G. H. Raisoni College of Engineering & Management, Wagholi, Pu

(NAAC ACCREDITED)

1200 Gat No, Domkhel Road Wagholi , Pune-412207

Email:- ghrcem.pune@raisoni.net Web:- https://ghrcem.raisoni.net/

Department of CSE (Cyber Security)

LAB MANUAL INDEX

INSTITUTE VISION AND MISSION

Department of CSE (Cyber Security)

VISION

To achieve excellent standards of quality education by keeping pace with rapidly changing technologies and to create technical manpower of global standards with capabilities of accepting new challenges.

MISSION

Our efforts are dedicated to impart quality and value-based education to raise the satisfaction level of all stakeholders. Our strength is directed to create competent professionals. Our endeavor is to provide all possible support to promote research and development activities.

DEPARTMENT VISION AND MISSION

VISION

Our vision is to prepare students to take their rightful place in the global community of computer professionals by fostering the skills and confidence to deal with challenges posed by rapidly changing technology.

MISSION

The department continuously strives

M1: To inculcate technical excellence in students through the dimension of technical knowledge, skills and intellectual stimulation.

M2: Empower students to contribute for the betterment of society individually and collectively.

Program Outcomes (POs)

Engineering Graduates will be able to:

Department of CSE (Cyber Security)

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

Department of CSE (Cyber Security)

11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

1. **PSO1:** Apply the fundamentals of domain knowledge in developing the effective computing solutions for real world problems.
2. **PSO2:** Formulate solutions for interdisciplinary problems using latest hardware and software tools.
3. **PSO3:** Enhance the abilities to qualify for employment, higher studies and research in Cyber Security with ethical values...

Program Educational Objectives (PEOs)

1. **PEO1:** To apply the basic skills in Engineering and have conceptual understanding to cater the needs of the industry.
2. **PEO2:** To understand the requirement of the society and provide solutions by applying the technical knowledge related to their field professionally.
3. **PEO3:** To enhance their programming skills with logical thinking for application design and deployment and be able to pursue higher studies and research.
4. **PEO4:** To communicate and apply the acquired engineering knowledge individually & as a team, for the successful implementation of projects.
5. **PEO5:** To understand moral, ethical, and social values to function as responsible citizens of society.

COURSE OBJECTIVE & COURSE OUTCOME

Sub Name: DATA ANALYTICS

Course Code: UCCSP211

Department of CSE (Cyber Security)

Course Objective:

o	

Course Outcome:

Sr.No	
1	CO1: Analyze the dataset and perform Descriptive Statistics.
2	CO2: Analyze the dataset and perform an Inferential Statistics.
3	CO3: Apply regression Analysis on the given dataset.
4	CO4: Create an interactive data visualization.

CO-PO Mapping Matrix and Attainment

Sub Name: DATA ANALYTICS

G. H. Raisoni College of Engineering & Management, Wagholi, Pu
(NAAC ACCREDITED)
1200 Gat No, Domkhel Road Wagholi , Pune-412207
Email:- ghrcem.pune@raisoni.net Web:- https://ghrcem.raisoni.net/

Department of CSE (Cyber Security)

Course Code: UCCSP211

se Outcome															

List of Practical

Sr.No	Title of Practical
1	Loading dataset and visualizing data

2	Producing descriptive statistics measures
3	Demonstrate handling of missing data.
4	Demonstrate Scatter Plot
5	Demonstrate Histogram Plot
6	Demonstrate Linear Regression for a given dataset
7	Demonstrate Multiple Linear Regression for a given dataset
8	Demonstrate Logistic Regression for a given dataset
9	Demonstrate Clustering for a given data
10	Demonstrate random Forest method for a given dataset
Open Ended Experiments / New Experiments	
11	Demonstration of data analysis on virtual Lab

Content Beyond Syllabus

Department of CSE (Cyber Security)

Sr.No	Content Beyond Syllabus Topic	CO
11	Demonstration of data analysis on virtual Lab	CO1

Subject Teacher **Dr. Deepika Ajalakar**
HOD (CS & DS)

Practical 1

Title: Loading dataset and visualizing data.

Aim:

Basic understanding of Python.

Basic understanding of Programming concepts.

Basic understanding of Data Structures.

Basic understanding of Data Types.

SOFTWARE REQUIREMENTS:

Windows10/ 11 and Ubuntu 14.04 / 14.10

Latest Python Versions

VS code/ PyCharm /Anaconda Spider/Jupiter Notebook

THEORY:

Data Visualization with Python

All over the world, a lot of data is being generated on a daily basis. And sometimes to analyze this data for certain trends, patterns may become difficult if the data is in its raw format. To overcome this data visualization comes into play. Data visualization provides a good, organized pictorial representation of the data which makes it easier to understand, observe, analyze. In this tutorial, we will discuss how to visualize data using Python. Python provides various libraries that come with different features for visualizing data. All these libraries come with different features and can support various types of graphs. In this tutorial, we will be discussing four such libraries.

Matplotlib

Seaborn

Database Used

Tips Database: - Tips database is the record of the tip given by the customers in a restaurant for two and a half months in the early 1990s, contains 6 columns such as total bill, tip, sex, smoker, day, time, size.

You can download the tips database from-

https://media.geeksforgeeks.org/wp-content/uploads/tips.csv

Example a Program: -

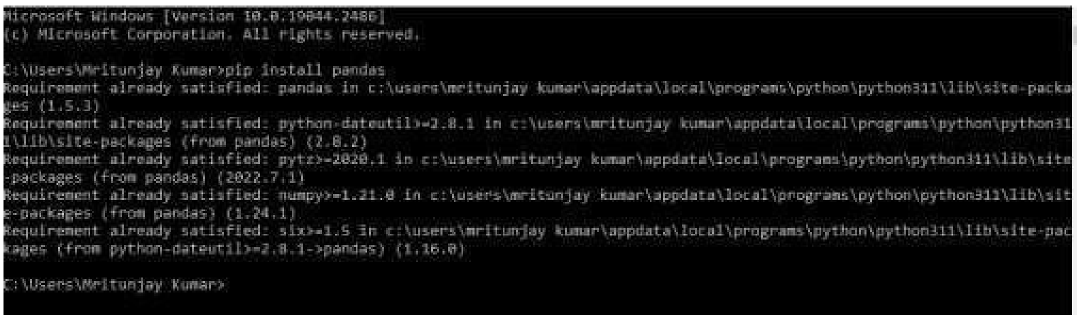
```
import pandas as pd
# reading the database
data = pd.read_csv("tips.csv")
# printing the top 10 rows
display(data.head(10))
```

Output:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
5	25.29	4.71	Male	No	Sun	Dinner	4
6	8.77	2.00	Male	No	Sun	Dinner	2
7	26.88	3.12	Male	No	Sun	Dinner	4
8	15.04	1.96	Male	No	Sun	Dinner	2
9	14.78	3.23	Male	No	Sun	Dinner	2

Matplotlib

Matplotlib is an easy-to-use, low-level data visualization library that is built on NumPy arrays. It consists of various plots like scatter plot, line plot, histogram, etc. Matplotlib provides a lot of flexibility. To install this type the below command in the terminal. Pip install matplotlib



```
Microsoft Windows [Version 10.0.19044.2486]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Mritunjay Kumar>pip install pandas
Requirement already satisfied: pandas in c:\users\mritunjay kumar\appdata\local\programs\python\python311\lib\site-packa
ges (1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\mritunjay kumar\appdata\local\programs\python\python31
1\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\mritunjay kumar\appdata\local\programs\python\python311\lib\site
-packages (from pandas) (2022.7.1)
Requirement already satisfied: numpy>=1.21.0 in c:\users\mritunjay kumar\appdata\local\programs\python\python311\lib\sit
e-packages (from pandas) (1.24.1)
Requirement already satisfied: six>=1.5 in c:\users\mritunjay kumar\appdata\local\programs\python\python311\lib\site-pac
kages (from python-dateutil>=2.8.1->pandas) (1.16.0)

C:\Users\Mritunjay Kumar>
```

After installing Matplotlib, let’s see the most commonly used plots using this library.

