**JSPM'S Bhivarabai Sawant Institute of Technology and Research, Wagholi, Pune**

**Department of Computer Engineering**

**Lab Manual (2021-22)**

**Pattern-2019**

**Class: TE Computer Term: II**

**Data Science and Big Data Analytics Laboratory(310256)**

**Faculty Name: Chhaya Nayak**

**JSPM'S Bhivarabai Sawant Institute of Technology and Research, Wagholi, Pune**

**Department of Computer Engineering**

**Course Details**

**Course : Data Science and Big Data Analytics Laboratory(310256)**

**Class: TE Division: A And B**

|  |  |
| --- | --- |
| **COURSE OUTCOME** | |
| CO1 | Apply principles of Data Science for the analysis of real time problems |
| CO2 | Implement data representation using statistical methods |
| CO3 | Implement and evaluate data analytics algorithms |
| CO4 | Perform text preprocessing |
| CO5 | Implement data visualization techniques |
| CO6 | Use cutting edge tools and technologies to analyze Big Data |

**List of Experiments**

|  |  |  |
| --- | --- | --- |
| **Sr.**  **No.** | **Experiment List** | **Software Required** |
| **1** | **Data Wrangling, I** | Anaconda |
| **2** | **Data Wrangling II** | Anaconda |
| **3** | **Descriptive Statistics - Measures of Central Tendency and variability** | Anaconda |
| **4** | **Data Analytics I** | Anaconda |
| **5** | **Data Analytics II** | Anaconda |
| **6** | **Data Analytics III** | Anaconda |
| **7** | **Text Analytics** | Anaconda |
| **8** | **Data Visualization I** | Anaconda |
| **9** | **Data Visualization II** | Anaconda |
| **10** | **Data Visualization III** | Anaconda |

# ASSIGNMENT NUMBER 1

## Title of the Assignment: Data Wrangling, I

Perform the following operations using Python on any open source dataset (e.g., data.csv) Import all the required Python Libraries.

1. Locate open source data from the web (e.g. [https://www.kaggle.com](https://www.kaggle.com/)).
2. Provide a clear description of the data and its source (i.e., URL of the web site).
3. Load the Dataset into the pandas data frame.
4. Data Preprocessing: check for missing values in the data using pandas insult(), 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.

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

## Prerequisite:

1. Basic of Python Programming
2. Concept of Data Preprocessing, Data Formatting , Data Normalization and Data Cleaning.
3. ---------------------------------------------------------------------------------------------------------------

**Contents for Theory:**

## Introduction to Dataset

1. **Python Libraries for Data Science**

## Description of Dataset

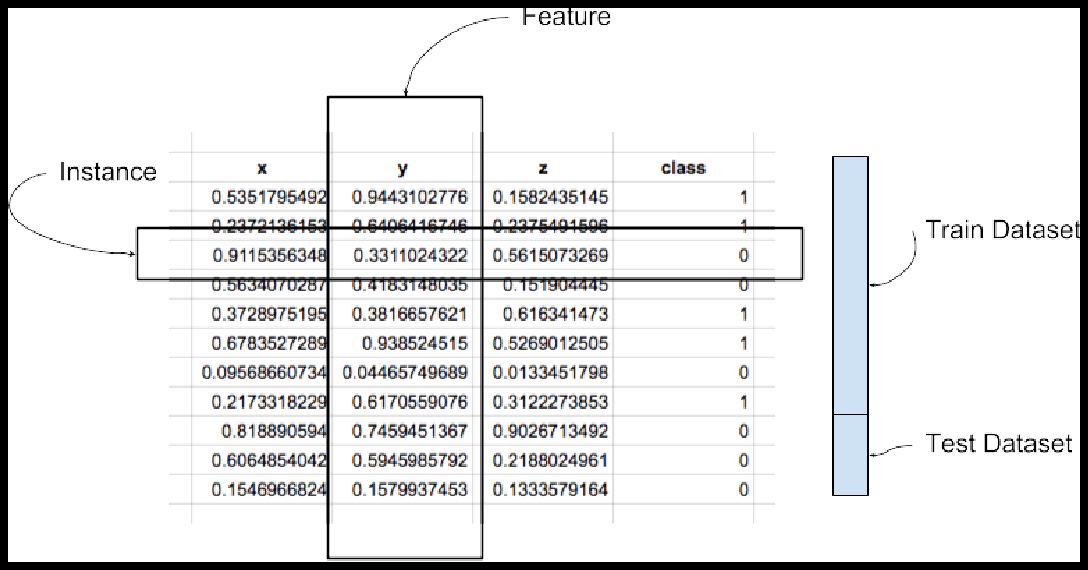
1. **Panda Dataframe functions for load the dataset**

## Panda functions for Data Preprocessing

1. **Panda functions for Data Formatting and Normalization**

## Panda Functions for handling categorical variables

## Introduction to Dataset

A dataset is a collection of records, similar to a relational database table. Records are similar to table rows, but the columns can contain not only strings or numbers, but also nested data structures such as lists, maps, and other records.

**Instance:** A single row of data is called an instance. It is an observation from the domain. **Feature:** A single column of data is called a feature. It is a component of an observation and is also called an attribute of a data instance. Some features may be inputs to a model (the predictors) and others may be outputs or the features to be predicted.

**Data Type**: Features have a data type. They may be real or integer-valued or may have a categorical or ordinal value. You can have strings, dates, times, and more complex types, but typically they are reduced to real or categorical values when working with traditional machine learning methods.

**Datasets**: A collection of instances is a dataset and when working with machine learning methods we typically need a few datasets for different purposes.

**Training Dataset:** A dataset that we feed into our machine learning algorithm to train our model.

**Testing Dataset:** A dataset that we use to validate the accuracy of our model but is not used to train the model. It may be called the validation dataset.

A possible confusing point about pandas data types is that there is some overlap between pandas, python and numpy. This table summarizes the key points:

|  |  |  |  |
| --- | --- | --- | --- |
| **Pandas dtype** | **Python type** | **NumPy type** | **Usage** |
| object | str or mixed | string\_, unicode\_, mixed types | Text or mixed numeric and non-numeric values |
| int64 | int | int\_, int8, int16, int32, int64, uint8, uint16, uint32, uint64 | Integer numbers |
| float64 | float | float\_, float16, float32, float64 | Floating point numbers |
| bool | bool | bool\_ | True/False values |
| datetime64 | NA | datetime64[ns] | Date and time values |
| timedelta[ns] | NA | NA | Differences between two datetimes |
| category | NA | NA | Finite list of text values |

## Python Libraries for Data Science

* 1. **Pandas**

Pandas is an open-source Python package that provides high-performance, easy-to-use data structures and data analysis tools for the labeled data in Python programming language.

## What can you do with Pandas?

* + 1. Indexing, manipulating, renaming, sorting, merging data frame
    2. Update, Add, Delete columns from a data frame
    3. Impute missing files, handle missing data or NANs
    4. Plot data with histogram or box plot

## NumPy

One of the most fundamental packages in Python, NumPy is a general-purpose array-processing package. It provides high-performance multidimensional array objects and tools to work with the arrays. NumPy is an efficient container of generic multi-dimensional data.

NumPy’s main object is the homogeneous multidimensional array. It is a table of elements or numbers of the same datatype, indexed by a tuple of positive integers. In NumPy, dimensions are called axes and the number of axes is called rank. NumPy’s array class is called ndarray aka array.

## What can you do with NumPy?

* + 1. Basic array operations: add, multiply, slice, flatten, reshape, index arrays
    2. Advanced array operations: stack arrays, split into sections, broadcast arrays
    3. Work with DateTime or Linear Algebra
    4. Basic Slicing and Advanced Indexing in NumPy Python

## Matplotlib

This is undoubtedly my favorite and a quintessential Python library. You can create stories with the data visualized with Matplotlib. Another library from the SciPy Stack, Matplotlib plots 2D figures.

## What can you do with Matplotlib?

Histogram, bar plots, scatter plots, area plot to pie plot, Matplotlib can depict a wide range of visualizations. With a bit of effort and tint of visualization capabilities, with Matplotlib, you can create just any visualizations:Line plots

* Scatter plots
* Area plots
* Bar charts and Histograms
* Pie charts
* Stem plots
* Contour plots
* Quiver plots
* Spectrograms

Matplotlib also facilitates labels, grids, legends, and some more formatting entities with Matplotlib.

## Seaborn

So when you read the official documentation on Seaborn, it is defined as the data visualization library based on Matplotlib that provides a high-level interface for drawing attractive and informative statistical graphics. Putting it simply, seaborn is an extension of Matplotlib with advanced features.

## What can you do with Seaborn?

* + 1. Determine relationships between multiple variables (correlation)
    2. Observe categorical variables for aggregate statistics
    3. Analyze univariate or bi-variate distributions and compare them between different data subsets
    4. Plot linear regression models for dependent variables
    5. Provide high-level abstractions, multi-plot grids
    6. Seaborn is a great second-hand for R visualization libraries like corrplot and ggplot.

## 5. Scikit Learn

Introduced to the world as a Google Summer of Code project, Scikit Learn is a robust machine learning library for Python. It features ML algorithms like SVMs, random forests, k-means clustering, spectral clustering, mean shift, cross-validation and more... Even NumPy, SciPy and related scientific operations are supported by Scikit Learn with Scikit Learn being a part of the SciPy Stack.

## What can you do with Scikit Learn?

* + 1. Classification: Spam detection, image recognition
    2. Clustering: Drug response, Stock price
    3. Regression: Customer segmentation, Grouping experiment outcomes
    4. Dimensionality reduction: Visualization, Increased efficiency
    5. Model selection: Improved accuracy via parameter tuning

## Description of Dataset:

The Iris dataset was used in R.A. Fisher's classic 1936 paper, The Use of Multiple Measurements in Taxonomic Problems, and can also be found on the UCI Machine Learning Repository.

It includes three iris species with 50 samples each as well as some properties about each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.

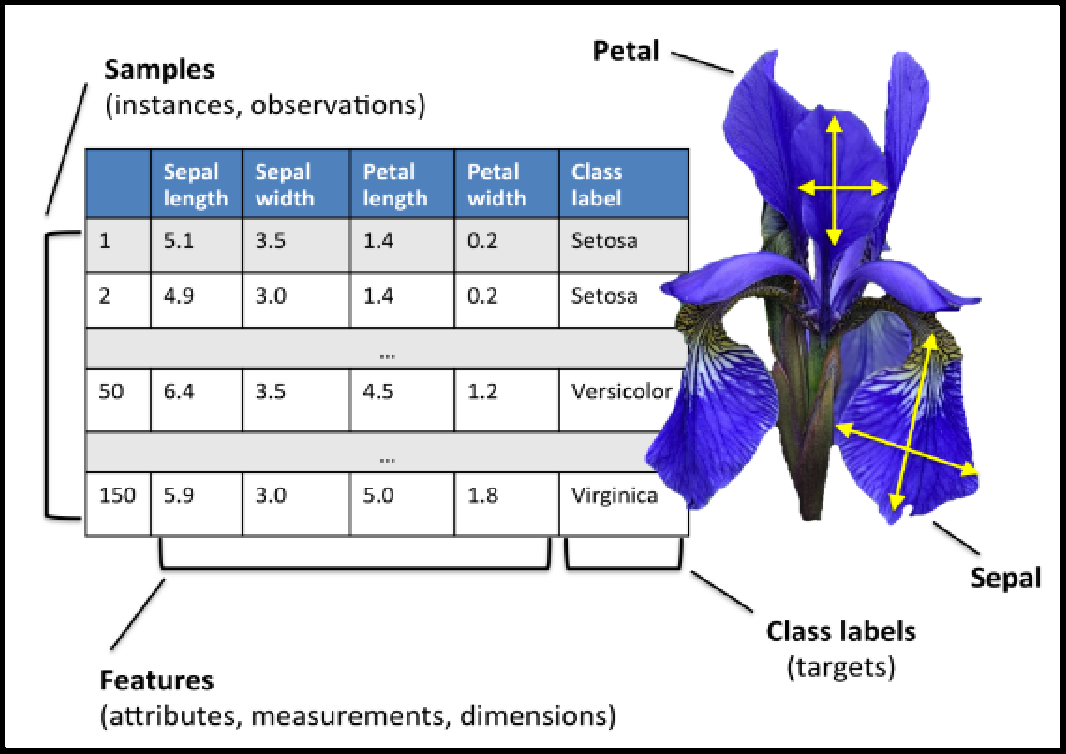
## Total Sample- 150

**The columns in this dataset are:**

* 1. Id
  2. SepalLengthCm
  3. SepalWidthCm
  4. PetalLengthCm
  5. PetalWidthCm
  6. Species

## 3 Different Types of Species each contain 50 Sample-

**Description of Dataset-**



## Panda Dataframe functions for Load Dataset

**# The columns of the resulting DataFrame have different dtypes. iris.dtypes**

1. The dataset is downloads from UCI repository.

## csv\_url='https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'

1. Now Read CSV File as a Dataframe in Python from from path where you saved the same The Iris data set is stored in .csv format. ‘.csv’ stands for comma separated values. It is easier to load .csv files in Pandas data frame and perform various analytical operations on it.

Load Iris.csv into a Pandas data frame —

## Syntax-

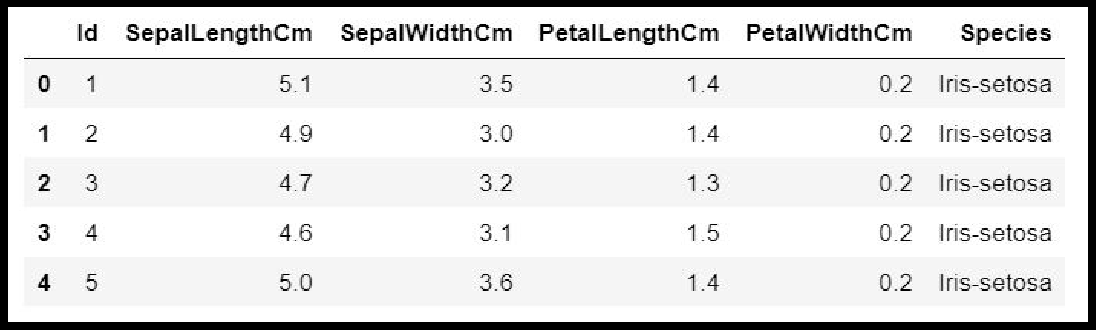
**iris = pd.read\_csv(csv\_url, header = None)**

1. The csv file at the UCI repository does not contain the variable/column names. They are located in a separate file.

## col\_names = ['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width','Species']

1. read in the dataset from the UCI Machine Learning Repository link and specify column names to use

## iris = pd.read\_csv(csv\_url, names = col\_names)



1. **Panda Dataframe functions for Data Preprocessing : Dataframe Operations:**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Data Frame Function** | **Description** |
| **1** | **dataset.head(n=5)** | **Return the first n rows.** |
| **2** | **dataset.tail(n=5)** | **Return the last n rows.** |
| **3** | **dataset.index** | The index (row labels) of the Dataset. |
| **4** | **dataset.columns** | The column labels of the Dataset. |
| **5** | **dataset.shape** | Return a tuple representing the dimensionality of the Dataset. |
| **6** | **dataset.dtypes** | Return the dtypes in the Dataset. |
| **7** | **dataset.columns.values** | Return the columns values in the Dataset in array format |
| **8** | **dataset.describe(include='all')** | Generate descriptive statisticsto view some basic statistical details like percentile, mean, std etc. of a data frame or a series of numeric values |
| **9** | **dataset['Column name]** | Read the Data Column wise. |
| **10** | **dataset.sort\_index(axis=1, ascending=False)** | Sort object by labels (along an axis). |
| **11** | **dataset.sort\_values(by="Colu mn name")** | Sort values by column name. |
| **12** | **dataset.iloc[5]** | Purely integer-location based indexing for selection by position. |
| **13** | **dataset[0:3]** | Selecting via [], which slices the rows. |
| **14** | **dataset.loc[:, "Col\_name1" , Col\_name2"]]** | Selection by label |
| **15** | **dataset.iloc[:n, :]** | a subset of the first n rows of the original data |
| **16** | **dataset.iloc[:, :n]** | a subset of the first n columns of the original data |
| **17** | **dataset.iloc[:m, :n]** | a subset of the first m rows and the first n columns |

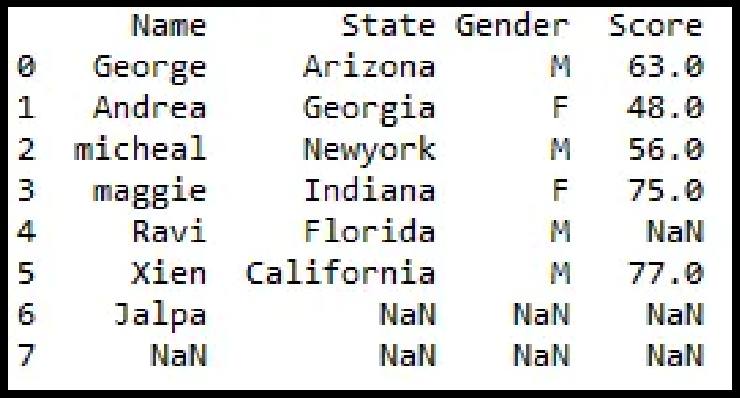
Few Examples of iLoc to slice data for iris Dataset

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **Data Frame Function** | **Description** | **Output** |
| **1** | **dataset.iloc[3:5, 0:2]** | Slice the data |  |
| **2** | **dataset.iloc[[1, 2,**  **4], [0, 2]]** | By lists of integer position locations, similar to the NumPy/Python style: |  |
| **3** | **dataset.iloc[1:3, :]** | For slicing rows explicitly: |  |
| **4** | **dataset.iloc[:, 1:3]** | For slicing Column explicitly: |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **4** | **dataset.iloc[1, 1]** | For getting a value explicitly: |  |
| **5** | **dataset['SepalLeng thCm'].iloc[5]** | Accessing Column and Rows by position |  |
| **6** | **cols\_2\_4=dataset.columns[2:4]dataset[cols\_2\_4]** | Get Column Name then get data from column |  |
| **7** | **dataset[dataset.columns[2:4]].iloc[5:1 0]** | in one Expression answer for the above two commands |  |

**Checking of Missing Values in Dataset:**

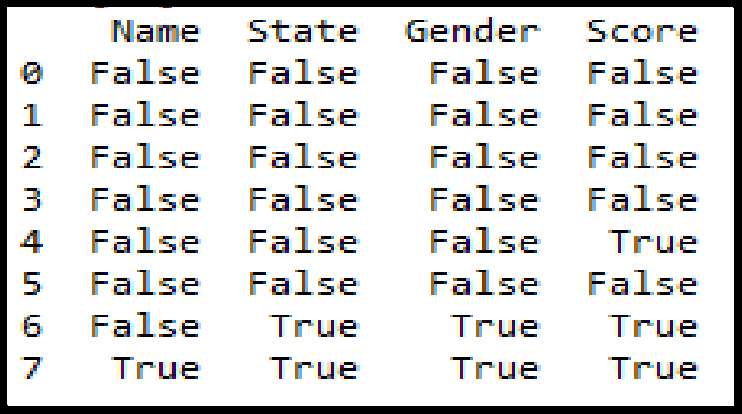
* **isnull()** is the function that is used to check missing values or null values in pandas python.
* isna() function is also used to get the count of missing values of column and row wise count of missing values
* The dataset considered for explanation is:



* 1. **is there any missing values in dataframe as a whole**

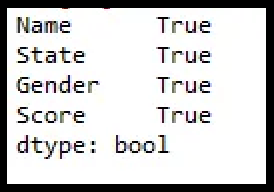
**Function:** DataFrame.isnull()

**Output:**



* 1. **is there any missing values across each column Function:** DataFrame**.**isnull().any()

**Output:**



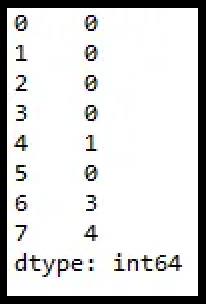
* 1. **count of missing values across each column using isna() and isnull()**

In order to get the count of missing values of the entire dataframe isnull() function is used. sum() which does the column wise sum first and doing another sum() will get the count of missing values of the entire dataframe.

**Function:** dataframe.isnull().sum().sum()

**Output** : 8

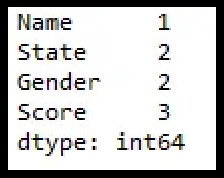
* 1. **count row wise missing value using isnull() Function:** dataframe.isnull().sum(axis = 1) **Output:**



* 1. **count Column wise missing value using isnull() Method 1:**

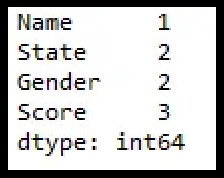
**Function:** dataframe.isnull().sum()

**Output:**



**Method 2:**

**unction:** dataframe.isna().sum()



* 1. **count of missing values of a specific column. Function:**dataframe.col\_name.isnull().sum()

# df1.Gender.isnull().sum()

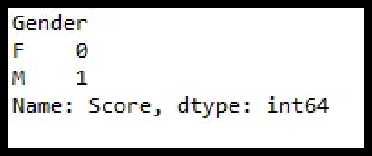
**Output: 2**

* 1. **groupby count of missing values of a column.**

In order to get the count of missing values of the particular column by group in pandas we will be using isnull() and sum() function with apply() and groupby() which performs the group wise count of missing values as shown below.

