Unit 1

1. Explain the importance of data analytics in decision-making across industries.

Importance of Data Analytics

- ❖ Data Analytics has a key role in improving your business as it is used to gather hidden insights, generate reports, perform market analysis, and improve business requirements.
- ❖ Data analytics is important because it helps businesses optimize their performances. Implementing it into the business model means companies can help reduce costs by identifying more efficient ways of doing business and by storing large amounts of data
- ❖ Data analysis helps businesses acquire relevant, accurate information, suitable for developing future marketing strategies, business plans, and realigning the company's vision or mission.
- ❖ Data analytics is important to understand trends and patterns from the massive amounts of data that are being collected. It helps optimize business performance, forecast future results, understand audiences, and reduce costs.
- ❖ Data analytics is the process of examining, transforming, and interpreting data to uncover meaningful patterns, insights, and trends. It is important because it enables organizations to make data-driven decisions, improve operational efficiency, identify opportunities, and gain a competitive edge.
- ❖ Data analytics gives a company a cost-effective solution. Even it will help them to improvise the decisions making process. The sectors like manufacturing, media, healthcare, real estate, and others require data analytics techniques and tools
- ❖ The scope of data analytics science involves extracting insights and patterns from data to drive decision-making and gain a competitive advantage. Data Analytics is a powerful discipline that transforms raw data into actionable insights, driving informed decision-making and fostering innovation across industries.
- ❖ Data analytics future growth is expected to continue to the increasing amount of data being generated, the growing importance of data-driven decision-making in businesses, and the continued development of big data and AI technologies
- ❖ Data analytics is crucial for informed decision-making across industries as it allows organizations to extract meaningful insights from vast amounts of data, enabling them to

identify trends, predict future outcomes, optimize operations, understand customer behavior, manage risks, and ultimately make more accurate and strategic choices, leading to improved profitability and competitive advantage.

Key benefits of data analytics in decision-making:

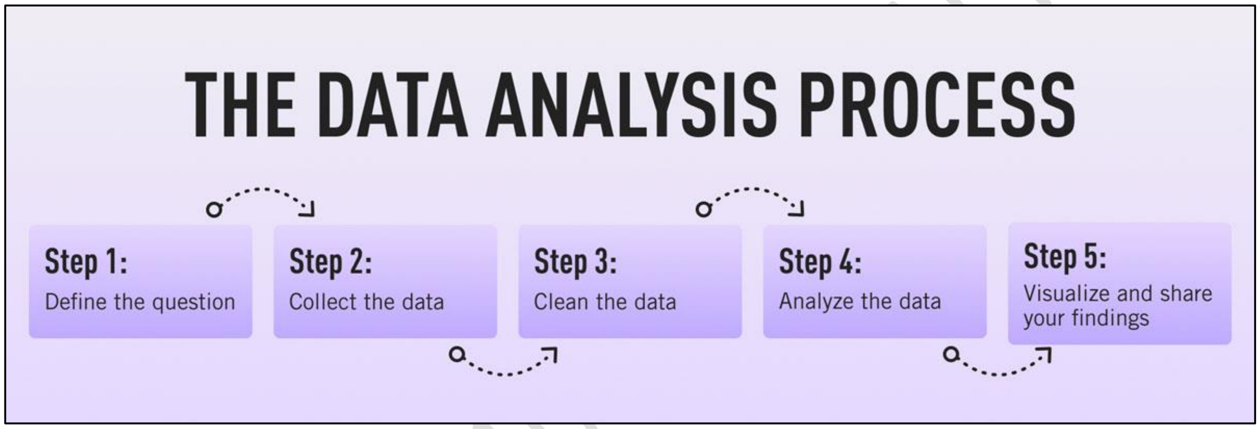
- **Deeper customer understanding:**
- Analyzing customer data helps businesses identify preferences, buying patterns, and pain points, allowing them to tailor products and services to better meet customer needs, increasing satisfaction and loyalty.
- **Improved decision accuracy:**
- By relying on objective data and insights rather than intuition, businesses can make more informed decisions with higher accuracy.
- **Risk management and forecasting:**
- Data analytics can identify potential risks and opportunities by analyzing historical data and market trends, allowing companies to proactively develop mitigation strategies.
- **Operational efficiency:**
- Analyzing operational data can reveal areas for improvement, leading to optimized processes, cost reductions, and increased productivity.
- **New product and service development:**
- By understanding customer needs through data analysis, companies can identify gaps in the market and develop new products and services that better cater to customer demands.
- **Targeted marketing campaigns:**
- Data analytics enables businesses to segment customers based on demographics and behaviors, allowing for more targeted and effective marketing campaigns.

Applications across industries:

- Retail: Predicting customer demand, optimizing inventory levels, personalized product recommendations based on customer data.
- Finance: Fraud detection, credit risk assessment, investment portfolio optimization
- Healthcare: Patient outcome analysis, identifying high-risk populations, optimizing treatment plans
- Manufacturing: Predictive maintenance, production optimization, quality control

- Marketing: Customer segmentation, campaign performance analysis, lead generation

2. **Design a data analytics workflow for a company aiming to improve customer satisfaction. Highlight the steps and tools involved.**
   - Like any scientific discipline, data analysis follows a rigorous step-by-step process. Each stage requires different skills and know-how.
   - To get meaningful insights, though, it's important to understand the process as a whole. An underlying framework is invaluable for producing results that stand up to scrutiny.


