VISUALIZATION

Visualization transforms raw data into interactive charts and graphs for clearer insights. Python's Plotly creates dynamic, web-ready visualizations with zooming, hovering, and animation features. Pandas provides built-in plotting tools for quick exploratory analysis of Data Frames. Together, they enable both rapid prototyping (via Pandas) and sophisticated dashboards (via Plotly). These libraries excel at handling real-world datasets while maintaining simple syntax.

Let's explore **Plotly** and **Pandas** visualization capabilities with practical examples.

Plotly

Plotly, introduced in **2012**, is an interactive visualization library built on JavaScript. It specializes in web-based, dynamic plots that work seamlessly in Jupyter notebooks and browsers. A **Plotly** figure contains:

1. **Figure** – The main object that represents the complete plot.

2. **Data** – A list of plot types (e.g., Scatter, Bar, Pie, etc.), each referred to as a "trace".

3. **Layout** – A dictionary-like structure that contains information about the appearance of the figure (titles, axes, background, legend, etc.).

4. **Frames** (optional) – Used for animations.

5. **Config** (optional when displaying) – Contains settings like display mode, responsiveness, toolbars, etc.

Plotly is a Python graphing library used to create interactive and web-based visualizations. It supports different types of plots including bar graphs, scatter plots, pie charts, histograms, and area charts.

Plotly has **two main sub-libraries** in Python that are commonly used for creating visualizations:
1. plotly. express 2. plotly. graph_objects

We import **plotly.express** from Plotly as following:

```
import plotly.express as px
```

We import **plotly.graph_objects** from Plotly as following:

```
import plotly.graph_objects as go
```

Some of the plots in Plotly:

LINE CHART

A **line chart** in Plotly is used to display trends or changes in data over intervals using connected data points. It offers interactive features like **zoom, hover tooltips, and export options,** making it ideal for data analysis and visualization in Python. It can be created using the line () method of plotly. express class.

Code snippet:

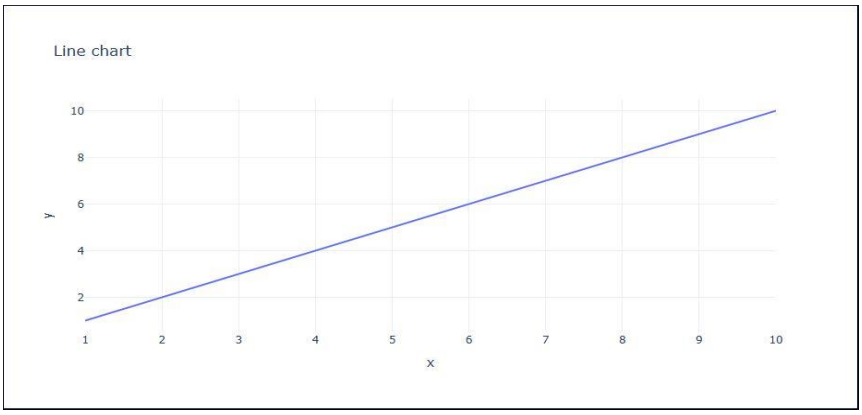
```
import plotly.express as px

# Define data
x = [1, 10]
y = [1, 10]

# Create line chart
fig = px.line(
    x=x,
    y=y,
    labels={'x': 'x', 'y': 'y'},
    title="Line chart"
)

fig.show()
```

Output:



Description:

x and y are simple Python lists used to define coordinates.

px. line () is used for plotting the **line chart** using Plotly Express.

title adds a **title** to the chart.

labels are used to define the **x-axis and y-axis labels** ("x" and "y" respectively).

show () displays the interactive chart with features like **hover, zoom, and export**.