# Function: df1.groupby(['Gender'])['Score'].apply(lambda x: x.isnull().sum())

**Output:**



## Panda functions for Data Formatting and Normalization

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **Data Frame Function** | **Description** | **Output** |
| 1. | **df.dtypes** | To check the data type |  |
| 2. | **df['petal length (cm)']= df['petal length** | To change the data type (data type of ‘petal length (cm)'changed to int) |  |
|  | **(cm)'].astype("int** |  |
|  | **")** |  |

The Transforming data stage is about converting the data set into a format that can be analyzed or modelled effectively, and there are several techniques for this process.

* 1. **Data Formatting:** Ensuring all data formats are correct (e.g. object, text, floating number, integer, etc.) is another part of this initial ‘cleaning’ process. If you are working with dates in Pandas, they also need to be stored in the exact format to use special date-time functions.

Functions used for data formatting

* 1. **Data normalization:** Mapping all the nominal data values onto a uniform scale (e.g. from 0 to 1) is involved in data normalization. Making the ranges consistent across variables helps with statistical analysis and ensures better comparisons later on.It is also known as Min-Max scaling.

## Algorithm:

**Step 1 :** Import pandas and sklearn library for preprocessing

from sklearn import preprocessing

**Step 2:** Load the iris dataset in dataframe object df

**Step 3:** Print iris dataset.

**df.head()**

**Step 5:** Create a minimum and maximum processor object

**min\_max\_scaler = preprocessing.MinMaxScaler()**

## Step 6: Separate the feature from the class label

**x=df.iloc[:,:4]**

**Step 6:** Create an object to transform the data to fit minmax processor

**x\_scaled = min\_max\_scaler.fit\_transform(x)**

**Step 7:**Run the normalizer on the dataframe

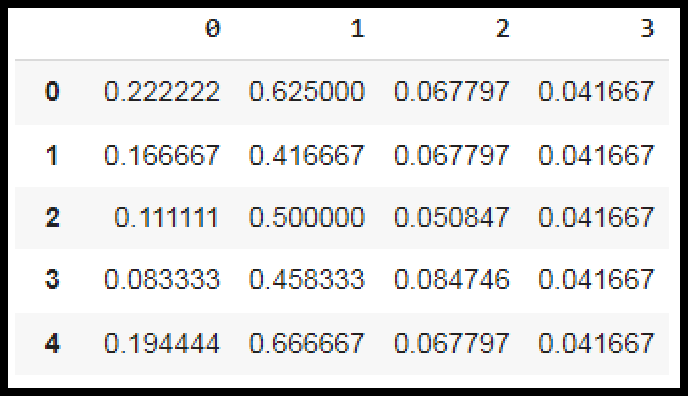
**df\_normalized = pd.DataFrame(x\_scaled)**

**Step 8:** View the dataframe

**df\_normalized**

## Output: After Step 3:

**Output after step 8:**



## Panda Functions for handling categorical variables

* **Categorical variables** have values that **describe a ‘quality’ or ‘characteristic’**

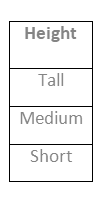
of a data unit, like **‘what type’ or ‘which category’.**

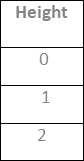
* Categorical variables fall into **mutually exclusive (in one category or in another)** and **exhaustive (include all possible options)** categories. Therefore, categorical variables are qualitative variables and t**end to be represented by a non-numeric value.**
* Categorical features refer **to string type data** and can be easily understood by human beings. But in case of a **machine, it cannot interpret the categorical data directly**. Therefore, the categorical data must be **translated into numerical data that can be understood by machine.**

There are many ways to convert categorical data into numerical data. Here the three most used methods are discussed.

* 1. **Label Encoding:** Label Encoding refers to **converting the labels into a numeric form** so as to convert them into the machine-readable form. **It is an important preprocessing step for the structured dataset** in supervised learning.

**Example :** Suppose we have a column Height in some dataset. After applying label encoding,theHeight column is converted into:





where 0 is the label for tall, 1 is the label for medium, and 2 is a label for short height. **Label Encoding on iris dataset:** For iris dataset the target column which is Species. It contains three species Iris-setosa, Iris-versicolor, Iris-virginica.

## Sklearn Functions for Label Encoding:

* + - **preprocessing.LabelEncoder :** It Encode labels with value between 0 and n\_classes-1.

## fit\_transform(y):

**Parameters:** yarray-like of shape (n\_samples,**) Target values.**

**Returns:** yarray-like of shape (n\_samples,)

## Encoded labels.

This transformer should be used to encode target values, and not the input.

## Algorithm:

**Step 1 :** Import pandas and sklearn library for preprocessing

from sklearn import preprocessing

**Step 2:** Load the iris dataset in dataframe object df

**Step 3:** Observe the unique values for the Species column.

df['Species'].unique()

**output: array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)**

**Step 4:** define label\_encoder object knows how to understand word labels.

label\_encoder = preprocessing.LabelEncoder()

**Step 5:** Encode labels in column 'species'.

df['Species']= label\_encoder.fit\_transform(df['Species'])

**Step 6:** Observe the unique values for the Species column.

## df['Species'].unique()

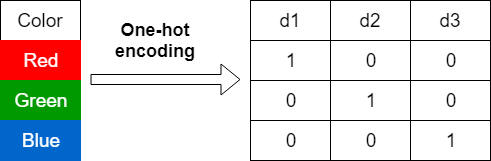
**Output: array([0, 1, 2], dtype=int64)**

* Use LabelEncoder when there are only two possible values of a categorical feature. For example, features having value such as yes or no. Or, maybe, gender features when there are only two possible values including male or female.

**Limitation:** Label encoding converts the data in machine-readable form, but it assigns a **unique number(starting from 0) to each class of data**. This may lead to the generation of **priority issues in the data sets**. A label with a high value may be considered to have high priority than a label having a lower value.

## One-Hot Encoding:

In one-hot encoding, we create a new set of dummy (binary) variables that is equal to the number of categories (k) in the variable. For example, let’s say we have a categorical variable Color with three categories called “Red”, “Green” and “Blue”, we need to use three dummy variables to encode this variable using one-hot encoding. A dummy (binary) variable just takes the value 0 or 1 to indicate the exclusion or inclusion of a category.



In one-hot encoding,

**“Red”** color is encoded as **[1 0 0]** vector of size 3.

**“Green”** color is encoded as **[0 1 0]** vector of size 3.

**“Blue”** color is encoded as **[0 0 1]** vector of size 3.

**One-hot encoding on iris dataset:** For iris dataset the target column which is Species. It contains three species Iris-setosa, Iris-versicolor, Iris-virginica.

## Sklearn Functions for One-hot Encoding:

* **sklearn.preprocessing.OneHotEncoder():** Encode categorical integer features using a one-hot aka one-of-K scheme

## Algorithm:

**Step 1 :** Import pandas and sklearn library for preprocessing

from sklearn import preprocessing

**Step 2:** Load the iris dataset in dataframe object df

**Step 3:** Observe the unique values for the Species column.

df['Species'].unique()

**output: array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)**

**Step 4:** Apply label\_encoder object for label encoding the Observe the unique values for the Species column.

## df['Species'].unique()

**Output: array([0, 1, 2], dtype=int64)**

**Step 5:** Remove the target variable from dataset

**features\_df=df.drop(columns=['Species'])**

**Step 6:** Apply one\_hot encoder for Species column.

enc = preprocessing.OneHotEncoder() enc\_df=pd.DataFrame(enc.fit\_transform(df[['Species']])).toarray()

**Step 7:** Join the encoded values with Features variable

df\_encode = features\_df.join(enc\_df)

**Step 8**: Observe the merge dataframe

df\_encode

**Step 9:** Rename the newly encoded columns.

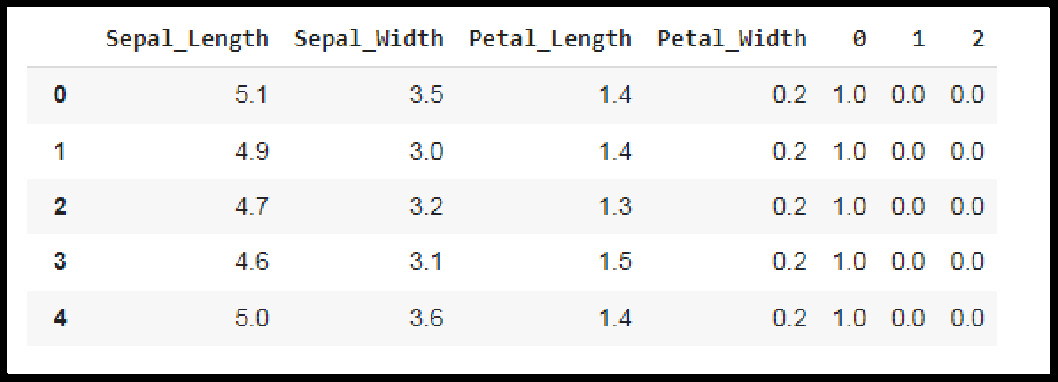
df\_encode.rename(columns = {0:'Iris-Setosa',

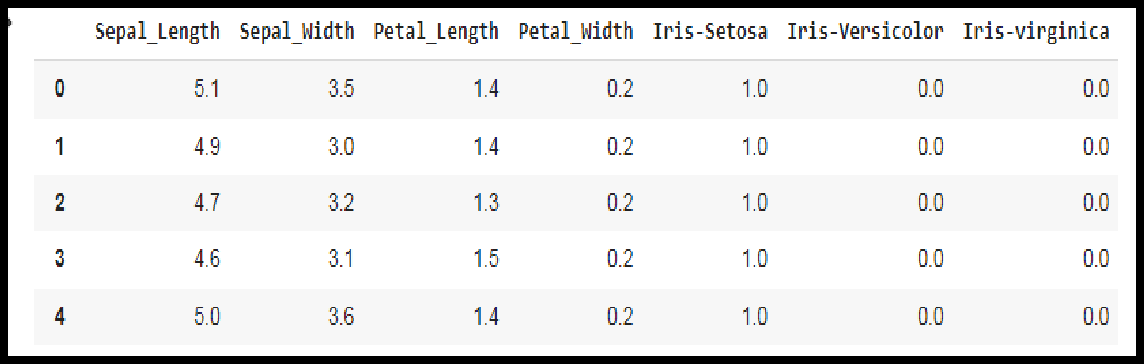
1:'Iris-Versicolor',2:'Iris-virginica'}, inplace = True)

**Step 10:** Observe the merge dataframe

df\_encode

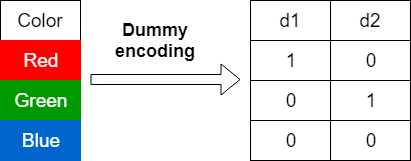
## Output after Step 8:



**Output after Step 10:**

## Dummy Variable Encoding

Dummy encoding also uses dummy (binary) variables. Instead of creating a number of dummy variables that is equal to the number of categories (k) in the variable, dummy encoding uses k-1 dummy variables. To encode the same Color variable with three categories using the dummy encoding, we need to use only two dummy variables.



In dummy encoding,

**“Red”** color is encoded as **[1 0]** vector of size 2. **“Green”** color is encoded as **[0 1]** vector of size 2. **“Blue”** color is encoded as **[0 0]** vector of size 2.

Dummy encoding removes a duplicate category present in the one-hot encoding.

## Pandas Functions for One-hot Encoding with dummy variables:

* **pandas.get\_dummies(data, prefix=None, prefix\_sep='\_', dummy\_na=False, columns=None, sparse=False, drop\_first=False, dtype=None):** Convert categorical variable into dummy/indicator variables.

## Parameters:

**data:**array-like, Series, or DataFrame Data of which to get dummy indicators.

**prefixstr**: list of str, or dict of str, default None String to append DataFrame column names. **prefix\_sep**: str, default ‘\_’

If appending prefix, separator/delimiter to use. Or pass a list or dictionary as with prefix.

**dummy\_nabool:** default False

Add a column to indicate NaNs, if False NaNs are ignored.

**columns:** list:like, default None

Column names in the DataFrame to be encoded. If columns is None then all the columns with object or category dtype will be converted.

## sparse: bool: default False

Whether the dummy-encoded columns should be backed by a SparseArray ( True) or a regular NumPy array (False).

## drop\_first:bool, default False

Whether to get k-1 dummies out of k categorical levels by removing the first level.

**dtype**: dtype, default np.uint8

Data type for new columns. Only a single dtype is allowed.

* **Return :** DataFrame with Dummy-coded data.

## Algorithm:

**Step 1 :** Import pandas and sklearn library for preprocessing

from sklearn import preprocessing

**Step 2:** Load the iris dataset in dataframe object df

**Step 3:** Observe the unique values for the Species column.

df['Species'].unique()

**output: array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)**

**Step 4:** Apply label\_encoder object for label encoding the Observe the unique

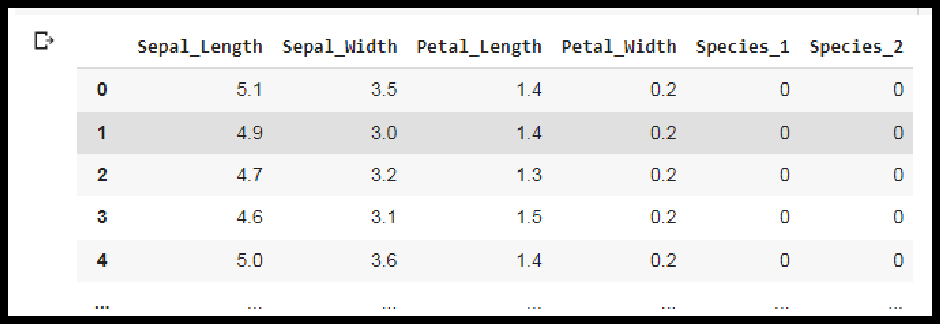
values for the Species column.

df['Species'].unique()

Output: array([0, 1, 2], dtype=int64)

**Step 6:** Apply one\_hot encoder with dummy variables for Species column.

one\_hot\_df = pd.get\_dummies(df, prefix="Species", columns=['Species'], drop\_first=True)

**Step 7**: Observe the merge dataframe one\_hot\_df

**Conclusion-**

In this way we have explored the functions of the python library for Data Preprocessing, Data Wrangling Techniques and How to Handle missing values on Iris Dataset.

**Assignment No: 2**

# Title of the Assignment: Data Wrangling, II

Create an “Academic performance” dataset of students and perform the following operations using Python.

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.

Reason and document your approach properly.

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

**Prerequisite:**

1. Basic of Python Programming
2. Concept of Data Preprocessing, Data Formatting , Data Normalization and Data Cleaning.

# -------------------------------------------------------------------------------------------

**Contents for Theory:**

# Creation of Dataset using Microsoft Excel.

1. **Identification and Handling of Null Values**

# Identification and Handling of Outliers

1. **Data Transformation for the purpose of :**

# To change the scale for better understanding

* 1. **To decrease the skewness and convert distribution into normal distribution**

# -------------------------------------------------------------------------------------------

**Theory:**

# Creation of Dataset using Microsoft Excel.

The dataset is created in “CSV” format.

* + The name of dataset is **StudentsPerformance**
  + **The features of the dataset are:** Math\_Score, Reading\_Score, Writing\_Score, Placement\_Score, Club\_Join\_Date .
  + **Number of Instance**s: 30
  + **The response variable is**: Placement\_Offer\_Count .

# Range of Values:

Math\_Score [60-80], Reading\_Score[75-,95], ,Writing\_Score [60,80],

Placement\_Score[75-100], Club\_Join\_Date [2018-2021].

* + **The response variable is** the number of placement offers facilitated to particular students, which is largely depend on Placement\_Score

To fill the values in the dataset the **RANDBETWEEN** is used. Returns a random integer number between the numbers you specify

**Syntax : RANDBETWEEN(bottom, top) Bottom** The smallest integer and

**Top** The largest integer RANDBETWEEN will return.

For better understanding and visualization, 20% impurities are added into each variable to the dataset.

The step to create the dataset are as follows:

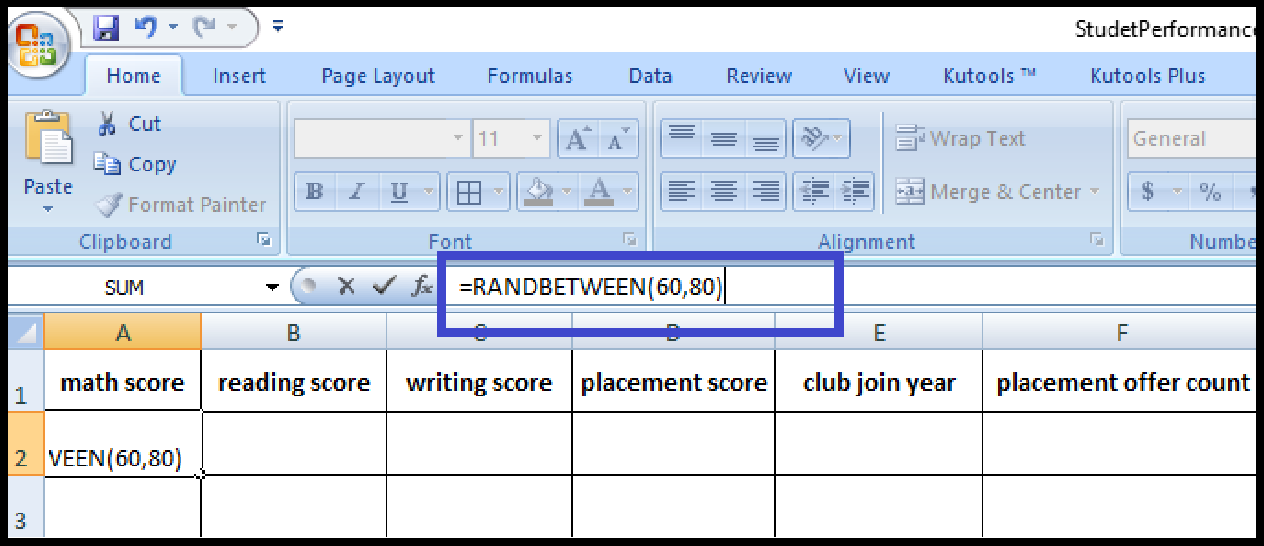
**Step 1:** Open Microsoft Excel and click on Save As. Select Other .Formats

**Step 2:** Enter the name of the dataset and Save the dataset astye CSV(MS-DOS).

**Step 3:** Enter the name of features as column header.

**Step 3:** Fill the dara by using **RANDOMBETWEEN** function. For every feature , fill the data by considering above spectified range.

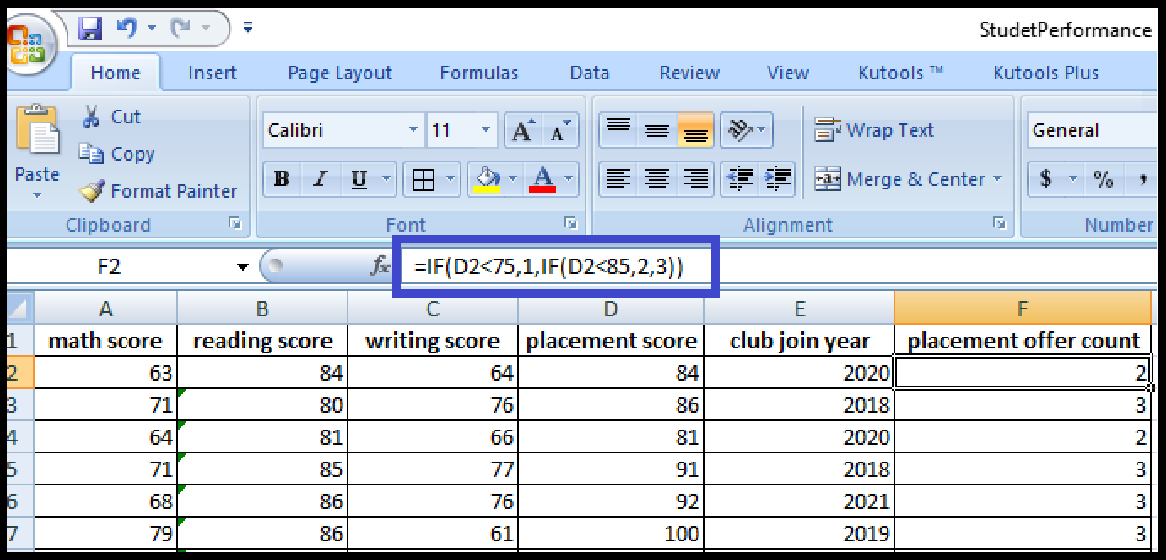
one example is given:

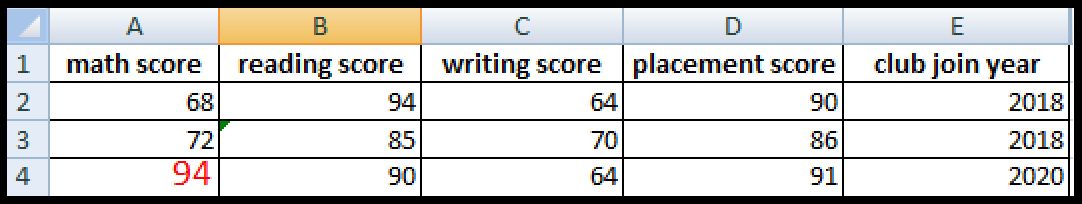


Scroll down the cursor for 30 rows to create 30 instances.

