Python Programming

Lab:- 23(Matplotlib more plots and formatting)

Student Id:- AF0417098

Student Name:- Ankush

Matplotlib:- **It is a widely-used Python library for creating static, animated, and interactive visualizations in Python. It provides a flexible framework for creating a wide range of plots and charts.**

Here are some key points about Matplotlib more plots and formatting:-

When working with Matplotlib to create complex and professional visualizations, understanding how to utilize multiple plots and advanced formatting options is essential. Below are key points for enhancing your plots and formatting in Matplotlib:

1. Multiple Plots

- Subplots: You can use plt.subplots() to create multiple plots within the same figure. This allows you to compare different datasets side-by-side, either vertically or horizontally.

  Example: fig, axs = plt.subplots(2, 2) creates a 2x2 grid of subplots.

  Share Axes: Use sharex or sharey parameters in plt.subplots() to create subplots that share the same axis.

- Multiple Figures: You can create multiple independent figures using plt.figure(). This allows you to keep different visualizations in separate windows or files.

2. Annotations

- **Data Labels:** Add labels to points, bars, or lines for clarity. Use plt.text() or ax.annotate() to display specific values.

  Example: Adding values on top of bars in a bar chart.

- **Trendlines:** You can fit a line through your data points using libraries like NumPy and plot the trend using plt.plot().

3. Customizing Axes and Ticks

- **Axes Labels and Titles:** Use plt.xlabel(), plt.ylabel(), and plt.title() to add labels and titles to your plots.

  Font Customization: You can adjust the font size, weight, and color. Example: plt.title('Title', fontsize=16, fontweight='bold').

- **Ticks Customization:** Customize the ticks (values along the axes) using plt.xticks() and plt.yticks(). You can control their location, labels, and orientation.

  Rotation: For long labels, rotate them using plt.xticks(rotation=45).

- **Logarithmic Scale:** Use plt.xscale('log') or plt.yscale('log') to apply logarithmic scales to your axes for better handling of data with large ranges.

4. Colors and Colormaps

- **Custom Colors:** You can specify colors in your plots using color names ('red'), hex codes ('#FF5733'), or RGBA values.

- **Colormaps:** Matplotlib has built-in colormaps that can be applied to various plots, especially heatmaps or scatter plots. Use plt.colormaps() to explore the available options.

Example: plt.scatter(x, y, c=z, cmap='viridis').

5. Legends and Gridlines

- **Legends:** Add a legend to your plot using plt.legend(). You can customize the position, font size, and appearance of the legend.

  Positioning: Use loc='best' to automatically find the best location for the legend.

- **Gridlines:** Add gridlines using plt.grid(True) and customize their appearance with line styles (linestyle='--'), colors, and transparency (alpha=0.7).

  6. **Line Styles and Markers**

- **Line Styles:** Customize the appearance of lines in line plots using linestyle='--' for dashed lines, or linestyle='-.' for dash-dot lines.

- **Markers:** Add markers to highlight data points in line or scatter plots using marker='o' (circle), marker='x' (cross), or other marker styles.

  Example: plt.plot(x, y, linestyle='-', marker='o', color='r').

  7. **Plotting Multiple Data Sets**

- **Overlaying Plots:** You can plot multiple datasets on the same plot using multiple plt.plot() or plt.scatter() calls. Make sure to use legends and distinct styles to differentiate between them.

  Example: Plotting multiple line plots in one figure using different colors and markers.

- **Secondary Axes:** Create a second Y-axis for the same plot using ax.twinx() to show two different scales on the same plot.

  Example: Plotting temperature and humidity on two Y-axes.

  8. **Saving Figures**

- **High-Quality Output:** Save your plots as high-resolution images using plt.savefig('filename.png', dpi=300) for publication-quality images.

You can also save in different formats like .pdf, .svg, and .eps.

9. Interactive Plots

- **Interactive Mode:** Use plt.ion() for interactive plotting, which allows real-time updates to figures.
- **Zooming and Panning:** Matplotlib's interactive features in the plot window allow for zooming and panning. You can also use matplotlib.widgets for adding interactive sliders, buttons, and more.

Assignment Questions:-

Ques1:- What is pie chart and give some important points of pie chart.