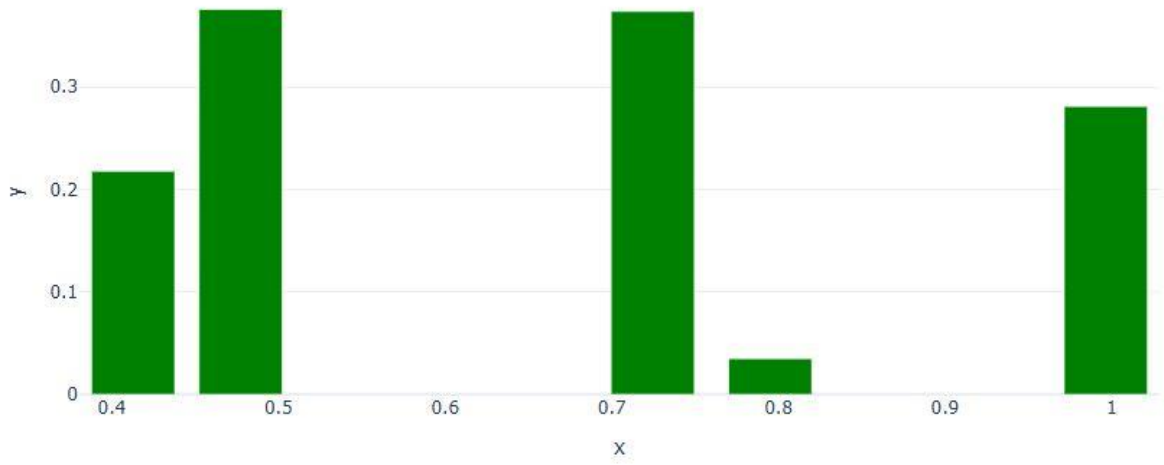
BAR GRAPH

A **Plotly bar graph** uses rectangular bars where **height represents values** (like counts or totals). Unlike static charts, Plotly adds **hover details, zooming, and toggling**—perfect for comparing categories (e.g., sales by month).

Code snippet:

```
import plotly.express as px
import numpy as np
x, y = np.random.rand(5), np.random.rand(5)
fig = px.bar(x=x, y=y, title="Bar Chart", color_discrete_sequence=["green"])
fig.update_traces(width=0.05).show()
```

Output:



Description:

x, y: Determine bar positions and heights

width=0.05: Sets thin bar width (default is wider)

Shows 5 vertical green bars at random x-positions (0.4-1.0)

color_discrete_sequence=["green"]: Colors all bars green

HISTOGRAM

A **Plotly histogram** is an **interactive** data visualization that displays the distribution of numerical data by grouping values into customizable ranges (called *bins*) and representing their frequency with bars.

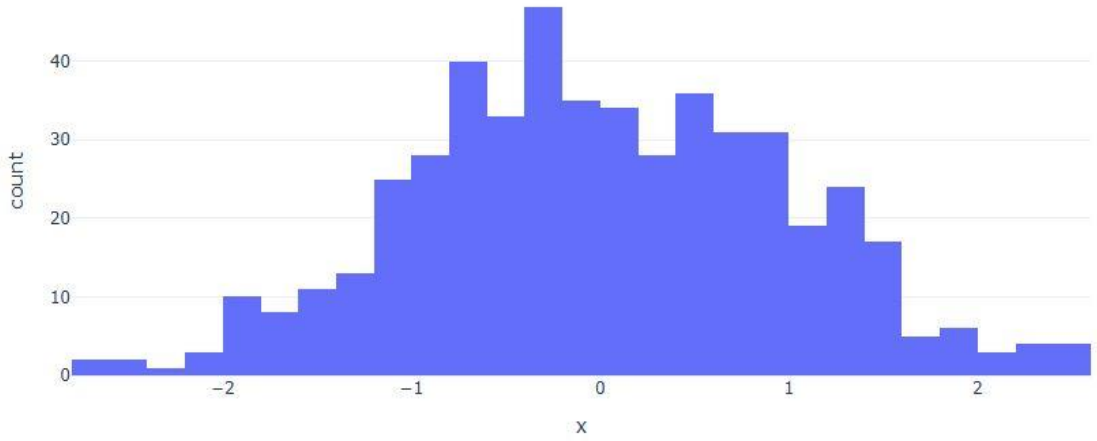
Code snippet:

```
import plotly.express as px
import numpy as np

# Generate random data
x = np.random.randn(500)

# Create histogram
fig = px.histogram(x=x, nbins=30, title="Histogram Example")
fig.show()
```