THE DATA ANALYSIS PROCESS
Step 1: Define the question
Step 2: Collect the data
Step 3: Clean the data
Step 4: Analyze the data
Step 5: Visualize and share your findings

Step 1: Define the question(s) you want to answer

- In the first step of process the data analyst is given a problem/business task. The analyst has to understand the task and the stakeholder's expectations for the solution.
- A stakeholder is a person that has invested their money and resources to a project. The analyst must be able to ask different questions in order to find the right solution to their problem.
- The analyst has to find the root cause of the problem in order to fully understand the problem.
- The analyst must make sure that he/she doesn't have any distractions while analyzing the problem.
- Communicate effectively with the stakeholders and other colleagues to completely understand what the underlying problem is.

- At this stage, you’ll take a clearly defined problem and come up with a relevant question or hypothesis you can test.
- Questions to ask yourself for the Ask phase are:
- What are the problems that are being mentioned by my stakeholders?
- What are their expectations for the solutions?

Step 2: Collect the data

- The second step is to prepare or Collect the Data. This step includes collecting data and storing it for further analysis.
- The analyst has to collect the data based on the task given from multiple sources. The data has to be collected from various sources, internal or external sources.
- Internal data is the data available in the organization that you work for while external data is the data available in sources other than your organization.
- The data that is collected by an individual from their own resources is called first-party data.
- The data that is collected and sold is called second-party data.
- Data that is collected from outside sources is called third-party data. The common sources from where the data is collected are Interviews, Surveys, Feedback, Questionnaires. The collected data can be stored in a spreadsheet or SQL database.
- Data analysts will usually gather structured data from primary or internal sources, such as CRM software or email marketing tools.

Step 3: Clean the data

- The third step is Clean and Process Data. After the data is collected from multiple sources, it is time to clean the data.
- Clean data means data that is free from misspellings, redundancies, and irrelevance.
- Clean data largely depends on data integrity. There might be duplicate data or the data might not be in a format, therefore the unnecessary data is removed and cleaned.
- There are different functions provided by SQL and Excel to clean the data. This is one of the most important steps in Data Analysis as clean and formatted data helps in finding trends and solutions.

- The most important part of the Process phase is to check whether your data is biased or not. Bias is an act of favoring a particular group/community while ignoring the rest. Biasing is a big no-no as it might affect the overall data analysis. The data analyst must make sure to include every group while the data is being collected.
- Your original dataset may contain duplicates, anomalies, or missing data which could distort how the data is interpreted, so these all need to be removed.
- Data cleaning can be a time-consuming task, but it’s crucial for obtaining accurate results.
- Ex. Field extraction algorithm, Clustering.

Step 4: Analyze the data

- The fourth step is to Analyze. The cleaned data is used for analyzing and identifying trends.
- It also performs calculations and combines data for better results. The tools used for performing calculations are Excel or SQL.
- These tools provide in-built functions to perform calculations or sample code is written in SQL to perform calculations.
- Using Excel, we can create pivot tables and perform calculations while SQL creates temporary tables to perform calculations.
- Programming languages are another way of solving problems. They make it much easier to solve problems by providing packages. The most widely used programming languages for data analysis are R and Python.
- Some common techniques include regression analysis, cluster analysis, and time-series analysis.
- This step in the process also ties in with the four different types of analysis we looked at in (Descriptive, Diagnostic, Predictive, and Prescriptive).

Step 5: Interpret and share the results

- This final step in the process is where data is transformed into valuable business insights.
- Depending on the type of analysis conducted, you’ll present your findings in a way that others can understand—in the form of a chart or graph.
- The data now transformed has to be made into a visual (chart, graph).
- The reason for making data visualizations is that there might be people, mostly stakeholders that are non-technical.

- Visualizations are made for a simple understanding of complex data. Tableau and Looker are the two popular tools used for compelling data visualizations.
- Tableau is a simple drag and drop tool that helps in creating compelling visualizations.
- Looker is a data viz tool that directly connects to the database and creates visualizations.
- Tableau and Looker are both equally used by data analysts for creating a visualization.
- R and Python have some packages that provide beautiful data visualizations.
- R has a package named ggplot which has a variety of data visualizations.
- A presentation is given based on the data findings.
- Sharing the insights with the team members and stakeholders will help in making better decisions.
- It helps in making more informed decisions and it leads to better outcomes.
- Presenting the data involves transforming raw information into a format that is easily comprehensible and meaningful for various stakeholders.
- This process includes the creation of visual representations, such as charts, graphs, and tables, to effectively communicate patterns, trends, and insights collected from the data analysis.
- The goal is to facilitate a clear understanding of complex information, making it accessible to both technical and non-technical audiences.
- Effective data presentation involves thoughtful selection of visualization techniques based on the nature of the data and the specific message planned.
- It goes beyond mere display to storytelling, where the presenter interprets the findings, highlights key points, and guides the audience through the narrative that the data unfolds.
- Whether through reports, presentations, or interactive dashboards, the art of presenting data involves balancing simplicity with depth, ensuring that the audience can easily grip the significance of the information presented and use it for informed decision-making.

