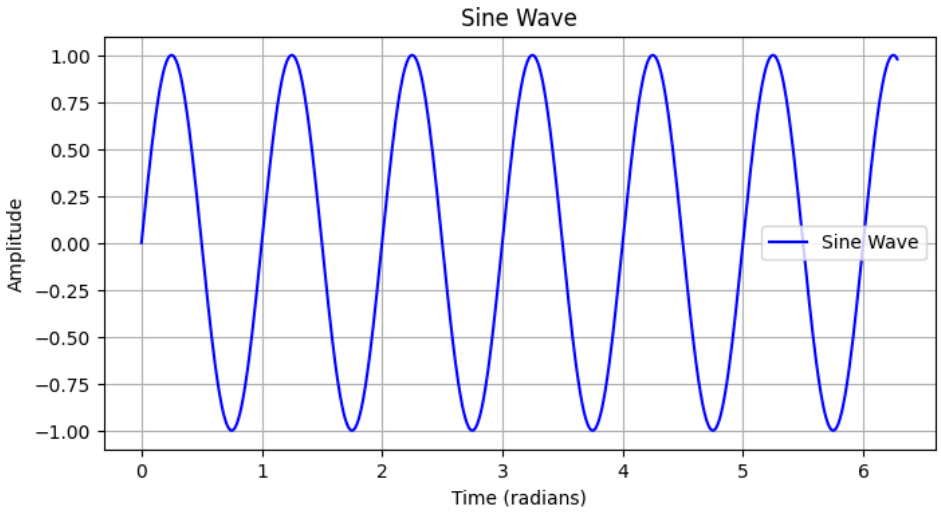
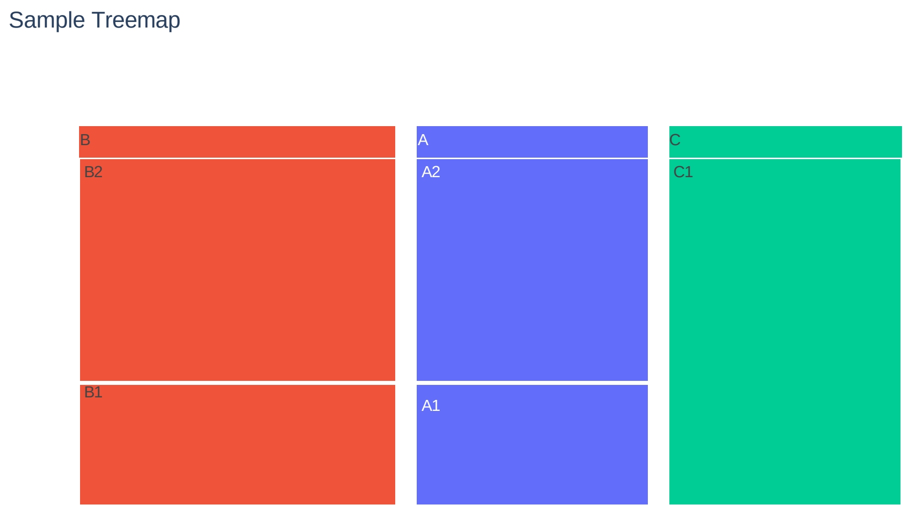
Name:-Prajkata Yuvraj Koli
Practical:-7
Roll no:-24 Sub:-DV

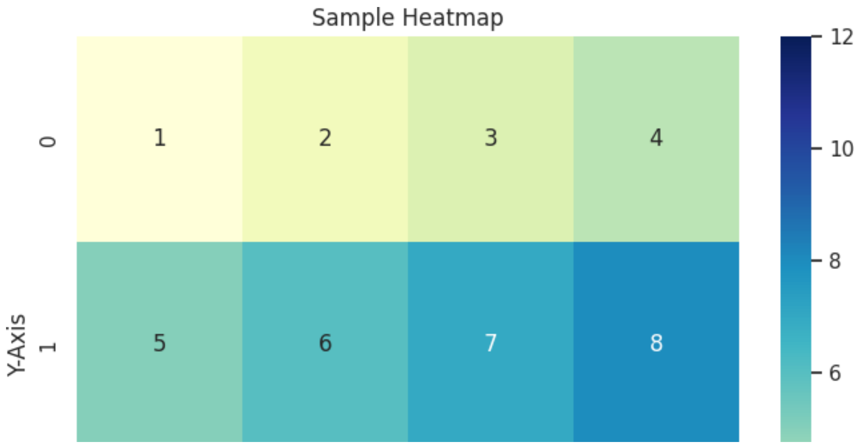
```
import numpy as np  # Import NumPy for numerical operations
import matplotlib.pyplot as plt  # Import Matplotlib for data visualization

# Step 1: Generate time values
t = np.linspace(0, 2 * np.pi, 1000)  # Create an array of time values from 0 to 2π

# Step 2: Define wave parameters
A = 1.0  # Amplitude of the wave
f = 1.0  # Frequency (1 cycle per 2π radians)
phi = 0  # Phase shift (no shift for this example)

# Step 3: Calculate the sine wave
y_sin = A * np.sin(2 * np.pi * f * t + phi)  # Calculate the sine wave

# Step 4: Create a figure and subplot
plt.figure(figsize=(8, 4))  # Create a figure with a specific size
plt.plot(t, y_sin, label='Sine Wave', color='blue')  # Plot the sine wave with a label and blue color
plt.title('Sine Wave')  # Set the title of the plot
plt.xlabel('Time (radians)')  # Label the x-axis
plt.ylabel('Amplitude')  # Label the y-axis
plt.grid(True)  # Add grid lines to the plot
plt.legend()  # Add a legend to the plot

# Step 5: Display the plot
plt.show()  # Show the plot
```



```
import plotly.express as px

# Sample hierarchical data
data = {
    'Category': ['A', 'A', 'B', 'B', 'C'],
    'Subcategory': ['A1', 'A2', 'B1', 'B2', 'C1'],
    'Value': [10, 20, 15, 25, 30]
}

# Create a DataFrame (or use an existing one)
import pandas as pd
df = pd.DataFrame(data)

# Create a treemap using Plotly Express
fig = px.treemap(
    df,  # DataFrame containing the data
    path=['Category', 'Subcategory'],  # Hierarchy defined by columns
    values='Value',  # Quantitative values to represent
    color='Category',  # Color-coded by 'Category' (optional)
    hover_name='Subcategory',  # Display subcategory names on hover (optional)
    title='Sample Treemap',  # Title for the treemap (optional)
    width=800,  # Width of the treemap (optional)
