Facilitation Guide

Index

(2 hrs) ILT

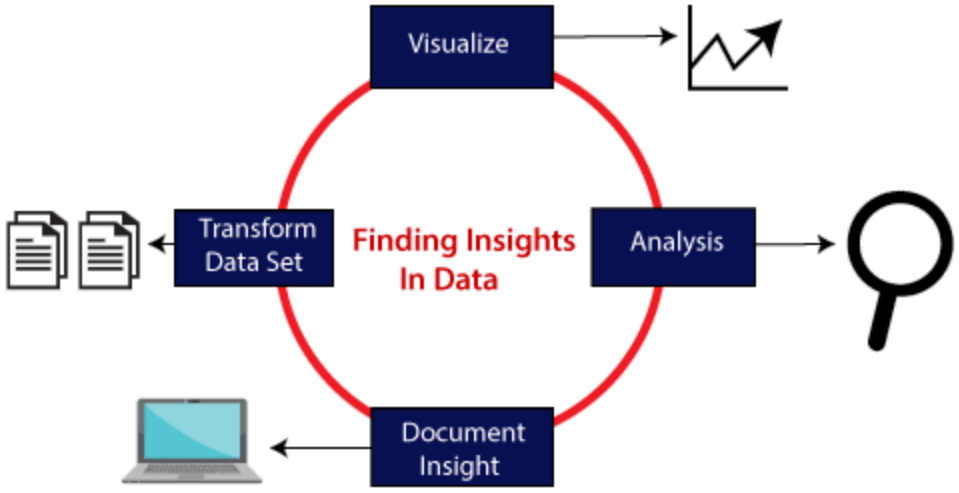
5mins Recap of Previous Session

In this session we are going to understand matplotlib,what is data visualization and python different libraries for data visualization.

I. What is Data Visualization?

Data visualization is the graphical representation of data using visual elements such as charts, graphs, and maps. It is a way of turning complex and often abstract data into visual images that are easy to understand, interpret, and communicate.

Data visualization is a powerful tool for gaining insights from data, making informed decisions, and conveying information to others.



Key Concepts in Data Visualization

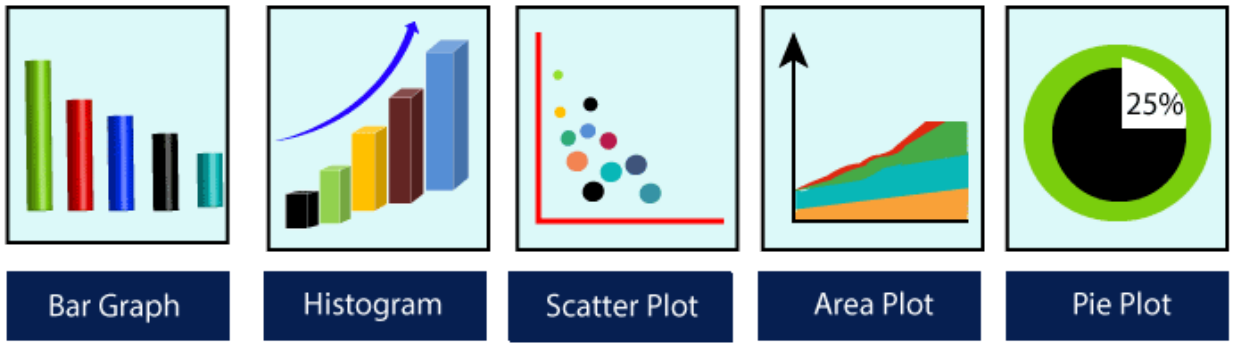
- **Data:** Data can include numbers, text, images, and more.It can be structured (e.g., in databases or spreadsheets) or unstructured (e.g., text documents or images).
- **Visual Representation:** Data is translated into visual elements like bars, lines, points, shapes, and colors.Visual representations should facilitate understanding and analysis.
- **Context:** The context in which data is presented is critical.Consider the audience, purpose, and domain-specific factors.
- **Interactivity:** Interactivity allows users to explore and interact with data visualizations.It can include zooming, filtering, and tooltips.

Importance of Data Visualization

- **Data Exploration:** Data visualization aids in exploring and understanding the underlying patterns and relationships in data.
- **Insights and Patterns:** Visualizations can reveal insights, trends, outliers, and hidden patterns in data that may not be apparent from raw data.
- **Effective Communication:** Visualizations simplify complex data, making it easier to convey information to a diverse audience.
- **Decision Making:** Well-designed visualizations support data-driven decision-making processes by presenting key findings clearly.

