identify the grammatical group of a given word. Whether it is a NOUN, PRONOUN, ADJECTIVE, VERB, ADVERBS, etc. based on the context. POS Tagging looks for relationships within the sentence and assigns a corresponding tag to the word.

```
sent = " "
tokens=nltk.word_tokenize(sent)
print(tokens)
['Albert', 'Einstein', 'was', 'born', 'in', 'Ulm', ',', 'Germany', 'in', '1879', '.']
nltk.pos_tag(tokens)
```

## 6. Text Classification

Text Classification Model using TF-IDF. First, import the MultinomialNB module and create a Multinomial Naive Bayes classifier object using MultinomialNB() function. Then, fit your model on a train set using fit() and perform prediction on the test set using predict().

```
from sklearn.naive_bayes import MultinomialNB


#Import scikit-learn metrics module for accuracy calculation


from sklearn import metrics
```

```
# Model Generation Using Multinomial Naive Bayes

clf = MultinomialNB().fit(X_train, y_train)

predicted= clf.predict(X_test)

print("MultinomialNB Accuracy:",metrics.accuracy_score(y_test, predicted))
```

**Feature Generation using TF-IDF**

In Term Frequency(TF), you just count the number of words occurred in each document. The main issue with this Term Frequency is that it will give more weight to longer documents. Term frequency is basically the output of the BoW model.

IDF(Inverse Document Frequency) measures the amount of information a given word provides across the document. IDF is the logarithmically scaled inverse ratio of the number of documents that contain the word and the total number of documents.

$$\text{idf}(W) = \log \frac{\#(\text{documents})}{\#(\text{documents containing word W})}$$

TF-IDF(Term Frequency-Inverse Document Frequency) normalizes the document term matrix. It is the product of TF and IDF. Word with high tf-idf in a document, it is most of the times occurred in given documents and must be absent in the other documents. So the words must be a signature word.

```
from sklearn.feature_extraction.text import TfidfVectorizer
tf=TfidfVectorizer()
text_tf= tf.fit_transform(data['Phrase'])
```

**Conclusion:** In this assignment, we have learned what Text Analytics is, NLP and text mining, basics of text analytics operations using NLTK such as Tokenization, Normalization, Stemming, Lemmatization and POS tagging. What is Text Classification Model using TF-IDF.

## Group A Assignment 8

**Title:**

1. Use the inbuilt dataset 'titanic'. The dataset contains 891 rows and contains information about the passengers who boarded the unfortunate Titanic ship. Use the Seaborn library to see if we can find any patterns in the data.

2. Write a code to check how the price of the ticket (column name: 'fare') for each passenger is distributed by plotting a histogram

**Objective:**

Use the Seaborn library to see if we can find any patterns in the data.

**Theory:**

**Contents for Theory:**

1. **Seaborn Library Basics**
2. **Know your Data**
3. **Finding patterns of data.**
4. **Checking how the price of the ticket (column name: 'fare') for each passenger is distributed by plotting a histogram.**

**-------------------------------------------------------------------------------------------------------------**

**- Theory:**

Data Visualisation plays a very important role in Data mining. Various data scientists spent their time exploring data through visualization. To accelerate this process we need to have a well-documentation of all the plots.

Even plenty of resources can’t be transformed into valuable goods without planning and architecture

### 1. Seaborn Library Basics

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. For the installation of Seaborn, you may run any of the following in your command line.

```
pip install seaborn
conda install seaborn
```

To import seaborn you can run the following command.

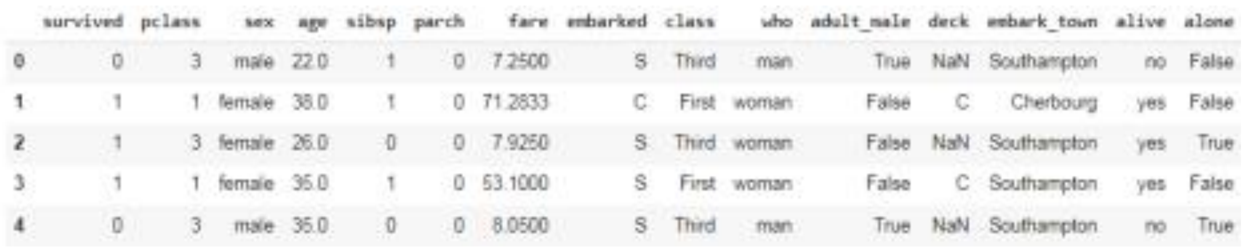
```
import seaborn as sns
```

### 2. Know your data

The dataset that we are going to use to draw our plots will be the Titanic dataset, which is downloaded by default with the Seaborn library. All you have to do is use the load_dataset function and pass it the name of the dataset.

Let's see what the Titanic dataset looks like. Execute the following script:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns



dataset = sns.load_dataset('titanic')

dataset.head()
```



```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class    who  adult_male deck  embark_town alive  alone
0         0       3    male  22.0      1      0   7.2500        S  Third    man        True  NaN  Southampton    no  False
1         1       1  female  38.0      1      0  71.2833        C  First  woman       False    C    Cherbourg   yes  False
2         1       3  female  26.0      0      0   7.9250        S  Third  woman       False  NaN  Southampton   yes   True
3         1       1  female  35.0      1      0  53.1000        S  First  woman       False    C  Southampton   yes  False
4         0       3    male  35.0      0      0   8.0500        S  Third    man        True  NaN  Southampton    no   True
```

The dataset contains 891 rows and 15 columns and contains information about the passengers who boarded the unfortunate Titanic ship. The original task is to predict whether or not the passenger survived depending upon different features such as their age, ticket, cabin they boarded, the class of the ticket, etc. We will use the Seaborn library to see if we can find any patterns in the data.

**3. Finding patterns of data.**

**Patterns of data can be find out with the help of different types of plots** Types of plots are:

A. **Distribution Plots**

a. Dist-Plot

b. Joint Plot

d. Rug Plot

B. **Categorical Plots**

a. Bar Plot

b. Count Plot

c. Box Plot

d. Violin Plot

C. **Advanced Plots**

a. Strip Plot

b. Swarm Plot

D. **Matrix Plots**

a. Heat Map

b. Cluster Map

**A. Distribution Plots:**

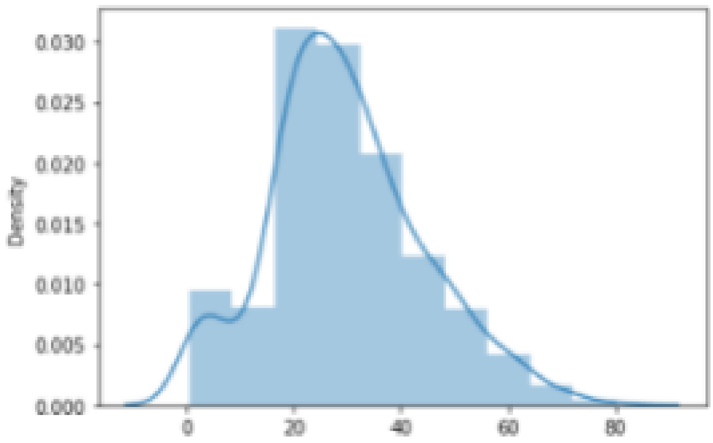
These plots help us to visualise the distribution of data. We can use these plots to understand the mean, median, range, variance, deviation, etc of the data.

a. Distplot

● Dist plot gives us the histogram of the selected continuous variable. ● It is an example of a univariate analysis.

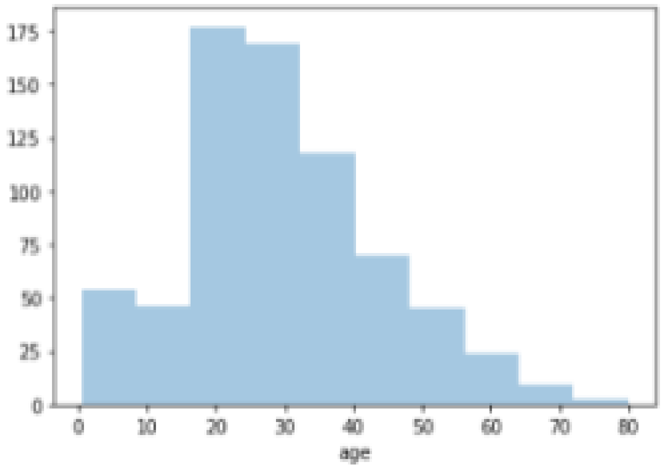
● We can change the number of bins i.e. number of vertical bars in a histogram

```
import seaborn as sns

sns.distplot(x = dataset['age'], bins = 10)
```



The line that you see represents the kernel density estimation. You can remove this line by passing False as the parameter for the kde attribute as shown below