    height=500  # Height of the treemap (optional)
)

# Show the treemap
fig.show()
```



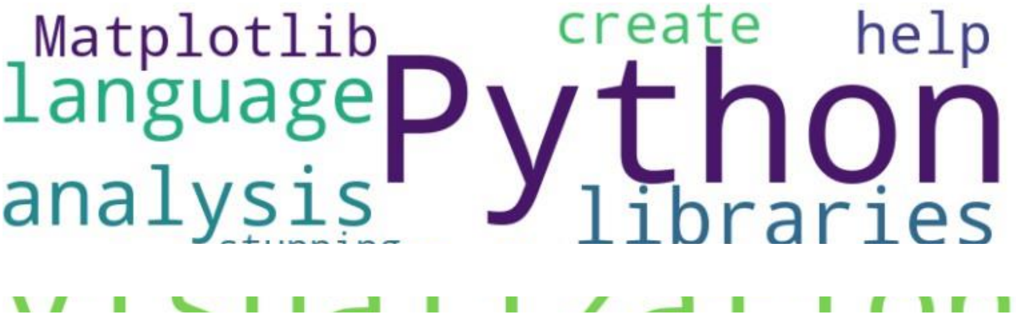
```
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data (a 2D array or DataFrame)
data = [
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12]
]

# Create a heatmap using Seaborn
sns.set()  # Set the default Seaborn style
plt.figure(figsize=(8, 6))  # Set the figure size

# Create the heatmap
sns.heatmap(data, annot=True, cmap="YlGnBu")

# Add labels and a title
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.title('Sample Heatmap')

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



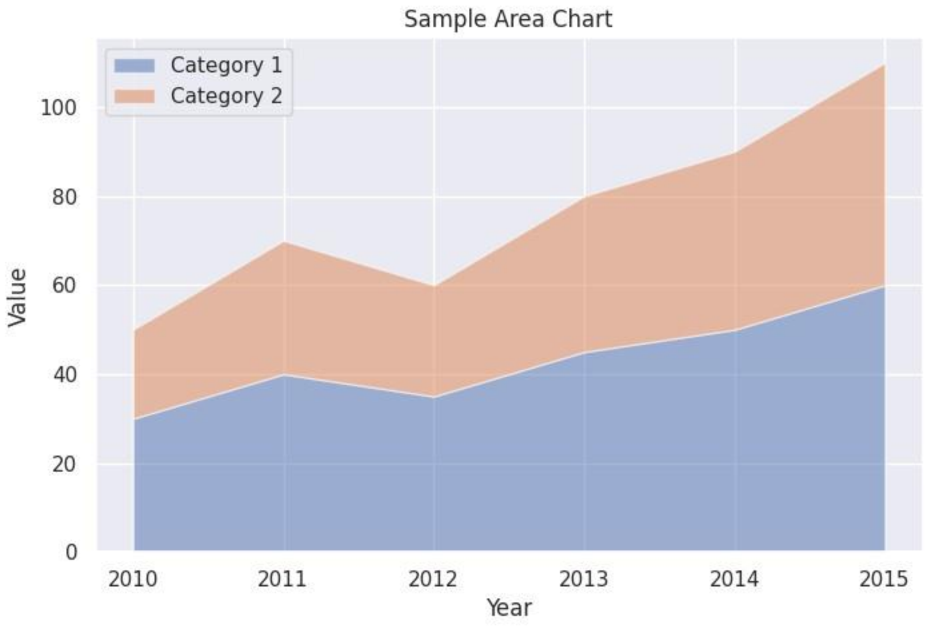
```
pip install wordcloud
```

```
Requirement already satisfied: wordcloud in c:\users\hp\appdata\local\programs\python\python310\lib\site-packages (1.9.2)
Requirement already satisfied: numpy>=1.6.1 in c:\users\hp\appdata\local\programs\python\python310\lib\site-packages (from wordcloud) (1.23.2)
Requirement already satisfied: pillow in c:\users\hp\appdata\local\programs\python\python310\lib\site-packages (from wordcloud) (9.2.0)
Requirement already satisfied: matplotlib in c:\users\hp\appdata\local\programs\python\python310\lib\site-packages (from wordcloud) (3.7.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\hp\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->wordcloud) (1.1.0)
Requirement already satisfied: cycler>=0.10 in c:\users\hp\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->wordcloud) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\hp\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->wordcloud) (4.42.0
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\hp\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->wordcloud) (1.4.4
Requirement already satisfied: packaging>=20.0 in c:\users\hp\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->wordcloud) (23.2)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in c:\users\hp\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->wordcloud) (3