Introduction to Pie Charts:

A pie chart is a circular graph used to represent proportional data. The chart is divided into slices, where each slice corresponds to a category or group, and the size of each slice represents the proportion or percentage of the whole. Pie charts are commonly used in business reports, marketing analytics, and social statistics to show how different parts contribute to a total.

While pie charts are visually appealing and easy to understand for small datasets, they are less effective when comparing categories with similar proportions or when there are too many categories. In such cases, alternative charts like bar charts may be more appropriate.

Program:-

lab23.py > ...

```python
import matplotlib.pyplot as plt

# Market share data
manufacturers = ['Samsung', 'Apple', 'Huawei', 'Xiaomi', 'Others']
market_share = [30, 25, 18, 12, 15]

# Colors for each slice
colors = ['lightblue', 'lightcoral', 'lightgreen', 'lightpink', 'lightgrey']

# Explode the largest slice (Samsung) to highlight it
explode = [0.1, 0, 0, 0, 0]

# Create the pie chart with additional customizations
plt.figure(figsize=(8, 6))  # Adjust figure size for clarity
plt.pie(market_share, labels=manufacturers, colors=colors, autopct='%1.1f%%',
        explode=explode, shadow=True, startangle=90)

# Title
plt.title('Smartphone Market Share (2024)', fontsize=16, fontweight='bold')

# Add a legend for better clarity
plt.legend(manufacturers, title="Manufacturers", loc="best")

# Show the pie chart
plt.tight_layout()  # Adjust layout to prevent overlaps
plt.show()
```

Pie plots are commonly used to visualize data with clear categorical distinctions, such as:

● Market share of different products or companies.

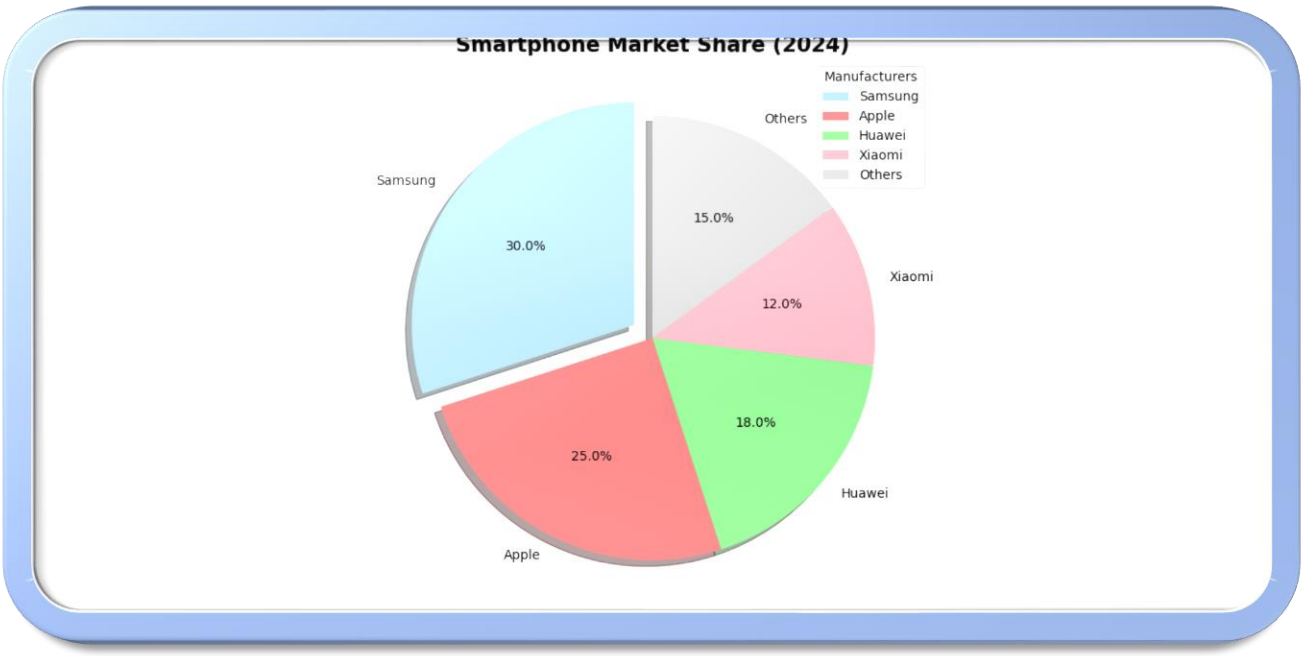
● Budget allocation in various spending categories.