```
sns.distplot(dataset['age'], bins = 10,kde=False)
```



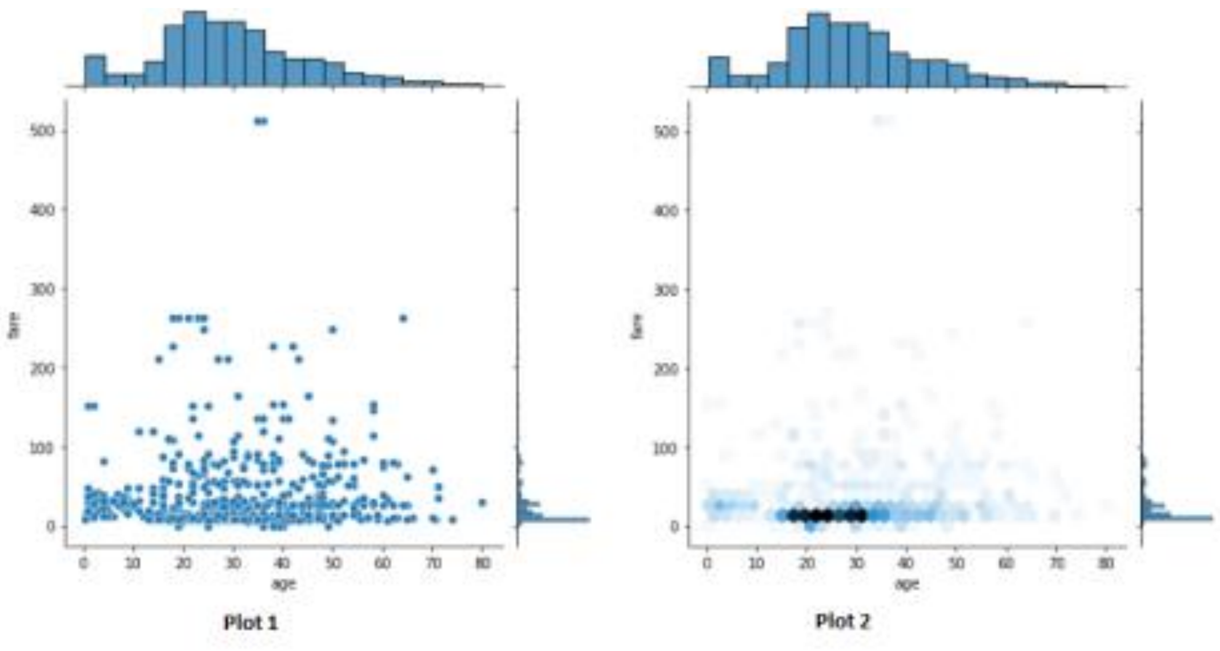
Here the x-axis is the age and the y-axis displays frequency. For example, for bins = 10,

there are around 50 people having age 0 to 10

**b. Joint Plot**

- It is the combination of the distplot of two variables.
- It is an example of bivariate analysis.
- We additionally obtain a scatter plot between the variables to reflect their linear relationship. We can customise the scatter plot into a hexagonal plot, where, the more the colour intensity, the more will be the number of observations.

```
import seaborn as sns

# For Plot 1

sns.jointplot(x = dataset['age'], y = dataset['fare'], kind =
'scatter')

# For Plot 2

sns.jointplot(x = dataset['age'], y = dataset['fare'], kind = 'hex')
```

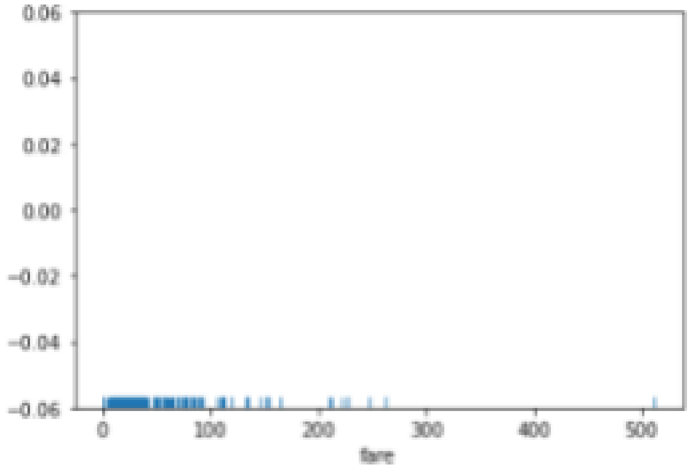


- From the output, you can see that a joint plot has three parts. A distribution plot at the top for the column on the x-axis, a distribution plot on the right for the column on the y-axis and a scatter plot in between that shows the mutual distribution of data for both the columns. You can see that there is no correlation observed between prices and the fares.
- You can change the type of the joint plot by passing a value for the kind parameter. For instance, if instead of a scatter plot, you want to display the distribution of data in the form of a hexagonal plot, you can pass the value hex for the kind parameter.
- In the hexagonal plot, the hexagon with the most number of points gets darker colour. So if you look at the above plot, you can see that most of the passengers are between the ages of 20 and 30 and most of them paid between 10-50 for the tickets.

**a. c. The Rug Plot**

b. The rugplot() is used to draw small bars along the x-axis for each point in the dataset. To plot a rug plot, you need to pass the name of the column. Let's plot a rug plot for fare.

```
sns.rugplot(dataset['fare'])
```



From the output, you can see that most of the instances for the fares have values between 0 and 100.

These are some of the most commonly used distribution plots offered by the Python's Seaborn Library. Let's see some of the categorical plots in the Seaborn library.

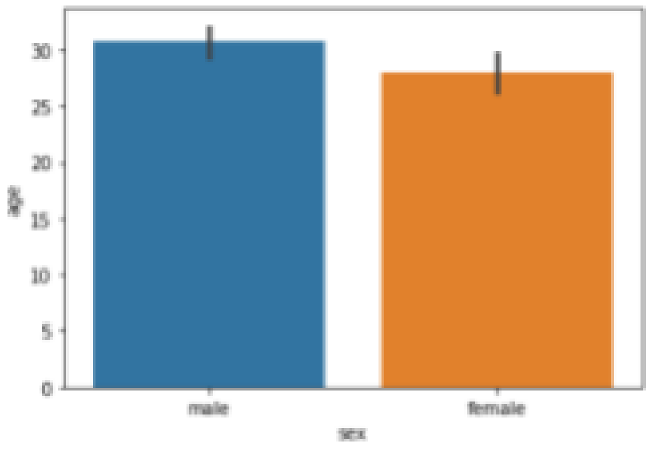
## 2. Categorical Plots

Categorical plots, as the name suggests, are normally used to plot categorical data. The categorical plots plot the values in the categorical column against another categorical column or a numeric column. Let's see some of the most commonly used categorical data.

### b. The Bar Plot

The barplot() is used to display the mean value for each value in a categorical column, against a numeric column. The first parameter is the categorical column, the second parameter is the numeric column while the third parameter is the dataset. For instance, if you want to know the mean value of the age of the male and female passengers, you can use the bar plot as follows.

```
    , y='age', data=dataset)
```

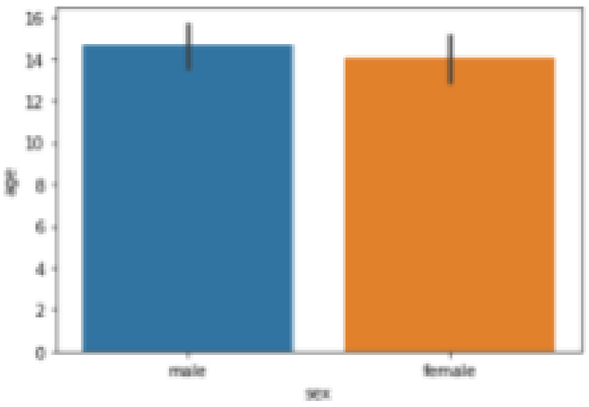


From the output, you can clearly see that the average age of male passengers is just less than 40 while the average age of female passengers is around 33.

In addition to finding the average, the bar plot can also be used to calculate other aggregate values for each category. To do so, you need to pass the aggregate function to the estimator. For instance, you can calculate the standard deviation for the age of each gender as follows:

```
import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

sns.barplot(x='sex', y='age', data=dataset, estimator=np.std)
```

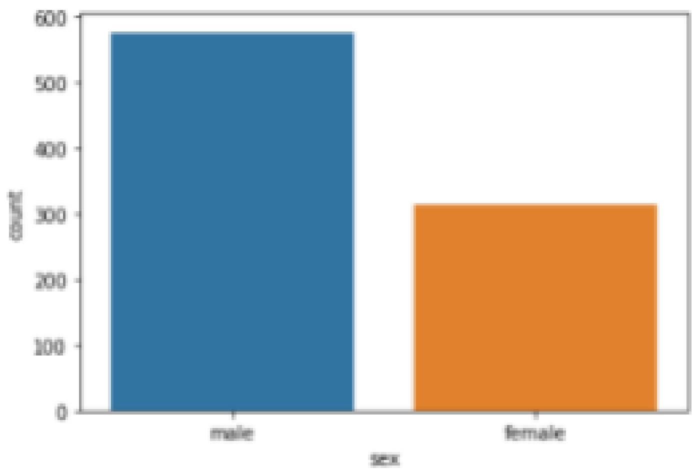
Notice, in the above script we use the std aggregate function from the numpy library to calculate the standard deviation for the ages of male and female passengers. The output looks like this:



**c. The Count Plot**

The count plot is similar to the bar plot, however it displays the count of the categories in a specific column. For instance, if we want to count the number of males and women passenger we can do so using count plot as follows:

```
sns.countplot(x='sex', data=dataset)
```

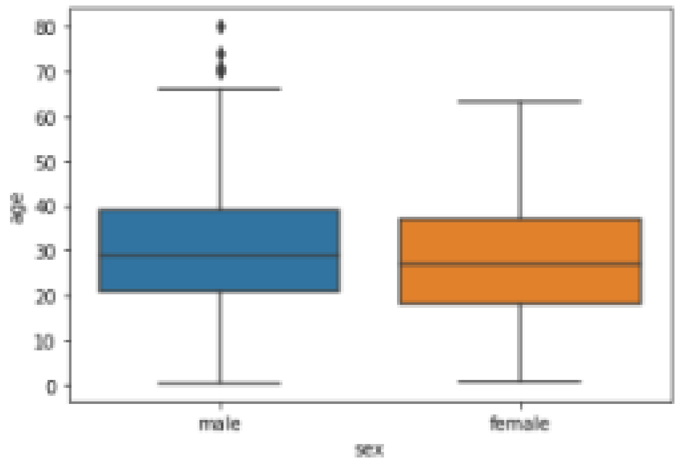


**d. The Box Plot**

The box plot is used to display the distribution of the categorical data in the form of quartiles. The centre of the box shows the median value. The value from the lower whisker to the bottom of the box shows the first quartile. From the bottom of the box to the middle of the box lies the second quartile. From the middle of the box to the top of the box lies the third quartile and finally from the top of the box to the top whisker lies the last quartile.

Now let's plot a box plot that displays the distribution for the age with respect to each gender. You need to pass the categorical column as the first parameter (which is sex in our case) and the numeric column (age in our case) as the second parameter. Finally, the dataset is passed as the third parameter, take a look at the following script:

```
sns.boxplot(x='sex', y='age', data=dataset)
```

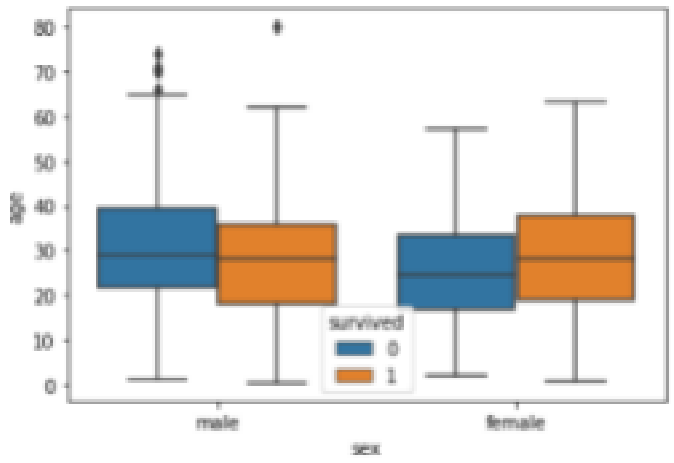


Let's try to understand the box plot for females. The first quartile starts at around 1 and ends at 20 which means that 25% of the passengers are aged between 1 and 20. The second quartile starts at around 20 and ends at around 28 which means that 25% of the passengers are aged between20 and 28. Similarly, the third quartile starts and ends between 28 and 38, hence 25% passengers are aged within this range and finally the fourth or last quartile starts at 38 and ends around 64.

If there are any outliers or the passengers that do not belong to any of the quartiles, they are called outliers and are represented by dots on the box plot.

You can make your box plots more fancy by adding another layer of distribution. For instance, if you want to see the box plots of forage of passengers of both genders, along with the information about whether or not they survived, you can pass the survived as value to the hue parameter as shown below:

```
sns.boxplot(x='sex', y='age', data=dataset, hue="survived")
```



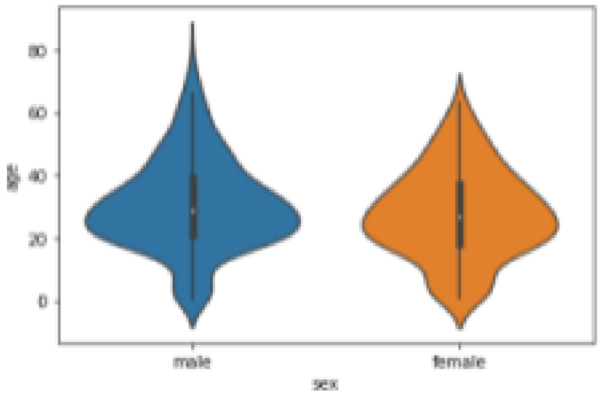
Now in addition to the information about the age of each gender, you can also see the distribution of the passengers who survived. For instance, you can see that among the male passengers, on average more younger people survived as compared to the older ones. Similarly, you can see that the variation among the age of female passengers who did not survive is much greater than the age of the surviving female passengers.

**e. The Violin Plot**

The violin plot is similar to the box plot, however, the violin plot allows us to display all the components that actually correspond to the data point. The violinplot() function is used to plot the violin plot. Like the box plot, the first parameter is the categorical column, the second parameter is the numeric column while the third parameter is the dataset.

Let's plot a violin plot that displays the distribution for the age with respect to each

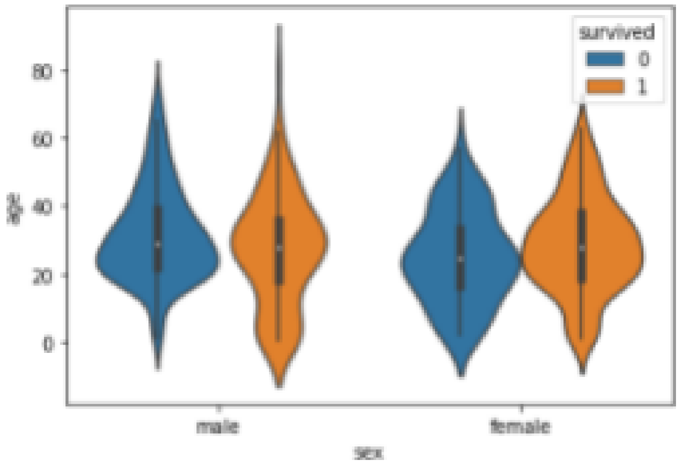
gender. `sns.violinplot(x='sex', y='age', data=dataset)`



You can see from the figure above that violin plots provide much more information about the data as compared to the box plot. Instead of plotting the quartile, the violin plot allows us to see all the components that actually correspond to the data. The area where the violin plot is thicker has a higher number of instances for the age. For instance, from the violin plot for males, it is clearly evident that the number of passengers with age between 20 and 40 is higher than all the rest of the age brackets.

Like box plots, you can also add another categorical variable to the violin plot using the hue parameter as shown below:

```
sns.violinplot(x='sex', y='age', data=dataset, hue='survived')
```



Now you can see a lot of information on the violin plot. For instance, if you look at the bottom of the violin plot for the males who survived (left-orange), you can see that it is thicker than the bottom of the violin plot for the males who didn't survive (left-blue). This means that the number of young male passengers who survived is greater than the number of young male passengers who did not survive

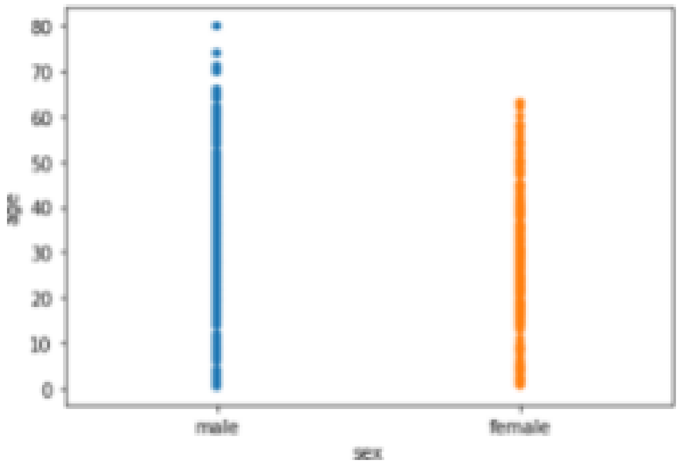
**Advanced Plots:**

**a. The Strip Plot**

The strip plot draws a scatter plot where one of the variables is categorical. We have seen scatter plots in the joint plot and the pair plot sections where we had two numeric variables. The strip plot is different in a way that one of the variables is categorical in this case, and for each category in the categorical variable, you will see a scatter plot with respect to the numeric column.

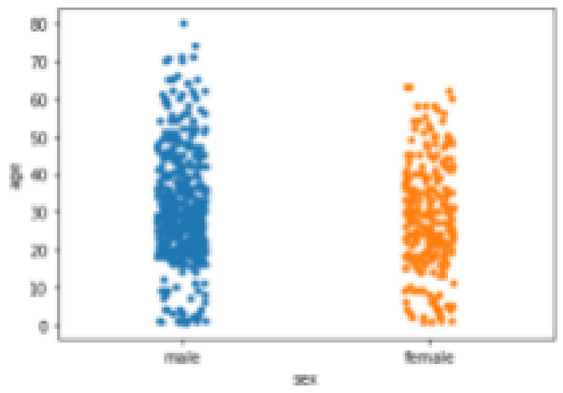
The stripplot() function is used to plot the violin plot. Like the box plot, the first parameter is the categorical column, the second parameter is the numeric column while the third parameter is the dataset. Look at the following script:

```
sns.stripplot(x='sex', y='age', data=dataset, jitter=False)
```



You can see the scattered plots of age for both males and females. The data points look like strips. It is difficult to comprehend the distribution of data in this form. To better comprehend the data, pass True for the jitter parameter which adds some random noise to the data. Look at the following script:

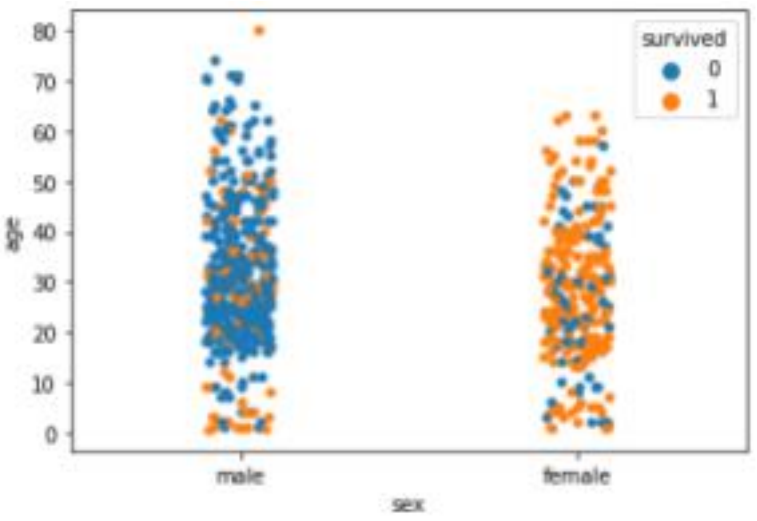
```
sns.stripplot(x='sex', y='age', data=dataset, jitter=True)
```



Now you have a better view for the distribution of age across the genders.

Like violin and box plots, you can add an additional categorical column to strip plot using hue parameter as shown below:

```
sns.stripplot(x='sex', y='age', data=dataset, jitter=True,
```



```
hue='survived')
```

### b. The Swarm Plot

The swarm plot is a combination of the strip and the violin plots. In the swarm plots, the points are adjusted in such a way that they don't overlap. Let's plot a swarm plot for the distribution of age against gender. The swarmplot() function is used to plot the violin plot. Like the box plot, the first parameter is the categorical column, the second parameter is the numeric column while the third parameter is the dataset. Look at the following script:

```
sns.swarmplot(x='sex', y='age', data=dataset)
```

You can clearly see that the above plot contains scattered data points like the strip plot and the data points are not overlapping. Rather they are arranged to give a view similar to that of a violin plot.

Let's add another categorical column to the swarm plot using the hue

parameter. `sns.swarmplot(x='sex', y='age', data=dataset,`

```
hue='survived')
```

From the output, it is evident that the ratio of surviving males is less than the ratio of surviving females. Since for the male plot, there are more blue points and less orange points. On the other hand, for females, there are more orange points (surviving) than the blue points (not surviving). Another observation is that amongst males of age less than 10, more passengers survived as compared to those who didn't.

## 1. Matrix Plots

Matrix plots are the type of plots that show data in the form of rows and columns. Heat maps are the prime examples of matrix plots.

### a. Heat Maps

Heat maps are normally used to plot correlation between numeric columns in the form of a matrix. It is important to mention here that to draw matrix plots, you need to have meaningful information on rows as well as columns. Let's plot the first five rows of the Titanic dataset to see if both the rows and column headers have meaningful information. Execute the following script:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

dataset = sns.load_dataset('titanic')

dataset.head()
```

From the output, you can see that the column headers contain useful information such as passengers surviving, their age, fare etc. However the row headers only contain indexes 0, 1, 2, etc. To plot matrix plots, we need useful information on both columns and row headers. One way to do this is to call the corr() method on the dataset. The corr() function returns the correlation between all the numeric columns of the dataset. Execute the following script:

```
dataset.corr()
```

In the output, you will see that both the columns and the rows have meaningful header information, as shown below:
Now to create a heat map with these correlation values, you need to call the heatmap() function and pass it your correlation dataframe. Look at the following script:

```
corr = dataset.corr()

sns.heatmap(corr)
```

From the output, it can be seen that what heatmap essentially does is that it plots a box for every combination of rows and column value. The colour of the box depends upon the gradient. For instance, in the above image if there is a high correlation between two features, the corresponding cell or the box is white, on the other hand if there is no correlation, the corresponding cell remains black.

The correlation values can also be plotted on the heatmap by passing True for the annot parameter. Execute the following script to see this in action:

```
corr = dataset.corr()

sns.heatmap(corr, annot=True)
```

You can also change the colour of the heatmap by passing an argument for the cmap parameter. For now, just look at the following script:

```
corr = dataset.corr()

sns.heatmap(corr)
```

**b. Cluster Map:**

In addition to the heat map, another commonly used matrix plot is the cluster map. The cluster map basically uses Hierarchical Clustering to cluster the rows and columns of the matrix.

Let's plot a cluster map for the number of passengers who travelled in a specific month of a specific year. Execute the following script:

**4. Checking how the price of the ticket (column name: 'fare') for each passenger is distributed by plotting a histogram.**

```
import seaborn as sns


dataset = sns.load_dataset('titanic')
sns.histplot(dataset['fare'], kde=False, bins=10)
```

From the histogram, it is seen that for around 730 passengers the price of the ticket is 50.

For 100 passengers the price of the ticket is 100 and so on.

## Conclusion

Seaborn is an advanced data visualization library built on top of Matplotlib library. In this assignment, we looked at how we can draw distributional and categorical plots using the Seaborn library. We have seen how to plot matrix plots in Seaborn. We also saw how to change plot styles and use grid functions to manipulate subplots.

# Assignment 9

**Title: Data Visualization**

**Aim:** 1. Use the inbuilt dataset 'titanic' as used in the above problem. Plot a box plot for distribution of age with respect to each gender along with the information about whether they survived or not. (Column names : 'sex' and 'age').
2. Write observations on the inference from the above statistics.

## Objective:

**Students are able to learn:**

1. How to plot boxplot. 2. How to generate inference from Data Visualization.

**Software/Hardware Requirements:** Python/R, on iris.csv dataset, Linux Operating System.

## Theory:

There are various techniques to understand your data, And the basic need is you should have the knowledge of Numpy for mathematical operations and Pandas for data manipulation. We are using Titanic dataset. For demonstrating some of the techniques we will also use an inbuilt dataset of seaborn as tips data which explains the tips each waiter gets from different customers.

### Import libraries and loading Data

```
import numpy as np

import pandas pd

import matplotlib.pyplot as plt

import seaborn as sns

from seaborn import load_dataset

#titanic dataset

data = pd.read_csv("titanic_train.csv")

#tips dataset

tips = load_dataset("tips")
```

**Univariate Analysis**

Univariate analysis is the simplest form of analysis where we explore a single variable. Univariate analysis is performed to describe the data in a better way. we perform Univariate analysis of Numerical and categorical variables differently because plotting uses different plots.

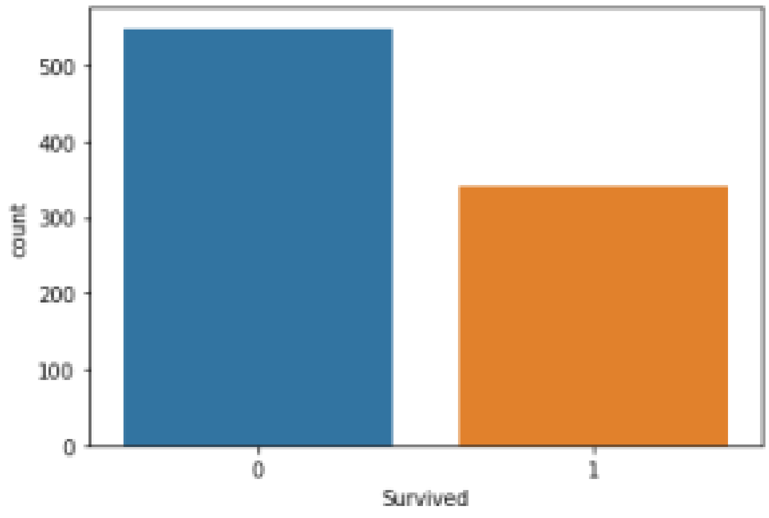
**Categorical Data:**

A variable that has text-based information is referred to as categorical variables. Now following are various plots which we can use for visualizing Categorical data.

**1) CountPlot:**

Countplot is basically a count of frequency plot in form of a bar graph. It plots the count of each category in a separate bar. When we use the pandas' value counts function on any column. It is the same visual form of the value counts function. In our data-target variable is survived and it is categorical so plot a countplot of this.

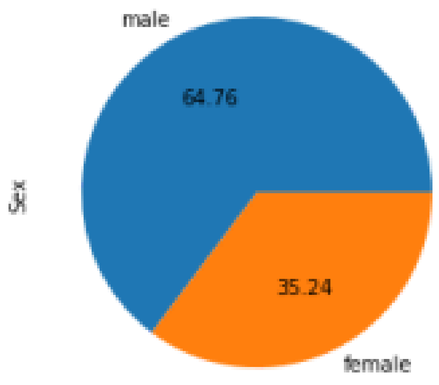
```
sns.countplot(data['Survived'])

plt.show()
```



**2) Pie Chart:**

The pie chart is also the same as the countplot, only gives us additional information about the percentage presence of each category in data means which category is getting how much weightage in data. Now we check about the Sex column, what is a percentage of Male and Female members traveling.

```
data['Sex'].value_counts().plot(kind="pie", autopct="%.2f")

plt.show()
```



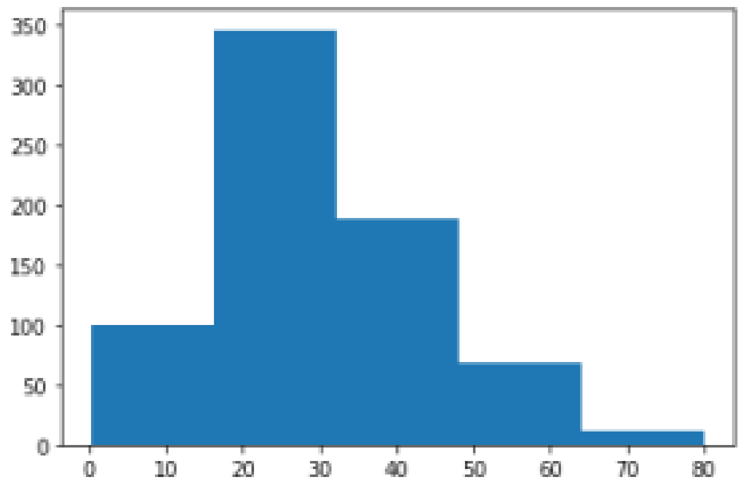
**Numerical Data:**

Analyzing Numerical data is important because understanding the distribution of variables helps to further process the data. Most of the time, we will find much inconsistency with numerical data so we have to explore numerical variables.

**1) Histogram:**

A histogram is a value distribution plot of numerical columns. It basically creates bins in various ranges in values and plots it where we can visualize how values are distributed. We can have a look where more values lie like in positive, negative, or at the center(mean). Let's have a look at the Age column.

