Python Visualization Library Documentation

Nowadays, both tech and non-tech individuals prefer data presented in visual formats, as it is easier to understand than reading lengthy textIn Python, data visualization is the process of creating graphical representations of data to help people understand patterns, trends, relationships, and outliers that might be difficult to see in raw, textual data.

Python has a rich ecosystem of libraries specifically designed for data visualization. These libraries provide functions and tools to transform your data into various types of plots.

Key Libraries for Visualization in Python:

- Matplotlib
- Seaborn
- Plotly
- Bokeh
- Pandas
- Altair.

Here, we need to focus more on **Matplotlib and Seaborn** for Python Visualization.

Matplotlib

Matplotlib is the fundamental and most widely used plotting library in Python. With Matplotlib, we can perform a wide range of visualization tasks, including:-

- Creating basic plots such as line, bar, pie, and scatter plots.
- Customizing plots with labels, titles, legends, and color schemes.
- Adjusting figure size, layout, and aspect ratios.
- Combining multiple plots into subplots for better data representation.

A Matplotlib plot primarily contains:-

1. **Figure:** The entire drawing canvas or window.
2. **Axes:** The actual region where data is plotted (a single graph within the Figure). A Figure can have multiple Axes.
3. **Axis:** The individual X and Y number lines (rulers) for a given Axes, with ticks and labels.
4. **Plot Elements (Artists):** The visual representations of your data *on the Axes*, such as lines, markers, bars, text (titles, labels), legends, grids, and colorbars.

For representing data in a visual format using Matplotlib, the Pyplot sublibrary is required, as it provides various types of plots.

To import **Pyplot** from **Matplotlib**, you use the following line in Python:

❖ Import matplotlib.pyplot as plt

We also import **NumPy** in **Matplotlib** to efficiently handle and plot numerical data using arrays and mathematical functions.

❖ Import numpy as np

💣 (So, why do we use plt and as np?

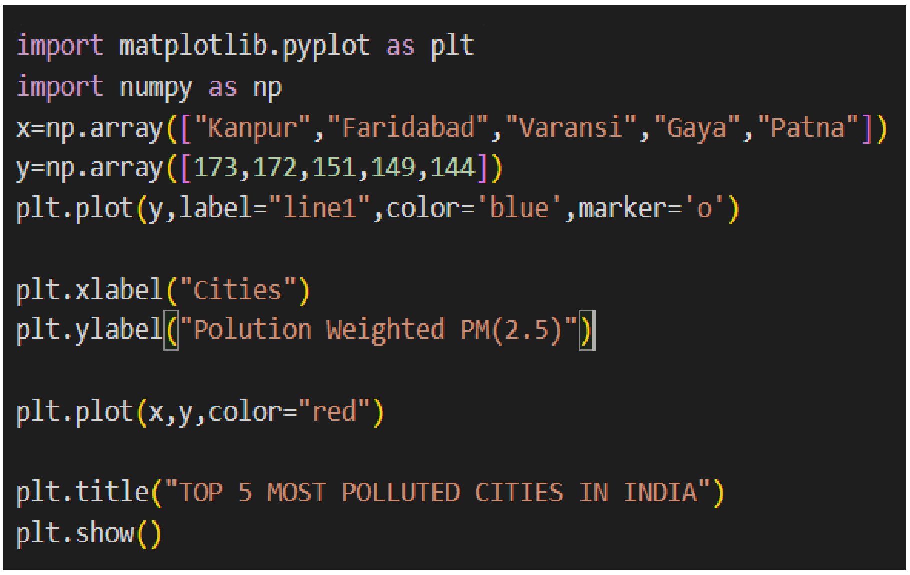
We use plt to avoid writing matplotlib.pyplot and np to avoid writing np repeatedly in each step. It simplifies the code by allowing us to use the shorter aliases plt and np whenever needed.)

Here I will try to explain some of the plots in matplotlib:

LINE CHART:-

Think of it like drawing dots on a piece of paper, and then connecting those dots with straight lines. You can use a line chart when you want to see a **trend** or how something **changes over time**.