● Demographic breakdowns, such as age groups in a population.

● The composition of a portfolio's investments.

● Distribution of survey responses in various answer choices.

Output:-



Ques 2. What is Scatter plot chart and give some important points of Scatter plot chart.

Brief Introduction to Scatter Plot:

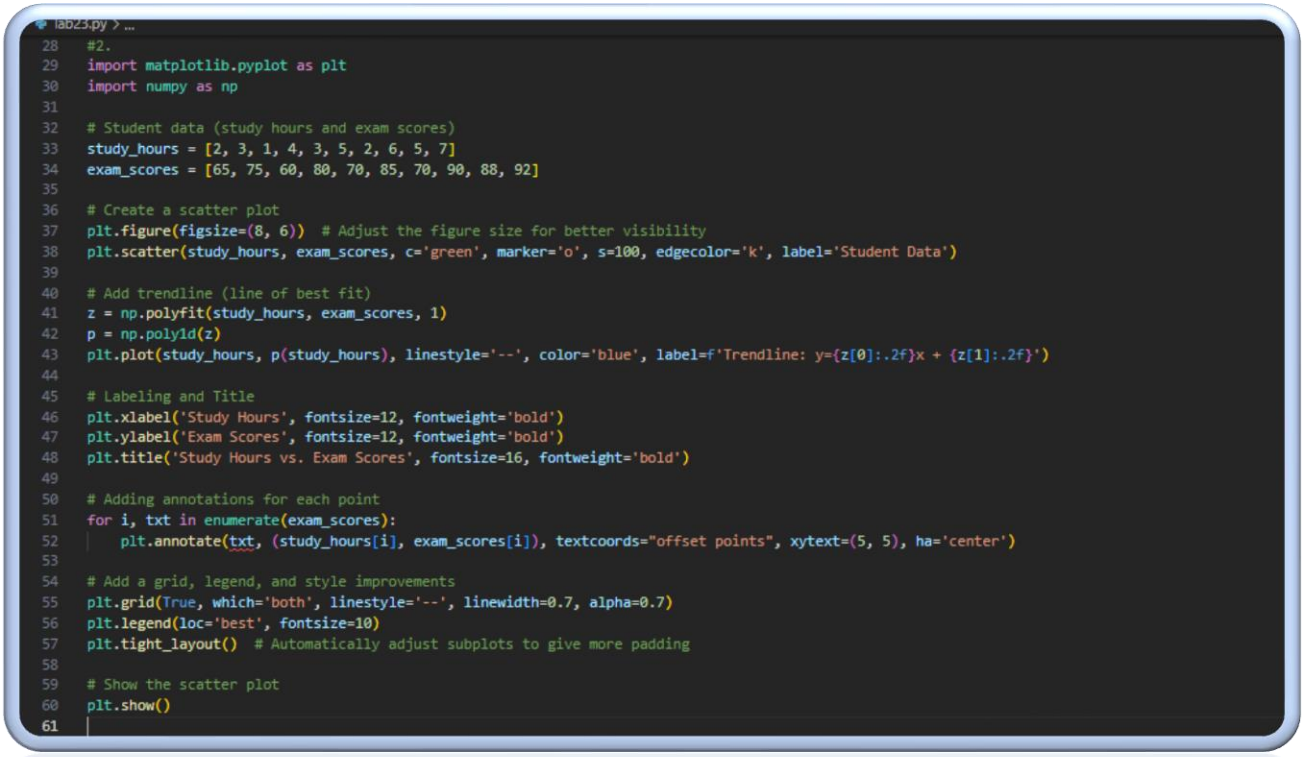
A scatter plot (or scatter diagram) is a type of chart used to display relationships between two continuous variables. Each point on the scatter plot represents an observation, with the position of the point determined by the values of the two variables. Scatter plots are commonly used to identify patterns or correlations (e.g., positive, negative, or no correlation) between the variables.