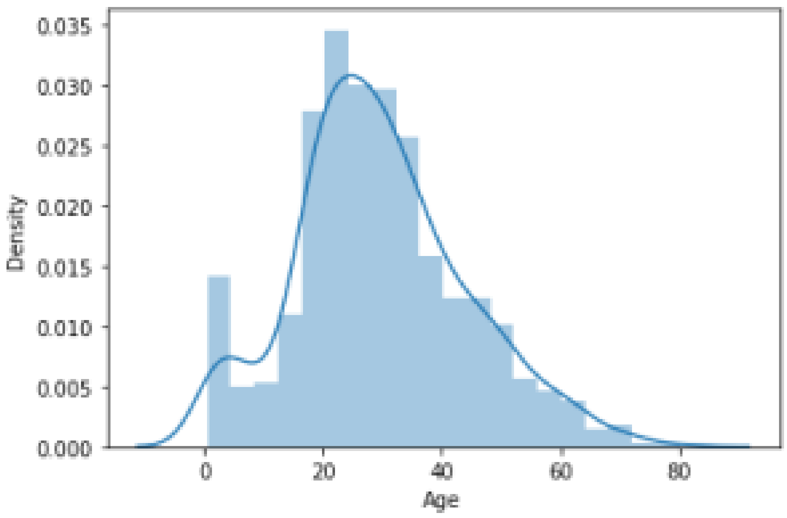
```
plt.hist(data['Age'], bins=5)

plt.show()
```



**2) Distplot:**

Distplot is also known as the second Histogram because it is a slight improvement version of the Histogram. Distplot gives us a KDE(Kernel Density Estimation) over histogram which explains PDF(Probability Density Function) which means what is the probability of each value occurring in this column.

```
sns.distplot(data['Age'])

plt.show()
```



### 3) Boxplot:

Boxplot is a very interesting plot that basically plots a 5 number summary. to get 5 number summary some terms we need to describe.

- Median – Middle value in series after sorting
- Percentile – Gives any number which is number of values present before this percentile like for example 50 under 25th percentile so it explains total of 50 values are there below 25th percentile
- Minimum and Maximum – These are not minimum and maximum values, rather they describe the lower and upper boundary of standard deviation which is calculated using Interquartile range(IQR).

```
IQR = Q3 - Q1

Lower_boundary = Q1 - 1.5 * IQR

Upper_bounday = Q3 + 1.5 * IQR
```

Here Q1 and Q3 is 1st quantile (25th percentile) and 3rd Quantile(75th percentile).

### Bivariate/ Multivariate Analysis:

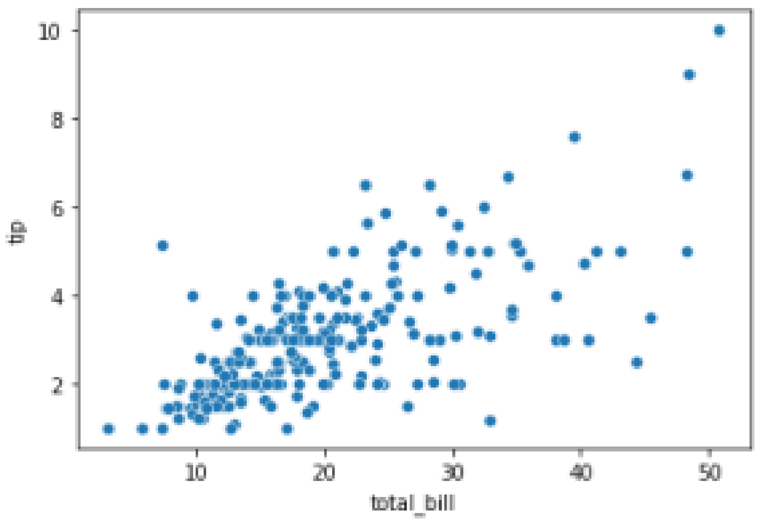
We have study about various plots to explore single categorical and numerical data. Bivariate Analysis is used when we have to explore the relationship between 2 different variables and we have to do this because, in the end, our main task is to explore the relationship between variables to build a powerful model. And when we analyze more than 2 variables together then it is known as Multivariate Analysis. we will work on different plots for Bivariate as well on Multivariate Analysis.

Explore the plots when both the variable is numerical.

### 1) Scatter Plot:

To plot the relationship between two numerical variables scatter plot is a simple plot to do. Let us see the relationship between the total bill and tip provided using a scatter plot.

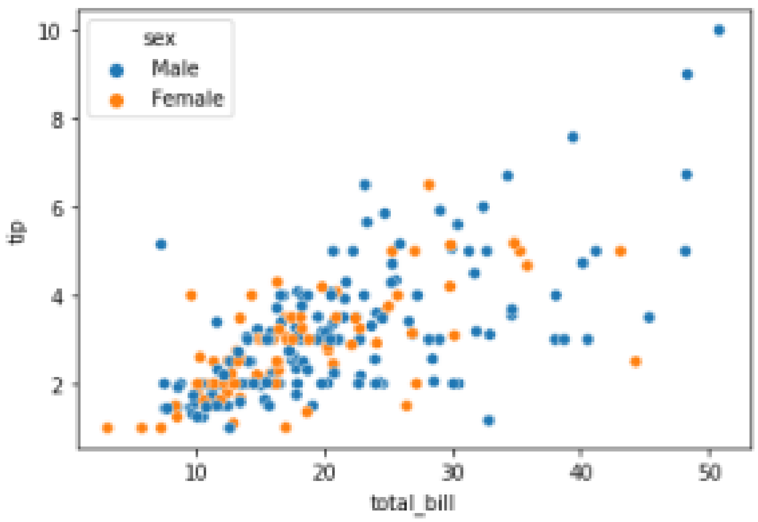
```
sns.scatterplot(tips["total_bill"], tips["tip"])
```



**Multivariate analysis with scatter plot:**

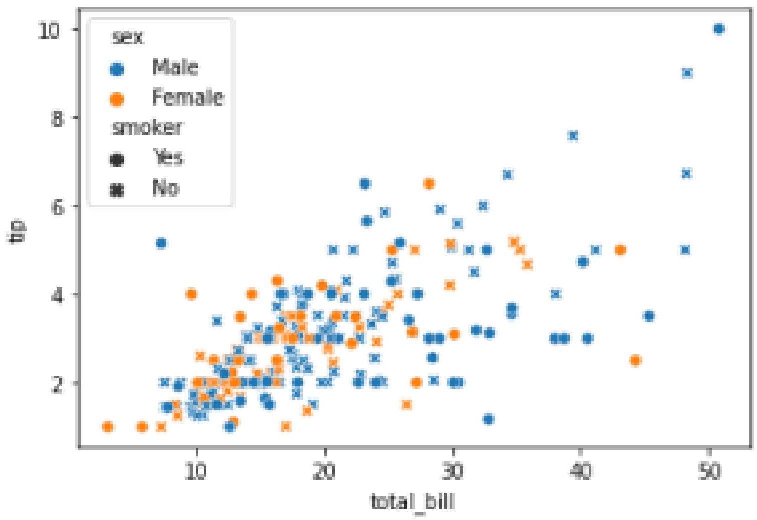
We can also plot 3 variable or 4 variable relationships with scatter plot. suppose we want to find the separate ratio of male and female with total bill and tip provided.

```
sns.scatterplot(tips["total_bill"], tips["tip"], hue=tips["sex"])
plt.show()
```



We can also see 4 variable multivariate analyses with scatter plots using style argument. Suppose along with gender we also want to know whether the customer was a smoker or not so we can do this.

```
sns.scatterplot(tips["total_bill"], tips["tip"], hue=tips["sex"],
style=tips['smoker'])
plt.show()
```



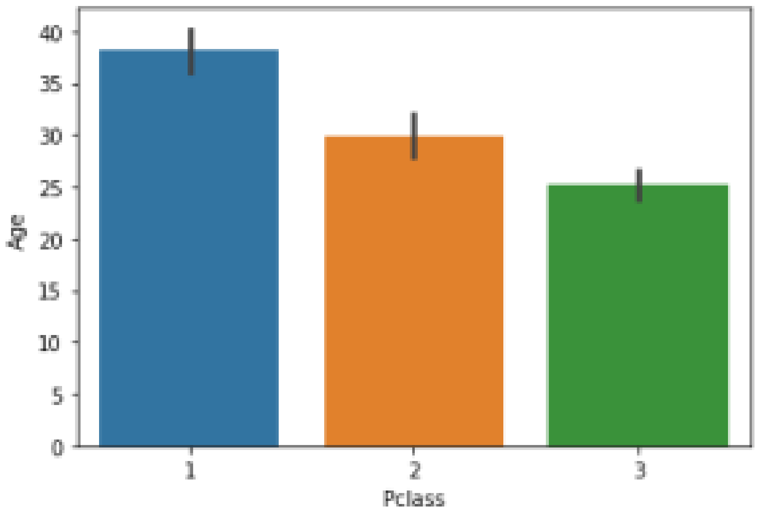
**Numerical and Categorical:**

If one variable is numerical and one is categorical then there are various plots that we can use for Bivariate and Multivariate analysis.

**1) Bar Plot:**

Bar plot is a simple plot which we can use to plot categorical variable on the x-axis and numerical variable on y-axis and explore the relationship between both variables. The blacktip on top of each bar shows the confidence Interval. let us explore P-Class with age.

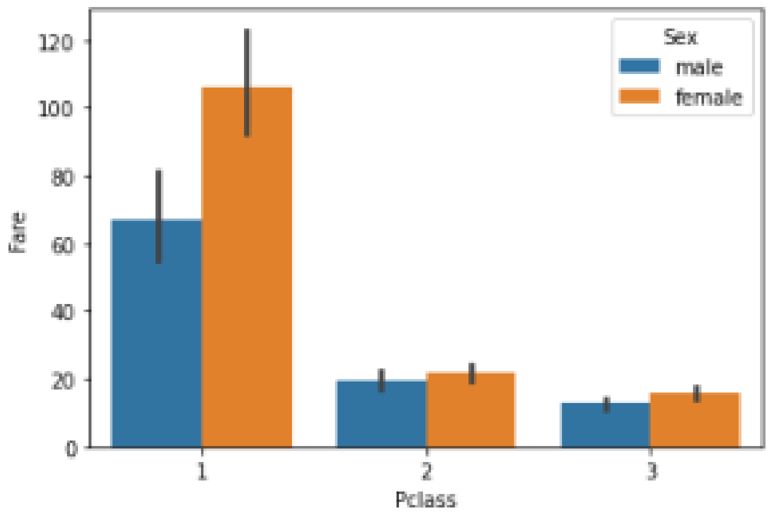
```
sns.barplot(data['Pclass'], data['Age'])

plt.show()
```



**Multivariate analysis using Bar plot:**

Hue’s argument is very useful which helps to analyze more than 2 variables. Now along with the above relationship we want to see with gender.