Requirement already satisfied: python-dateutil>=2.7 in c:\users\hp\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->wordcloud) (2.
Requirement already satisfied: six>=1.5 in c:\users\hp\appdata\local\programs\python\python310\lib\site-packages (from python-dateutil>=2.7->matplotlib->wor
Note: you may need to restart the kernel to use updated packages.
WARNING: Ignoring invalid distribution -ip (c:\users\hp\appdata\local\programs\python\python310\lib\site-packages)
WARNING: Ignoring invalid distribution -treamlit (c:\users\hp\appdata\local\programs\python\python310\lib\site-packages)
WARNING: Ignoring invalid distribution -ip (c:\users\hp\appdata\local\programs\python\python310\lib\site-packages)
WARNING: Ignoring invalid distribution -treamlit (c:\users\hp\appdata\local\programs\python\python310\lib\site-packages)
WARNING: There was an error checking the latest version of pip.
```

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

text_data = "Python is a popular programming language used for data analysis and visualization. Python libraries like Matplotlib and Seaborn help create stunning visuali

wordcloud = WordCloud(
    width=800, height=400,
    background_color='white',
    colormap='viridis',       #colormap determines the color map for the word cloud.
    max_words=100       #max_words limits the number of words displayed in the word cloud.
)

wordcloud.generate(text_data)

plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')     #plt.imshow displays the word cloud.
                                                    # interpolation parameter is used to specify the interpolation method for displaying the image.
                                                     #Bilinear interpolation is a method for resampling an image to a different size.
plt.axis('off')  #plt.axis('off') removes axis labels and ticks.