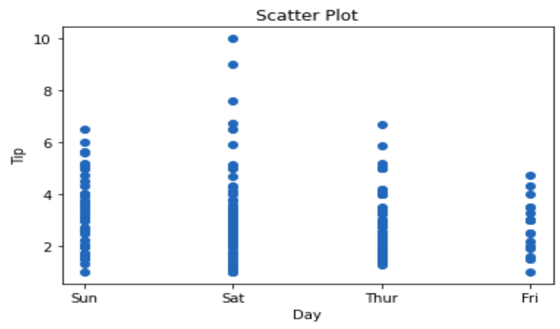
Scatter Plot

Scatter plots are used to observe relationships between variables and uses dots to represent the relationship between them. The **scatter()** method in the matplotlib library is used to draw a scatter plot.

Example a Program:

```
import pandas as pd
import matplotlib.pyplot as plt
# reading the database
data = pd.read_csv("tips.csv")
# Scatter plot with day against tip
plt.scatter(data['day'], data['tip'])
# Adding Title to the Plot
plt.title("Scatter Plot")
# Setting the X and Y labels
plt.xlabel('Day')
plt.ylabel('Tip')
plt.show()
```

Output:



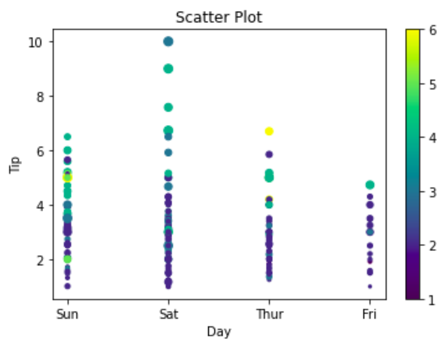
This graph can be more meaningful if we can add colors and also change the size of the points. We can do this by using the **c and s** parameter respectively of the scatter function. We can also show the color bar using the **colorbar()** method.

Example:

```
import pandas as pd
import matplotlib.pyplot as plt
# reading the database
data = pd.read_csv("tips.csv")
# Scatter plot with day against tip
```

```
plt.scatter(data['day'], data['tip'], c=data['size'],s=data['total_bill'])
# Adding Title to the Plot
plt.title("Scatter Plot")
# Setting the X and Y labels
plt.xlabel('Day')
plt.ylabel('Tip')
plt.colorbar()
plt.show()
```

Output:



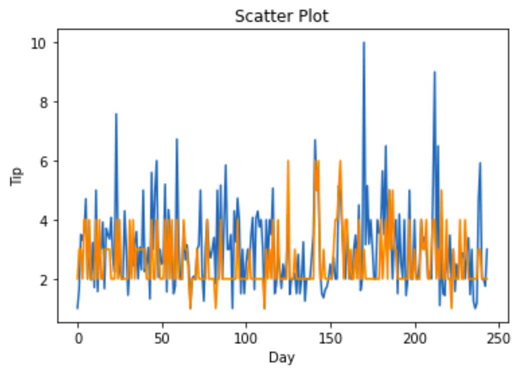
Line Chart

Line Chart is used to represent a relationship between two data X and Y on a different axis. It is plotted using the **plot()** function. Let’s see the below example.

Example:

```
import pandas as pd
import matplotlib.pyplot as plt
# reading the database
data = pd.read_csv("tips.csv")
# Scatter plot with day against tip
plt.plot(data['tip'])
plt.plot(data['size'])
# Adding Title to the Plot
plt.title("Scatter Plot")
# Setting the X and Y labels
plt.xlabel('Day')
plt.ylabel('Tip')
plt.show()
```

Output:



Bar Chart

A bar plot or bar chart is a graph that represents the category of data with rectangular bars with lengths and heights that is proportional to the values which they represent. It can be created using the **bar()** method.

Example:
```
import pandas as pd
import matplotlib.pyplot as plt
# reading the database
data = pd.read_csv("tips.csv")
# Bar chart with day against tip
plt.bar(data['day'], data['tip'])
plt.title("Bar Chart")
# Setting the X and Y labels
plt.xlabel('Day')
plt.ylabel('Tip')
# Adding the legends
plt.show()
```

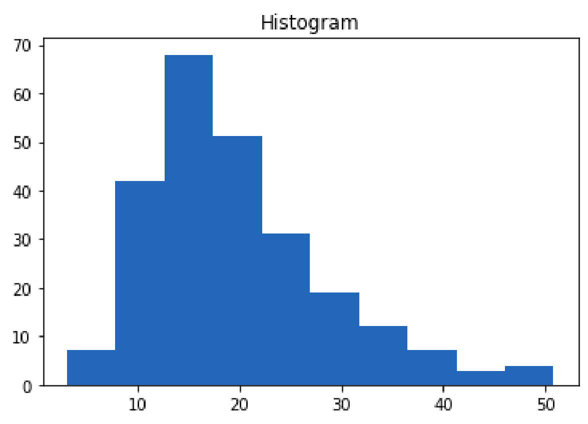
□ Histogram-

A histogram is basically used to represent data in the form of some groups. It is a type of bar plot where the X-axis represents the bin ranges while the Y-axis gives information about frequency. The **hist()** function is used to compute and create a histogram. In histogram, if we pass categorical data then it will automatically compute the frequency of that data i.e. how often each value occurred.

Example:

```
import pandas as pd
import matplotlib.pyplot as plt
# reading the database
data = pd.read_csv("tips.csv")
# histogram of total_bills
plt.hist(data['total_bill'])
plt.title("Histogram")
# Adding the legends
plt.show()
```

Output:

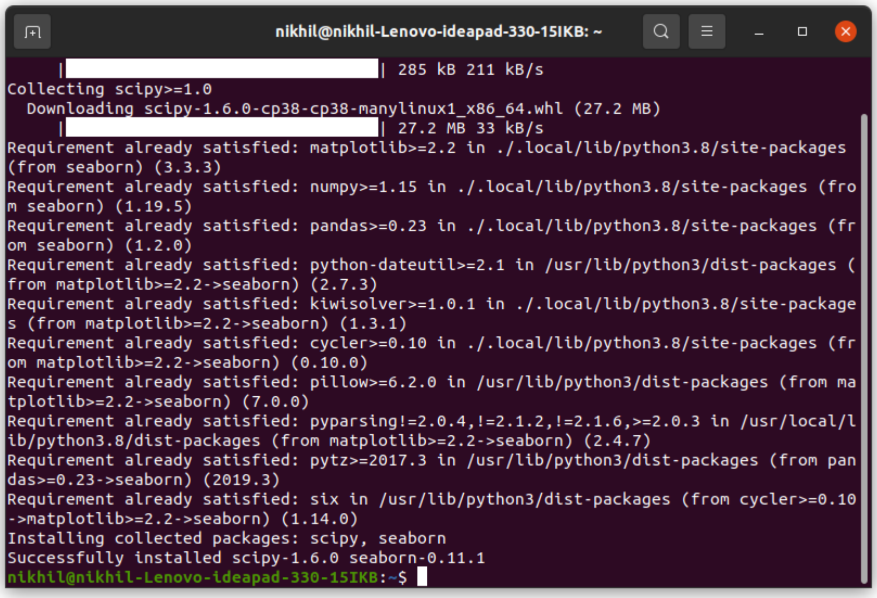


Seaborn

Seaborn is a high-level interface built on top of the Matplotlib. It provides beautiful design styles and color palettes to make more attractive graphs. To install seaborn type the below command in the terminal.

```
pip install seaborn
```