- **Storytelling:** Data visualizations can tell a story, making data more engaging and memorable.

Types of Data Visualizations
There are various types of data visualizations, including:



- **Bar Charts:** Used for comparing categories or showing trends over time.
- **Line Charts:** Ideal for showing trends, changes, and fluctuations.
- **Scatter Plots:** Depict relationships and correlations between two variables.
- **Pie Charts:** Display parts of a whole and proportions.
- **Heatmaps:** Visualize data using color intensity.
- **Maps:** Show geographic data and spatial patterns.
- **Histograms:** Display distributions of continuous data.
- **Box Plots:** Reveal data distribution, outliers, and quartiles.
- **Area Plot**:Used to display the cumulative contribution of multiple data series as they change over a continuous or categorical axis.

Tools for Data Visualization
There are numerous tools and libraries available for creating data visualizations, including:

- **Matplotlib:** A popular Python library for creating static and animated visualizations.
- **Seaborn:** Built on Matplotlib, it provides a higher-level interface for creating attractive statistical plots.
- **D3.js:** A JavaScript library for creating interactive and customizable data visualizations.
- **Tableau:** A powerful data visualization tool for creating interactive dashboards.

- **Power BI:** Microsoft's business analytics service for visualizing data and sharing insights.

Best Practices

- Choose the right visualization type for your data and message.
- Keep visualizations simple and avoid clutter.
- Use color effectively but sparingly.
- Label axes, provide legends, and add context.
- Test and refine your visualizations based on user feedback.

Data visualization is a crucial tool for turning data into actionable insights, and it plays a vital role in various fields, including data science, business analytics, scientific research, and journalism. Well-designed data visualizations can make complex data accessible and impactful.

II. Different Python libraries for data visualization

Python offers a variety of libraries for data visualization, each with its own set of features, capabilities, and use cases. These libraries enable data scientists, analysts, and researchers to create informative and visually appealing visualizations. Here's an overview of some of the most popular data visualization libraries in Python.

1. Matplotlib

Matplotlib is one of the most widely used and versatile libraries for creating static, animated, and interactive visualizations in Python. It provides a comprehensive set of tools for creating a wide range of plots and charts, from simple line plots to complex 3D visualizations. Matplotlib is highly customizable and allows users to fine-tune the appearance of their plots.

Pros:

- Extensive customization options.
- Supports various plot types.
- Integrates with Jupyter notebooks.

Cons:

- May require more code for customization.

2. Seaborn

Seaborn is built on top of Matplotlib and provides a higher-level interface for creating aesthetically pleasing statistical visualizations. It simplifies complex tasks such as

creating informative histograms, heatmaps, and pair plots. Seaborn is particularly useful for data exploration and presentation.

Pros:
- Beautiful default styles.
- Specialized functions for statistical visualizations.
- Integration with Pandas data structures.

Cons:
- Not as flexible for general-purpose plotting.

3. Plotly
Plotly is an interactive graphing library that allows users to create interactive and web-based visualizations. It supports a wide range of chart types, including scatter plots, bar charts, and 3D visualizations. Plotly is well-suited for creating interactive dashboards and data applications.

Pros:
- Interactivity with hover, zoom, and filtering.
- Integration with web applications.
- Online sharing and collaboration.

Cons:
- Some advanced features may require a paid subscription.

4. Bokeh
Bokeh is another interactive visualization library that emphasizes interactivity, elegance, and simplicity. It is designed for creating interactive, web-ready visualizations with minimal code. Bokeh supports various plotting techniques, including server-based applications for live data updates.

Pros:
- High-quality interactive plots.
- Server-based applications for real-time updates.
- Cross-platform support.

Cons:
- Complex customization may require more code.

5. Altair
Altair is a declarative statistical visualization library for Python. It allows users to create visually appealing and informative charts using a simple and intuitive syntax. Altair is

built on Vega and Vega-Lite, which are visualization grammars for creating expressive, concise, and consistent graphics.

Pros:

- Simple and expressive syntax.
- Automatic data transformations.
- Easy customization and interaction.

Cons:

- Limited chart types compared to other libraries.

6. ggplot

ggplot is a Python implementation of the ggplot2 library from R. It follows the Grammar of Graphics principles, making it a powerful choice for creating complex visualizations. ggplot allows users to create layered, customized plots with ease.

Pros:

- Grammar of Graphics-based approach.
- Highly customizable.
- Supports a wide range of aesthetics and geoms.