Output:



Description:

Data (x): Normally distributed values (peaking around 0, range -2 to 2)

Automatic blue coloring of bars

The x-axis ranges from -2 to 2

The y-axis shows frequency counts from 0 to 40

Frequency decreases smoothly in both directions

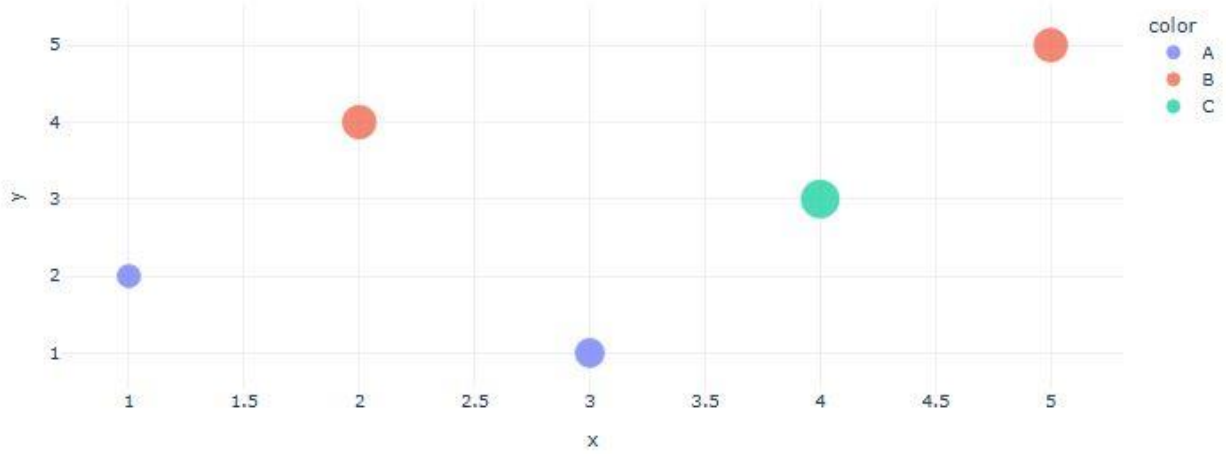
SCATTER PLOT

A **Plotly scatter plot** displays relationships between variables using interactive dots, with automatic color-coding for different groups. Hover reveals exact values; while zooming and clickable legends enable effortless exploration. Ideal for **spotting trends, clusters, or outliers** in your data.

Code snippet:

```
import plotly.express as px
import pandas as pd
data = {'x': [1, 2, 3, 4, 5],'y': [2, 4, 1, 3, 5],
'color': ['A', 'B', 'A', 'C', 'B'],'size': [10, 20, 15, 25, 20]}
df = pd.DataFrame(data)
fig = px.scatter(df, x='x', y='y', color='color', size='size',
title='My Scatter Plot')
fig.show()
```

Output:



Description:

Scatter plot with variable **point sizes** and **color groups.**

Plots **5 points** with custom (x,y) positions, colors (A/B/C), and sizes (10-25).

Size of points varies based on size column.