Department of CSE (Cyber Security)



```
     |████████████████████████████████| 285 kB 211 kB/s
Collecting scipy>=1.0
  Downloading scipy-1.6.0-cp38-cp38-manylinux1_x86_64.whl (27.2 MB)
     |████████████████████████████████| 27.2 MB 33 kB/s
Requirement already satisfied: matplotlib>=2.2 in ./.local/lib/python3.8/site-packages
(from seaborn) (3.3.3)
Requirement already satisfied: numpy>=1.15 in ./.local/lib/python3.8/site-packages (fro
m seaborn) (1.19.5)
Requirement already satisfied: pandas>=0.23 in ./.local/lib/python3.8/site-packages (fr
om seaborn) (1.2.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/lib/python3/dist-packages (
from matplotlib>=2.2->seaborn) (2.7.3)
Requirement already satisfied: kiwisolver>=1.0.1 in ./.local/lib/python3.8/site-package
s (from matplotlib>=2.2->seaborn) (1.3.1)
Requirement already satisfied: cycler>=0.10 in ./.local/lib/python3.8/site-packages (fr
om matplotlib>=2.2->seaborn) (0.10.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/lib/python3/dist-packages (from ma
tplotlib>=2.2->seaborn) (7.0.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in /usr/local/l
ib/python3.8/dist-packages (from matplotlib>=2.2->seaborn) (2.4.7)
Requirement already satisfied: pytz>=2017.3 in /usr/lib/python3/dist-packages (from pan
das>=0.23->seaborn) (2019.3)
Requirement already satisfied: six in /usr/lib/python3/dist-packages (from cycler>=0.10
->matplotlib>=2.2->seaborn) (1.14.0)
Installing collected packages: scipy, seaborn
Successfully installed scipy-1.6.0 seaborn-0.11.1
nikhil@nikhil-Lenovo-ideapad-330-15IKB:~$
```

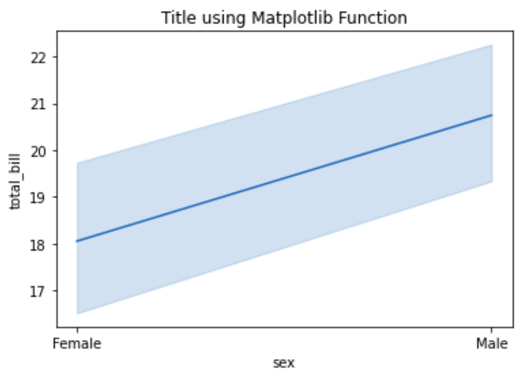
Seaborn is built on the top of Matplotlib, therefore it can be used with the Matplotlib as well. Using both Matplotlib and Seaborn together is a very simple process. We just have to invoke the Seaborn Plotting function as normal, and then we can use Matplotlib's customization function.

Note: Seaborn comes loaded with dataset such as tips, iris, etc. but for the sake of this tutorial we will use Pandas for loading these datasets.

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
# reading the database
data = pd.read_csv("tips.csv")
# draw lineplot
sns.lineplot(x="sex", y="total_bill", data=data)
# setting the title using Matplotlib
plt.title('Title using Matplotlib Function')
plt.show()
```

Output:



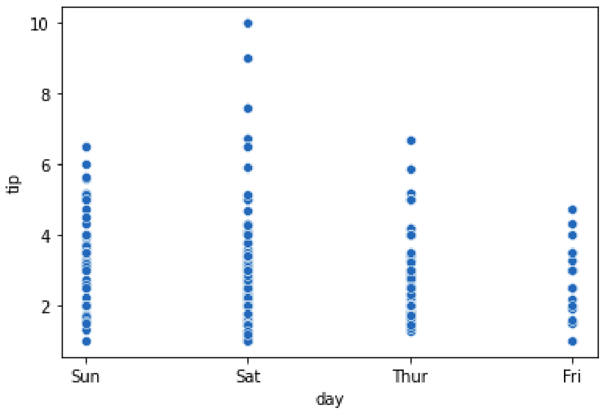
Scatter Plot

Scatter plot is plotted using the **scatterplot()** method. This is similar to Matplotlib, but additional argument data is required.

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
# reading the database
data = pd.read_csv("tips.csv")
sns.scatterplot(x='day', y='tip', data=data,)
plt.show()
```

Output:



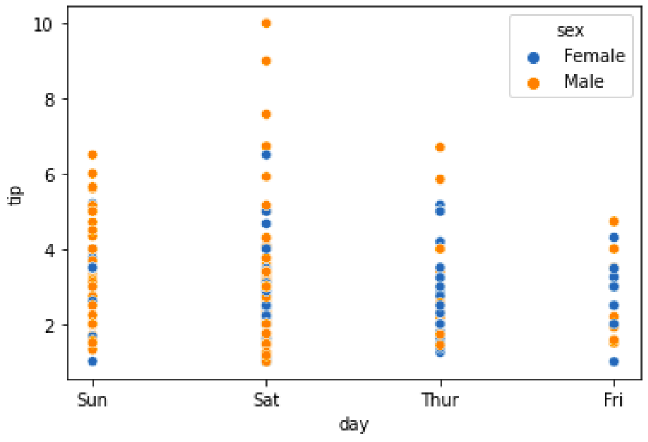
Department of CSE (Cyber Security)

You will find that while using Matplotlib it will a lot difficult if you want to color each point of this plot according to the sex. But in scatter plot it can be done with the help of hue argument.

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
# reading the database
data = pd.read_csv("tips.csv")
sns.scatterplot(x='day', y='tip', data=data, hue='sex')
plt.show()
```

Output:



Line Plot

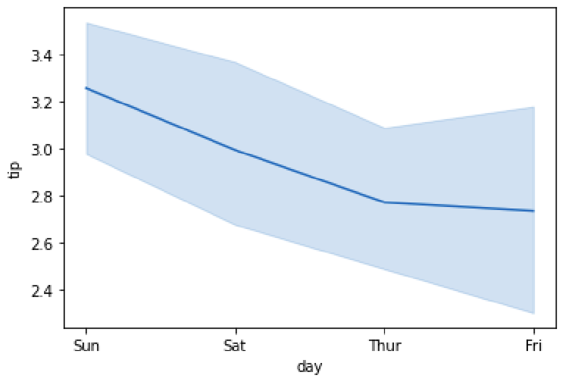
Line Plot in Seaborn plotted using the **lineplot()** method. In this, we can pass only the data argument also.

Example:

Department of CSE (Cyber Security)

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
# reading the database
data = pd.read_csv("tips.csv")
sns.lineplot(x='day', y='tip', data=data)
plt.show()
```

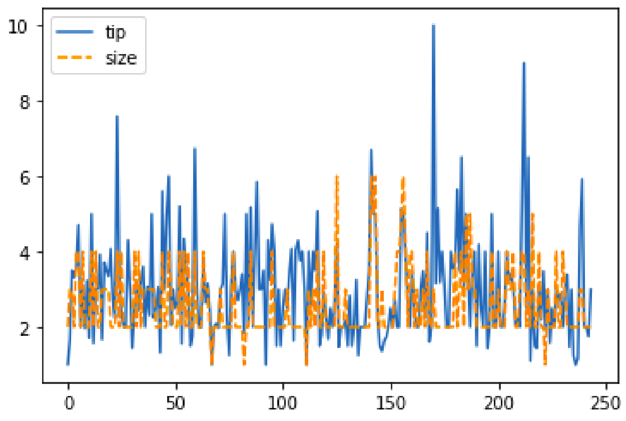
Output:



Example 2:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
# reading the database
data = pd.read_csv("tips.csv")
# using only data attribute
sns.lineplot(data=data.drop(['total_bill'], axis=1))
plt.show()
```

Output:

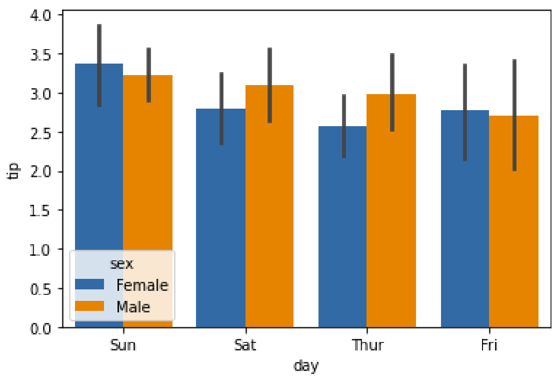


Bar Plot: Bar Plot in Seaborn can be created using the **barplot()** method.

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
# reading the database
data = pd.read_csv("tips.csv")
sns.barplot(x='day',y='tip', data=data,
            hue='sex')
plt.show()
```

Output:



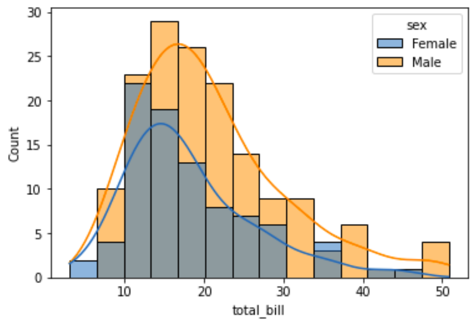
Histogram:

The histogram in Seaborn can be plotted using the **histplot()** function**.**

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
# reading the database
data = pd.read_csv("tips.csv")
sns.histplot(x='total_bill', data=data, kde=True, hue='sex')
plt.show()
```

Output:



After going through all these plots, you must have noticed that customizing plots using Seaborn is a lot more easier than using Matplotlib. And it is also built over matplotlib then we can also use matplotlib functions while using Seaborn

Department of CSE (Cyber Security)

Practical-2

Title: - Producing descriptive statistics measures

Theory: - Statistics, in general, is the method of collection of data, tabulation, and interpretation of numerical dat
It is an area of applied mathematics concern with data collection analysis, interpretation, and presentation. Wi
statistics, we can see how data can be used to solve complex problems.

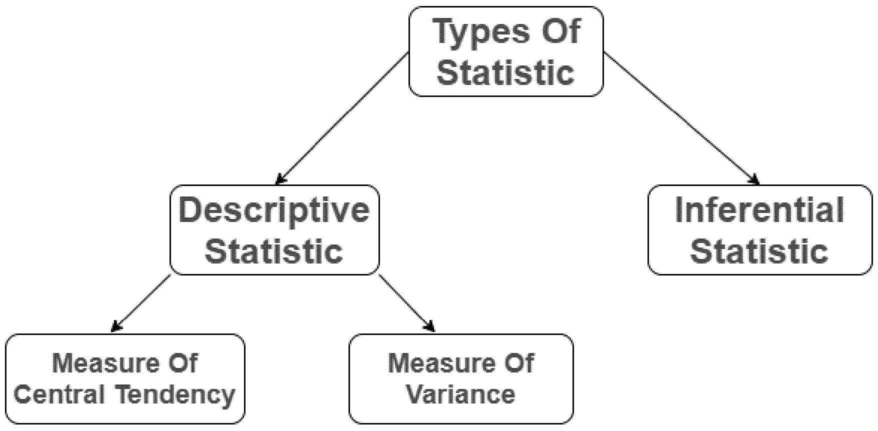
Understanding the Descriptive Statistics

In a layman's term, descriptive statistics generally means describing the data with the help of some representati
methods like charts, tables, Excel files, etc. The data is described in such a way that it can express some meaningf
information that can also be used to find some future trends. Describing and summarizing a single variable is calle
univariate analysis. Describing a statistical relationship between two variables is called bivariate analysi
Describing the statistical relationship between multiple variables is called multivariate analysis.

Types of Descriptive Statistics

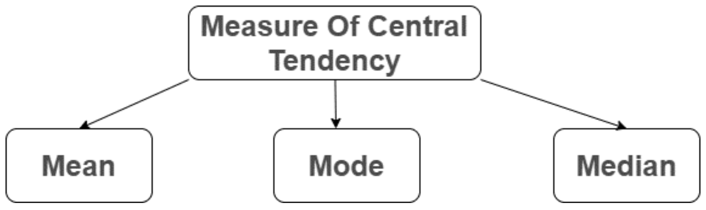
- Measures of Central Tendency
- Measure of Variability
- Measures of Frequency Distribution

Department of CSE (Cyber Security)



Measures of Central Tendency
It represents the whole set of data by a single value. It gives us the location of the central points. There are thre
main measures of central tendency:

- **Mean**
- **Mode**
- **Median**



Mean
It is the sum of observations divided by the total number of observations. It is also defined as average which is the sum divided by count.

$$\bar{x} = \frac{\Sigma x}{n}$$

where,

- x = Observations
- n = number of terms

Let's look at an example of how can we find the mean of a data set using Python code implementation.

Example:

```
import numpy as np
# Sample Data
arr = [5, 6, 11]
# Mean
mean = np.mean(arr)

print("Mean = ", mean)
```

Output:
Mean = 7.333333333333333

Mode
It is the value that has the highest frequency in the given data set. The data set may have no mode if the frequency o
all data points is the same. Also, we can have more than one mode if we encounter two or more data points havin
the same frequency.

Example

```
from scipy import stats

# sample Data
arr = [1, 2, 2, 3]

# Mode
mode = stats.mode(arr)
print("Mode = ", mode)
```

Output:

```
Mode = ModeResult(mode=array([2]), count=array([2]))
```

Median
It is the middle value of the data set. It splits the data into two halves. If the number of elements in the data set is od
then the center element is the median and if it is even then the median would be the average of two central elements

Example:

```
import numpy as np

# sample Data
arr = [1, 2, 3, 4]

# Median
median = np.median(arr)

print("Median = ", median)
```

Output:

```
Median = 2.5
```

Measure of Variability

Measures of variability are also termed measures of dispersion as it helps to gain insights about the dispersion or th spread of the observations at hand. Some of the measures which are used to calculate the measures of dispersion i the observations of the variables are as follows:

- Range
- Variance
- Standard deviation

Range

The range describes the difference between the largest and smallest data point in our data set. The bigger the rang the more the spread of data and vice versa.

Range = Largest data value – smallest data value

```
import numpy as np
# Sample Data
arr = [1, 2, 3, 4, 5]
# Finding Max
Maximum = max(arr)
# Finding Min
Minimum = min(arr)
# Difference Of Max and Min
Range = Maximum-Minimum
print("Maximum = {}, Minimum = {} and Range = {}".format(
    Maximum, Minimum, Range))
```

Output:

```
Maximum = 5, Minimum = 1 and Range = 4
```

Variance
It is defined as an average squared deviation from the mean. It is calculated by finding the difference between ever data point and the average which is also known as the mean, squaring them, adding all of them, and then dividing b the number of data points present in our data set.

$$\sigma^2 = \frac{\sum(x-\mu)^2}{N}$$

where,

- x -> Observation under consideration
- N -> number of terms
- μ -> Mean

Example:

```
import statistics

# sample data
arr = [1, 2, 3, 4, 5]
# variance
print("Var = ", (statistics.variance(arr)))
```

Output:
```
Var = 2.5
```

Standard Deviation
It is defined as the square root of the variance. It is calculated by finding the Mean, then subtracting each numb from the Mean which is also known as the average, and squaring the result. Adding all the values and then dividir by the no of terms followed by the square root.

$$\sigma = \sqrt{\frac{\sum(x-\mu)^2}{N}}$$

where,

- x = Observation under consideration
- N = number of terms
- μ = Mean

Example

```
import statistics
# sample data
arr = [1, 2, 3, 4, 5]
# Standard Deviation
print("Std = ", (statistics.stdev(arr)))
```

Output:

```
Std = 1.5811388300841898
```

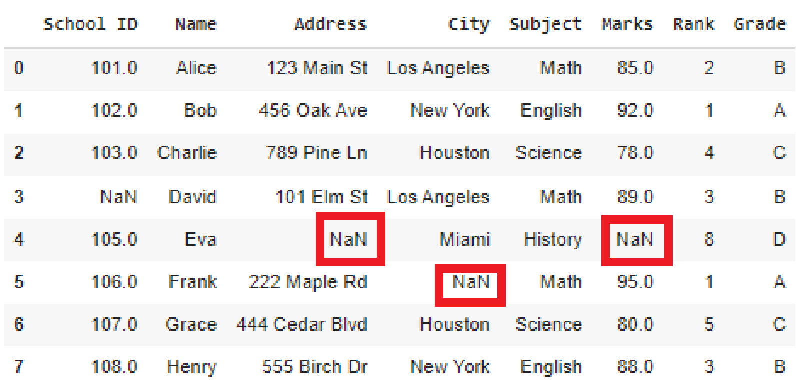
Practical-3

Title: - Demonstrate handling of missing data.

Theory:

What is a Missing Value?

Missing values are data points that are absent for a specific variable in a dataset. They can be represented in various ways, such as blank cells, null values, or special symbols like “NA” or “unknown.” These missing data points pose a significant challenge in data analysis and can lead to inaccurate or biased results.



	School ID	Name	Address	City	Subject	Marks	Rank	Grade
0	101.0	Alice	123 Main St	Los Angeles	Math	85.0	2	B
1	102.0	Bob	456 Oak Ave	New York	English	92.0	1	A
2	103.0	Charlie	789 Pine Ln	Houston	Science	78.0	4	C
3	NaN	David	101 Elm St	Los Angeles	Math	89.0	3	B
4	105.0	Eva	NaN	Miami	History	NaN	8	D
5	106.0	Frank	222 Maple Rd	NaN	Math	95.0	1	A
6	107.0	Grace	444 Cedar Blvd	Houston	Science	80.0	5	C
7	108.0	Henry	555 Birch Dr	New York	English	88.0	3	B

Missing Values

Missing values can pose a significant challenge in data analysis, as they can:

1. **Reduce the sample size:** This can decrease the accuracy and reliability of your analysis.
2. **Introduce bias:** If the missing data is not handled properly, it can bias the results of your analysis.

3. **Make it difficult to perform certain analyses:** Some statistical techniques require complete data for all variables, making them inapplicable when missing values are present
   Why Is Data Missing from the Dataset?
   Data can be missing for many reasons like technical issues, human errors, privacy concerns, data processing issues, or the nature of the variable itself. Understanding the cause of missing data helps choose appropriate handling strategies and ensure the quality of your analysis.
4. **It's important to understand the reasons behind missing data:**
5. **Identifying the type of missing data:** Is it Missing Completely at Random (MCAR), Missing at Random (MAR), or Missing Not at Random (MNAR)?
6. **Evaluating the impact of missing data:** Is the missingness causing bias or affecting the analysis?
7. **Choosing appropriate handling strategies:** Different techniques are suitable for different types of missing data.

Types of Missing Values

There are three main types of missing values:

1. **Missing Completely at Random (MCAR):** MCAR is a specific type of missing data in which the probability of a data point being missing is entirely random and independent of any other variable in the dataset. In simpler terms, whether a value is missing or not has nothing to do with the values of other variables or the characteristics of the data point itself.
2. **Missing at Random (MAR):** MAR is a type of missing data where the probability of a data point missing depends on the values of other variables in the dataset, but not on the missing variable itself. This means that the missingness mechanism is not entirely random, but it can be predicted based on the available information.
3. **Missing Not at Random (MNAR):** MNAR is the most challenging type of missing data to deal with. It occurs when the probability of a data point being missing is related to the missing value itself. This means that the reason for the missing data is informative and directly associated with the variable that is missing.

Methods for Identifying Missing Data

Locating and understanding patterns of missingness in the dataset is an important step in addressing its impact on analysis. Working with Missing Data in Pandas there are several useful functions for detecting, removing, and replacing null values in Pandas DataFrame.

Functions	Descriptions
`.isnull()`	Identifies missing values in a Series or DataFrame.
`.notnull()`	check for missing values in a pandas Series or DataFrame. It returns a boolean Series or DataFrame, where True indicates non-missing values and False indicates missing values.

Department of CSE (Cyber Security)

Functions	Descriptions
`.info()`	Displays information about the DataFrame, including data types, memory usage, and presence of missing values.
`.isna()`	similar to notnull() but returns True for missing values and False for non-missing values.
`dropna()`	Drops rows or columns containing missing values based on custom criteria.
`fillna()`	Fills missing values with specific values, means, medians, or other calculated values.
`replace()`	Replaces specific values with other values, facilitating data correction and standardization.
`drop_duplicates()`	Removes duplicate rows based on specified columns.
`unique()`	Finds unique values in a Series or DataFrame.

How Is a Missing Value Represented in a Dataset?
Missing values can be represented by blank cells, specific values like “NA”, or codes. It’s important to use consistent and documented representation to ensure transparency and facilitate data handling.
Common Representations

- **Blank cells:** Empty cells in spreadsheets or databases often signify missing data.
- **Specific values:** Special values like “NULL”, “NA”, or “-999” are used to represent missing data explicitly.
- **Codes or flags:** Non-numeric codes or flags can be used to indicate different types of missing values.

Effective Strategies for Handling Missing Values in Data Analysis

Department of CSE (Cyber Security)

Missing values are a common challenge in data analysis, and there are several strategies for handling them. Here's an overview of some common approaches:

1. Creating a Sample Dataframe

```
import pandas as pd

import numpy as np



# Creating a sample DataFrame with missing values

data = {

    'School ID': [101, 102, 103, np.nan, 105, 106, 107, 108],

    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva', 'Frank', 'Grace', 'Henry'],

    'Address': ['123 Main St', '456 Oak Ave', '789 Pine Ln', '101 Elm St', np.nan, '222
Maple Rd', '444 Cedar Blvd', '555 Birch Dr'],

    'City': ['Los Angeles', 'New York', 'Houston', 'Los Angeles', 'Miami', np.nan,
'Houston', 'New York'],

    'Subject': ['Math', 'English', 'Science', 'Math', 'History', 'Math', 'Science',
'English'],

    'Marks': [85, 92, 78, 89, np.nan, 95, 80, 88],

    'Rank': [2, 1, 4, 3, 8, 1, 5, 3],

    'Grade': ['B', 'A', 'C', 'B', 'D', 'A', 'C', 'B']

}

df = pd.DataFrame(data)

print("Sample DataFrame:")
print(df)
```

Output:

```
Sample Dataframe:
   School ID     Name          Address         City  Subject  Marks  Rank Grade
0      101.0    Alice      123 Main St  Los Angeles     Math   85.0     2     B
1      102.0      Bob      456 Oak Ave     New York  English   92.0     1     A
2      103.0  Charlie      789 Pine Ln      Houston  Science   78.0     4     C
3        NaN    David       101 Elm St  Los Angeles     Math   89.0     3     B
4      105.0      Eva              NaN        Miami  History    NaN     8     D
5      106.0    Frank     222 Maple Rd          NaN     Math   95.0     1     A
6      107.0    Grace   444 Cedar Blvd      Houston  Science   80.0     5     C
7      108.0    Henry     555 Birch Dr     New York  English   88.0     3     B
```

Removing Rows with Missing Values

- Simple and efficient: Removes data points with missing values altogether.
- Reduces sample size: Can lead to biased results if missingness is not random.
- Not recommended for large datasets: Can discard valuable information.
  In this example, we are removing rows with missing values from the original DataFrame (df) using the dropna() method and then displaying the cleaned DataFrame (df_cleaned).

```
# Removing rows with missing values

df_cleaned = df.dropna()

# Displaying the DataFrame after removing missing values

print("\nDataFrame after removing rows with missing values:")

print(df_cleaned)
```

Output:

```
DataFrame after removing rows with missing values:
   School ID     Name          Address         City  Subject  Marks  Rank Grade
0      101.0    Alice      123 Main St  Los Angeles     Math   85.0     2     B
1      102.0      Bob      456 Oak Ave     New York  English   92.0     1     A
2      103.0  Charlie      789 Pine Ln      Houston  Science   78.0     4     C
```

```
6      107.0    Grace  444 Cedar Blvd      Houston  Science   80.0     5     C
7      108.0    Henry    555 Birch Dr     New York  English   88.0     3     B
```

Imputation Methods

- Replacing missing values with estimated values.
- Preserves sample size: Doesn't reduce data points.
- Can introduce bias: Estimated values might not be accurate.

  Here are some common imputation methods:

1- Mean, Median, and Mode Imputation:

- Replace missing values with the mean, median, or mode of the relevant variable.
- Simple and efficient: Easy to implement.
- Can be inaccurate: Doesn't consider the relationships between variables.

  In this example, we are explaining the imputation techniques for handling missing values in the 'Marks' column of the DataFrame (df). It calculates and fills missing values with the mean, median, and mode of the existing values in that column, and then prints the results for observation.

1. **Mean Imputation:** Calculates the mean of the 'Marks' column in the DataFrame (df).
   - *df['Marks'].fillna(...):* Fills missing values in the 'Marks' column with the mean value.
   - *mean_imputation:* The result is stored in the variable mean_imputation.

2. **Median Imputation:** Calculates the median of the 'Marks' column in the DataFrame (df).
   - *df['Marks'].fillna(...):* Fills missing values in the 'Marks' column with the median value.
   - *median_imputation:* The result is stored in the variable median_imputation.

3. **Mode Imputation:** Calculates the mode of the 'Marks' column in the DataFrame (df). The result is a Series.
   - *.iloc[0]:* Accesses the first element of the Series, which represents the mode.
   - *df['Marks'].fillna(...):* Fills missing values in the 'Marks' column with the mode value.