Key Points:

- **X and Y Axes:** Each point is plotted using an X and a Y coordinate, where the X-axis typically represents one variable and the Y-axis represents another.
- **Correlation:** Scatter plots can help visualize if there is a relationship between the variables (e.g., as X increases, Y may increase or decrease).

- **Outliers: Scatter plots can help easily identify outliers, which are data points that deviate significantly from the overall pattern.**
- **Trendline: A line of best fit (trendline) can be added to highlight the overall trend in the data.**

Program:-



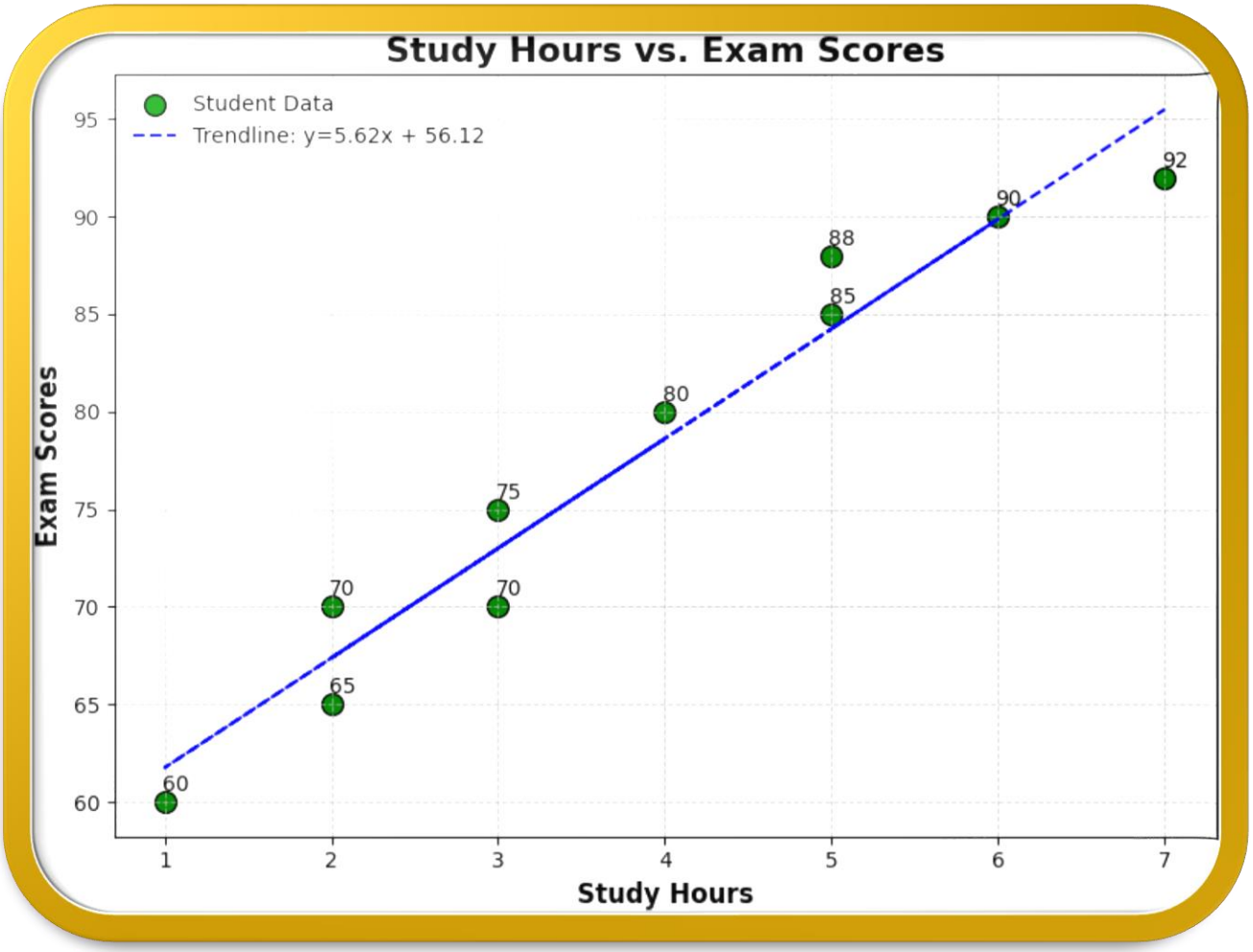
```
lab23.py > ...
#2.
import matplotlib.pyplot as plt
import numpy as np

# Student data (study hours and exam scores)
study_hours = [2, 3, 1, 4, 3, 5, 2, 6, 5, 7]
exam_scores = [65, 75, 60, 80, 70, 85, 70, 90, 88, 92]

# Create a scatter plot
plt.figure(figsize=(8, 6))  # Adjust the figure size for better visibility
plt.scatter(study_hours, exam_scores, c='green', marker='o', s=100, edgecolor='k', label='Student Data')

# Add trendline (line of best fit)
z = np.polyfit(study_hours, exam_scores, 1)
p = np.poly1d(z)
plt.plot(study_hours, p(study_hours), linestyle='--', color='blue', label=f'Trendline: y={z[0]:.2f}x + {z[1]:.2f}')

# Labeling and Title
plt.xlabel('Study Hours', fontsize=12, fontweight='bold')
plt.ylabel('Exam Scores', fontsize=12, fontweight='bold')
plt.title('Study Hours vs. Exam Scores', fontsize=16, fontweight='bold')

# Adding annotations for each point
for i, txt in enumerate(exam_scores):
    plt.annotate(txt, (study_hours[i], exam_scores[i]), textcoords="offset points", xytext=(5, 5), ha='center')

# Add a grid, legend, and style improvements
plt.grid(True, which='both', linestyle='--', linewidth=0.7, alpha=0.7)
plt.legend(loc='best', fontsize=10)
plt.tight_layout()  # Automatically adjust subplots to give more padding

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

Key Enhancements:

- **Figure Size: The plt.figure(figsize=(8, 6)) command ensures the plot is larger, making it more readable.**