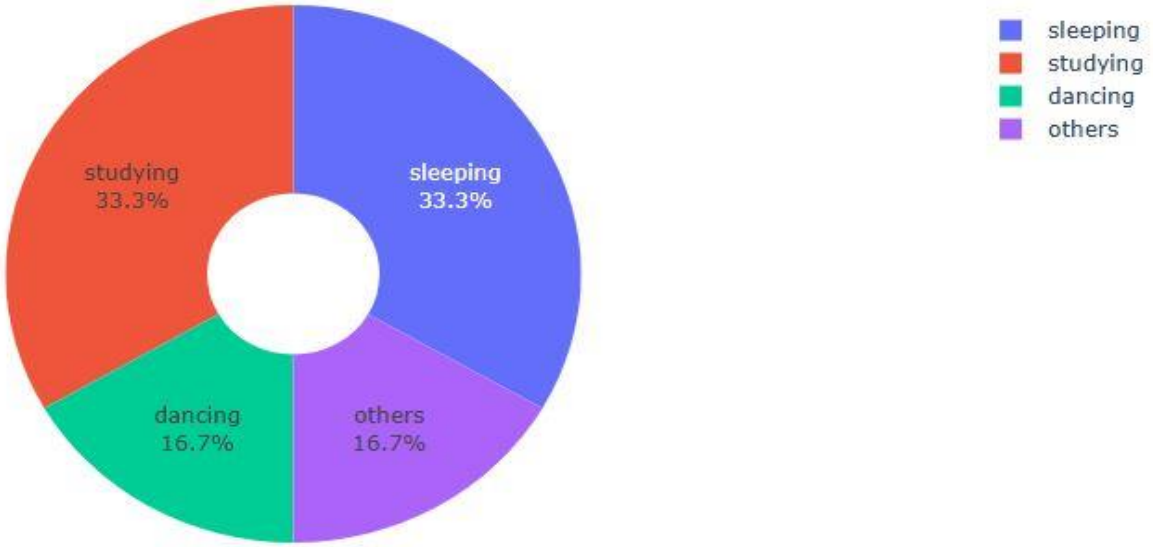
PIE CHART

A **Plotly pie chart** displays categorical data as proportional slices, with interactive hover details (values/percentages) and clickable legend toggles. It automatically calculates **percentages, applies distinct colors, and supports smooth animations** — ideal visualizing market shares, survey results, or budget splits.

Code snippet:

```
import plotly.express as px
time = [8,8,4,4]
work = ["sleeping","studying","dancing","others"]
fig = px.pie(values=time, names=work, hole=0.3)
fig.update_traces(textposition='inside', textinfo='percent+label')
fig.show()
```

Output:



Description:

values=time: Array [8,8,4,4] defines slice sizes (represents hours)

names=work: Labels each slice ("sleeping", "studying", etc.)

hole=0.3: Creates a donut chart (30%-hole radius)