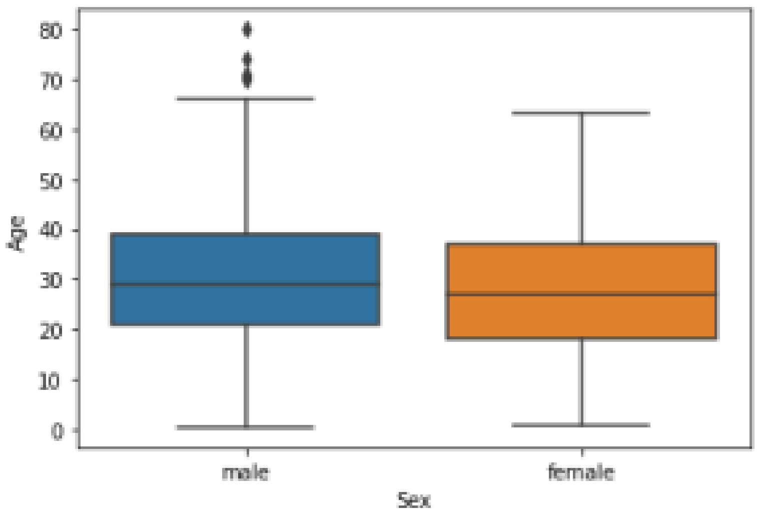
```
sns.barplot(data['Pclass'], data['Fare'], hue = data["Sex"])

plt.show()
```



**2) Boxplot:**

We have already study about boxplots in the Univariate analysis above. we can draw a separate boxplot for both the variable. let us explore gender with age using a boxplot.

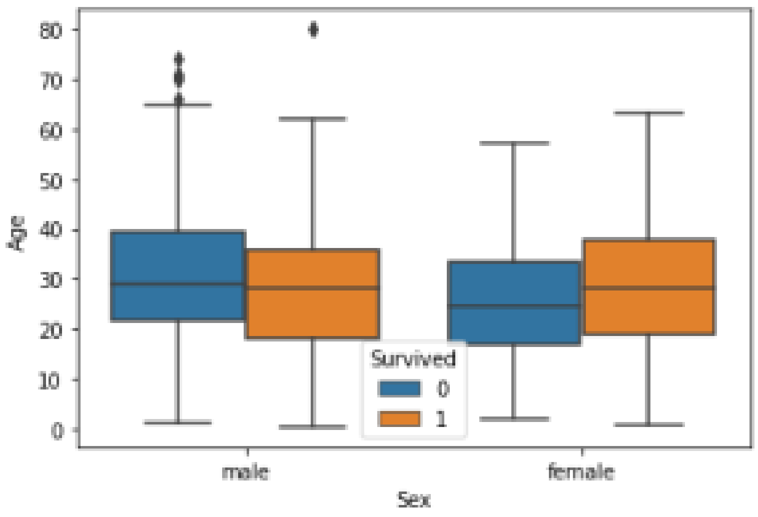
```
sns.boxplot(data['Sex'], data["Age"])
```



**Multivariate analysis with boxplot:**

Along with age and gender let’s see who has survived and who has not.

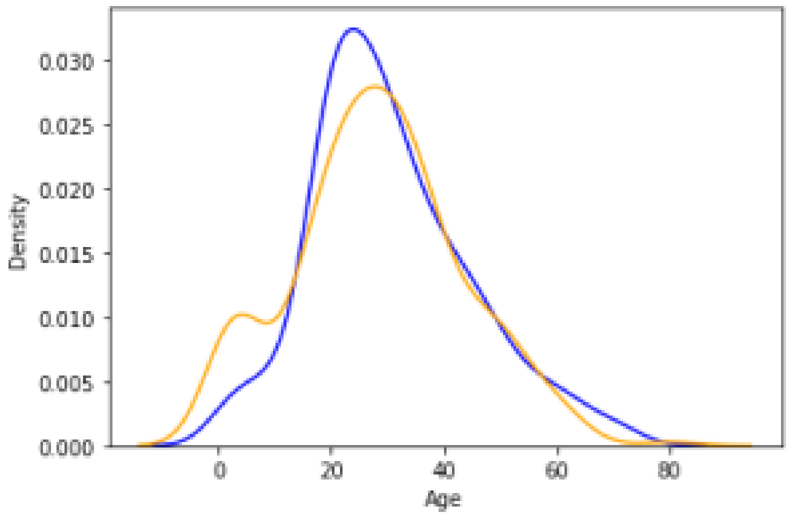
```
sns.boxplot(data['Sex'], data["Age"], data["Survived"])

plt.show()
```



**3) Distplot:**

Distplot explains the PDF function using kernel density estimation. Distplot does not have a hue parameter but we can create it. Suppose we want to see the probability of people with an age range that of survival probability and find out whose survival probability is high to the age range of death ratio.

```
sns.distplot(data[data['Survived'] == 0]['Age'], hist=False, color="blue")
sns.distplot(data[data['Survived'] == 1]['Age'], hist=False, color="orange")
plt.show()
```



In above graph, the blue one shows the probability of dying and the orange plot shows the survival probability. If we observe it we can see that children's survival probability is higher than death and which is the opposite in the case of aged peoples. This small analysis tells sometimes some big things about data and it helps while preparing data stories.

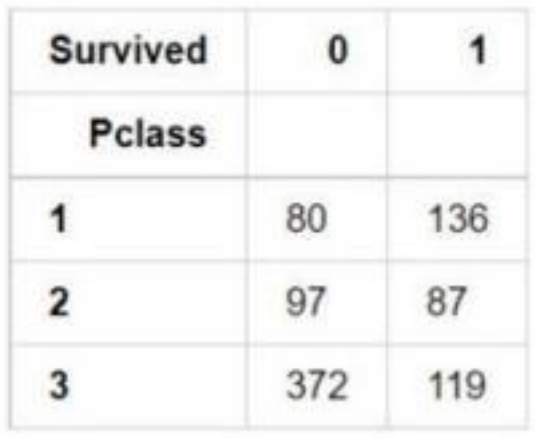
## Categorical and Categorical:

Now, we will work on categorical and categorical columns.

**1) Heatmap:**

If you have ever used a crosstab function of pandas then Heatmap is a similar visual representation of that only. It basically shows that how much presence of one category concerning another category is present in the dataset. let me show first with crosstab and then with heatmap.

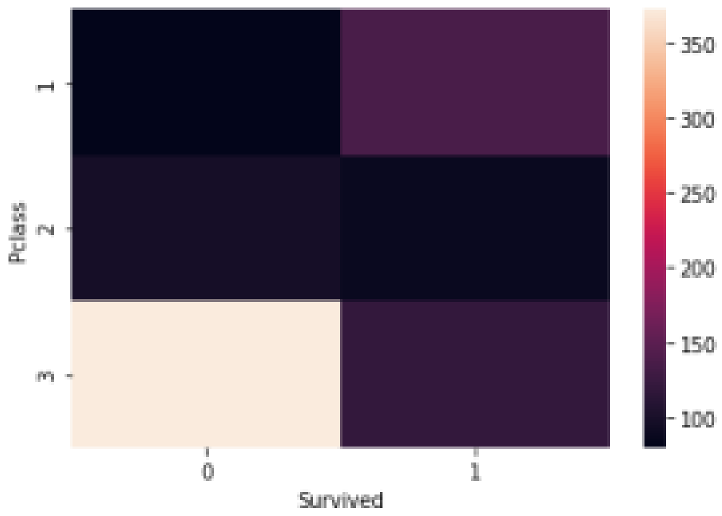
```
pd.crosstab(data['Pclass'], data['Survived'])
```



| Survived | 0 | 1 |
|---|---|---|
| Pclass | | |
| 1 | 80 | 136 |
| 2 | 97 | 87 |
| 3 | 372 | 119 |

Now with heatmap, we have to find how many people survived and died.

```
sns.heatmap(pd.crosstab(data['Pclass'], data['Survived']))
```



**2) Cluster map:**

We can also use a cluster map to understand the relationship between two categorical variables. A cluster map basically plots a dendrogram that shows the categories of similar behavior together.

```
sns.clustermap(pd.crosstab(data['Parch'], data['Survived']))

plt.show()
```

**Conclusion:** In this assignment, we covered how to plot different type of Data Visualization plots. Also studied how to generate inference from it.

## Group A

## Assignment No: 10

**Title: Data Visualization III**

Download the Iris flower dataset or any other dataset into a DataFrame. (e.g., https://archive.ics.uci.edu/ml/datasets/Iris ). Scan the dataset and give the inference as: 1. List down the features and their types (e.g., numeric, nominal) available in the dataset. 2. Create a histogram for each feature in the dataset to illustrate the feature distributions. 3. Create a box plot for each feature in the dataset.

4. Compare distributions and identify outliers.

**Objective of the Assignment:** Students should be able to perform the data Visualization operation using Python on any open source dataset .

**Prerequisite:**

1. Basic of Python Programming
2. Seaborn Library, Concept of Data Visualization.
3. Types of variables

**Theory:**

**Boxplot:A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable. The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution, except for points that are determined to be "outliers" using a method that is a function of the inter-quartile range.**

**Draw a box plot to show distributions with respect to categories.**

**seaborn.boxplot(*data=None, *, x=None, y=None, hue=None, order=None, hue_order=None, orient=None, color=None, palette=None, saturation=0.75, width=0.8, dodge=True, fliersize=5, linewidth=None, whis=1.5, ax=None, **kwargs*)**

**Draw a single horizontal boxplot, assigning the data directly to the coordinate variable:**

```
df = sns.load_dataset("iris")
```

```
sns.boxplot(x=df["sepal_length"])
```

**Group by a categorical variable, referencing columns in a dataframe:**

```
sns.boxplot(data=df, x="Sepal_length", y="species")
```

## Histograms:

A histogram is basically used to represent data provided in a form of some groups.It is accurate method for the graphical representation of numerical data distribution. It is a type of bar plot where X-axis represents the bin ranges while Y-axis gives information about frequency.