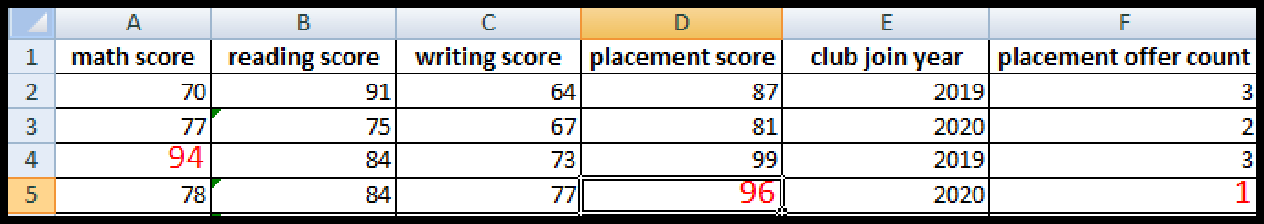
Repeat this for the features, Reading\_Score, Writing\_Score, Placement\_Score, Club\_Join\_Date.

The placement count largely depends on the placement score. It is considered that if placement score <75, 1 offer is facilitated; for placement score >75 , 2 offer is facilitated and for else (>85) 3 offer is facilitated. Nested If formula is used for ease of data filling.

 **Step 4:** In 20% data, fill the impurities. The range of math score is [60,80], updating a few instances values below 60 or above 80. Repeat this for Writing\_Score [60,80], Placement\_Score[75-100], Club\_Join\_Date [2018-2021].



**Step 5:** To violate the ruleof response variable, update few valus . If placement scoreis greater then 85, facilated only 1 offer.



The dataset is created with the given description.

# Identification and Handling of Null Values

Missing Data can occur when no information is provided for one or more items or for a whole unit. Missing Data is a very big problem in real-life scenarios. Missing Data can also refer to as NA(Not Available) values in pandas. In DataFrame sometimes many datasets simply arrive with missing data, either because it exists and was not collected or it never existed. For Example, Suppose different users being surveyed may choose not to share their income, some users may choose not to share the address in this way many datasets went missing.

In Pandas missing data is represented by two value:

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

Pandas treat None and NaN as essentially interchangeable for indicating missing or null values. To facilitate this convention, there are several useful functions for detecting, removing, and replacing null values in Pandas DataFrame :

* + isnull()
  + notnull()
  + dropna()
  + fillna()
  + replace()

# Checking for missing values using isnull() and notnull()

* + **Checking for missing values using isnull()**

In order to check null values in Pandas DataFrame, isnull() function is used. This function return dataframe of Boolean values which are True for NaN values.

# Algorithm:

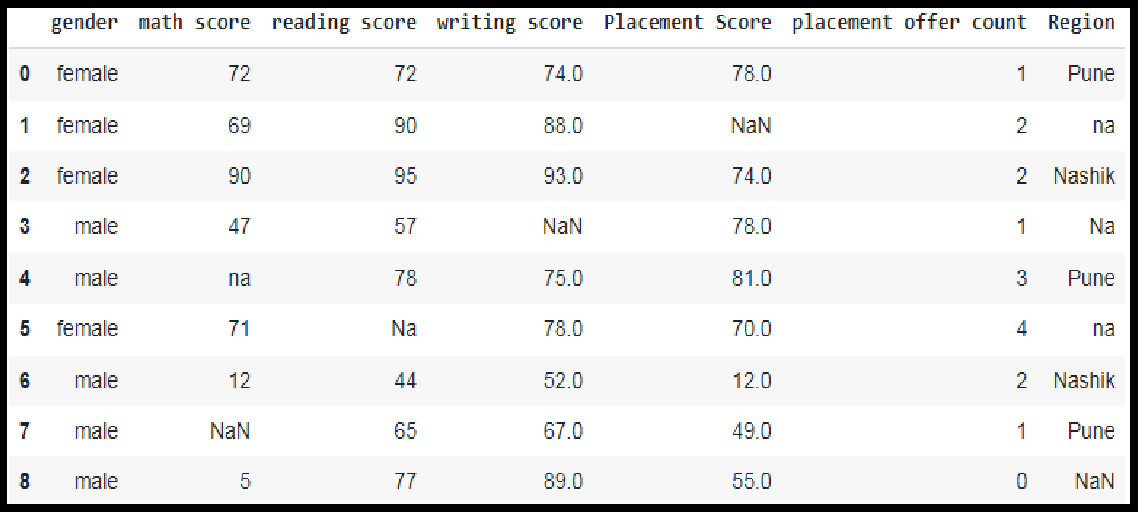
**Step 1 :** Import pandas and numpy in order to check missing values in Pandas DataFrame

import pandas as pd

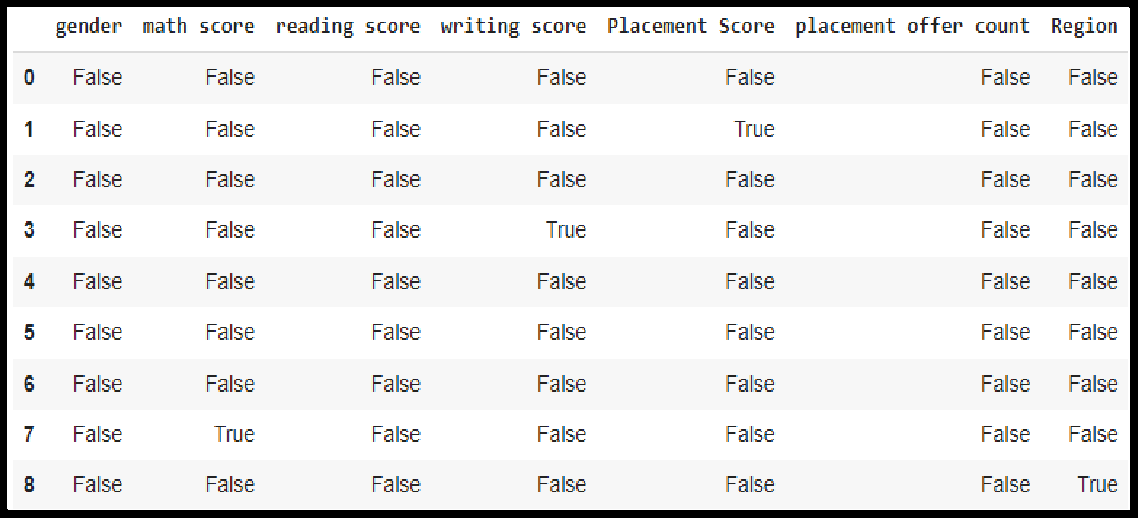
import numpy as np

**Step 2:** Load the dataset in dataframe object df

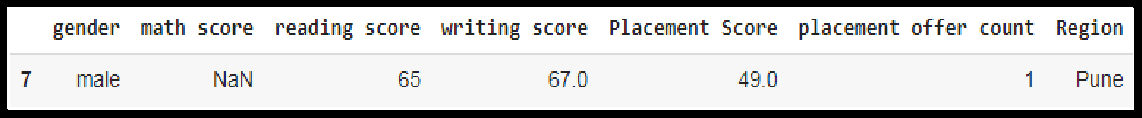
df=pd.read\_csv("/content/StudentsPerformanceTest1.csv")

**Step 3:** Display the data frame **df**

**Step 4:** Use isnull() function to check null values in the dataset.

 df.isnull()

**Step 5:** To create a series true for NaN values for specific columns. for example math score in dataset and display data with only math score as NaN

series = pd.isnull(df["math score"]) df[series]

# Checking for missing values using notnull()

In order to check null values in Pandas Dataframe, notnull() function is used. This function return dataframe of Boolean values which are False for NaN values.

# Algorithm:

**Step 1 :** Import pandas and numpy in order to check missing values in Pandas DataFrame

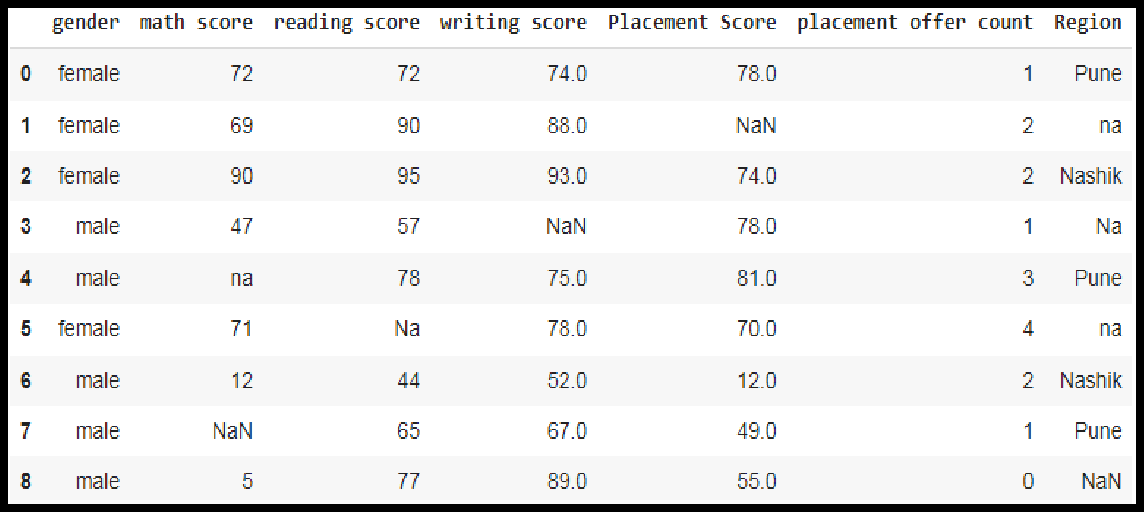
import pandas as pd

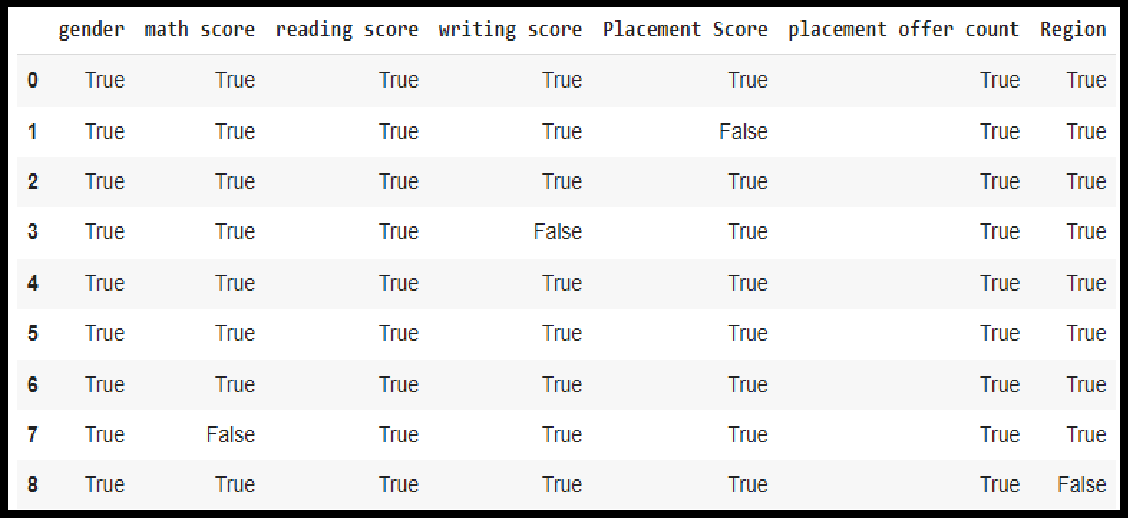
import numpy as np

**Step 2:** Load the dataset in dataframe object df

df=pd.read\_csv("/content/StudentsPerformanceTest1.csv")

**Step 3:** Display the data frame **df**



**Step 4:** Use notnull() function to check null values in the dataset.

df.notnull()

# Filling missing values using dropna(), fillna(), replace()

In order to fill null values in a datasets, fillna(), replace() functions are used. These functions replace NaN values with some value of their own. All these functions help in filling null values in datasets of a DataFrame.

# Filling null values with a single value

**Step 1 :** Import pandas and numpy in order to check missing values in Pandas DataFrame

import pandas as pd

import numpy as np

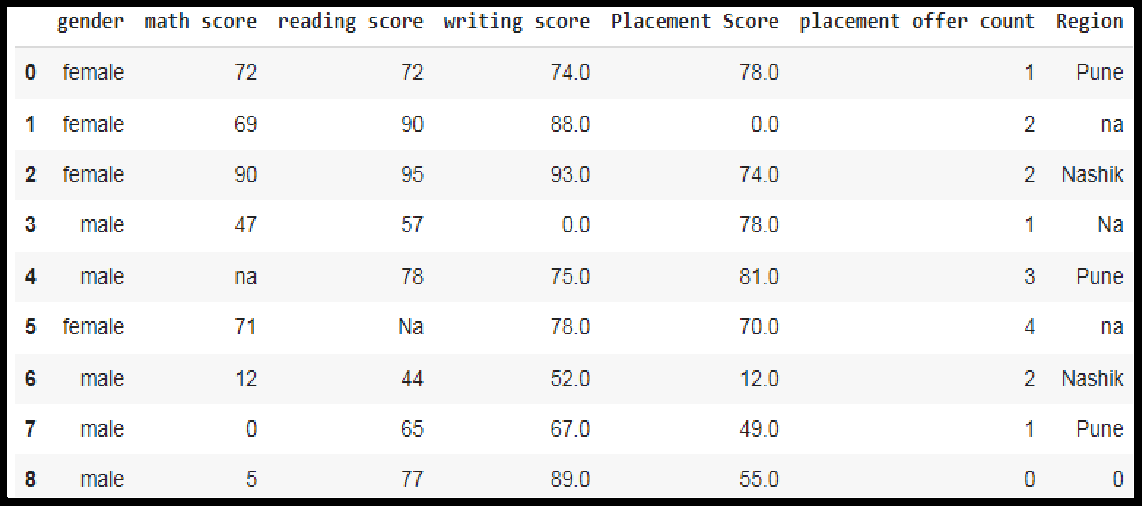
**Step 2:** Load the dataset in dataframe object df

df=pd.read\_csv("/content/StudentsPerformanceTest1.csv")

**Step 3:** Display the data frame

**df**

**Step 4:** filling missing value using fillna()

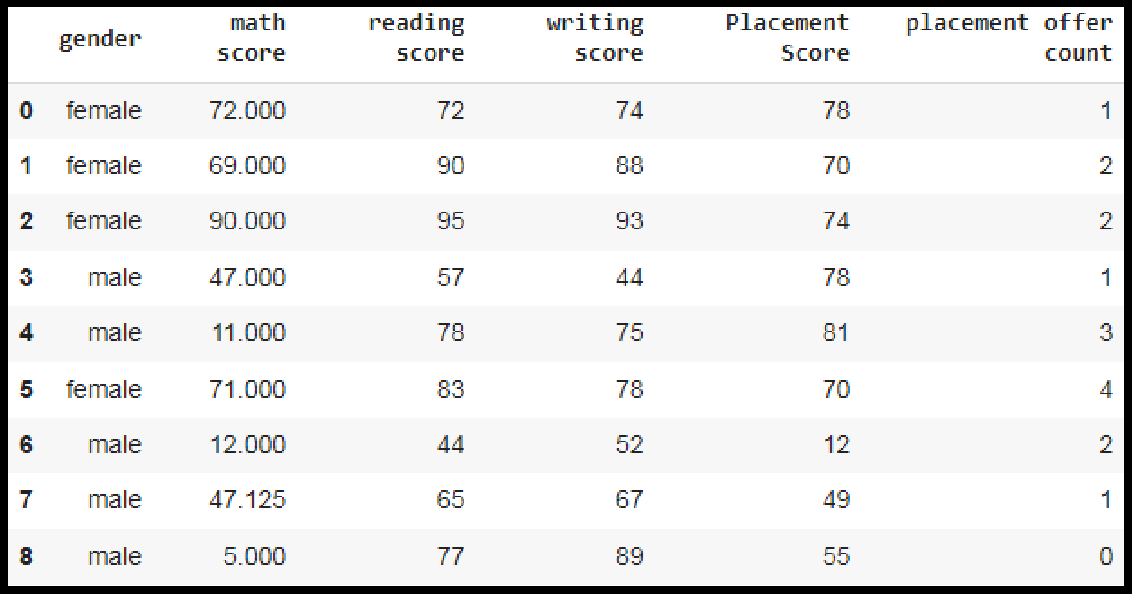
ndf=df ndf.fillna(0)

# Filling null values in dataset

To fill null values in dataset use inplace=true

m\_v=df['math score'].mean()

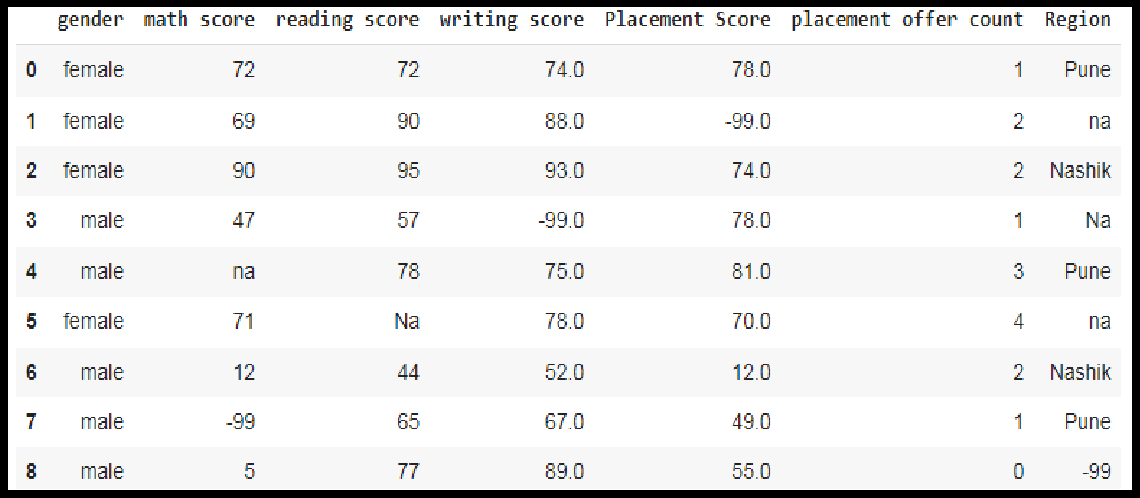
df['math score'].fillna(value=m\_v, inplace=True) df



# Filling a null values using replace() method

Following line will replace Nan value in dataframe with value -99

ndf.replace(to\_replace = np.nan, value = -99)



# Deleting null values using dropna() method

In order to drop null values from a dataframe, dropna() function is used. This function drops Rows/Columns of datasets with Null values in different ways.

1. Dropping rows with at least 1 null value
2. Dropping rows if all values in that row are missing
3. Dropping columns with at least 1 null value.
4. Dropping Rows with at least 1 null value in CSV file

# Algorithm:

**Step 1 :** Import pandas and numpy in order to check missing values in Pandas DataFrame

import pandas as pd import numpy as np

**Step 2:** Load the dataset in dataframe object df

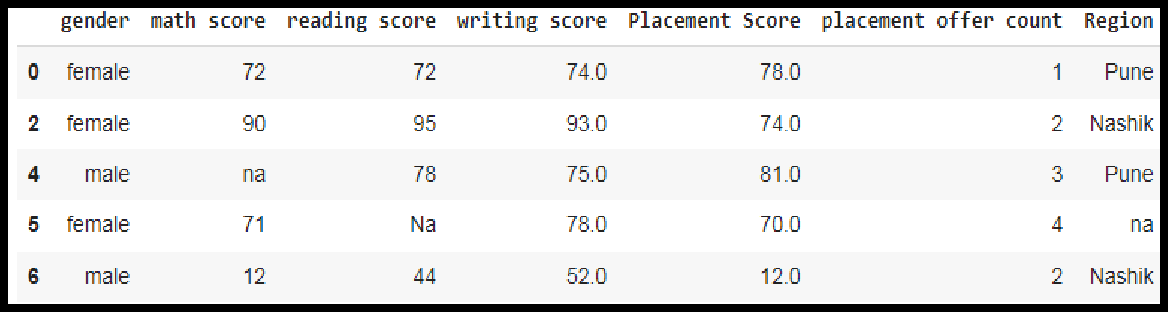
df=pd.read\_csv("/content/StudentsPerformanceTest1.csv")

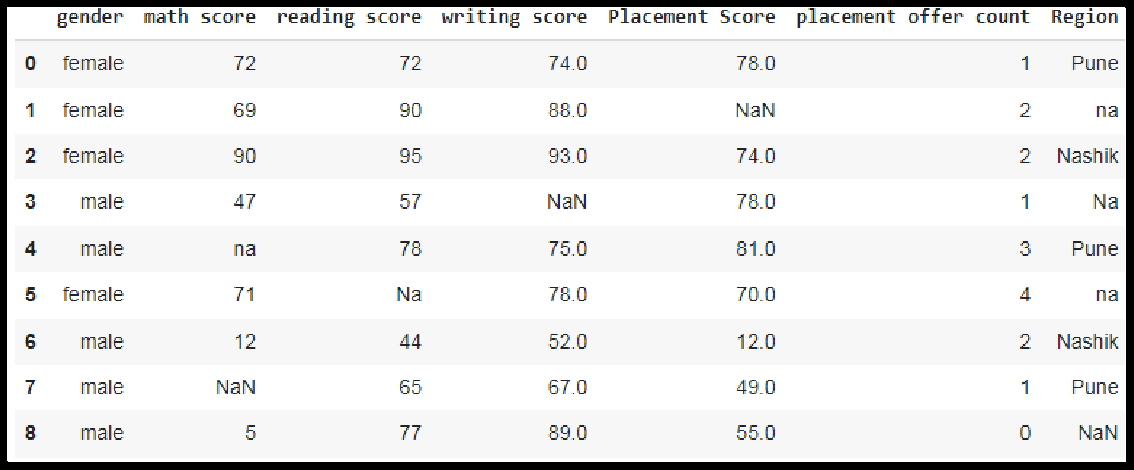
**Step 3:** Display the data frame

**df**

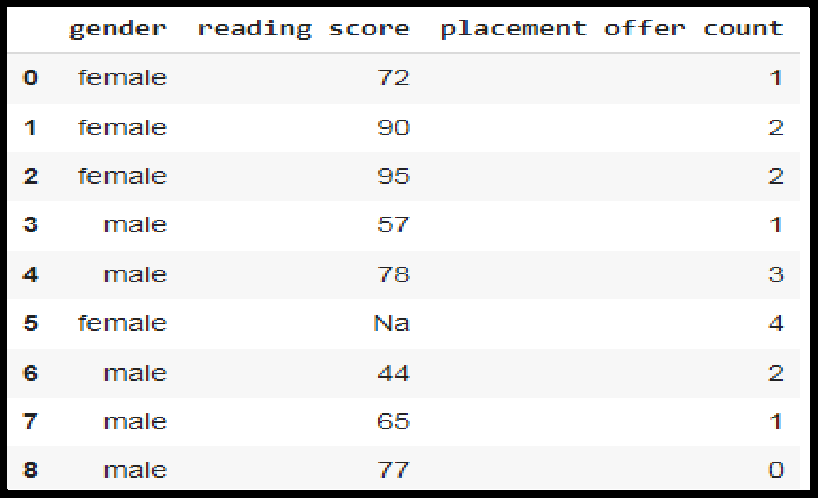
**Step 4:To** drop rows with at least 1 null value

**ndf.dropna()**



**Step 5:** To Drop rows if all values in that row are missing

ndf.dropna(how = 'all')

**Step 6:** To Drop columns with at least 1 null value.ndf.dropna(axis = 1)

**Step 7 :** To drop rows with at least 1 null value in CSV file. making new data frame with dropped NA values

new\_data = ndf.dropna(axis = 0, how ='any')

new\_data

# Identification and Handling of Outliers

* 1. **Identification of Outliers**

One of the most important steps as part of data preprocessing is detecting and treating the outliers as they can negatively affect the statistical analysis and the training process of a machine learning algorithm resulting in lower accuracy.