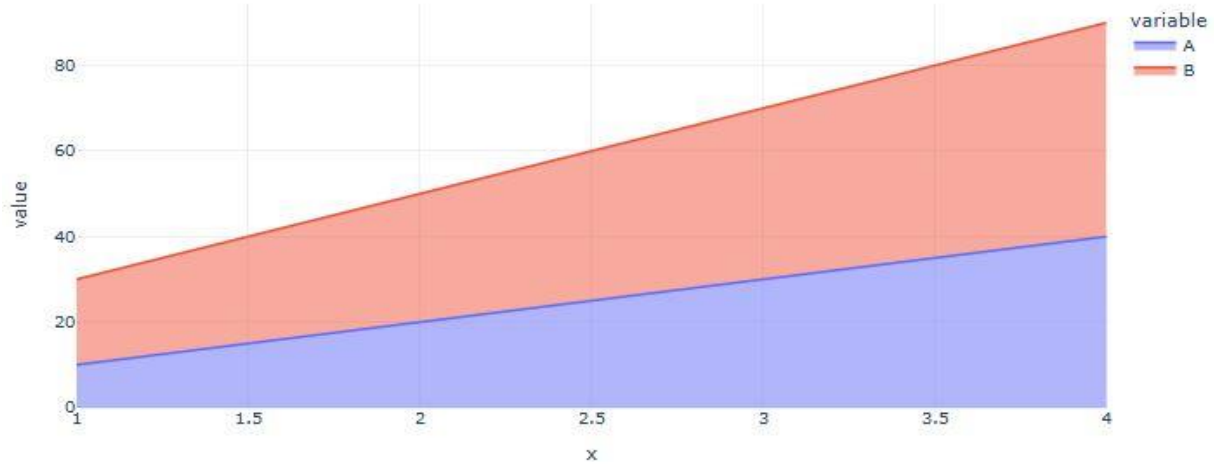
AREA CHART

A **Plotly area chart** visualizes trends by filling the space beneath line plots, making it perfect for showing cumulative totals or part-to-whole relationships over time. Its interactive features—like **hover tooltips, zooming, and toggleable categories**—transform static data into dynamic stories. Ideal for **dashboards or analyzing trends in sales, populations, or resource allocation**, all with Plotly's automatic styling.

Code snippet:

```
import plotly.express as px
import pandas as pd
df = pd.DataFrame({
    'x': [1, 2, 3, 4],
    'A': [10, 20, 30, 40],
    'B': [20, 30, 40, 50]
})
fig = px.area(df, x='x', y=['A', 'B'])
fig.show()
```

Output:



Description:

Stacked area chart showing two variables (A and B) across x-values 1 to 4.

Variable A and B differentiated by color

X-axis: Discrete values (1.5 to 4)

Y-axis: Values range 0 to 40 (visible portion, actual max is 90 due to stacking

MATPLOTLIB

Matplotlib, introduced in **2002 by John Hunter**. It is for plotting **2D plots** which is built on **NumPy**. A plot of matplotlib contains:

1. **Figure** - Figure is the container in which plots are present. It can have a single plot or multiple plots.

2. **Axes** - Axes are the plots, and they are artist attached to the figure. A plot can have 2 or 3 axes. set_xlabel(), set_ylabel() are the functions used to set the labels of x and y respectively.

3. **Axis** - It is responsible for generating ticks or the limits on the axes.

4. **Artists** - The whatever content visible in a figure are the artists.

PYPLOT

Pyplot is a sub library of matplotlib where all the utilities lie under. It has different types of plots including bar graphs, scatter plots, pie charts, histograms, area charts.

We import pyplot from matplotlib as following:

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

We also import NumPy as a support for the matplotlib:

```
import numpy as np
```

Some of the plots in matplotlib:

LINE CHART

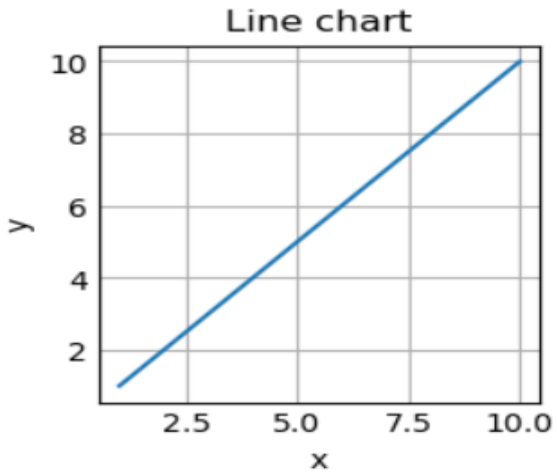
Line plots are the basic charts where dot consecutive points are connected by a continuous line.

The default plot() is used for line charts.

Code snippet:

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  x=np.array([1,10])
5  y=np.array([1,10])
6
7  plt.plot(x,y)
8  plt.title("Line chart")
9  plt.xlabel("x")
10 plt.ylabel("y")
11 plt.show()
```

Output:



Description:

np.array() will generate an array in the specified range.

plot () for the plotting the line chart.

title () for defining the title, ylabel() and xlabel () for labelling y and x axis respectively.

show () displays the graph.

BAR GRAPH

Rectangular bars are used in bar chart as the height represent the frequency of a particular element. bar() is used for plotting bar graphs, it has parameters like x, y, width, color.

bar(x, y, color, width)

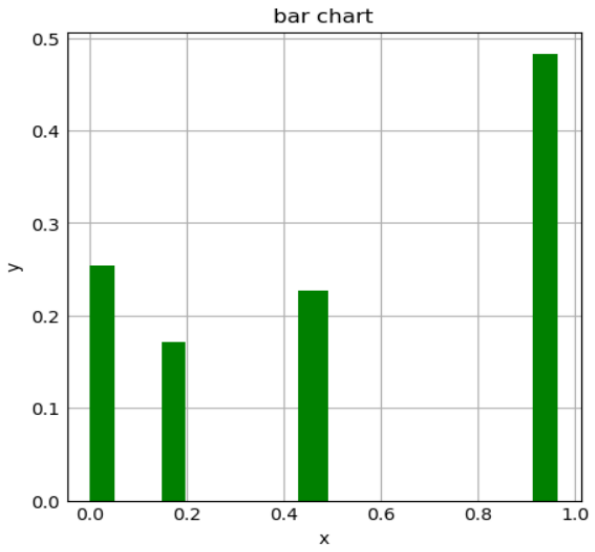
Code snippet:

```
import matplotlib.pyplot as plt
import numpy as np

x=np.random.rand(5)
y=np.random.rand(5)
print(x)

fig = plt.figure(figsize = (4, 4))
plt.bar(x,y, width=0.05, color="green")
plt.title("bar chart")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

Output:



Description:

random.rand(5) generates a random array.

Here bar() is used along with the parameters, width which denoted the width of the rectangular bars and color for the bars.

HISTOGRAM

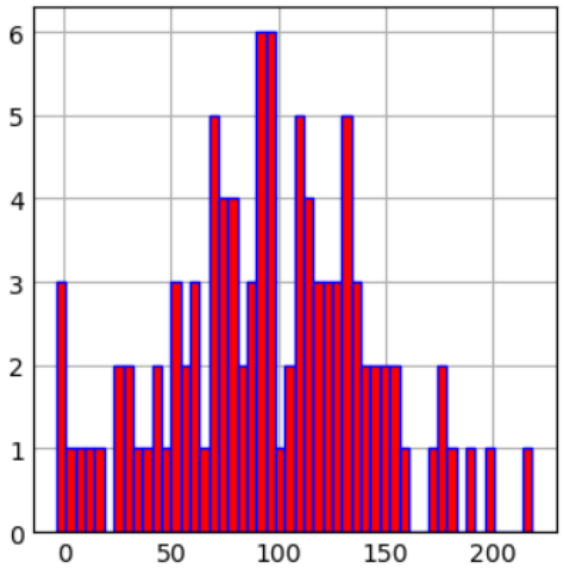
Histogram is a type of bar graph where the graph is represented in groups. hist() is used for plotting histogram with parameters x, bins, color, edgecolor.

Code snippet:

```
import matplotlib.pyplot as plt
import numpy as np

x=np.random.normal(100,50,100)
fig = plt.figure(figsize = (3, 3))
plt.hist(x,bins=50,color="red", edgecolor="blue")
plt.show()
```

Output:



Description:

hist() have parameters x which is a random generated array around one hundred with standard deviation 50 of 100 values.

bins indicate the number of sections on x axis.

color indicates the color of rectangular bars.

edge color that divides the rectangular bars.

SCATTER PLOT

Scatter plot uses dots to depict the relationship between the data. scatter() is used for plotting scatter plot and scatter plot can be done for multiple datasets at the same time by differentiating it with colors. Legends help to achieve this difference.