plt.show()
```



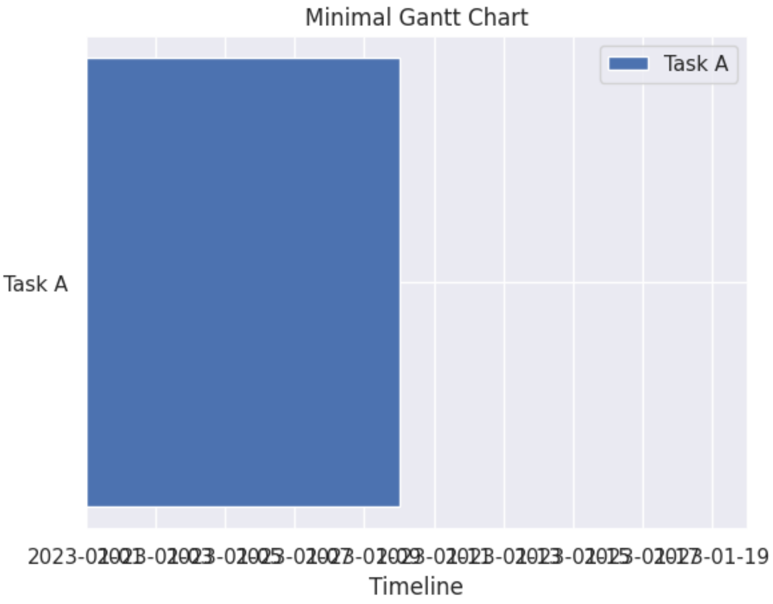
```
import matplotlib.pyplot as plt

# Sample data for an area chart
years = [2010, 2011, 2012, 2013, 2014, 2015]
category1 = [30, 40, 35, 45, 50, 60]
category2 = [20, 30, 25, 35, 40, 50]

# Create an area chart
plt.figure(figsize=(8, 5))  # Create a figure with specified dimensions

# Create the area chart using plt.stackplot
plt.stackplot(years, category1, category2, labels=['Category 1', 'Category 2'], alpha=0.5)
# - 'years' on the x-axis, 'category1' and 'category2' on the y-axis
# - 'labels' specify the legend labels for the data series
# - 'alpha' controls the transparency of the filled areas

plt.legend(loc='upper left')  # Add a legend to the upper-left corner
plt.title('Sample Area Chart')  # Set the title for the chart
plt.xlabel('Year')  # Label the x-axis
plt.ylabel('Value') # Label the y-axis

plt.show()  # Display the area chart
```



```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# 1. Import the necessary libraries, including matplotlib for data visualization and matplotlib.dates to work with dates.

# Define task names and their start and end dates
tasks = [
    {"task": "Task A", "start": "2023-01-01", "end": "2023-01-10"},
    {"task": "Task B", "start": "2023-01-05", "end": "2023-01-15"},
    {"task": "Task C", "start": "2023-01-12", "end": "2023-01-25"},
    {"task": "Task D", "start": "2023-01-20", "end": "2023-01-30"}
]

# 2. Define task data, including the task names and their start and end dates.

# Convert date strings to datetime objects
for task in tasks:
    task["start"] = mdates.datestr2num(task["start"])
    task["end"] = mdates.datestr2num(task["end"])

# 3. Convert the date strings in the task data to datetime objects using mdates.datestr2num() for compatibility with Matplotlib.

# Create a figure and axis
fig, ax = plt.subplots(figsize=(10, 4))

# 4. Create a figure and axis using plt.subplots() to prepare the plotting area.

# Plot Gantt bars for each task
 for i, task in enumerate(tasks):
    start_date = task["start"]
     end_date = task["end"]
    ax.barh(i, end_date - start_date, left=start_date, height=0.6, label=task["task"])

# 5. Iterate through the tasks and use ax.barh() to plot Gantt bars for each task. The left parameter specifies the start date, the height parameter controls the height of the b

# Customize the plot

# 6. Configure the x-axis as a date axis using ax.xaxis_date(), and format the date labels to display in the "YYYY-MM-DD" format.

# Set the x-axis limits to display the entire month of January 2023
ax.set_xlim(mdates.datestr2num("2023-01-01"), mdates.datestr2num("2023-01-31"))

# 7. Set the x-axis limits to display the entire month of January 2023.

# Label the x-axis and y-axis
ax.set_xlabel("Timeline")
ax.set_yticks(range(len(tasks)))
ax.set_yticklabels([task["task"] for task in tasks])

# 8. Label the x-axis as "Timeline" and set the y-axis ticks and labels to represent task names.

# Add a title and legend
plt.title("Gantt Chart Example")
plt.legend()

# 9. Add a title to the Gantt chart, and include a legend to identify the tasks.

# Show the Gantt chart
plt.show()

# 10. Finally, use plt.show() to display the Gantt chart.
```

Gantt Chart Example

```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# Define task details
task = {"task": "Task A", "start": "2023-01-01", "end": "2023-01-10"}

# Convert date strings to datetime objects
task["start"] = mdates.datestr2num(task["start"])
task["end"] = mdates.datestr2num(task["end"])

# Create a figure and axis
fig, ax = plt.subplots()

# Plot the Gantt bar for the task
ax.barh(0, task["end"] - task["start"], left=task["start"], height=0.6, label=task["task"])

# Customize the plot
ax.xaxis_date()
ax.set_xlim(mdates.datestr2num("2023-01-01"), mdates.datestr2num("2023-01-20"))
ax.set_xlabel("Timeline")
ax.set_yticks([0])
ax.set_yticklabels([task["task"]])
plt.title("Minimal Gantt Chart")
plt.legend(loc="upper right")

# Show the Gantt chart
plt.show()
```


Minimal Gantt Chart
Task A
Task A
Timeline