# What are Outliers?

We all have heard of the idiom ‘odd one out' which means something unusual in comparison to the others in a group.

Similarly, an Outlier is an observation in a given dataset that lies far from the rest of the observations. That means an outlier is vastly larger or smaller than the remaining values in the set.

# Why do they occur?

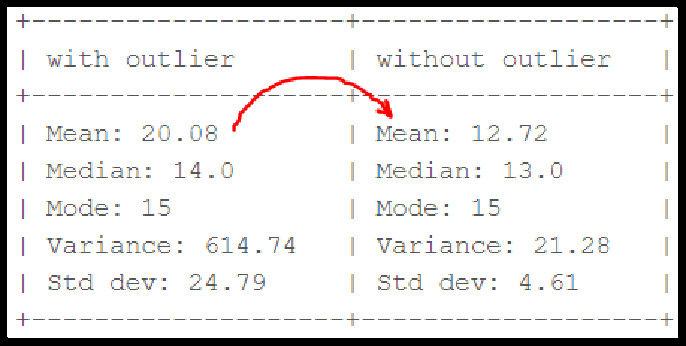
An outlier may occur due to the variability in the data, or due to experimental error/human error.

They may indicate an experimental error or heavy skewness in the data(heavy-tailed distribution).

‘Mean’ is the only measure of central tendency that is affected by the outliers which in turn impacts Standard deviation.

Example:

Consider a small dataset, sample= [15, 101, 18, 7, 13, 16, 11, 21, 5, 15, 10, 9]. By looking at it, one can quickly say ‘101’ is an outlier that is much larger than the other values.



*fig. Computation with and without outlier*

From the above calculations, we can clearly say the Mean is more affected than the Median.

# Detecting Outliers

If our dataset is small, we can detect the outlier by just looking at the dataset. But what if we have a huge dataset, how do we identify the outliers then? We need to use visualization and mathematical techniques.

Below are some of the techniques of detecting outliers

* + Boxplots
  + Scatterplots
  + Z-score
  + Inter Quantile Range(IQR)

# Detecting outliers using Boxplot:

It captures the summary of the data effectively and efficiently with only a simple box and whiskers. Boxplot summarizes sample data using 25th, 50th, and 75th percentiles. One can just get insights(quartiles, median, and outliers) into the dataset by just looking at its boxplot.

# Algorithm:

**Step 1 :** Import pandas and numpy libraries

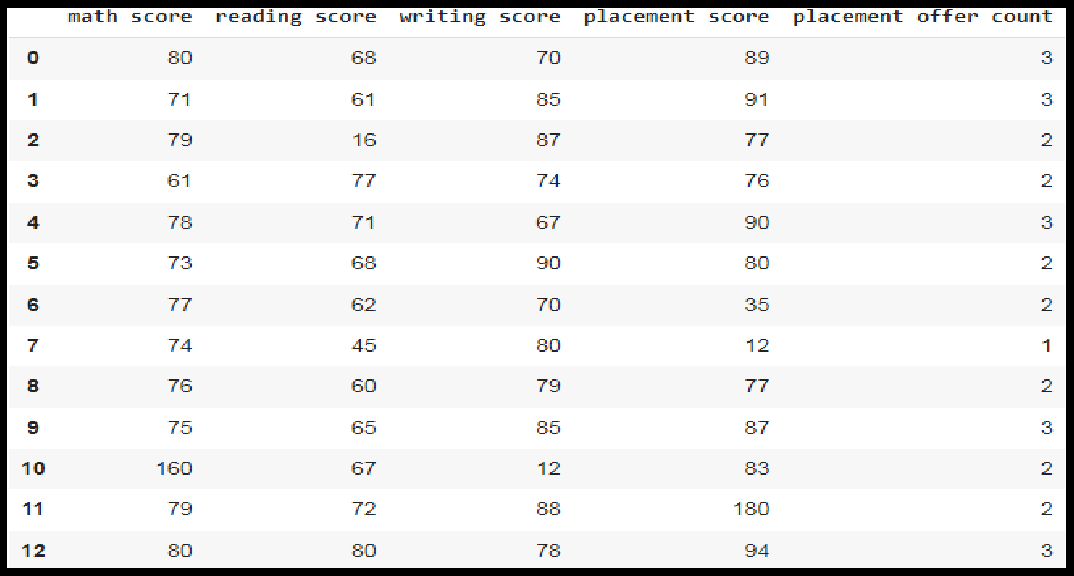
import pandas as pd

import numpy as np

**Step 2:** Load the dataset in dataframe object df

df=pd.read\_csv("/content/demo.csv")

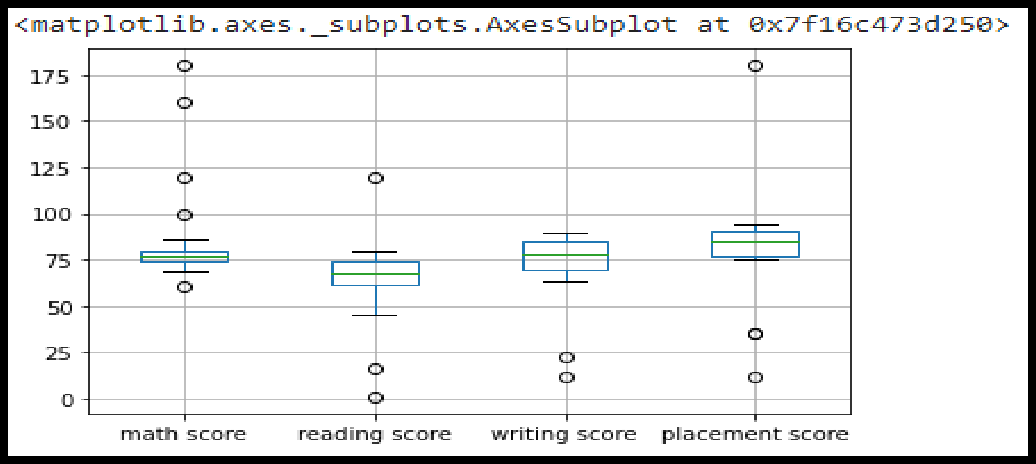
**Step 3:** Display the data frame **df**



**Step 4:Select the columns for boxplot and draw the boxplot.**

**col = ['math score', 'reading score' , 'writing score','placement score']**

**df.boxplot(col)**

**Step 5:** We can now print the outliers for each column with reference to the box plot.

print(np.where(df['math score']>90))

print(np.where(df['reading score']<25)) print(np.where(df['writing score']<30))

# Detecting outliers 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. In the process of utilizing the scatter plot, one can also use it for outlier detection.

To plot the scatter plot one requires two variables that are somehow related to each other. So here Placement score and Placement count features are used.

# Algorithm:

**Step 1 :** Import pandas , numpy and matplotlib libraries

import pandas as pd import numpy as np

import matplotlib.pyplot as plt

**Step 2:** Load the dataset in dataframe object df

df=pd.read\_csv("/content/demo.csv")

**Step 3:** Display the data frame **df**

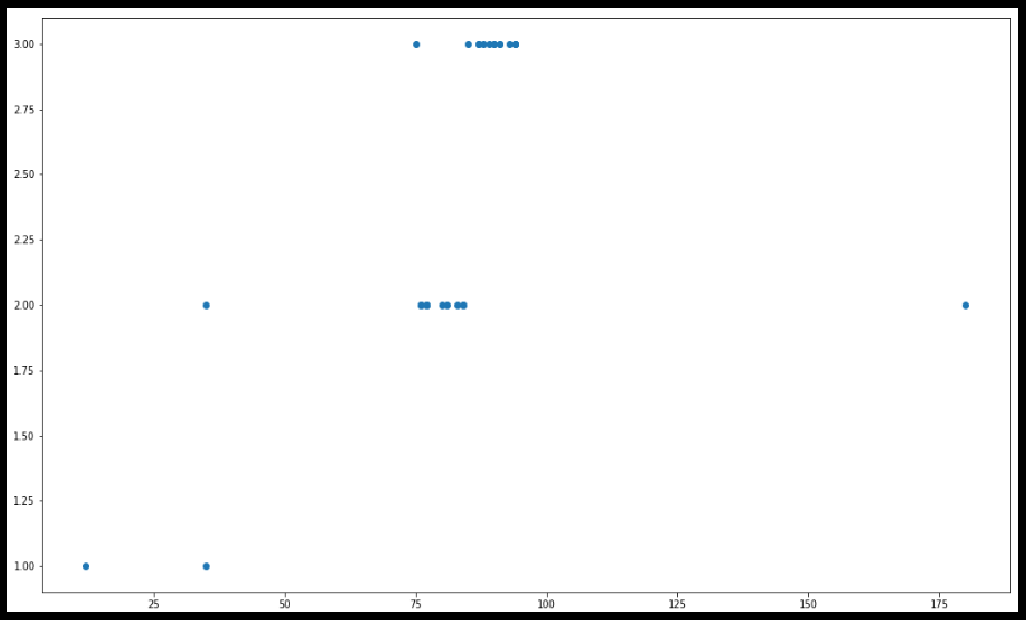
**Step 4:** Draw the scatter plot with placement score and placement offer count **fig, ax = plt.subplots(figsize = (18,10)) ax.scatter(df['placement score'], df['placement offer count'])**

**plt.show()**

Labels to the axis can be assigned (Optional)

**ax.set\_xlabel('(Proportion non-retail business acres)/(town)')**

**ax.set\_ylabel('(Full-value property-tax rate)/($10,000)')**



**Step 5:** We can now print the outliers with reference to scatter plot.

print(np.where((df['placement score']<50) & (df['placement offer count']>1)))

print(np.where((df['placement score']>85) & (df['placement offer count']<3)))

# Detecting outliers using Z-Score:

Z-Score is also called a standard score. This value/score helps to understand 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

# Algorithm:

**Step 1 :** Import numpy and stats from scipy libraries

import numpy as np

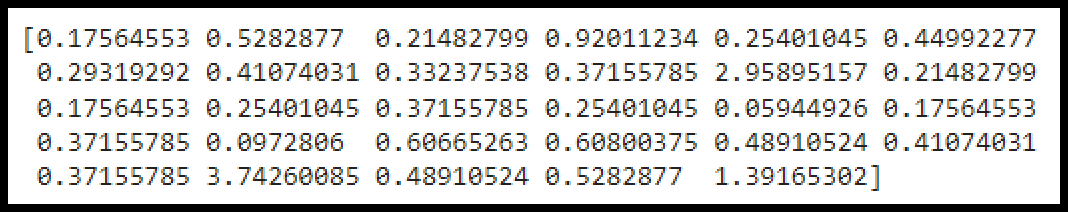
from scipy import stats

**Step 2:** Calculate Z-Score for mathscore column

z = np.abs(stats.zscore(df['math score']))

**Step 3:** Print Z-Score Value. It prints the z-score values of each data item of the column

**print(z)**

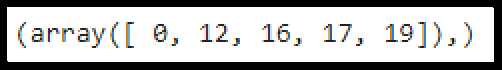


**Step 4:** Now to define an outlier threshold value is chosen.

threshold = 0.18

**Step 5:** Display the sample outliers

sample\_outliers = np.where(z <threshold) sample\_outliers



# Detecting outliers using Inter Quantile Range(IQR):

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

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

In the above formula as according to statistics, the 0.5 scale-up of IQR (new\_IQR = IQR + 0.5\*IQR) is taken.

# Algorithm:

**Step 1 :** Import numpy library

import numpy as np

**Step 2:** Sort Reading Score feature and store it into sorted\_rscore.

sorted\_rscore= sorted(df['reading score'])

**Step 3:** Print sorted\_rscore

sorted\_rscore

**Step 4:** Calculate and print Quartile 1 and Quartile 3

q1 = np.percentile(sorted\_rscore, 25) q3 = np.percentile(sorted\_rscore, 75) print(q1,q3)



**Step 5:** Calculate value of IQR (Inter Quartile Range)

IQR = q3-q1

**Step 6:** Calculate and print Upper and Lower Bound to define the outlier base value.

lwr\_bound = q1-(1.5\*IQR) upr\_bound = q3+(1.5\*IQR) print(lwr\_bound, upr\_bound)



**Step 7:** Print Outliers

r\_outliers = []

for i in sorted\_rscore:

if (i<lwr\_bound or i>upr\_bound): r\_outliers.append(i)

print(r\_outliers)

# Handling of 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.

Below are some of the methods of treating the outliers

* Trimming/removing the outlier
* Quantile based flooring and capping
* Mean/Median imputation

# Trimming/removing the outlier:

In this technique, we remove the outliers from the dataset. Although it is not a good practice to follow.

new\_df=df

for i in sample\_outliers: new\_df.drop(i,inplace=True)

new\_df

Here Sample\_outliers are So instances with index 0, 12 ,16 and 17 are deleted.

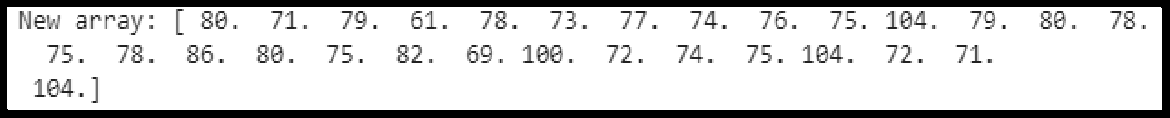
# Quantile based flooring and capping:

In this technique, the outlier is capped at a certain value above the 90th percentile value or floored at a factor below the 10th percentile value **df=pd.read\_csv("/demo.csv")**

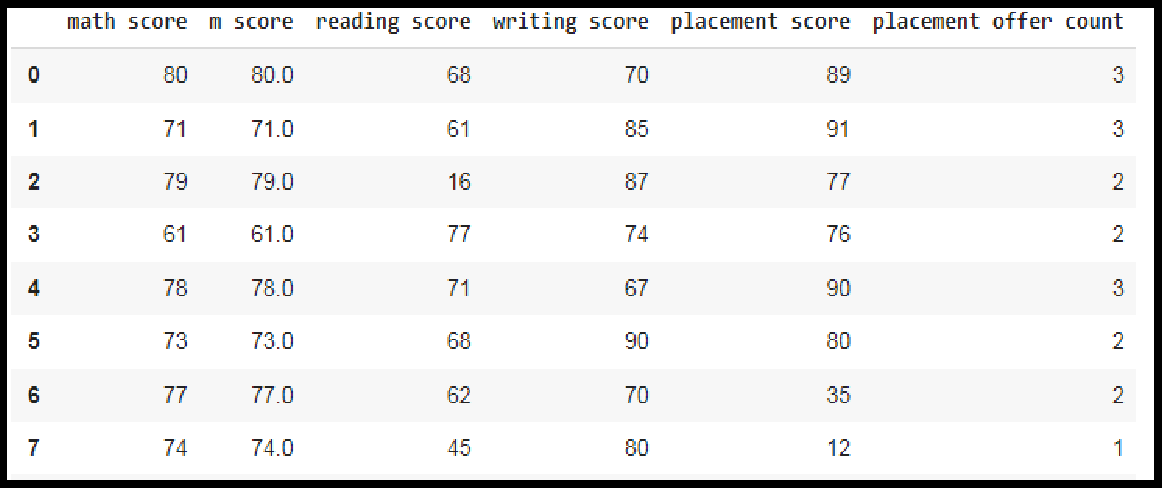
**df\_stud=df**

**ninetieth\_percentile = np.percentile(df\_stud['math score'], 90)**

**b = np.where(df\_stud['math score']>ninetieth\_percentile, ninetieth\_percentile, df\_stud['math score'])**

**print("New array:",b)**

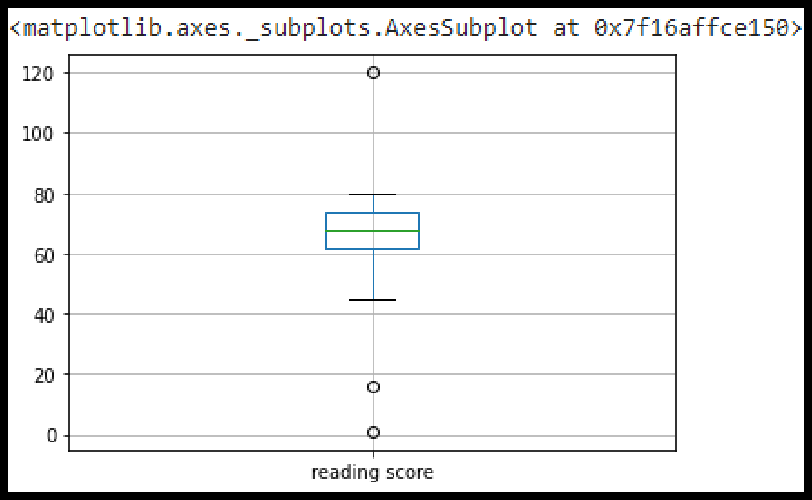
**df\_stud.insert(1,"m score",b,True) df\_stud**



# Mean/Median imputation:

As the mean value is highly influenced by the outliers, it is advised to replace the outliers with the median value.