Cons:

- Learning curve for beginners.

ChatGPT Exercise:
Hi! I want to learn more about this following data visualization libraries Seaborn, Plotly, Bokeh, Altair and ggplot. Can you please generate a complete note with code for me?

7. Pandas Plotting

Pandas, a popular data manipulation library, provides a simple interface for creating basic visualizations directly from DataFrames. While not as feature-rich as the aforementioned libraries, Pandas Plotting is convenient for quick exploratory data analysis and basic plotting tasks.

Pros:

- **Ease of Use:** One of the most significant advantages of Pandas Plotting is its simplicity. It provides an easy and intuitive way to create basic visualizations without the need for extensive coding.

- **Seamless Integration:** Pandas Plotting seamlessly integrates with Pandas DataFrames, allowing you to create visualizations directly from your data. This integration streamlines the process of data exploration and analysis.

- **Quick Data Exploration:** When you need to quickly explore your data and gain initial insights, Pandas Plotting is a handy tool. You can generate basic plots with just a few lines of code, making it suitable for data exploration tasks.

- **Basic Customization:** While not as powerful as dedicated visualization libraries like Matplotlib, Pandas Plotting still offers basic customization options for common chart types. You can adjust things like labels, titles, colors, and axes

- **Rapid Prototyping:** Pandas Plotting is suitable for rapid prototyping and generating initial visualizations for communication and understanding. It's particularly useful when you need to produce simple plots during data analysis.

Cons:

- **Limited Customization:** The customization options provided by Pandas Plotting are relatively basic. If you require highly customized or intricate plots, you may find it limiting. More advanced customization often necessitates using other visualization libraries.

- **Limited Chart Types:** Pandas Plotting primarily supports a limited set of common chart types, such as line plots, bar charts, and histograms. If you need to create specialized or less common chart types, you may need to use other libraries.

- **Lack of Advanced Features:** Complex features like interactive visualizations, 3D plots, and complex statistical visualizations are not natively available in Pandas Plotting. For these advanced needs, you'll need to explore other libraries or tools that offer such capabilities.

- **Limited Control Over Aesthetics:** While you can perform basic customization, Pandas Plotting may not provide fine-grained control over aesthetics, such as colors, fonts, and text positioning. This can be limiting if you have specific design requirements.

- **Performance:** For large datasets or complex visualizations, Pandas Plotting might not be as performant as more specialized visualization libraries. This can result in slower rendering times and increased memory usage.

These Python data visualization libraries cater to different needs and preferences, allowing users to select the most appropriate library for their specific use cases. Whether you need static, interactive, or web-based visualizations, there is a library to suit your requirements.

III. Matplotlib

Matplotlib is a popular Python library for creating static, animated, and interactive visualizations. It offers a wide range of tools for data visualization and is widely used in data analysis, scientific research, and various other fields.

Key Features of Matplotlib

- **High-Quality Plots:** Matplotlib can generate publication-quality figures and supports a variety of plot types, including line plots, scatter plots, bar charts, histograms, and more.

- **Customization:** You can fully customize the appearance of your plots, including colors, line styles, markers, fonts, labels, legends, and more.

- **Multiple Backends:** It provides various backends for rendering plots, including GUIs for interactive exploration and various file formats for static image or vector graphic export.

- **Integration with Jupyter:** Matplotlib is commonly used with Jupyter notebooks for interactive data exploration and inline plotting.

- **3D Plotting:** Matplotlib provides support for creating 3D plots and visualizations.

- **Animation:** You can create animated visualizations for time-series data or dynamic simulations.

- **Interactivity:** With additional libraries like mpl_toolkits and mpld3, you can add interactive features to your plots.

- **Wide Adoption:** Matplotlib is widely adopted in the scientific and data analysis communities, making it a standard tool for creating static visualizations.

- **Rich Ecosystem:** Matplotlib is part of a rich ecosystem of Python libraries for data science and visualization, including libraries like NumPy, Pandas, Seaborn, and others.