3. Illustrate Python Libraries for Data Visualization with suitable examples

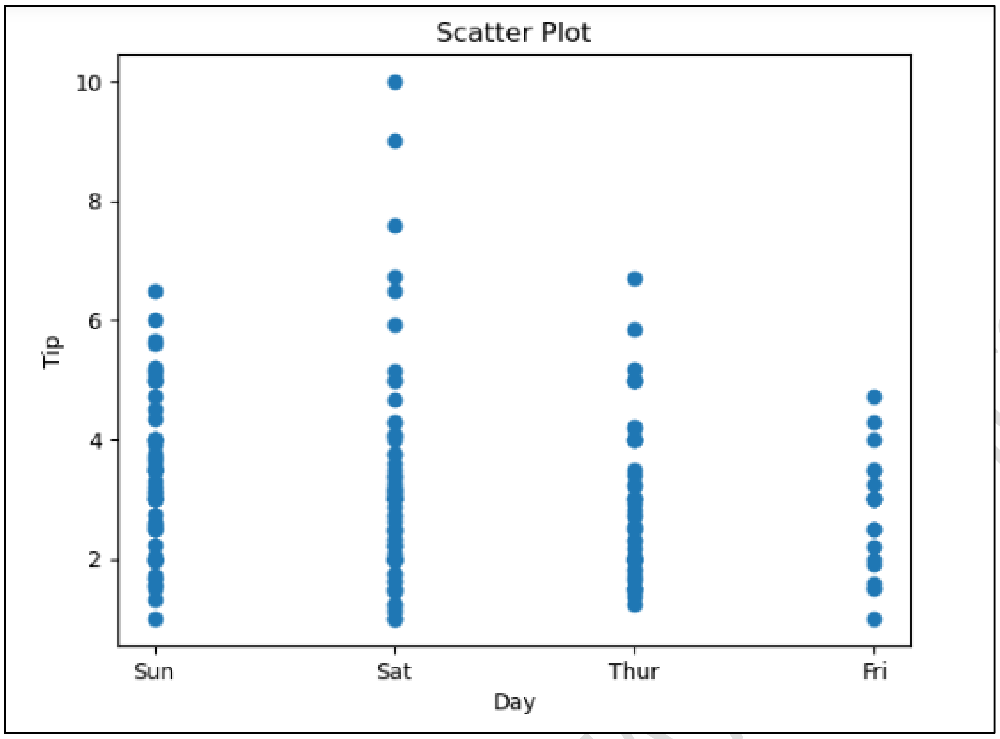
a) Matplotlib

b) Seaborn

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
- ❖ 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:

1. import pandas as pd
2. import matplotlib.pyplot as plt
3. reading the database
4. data = pd.read_csv("tips.csv")
5. Scatter plot with day against tip
6. plt.scatter(data['day'], data['tip'])
7. Adding Title to the Plot
8. plt.title("Scatter Plot")
9. Setting the X and Y labels
10. plt.xlabel('Day')
11. plt.ylabel('Tip')
12. plt.show()



2. 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:

1. import pandas as pd
2. import matplotlib.pyplot as plt

# reading the database

1. data = pd.read_csv("tips.csv")

# Scatter plot with day against tip

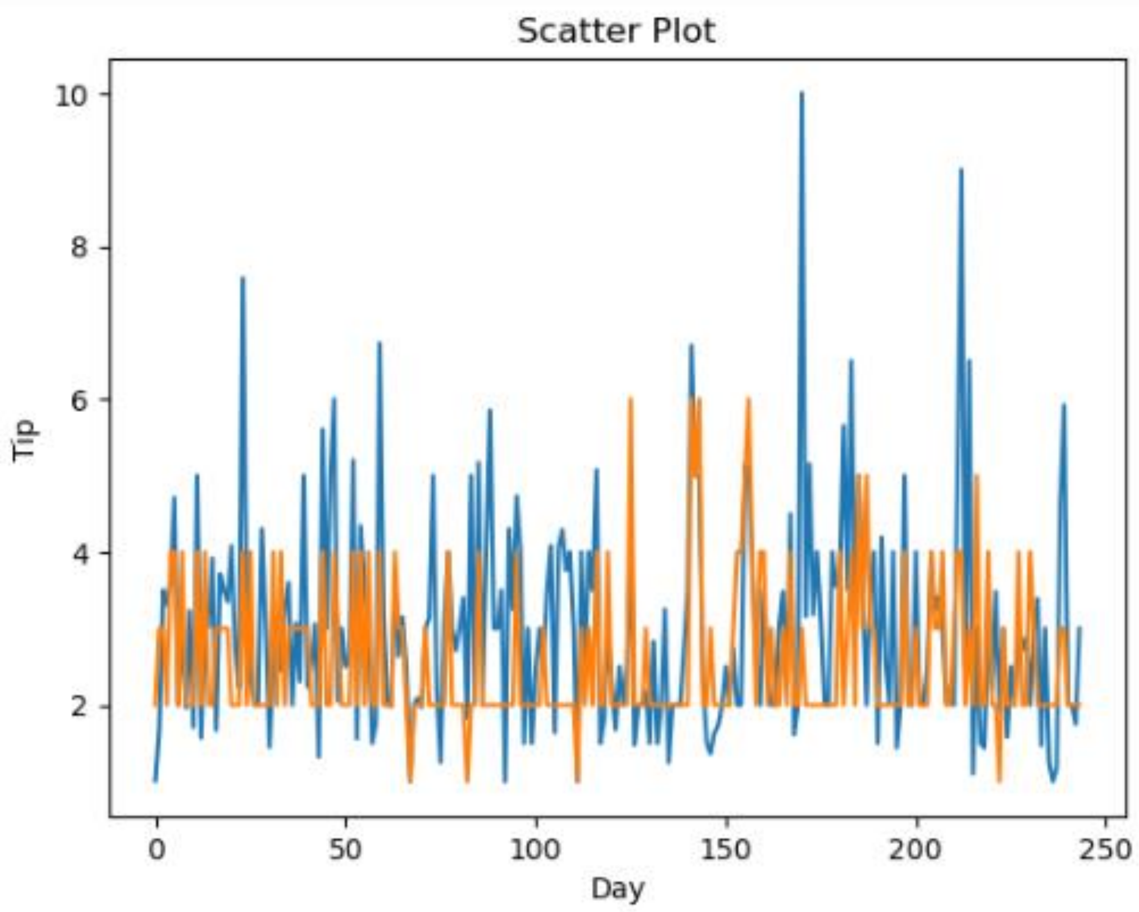
1. **plt.plot(data['tip'])**
2. **plt.plot(data['size'])**