- **Trendline: A line of best fit (trendline) is added using NumPy's polyfit function to visualize the relationship between study hours and exam scores. The line equation is also displayed on the plot.**
- **Point Annotations: Each point is labeled with the corresponding exam score using the plt.annotate() function, making it easier to identify the exact values.**

- **Point Customization:** The size of each point is increased (s=100) for better visibility, and an edge color (edgecolor='k') is added to make the points stand out.
- **Font Customization:** Increased font sizes for labels and title for professional readability, and added bold font style for emphasis.
- **Gridlines:** A grid is added with both major and minor lines (which='both'), improving the chart's readability.
- **Tight Layout:** plt.tight_layout() is used to automatically adjust the padding and layout of the plot to avoid overlap.

Output:-



Ques 3. What is Sub-plot chart.

A subplot is a collection of multiple smaller plots displayed in a single figure. Subplots allow for the comparison of multiple datasets or views in one visual layout, making it easier to spot patterns across different categories or dimensions.

Key points about subplots:

Multiple Plots in One Figure: Subplots enable you to divide a single figure into a grid or arrangement of smaller plots, each of which can display different data or visualizations.

Grid Layout: Subplots are organized in a grid-like layout with rows and columns. You can specify the number of rows and columns to create a grid that suits your data visualization needs.

Shared or Independent Axes: Subplots can have shared or independent axes, meaning that multiple subplots within the same figure can either share the same axis scales or have their own separate scales.

Common Use Cases: Subplots are useful for various tasks, such as comparing multiple datasets, visualizing data from different categories or groups, or displaying the same data with different representations (e.g., line plot, bar plot, scatter plot).

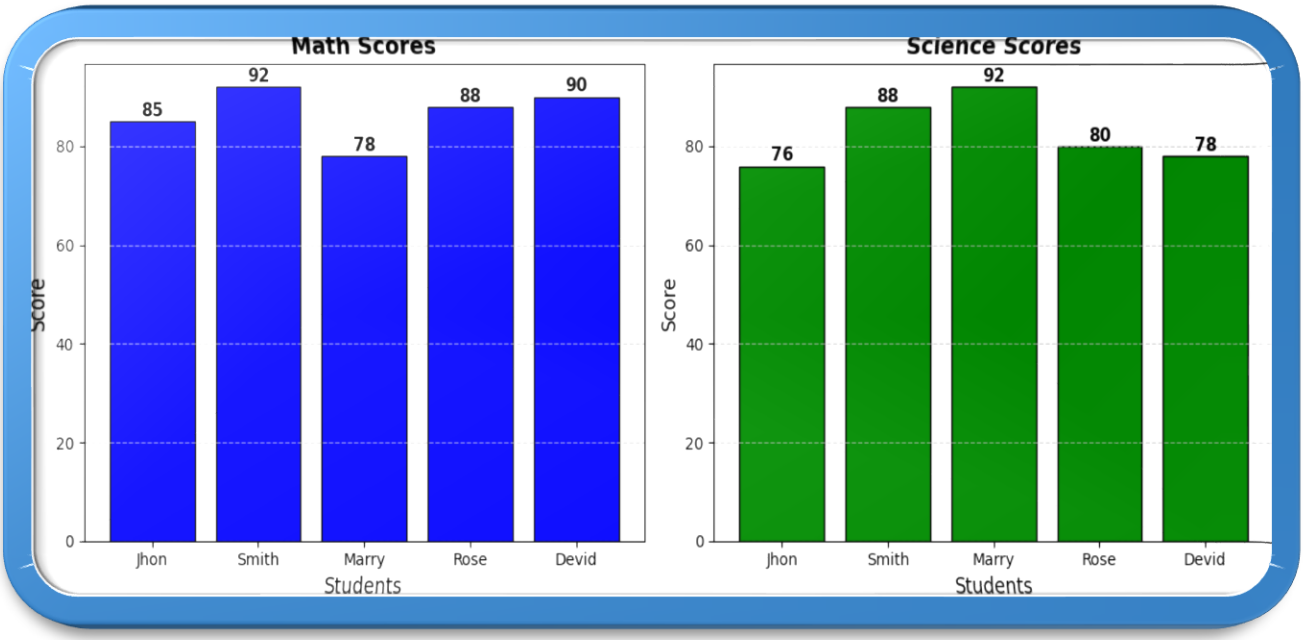
Let's take a program using axis:-

Program:-

lab23.py > ...