Getting Started with Matplotlib

Installation

Matplotlib is often included as part of scientific Python distributions such as Anaconda. You can also install it using pip:

pip install matplotlib

Importing Matplotlib

To use Matplotlib, you need to import it:

```
import matplotlib.pyplot as plt
```

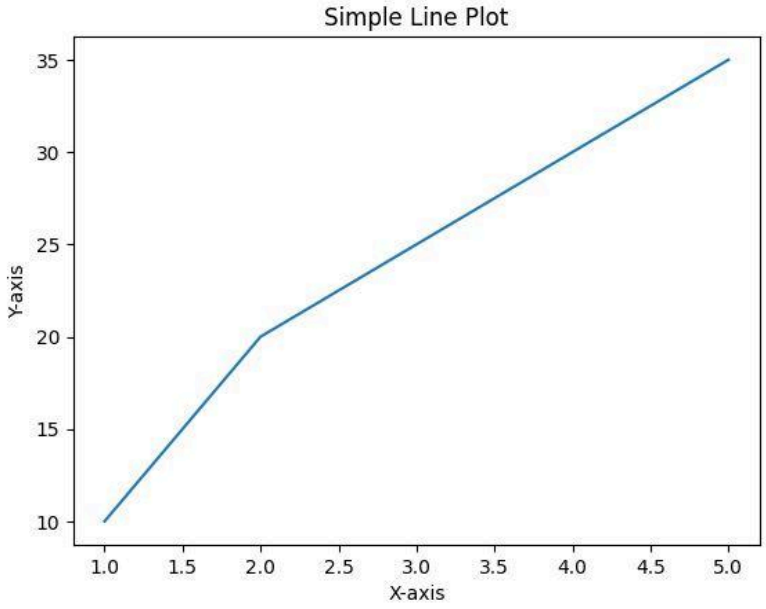
Basic Plotting

Let's start with a basic example of creating a simple line plot:

```
import matplotlib.pyplot as plt

# Data
x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 35]

# Create a line plot
plt.plot(x, y)

# Add labels and title
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Simple Line Plot")

# Show the plot
plt.show()
```

Explanation:
We import the Matplotlib library with the alias "plt."
We define two lists, x and y, to represent the data points for the X-axis and Y-axis, respectively.
We create a line plot using plt.plot(x, y), where x and y are used to specify the data for the X-axis and Y-axis, respectively. This code generates a basic line plot.
We add labels to the X-axis and Y-axis using plt.xlabel("X-axis") and plt.ylabel("Y-axis").
We set a title for the plot with plt.title("Simple Line Plot").
Finally, We display the plot using plt.show().

Output:



IV. Line plot
A line plot, also known as a line chart or line graph, is a fundamental data visualization technique used to display data points as a series of data connected by straight lines.

This type of plot is commonly used to represent data that varies continuously over a range, typically along a single dimension, such as time, distance, or some other ordered variable.

Continuous Data: Line plots are particularly useful for visualizing continuous or sequential data, such as time series data, where data points are ordered chronologically or by some other continuous variable.

Line plots are useful for various applications, including:

- Visualizing stock price changes over time.
- Displaying temperature variations during a day or year.
- Analyzing trends in website traffic.
- Monitoring changes in sensor data.
- Comparing data series, such as sales figures for different products.

In Python, the Matplotlib library provides the **plot()** function for creating line plots.

We already studied a simple line plot in the previous session. Now lets use line plot to study …..

Example:Visualize daily temperature in a specific location

```
import matplotlib.pyplot as plt

# Sample data: Daily temperatures for a month
days = list(range(1, 31))
temperatures = [68, 70, 72, 75, 77, 80, 82, 83, 81, 78, 75, 72, 71, 70, 72, 74, 77, 79,
80, 82, 84, 86, 88, 87, 85, 82, 80, 77, 75, 73]

# Create the line plot
plt.plot(days, temperatures,marker='o')

# Add labels and a title
plt.xlabel("Day of the Month")
plt.ylabel("Temperature (°F)")
plt.title("Daily Temperature Over a Month")

plt.show()
```

Explanation:
We import the matplotlib.pyplot library as plt to create the line plot.

