
USN

Internal Assessment Test 3 – July 2024

Sub:	Data Science and Visualization (Professional Elective)				Sub Code:	21CS644	Branch:	CSE	
Date:	31.7.2024	90 mins	Max Marks:	50	Sem/Sec:	VI/ A, B, C		OBE	

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	Explain the importance of data visualization with help of Matplotlib.pyplot. Discuss the anatomy of a Figure. Answer: importance -3 marks , Anatomy of figure- 7 marks	3+7	CO4	L2

Data visualization is essential in interpreting and understanding complex data, as it transforms numbers and statistics into visuals that are easier to analyze and communicate. In Python, `matplotlib.pyplot` is a powerful library that allows for creating a variety of static, animated, and interactive visualizations. Here's a breakdown of why data visualization with `matplotlib.pyplot` is so important:

1. Simplifying Complex Data

- **Purpose**: Visualization helps make sense of complex data by breaking it down into graphs and charts.
- **Example with `matplotlib.pyplot`**: Using a bar chart to compare sales figures across multiple regions simplifies large datasets, making trends more apparent.

2. Uncovering Patterns and Trends

- **Purpose**: Visualizations reveal underlying trends, allowing businesses to identify opportunities and areas for improvement.
- **Example with `matplotlib.pyplot`**: A line plot can highlight seasonal changes in sales or demand, aiding in strategic planning.

3. Improving Decision Making

- **Purpose**: Data visualization aids decision-making by providing a clear picture of potential outcomes.

Example with **`matplotlib.pyplot`**: A pie chart can display the market share of different products, allowing for better resource allocation. 4. Communicating Insights Effectively

- **Purpose**: Visualizations make data accessible to non-technical audiences.
- **Example with `matplotlib.pyplot`**: A scatter plot can be used to demonstrate the relationship between two variables, such as advertising spend vs. revenue.

4. Communicating Insights Effectively

- **Purpose**: Visualizations make data accessible to non-technical audiences.

- **Example with `matplotlib.pyplot`**: A scatter plot can be used to demonstrate the relationship between two variables, such as advertising spend vs. revenue.

Anatomy of a figure:

In data visualization using libraries like Matplotlib, the concept of a **Figure** is fundamental. A Figure represents the entire window or canvas where you create visualizations, and it can contain one or more **Axes**, which are the areas where the actual data is plotted. Understanding the anatomy of a Figure helps in creating effective visualizations. Here's a breakdown of the key components:

1. Figure

- **Definition**: The Figure is the top-level container for all elements of the plot. It can be thought of as the entire plotting area or canvas.
- **Attributes**: It can have properties such as size, background color, and DPI (dots per inch) which affects the resolution.
- **Creation**: You can create a Figure using `plt.figure()` or by creating a plot directly (e.g., `plt.plot()`), which implicitly creates a Figure.

2. Axes

- **Definition**: An Axes is the area on which data is plotted. A Figure can contain multiple Axes, allowing for multiple subplots in one Figure.
- **Characteristics**: Each Axes has its own set of X and Y axes, which can have different scales, limits, and labels.
- **Creation**: You can create Axes using `fig.add_axes()` or `plt.subplot()`, and they are where the actual plotting commands are applied.

3. Labels

- **X-axis and Y-axis Labels**: Labels for the X and Y axes provide context to the data being visualized.
- **Title**: A title for the Figure gives a summary or context of what the plot represents.

4. Ticks and Tick Labels

- **Ticks**: These are the markers on the axes that represent specific values.
- **Tick Labels**: The numbers or labels associated with each tick, indicating the scale or categories represented.

5. Legend

- **Definition**: A legend explains the elements of the plot, indicating what different colors, shapes, or line styles represent.
- **Creation**: It can be added using `ax.legend()` and is particularly useful when multiple datasets are plotted together.

