Ex No.10

Date:

Using Color Palettes and Customizing Visuals

Aim:

To explore different color palettes in Seaborn and customize visualizations for better readability and aesthetics.

Procedure:

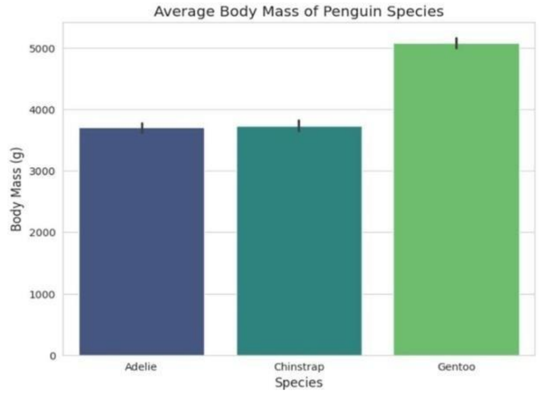
1. Install **Seaborn** and **Matplotlib** if not installed.
2. Import required libraries.
3. Load a sample dataset.
4. Apply different color palettes to Seaborn visualizations such as:
    - Bar plot with a **custom color palette**
    - Boxplot with a **setstyle and color palette**
    - Violinplot with **hue-based colors**
    - Customizing **figuresize, labels, and gridstyles**

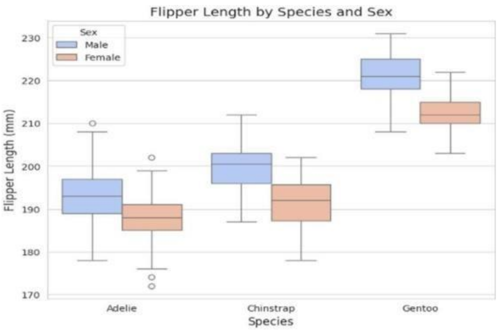
Code:

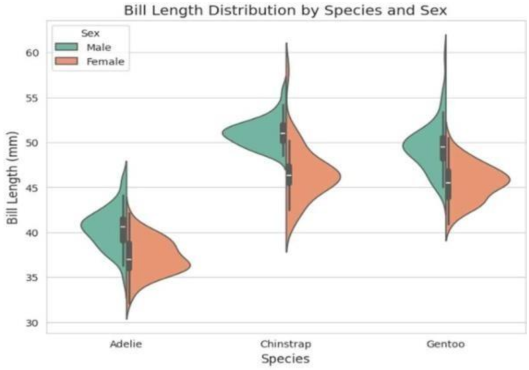
```
#Step1:Importrequiredlibraries
import seaborn as sns
importmatplotlib.pyplotasplt

#Step2:Loadasampledataset
df=sns.load_dataset("penguins")#Built-indatasetinSeaborn

# Step 3: Set a Seaborn style and color palette
sns.set_style("whitegrid")#Options:darkgrid,whitegrid,dark,white, ticks
sns.set_palette("pastel")#Options:deep,muted,bright,pastel,dark,  colorblind

#Step4:BarPlotwithCustomColors plt.figure(figsize=(8,
6))

sns.barplot(x="species",y="body_mass_g",data=df,palette="viridis")
plt.title("Average Body Mass of Penguin Species", fontsize=14)
plt.xlabel("Species", fontsize=12)

plt.ylabel("BodyMass(g)",fontsize=12)
plt.show()
```

```
#Step5:BoxPlotwithDifferentColorPalettes
plt.figure(figsize=(8, 6))
sns.boxplot(x="species",y="flipper_length_mm",data=df,
palette="coolwarm", hue="sex")

plt.title("FlipperLengthbySpeciesandSex",fontsize=14) plt.xlabel("Species",
fontsize=12)

plt.ylabel("FlipperLength(mm)",fontsize=12)
plt.legend(title="Sex")

plt.show()


#Step6:ViolinPlotwithHue-basedColoring  plt.figure(figsize=(8,  6))

sns.violinplot(x="species",y="bill_length_mm",data=df,palette="Set2",
hue="sex", split=True)

plt.title("BillLengthDistributionbySpeciesandSex",fontsize=14)  plt.xlabel("Species",
fontsize=12)

plt.ylabel("BillLength(mm)",fontsize=12)
plt.legend(title="Sex")

plt.show()
```

Implementation:







Result:

Successfully applied various Seaborn color palettes and customized visualization aesthetics using styles, figure sizes, and legend placements.