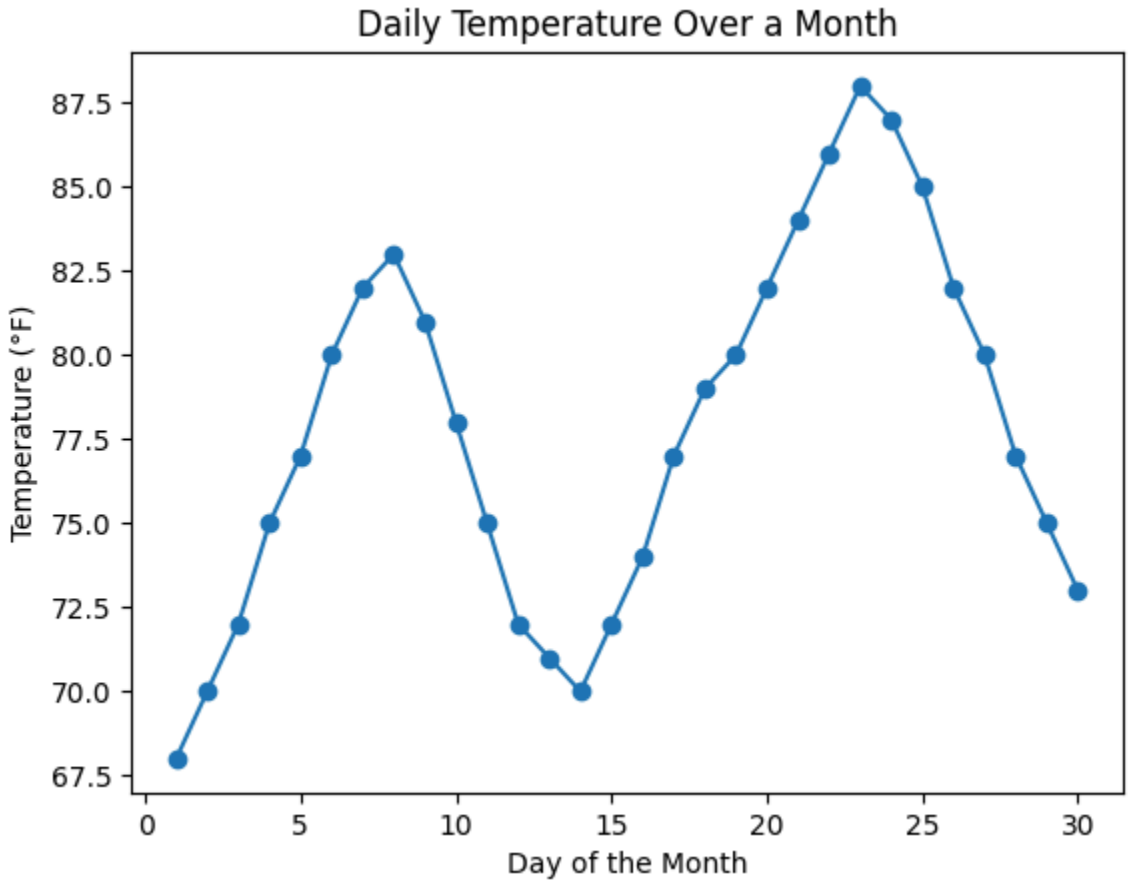
We define the sample data, which includes a list of days ranging from 1 to 30 (assuming a 30-day month) and a corresponding list of daily temperatures in Fahrenheit.

We use plt.plot() to create the line plot. The days list serves as the x-axis values, and the temperatures list is used as the y-axis values. We specify the marker as 'o' to indicate data points with circles.

Labels and a title are added using plt.xlabel(), plt.ylabel(), and plt.title() to provide context and description to the chart.

Finally, we display the chart using plt.show()

Output:



Conclusion: This code generates a line plot that shows the daily temperature variations over the course of a month. It's a simple and effective way to visualize how temperatures change from day to day. In this example, you can observe temperature fluctuations, and this type of chart can be helpful for tracking weather data, making observations about climate, and more.

V. What is the bar plot?

A bar plot, also known as a bar chart or bar graph, is a data visualization technique that uses rectangular bars of varying heights to represent and compare the values of different categories or groups. Each bar's height is proportional to the value it represents, making it a simple and effective way to visualize and compare data.

Bar plots are particularly useful for displaying categorical or discrete data, making them one of the most common chart types in data visualization.

Categorical data or discrete data:
A simple example of categorical data is the "Eye Color" of individuals. In this case, "Eye Color" is a categorical variable because it represents distinct categories or groups that individuals can fall into. The possible categories for eye color include:

Blue
Brown
Green
Hazel
Gray

Each individual's eye color can be described using one of these categories, and there is no inherent numerical value associated with these categories. It's a classic example of nominal categorical data because the categories have no particular order or ranking; they are simply labels for different eye colors.

Bar plots are commonly used to illustrate various types of data, such as:

- Comparing sales figures for different products or regions.
- Showing the distribution of age groups in a population.
- Visualizing the frequency of specific events or outcomes.
- Representing survey responses by category.

In Python, the Matplotlib library provides the **bar()** function for creating bar plots. You can customize bar plots by adjusting parameters and adding labels, titles, and other elements to make the visualization more meaningful and informative.