***Syntax:*** *matplotlib.pyplot.hist(x, bins=None, range=None, density=False, weights=None, cumulative=False, bottom=None, histtype='bar', align='mid', orientation='vertical', rwidth=None, log=False, color=None, label=None, stacked=False, *, data=None, **kwargs)*
The following table shows the parameters accepted by matplotlib.pyplot.hist() function :

| Attribute Parameter |
|---|
| x array or sequence of array |
| bins optional parameter contains integer or sequence or strings |
| density optional parameter contains boolean values |
| range optional parameter represents upper and lower range of bins |
| histtype optional parameter used to create type of histogram [bar, barstacked, step, stepfilled], default is"bar"<br>align optional parameter controls the plotting of histogram [left, right, mid] weights<br>optional parameter contains array of weights having same dimensions as x |

| |
|---|
| bottom location of the basline of each bin |
| rwidth optional parameter which is relative width of the bars with respect to bin width |
| color optional parameter used to set color or sequence of color specs |

| |
|---|
| label optional parameter string or sequence of string to match with multiple datasets |
| log optional parameter used to set histogram axis on log scale |

**Algorithm:**

**1. Import required libraries.**

*import numpy as np*

*import pandas as pd*

*import matplotlib.pyplot as plt*

import seaborn as sns

*import os*

**2. Create the data frame for downloaded iris.csv dataset.**

*os.chdir("D:\Pandas")*

*df =*

*pd.read_csv("Iris.csv")df*

**3. Apply data preprocessing techniques.**

*df.isnull().sum()*

*df.describe()*

**4. Plot the box plot for each feature in the dataset and observe and detect the outliers.**

*sns.boxplot(x='Species', y='SepalLengthCm', data=df)*

```
plt.title('Distribution of sepal length')

   plt.show()
```

**5. Plot the histogram for each feature in the dataset.**

```
df.hist()
```

**Conclusion:** In this assignment, we covered how to plot boxplot and Histogram on iris dataset. Also studied how to generate inference from it.

# DS&BDL
# Assignment 11

## Title: Hadoop MapReduce Framework- WordCount application

## Aim:

Write a code in JAVA for a simple WordCount application that counts the number of occurrences of each word in a given input set using the Hadoop MapReduce framework on local-standalone set-up.

## Objective:

**By completing this task, students will learn the following**

1. Hadoop Distributed File System.
2. MapReduce Framework.

**Software/Hardware Requirements:** 64-bit Open source OS-Linux, Java, Hadoop.

## Theory:

Map and Reduce tasks in Hadoop-With in a MapReduce job there are two separate tasks map task and reduce task.

**Map task-** A MapReduce job splits the input dataset into independent chunks known as input splits in Hadoop which are processed by the map tasks in a completely parallel manner. Hadoop framework creates separate map task for each input split.

**Reduce task-** The output of the maps is sorted by the Hadoop framework which then becomes input to the reduce tasks.

Hadoop MapReduce framework operates exclusively on <key, value> pairs. In a MapReduce job, the input to the Map function is a set of <key, value> pairs and output is also a set of <key, value> pairs. The output <key, value> pair may have different type from the input <key, value> pair.

<K1, V1> -> map -> (K2, V2)

The output from the map tasks is sorted by the Hadoop framework. MapReduce guarantees that the input to every reducer is sorted by key. Input and output of the reduce task can be represented as follows.

<K2, list(V2)> -> reduce -> <K3, V3>

### 1. Creating and copying input file to HDFS

If you already have a file in HDFS which you want to use as input then you can skip this step. First thing is to create a file which will be used as input and copy it to HDFS. Let's say you have a file wordcount.txt with the following content.

Hello wordcount MapReduce Hadoop program.
This is my first MapReduce program.

You want to copy this file to /user/process directory with in HDFS. If that path doesn't exist then you need to create those directories first.

HDFS Command Refernce List Link- https://www.netjstech.com/2018/02/hdfs-commands reference-list.html

```
hdfs dfs -mkdir -p /user/process
```

Then copy the file wordcount.txt to this directory.

```
hdfs dfs -put /netjs/MapReduce/wordcount.txt
/user/process
```

## 2. Write your wordcount mapreduce code.

WordCount example reads text files and counts the frequency of the words. Each mapper takes a line of the input file as input and breaks it into words. It then emits a key/value pair of the word (In the form of (word, 1)) and each reducer sums the counts for each word and emits a single key/value with the word and sum.

In the word count MapReduce code there is a Mapper class (MyMapper) with map function and a Reducer class (MyReducer) with a reduce function.

### 1. Map function

From the wordcount.txt file Each line will be passed to the map function in the following format.
<0, Hello wordcount MapReduce Hadoop program.>
<41, This is my first MapReduce program.>
In the map function the line is split on space and each word is written to the context along with the value as 1.
So the output from the map function for the two lines will be as follows.

**Line 1 <key, value> output**
(Hello, 1)
(wordcount, 1)
(MapReduce, 1)
(Hadoop, 1)
(program., 1)
**Line 2 <key, value> output**
(This, 1)
(is, 1)
(my, 1)
(first, 1)
(MapReduce, 1)
(program., 1)

### 2. Shuffling and sorting by Hadoop Framework

The output of map function doesn't become input of the reduce function directly. It goes through shuffling and sorting by Hadoop framework. In this processing the data is sorted and grouped by keys.
After the internal processing the data will be in the following format. This is the input to reduce function.
<Hadoop, (1)>
<Hello, (1)>
<MapReduce, (1, 1)>

<This, (1)>
<first, (1)>
<is, (1)>
<my, (1)>
<program., (1, 1)>
<wordcount, (1)>

### 3. How reduce task works in Hadoop

As we just saw the input to the reduce task is in the format (key, list<values>). In the reduce function, for each input <key, value> pair, just iterate the list of values for each key and add the values that will give the count for each key.
Write that key and sum of values to context, that <key, value> pair is the output of the reduce function.

```
Hadoop 1
Hello 1
MapReduce 2
This 1
first 1
is 1
my 1
program. 2
wordcount 1
```

## 3. Creating jar of your wordcount MapReduce code

You will also need to add at least the following Hadoop jars so that your code can compile. You will find these jars inside the /share/hadoop directory of your Hadoop installation. With in **/share/hadoop** path look in hdfs, mapreduce and common directories for required jars.

```
hadoop-common-2.9.0.jar
hadoop-hdfs-2.9.0.jar
hadoop-hdfs-client-2.9.0.jar
hadoop-mapreduce-client-core-2.9.0.jar
hadoop-mapreduce-client-common-2.9.0.jar
hadoop-mapreduce-client-jobclient-2.9.0.jar
hadoop-mapreduce-client-hs-2.9.0.jar
hadoop-mapreduce-client-app-2.9.0.jar
commons-io-2.4.jar
```

Once you are able to compile your code you need to create jar file.

**In the eclipse IDE righ click on your Java program and select Export – Java – jar file.**

## 4. Running the MapReduce code

You can use the following command to run the program. Assuming you are in your hadoop installation directory**.**

```
bin/hadoop jar /netjs/MapReduce/wordcount.jar
org.netjs.WordCount /user/process /user/out
```

Explanation for the arguments passed is as follows-

**/netjs/MapReduce/wordcount.jar** is the path to your jar file.

**org.netjs.WordCount** is the fully qualified path to your Java program class.

**/user/process** – path to input directory.

**/user/out** – path to output directory.

One your word count MapReduce program is succesfully executed you can verify the output file.

```
hdfs dfs -ls /user/out
```

Found 2 items

-rw-r--r-- 1 netjs supergroup 0 2018-02-27 13:37 /user/out/_SUCCESS

-rw-r--r-- 1 netjs supergroup 77 2018-02-27 13:37 /user/out/part-r-00000

As you can see Hadoop framework creates output files using part-r-xxxx format. Since only one reducer is used here so there is only one output file part-r-00000. You can see the content of the file using the following command.

```
hdfs dfs -cat /user/out/part-r-00000
```

```
Hadoop 1
Hello 1
MapReduce 2
This 1
first 1
is 1
my 1
program. 2
wordcount 1
```

**Conclusion:** In this assignment, we have learned what is HDFS and How Hadoop MapReduce framework is used to counts the number of occurrences of each word in a given input set.

## Group B

## Assignment No: 2

**Title of the Assignment: Big Data Analytics I**

Design a distributed application using MapReduce which processes a log file of a system.