Ex No. 11

Date:

Plotting Univariate, Bivariate, Hexbin and Scatter plot

Aim:
To visualize data distribution and relationships using Univariate, Bivariate, Hexbin, and Scatter plots with the Seaborn library in Python.

Procedure:

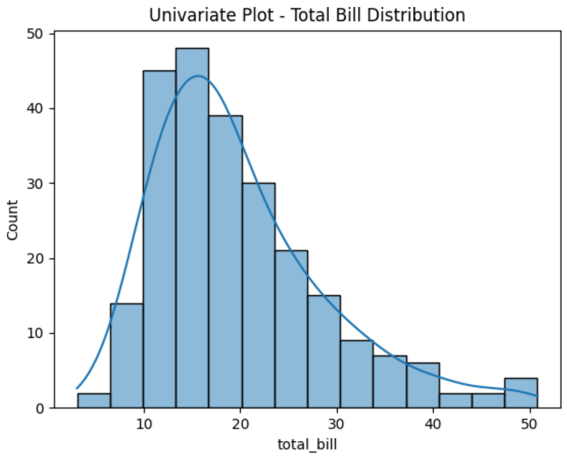
1. **Start the program.**
2. **Import required libraries**:
   - Import `seaborn` for plotting,
   - `matplotlib.pyplot` for displaying graphs,
   - optionally `pandas` and `numpy` for handling data.
3. **Load Dataset**:
   - Use any Seaborn built-in dataset (example: `tips`, `iris`, etc.).
   - Store it in a DataFrame for analysis.
4. **Univariate Plot**:
   - Select a single numeric column (e.g. `total_bill`).
   - Use `sns.histplot()` or `sns.distplot()` to plot histogram/KDE.
   - Display the distribution of one variable.
5. **Bivariate Plot**:
   - Select two related columns (e.g. `size` vs `tip`).
   - Use `sns.lineplot()` to show the relationship between the two variables.
   - Helps to observe patterns or trends.
6. **Hexbin Plot**:
   - Select two continuous variables (e.g. `total_bill` vs `tip`).
   - Use `sns.jointplot(kind='hex')` to show data density.
   - This helps to visualize where most data points are concentrated.
7. **Scatter Plot**:
   - Again select two numeric columns (e.g. `total_bill` vs `tip`).
   - Use `sns.scatterplot()` with a hue parameter to classify by a category.
   - Helps in identifying correlation.
8. **Customize Plots**:
   - Add titles, axis labels, and legends for readability.
9. **Display Plots**:
   - Use `plt.show()` after each plot to display the output.
10. End the program.

Code:

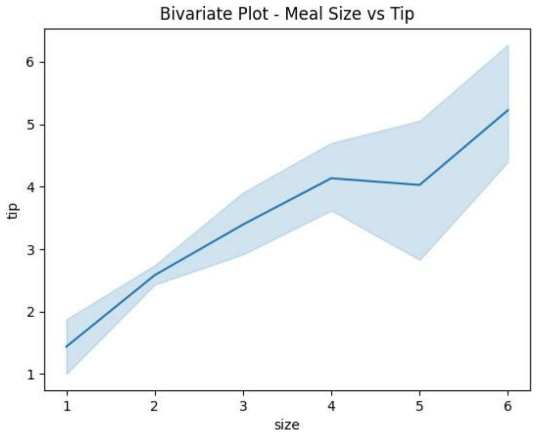
```
import seaborn as sns
import matplotlib.pyplot as plt

data = sns.load_dataset("tips")

plt.figure()
sns.histplot(data['total_bill'], kde=True)
plt.title("Univariate Plot - Total Bill Distribution")
plt.show()

plt.figure()
sns.lineplot(x='size', y='tip', data=data)
plt.title("Bivariate Plot - Meal Size vs Tip")
plt.show()

plt.figure()
sns.jointplot(x='total_bill', y='tip', data=data, kind='hex')
plt.show()

plt.figure()
sns.scatterplot(x='total_bill', y='tip', hue='sex', data=data)
plt.title("Scatter Plot - Total Bill vs Tip")
plt.show()
```

Implementation:

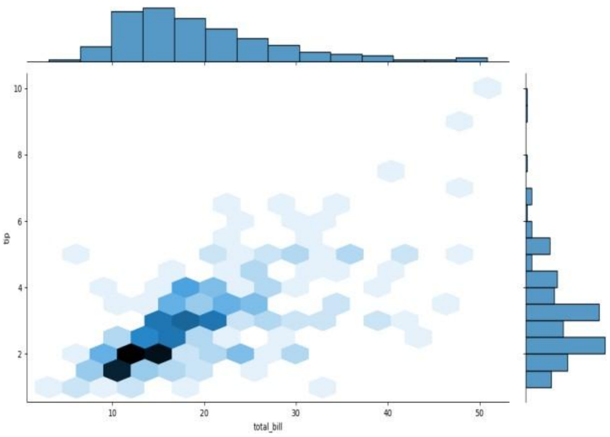
Histplot



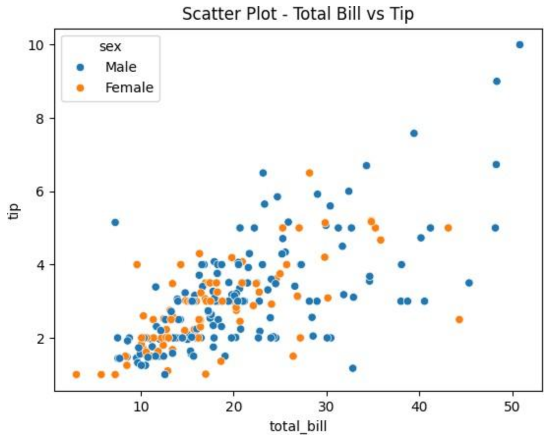
Bivariate plot



Hexbin Plot



Scatter Plot



Result:

Statistical plots like bar plot, count plot, and point plot were successfully created using Seaborn. The graphs showed mean values, data distribution, and category-wise comparisons.

Ex No. 12

Date:

Statistical Estimation using Seaborn

Aim:
To perform statistical estimation and visualize relationships between variables using Seaborn's estimation plots (like `barplot`, `countplot`, `pointplot`) in Python.

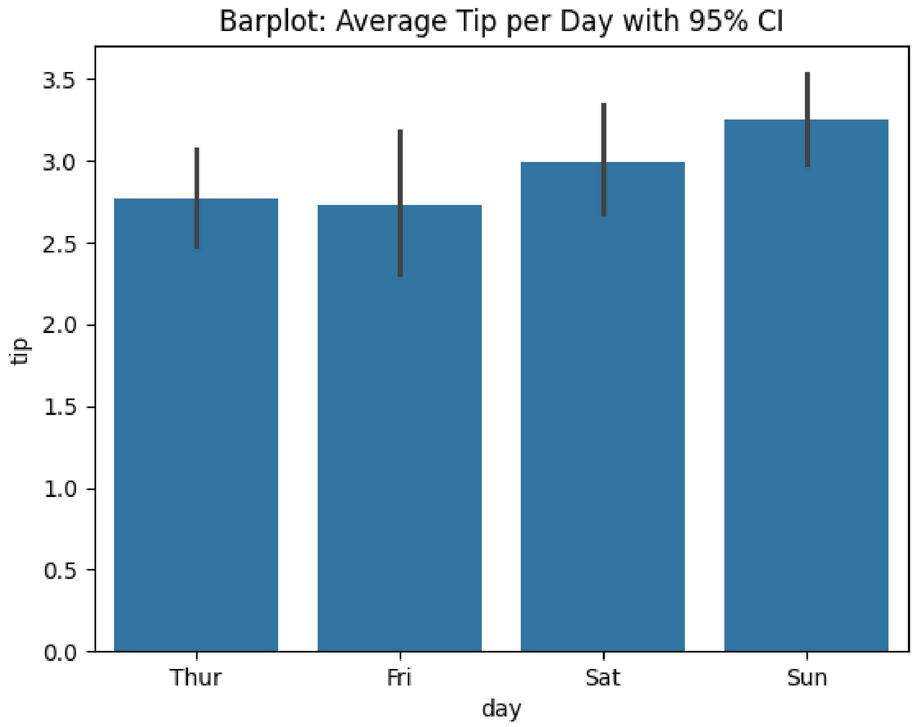
Procedure:

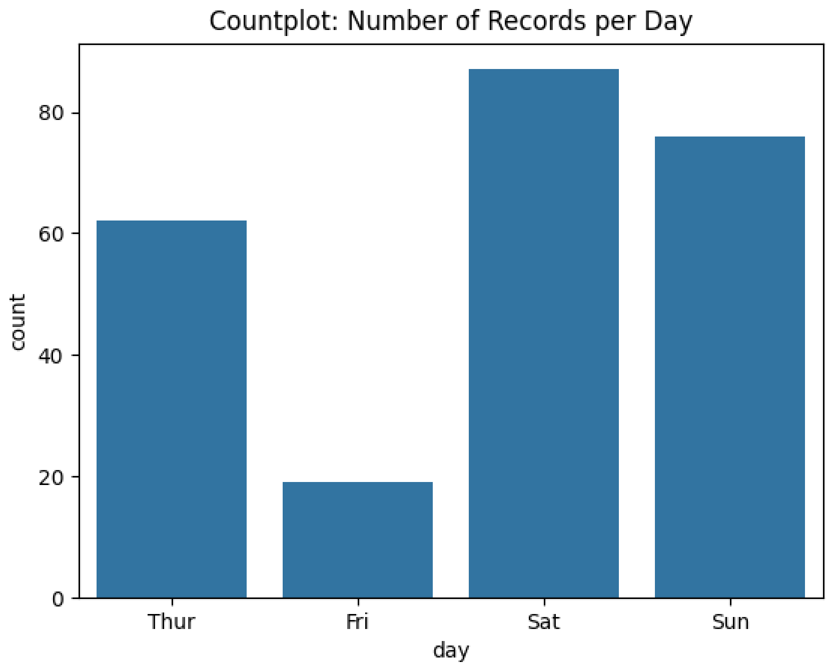
1. **Start the Program**
   Open Python IDE or Jupyter Notebook to begin writing the program.
2. **Import Libraries**
   - Import `seaborn as sns` for statistical plotting.
   - Import `matplotlib.pyplot as plt` for displaying the plots.
3. **Load Dataset**
   - Use a built-in Seaborn dataset like `tips` or `iris`.
   - Store it in a DataFrame for easy access and analysis.
4. **Bar Plot (Statistical Estimation)**
   - Choose a numeric variable (e.g., `tip`) and a categorical variable (e.g., `day`).
   - Use `sns.barplot()` to display the mean (or another estimator) of the numeric variable per category.
   - The `ci` parameter (confidence interval) shows the reliability of the estimate.
5. **Count Plot**
   - Use `sns.countplot()` to display the count of records for each category of a categorical variable.
   - Useful to understand the distribution of categorical data.
6. **Point Plot**
   - Shows point estimates (like mean) for a numeric variable grouped by a categorical variable.
   - Optionally use `hue` to add a second categorical variable to compare groups.
   - Confidence intervals can also be displayed.
7. **Customize Plots**
   - Add titles using `plt.title()`.
   - Label axes using `plt.xlabel()` and `plt.ylabel()`.
   - Adjust colors and style.
   - Set confidence intervals for better readability.
8. **Display Plots**
   - Use `plt.show()` to render each plot.
9. **End Program**
   - Analyze the visualizations to observe statistical patterns and insights.

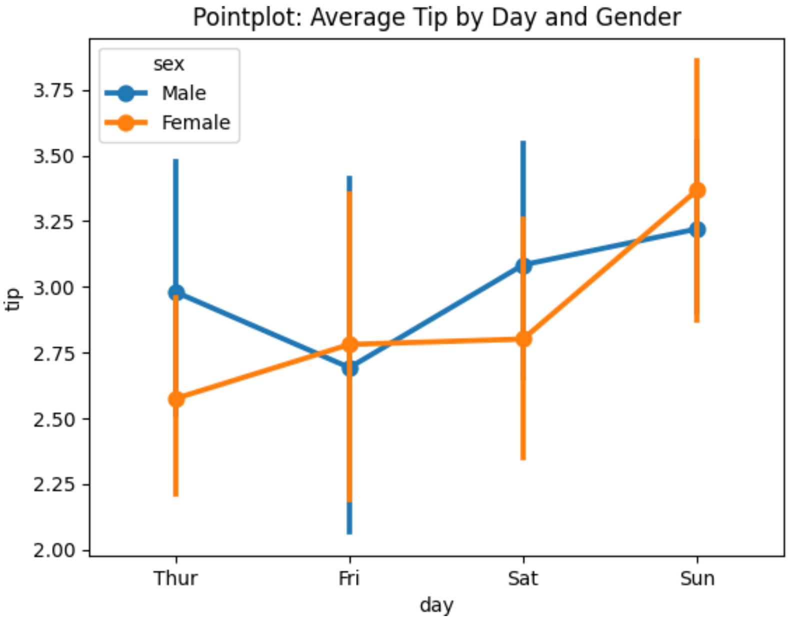
Code:

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

data = sns.load_dataset("tips")

plt.figure()
sns.barplot(x="day", y="tip", data=data, ci=95)  # ci=confidence interval
plt.title("Barplot: Average Tip per Day with 95% CI")
plt.show()

plt.figure()
sns.countplot(x="day", data=data)
plt.title("Countplot: Number of Records per Day")
plt.show()

plt.figure()
sns.pointplot(x="day", y="tip", hue="sex", data=data, ci=95)
plt.title("Pointplot: Average Tip by Day and Gender")
plt.show()
```

Implementation:







Result:

Statistical estimation plots were successfully created using Seaborn.

Ex No.13

Date:

Creating Different Types of Plots Using Matplotlib

Aim:

To Create Different types of plots using Matplotlib.

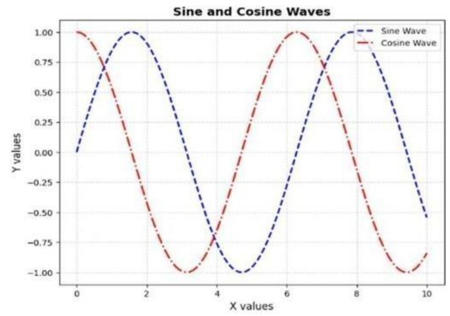
Procedure:

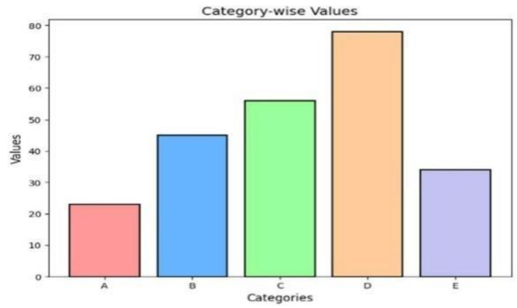
1. Install **Matplotlib** if not installed.
2. Import required libraries.
3. Load a dataset or generate sampledata.
4. Create different **visualizations**:
   - **LinePlot**
   - **BarChart**
   - **ScatterPlot**
   - **PieChart**

Code:

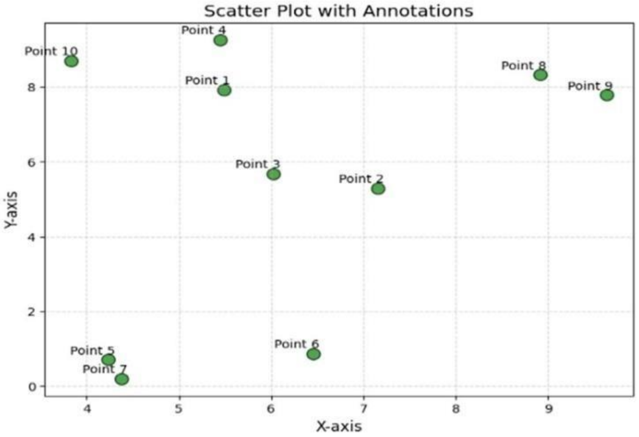
```
#Step1:Import required libraries import
matplotlib.pyplot as plt import
numpy as np

#Step2:Generatesampledata x = np.linspace(0, 10,100)

y1=np.sin(x)

y2=np.cos(x)


#Step3:CreateaLinePlot

plt.figure(figsize=(8, 6))

plt.plot(x,y1,label="SineWave",color="blue",linestyle="--", linewidth=2)
plt.plot(x,y2,label="CosineWave",color="red",linestyle="-.",  linewidth=2)

plt.title("SineandCosineWaves",fontsize=14,fontweight='bold') plt.xlabel("X
values", fontsize=12)

plt.ylabel("Y values", fontsize=12)
plt.legend(loc="upper right")
plt.grid(True,linestyle="--",alpha=0.5)
plt.show()