Prerequisites:
Before creating a bar plot, ensure you have Matplotlib installed. You can install it using pip if it's not already install

```
pip install matplotlib
```

Here's a step-by-step guide to creating a bar plot:

Import Matplotlib:

First, import the Matplotlib library to use its functions for creating plots.

```
import matplotlib.pyplot as plt
```

Prepare Data:
Define the data you want to visualize. In this example, we'll use sample data for a bar plot.

```
# Sample data: Number of books sold by genre over a year
genres = ["Mystery", "Romance", "Science Fiction", "Fantasy", "Thriller"]
books_sold = [120, 90, 80, 110, 70]
```

Create the Bar Plot:
Use the plt.bar() function to create the bar plot. You'll provide the categories (x-axis) and their corresponding values (y-axis).

```
plt.bar(genres, books_sold)
```

Customize the Plot (Optional):
You can customize the plot by adding labels, titles, changing colors, and more.

```
# Add labels and a title
plt.xlabel("Genre")
plt.ylabel("Number of Books Sold")
plt.title("Books Sold by Genre Over a Year")
```

Display the Plot:
Finally, use plt.show() to display the bar plot.

```
plt.show()
```

Complete Example:
Visualize the number of books sold in a bookstore by genre over a year

```
import matplotlib.pyplot as plt

# Sample data: Number of books sold by genre over a year
genres = ["Mystery", "Romance", "Science Fiction", "Fantasy", "Thriller"]
books_sold = [120, 90, 80, 110, 70]

# Create the bar plot
plt.bar(genres, books_sold)

# Add labels and a title
plt.xlabel("Genre")
plt.ylabel("Number of Books Sold")
plt.title("Books Sold by Genre Over a Year")
# Display the plot
plt.show()
```

Explanation:
We import the matplotlib.pyplot library for creating plots.

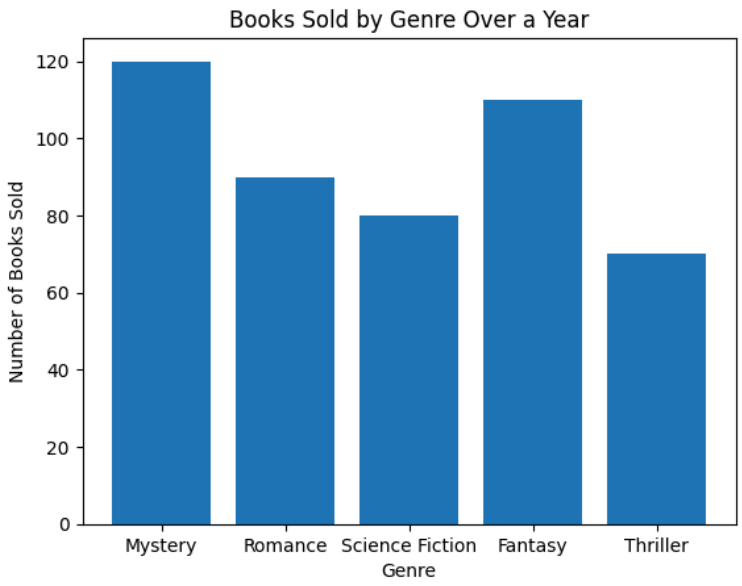
We define the genres and books_sold lists, representing the x and y data points, respectively.

We create the bar plot using plt.bar(genres, books_sold). By default, it creates vertical bars for each genre's book sales.

We add labels to the x and y axes, as well as a title to make the plot more informative.

Finally, We display the plot using plt.show().

Output:



Conclusion :
From the above chart we can see that Mystery genre books are the highest selling books.

Horizontal Bar Plot in Python

A horizontal bar plot, also known as a horizontal bar chart, is a data visualization that represents data in horizontal bars. It is particularly useful for comparing data within categories or displaying ranked data. In Python, you can create horizontal bar plots using the Matplotlib library. Here's a guide on how to create a horizontal bar plot with code:

Here's a step-by-step guide to creating a horizontal bar plot:

Import Matplotlib:
First, import the Matplotlib library to use its functions for creating plots.

```
import matplotlib.pyplot as plt
```

Prepare Data:
Define the data you want to visualize. In this example, we'll use sample data for a horizontal bar plot.

```
# Sample data: Population distribution by age group in a city
age_groups = ["0-10", "11-20", "21-30", "31-40", "41-50", "51-60", "61-70", "71+"]
population = [15000, 22000, 30000, 28000, 25000, 18000, 12000, 8000]
```