* + 1. Plot the box plot for reading score col = ['reading score'] df.boxplot(col)



* + 1. Outliers are seen in box plot.
    2. Calculate the median of reading score by using sorted\_rscore

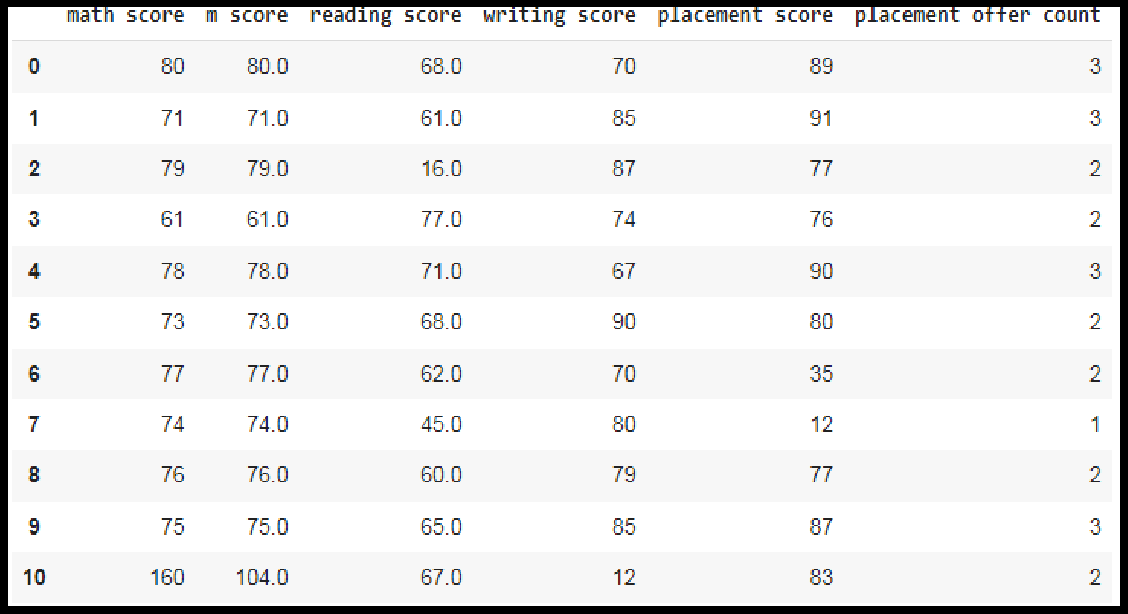
median=np.median(sorted\_rscore) median

* + 1. Replace the upper bound outliers using median value

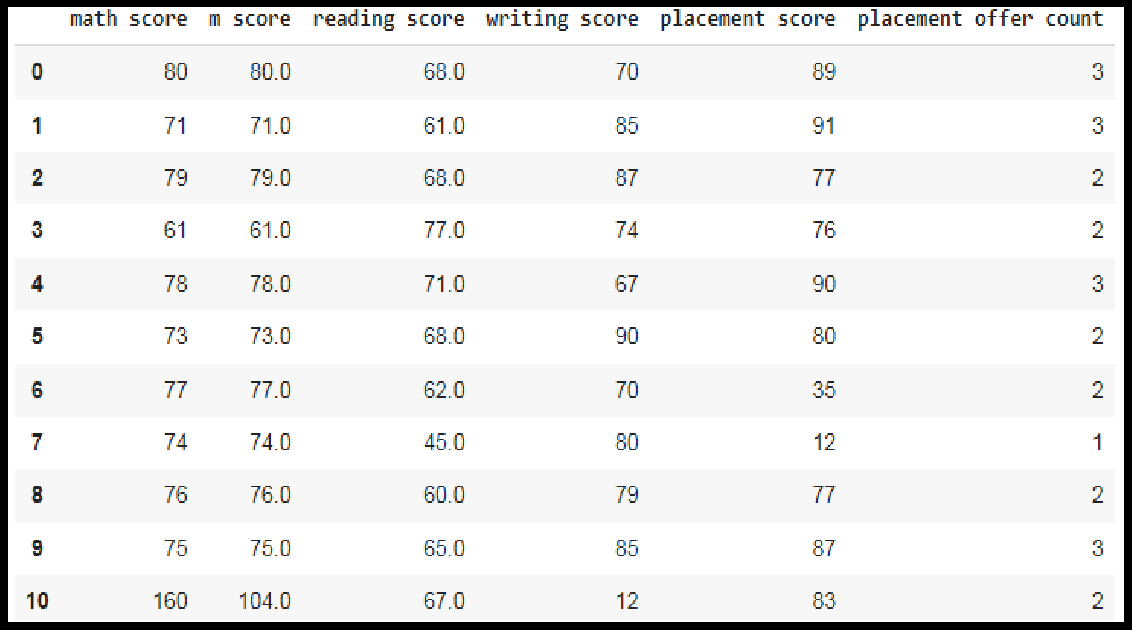
refined\_df=df

refined\_df['reading score'] = np.where(refined\_df['reading score'] >upr\_bound, median,refined\_df['reading score'])

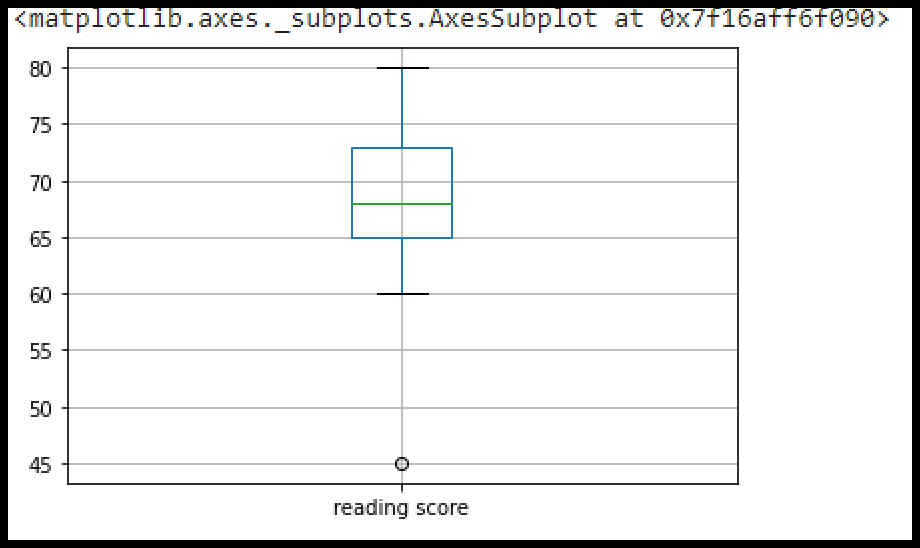
* + 1. Display redefined\_df



* + 1. Replace the lower bound outliers using median value refined\_df['reading score'] = np.where(refined\_df['reading score'] <lwr\_bound, median,refined\_df['reading score'])
    2. Display redefined\_df



Draw the box plot for redefined\_df col = ['reading score'] refined\_df.boxplot(col)



# Data Transformation for the purpose of :

Data transformation is the process of converting raw data into a format or structure that would be more suitable for model building and also data discovery in general.The process of data transformation can also be referred to as extract/transform/load (ETL). The extraction phase involves identifying and pulling data from the various source systems that create data and then moving the data to a single repository. Next, the raw data is cleansed, if needed. It's then transformed into a target format that can be fed into operational systems or into a data warehouse, a date lake or another repository for use in business intelligence and analytics applications. The transformation The data are transformed in ways that are ideal for mining the data. The data transformation involves steps that are.

* + **Smoothing:** It is a process that is used to remove noise from the dataset using some algorithms It allows for highlighting important features present in the dataset. It helps in predicting the patterns
  + **Aggregation**: Data collection or aggregation is the method of storing and presenting data in a summary format. The data may be obtained from multiple data sources to integrate these data sources into a data analysis description. This is a crucial step since the accuracy of data analysis insights is highly dependent on the quantity and quality of the data used.
  + **Generalization**:It converts low-level data attributes to high-level data attributes

using concept hierarchy. For Example Age initially in Numerical form (22, 25) is converted into categorical value (young, old).

* + **Normalization:** Data normalization involves converting all data variables into a

given range. Some of the techniques that are used for accomplishing normalization are:

* + - **Min–max normalization**: This transforms the original data linearly.
    - **Z-score normalization**: In z-score normalization (or zero-mean normalization) the values of an attribute (A), are normalized based on the mean of A and its standard deviation.
    - **Normalization by decimal scaling**: It normalizes the values of an attribute by changing the position of their decimal points

# Attribute or feature construction.

* + - **New attributes constructed from the given ones**: Where new attributes are created & applied to assist the mining process from the given set of attributes. This simplifies the original data & makes the mining more efficient.

In this assignment , The purpose of this transformation should be one of the following reasons:

# To change the scale for better understanding (Attribute or feature construction)

Here the Club\_Join\_Date is transferred to Duration.

# Algorithm:

**Step 1 :** Import pandas and numpy libraries

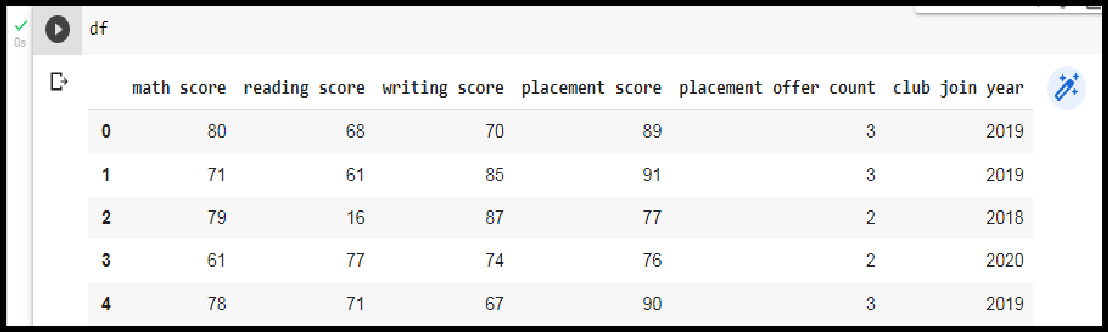
import pandas as pd

import numpy as np

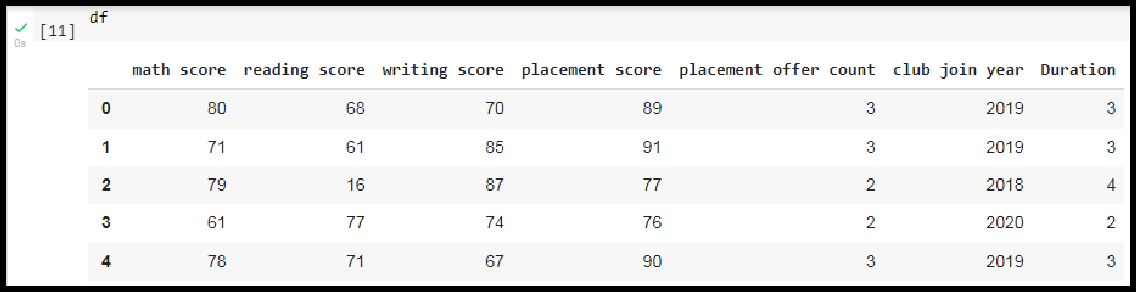
**Step 2:** Load the dataset in dataframe object df

df=pd.read\_csv("/content/demo.csv")

**Step 3:** Display the data frame

df

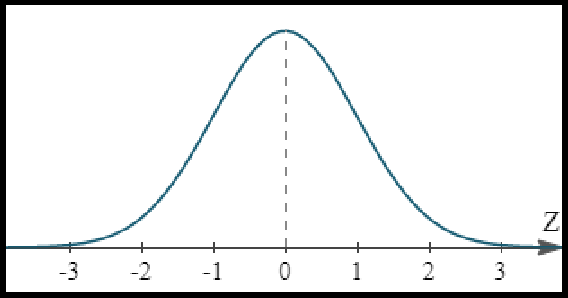
**Step 3:** Change the scale of Joining year to duration.



# To decrease the skewness and convert distribution into normal distribution (Normalization by decimal scaling)

**Data Skewness**: It is asymmetry in a statistical distribution, in which the curve appears distorted or skewed either to the left or to the right. Skewness can be quantified to define the extent to which a distribution differs from a normal distribution.

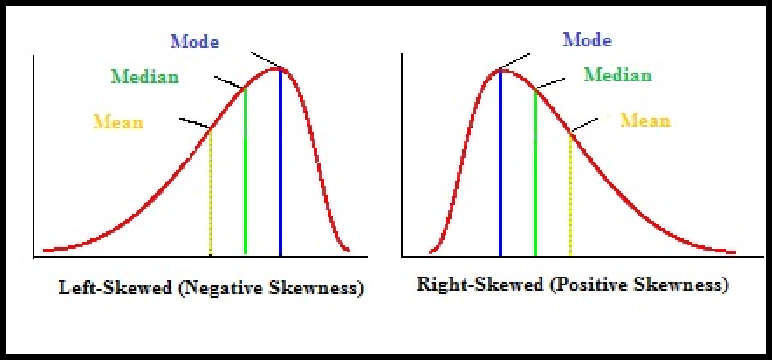
**Normal Distribution:** In a normal distribution, the graph appears as a classical, symmetrical “bell-shaped curve.” The mean, or average, and the mode, or maximum point on the curve, are equal.



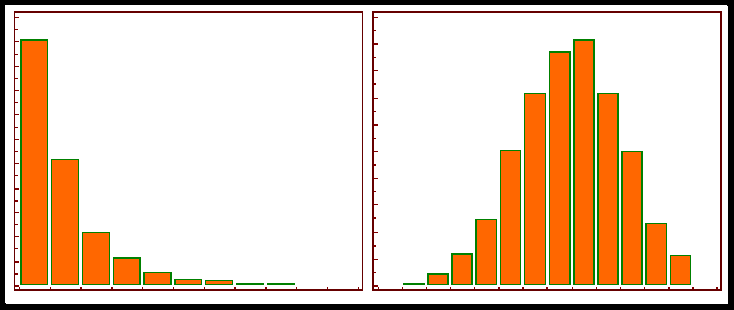
Positively Skewed Distribution

**A positively skewed distribution** means that the extreme data results are larger. This skews the data in that it brings the mean (average) up. The mean will be larger than the median in a Positively skewed distribution.

**A negatively skewed distribution** means the opposite: that the extreme data results are smaller. This means that the mean is brought down, and the median is larger than the mean in a negatively skewed distribution.



**Reducing skewness** A data transformation may be used to reduce skewness. A distribution that is symmetric or nearly so is often easier to handle and interpret than a skewed distribution. The logarithm, x to log base 10 of x, or x to log base e of x (ln x), or x to log base 2 of x, is a strong transformation with a major effect on distribution shape. It is commonly used for reducing right skewness and is often appropriate for measured variables. It can not be applied to zero or negative values.



# Algorithm:

**Step 1 :** Detecting outliers using Z-Score for the Math\_score variable and remove the outliers.

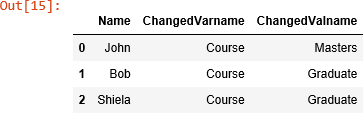
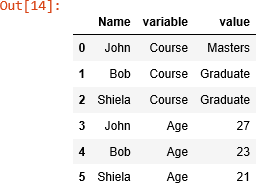
**Step 2**:Observe the histogram for math\_score variable. import matplotlib.pyplot as plt new\_df['math score'].plot(kind = 'hist')

**Step 3:**Convert the variables to logarithm at the scale 10.

df['log\_math'] = np.log10(df['math score'])

**Step 4:** Observe the histogram for math\_score variable.

df['log\_math'].plot(kind = 'hist')



It is observed that skewness is reduced at some level.

**Conclusion:** In this way we have explored the functions of the python library for Data Identifying and handling the outliers. Data Transformations Techniques are explored with the purpose of creating the new variable and reducing the skewness from datasets.

Assignment No:3

**Descriptive Statistics - Measures of Central Tendency and variability**

Perform the following operations on any open source dataset (e.g., data.csv)

1. 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. For example, if your categorical variable is age groups and quantitative variable is income, then provide summary statistics of income grouped by the age groups. Create a list that contains a numeric value for each response to the categorical variable.

2. Write a Python program to 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.

Provide the codes with outputs and explain everything that you do in this step.

Let's start by loading the required libraries and the data.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

df = pd.read\_csv(r"D:\dsbldlab\demo1.csv")

print(df.shape)

print(df.info())

Output:

1 (29, 6)

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 29 entries, 0 to 28

Data columns (total 6 columns):

# Column Non-Null Count Dtype

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

0 math score 29 non-null int64

1 reading score 29 non-null int64

2 writing score 29 non-null int64

3 placement score 29 non-null int64

4 placement offer count 29 non-null int64

5 club join year 29 non-null int64

dtypes: int64(6)

memory usage: 1.5 KB

None

**Measures of Central Tendency**

### Mean

Mean represents the arithmetic average of the data. The line of code below prints the mean of the numerical variables in the data

The command **df.mean(axis = 0)** will also give the same output.

1 df.mean()

Output:

1 math score 84.482759

reading score 65.896552

writing score 73.793103

placement score 82.724138

placement offer count 2.448276

club join year 2019.448276

dtype: float64

it is also possible to calculate the mean of a particular variable in a data, as shown below

print(df.loc[:,'math score'].mean())

Output:

84.48275862068965

### Median

In simple terms, median represents the 50th percentile, or the middle value of the data, that separates the distribution into two halves. The line of code below prints the median of the numerical variables in the data. The command **df.median(axis = 0)** will also give the same output.

1df.median()

python

Output:

1 math score 77.0

reading score 68.0

writing score 78.0

placement score 85.0

placement offer count 3.0

club join year 2019.0

dtype: float64

It is also possible to calculate the median of a particular variable in a data, as shown in the first two lines of code below. We can also calculate the median of the rows by specifying the **(axis = 1)** argument. The third line below calculates the median of the first five rows.

1#to calculate a median of a particular column

2print(df.loc[:,'Age'].median())

3print(df.loc[:,'Income'].median())

4

5df.median(axis = 1)[0:5]

Output:

1 77.0

0 75.0

1 78.0

2 78.0

3 75.0

4 74.5

dtype: float64

## Measures of Dispersion

In the previous sections, we have discussed the various measures of central tendency. However, as we have seen in the data, the values of these measures differ for many variables. This is because of the extent to which a distribution is stretched or squeezed. In statistics, this is measured by dispersion which is also referred to as variability, scatter, or spread. The most popular measures of dispersion are standard deviation, variance, and the interquartile range.

### Standard Deviation

Standard deviation is a measure that is used to quantify the amount of variation of a set of data values from its mean. A low standard deviation for a variable indicates that the data points tend to be close to its mean, and vice versa. The line of code below prints the standard deviation of all the numerical variables in the data.

1df.std()

python

Output:

math score 25.973366

reading score 20.098587

writing score 17.341897

placement score 26.891178

placement offer count 0.631676

club join year 0.985111

dtype: float64

It is also possible to calculate the standard deviation of a particular variable, as shown in the first two lines of code below. The third line calculates the standard deviation for the first five rows.

1print(df.loc[:,'math score'].std())

2#calculate the standard deviation of the first five rows

3df.std(axis = 1)[0:5]

Output:

25.973366426915053

1 799.477496

2 803.328430

3 801.493107

4 799.602401

dtype: float64

### Variance

Variance is another measure of dispersion. It is the square of the standard deviation and the covariance of the random variable with itself. The line of code below prints the variance of all the numerical variables in the dataset. The interpretation of the variance is similar to that of the standard deviation.

1df.var()

python

Output:

math score 674.615764

reading score 403.953202

writing score 300.741379

placement score 723.135468

placement offer count 0.399015

club join year 0.970443

dtype: float64

### Skewness

Another useful statistic is skewness, which is the measure of the symmetry, or lack of it, for a real-valued random variable about its mean. The skewness value can be positive, negative, or undefined.

print(df.skew())

Output:

1 math score 2.879976

reading score -1.118335

writing score -2.551347

placement score 0.704166

placement offer count -0.705767

club join year 0.395053

dtype: float64

**Putting Everything Together**

 It is important to analyse these individually, however, because there are certain useful functions in python that can be called upon to find these values. One such important function is the **.describe()** function that prints the summary statistic of the numerical variables. The line of code below performs this operation on the data.

df.describe()

Write a Python program to view some basic statistical details like percentile, mean, std etc. of the species of ‘Iris-setosa’, ‘Iris-versicolor’ and ‘Iris-virginica’.

**Sample Solution:**

**Python Code:**

import pandas as pd

data = pd.read\_csv("iris.csv")

print('Iris-setosa')

setosa = data['Species'] == 'Iris-setosa'

print(data[setosa].describe())

print('\nIris-versicolor')

setosa = data['Species'] == 'Iris-versicolor'

print(data[setosa].describe())

print('\nIris-virginica')

setosa = data['Species'] == 'Iris-virginica'

print(data[setosa].describe())

Iris-setosa

Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm

count 50.00000 50.00000 50.000000 50.000000 50.00000

mean 25.50000 5.00600 3.418000 1.464000 0.24400

std 14.57738 0.35249 0.381024 0.173511 0.10721

min 1.00000 4.30000 2.300000 1.000000 0.10000

25% 13.25000 4.80000 3.125000 1.400000 0.20000

50% 25.50000 5.00000 3.400000 1.500000 0.20000

75% 37.75000 5.20000 3.675000 1.575000 0.30000

max 50.00000 5.80000 4.400000 1.900000 0.60000

Iris-versicolor

Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm

count 50.00000 50.000000 50.000000 50.000000 50.000000

mean 75.50000 5.936000 2.770000 4.260000 1.326000

std 14.57738 0.516171 0.313798 0.469911 0.197753

min 51.00000 4.900000 2.000000 3.000000 1.000000

25% 63.25000 5.600000 2.525000 4.000000 1.200000

50% 75.50000 5.900000 2.800000 4.350000 1.300000

75% 87.75000 6.300000 3.000000 4.600000 1.500000

max 100.00000 7.000000 3.400000 5.100000 1.800000

Iris-virginica

Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm

count 50.00000 50.00000 50.000000 50.000000 50.00000

mean 125.50000 6.58800 2.974000 5.552000 2.02600

std 14.57738 0.63588 0.322497 0.551895 0.27465

min 101.00000 4.90000 2.200000 4.500000 1.40000

25% 113.25000 6.22500 2.800000 5.100000 1.80000

50% 125.50000 6.50000 3.000000 5.550000 2.00000

75% 137.75000 6.90000 3.175000 5.875000 2.30000

max 150.00000 7.90000 3.800000 6.900000 2.50000

Assignment No: 4

**Title of the Assignment: Create a Linear Regression Model using Python/R to predict home prices using Boston Housing Dataset** [(https://www.kaggle.com/c/boston-housing).](http://www.kaggle.com/c/boston-housing)) The Boston Housing dataset contains information about various houses in Boston through different parameters. 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.