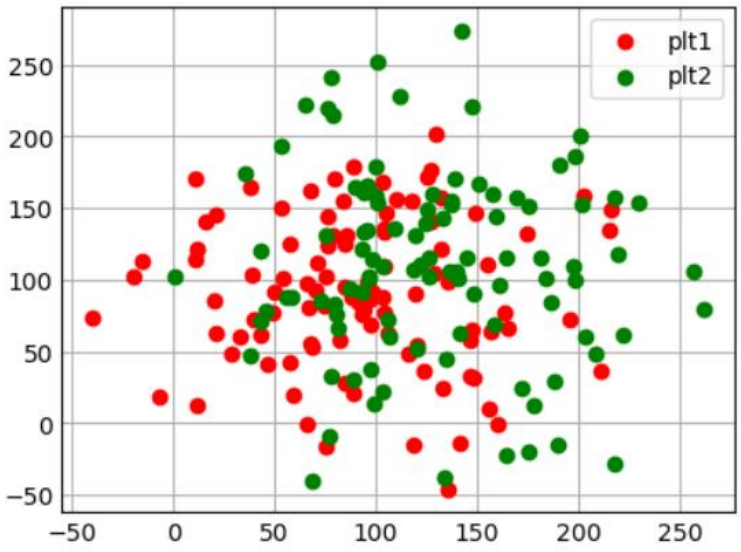
Code snippet:

```
import matplotlib.pyplot as plt
import numpy as np

x1=np.random.normal(100,50,100)
y1=np.random.normal(100,50,100)
x2=np.random.normal(120,60,100)
y2=np.random.normal(120,60,100)
fig = plt.figure(figsize = (4, 3))
plt.scatter(x1,y1,c="r")
plt.scatter(x2,y2,c="g")
plt.legend(["plt1","plt2"])

plt.show()
```

Output:



Description:

x1, y1 belong to one dataset and x2, y2 belong to another dataset.

Here the scatter() is used for the scatter plot and the parameters x, y and c which represents color.

Legend adds the label which helps to differentiate the plots.

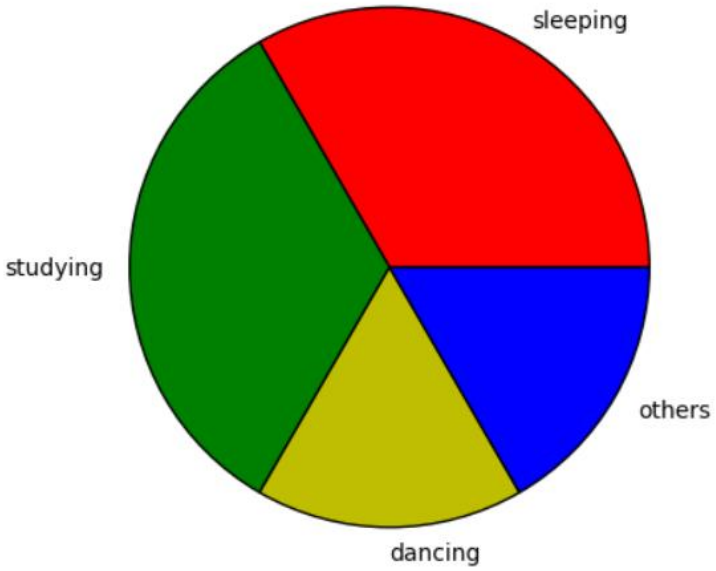
PIECHART

Pie charts are used to plot data of same kind which means the same series of data where the different elements are divide based on their percentage.

Code snippet:

```
from matplotlib import pyplot as plt
import numpy as np
time = [8,8,4,4]
color=["r","g","y","b"]
work= ["sleeping","studying","dancing","others"]
fig = plt.figure(figsize=(4, 4))
plt.pie(time, labels=work, wedgeprops={"edgecolor":"black","linewidth":1}, colors=color)
plt.show()
```

Output:



Description:

Here in the code, time is the x variable, and labels are defined using labels keyword.

wedgeprops attribute is used to define the linewidth and the color to separate the elements in the pie chart.

color array contains the colors to each element.

AREA CHART

In area chart, the area under the line to x axis is filled or shaded. It can be done by using fill_between() function or stackplot() function.