Code snippet:-



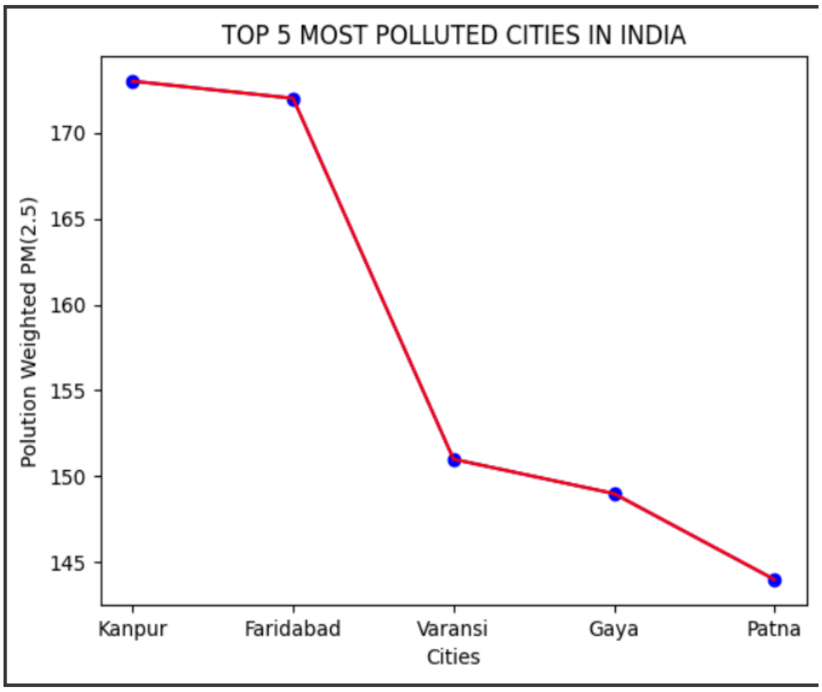
```python
import matplotlib.pyplot as plt
import numpy as np
x=np.array(["Kanpur","Faridabad","Varansi","Gaya","Patna"])
y=np.array([173,172,151,149,144])
plt.plot(y,label="line1",color='blue',marker='o')

plt.xlabel("Cities")
plt.ylabel("Polution Weighted PM(2.5)")

plt.plot(x,y,color="red")

plt.title("TOP 5 MOST POLLUTED CITIES IN INDIA")
plt.show()
```

Output:-



Description:-

x=np.array():-Creates a NumPy array 'x' containing strings representing the names of five Indian cities.

np.array():-Creates a NumPy array 'y' containing numerical data representing pollution weighted PM(2.5) values for the respective cities.

plt.plot():-Plots 'y' values on the y-axis , styles it blue with circle markers, and labels it "line1".

plt.xlabel():-Sets the label for the x-axis of the plot to "Cities".

plt.ylabel():-Sets the label for the y-axis of the plot to "Polution Weighted PM(2.5)".

Plots 'x' values on the x-axis and 'y' values on the y-axis, connecting them with a red line.

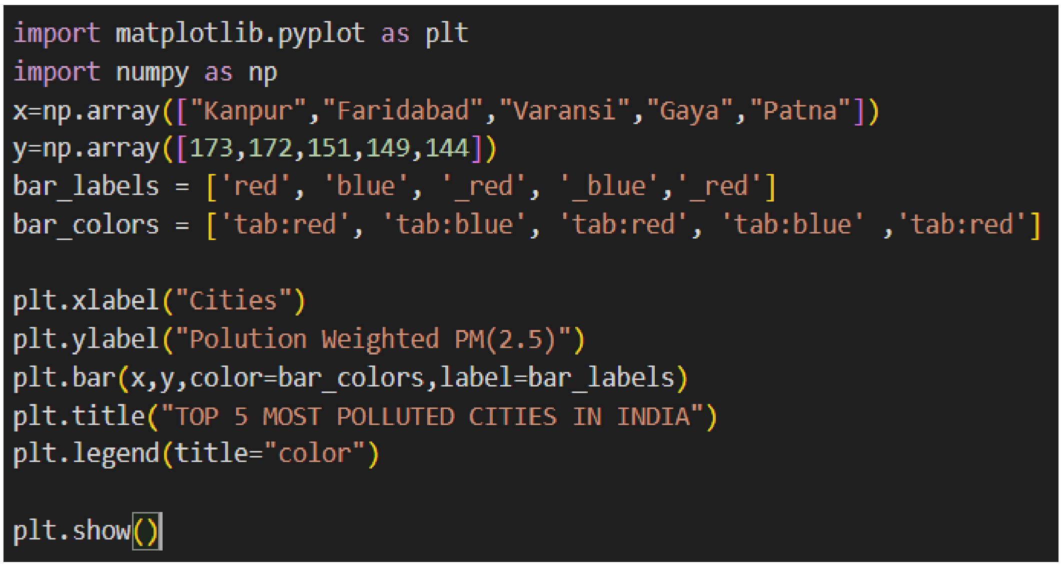
plt.title():- Sets the title of the plot to "TOP 5 MOST POLLUTED CITIES IN INDIA".