```python
#  Mean, Median, and Mode Imputation

mean_imputation = df['Marks'].fillna(df['Marks'].mean())

median_imputation = df['Marks'].fillna(df['Marks'].median())

mode_imputation = df['Marks'].fillna(df['Marks'].mode().iloc[0])

print("\nImputation using Mean:")

print(mean_imputation)

print("\nImputation using Median:")

print(median_imputation)

print("\nImputation using Mode:")

print(mode_imputation)
```

Output:

```
Imputation using Mean:
0    85.000000
1    92.000000
2    78.000000
3    89.000000
4    86.714286
5    95.000000
6    80.000000
7    88.000000
Name: Marks, dtype: float64

Imputation using Median:
0    85.0
1    92.0
2    78.0
3    89.0
4    88.0
5    95.0
6    80.0
7    88.0
Name: Marks, dtype: float64
```

Department of CSE (Cyber Security)

```
Imputation using Mode:
0    85.0
1    92.0
2    78.0
3    89.0
4    78.0
5    95.0
6    80.0
7    88.0
Name: Marks, dtype: float64
```

2. Forward and Backward Fill

- Replace missing values with the previous or next non-missing value in the same variable.
- *Simple and intuitive:* Preserves temporal order.
- *Can be inaccurate:* Assumes missing values are close to observed values
  These fill methods are particularly useful when there is a logical sequence or order in the data, and missing values can be reasonably assumed to follow a pattern. The method parameter in fillna() allows to specify the filling strategy, and here, it's set to 'ffill' for forward fill and 'bfill' for backward fill.

1. **Forward Fill (forward_fill)**
   - *df['Marks'].fillna(method='ffill'):* This method fills missing values in the 'Marks' column of the DataFrame (df) using a forward fill strategy. It replaces missing values with the last observed non-missing value in the column.
   - *forward_fill:* The result is stored in the variable forward_fill.
2. **Backward Fill (backward_fill)**
   - *df['Marks'].fillna(method='bfill'):* This method fills missing values in the 'Marks' column using a backward fill strategy. It replaces missing values with the next observed non-missing value in the column.
   - *backward_fill:* The result is stored in the variable backward_fill.

```python
# Forward and Backward Fill

forward_fill = df['Marks'].fillna(method='ffill')

backward_fill = df['Marks'].fillna(method='bfill')

print("\nForward Fill:")

print(forward_fill)

print("\nBackward Fill:")

print(backward_fill)
```

Output:

```
Forward Fill:
0    85.0
```

```
1    92.0
2    78.0
3    89.0
4    89.0
5    95.0
6    80.0
7    88.0
Name: Marks, dtype: float64

Backward Fill:
0    85.0
1    92.0
2    78.0
3    89.0
4    95.0
5    95.0
6    80.0
7    88.0
Name: Marks, dtype: float64
```

Note

- Forward fill uses the last valid observation to fill missing values.
- Backward fill uses the next valid observation to fill missing values.

3. Interpolation Techniques

- Estimate missing values based on surrounding data points using techniques like linear interpolation or spline interpolation.
- More sophisticated than mean/median imputation: Captures relationships between variables.
- Requires additional libraries and computational resources.

These interpolation techniques are useful when the relationship between data points can be reasonably assumed to follow a linear or quadratic pattern. The method parameter in the interpolate() method allows to specify the interpolation strategy.

1. **Linear Interpolation**
   - *df['Marks'].interpolate(method='linear'):* This method performs linear interpolation on the 'Marks' column of the DataFrame (df). Linear interpolation estimates missing values by considering a straight line between two adjacent non-missing values.
   - *linear_interpolation*: The result is stored in the variable linear_interpolation.
2. **Quadratic Interpolation**
   - *df['Marks'].interpolate(method='quadratic'):* This method performs quadratic interpolation on the 'Marks' column. Quadratic interpolation estimates missing values by considering a quadratic curve that passes through three adjacent non-missing values.
   - *quadratic_interpolation:* The result is stored in the variable `quadratic_interpolation`.

```
#  Interpolation Techniques

linear_interpolation = df['Marks'].interpolate(method='linear')

quadratic_interpolation = df['Marks'].interpolate(method='quadratic')

print("\nLinear Interpolation:")

print(linear_interpolation)

print("\nQuadratic Interpolation:")

print(quadratic_interpolation)
```

Output:

```
Linear Interpolation:
0    85.0
1    92.0
2    78.0
3    89.0
4    92.0
5    95.0
6    80.0
7    88.0
Name: Marks, dtype: float64

Quadratic Interpolation:
0    85.00000
1    92.00000
2    78.00000
3    89.00000
4    98.28024
5    95.00000
6    80.00000
7    88.00000
Name: Marks, dtype: float64
```

Note:

- Linear interpolation assumes a straight line between two adjacent non-missing values.
- Quadratic interpolation assumes a quadratic curve that passes through three adjacent non-missing values.

Choosing the right strategy depends on several factors:

- Type of missing data: MCAR, MAR, or MNAR.
- Proportion of missing values.
- Data type and distribution.
- Analytical goals and assumptions.

Impact of Handling Missing Values

Missing values are a common occurrence in real-world data, negatively impacting data analysis and modeling if not addressed properly. Handling missing values effectively is crucial to ensure the accuracy and reliability of your findings.

Here are some key impacts of handling missing values:

1. **Improved data quality:** Addressing missing values enhances the overall quality of the dataset. A cleaner dataset with fewer missing values is more reliable for analysis and model training.
2. **Enhanced model performance:** Machine learning algorithms often struggle with missing data, leading to biased and unreliable results. By appropriately handling missing values, models can be trained on a more complete dataset, leading to improved performance and accuracy.
3. **Preservation of Data Integrity**: Handling missing values helps maintain the integrity of the dataset. Imputing or removing missing values ensures that the dataset remains consistent and suitable for analysis.
4. **Reduced bias:** Ignoring missing values may introduce bias in the analysis or modeling process. Handling missing data allows for a more unbiased representation of the underlying patterns in the data.
5. Descriptive statistics, such as means, medians, and standard deviations, can be more accurate when missing values are appropriately handled. This ensures a more reliable summary of the dataset.
6. **Increased efficiency:** Efficiently handling missing values can save you time and effort during data analysis and modeling.

AIM: Demonstrate Scatter Plotting.

THEORY:

What is Scatter Plot?

When we want to **build graphs and visualize the relationship** between two or more variables we make use of scatter plots in python. The scatter plot can be defined as a type of plot that illustrates the data as a collection of points or dots. The two-dimensional graph with an axis that is the x-axis and y-axis represents the position of a dot's data. The graph that is plotted for two sets of data along the two axes helps to visualize the relationship between the two or more core variables, that graph is defined as the scatter plot between the variables.

Here the information is gathered according to the movement of the data points along the two-axis whether the if the movement of data points is dependent on each other or not. Sometimes it is found that the data points are randomly arranged and distributed with no obvious pattern which depicts a lack of dependent relationship. Whenever you want to create the scatter plot in python, first import the matplotlib python library where you have two options to implement the scatter plot in python that is, via the pyplot.plot() or the pyplot.scatter() functions. While implementing, we can surely add more features to our scatter plots, such as changing the color, size, or even the shape of the data points.

So whats exactly is the difference between pyplot.scatter() vs pyplot.plot()?

Well, the major difference is when you work with pyplot.plot() then for any property we want to implement like color, shape, or size of data points it gets applied across all the data points present in the graph. While for pyplot.scatter(), we have control over each data point’s property like color, shape, and size of data points also. The below diagram demonstrates what a scatter plot in python looks like:

matplotlib.pyplot.scatter()

While studying the various library, we may have come across the matplotlib while working on the **dashboard and visualizations**.

The Matplotlib is an **extensive library** for building static, animated, or even interactive visualizations in Python. It is widely used for graph plotting various plots implemented in python (such as scatter plots, bar charts, pie charts, line plots, histograms, 3-D plots, etc.)

Out of all the methods explained in the matplotlib library, the scatter plots in python are widely implemented to visualize the relationship between variables (two or more depending on the number of variables). The scatter method uses the dots to demonstrate the relationship between the variables. We use the scatter() method from the matplotlib library to draw a scatter plot which helps to not only represent relations among variables but also provide significant information on if any change is brought in one can affect the other.

Syntax

The syntax for scatter plot in python is as follows:

```
matplotlib.pyplot.scatter(x_axis_data, y_axis_data, s=None, c=None, marker=None,
cmap=None, vmin=None, vmax=None, alpha=None, linewidths=None, edgecolors=None)
```

The syntax explains the various parameters that can be passed in the matplotlib.pyplot.scatter() function.

Parameters

Now as seen above we have various parameters that are passed while implementing the scatter() method in scatter plot in python:

x_axis_data- Represents the data in an array format to be presented on the x-axis.

y_axis_data- Represents the data in an array format to be presented on the y-axis.

s- Represent the marker size. It could be a scalar or an array of sizes equal to the size of x or y.

c- Represent the color for the sequence of colors dedicated to markers. **marker-** Represent the marker style. The various types of markers that we can use while creating a scatter plot in python are: **['<‘,’s’,’p’,’H’,’D’,’d’,‘.’,’o’,’v’,’h’,’1′,”,”,’^’,’>’,’*’]**

cmap- Represent the cmap name **linewidths-** Represents the width of the marker border **edge color-** Represents the border color of the marker **alpha-** Represents the blending value usually lying between 0 (denoting transparent) and 1 (denoting opaque)

Note: All the above parameters **except x_axis_data and y_axis_data** are optional. If they are not explicitly mentioned then their default value is taken to be None.

Examples

Let us explore more scenarios where we shall be implementing the concepts around scatter plots in python. The below examples cover the scenario description along with code examples and explanations capturing it with visuals as well.

Let's dive in!

A Scatter Plot with 1000 Dots

Now let us start with a very basic scatter plot where we shall be creating a scatter plot in python with 1000 dots.

Department of CSE (Cyber Security)

We plot the graph to showcase how we can allot 1000 data points as a scatter plot with the below-given code.

Code:

```
import sys
import matplotlib
matplotlib.use('Agg')

# importing the matpotlib library

import numpy
import matplotlib.pyplot as plot

# importing numpy library and pyplot


x = numpy.random.normal(9.0, 3.0, 1000)
y = numpy.random.normal(18.0, 4.0, 1000)
# mentioning the x_axis_data, y_axis_data along with 1000 data dots

plot.scatter(x, y)
plot.show()
# creating the scatter plot

#Enabling the compiler to be able to draw:
plot.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

Output:

The output for the above code is as below:

Explanation:

Here we have created a scatter plot plotting 1000 dots at once. This is a very easy-to-understand plot where we simply made use of the scatter plot syntax and defining value as shown in code to display the scatter plot in python with 1000 data points at once.

Scatter Plot with Different Shapes and Colors for Two Datasets

Now let's move to create a scatter plot where we shall be implementing the scenario where we shall learn how to draw a scatter plot in python with different shapes and colors for two datasets.

We shall be starting by importing the package and moving on to defining the two datasets (the values of their coordinates) with their unique marker, color, and edge color. Finally, we shall be using the scatter function to plot the graph.

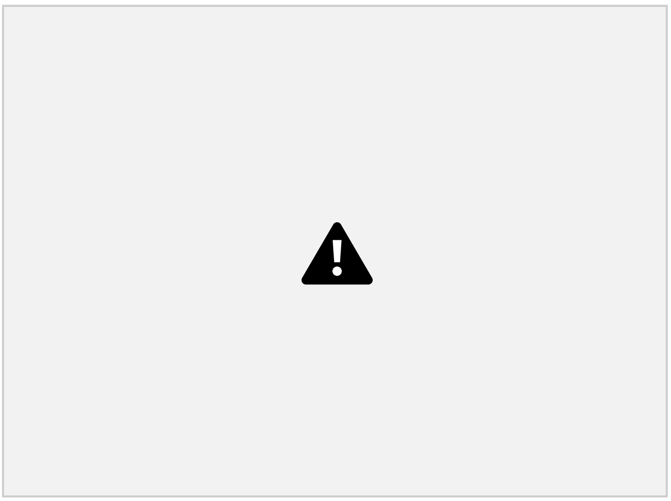
Code:

```
import matplotlib.pyplot as plot
# importing the required library.

# Defining the first dataset
x_1 = [91, 42, 86, 95, 32, 16,
      69, 24, 18, 50]

y_1 = [71, 6, 43, 93, 87, 63,
      93, 73, 19, 10]

# Defining the second dataset
x_2 = [18, 99, 28, 70, 16, 9,
      56, 65, 32, 82]

y_2 = [26, 78, 90, 33, 8,
      23, 56, 12, 57, 45]

plot.scatter(x_1, y_1, c ="blue",
            linewidths = 2,
            marker ="s",
            edgecolor ="pink",
            s = 70)

# Defining the color of the first data points as blue with its edge color like pink.

plot.scatter(x_2, y_2, c ="red",
            linewidths = 2,
            marker ="^",
            edgecolor ="black",
            s = 300)

# Defining the color of the first data points as red with its edge color as black.
plot.xlabel("X-axis")
```

```
# Labeling the x-axis
plot.ylabel("Y-axis")
# Labeling the y-axis
plot.show()
```

Output:

The output for the above code is as below:



Explanation:

Here we are plotting the Scatter plot with different shapes and colors for two datasets. We have first started by giving the various positions to the dots defining their x-axis and y-axis. Then by making use of different parameters we bring the different shapes and colors to our visualizations.

Correlation with Scatter Plot

Now let us understand one very important concept of scatter plot in python which is, Correlation with Scatter plot in python. There are three major correlations in scatter plots in python as defined below:

1. **Positive correlation:** When the value of data points along the y-axis starts to increase with respect to the value of data points along the x-axis, then the variables are said to possess a positive correlation.
2. **Negative correlation:** When the value of data points along the y-axis starts to decrease concerning the value of data points along the x-axis, then the variables are said to possess a negative correlation.
3. **Zero correlation:** When the value of data points along the y-axis started to change randomly independent of the value of data points at the x-axis, then the variables are said to possess zero correlation signifying both the data sets are **independent of each other**.

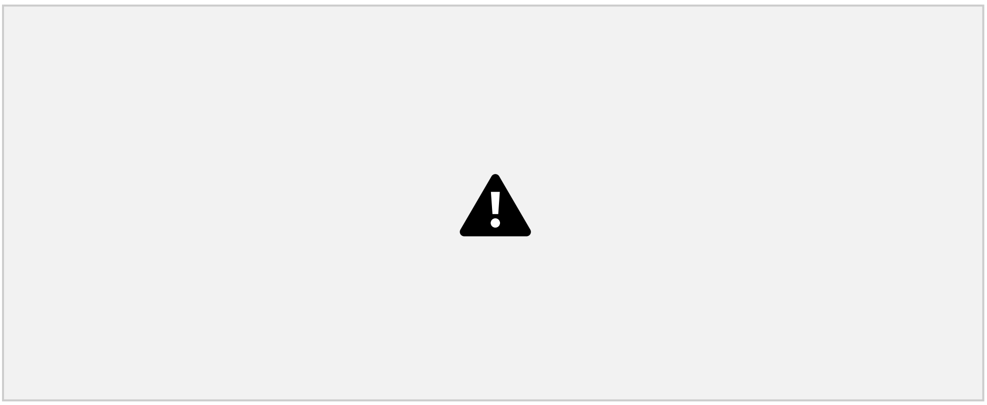
Let us create a scatter plot in python to visualize the same as learned above.