Create the Horizontal Bar Plot:
Use the plt.barh() function to create the horizontal bar plot. You'll provide the values (x-axis) and their corresponding categories (y-axis). To create a horizontal bar plot, use barh instead of bar.

```
# Create the bar plot
plt.bar(age_groups, population, color='lightblue')
```

Customize the Plot (Optional):
You can customize the plot by adding labels, titles, changing colors, and more.

```
# Add labels and a title
plt.xlabel("Age Group")
plt.ylabel("Population")
plt.title("Population Distribution by Age Group")
```

Display the Plot:
Finally, use plt.show() to display the horizontal bar plot.

```
plt.show()
```

Complete Example:
Visualize the distribution of the population in a city by age group

```
import matplotlib.pyplot as plt

# Sample data: Population distribution by age group in a city
age_groups = ["0-10", "11-20", "21-30", "31-40", "41-50", "51-60", "61-70", "71+"]
population = [15000, 22000, 30000, 28000, 25000, 18000, 12000, 8000]
```

```
# Create the bar plot
plt.barh(age_groups, population, color='lightblue')

# Add labels and a title
plt.xlabel("Age Group")
plt.ylabel("Population")
plt.title("Population Distribution by Age Group")

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

Explanation:
We import the matplotlib.pyplot library for creating plots.

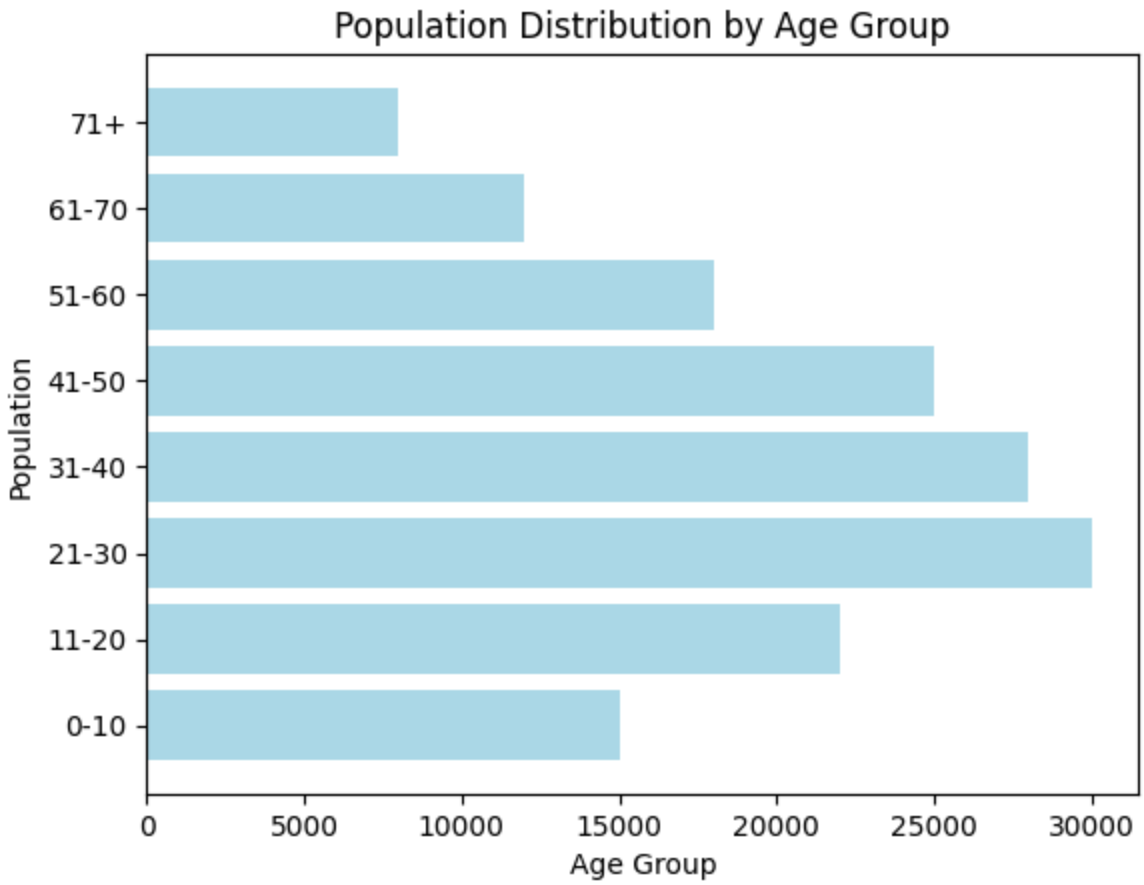
We define the age_groups and population lists, representing the x and y data points, respectively.

We create the bar plot using plt.bar(age_groups, population). By default, it creates vertical bars for each age group's population.

We add labels to the x and y axes and a title to make the plot more informative.

Finally, We display the plot using plt.show().

Output:



Conclusion: From the above chart we can see that most of the populations fall under the 21-30 age group.

VI. What is a histogram?

A histogram is a visual representation of the distribution of data. It helps you understand the frequency or probability of different values within a dataset.
Let's say you have a dataset of test scores in a class. By creating a histogram of these scores, you can see how many students scored in different score ranges, helping you identify the most common score and any unusual patterns in the data.

Let's explore more

Example: Histogram of Ages of Survey Respondents

Suppose you conducted a survey and collected data on the ages of 200 respondents. You want to create a histogram to understand the age distribution of the survey participants.