2	a) No.of overs = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] run_scored = [5, 12, 20, 26, 30, 42, 55, 60, 75, 90] Which plot is suitable for finding the relationship between the above given set of values. Write a python code to illustrate the same. b) Describe following functions: np.arange(), np.linspace(), plt.plot(), plt.gca(), plt.barh()	5+5	CO5	L3

Answer: a) ,b) – 5 marks eaach

a) To find the relationship between the number of overs and the runs scored, a **scatter plot** is suitable. A scatter plot helps visualize the relationship between two continuous variables, making it easy to identify trends or correlations.

```
import matplotlib.pyplot as plt

# Data
no_of_overs = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
runs_scored = [5, 12, 20, 26, 30, 42, 55, 60, 75, 90]

# Create scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(no_of_overs, runs_scored, color='blue', marker='o')

# Add labels and title
plt.title('Relationship between Overs and Runs Scored')
plt.xlabel('Number of Overs')
plt.ylabel('Runs Scored')

# Optionally, add a trend line
# Fit a polynomial of degree 1 (linear) to the data
coefficients = np.polyfit(no_of_overs, runs_scored, 1)
polynomial = np.poly1d(coefficients)
plt.plot(no_of_overs, polynomial(no_of_overs), color='red', linestyle='--', label='Trend Line')

# Add legend
plt.legend()

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

b) 1. `np.arange()`

Description: `np.arange()` is a NumPy function used to create an array of evenly spaced values within a specified range. It's similar to Python's built-in `range()` function but returns a NumPy array instead.

Syntax:

```
python
Copy code
numpy.arange([start, ]stop[, step])
```

- **Parameters**:

- `start`: (optional) The starting value of the sequence (default is 0).
- `stop`: The end value of the sequence (exclusive).
- `step`: (optional) The spacing between values (default is 1).

Example:

```
python
Copy code
import numpy as np

# Create an array from 0 to 9
arr = np.arange(10)
print(arr)  # Output: [0 1 2 3 4 5 6 7 8 9]

# Create an array from 1 to 10 with a step of 2
arr_step = np.arange(1, 10, 2)
print(arr_step)  # Output: [1 3 5 7 9]
```

2. `np.linspace()`

Description: `np.linspace()` is a NumPy function that creates an array of evenly spaced numbers over a specified range. Unlike `np.arange()`, which specifies the step size, `np.linspace()` specifies the number of samples.

Syntax:

```
python
Copy code
numpy.linspace(start, stop, num=50, endpoint=True, retstep=False)
```

- **Parameters**:
  - `start`: The starting value of the sequence.
  - `stop`: The end value of the sequence (inclusive by default).
  - `num`: (optional) The number of samples to generate (default is 50).
  - `endpoint`: (optional) If True, `stop` is the last sample (default is True).
  - `retstep`: (optional) If True, return the samples and the step size.

Example:

```
python
Copy code
import numpy as np

# Create an array of 5 evenly spaced values between 0 and 1
linspace_arr = np.linspace(0, 1, num=5)
print(linspace_arr)  # Output: [0.   0.25 0.5  0.75 1.  ]
```

3. `plt.plot()`

Description: `plt.plot()` is a Matplotlib function used to create a 2D line plot. It can take various parameters to customize the appearance of the plot, such as color, marker, and line style.

Syntax:

```
python
Copy code
matplotlib.pyplot.plot(x, y, fmt, *args, **kwargs)
```

- **Parameters**:

- `x`: The data for the x-axis.
- `y`: The data for the y-axis.
- `fmt`: (optional) A format string that specifies the color and line style.
- `*args`: Additional arguments for customization.
- `**kwargs`: Keyword arguments for further customization.

Example:

```
python
Copy code
import matplotlib.pyplot as plt
import numpy as np

# Data
x = np.linspace(0, 10, 100)
y = np.sin(x)

# Create a line plot
plt.plot(x, y, color='blue', linestyle='--', marker='o')

# Add labels and title
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Sine Wave')

# Show plot
plt.show()
```

4. `plt.gca()`

Description: `plt.gca()` (Get Current Axes) is a Matplotlib function that returns the current axes instance on the current figure. It allows you to modify properties of the axes without explicitly storing them in a variable.

Syntax:

```
python
Copy code
matplotlib.pyplot.gca()
```

Example:

```
python
Copy code
import matplotlib.pyplot as plt

# Create a simple plot
plt.plot([1, 2, 3], [4, 5, 6])

# Get current axes
ax = plt.gca()

# Modify properties
ax.set_title('Current Axes Title')
ax.set_xlabel('X Axis')
ax.set_ylabel('Y Axis')

# Show plot
plt.show()
```

	5. `plt.barh()` **Description**: `plt.barh()` is a Matplotlib function used to create horizontal bar charts. It is similar to `plt.bar()` but draws bars horizontally. **Syntax**: ``` python Copy code matplotlib.pyplot.barh(y, width, height=0.8, left=None, align='center', **kwargs) ``` - **Parameters**: - `y`: The y-coordinates of the bars (the categories). - `width`: The width of the bars (the values). - `height`: (optional) The height of the bars (default is 0.8). - `left`: (optional) The x-coordinates of the left sides of the bars (default is 0). - `align`: (optional) Alignment of the bars (default is 'center'). - `**kwargs`: Additional keyword arguments for customization (color, edge color, etc.). **Example**: ``` python Copy code import matplotlib.pyplot as plt # Data categories = ['A', 'B', 'C', 'D'] values = [3, 7, 5, 4] # Create a horizontal bar chart plt.barh(categories, values, color='green') # Add title and labels plt.title('Horizontal Bar Chart Example') plt.xlabel('Values') plt.ylabel('Categories') # Show plot plt.show() ```			
3	What are the different text and legend functions available in Matplotlib? Explain the usage of labels, titles, text annotations, and legends with suitable code snippets. Answer: 5+5 marks Matplotlib provides several functions to add text elements and legends to your plots. These features are crucial for making plots informative and easier to understand. Here's an overview of the different text and legend functions, along with their usage through code snippets. 1. Labels Labels are used to describe the axes of the plot. - **Usage**:	5+5	CO4	L2

- `set_xlabel()`: Sets the label for the x-axis.
- `set_ylabel()`: Sets the label for the y-axis.

2. Title

The title provides a brief description of what the plot is about.

- **Usage**:
  - `title()`: Sets the title of the plot.

3. Text Annotations

Annotations can add additional information at specific points on the plot, helping to clarify or highlight important data points.

- **Usage**:
  - `text()`: Adds text at a specified location in data coordinates.
  - `annotate()`: Adds an annotation with an arrow pointing to a specific point.

4. Legends

Legends are used to describe different data series in a plot, especially when multiple datasets are plotted together.

- **Usage**:
  - `legend()`: Adds a legend to the plot.

```
# Sample data

y1 = np.sin(x)

y2 = np.cos(x)

# Create a plot

plt.plot(x, y1, label='Sine Wave')  # Label for sine wave

plt.plot(x, y2, label='Cosine Wave')  # Label for cosine wave

# Set labels and title

plt.xlabel('X Axis')

plt.ylabel('Y Axis')

plt.title('Sine and Cosine Waves')



# Add legend

plt.legend()
```

	# Show plot plt.show()			
4	Discuss the different types of basic plots available in Matplotlib such as bar charts, pie charts, histograms boxplot and scatter plots. Provide examples demonstrating how to create and customize these plots. Answer: Matplotlib is a powerful visualization library in Python that supports a variety of basic plots, each suited for different types of data and analyses. Here's a discussion of some common plot types—bar charts, pie charts, histograms, box plots, and scatter plots—along with examples demonstrating how to create and customize them. 1. Bar Charts **Description**: Bar charts are used to represent categorical data with rectangular bars. The height of each bar corresponds to the value it represents. **Example**: python Copy code import matplotlib.pyplot as plt # Data categories = ['A', 'B', 'C', 'D'] values = [10, 15, 7, 20] # Create a bar chart plt.bar(categories, values, color='skyblue') # Add title and labels plt.title('Bar Chart Example') plt.xlabel('Categories') plt.ylabel('Values') # Show plot plt.show() 2. Pie Charts **Description**: Pie charts display the proportions of categories as slices of a circle. They are useful for showing relative sizes of parts to a whole. **Example**: python Copy code # Data sizes = [30, 25, 20, 25] labels = ['Category A', 'Category B', 'Category C', 'Category D'] colors = ['gold', 'lightcoral', 'lightskyblue', 'lightgreen'] # Create a pie chart	10	CO5	L2

```
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%',
startangle=140)

# Add title
plt.title('Pie Chart Example')

# Show plot
plt.axis('equal')  # Equal aspect ratio ensures that pie chart is
circular
plt.show()
```

3. Histograms

Description: Histograms are used to represent the distribution of a dataset. They display the frequency of data points within specified ranges (bins).

Example:

```
python
Copy code
import numpy as np

# Generate random data
data = np.random.randn(1000)

# Create a histogram
plt.hist(data, bins=30, color='purple', alpha=0.7)

# Add title and labels
plt.title('Histogram Example')
plt.xlabel('Value')
plt.ylabel('Frequency')

# Show plot
plt.show()
```

4. Box Plots

Description: Box plots provide a summary of a dataset using five statistics: minimum, first quartile, median, third quartile, and maximum. They are useful for identifying outliers and comparing distributions.

Example:

```
python
Copy code
# Generate random data
data = [np.random.normal(0, std, 100) for std in range(1, 4)]

# Create a box plot
plt.boxplot(data, vert=True, patch_artist=True, labels=['Group 1',
'Group 2', 'Group 3'])

# Add title and labels
plt.title('Box Plot Example')
plt.ylabel('Value')

# Show plot
plt.show()
```

5. Scatter Plots

Description: Scatter plots are used to observe relationships between two numerical variables. Each point represents an observation.

Example:

```
python
Copy code
# Data
x = np.random.rand(50)
y = np.random.rand(50)

# Create a scatter plot
plt.scatter(x, y, color='red', alpha=0.5)

# Add title and labels
plt.title('Scatter Plot Example')
plt.xlabel('X Values')
plt.ylabel('Y Values')

# Show plot
plt.show()
```

Customization Options

Matplotlib provides various customization options for these plots, including:

- **Colors**: Use the `color` parameter to change the color of the plot elements.
- **Transparency**: The `alpha` parameter controls transparency, with 0 being fully transparent and 1 being fully opaque.
- **Labels**: Use `xlabel()`, `ylabel()`, and `title()` to add descriptive labels and titles.
- **Grid**: Enable grids using `plt.grid(True)` for better readability.

5	Explain the concept of subplots and layout management in Matplotlib. How do tight layout, GridSpec, and radar charts enhance the presentation of multiple plots? Include examples to support your explanation.	5+5	CO4	L2

Answer:

In Matplotlib, subplots and layout management allow you to create complex figures containing multiple plots. This is especially useful for comparing data across different categories or visualizing different aspects of a dataset. Here's a detailed explanation of subplots, layout management techniques (including tight layout, GridSpec), and the concept of radar charts, along with examples.

1. Subplots

Subplots are individual plots that share the same figure space. They can be arranged in a grid and allow for the visualization of multiple datasets or different perspectives on the same dataset within a single figure.

Creating Subplots:

- You can create subplots using the `plt.subplot()` function or the `plt.subplots()` function.

Example:

```
python
Copy code
import matplotlib.pyplot as plt
import numpy as np

# Sample data
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

# Create subplots
fig, axs = plt.subplots(2, 1)  # 2 rows, 1 column

# Plotting in the first subplot
axs[0].plot(x, y1, color='blue')
axs[0].set_title('Sine Function')
axs[0].set_ylabel('sin(x)')

# Plotting in the second subplot
axs[1].plot(x, y2, color='orange')
axs[1].set_title('Cosine Function')
axs[1].set_ylabel('cos(x)')
axs[1].set_xlabel('X Axis')

# Show plot
plt.tight_layout()  # Adjust layout to prevent overlap
plt.show()
```

2. Layout Management

Matplotlib provides several layout management techniques to optimize the presentation of multiple plots:

a. Tight Layout

The `tight_layout()` function automatically adjusts subplot parameters to give specified padding and prevent overlapping content.

Example:

```
python
Copy code
# Same data as before
fig, axs = plt.subplots(2, 1)

# Plotting
axs[0].plot(x, y1)
axs[1].plot(x, y2)

# Using tight_layout to avoid overlap
plt.tight_layout()
plt.show()
```

b. GridSpec

`GridSpec` provides more flexible subplot arrangements by allowing for different row and column spans for subplots. It gives more control over the layout compared to the basic `subplot()` method.

Example:

```
python
Copy code
from matplotlib.gridspec import GridSpec

# Create a GridSpec object
gs = GridSpec(3, 2)  # 3 rows, 2 columns

# Create subplots
fig = plt.figure(figsize=(10, 6))

# Adding subplots with GridSpec
ax1 = fig.add_subplot(gs[0, :])  # First row spans all columns
ax2 = fig.add_subplot(gs[1, 0])  # Second row, first column
ax3 = fig.add_subplot(gs[1, 1])  # Second row, second column
ax4 = fig.add_subplot(gs[2, :])  # Third row spans all columns

# Plotting
ax1.plot(x, y1)
ax2.plot(x, y2)
ax3.plot(x, -y1)
ax4.plot(x, -y2)

# Titles and layout
ax1.set_title('Sine Function')
ax2.set_title('Cosine Function')
ax3.set_title('Negative Sine Function')
ax4.set_title('Negative Cosine Function')
plt.tight_layout()
plt.show()
```

3. Radar Charts

Radar charts (or spider charts) allow you to visualize multivariate data in a two-dimensional chart of three or more quantitative variables represented on axes starting from the same point. They are useful for comparing multiple variables for different groups.

Example:

```
python
Copy code
# Radar chart example
import numpy as np
import matplotlib.pyplot as plt

# Define data
labels = np.array(['A', 'B', 'C', 'D'])
values = np.array([4, 3, 2, 5])
values = np.concatenate((values,[values[0]]))  # Repeat the first
value to close the loop

# Create radar chart
angles = np.linspace(0, 2 * np.pi, len(labels),
endpoint=False).tolist()
```

```
values = np.concatenate((values,[values[0]]))  # Repeat the first
value to close the loop
angles += angles[:1]  # Close the loop for angles

fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(polar=True))
ax.fill(angles, values, color='orange', alpha=0.25)
ax.plot(angles, values, color='orange', linewidth=2)

# Add labels
ax.set_yticklabels([])
ax.set_xticks(angles[:-1])
ax.set_xticklabels(labels)
ax.set_title('Radar Chart Example')

# Show plot
plt.show()
```

6	Describe the basic image operations that can be performed using Matplotlib. How can mathematical expressions be written and displayed in plots? Provide examples of image manipulation and incorporating mathematical expressions in plots.	3+3+4	CO5	L2

Answer:

Matplotlib provides a variety of functions for basic image operations and allows for the incorporation of mathematical expressions into plots. Here's an overview of the basic image operations you can perform with Matplotlib, along with examples of image manipulation and displaying mathematical expressions.

Basic Image Operations with Matplotlib

1. **Displaying Images**: You can display images using `plt.imshow()`, which takes an array representing image data.
2. **Adjusting Image Properties**: You can modify image properties such as color maps, interpolation, and aspect ratio.
3. **Manipulating Image Data**: This includes basic transformations like rotating, flipping, and resizing.
4. **Overlaying Annotations**: You can overlay text, lines, and other shapes on images for labeling or highlighting features.

Example of Basic Image Operations

Displaying and Modifying an Image:

```
python
Copy code
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import image as mpimg

# Load an image
img = mpimg.imread('example_image.png')  # Load your image file

# Display the original image
plt.subplot(1, 2, 1)
plt.imshow(img)
plt.title('Original Image')
plt.axis('off')  # Hide axes

# Display a modified image (grayscale)
```

```
gray_img = np.mean(img, axis=2)  # Convert to grayscale by
averaging color channels
plt.subplot(1, 2, 2)
plt.imshow(gray_img, cmap='gray')
plt.title('Grayscale Image')
plt.axis('off')  # Hide axes

# Show both images
plt.tight_layout()
plt.show()
```

In this example:

- We load an image using `mpimg.imread()`.
- We display the original and a grayscale version of the image side by side using `plt.imshow()`.

Mathematical Expressions in Plots

Matplotlib supports LaTeX-like syntax for rendering mathematical expressions in plots. You can use dollar signs (`$`) to enclose the mathematical expressions you want to display.

Example of Incorporating Mathematical Expressions

Displaying Mathematical Expressions:

```
python
Copy code
import matplotlib.pyplot as plt
import numpy as np

# Data for plotting
x = np.linspace(0, 2 * np.pi, 100)
y1 = np.sin(x)
y2 = np.cos(x)

# Create the plot
plt.plot(x, y1, label=r'$\sin(x)$', color='blue')
plt.plot(x, y2, label=r'$\cos(x)$', color='red')

# Adding titles and labels
plt.title('Trigonometric Functions')
plt.xlabel(r'$x$ (radians)')
plt.ylabel(r'$f(x)$')
plt.legend()

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

In this example:

- We plot the sine and cosine functions.
- Mathematical expressions for the sine and cosine functions are incorporated using `r'$\sin(x)$'` and `r'$\cos(x)$'`.
- The `r` before the string indicates that it is a raw string, which is useful for LaTeX formatting.

4	Discuss the different types of basic plots available in Matplotlib such as bar charts, pie charts, histograms boxplot and scatter plots. Provide examples demonstrating how to create and customize these plots.	10	CO5	L2
5	Explain the concept of subplots and layout management in Matplotlib. How do tight layout, GridSpec, and radar charts enhance the presentation of multiple plots? Include examples to support your explanation.	5+5	CO4	L2
6	Describe the basic image operations that can be performed using Matplotlib. How can mathematical expressions be written and displayed in plots? Provide examples of image manipulation and incorporating mathematical expressions in plots.	3+3+4	CO5	L2

CI CCI HoD