#Step4:CreateaBarChart
categories = ["A", "B", "C", "D", "E"]
values=[23,45,56,78,34]
colors=['#ff9999','#66b3ff','#99ff99','#ffcc99','#c2c2f0']
plt.figure(figsize=(8,6))
plt.bar(categories,values,color=colors,edgecolor="black",linewidth=1.5)
plt.title("Category-wiseValues",fontsize=14)
plt.xlabel("Categories", fontsize=12) plt.ylabel("Values",
fontsize=12)
plt.show()

#Step5:CreateaScatterPlot
np.random.seed(0)
x=np.random.rand(10)*10 y=np.random.rand(10)*10
labels=["Point"+str(i)foriinrange(1,11)]  plt.figure(figsize=(8,6))
plt.scatter(x,y,c="green",marker="o",s=100,edgecolor="black", alpha=0.7)
plt.title("ScatterPlotwithAnnotations",fontsize=14)
plt.xlabel("X-axis", fontsize=12)
plt.ylabel("Y-axis",fontsize=12)
fori,txtinenumerate(labels): plt.annotate(txt,(x[i],y[i]),textcoords="offsetpoints",
xytext=(5,5),
ha='right',  fontsize=10,  color="black")

plt.grid(True,linestyle="--",alpha=0.5)  plt.show()
#Step6:CreateaPieChart
labels = ['Apple', 'Banana', 'Grapes', 'Orange', 'Mango'] sizes=[25,15,30,10,20]
explode=(0,0,0.1,0,0)#Slightlyseparatethethirdslice

plt.figure(figsize=(7,7))
plt.pie(sizes,labels=labels,autopct='%1.1f%%',colors=colors, explode=explode,
startangle=140, shadow=True)
plt.title("FruitDistribution",fontsize=14) plt.show()
```

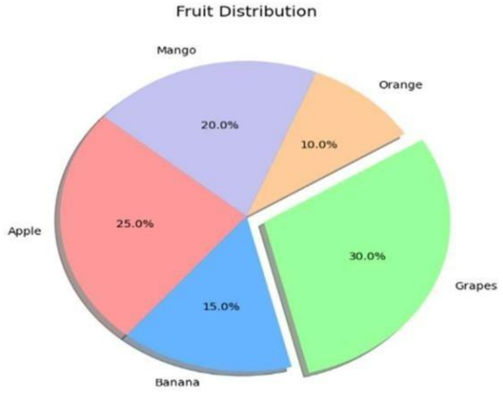
Implementation:





Result:





Result:

Successfully customized Matplotlib visualizations using labels, legends, colors, grid styling, and annotations for enhan`ced presentation.

Ex No. 14

Date:

Plotting Frequency Polygon for Temperature Series

Aim:
To plot the frequency polygon for the minimum and maximum temperature series data using a histogram to visualize the distribution pattern.

Procedure:

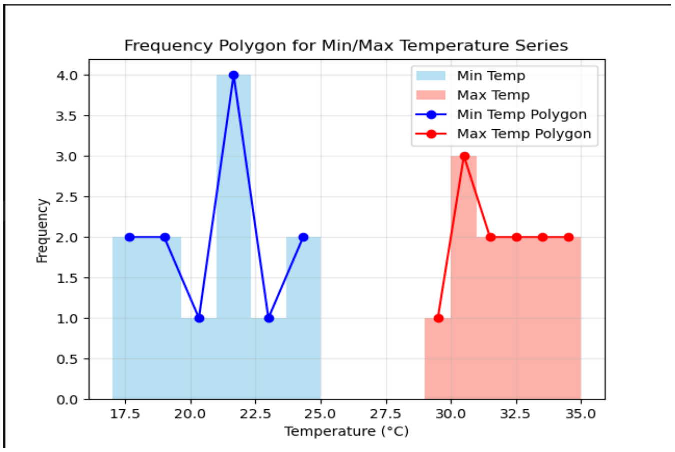
1. Start the program. Open Python or Jupyter Notebook.
2. Import required libraries. Import matplotlib.pyplot and pandas.
3. Load the dataset. Use a CSV or dictionary to store the minimum and maximum temperature values.
4. Plot Histogram. Use plt.hist() to create histograms for both min and max temperatures.
5. Plot Frequency Polygon. Overlay a frequency polygon using plt.plot() with the bin centers.
6. Customize the Plot. Add titles, legends, and axis labels.
7. Display the Plot. Use plt.show() to visualize the histogram and polygon.

Code:

```
import pandas as pd
import matplotlib.pyplot as plt

# Sample Temperature Data (can be replaced with actual dataset)
data = {
'MinTemp': [21, 19, 22, 18, 17, 20, 23, 25, 24, 19, 21, 22],
'MaxTemp': [30, 32, 33, 31, 29, 30, 34, 35, 33, 32, 31, 30]
}
df = pd.DataFrame(data)

# Plot Histogram for Min and Max Temperature
counts_min, bins_min, _ = plt.hist(df['MinTemp'], bins=6, color='skyblue', alpha=0.6, label='Min
Temp')
counts_max, bins_max, _ = plt.hist(df['MaxTemp'], bins=6, color='salmon', alpha=0.6, label='Max
Temp')

# Plot Frequency Polygon
plt.plot(bins_min[:-1] + (bins_min[1] - bins_min[0]) / 2, counts_min, color='blue', marker='o',
label='Min Temp Polygon')
plt.plot(bins_max[:-1] + (bins_max[1] - bins_max[0]) / 2, counts_max, color='red', marker='o',
label='Max Temp Polygon')
```

```
# Customize the plot
plt.title("Frequency Polygon for Min/Max Temperature Series")
plt.xlabel("Temperature (°C)")
plt.ylabel("Frequency")
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```

Implementation:



Result:

Frequency polygon for the minimum and maximum temperature series data was plotted Successfully.

Ex No. 15

Date:

Assessing the Students Performance with Python

Aim:

To assess students’ performance in multiple subjects using boxplots and to perform a comparative analysis of marks distribution across subjects.

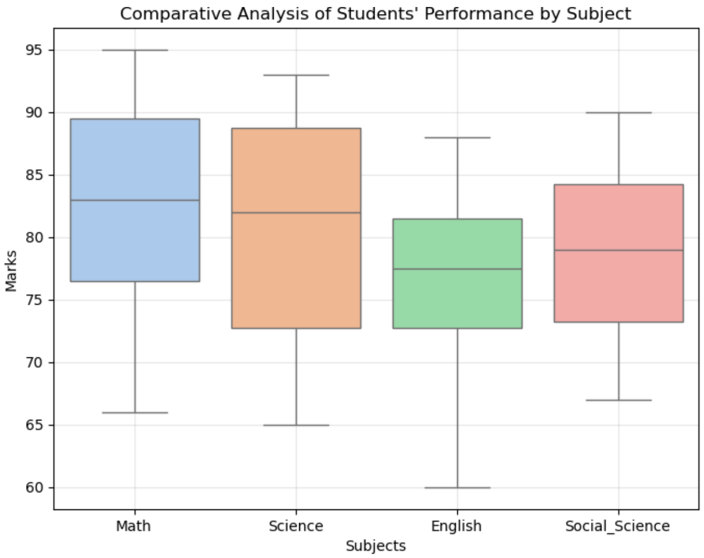
Procedure:

1. Start the Program. Open Python or Jupyter Notebook.
2. Import Required Libraries. Import pandas, seaborn, and matplotlib.pyplot.
3. Create Dataset. Enter marks of students in subjects like Math, Science, English, and Social Science.
4. Plot Boxplots. Use sns.boxplot() to visualize the marks distribution in each subject.
5. Analyze Distribution. Observe median, quartiles, and outliers to assess performance spread.
6. Customize and Display Plot. Add title, axis labels, and grid for clarity.

Code:

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

# Sample Data of students' marks
data = {
'Math': [78, 85, 90, 66, 70, 92, 88, 76, 81, 95],
'Science': [72, 88, 80, 65, 75, 84, 91, 70, 89, 93],
'English': [68, 75, 82, 60, 77, 85, 80, 72, 78, 88],
'Social_Science': [70, 78, 85, 67, 74, 80, 88, 73, 82, 90]
}

df = pd.DataFrame(data)

# Plot Boxplot
plt.figure(figsize=(8, 6))
sns.boxplot(data=df, palette="pastel")
plt.title("Comparative Analysis of Students' Performance by Subject")
plt.ylabel("Marks")
plt.xlabel("Subjects")
plt.grid(alpha=0.3)
plt.show()
```

Implementation:



Result:

A comparative analysis of marks distribution across subjects was performed successfully using boxplots.