**Objective:** Student will be able to learn:
Installation of Hadoop for Distributed Environment.

**Theory**:

- **Steps to Install Hadoop for distributed environment**

**Steps to Install Hadoop for distributed environment:**

Initially create one folder logfiles1 on the desktop. In that folder store input file (access_log_short.csv), SalesMapper.java, SalesCountryReducer.java, SalesCountryDriver.java files)

**Step 1)** Go to the Hadoop home directory and format the NameNode.

```
cd hadoop-2.7.3
```

```
bin/hadoop namenode -format
```

**Step 2)** Once the NameNode is formatted, go to hadoop-2.7.3/sbin directory and start all the daemons/nodes.

```
cd hadoop-2.7.3/sbin
```

**1) Start NameNode:**

The NameNode is the centerpiece of an HDFS file system. It keeps the directory tree of all files stored in the HDFS and tracks all the files stored across the cluster.

```
./hadoop-daemon.sh start namenode
```

**2) Start DataNode:**

On startup, a DataNode connects to the Namenode and it responds to the requests from the Namenode for different operations.

```
./hadoop-daemon.sh start datanode
```

**3) Start ResourceManager:**
ResourceManager is the master that arbitrates all the available cluster resources and thus helps in managing the distributed applications running on the YARN system. Its work is to manage each NodeManagers and each application's ApplicationMaster.

```
./yarn-daemon.sh start resourcemanager
```

**4) Start NodeManager:**

The NodeManager in each machine framework is the agent which is responsible for managing containers, monitoring their resource usage and reporting the same to the ResourceManager.

```
./yarn-daemon.sh start nodemanager
```

**5) Start JobHistoryServer:**

JobHistoryServer is responsible for servicing all job history related requests from client.

```
./mr-jobhistory-daemon.sh start historyserver
```

**Step 3)** To check that all the Hadoop services are up and running, run the below

command. jps

**Step 4)** cd

**Step 5)** sudo mkdir mapreduce_vijay

**Step 6)** sudo chmod 777 -R mapreduce_vijay/

**Step 7)** sudo chown -R vijay mapreduce_vijay/

**Step 8)** sudo cp /home/vijay/Desktop/logfiles1/* ~/mapreduce_vijay/

**Step 9)** cd mapreduce_vijay/

**Step 10)** ls
**Step 11)** sudo chmod +r *.*

**Step 12)** export CLASSPATH="/home/vijay/hadoop-2.7.3/share/hadoop/mapreduce/hadoop mapreduce-client-core-2.7.3.jar:/home/vijay/hadoop-2.7.3/share/hadoop/mapreduce/hadoop mapreduce-client-common-2.7.3.jar:/home/vijay/hadoop-2.7.3/share/hadoop/common/hadoop common-2.7.3.jar:~/mapreduce_vijay/SalesCountry/*:$HADOOP_HOME/lib/*"

**Step 13)** javac -d . SalesMapper.java SalesCountryReducer.java

SalesCountryDriver.java **Step 14)** ls

**Step 15)** cd SalesCountry/

**Step 16)** ls (check is class files are created)

**Step 17)** cd ..

**Step 18)** gedit Manifest.txt
(add following lines to it:
Main-Class: SalesCountry.SalesCountryDriver)

**Step 19)** jar -cfm mapreduce_vijay.jar Manifest.txt SalesCountry/*.class

**Step 20)** ls

**Step 21)** cd

**Step 22)** cd mapreduce_vijay/

**Step 23)** sudo mkdir /input200

**Step 24)** sudo cp access_log_short.csv /input200

**Step 25)** $HADOOP_HOME/bin/hdfs dfs -put /input200 /

**Step 26)** $HADOOP_HOME/bin/hadoop jar mapreduce_vijay.jar /input200 /output200

**Step 27)** hadoop fs -ls /output200

**Step 28)** hadoop fs -cat /out321/part-00000
**Step 29)** Now open the Mozilla browser and go to **localhost:50070/dfshealth.html** to check the NameNode interface.

**Conclusion:** In this way we have designed a distributed application using MapReduce which processes a log file of a system

## Group B

## Assignment No: 3

------------------------------------------------------------------------------------------------------------

**Title:Write a simple program in SCALA using Apache Spark framework**

**Objective:**

- **Steps to Install Scala**
- **Apache Spark Framework Installation**

**Theory**:

### 1) Install Scala

**Step 1)** java -version

**Step 2)** Install **Scala** from the apt repository by running the following commands to search for scala and install it.

sudo apt search scala ⇒ Search for the package

sudo apt install scala ⇒ Install the package

**Step 3)** To verify the installation of **Scala**, run the following command.

scala -version

### 2) Apache Spark Framework Installation

Apache Spark is an open-source, distributed processing system used for **big data workloads**. It utilizes in-memory caching, and optimized query execution for fast analytic queries against data of any size.

**Step 1)** Now go to the official Apache Spark download page and grab the latest version (i.e. 3.2.1) at the time of writing this article. Alternatively, you can use the wget command to download the file directly in the terminal.

```
wget https://apachemirror.wuchna.com/spark/spark-3.2.1/spark-3.2.1-bin-hadoop2.7.tgz
```

**Step 2)** Extract the Apache Spark tar file.

```
tar -xvzf spark-3.1.1-bin-hadoop2.7.tgz
```

**Step 3)** Move the extracted **Spark** directory to **/opt** directory.

```
sudo mv spark-3.1.1-bin-hadoop2.7 /opt/spark
```

## Configure Environmental Variables for Spark

**Step 4)** Now you have to set a few environmental variables in **.profile** file before starting up the spark.

```
echo "export SPARK_HOME=/opt/spark" >> ~/.profile
echo "export PATH=$PATH:/opt/spark/bin:/opt/spark/sbin" >> ~/.profile
echo "export PYSPARK_PYTHON=/usr/bin/python3" >> ~/.profile
```

**Step 5)** To make sure that these new environment variables are reachable within the shell and available to Apache Spark, it is also mandatory to run the following command to take recent changes into effect.

```
source ~/.profile
```

**Step 6)** ls -l /opt/spark

## Start Apache Spark in Ubuntu

**Step 7)** Run the following command to start the **Spark** master service and slave service. start-master.sh

```
start-workers.sh spark://localhost:7077
```

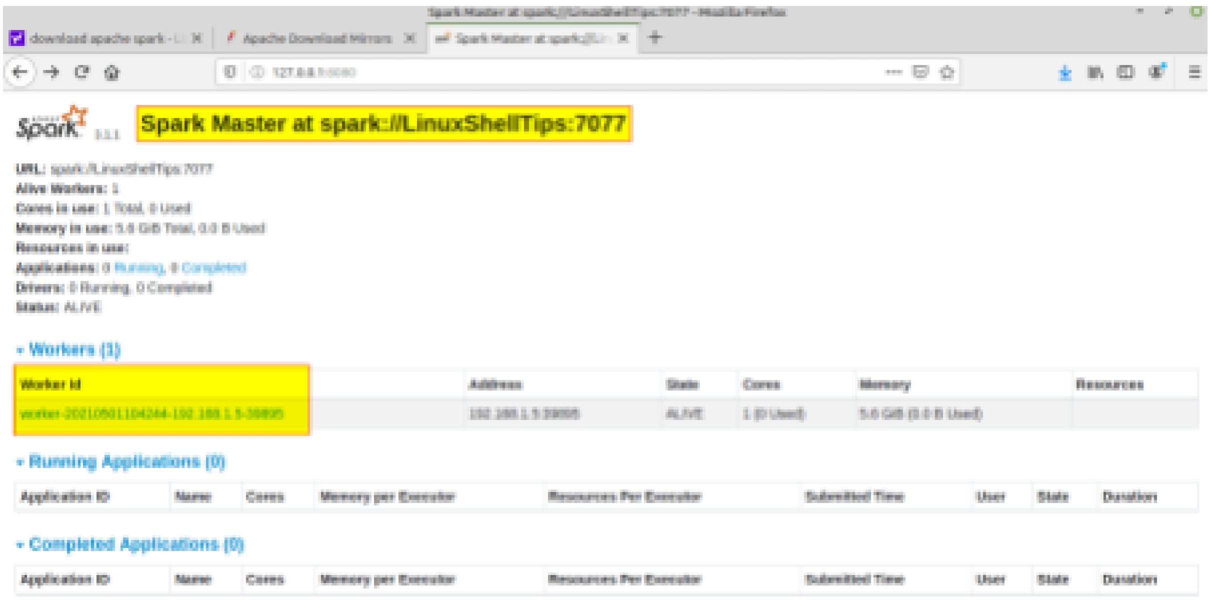
(if workers not starting then remove and install openssh:

sudo apt-get remove openssh-client openssh-server

sudo apt-get install openssh-client openssh-server)

**Step 8)** Once the service is started go to the browser and type the following URL access spark page. From the page, you can see my master and slave service is started.

http://localhost:8080/



Spark Master at spark://LinuxShellTips:7077

URL: spark://LinuxShellTips:7077
Alive Workers: 1
Cores in use: 1 Total, 0 Used
Memory in use: 5.8 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (1)

| Worker Id | Address | State | Cores | Memory | Resources |
|---|---|---|---|---|---|
| worker-20210501104244-192.168.1.5-39895 | 192.168.1.5:39895 | ALIVE | 1 (0 Used) | 5.8 GiB (0.0 B Used) | |

Running Applications (0)

| Application ID | Name | Cores | Memory per Executor | Resources Per Executor | Submitted Time | User | State | Duration |
|---|---|---|---|---|---|---|---|---|

Completed Applications (0)

| Application ID | Name | Cores | Memory per Executor | Resources Per Executor | Submitted Time | User | State | Duration |
|---|---|---|---|---|---|---|---|---|

**Step 9)** You can also check if **spark-shell** works fine by launching the **spark-shell** command.
Spark-shell

sudo apt install snapd

snap find “intellij”
    intellij-idea-community - - classic

**Start Intellij IDE community Edition**
**Source Code:**

**/* Sample Code to print Statement */**

```
object ExampleString {
   def main(args: Array[String]) {

       //declare and assign string variable "text"
       val text : String = "You are reading SCALA programming language.";

       //print the value of string variable "text"


       println("Value of text is: " + text);

    }
 }
```

**/**Scala program to find a number is positive, negative or positive.*/**

```
       object ExCheckNumber {
       def main(args: Array[String]) {

              /**declare a variable*/
              var number= (-100);

              if(number==0){
              println("number is zero");
               }
              else if(number>0){


              println("number is positive");
               }
              else{
              println("number is negative");
               }
        }
        }
```

**/*Scala program to print your name*/**

```
object ExPrintName {
       def main(args: Array[String]) {
              println("My name is Mike!")
        }
        }
```

**/**Scala Program to find largest number among two numbers.*/**

```
object ExFindLargest {
def main(args: Array[String]) {
var number1=20;
var number2=30;
var x = 10;

if( number1>number2){
      println("Largest number is:" + number1);
 }
else{
      println("Largest number is:" + number2);
 }
 }
 }
```

Conclusion:Thus we have studied how to execute a program in SCALA using Apache Spark Framework.