# Adding Title to the Plot

1. plt.title(" **Line Chart** ")

# Setting the X and Y labels

1. plt.xlabel('Day')
2. plt.ylabel('Tip')
3. plt.show()



3. 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:

1. import pandas as pd
2. import matplotlib.pyplot as plt

# reading the database

1. data = pd.read_csv("tips.csv")

# Bar chart with day against tip

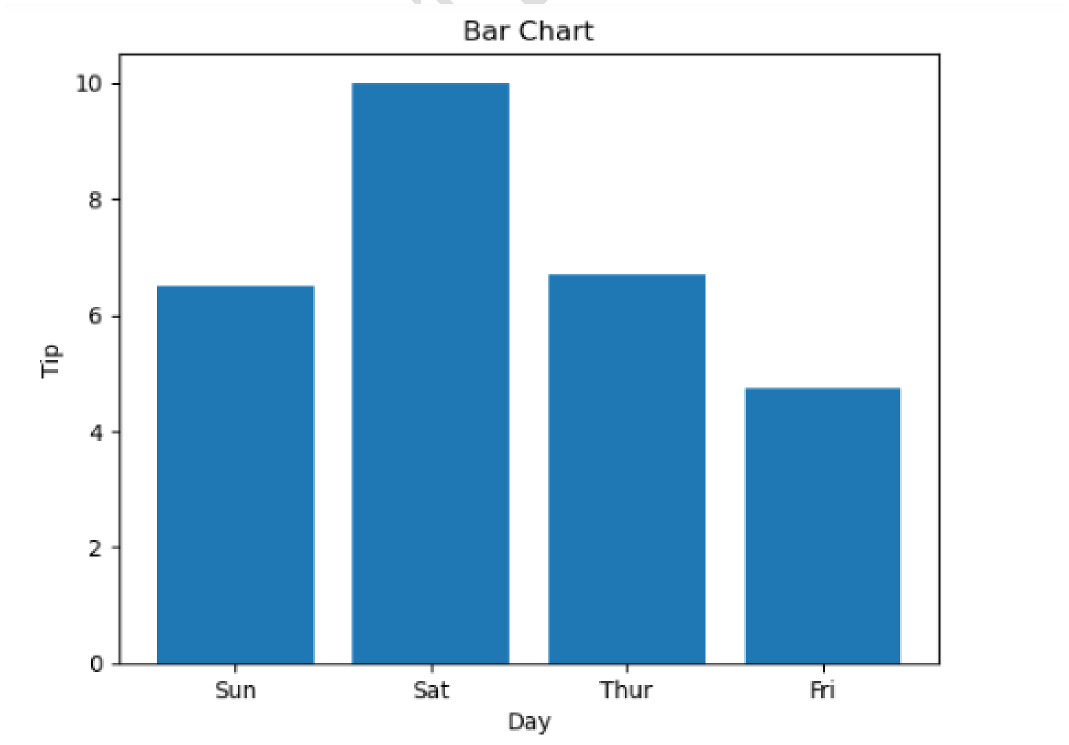
1. **plt.bar(data['day'], data['tip'])**
2. **plt.title("Bar Chart")**

# Setting the X and Y labels

1. plt.xlabel('Day')
2. plt.ylabel('Tip')

# Adding the legends

1. plt.show()



4. 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:

1. import pandas as pd
2. import matplotlib.pyplot as plt

# reading the database

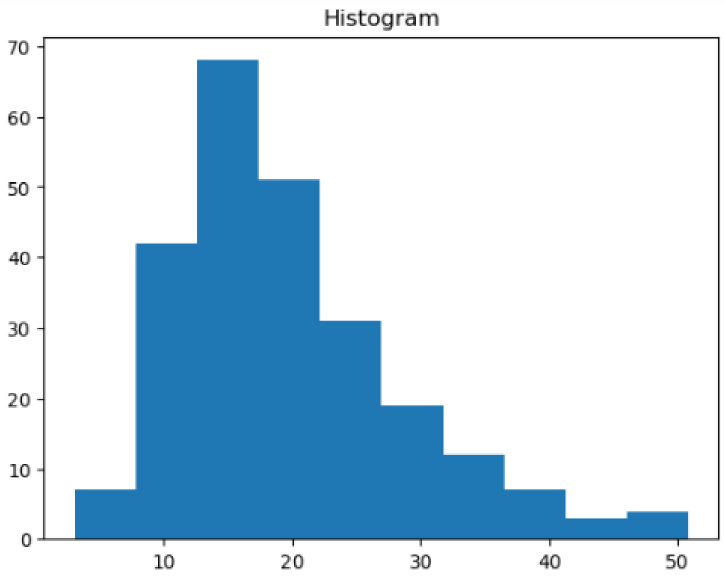
1. data = pd.read_csv("tips.csv")

# histogram of total_bills

1. plt.hist(data['total_bill'])
2. plt.title("Histogram")

# Adding the legends

1. plt.show()



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.

    1. pip install seaborn

- ❖ 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.

1. Scatter Plot

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

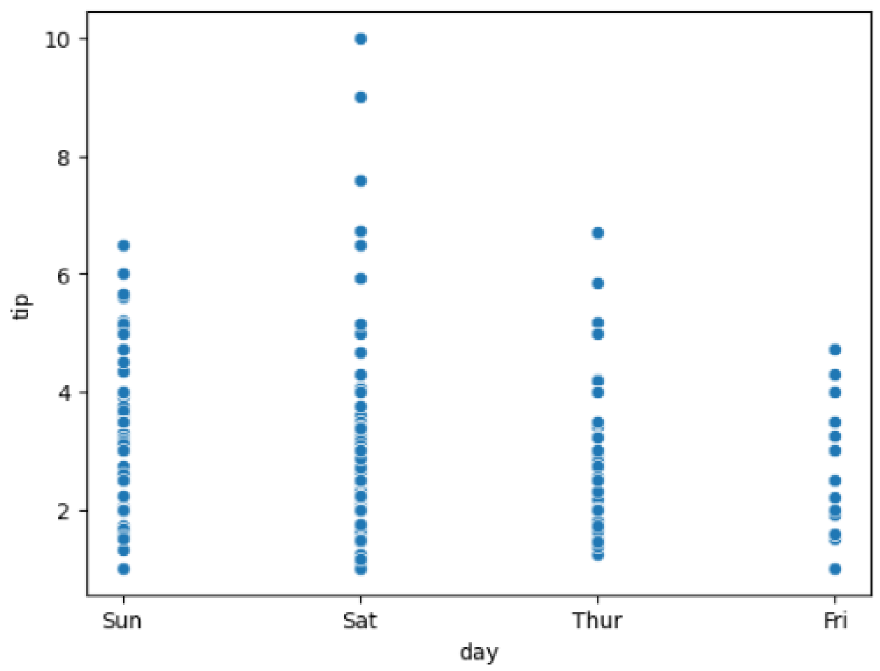
Example:

# importing packages

1. import seaborn as sns
2. import matplotlib.pyplot as plt
3. import pandas as pd

# reading the database

1. data = pd.read_csv("tips.csv")
2. **sns.scatterplot(x='day', y='tip', data=data,)**
3. plt.show()



2. Line Plot

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

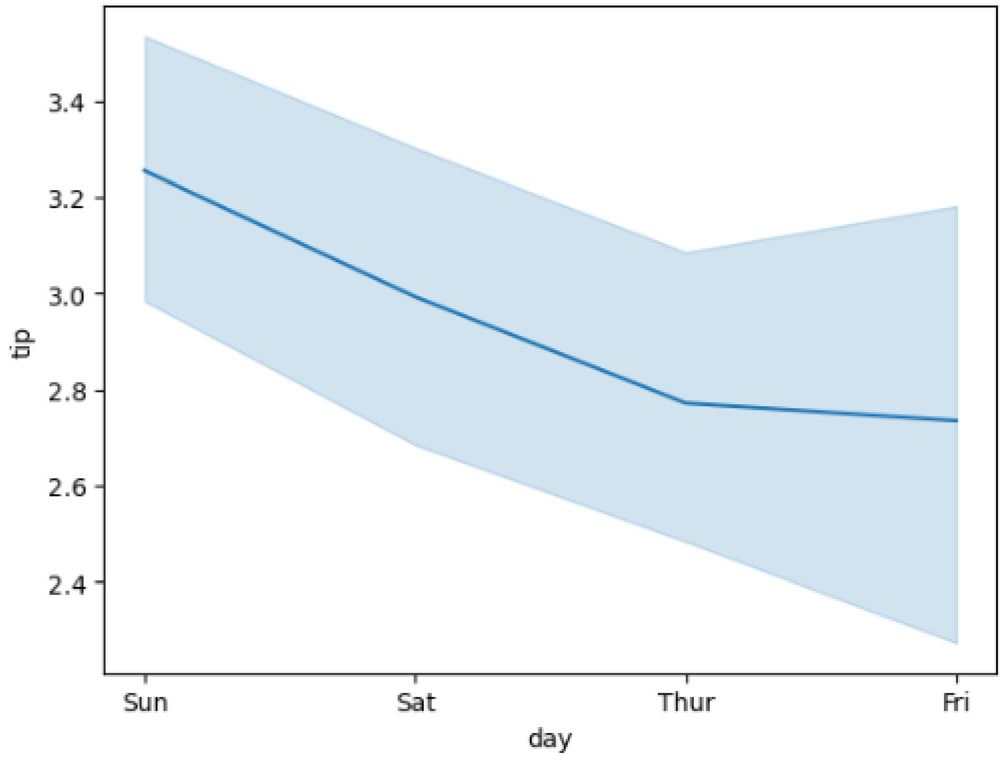
Example:

# importing packages

1. import seaborn as sns
2. import matplotlib.pyplot as plt
3. import pandas as pd

# reading the database

1. data = pd.read_csv("tips.csv")
2. sns.lineplot(x='day', y='tip', data=data)
3. plt.show()



3. Bar Plot

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

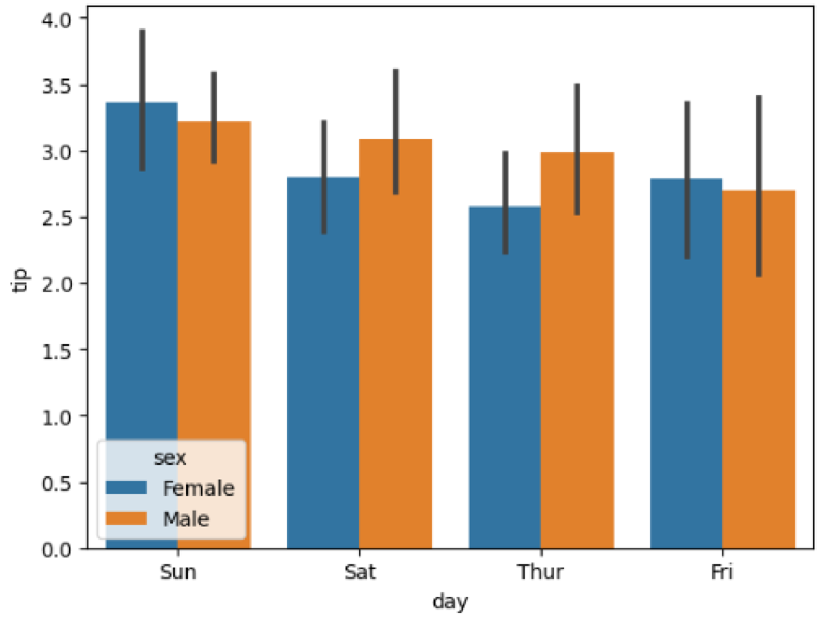
Example:

# importing packages

1. import seaborn as sns
2. import matplotlib.pyplot as plt
3. import pandas as pd

# reading the database

1. data = pd.read_csv("tips.csv")
2. sns.barplot(x='day',y='tip', data=data, hue='sex')
3. plt.show()



4. Histogram

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

Example:

# importing packages

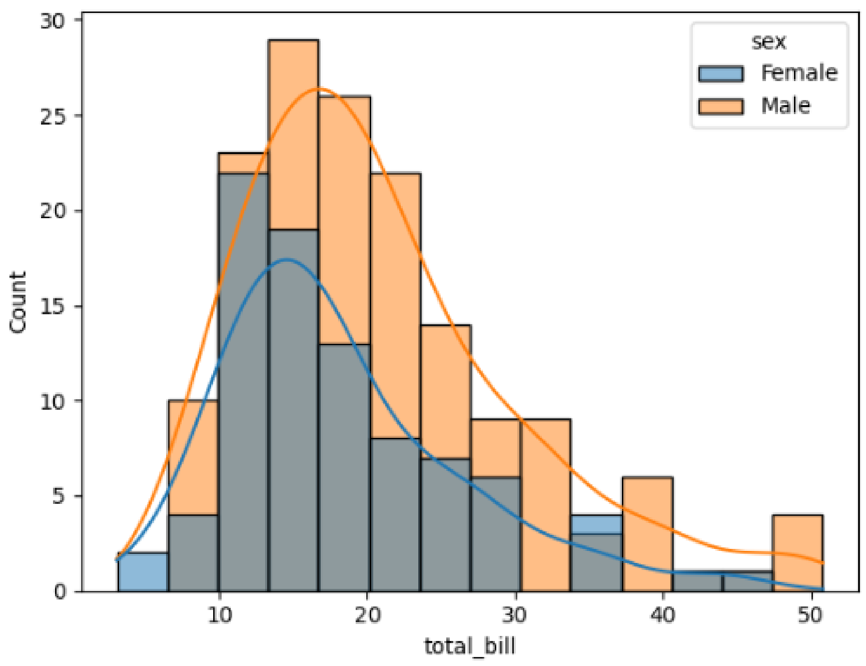
1. import seaborn as sns
2. import matplotlib.pyplot as plt
3. import pandas as pd

# reading the database

1. data = pd.read_csv("tips.csv")
2. **sns.histplot(x='total_bill', data=data,**
    a. **kde=True, hue='sex')**

3. plt.show()

❖ Kernel Density Estimate (KDE) curve to the histogram, providing a smoothed version of the data distribution