## ----------------------------------------------------------------------------------------------------------

**Objective of the Assignment:** Students should be able to data analysis using liner regression using Python for any open source dataset

## ----------------------------------------------------------------------------------------------------------

**Prerequisite:**

## 1. Basic of Python Programming 2.Concept of Regresion.

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

## Contents for Theory:

1. **Linear Regression :**

## Example of Linear Regression

1. **Training data set and Testing data set**
2. **Linear Regression:** It is a machine learning algorithm based on supervised learning. It targets prediction values on the basis of independent variables.
   * It is preferred to find out the relationship between forecasting and variables.
   * A linear relationship between a dependent variable (X) is continuous; while independent variable(Y) relationship may be continuous or discrete. A linear relationship should be available in between predictor and target variable so known as Linear Regression.
   * Linear regression is popular because the cost function is Mean Squared Error (MSE) which is equal to the average squared difference between an observation’s actual and predicted values.
   * It is shown as an equation of line like : Y = m\*X + b + e

Where : b is intercepted, m is slope of the line and e is error term.

This equation can be used to predict the value of target variable Y based on given predictor variable(s) X, as shown in Fig. 1.

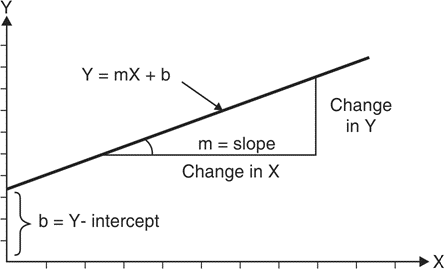


Fig. 1: geometry of linear regression

## Example of Linear Regression

Consider following data for 5 students.

Each Xi (i = 1 to 5) represents the score of ith student in standard X and corresponding Yi (i = 1 to 5) represents the score of ith student in standard XII.

1. Linear regression equation best predicts standard XIIth score
2. Interpretation for the equation of Linear Regression
3. If a student's score is 80 in std X, then what is his expected score in XII standard?

|  |  |  |
| --- | --- | --- |
| Student | Score in X standard (Xi) | Score in XII standard (Yi) |
| 1 | 95 | 85 |
| 2 | 85 | 95 |
| 3 | 80 | 70 |
| 4 | 70 | 65 |
| 5 | 60 | 70 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **x** | **y** | 𝑥 −𝑥 | 𝑦 −𝑦 | **(**𝑥 −𝑥 **)2** | (𝑥 −𝑥 **)(**𝑦 − 𝑦 **)** |
| 95 | 85 | 17 | 8 | 289 | 136 |
| 85 | 95 | 7 | 18 | 49 | 126 |
| 80 | 70 | 2 | -7 | 4 | -14 |
| 70 | 65 | -8 | -12 | 64 | 96 |
| 60 | 70 | -18 | -7 | 324 | 126 |
| 𝑥 = 78 | 𝑦= 77 |  |  | **ε (**𝑥 −𝑥 **)2= 730** | **ε** (𝑥 −𝑥 **)(**𝑦 − 𝑦 **) = 470** |

1. linear regression equation that best predicts standard XIIth score

# 𝑦 = β + β 𝑥

0 1

𝑛

𝑛 2

β = ∑ (𝑥

1 𝑖

𝑖=1

− 𝑥 ) (𝑦

− 𝑦 )/

𝑖

∑ (𝑥

𝑖=1

𝑖 −

𝑥 )

β = 470/730 = 0. 644

1

β = 𝑦 − β 𝑥

0 1

β = 77 − (0. 644 \* 78) = 26. 768

0

# 𝑦 = 26. 76 + 0. 644 𝑥

## Interpretation of the regression line.

**Interpretation 1**

For an increase in value of x by 0.644 units there is an increase in value of y in one unit.

## Interpretation 2

Even if x = 0 value of independent variable, it is expected that value of y is 26.768

Score in XII standard (Yi) is 0.644 units depending on Score in X standard (Xi) but other factors will also contribute to the result of XII standard by 26.768 .

## If a student's score is 65 in std X, then his expected score in XII standard is 78.288

For x = 80 the y value will be

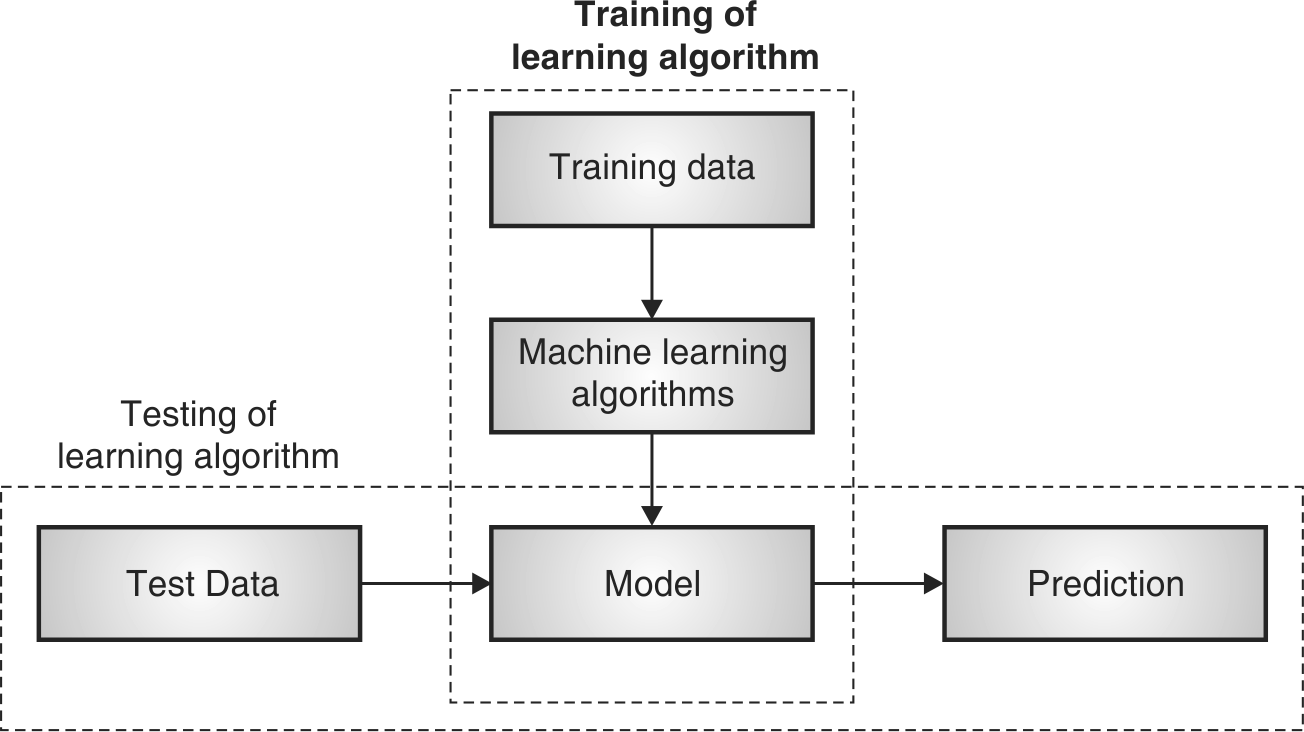
# 𝑦 = 26. 76 + 0. 644 \* 65 = 68. 38

## Training data set and Testing data set

* Machine Learning algorithm has two phases

1. Training and 2. Testing.

* The input of the training phase is training data, which is passed to any machine learning algorithm and machine learning model is generated as output of the training phase.
* The input of the testing phase is test data, which is passed to the machine learning model and prediction is done to observe the correctness of mode.



## Fig. 1.3.1 : Training and Testing Phase in Machine Learning

1. **Training Phase**
   * Training dataset is provided as input to this phase.
   * Training dataset is a dataset having attributes and class labels and used for training Machine Learning algorithms to prepare models.

Machines can learn when they observe enough relevant data. Using this one can model algorithms to find relationships, detect patterns, understand complex problems and make decisions**.**

* + Training error is the error that occurs by applying the model to the same data from which the model is trained**.**
  + In a simple way the actual output of training data and predicted output of the model does not match the training error Ein is said to have occurred.
  + Training error is much easier to compute.

## Testing Phase

* Testing dataset is provided as input to this phase.
* Test dataset is a dataset for which class label is unknown. It is tested using model
* A test dataset used for assessment of the finally chosen model.
* Training and Testing dataset are completely different.
* Testing error is the error that occurs by assessing the model by providing the unknown data to the model.
* In a simple way the actual output of testing data and predicted output of the model does not match the testing error Eout is said to have occurred.
* E out is generally observed larger than Ein.

## Algorithm (Synthesis Dataset):

Step 1: Import libraries and create alias for Pandas, Numpy and Matplotlib

import pandas as pd import numpy as np

import matplotlib.pyplot as plt

## Step 2: Create a Dataframe with Dependent Variable(x) and independent variable y.

x=np.array([95,85,80,70,60])

y=np.array([85,95,70,65,70])

## Step 3 : Create Linear Regression Model using Polyfit Function:

model= np.polyfit(x, y, 1)

## Step 4: Observe the coefficients of the model.

model

## Output:

array([ 0.64383562, 26.78082192])

## Step 5: Predict the Y value for X and observe the output.

predict = np.poly1d(model) predict(65)

## Output:

68.63

## Step 6: Predict the y\_pred for all values of x.

y\_pred= predict(x) y\_pred

## Output:

array([81.50684932, 87.94520548, 71.84931507, 68.63013699, 71.84931507])

## Step 7: Evaluate the performance of Model (R-Suare)

R squared calculation is not implemented in numpy… so that one should be borrowed from sklearn.

from sklearn.metrics import r2\_score r2\_score(y, y\_pred)

## Output:

0.4803218090889323

Step 8: Plotting the linear regression model y\_line = model[1] + model[0]\* x plt.plot(x, y\_line, c = 'r') plt.scatter(x, y\_pred) plt.scatter(x,y,c='r')

## Output:

Algorithm (Boston Dataset):

## Step 1: Import libraries and create alias for Pandas, Numpy and Matplotlib

import numpy as np import pandas as pd

import matplotlib.pyplot as plt

## Step 2: Import the Boston Housing dataset

from sklearn.datasets import load\_boston boston = load\_boston()

## Step 3: Initialize the data frame

data = pd.DataFrame(boston.data)

Step 4: Add the feature names to the dataframe data.columns = boston.feature\_names data.head()

## Step 5: Adding target variable to dataframe

data['PRICE'] = boston.target

## Step 6: Perform Data Preprocessing( Check for missing values)

data.isnull().sum()

## Step 7: Split dependent variable and independent variables

x = data.drop(['PRICE'], axis = 1) y = data['PRICE']

## Step 8: splitting data to training and testing dataset.

from sklearn.model\_selection import train\_test\_split xtrain, xtest, ytrain, ytest =

train\_test\_split(x, y, test\_size =0.2,random\_state = 0)

## Step 9: Use linear regression( Train the Machine ) to Create Model

import sklearn

from sklearn.linear\_model import LinearRegression lm = LinearRegression()

model=lm.fit(xtrain, ytrain)

## Step 10: Predict the y\_pred for all values of train\_x and test\_x

ytrain\_pred = lm.predict(xtrain) ytest\_pred = lm.predict(xtest)

## Step 11:Evaluate the performance of Model for train\_y and test\_y

df=pd.DataFrame(ytrain\_pred,ytrain) df=pd.DataFrame(ytest\_pred,ytest)

## Step 12: Calculate Mean Square Paper for train\_y and test\_y

from sklearn.metrics import mean\_squared\_error, r2\_score mse = mean\_squared\_error(ytest, ytest\_pred)

print(mse)

mse = mean\_squared\_error(ytrain\_pred,ytrain) print(mse)

## Output:

33.44897999767638

mse = mean\_squared\_error(ytest, ytest\_pred) print(mse)

## Output:

19.32647020358573

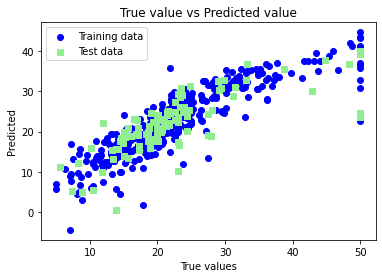
## Step 13: Plotting the linear regression model

lt.scatter(ytrain ,ytrain\_pred,c='blue',marker='o',label='Training data') plt.scatter(ytest,ytest\_pred ,c='lightgreen',marker='s',label='Test data') plt.xlabel('True values')

plt.ylabel('Predicted')

plt.title("True value vs Predicted value") plt.legend(loc= 'upper left') #plt.hlines(y=0,xmin=0,xmax=50)

plt.plot() plt.show()



C**onclusion:**

In this way we have done data analysis using linear regression for Boston Dataset and predict the price of houses using the features of

**Assignment No: 5**

### Title of the Assignment:

1. Implement logistic regression using Python/R to perform classification on Social\_Network\_Ads.csv dataset.
2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset..

### ----------------------------------------------------------------------------------------------------------------

**Objective of the Assignment:** Students should be able to data analysis using logistic regression using Python for any open source dataset

### ---------------------------------------------------------------------------------------------------------------

**Prerequisite:**

### 1. Basic of Python Programming 2.Concept of Regression.

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

### Contents for Theory:

1. **Logistic Regression**

### Differentiate between Linear and Logistic Regression

1. **Sigmoid Function**

### Types of LogisticRegression

1. **Confusion Matrix Evaluation Metrics**
2. **Logistic Regression:**

Logistic Regression is one of the most simple and commonly used Machine Learning algorithms for two-class classification. It is easy to implement and can be used as the baseline for any binary classification problem. Its basic fundamental concepts are also constructive in deep learning. Logistic regression describes and estimates the relationship between one dependent binary variable and independent variables.

Logistic regression is a statistical method for predicting binary classes. The outcome or target variable is dichotomous in nature. Dichotomous means there are only two possible classes. For example, it can be used for cancer detection problems. It computes the probability of an event occurring.

It is a special case of linear regression where the target variable is categorical in nature. It uses a log of odds as the dependent variable. Logistic Regression predicts the probability of occurrence of a binary event utilising a logit function.

### Linear Regression Equation:



Where, y is a dependent variable and x1, x2 ... and Xn are explanatory variables.

### Sigmoid Function:

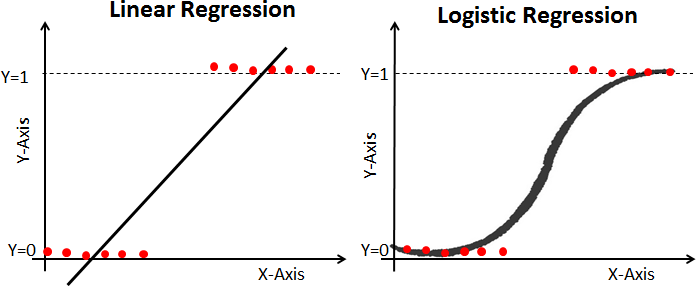


**Apply Sigmoid function on linear regression:**



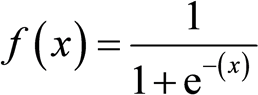
### Differentiate between Linear and Logistic Regression

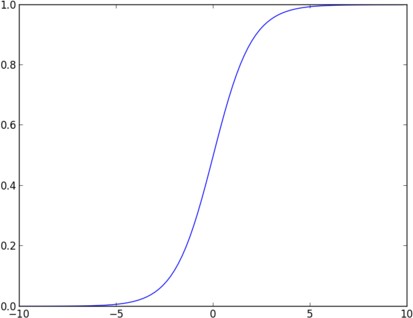
Linear regression gives you a continuous output, but logistic regression provides a constant output. An example of the continuous output is house price and stock price. Example's of the discrete output is predicting whether a patient has cancer or not, predicting whether the customer will churn. Linear regression is estimated using Ordinary Least Squares (OLS) while logistic regression is estimated using Maximum Likelihood Estimation (MLE) approach.



### Sigmoid Function

The sigmoid function, also called logistic function, gives an ‘S’ shaped curve that can take any real-valued number and map it into a value between 0 and 1. If the curve goes to positive infinity, y predicted will become 1, and if the curve goes to negative infinity, y predicted will become 0. If the output of the sigmoid function is more than 0.5, we can classify the outcome as 1 or YES, and if it is less than 0.5, we can classify it as 0 or NO. The outputcannotFor example: If the output is 0.75, we can say in terms of probability as: There is a 75 percent chance that a patient will suffer from cancer.





### Types of LogisticRegression

**Binary Logistic Regression:** The target variable has only two possible outcomes such as Spam or Not Spam, Cancer or No Cancer.

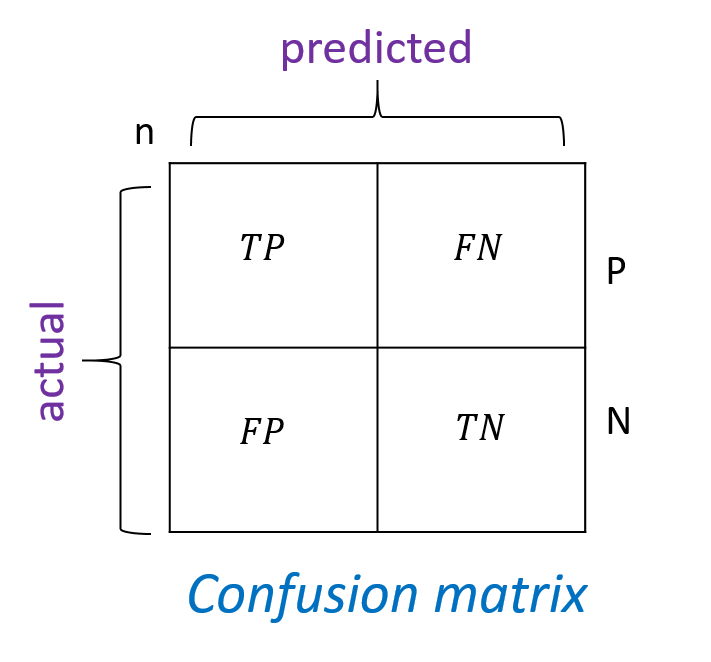
**Multinomial Logistic Regression:** The target variable has three or more nominal categories such as predicting the type of Wine.

**Ordinal Logistic Regression:** the target variable has three or more ordinal categories such as restaurant or product rating from 1 to 5.

**Confusion Matrix Evaluation Metrics**

Contingency table or Confusion matrix is often used to measure the performance of classifiers. A confusion matrix contains information about actual and predicted classifications done by a classification system. Performance of such systems is commonly evaluated using the data in the matrix.

The following table shows the confusion matrix for a two class classifier.



Here each row indicates the actual classes recorded in the test data set and the each column indicates the classes as predicted by the classifier.

Numbers on the descending diagonal indicate correct predictions, while the ascending diagonal concerns prediction errors.

Some Important measures derived from confusion matrix are:

* **Number of positive (Pos) :** Total number instances which are labelled as positive in a given dataset.
* **Number of negative (Neg)** : Total number instances which are labelled as negative in a given dataset.
* **Number of True Positive (TP) :** Number of instances which are actually labelled as positive and the predicted class by classifier is also positive.
* **Number of True Negative (TN) :** Number of instances which are actually labelled as negative and the predicted class by classifier is also negative.
* **Number of False Positive (FP) :** Number of instances which are actually labelled as negative and the predicted class by classifier is positive.
* **Number of False Negative (FN):** Number of instances which are actually labelled as positive and the class predicted by the classifier is negative.

**Accuracy:** Accuracy is calculated as the number of correctly classified instances divided by total number of instances.The ideal value of accuracy is 1, and the worst is 0. It is also calculated as the sum of true positive and true negative (TP + TN) divided by the total number of instances.

𝑇𝑃+𝑇𝑁

𝑎𝑐𝑐 =

𝑇𝑃+𝐹𝑃+𝑇𝑁+𝐹𝑁

𝑇𝑃+𝑇𝑁

𝑃𝑜𝑠+𝑁𝑒𝑔

=

* **Error Rate:** Error Rate is calculated as the number of incorrectly classified instances divided by total number of instances.

The ideal value of accuracy is 0, and the worst is 1. It is also calculated as the sum of false positive and false negative (FP + FN) divided by the total number of instances.

𝐹𝑃+𝐹𝑁

𝑒𝑟𝑟 =

𝑇𝑃+𝐹𝑃+𝑇𝑁+𝐹𝑁

𝐹𝑃+𝐹𝑁

𝑃𝑜𝑠+𝑁𝑒𝑔 Or

=

# 𝑒𝑟𝑟 = 1 − 𝑎𝑐𝑐

* **Precision:** It is calculated as the number of correctly classified positive instances divided by the total number of instances which are predicted positive. It is also called confidence value. The ideal value is 1, whereas the worst is 0.

## 𝑝𝑟𝑒𝑐𝑖𝑠𝑖𝑜𝑛 = 𝑇𝑃

𝑇𝑃+𝐹𝑃

* **Recall: .**It is calculated as the number of correctly classified positive instances divided by the total number of positive instances. It is also called recall or sensitivity. The ideal value of sensitivity is 1, whereas the worst is 0.

It is calculated as the number of correctly classified positive instances divided by the total number of positive instances.