```
import matplotlib.pyplot as plt
ages = [1, 1, 2, 3, 3, 5, 7, 8, 9, 10,
    10, 11, 11, 13, 13, 15, 16, 17, 18, 18,
    18, 19, 20, 21, 21, 23, 24, 24, 25, 25,
    25, 25, 26, 26, 26, 27, 27, 27, 27, 27,
    29, 30, 30, 31, 33, 34, 34, 34, 35, 36,
    36, 37, 37, 38, 38, 39, 40, 41, 41, 42,
    43, 44, 45, 45, 46, 47, 48, 48, 49, 50,
    51, 52, 53, 54, 55, 55, 56, 57, 58, 60,
    61, 63, 64, 65, 66, 68, 70, 71, 72, 74,
    75, 77, 81, 83, 84, 87, 89, 90, 90, 91]
b=[0,10,20,30,40,50,60,70,80,90,100]
plt.hist(ages, bins=b, edgecolor='k')
plt.xlabel('Age')
plt.ylabel('Number of Respondents')
plt.title('Age Distribution of Survey Respondents')
plt.show()
```

Explanation:
We import the matplotlib.pyplot library for creating plots.

We define the ages list, which contains the ages of survey respondents.

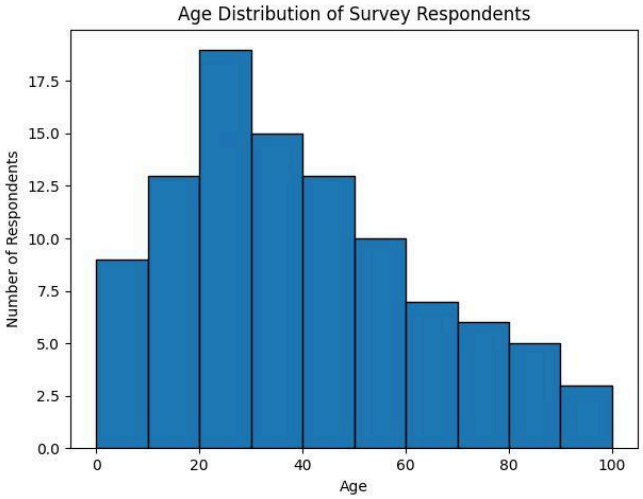
We specify the bin edges for the histogram in the b list. These bins group the ages into 10-year intervals.

We create the histogram using plt.hist(ages, bins=b, edgecolor='k'). It uses the specified bin edges and adds a black edge color to the bars.

We add labels to the x and y axes and a title to make the plot more informative.

Finally, We display the histogram using plt.show().

Output:



Conclusion:
In this histogram, you can quickly grasp the age distribution of survey respondents. It's evident that there is a significant number of respondents in the 25-30 age range, and there is a general trend of decreasing numbers as respondents get older.

Based on this information, the frequency table would look like this:

Intervals (bins)	Frequency (No of Respondents)
0-9	9
10-19	13
20-29	19
30-39	15
40-49	13
50-59	10
60-69	7
70-79	6
80-89	5
90–99	3

Exercise

Use GPT to write a program:

1.Hi! I am new in python matplotlib can you please generate few basic charts for me

2. Hi! Suppose I am a teacher, and I want to analyze the exam scores of my students to understand how they performed on a recent math test. I have collected the exam scores, and I want to create a histogram to visualize the distribution of scores. Can you please generate a complete code.
[**Note**: You need to share the exam scores in list format. For eg. 22,45,60,75,80]

Please refer to the ILT4 Lab doc in the LMS.