4. **Implement Python's built-in data structures with examples program.**

Built-in Data Structures

1. **List:** An ordered and mutable collection that allows duplicate elements and supports indexing, slicing, and various built-in methods like append() and remove().
2. **Tuple:** An ordered and immutable collection, often used to group related data, with support for indexing and iteration.
3. **Set:** An unordered collection of unique elements that supports operations like union, intersection, and difference, but does not allow indexing.
4. **Dictionary:** A collection of key-value pairs where keys are unique and values can be accessed, added, or updated efficiently using the keys.

```
# Example of List in Python with additional operations

# Creating a List
numbers = [10, 20, 30, 40, 50]

# Accessing elements using indexing
print("First element:", numbers[0])  # Output: 10
print("Last element:", numbers[-1])  # Output: 50

# Slicing the List
print("Elements from index 1 to 3:", numbers[1:4])  # Output: [20, 30, 40]

# Adding multiple elements using extend()
numbers.extend([60, 70])
print("After extending the list:", numbers)  # Output: [10, 20, 30, 40, 50, 60, 70]

# Inserting an element at a specific index using insert()
numbers.insert(2, 25)  # Insert 25 at index 2
print("After inserting 25 at index 2:", numbers)  # Output: [10, 20, 25, 30, 40, 50, 60, 70]

# Removing an element using pop()
removed_element = numbers.pop()  # Removes the last element
print("Removed element:", removed_element)  # Output: 70
print("After popping the last element:", numbers)  # Output: [10, 20, 25, 30, 40, 50, 60]

# Checking if an element exists in the list
if 25 in numbers:
   print("25 is in the list")
else:
   print("25 is not in the list")

# Clearing all elements in the List
numbers.clear()
print("List after clearing all elements:", numbers)  # Output: []
```

```
First element: 10
```

Output:
```
Last element: 50
Elements from index 1 to 3: [20, 30, 40]
After extending the list: [10, 20, 30, 40, 50, 60, 70]
After inserting 25 at index 2: [10, 20, 25, 30, 40, 50, 60, 70]
Removed element: 70
After popping the last element: [10, 20, 25, 30, 40, 50, 60]
25 is in the list
List after clearing all elements: []
```
:

```
# Example of Tuple in Python

# Creating a Tuple
person = ("John", 25, "New York")

# Accessing elements using indexing
print("Name:", person[0])  # Output: John
print("Age:", person[1])   # Output: 25
print("City:", person[2])  # Output: New York

# Slicing the Tuple
print("First two elements:", person[:2])  # Output: ('John', 25)

# Iterating over the Tuple
print("Iterating over the Tuple:")
for item in person:
    print(item)

# Checking if an element exists in the Tuple
if "John" in person:
    print("John is in the tuple")

# Concatenating Tuples
address = ("USA", "East Coast")
full_info = person + address
```

```
print("Concatenated Tuple:", full_info)  # Output: ('John', 25, 'New York', 'USA', 'East
Coast')

# Repeating a Tuple
repeat_tuple = person * 2
print("Repeated Tuple:", repeat_tuple)  # Output: ('John', 25, 'New York', 'John', 25, 'New
York')

# Counting occurrences of an element
count_john = person.count("John")
print("Number of 'John' in the tuple:", count_john)  # Output: 1

# Finding the index of an element
index_age = person.index(25)
print("Index of 25:", index_age)  # Output: 1
```

Output:

```
Name: John
Age: 25
City: New York
First two elements: ('John', 25)
Iterating over the Tuple:
John
25
New York
John is in the tuple
Concatenated Tuple: ('John', 25, 'New York', 'USA', 'East Coast')
Repeated Tuple: ('John', 25, 'New York', 'John', 25, 'New York')
Number of 'John' in the tuple: 1
Index of 25: 1
```

# Example of Set in Python with more operations

```
# Creating a Set
colors = {"red", "green", "blue", "yellow"}
```

```
print("Original colors set:", colors)  # Output: {'yellow', 'blue', 'green', 'red'}

# Adding multiple elements using update()
colors.update({"purple", "orange"})
print("After updating with new colors:", colors)  # Output: {'yellow', 'blue', 'green', 'red',
'purple', 'orange'}

# Removing an element using discard()
colors.discard("green")  # Removes 'green' if present, no error if not found
print("After discarding 'green':", colors)  # Output: {'yellow', 'blue', 'red', 'purple', 'orange'}

# Pop an arbitrary element from the set
popped_element = colors.pop()  # Removes and returns an arbitrary element
print("Popped element:", popped_element)
print("After popping an element:", colors)

# Clear the entire set
colors.clear()
print("After clearing the set:", colors)  # Output: set()

# Creating another set for comparison
set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}

# Symmetric Difference using ^ operator
sym_diff = set1 ^ set2
print("Symmetric Difference between set1 and set2:", sym_diff)  # Output: {1, 2, 3, 6, 7, 8}

# Subset check using <= operator
subset_check = {1, 2, 3} <= set1
print("Is {1, 2, 3} a subset of set1?", subset_check)  # Output: True

# Superset check using >= operator
superset_check = set1 >= {1, 2, 3}
print("Is set1 a superset of {1, 2, 3}?", superset_check)  # Output: True
```