## 𝑟𝑒𝑐𝑎𝑙𝑙 = 𝑇𝑃

𝑇𝑃+𝐹𝑁

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv(r"D:\dsbldlab\Social\_Network\_Ads.csv")

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set

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)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# Fitting Logistic Regression to the Training set

from sklearn.linear\_model import LogisticRegression

log\_reg = LogisticRegression(random\_state = 0)

log\_reg.fit(X\_train, y\_train)

# Predicting the Test set results

y\_pred = log\_reg.predict(X\_test)

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

# Visualising the Training set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01), np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1,X2,log\_reg.predict(np.array([X1.ravel(),X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Logistic Regression (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

# Visualising the Test set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01), np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1,X2,log\_reg.predict(np.array([X1.ravel(),X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

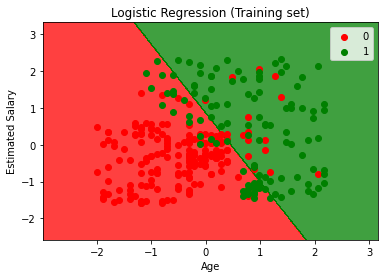
plt.title('Logistic Regression (Test set)')

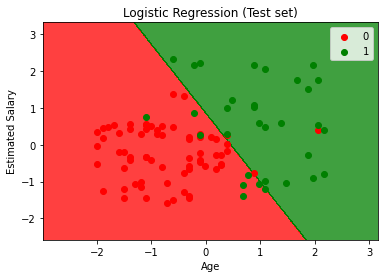
plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

****

****

**Conclusion**

Logistic Regression works fine only when the target variable is discrete in nature. They do not have the flexibility to act as regression analysis. Also, they have less chance of overfitting but in data having a higher dimension, logistic can overfit. For such cases, there are regularizing techniques called L1 and L2 which shrink the coefficients of the algorithm to avoid overfitting.  Logistic regression can also work fine with the discretised data as they do not follow a decision-based

Assignment No: 6

# Title of the Assignment:

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 of the Assignment:** Students should be able to data analysis using Naïve Bayes classification algorithm using Python for any open source dataset

# ------------------------------------------------------------------------------------------Prerequisite:

# Basic of Python Programming

1. **Concept of Join and Marginal Probability.**

# ------------------------------------------------------------------------------------------

**Contents for Theory:**

# Concepts used in Naïve Bayes classifier

1. **Naive Bayes Example**

# Confusion Matrix Evaluation Metrics

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

# 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.

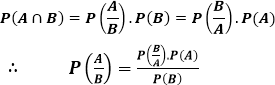
For example, P(A), P(B), P(C) are prior probabilities because while calculating P(A), occurrences of event B or C are not concerned i.e. no information about occurrence of any other event is used.

**Conditional Probabilities:**





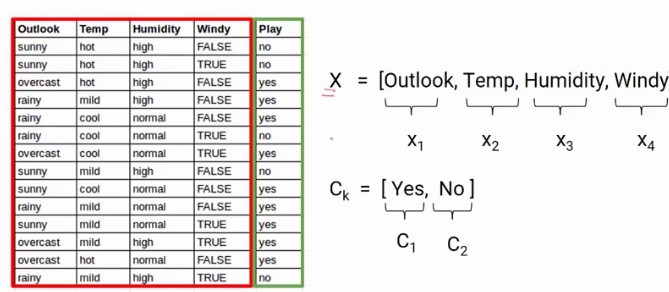
From equation (1) and (2) ,



Is called the Bayes Rule.

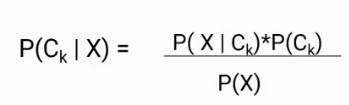
# Example of Naive Bayes

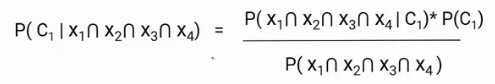
We have a dataset with some features Outlook, Temp, Humidity, and Windy, and the target here is to predict whether a person or team will play tennis or not.



Conditional Probability

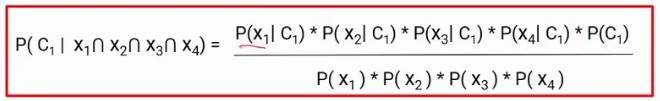
Here, we are predicting the probability of class1 and class2 based on the given condition. If I try to write the same formula in terms of classes and features, we will get the following equation



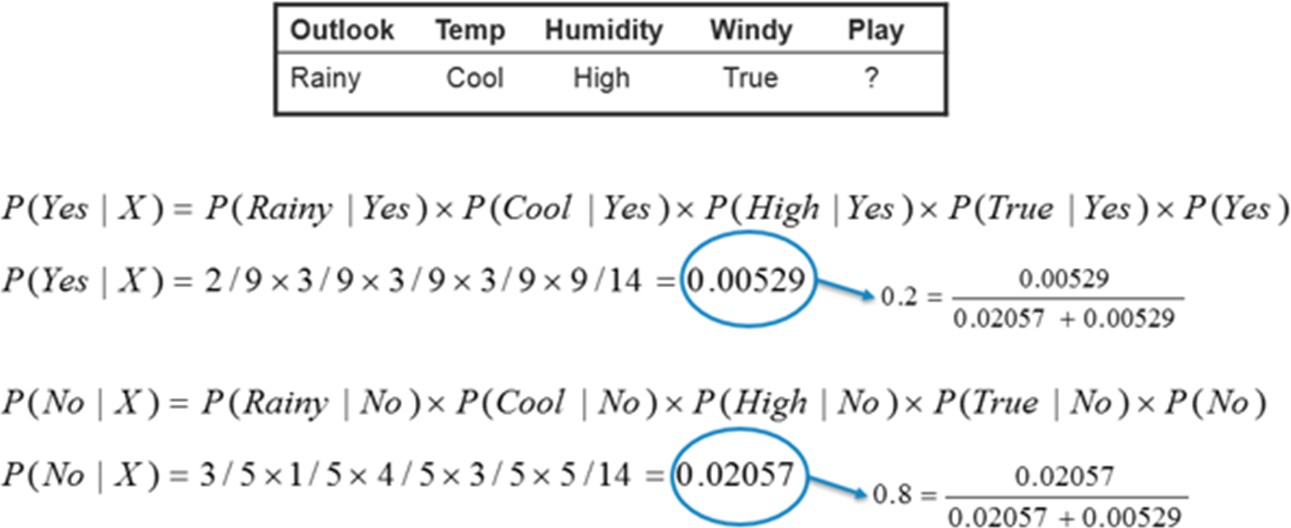
Now we have two classes and four features, so if we write this formula for class C1, it will be something like this.

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



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.



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

# 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.

# 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\_score cm = confusion\_matrix(y\_test, Y\_pred)

C**onclusion:**

In this way we have done data analysis using Naive Bayes Algorithm for Iris dataset and evaluated the performance of the model.

Assignment No: 7

# Title of the Assignment:

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

# ---------------------------------------------------------------------------------------

**Objective of the Assignment:** Students should be able to perform **Text Analysis** using TF IDF Algorithm

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

**Prerequisite:**

# Basic of Python Programming

1. **Basic of English language.**

# --------------------------------------------------------------------------------------

**Contents for Theory:**

# Basic concepts of Text Analytics

1. **Text Analysis Operations using natural language toolkit**

# Text Analysis Model using TF-IDF.

1. **Bag of Words (BoW)**

# ---------------------------------------------------------------------------------------

1. **Basic concepts of Text Analytics**

One of the most frequent types of day-to-day conversion is text communication. In our everyday routine, we chat, message, tweet, share status, email, create blogs, and offer opinions and criticism. All of these actions lead to a substantial amount of unstructured text being produced. It is critical to examine huge amounts of data in this sector of the online world and social media to determine people's opinions.

Text mining is also referred to as text analytics. Text mining is a process of exploring sizable textual data and finding patterns. Text Mining processes the text itself, while NLP processes with the underlying metadata. Finding frequency counts of words, length of the sentence, presence/absence of specific words is known as text mining. Natural language processing is one of the components of text mining. NLP helps identify sentiment, finding entities in the sentence, and category of blog/article. Text mining is preprocessed data for text analytics. In Text Analytics, statistical and machine learning algorithms are used to classify information.

# Text Analysis Operations using natural language toolkit

NLTK(natural language toolkit) is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning and many more.

Analysing movie reviews is one of the classic examples to demonstrate a simple NLP Bag-of-words model, on movie reviews.

# Tokenization:

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

* + - Sentence tokenization : split a paragraph into **list of sentences** using **sent\_tokenize()** method
    - Word tokenization : split a sentence into **list of words** using **word\_tokenize()** method

# Stop words removal

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.

# Stemming and Lemmatization

**Stemming** is a normalization technique where lists of tokenized words are converted into shortened root words to remove redundancy. Stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form.

A computer program that stems word may be called a stemmer. E.g.

A stemmer reduces the words like fishing, fished, and fisher to the stem fish.

The stem need not be a word, for example the Porter algorithm reduces, argue, argued, argues, arguing, and argus to the stem argu .

**Lemmatization** in NLTK is the algorithmic process of finding the lemma of a word depending on its meaning and context. Lemmatization usually refers to the morphological analysis of words, which aims to remove inflectional endings. It helps in returning the base or dictionary form of a word known as the lemma.

Eg. Lemma for studies is study

# Lemmatization Vs Stemming

Stemming algorithm works by cutting the suffix from the word. In a broader sense cuts either the beginning or end of the word.

On the contrary, Lemmatization is a more powerful operation, and it takes into consideration morphological analysis of the words. It returns the lemma which is the base form of all its inflectional forms. In-depth linguistic knowledge is required to create dictionaries and look for the proper form of the word. Stemming is a general operation while lemmatization is an intelligent operation where the proper form will be looked in the dictionary. Hence, lemmatization helps in forming better machine learning features.

# POS Tagging

POS (Parts of Speech) tell us about grammatical information of words of the sentence by assigning specific token (Determiner, noun, adjective , adverb , verb,Personal Pronoun etc.) as tag (DT,NN ,JJ,RB,VB,PRP etc) to each words.

Word can have more than one POS depending upon the context where it is used. We can use POS tags as statistical NLP tasks. It distinguishes a sense of word which is very helpful in text realization and infer semantic information from text for sentiment analysis.

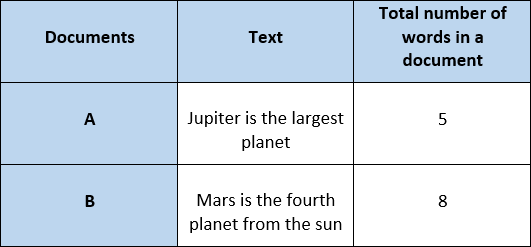
# Text Analysis Model using TF-IDF.

Term frequency–inverse document frequency(TFIDF) , is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.

# Term Frequency (TF)

It is a measure of the frequency of a word (w) in a document (d). TF is defined as the ratio of a word’s occurrence in a document to the total number of words in a document. The denominator term in the formula is to normalize since all the corpus documents are of different lengths.

Example:



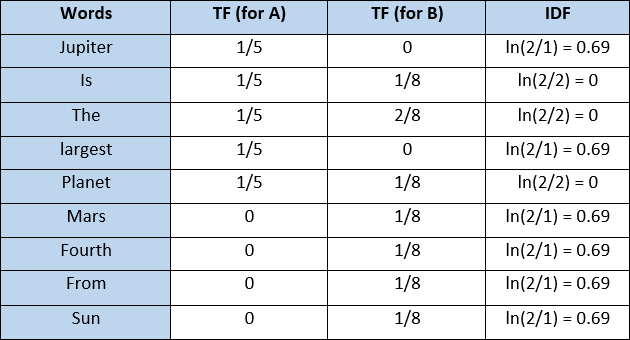
The initial step is to make a vocabulary of unique words and calculate TF for each document. TF will be more for words that frequently appear in a document and less for rare words in a document.

# Inverse Document Frequency (IDF)

It is the measure of the importance of a word. Term frequency (TF) does not consider the importance of words. Some words such as’ of’, ‘and’, etc. can be most frequently present but are of little significance. IDF provides weightage to each word based on its frequency in the corpus D.



In our example, since we have two documents in the corpus, N=2.

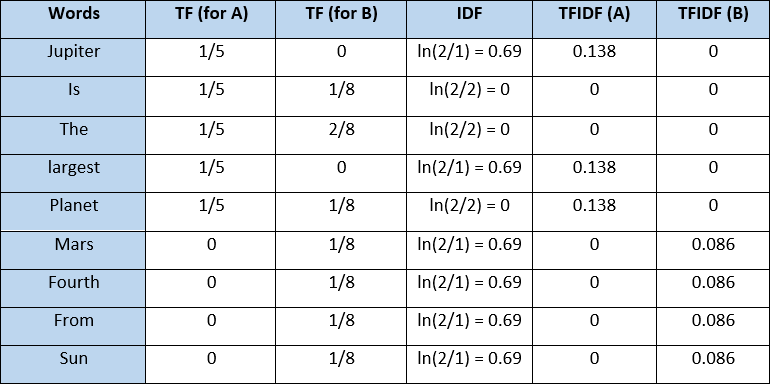


# Term Frequency — Inverse Document Frequency (TFIDF)

It is the product of TF and IDF.

TFIDF gives more weightage to the word that is rare in the corpus (all the documents). TFIDF provides more importance to the word that is more frequent in the document.





After applying TFIDF, text in A and B documents can be represented as a TFIDF vector of dimension equal to the vocabulary words. The value corresponding to each word represents the importance of that word in a particular document.

TFIDF is the product of TF with IDF. Since TF values lie between 0 and 1, not using ***ln*** can result in high IDF for some words, thereby dominating the TFIDF. We don’t want that, and therefore, we use ***ln*** so that the IDF should not completely dominate the TFIDF.

# Disadvantage of TFIDF

It is unable to capture the semantics. For example, funny and humorous are synonyms, but TFIDF does not capture that. Moreover, TFIDF can be computationally expensive if the vocabulary is vast.

# Bag of Words (BoW)

Machine learning algorithms cannot work with raw text directly. Rather, the text must be converted into vectors of numbers. In natural language processing, a common technique for extracting features from text is to place all of the words that occur in the text in a bucket. This approach is called a bag of words model or BoW for short. It’s referred to as a “bag” of words because any information about the structure of the sentence is lost.

# Algorithm for Tokenization, POS Tagging, stop words removal, Stemming and Lemmatization:

**Step 1: Download the required packages**

nltk.download('punkt') nltk.download('stopwords') nltk.download('wordnet') nltk.download('averaged\_perceptron\_tagger')

# Step 2: Initialize the text

text= "Tokenization is the first step in text analytics. The process of breaking down a text paragraph into smaller chunks such as words or sentences is called Tokenization."

# Step 3: Perform Tokenization

#Sentence Tokenization

from nltk.tokenize import sent\_tokenize

tokenized\_text= sent\_tokenize(text) print(tokenized\_text)

#Word Tokenization

from nltk.tokenize import word\_tokenize tokenized\_word=word\_tokenize(text) print(tokenized\_word)

# Step 4: Removing Punctuations and Stop Word

# print stop words of English

from nltk.corpus import stopwords stop\_words=set(stopwords.words("english")) print(stop\_words)

text= "How to remove stop words with NLTK library in Python?" text= re.sub('[^a-zA-Z]', ' ',text)

tokens = word\_tokenize(text.lower()) filtered\_text=[]

for w in tokens:

if w not in stop\_words: filtered\_text.append(w)

print("Tokenized Sentence:",tokens) print("Filterd Sentence:",filtered\_text)

# Step 5 : Perform Stemming

from nltk.stem import PorterStemmer

e\_words= ["wait", "waiting", "waited", "waits"] ps =PorterStemmer()

for w in e\_words:

rootWord=ps.stem(w) print(rootWord)

# Step 6: Perform Lemmatization

from nltk.stem import WordNetLemmatizer wordnet\_lemmatizer = WordNetLemmatizer() text = "studies studying cries cry" tokenization = nltk.word\_tokenize(text)

for w in tokenization:

print("Lemma for {} is {}".format(w, wordnet\_lemmatizer.lemmatize(w)))

# Step 7: Apply POS Tagging to text

import nltk

from nltk.tokenize import word\_tokenize data="The pink sweater fit her perfectly" words=word\_tokenize(data)

for word in words:

print(nltk.pos\_tag([word]))

# Algorithm for Create representation of document by calculating TFIDF Step 1: Import the necessary libraries.

**import pandas as pd**

from sklearn.feature\_extraction.text import TfidfVectorizer

# Step 2: Initialize the Documents.

documentA = 'Jupiter is the largest Planet' documentB = 'Mars is the fourth planet from the Sun'

# Step 3: Create BagofWords (BoW) for Document A and B.

bagOfWordsA = documentA.split(' ') bagOfWordsB = documentB.split(' ')

# Step 4: Create Collection of Unique words from Document A and B.

uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))

# Step 5: Create a dictionary of words and their occurrence for each document in the corpus

numOfWordsA = dict.fromkeys(uniqueWords, 0)

for word in bagOfWordsA: numOfWordsA[word] += 1

numOfWordsB = dict.fromkeys(uniqueWords, 0) for word in bagOfWordsB:

numOfWordsB[word] += 1

# Step 6: Compute the term frequency for each of our documents.

def computeTF(wordDict, bagOfWords): tfDict = {}

bagOfWordsCount = len(bagOfWords) for word, count in wordDict.items():

tfDict[word] = count / float(bagOfWordsCount) return tfDict

tfA = computeTF(numOfWordsA, bagOfWordsA) tfB = computeTF(numOfWordsB, bagOfWordsB)

# Step 7: Compute the term Inverse Document Frequency.

def computeIDF(documents):

import math

N = len(documents)

idfDict = dict.fromkeys(documents[0].keys(), 0) for document in documents:

for word, val in document.items(): if val > 0:

idfDict[word] += 1

for word, val in idfDict.items(): idfDict[word] = math.log(N / float(val))

return idfDict

idfs = computeIDF([numOfWordsA, numOfWordsB]) idfs

# Step 8: Compute the term TF/IDF for all words.

def computeTFIDF(tfBagOfWords, idfs): tfidf = {}

for word, val in tfBagOfWords.items(): tfidf[word] = val \* idfs[word]

return tfidf

tfidfA = computeTFIDF(tfA, idfs) tfidfB = computeTFIDF(tfB, idfs)

df = pd.DataFrame([tfidfA, tfidfB]) df

C**onclusion:**

In this way we have done text data analysis using TF IDF algorithm

Assignment No: 8

# Title of the Assignment:

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 of the Assignment:** Students should be able to do data visualization in Python

[Seaborn](https://seaborn.pydata.org/) which is another extremely useful library for data visualization in Python. The Seaborn library is built on top of Matplotlib and offers many advanced data visualization capabilities.

Though, the Seaborn library can be used to draw a variety of charts such as matrix plots, grid plots, regression plots etc

**The Dataset**

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.

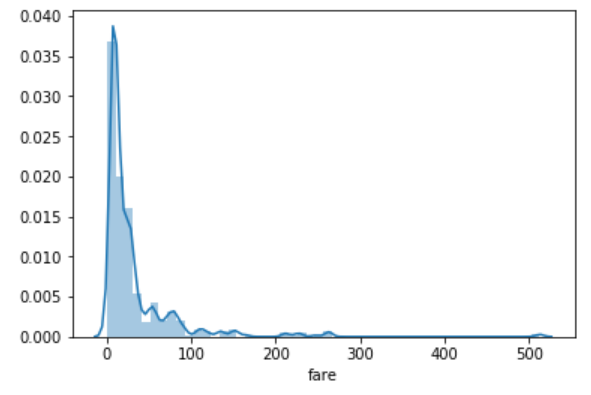
### Distributional Plots

Distributional plots, as the name suggests are type of plots that show the statistical distribution of data. In this section we will see some of the most commonly used distribution plots in Seaborn.

#### The Dist Plot

The distplot() shows the histogram distribution of data for a single column. The column name is passed as a parameter to the distplot() function.

sns.distplot(dataset['fare'])



You can see that most of the tickets have been solved between 0-50 dollars. The line that you see represents the [kernel density estimation](https://en.wikipedia.org/wiki/Kernel_density_estimation). You can remove this line by passing False as the parameter for the kde attribute as shown below:

sns.distplot(dataset['fare'], kde=False)

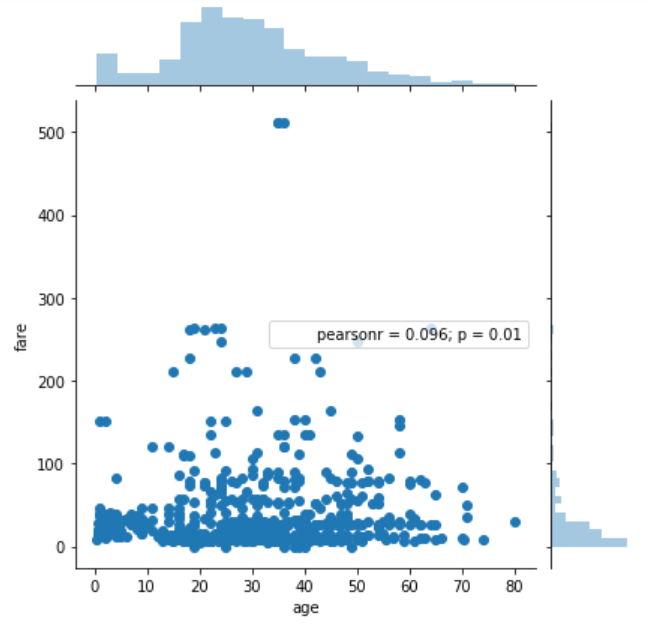
#### The Joint Plot

The jointplot()is used to display the mutual distribution of each column. You need to pass three parameters to jointplot. The first parameter is the column name for which you want to display the distribution of data on x-axis. The second parameter is the column name for which you want to display the distribution of data on y-axis. Finally, the third parameter is the name of the data frame.