```python
import matplotlib.pyplot as plt

# Sample student data
students = ['Jhon', 'Smith', 'Marry', 'Rose', 'Devid']
math_scores = [85, 92, 78, 88, 90]
science_scores = [76, 88, 92, 80, 78]

# Create a subplot with two subplots (1 row, 2 columns)
fig, axs = plt.subplots(1, 2, figsize=(12, 5))

# Plot Math Scores
bars1 = axs[0].bar(students, math_scores, color='b', edgecolor='black')
axs[0].set_title('Math Scores', fontsize=14, fontweight='bold')
axs[0].set_xlabel('Students', fontsize=12)
axs[0].set_ylabel('Score', fontsize=12)
axs[0].grid(True, axis='y', linestyle='--', alpha=0.7)

# Add data labels on top of bars for Math Scores
for bar in bars1:
    yval = bar.get_height()
    axs[0].text(bar.get_x() + bar.get_width()/2, yval + 1, f'{yval}', ha='center', fontsize=11, fontweight='bold')

# Plot Science Scores
bars2 = axs[1].bar(students, science_scores, color='g', edgecolor='black')
axs[1].set_title('Science Scores', fontsize=14, fontweight='bold')
axs[1].set_xlabel('Students', fontsize=12)
axs[1].set_ylabel('Score', fontsize=12)
axs[1].grid(True, axis='y', linestyle='--', alpha=0.7)

# Add data labels on top of bars for Science Scores
for bar in bars2:
    yval = bar.get_height()
    axs[1].text(bar.get_x() + bar.get_width()/2, yval + 1, f'{yval}', ha='center', fontsize=11, fontweight='bold')

# Adjust spacing between subplots
plt.tight_layout()

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

Output:-


Math Scores
Score
Students
Jhon
Smith
Marry
Rose
Devid
85
92
78
88
90
0
20
40
60
80
Science Scores
Score
Students
Jhon
Smith
Marry
Rose
Devid
76
88
92
80
78
0
20
40
60
80