Code:

```
# Correlations with Scatter plot in python
import numpy as np
import matplotlib.pyplot as plot
# importing the required packages.

x=np.random.randn(200)
y1= x*6 +3
y2= -3*x
y3=np.random.randn(200)

# Plotting the different correlations
plot.rcParams.update({'figure.figsize':(15,5), 'figure.dpi':200})
plot.scatter(x, y1, label=f'y1 correlation = {np.round(np.corrcoef(x,y1)[0,2], 4)}')
plot.scatter(x, y2, label=f'y2 correlation = {np.round(np.corrcoef(x,y2)[0,2], 4)}')
plot.scatter(x, y3, label=f'y3 correlation = {np.round(np.corrcoef(x,y3)[0,2], 4)}')

plot.title('Correlations with Scatter plot in python')
# giving the graph a title.
plot.legend()
plot.show()
```

Output:

The output for the above code is as below:



Explanation:

As seen above, we are creating a scatter plot in python to understand the relationship between the data points that are represented by the data dots having a position with the x-axis and y-axis. Here the f'y1 correlation denoted the positive correlations, the f'y2 correlation depicted the negative correlation, and the f'y3 correlation represented the zero correlation.

Changing the Color of Groups of Points

Let us learn to draw a scatter plot in python where we want to change the color of groups of points. To do so, we make use of the color =" command to understand how we can be changing the color of groups of points while representing a scatter plot in python by simplifying passing the color function (as a parameter) with any random color of your choice.

Code:

```
# Changing the color of groups of points
import numpy as np
import matplotlib.pyplot as plot
# importing the required packages.
x = np.random.randn(50)
y1 = np.random.randn(50)
y2= np.random.randn(50)

# Plot the graph by assigning different colours
plot.scatter(x,y1,color='green')
plot.scatter(x,y2,color= 'blue')
plot.rcParams.update({'figure.figsize':(10,8), 'figure.dpi':100})

# Giving  the graph a title.
plot.title('Changing the colour of groups of points')
plot.xlabel('X - value')
# labels the x-axis
plot.ylabel('Y - value')
# labels the y-axis
plot.show()
```

Output: The output for the above code is as below:

Explanation:

As can be seen above, we have plotted a scatter plot in python where we are Changing the color of groups of points. we made use of the color =" command to assign different colors to the data points or dots to make the graph easy to interpret and understand.

Changing the Color and Marker

Let us learn to draw a scatter plot in python where we want to Change the Color and Marker of data points. To do so, we make use of the marker =_ command to understand how we can be Changing the Color and Marker while representing a scatter plot in python.

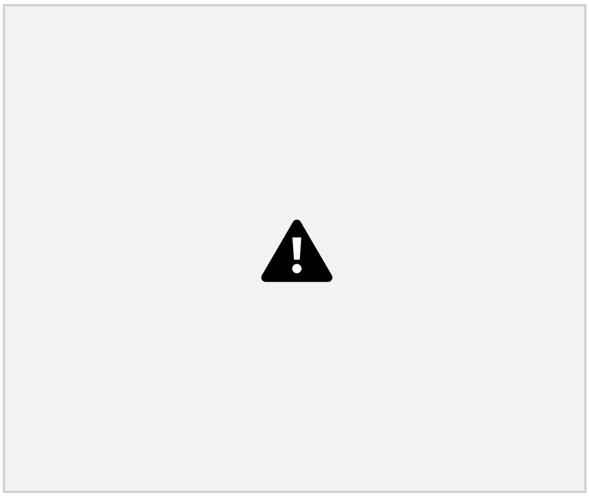
The various types of markers that we can use while creating a scatter plot in python are as below: **['<‘,’s’,’p’,’H’,’D’,’d’,‘.’,’o’,’v’,’h’,’1′,”,”,’^’,’>’,’*’]**

Code:

```
# Scatterplot in python with changing Color and marker.

import numpy as np
import matplotlib.pyplot as plot

# importing the required packages.

x = np.random.randn(200)
y1 = np.random.randn(200)
y2 = np.random.chisquare(5, 200)
y3 = np.random.poisson(3, 200)

# Plot
plot.rcParams.update({'figure.figsize':(5,4), 'figure.dpi':50})
plot.scatter(x,y1,color='pink', marker= '*', label='Standard')
plot.scatter(x,y2,color= 'black', marker='v', label='Square')
plot.scatter(x,y3,color= 'green', marker='.', label='Poisson')
# labelling different colour and marker spread across the scatter plot graph

plot.title('Changing the Color and Marker')
# helps to give the chart a title.
plot.xlabel('X - value')
# labels the x-axis
plot.ylabel('Y - value')
# labels the y-axis
plot.legend(loc='best')
plot.show()
```

Output:

The output for the above code is as below:



Explanation:

Here as seen above, we are creating a scatter plot to understand how we can be Changing the Color and Marker of the same. We used the scatter plot syntax to specify the color and marker for the different variables represented by the data points.

Scatter Plot with Linear Fit Plot Using Seaborn

Let us see how we can create a scatter plot with a linear fit plot using seaborn. We shall be making use of the lmplot() function in seaborn.

We shall be referring to the mtcars dataset from below links:

https://www.kaggle.com/ruiromanini/mtcars/download

The below code explains the step-by-step process of how we can know after the graph is plotted whether the relationship between the two data sets is a linear fit relationship or not between the mpg and the disp column from the mtcars dataset.

Code:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plot

# importing the required packages.
link                                                                          =
'https://gist.githubusercontent.com/seankross/a412dfbd88b3db70b74b/raw/5f23f993cd87c283ce7
66e7ac6b329ee7cc2e1d1/mtcars.csv'
df=pd.read_csv(link)
# using pandas and reading the data from the link given above
plot.rcParams.update({'figure.figsize':(10,8), 'figure.dpi':100})
sns.lmplot(x='mpg', y='disp', data=df);
```

```
#making use of the lmplot function from seaborn package to display the scatter plot
# marking the x axis as 'mpg' and y axis as 'disp'
```

Output: The output for the above code is as below:

Explanation:

As can be seen above, we have plotted the Scatter Plot in python with a Linear fit plot using the Seaborn package. We used the sns.lmplot() function from the seaborn package to find out the linear relationship between the mpg and the disp column from the mtcars dataset. Here we marked the x-axis as 'mpg' and the y-axis as 'disp' along with data as the data frame.

Scatter Plot with Histograms Using Seaborn

Let us move forward to explore how we can create scatter plots with histograms using seaborn. We shall be using the joint plot function in the seaborn package to represent the distribution of both x and y values as histograms through the scatter plot in python for which we first import the seaborn package.

We shall be using the sns.jointplot() function with x, y, and dataset as arguments.

Code:

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

# importing all required library

x = np.random.randn(100)
y1 = np.random.randn(100)
```

```
# marking the x and y axis with random.randn as 100.
plot.rcParams.update({'figure.figsize':(10,8), 'figure.dpi':100})

sns.jointplot(x=x,y=y1);
# making use of joinplot to plot scatter plot iwth histograms
```

Output: The output for the above code is as below:

Explanation:

As seen above, we are creating a Scatter Plot in python with Histograms using the seaborn library. Here we are getting the distribution plot for the x and y value as histograms using the sns.jointplot() function from the **seaborn package**.

Bubble Plot

An interesting concept around the scatter plot is the bubble plot.

The bubble plot can be defined as the type of scatterplot where we can include a third dimension or a third variable. The size of the data dots or points represents the value given to this third variable.

Now to represent the bubble plot, the third variable representing the size of the data dots is added in the scatter plot in python denoted by **s** that explains the size of the data points.

Code:

```
import numpy as np
import matplotlib.pyplot as plot
# The type of sctter plot : Bubble Plot
x = np.random.rand(200)
y = np.random.rand(200)
# marking the x and y axis with random.rand as 200.
s = np.random.rand(200)*100
```

```
# s will represent the size of the bubble that represents the scatter plot
plot.scatter(x, y, s=s,color='blue')
# plotting a scatter plot with bubble colored as blue
plot.show()
```

Output: The output for the above code is as below:

Explanation:

As seen above, we have added a third variable to the two-dimensional scatter plot in python. Here the third dimension is represented by the variable 's' whose value demonstrates the size of the dots along with the blue color of the dots as can be seen in the output.