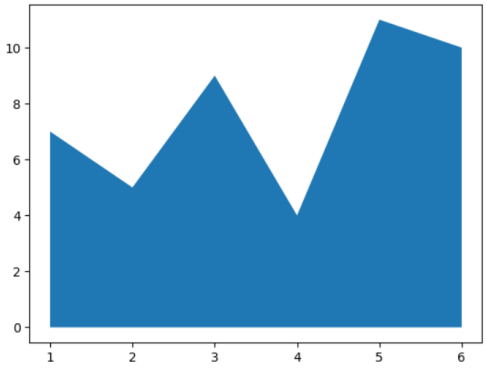
Code snippet:

Using fill_between() function

```
import numpy as np
import matplotlib.pyplot as plt


x=[1,2,3,4,5,6]
y=[7,5,9,4,11,10]

plt.fill_between(x, y)
plt.show()
```

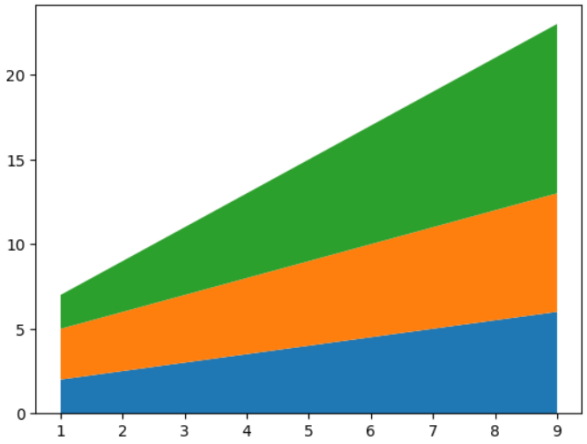


Using stackplot() function

```
import numpy as np
import matplotlib.pyplot as plt

x=[1,3,5,7,9]
y1=[2,3,4,5,6]
y2=[3,4,5,6,7]
y3=[2,4,6,8,10]

plt.stackplot(x,y1, y2, y3)
plt.show()
```



Description:

fill_between() function have parameters of x and y.

stackplot() function parameters are x, y1, y1, y3 where y1, y2, y3 are the different groups.

Legends are added by mentioning the labels attribute in stackplot() function and giving the position to the legend using plt.legend() function with attribute loc.

Comparison of Plotly and Matplotlib: -

1. Plotly

Plotly is a **high-level**, interactive visualization library that specializes in **web-based, dynamic plots**. It supports rich interactivity (hover effects, zooming, panning) and is widely used for dashboards, real-time data, and online reports.

Advantages of Plotly:

☑ **Built-in Interactivity** – Supports hover tooltips, zoom, pan, click events, and animations.

☑ **Easy Syntax** – Plotly Express(px) allows creating complex plots with minimal code.

☑ **Web & Dashboards** – Outputs interactive HTML plots, ideal for Dash apps and web embedding.

☑ **Modern Aesthetics** – Attractive default themes, 3D plots, and animations.

☑ **Real-time Updates** – Great for live data streaming and dynamic visualizations.

☑ **Works with Pandas** – Seamlessly plots Data Frames without manual data. reshaping.

2. Matplotlib

Matplotlib is a **low-level**, highly customizable Python library for creating static, animated, and interactive **2D/3D plots.** It is the foundation of many other plotting tools and is widely used in scientific publishing.

Advantages of Matplotlib:

☑ **Full Control –** Fine-tune every element (axes, ticks, legends, annotations).

☑ **Versatile Plot Types –** Supports basic (line, bar) to advanced plots (3D, contours, histograms).

☑ **Publication-Ready –** High-quality output for academic papers and print media (PDF/SVG).

☑ **Mature Ecosystem –** Integrates with NumPy, Pandas, SciPy, and Jupyter notebooks.

☑ **Extensive Customization –** Modify colors, fonts, grids, and styles manually.

☑ **Lightweight** – No external dependencies for basic plotting.