Let's plot a joint plot of age and fare columns to see if we can find any relationship between the two.

sns.jointplot(x='age', y='fare', data=dataset)

**Output:**

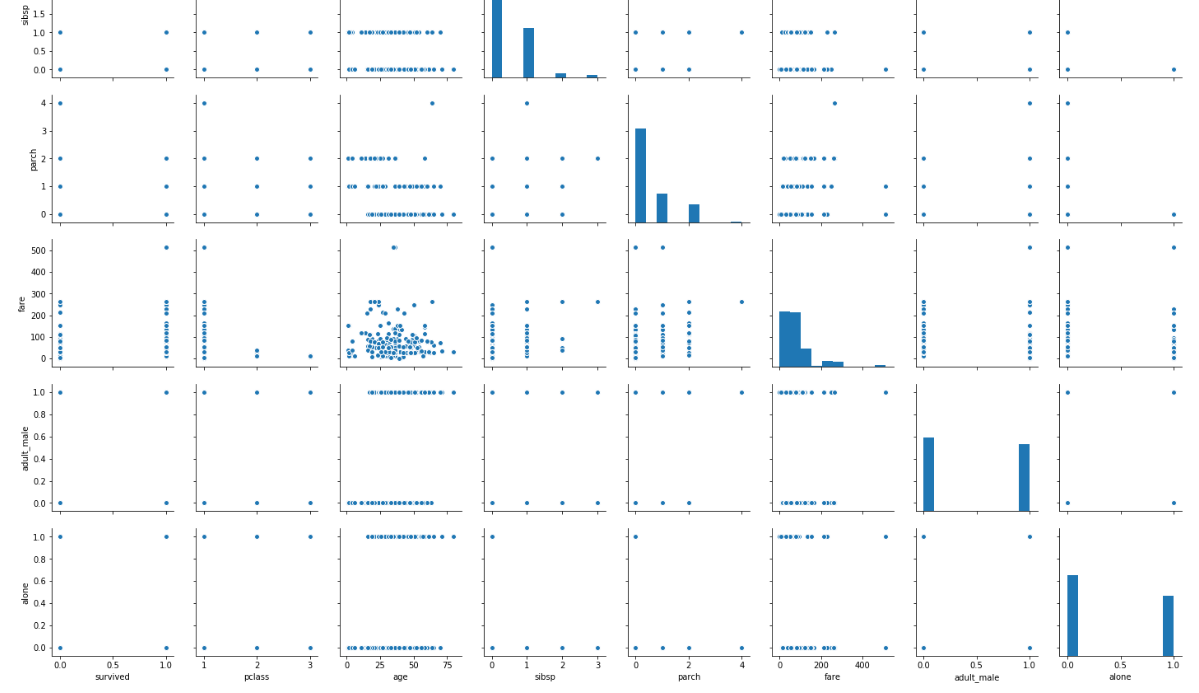


From the output, you can see that a joint plot has three parts. A distribution plot at the top for the column on the x-axis, a distribution plot on the right for the column on the y-axis and a scatter plot in between that shows the mutual distribution of data for both the columns.

#### The Pair Plot

The paitplot() is a type of distribution plot that basically plots a joint plot for all the possible combination of numeric and Boolean columns in your dataset. You only need to pass the name of your dataset as the parameter to the pairplot() function as shown below:

sns.pairplot(dataset)



Before executing the script above, remove all null values from the dataset using the following command:

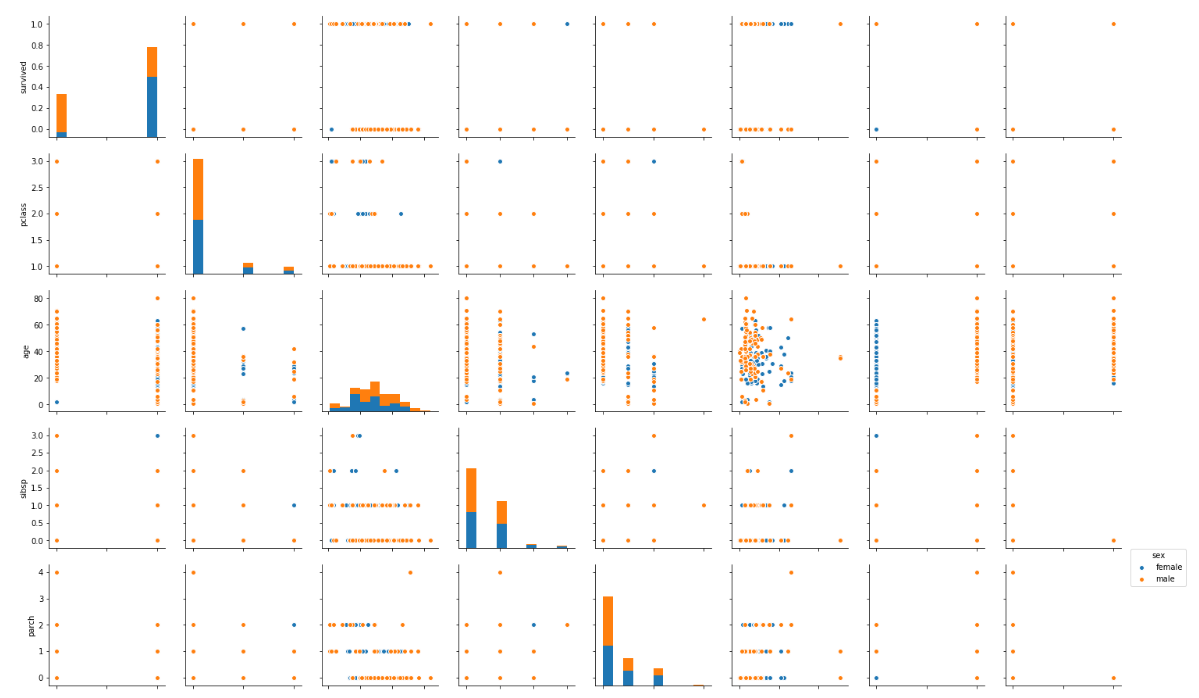
dataset = dataset.dropna()

From the output of the pair plot you can see the joint plots for all the numeric and Boolean columns in the Titanic dataset.

o add information from the categorical column to the pair plot, you can pass the name of the categorical column to the hue parameter. For instance, if we want to plot the gender information on the pair plot, we can execute the following script:

sns.pairplot(dataset, hue='sex')

**Output:**

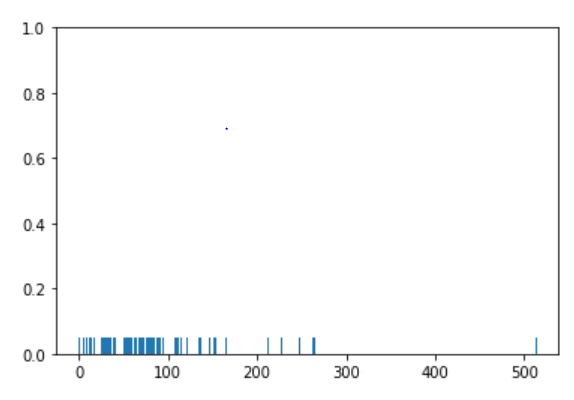


#### The Rug Plot

The rugplot() is used to draw small bars along 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'])

**Output:**



### 

### Categorical Plots

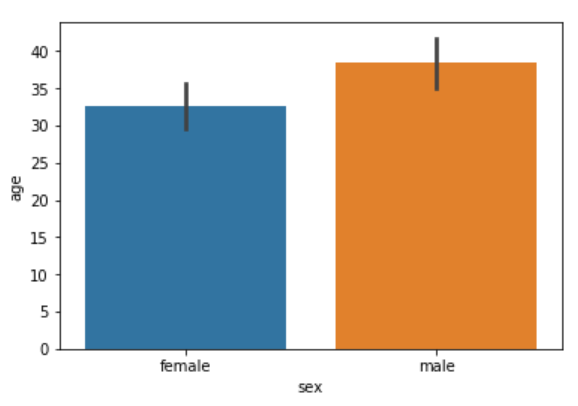
Categorical plots, as the name suggests are normally used to plot categorical data. The categorical plots plot the values in the categorical column against another categorical column or a numeric column. Let's see some of the most commonly used categorical data.

#### 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.

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

**Output:**



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.

#### 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:

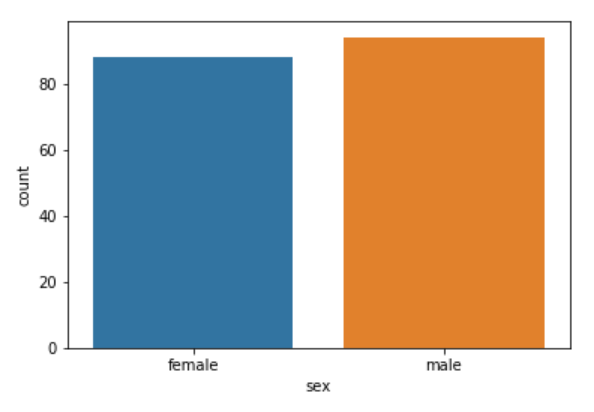
sns.countplot(x='sex', data=dataset)

The output shows the count as follows:

**Output:**

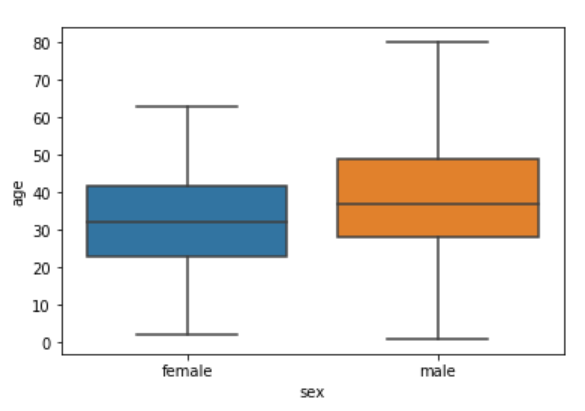
#### https://s3.amazonaws.com/stackabuse/media/seaborn-library-data-visualization-python-part-1-12.png The Box Plot

The box plot is used to display the distribution of the categorical data in the form of quartiles. The center 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.



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

**Output:**



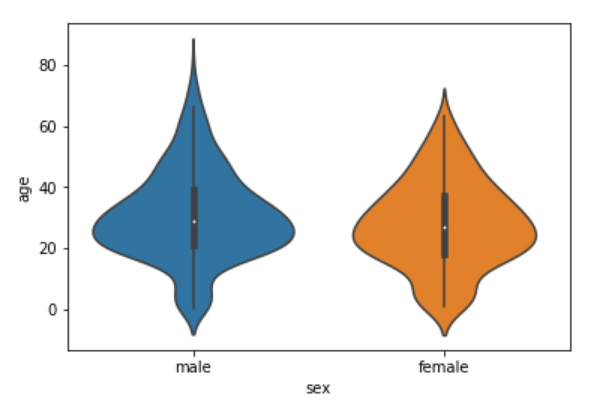
#### The Violin Plot

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)

**Output:**



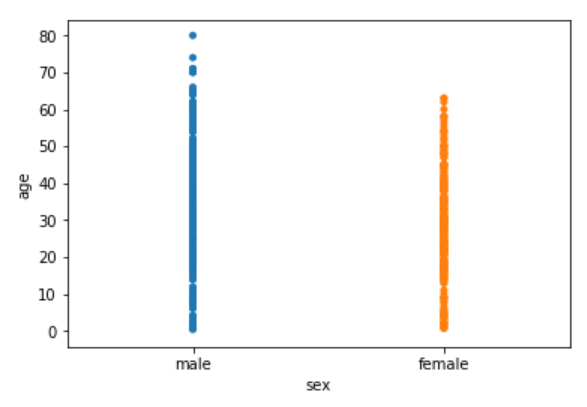
#### The Strip Plot

The strip plot draws a scatter plot where one of the variables is categorical.

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)

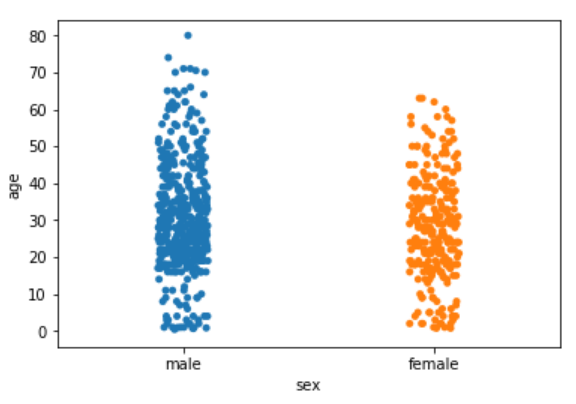
**Output:**



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)

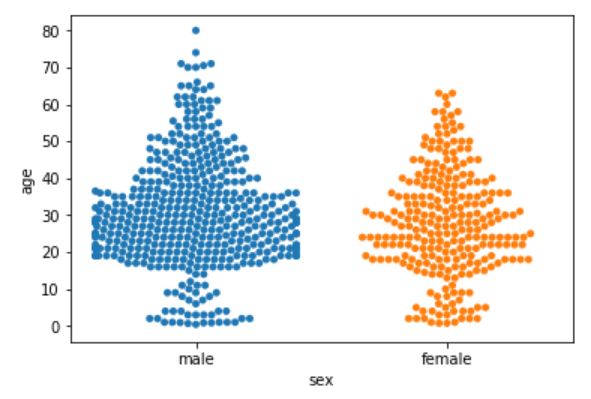
**Output:**



#### 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

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



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

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

**Output:**

We can also split swarm plots as we did in the case of strip and box plots. Execute the following script to do so:

sns.swarmplot(x='sex', y='age', data=dataset, hue='survived', split=True)

**Output:**

### https://s3.amazonaws.com/stackabuse/media/seaborn-library-data-visualization-python-part-1-24.pngConclusion

[Seaborn](https://seaborn.pydata.org/) is an advanced data visualization library built on top of [Matplotlib library](https://matplotlib.org/). In this article, we looked at how we can draw distributional and categorical plots using Seaborn library.

Assignment No: 9

### Title of the Assignment: Data Visualization II

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 of the Assignment:** Students should be able to perform Data Visualization operations using Python for any open source dataset

### The Dataset

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()

**The Box Plot**

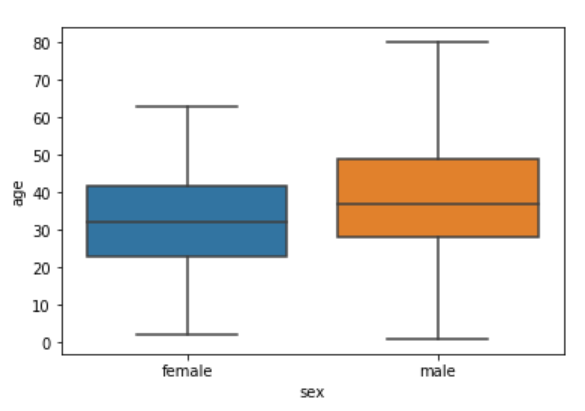
The box plot is used to display the distribution of the categorical data in the form of quartiles. The center of the box shows the median value. The value from the lower whisker to the bottom of the box shows the first quartile. From the bottom of the box to the middle of the box lies the second quartile.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

You can study more about quartiles and box plots at [this link](http://www.physics.csbsju.edu/stats/box2.html).

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)

**Output:**



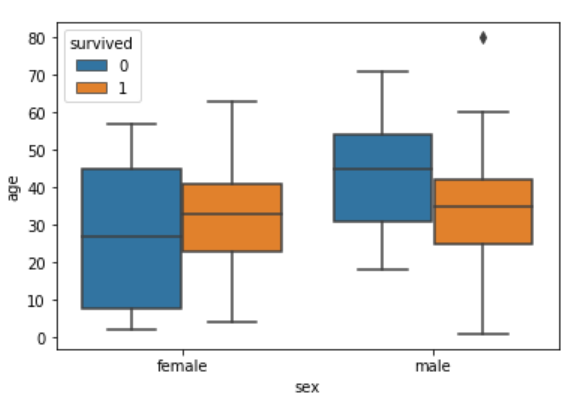
The first quartile starts at around 5 and ends at 22 which means that 25% of the passengers are aged between 5 and 25. The second quartile starts at around 23 and ends at around 32 which means that 25% of the passengers are aged between 23 and 32. Similarly, the third quartile starts and ends between 34 and 42, hence 25% passengers are aged within this range and finally the fourth or last quartile starts at 43 and ends around 65.

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.

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")

**Output:**



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.

.

Assignment No: 10

### Title of the Assignment: 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 Data Visualization operations using Python for any open source dataset

import matplotlib.pyplot as plt

import pandas as pd

path = "iris.csv"

df = pd.read\_csv("D:\dsbldlab\Iris1.csv")

headers = ["Sepal-length", "Sepal-width", "Petal-length", "Petal-width", "Species"]

df.columns = headers

print(df.head())

print(df.tail())

print(df.info())

print(df.shape)

print(df.dtypes)

print(df.describe())

df.hist()

plt.show()

df.boxplot()

plt.show()

plt.scatter(df["Sepal-length"], df["Sepal-width"])

plt.xlabel('Sepal Length')

plt.ylabel('Sepal Width')

plt.show()

plt.scatter(df["Sepal-length"], df["Petal-length"])

plt.xlabel('Sepal Length')

plt.ylabel('Petal Width')

plt.show()

plt.scatter(df["Sepal-length"], df["Petal-width"])

plt.xlabel('Sepal Length')

plt.xlabel('Petal Width')

plt.show()

plt.scatter(df["Sepal-width"], df["Sepal-length"])

plt.xlabel('Sepal Width')

plt.ylabel('Sepal Length')

plt.show()

plt.scatter(df["Sepal-width"], df["Petal-length"])

plt.xlabel('Sepal Width')

plt.ylabel('Petal Length')

plt.show()

plt.scatter(df["Sepal-width"], df["Petal-width"])

plt.xlabel('Sepal Width')

plt.ylabel('Petal Width')

plt.show()

plt.scatter(df["Petal-length"], df["Sepal-length"])

plt.xlabel('Petal Length')

plt.ylabel('Sepal Length')

plt.show()

plt.scatter(df["Petal-length"], df["Sepal-width"])

plt.xlabel('Petal Length')

plt.ylabel('Sepal Width')

plt.show()

plt.scatter(df["Petal-length"], df["Petal-width"])

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

plt.show()

plt.scatter(df["Petal-width"], df["Sepal-length"])

plt.xlabel('Petal Width')

plt.ylabel('Sepal Length')

plt.show()

plt.scatter(df["Petal-width"], df["Sepal-width"])

plt.xlabel('Petal Width')

plt.ylabel('Sepal Width')

plt.show()

plt.scatter(df["Petal-width"], df["Petal-length"])

plt.xlabel('Petal Width')

plt.ylabel('Petal Length')

plt.show()

Sepal-length Sepal-width Petal-length Petal-width Species

0 5.1 3.5 1.4 0.2 Iris-setosa

1 4.9 3.0 1.4 0.2 Iris-setosa

2 4.7 3.2 1.3 0.2 Iris-setosa

3 4.6 3.1 1.5 0.2 Iris-setosa

4 5.0 3.6 1.4 0.2 Iris-setosa

Sepal-length Sepal-width Petal-length Petal-width Species

145 6.7 3.0 5.2 2.3 Iris-virginica

146 6.3 2.5 5.0 1.9 Iris-virginica

147 6.5 3.0 5.2 2.0 Iris-virginica

148 6.2 3.4 5.4 2.3 Iris-virginica

149 5.9 3.0 5.1 1.8 Iris-virginica

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 150 entries, 0 to 149

Data columns (total 5 columns):

# Column Non-Null Count Dtype

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

0 Sepal-length 150 non-null float64

1 Sepal-width 150 non-null float64

2 Petal-length 150 non-null float64

3 Petal-width 150 non-null float64

4 Species 150 non-null object

dtypes: float64(4), object(1)

memory usage: 6.0+ KB

None

(150, 5)

Sepal-length float64

Sepal-width float64

Petal-length float64

Petal-width float64

Species object

dtype: object

Sepal-length Sepal-width Petal-length Petal-width

count 150.000000 150.000000 150.000000 150.000000

mean 5.843333 3.054000 3.758667 1.198667

std 0.828066 0.433594 1.764420 0.763161

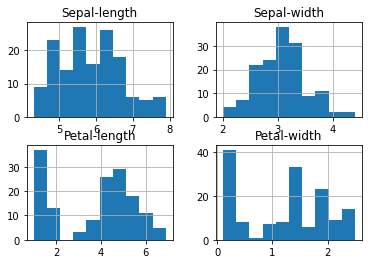
min 4.300000 2.000000 1.000000 0.100000

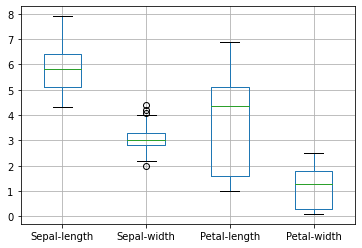
25% 5.100000 2.800000 1.600000 0.300000

50% 5.800000 3.000000 4.350000 1.300000

75% 6.400000 3.300000 5.100000 1.800000

max 7.900000 4.400000 6.900000 2.500000





|  |  |
| --- | --- |
| C:\Users\dbms15\Downloads\3.png | C:\Users\dbms15\Downloads\4.png |
| C:\Users\dbms15\Downloads\5.png | C:\Users\dbms15\Downloads\6.png |
| C:\Users\dbms15\Downloads\7.png | C:\Users\dbms15\Downloads\8.png |
| C:\Users\dbms15\Downloads\9.png | C:\Users\dbms15\Downloads\10.png |
| C:\Users\dbms15\Downloads\11.png | C:\Users\dbms15\Downloads\12.png |