Output:

```
Original colors set: {'yellow', 'green', 'blue', 'red'}
After updating with new colors: {'yellow', 'blue', 'orange', 'red', 'green', 'purple'}
After discarding 'green': {'yellow', 'blue', 'orange', 'red', 'purple'}
Popped element: yellow
After popping an element: {'blue', 'orange', 'red', 'purple'}
After clearing the set: set()
Symmetric Difference between set1 and set2: {1, 2, 3, 6, 7, 8}
Is {1, 2, 3} a subset of set1? True
Is set1 a superset of {1, 2, 3}? True
```

# Example of Dictionary in Python

```python
# Creating a Dictionary
student = {
    "name": "John",
    "age": 21,
    "city": "New York"
}

# Accessing values using keys
print("Name:", student["name"])  # Output: John
print("Age:", student["age"])    # Output: 21
print("City:", student["city"])  # Output: New York

# Adding a new key-value pair
student["major"] = "Computer Science"
print("After adding major:", student)  # Output: {'name': 'John', 'age': 21, 'city': 'New York',
'major': 'Computer Science'}

# Updating a value for an existing key
student["age"] = 22
print("After updating age:", student)  # Output: {'name': 'John', 'age': 22, 'city': 'New York',
'major': 'Computer Science'}
```

```
# Removing a key-value pair using pop()
removed_value = student.pop("city")
print("Removed value:", removed_value)  # Output: New York
print("After popping 'city':", student)  # Output: {'name': 'John', 'age': 22, 'major':
'Computer Science'}

# Checking if a key exists in the dictionary
if "name" in student:
    print("The key 'name' exists in the dictionary.")

# Iterating over the dictionary (keys and values)
print("Iterating over the dictionary:")
for key, value in student.items():
    print(key, ":", value)

# Getting the value for a key using get()
major = student.get("major", "Not Available")  # Returns 'Not Available' if 'major' is not
found
print("Major:", major)  # Output: Computer Science

# Getting all keys and values
keys = student.keys()
values = student.values()
print("Keys:", keys)  # Output: dict_keys(['name', 'age', 'major'])
print("Values:", values)  # Output: dict_values(['John', 22, 'Computer Science'])
```

Output:

```
Name: John
Age: 21
City: New York
After adding major: {'name': 'John', 'age': 21, 'city': 'New York', 'major': 'Computer Science'}
After updating age: {'name': 'John', 'age': 22, 'city': 'New York', 'major': 'Computer Science'}
Removed value: New York
After popping 'city': {'name': 'John', 'age': 22, 'major': 'Computer Science'}
```

```
The key 'name' exists in the dictionary.
Iterating over the dictionary:
name : John
age : 22
major : Computer Science
Major: Computer Science
Keys: dict_keys(['name', 'age', 'major'])
Values: dict_values(['John', 22, 'Computer Science'])
```

5. Assume a dataset to calculate Central tendency. What are the limitations of each measure in representing the data?

Measures of Central Tendency

❖ Measures of central tendency are statistical measures that describe the center or average of a set of data values. They provide a single value that represents the central or typical value in a dataset. The three main measures of central tendency are:

Mean:

❖ The mean, often referred to as the average, is calculated by adding up all the values in a dataset and then dividing by the number of values.

Formula:

$$m = \frac{\text{sum of the terms}}{\text{number of terms}}$$

m = mean

Assume Dataset: [10, 20, 30, 40, 50, 60, 70, 80]

$$\text{Mean} = \frac{10+20+30+40+50+60+70+80}{8} = \frac{360}{8} = 45$$

Limitations:

- ❖ **Sensitive to Outliers:** If any value were much higher (e.g., 300 instead of 80), the mean would shift significantly.
- ❖ **Doesn't Reflect Distribution:** The mean may not indicate how data is spread or whether data is concentrated around certain values.

Median:

- ❖ The median is the middle value in a dataset when it is ordered from least to greatest. If there is an even number of values, the median is the average of the two middle values.
- ❖ To find the median, the data must be sorted first.
- ❖ For an odd number of values: Median = Middle value
- ❖ For an even number of values: Median =Sum of two middle values / 2

Assume Dataset: [10, 20, 30, 40, 50, 60, 70, 80]

- ❖ Since there are 8 values, the median is the average of the two middle values:

$$\text{Median} = \frac{40+50}{2} = 45$$

Limitations:

- ❖ **Ignores Distribution:** The median doesn't reflect if the data is clustered or evenly spaced.
- ❖ **Insensitive to Outliers:** While this can be beneficial, it may miss the influence of large values.

Mode:

- ❖ The mode is the value that appears most frequently in a dataset.
- ❖ A dataset may have no mode (if all values occur with the same frequency), one mode (unimodal), or multiple modes (multimodal).

Assume Dataset: [10, 20, 30, 40, 50, 50, 60, 60, 70, 70, 80, 90]

- Frequency of values:

- 10, 20, 30, 40, 80, 90: each occurs **once**
- 50, 60, 70: each occurs **twice**

Modes: [50, 60, 70]

Limitations:

- **No Mode:** Some datasets, like this one, may not have a mode, making it an unreliable measure.
- **Ignores Magnitude:** The mode only reflects frequency, not the size of the values.
- **Multiple Modes:** If there are multiple repeating values, interpretation becomes unclear.