plt.show() :- Displays the generated plot.

BAR CHART:-

A **bar graph** is used to display **categorical data** with rectangular bars. Each bar's **height** shows the **value or frequency** of the category.

Code snippet:-

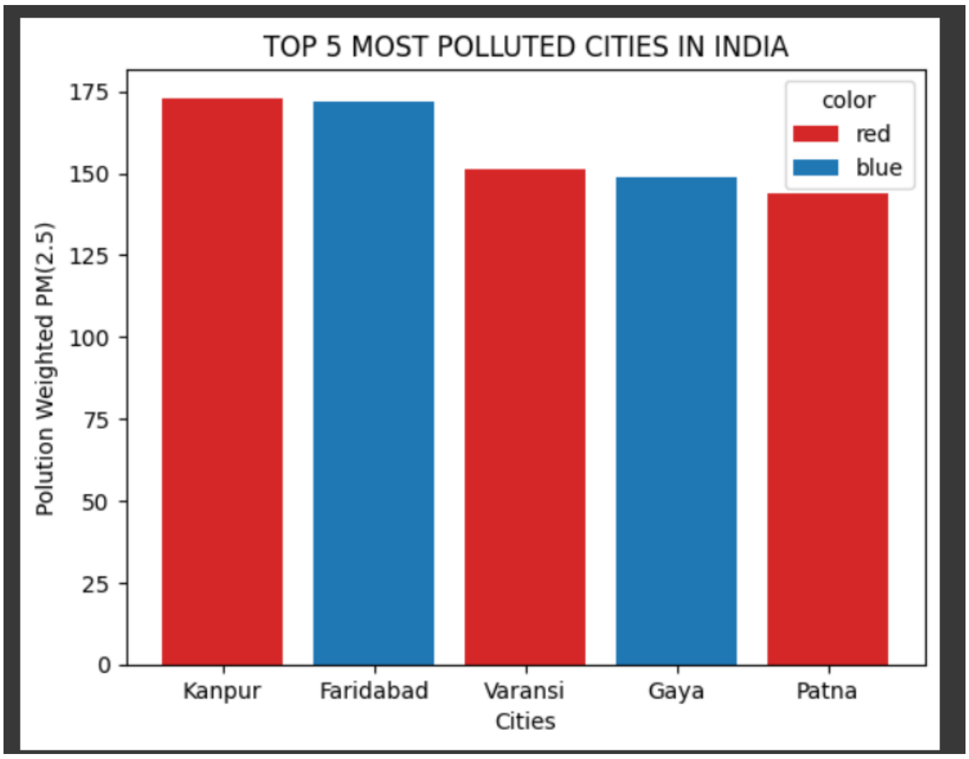


```
import matplotlib.pyplot as plt
import numpy as np
x=np.array(["Kanpur","Faridabad","Varansi","Gaya","Patna"])
y=np.array([173,172,151,149,144])
bar_labels = ['red', 'blue', '_red', '_blue','_red']
bar_colors = ['tab:red', 'tab:blue', 'tab:red', 'tab:blue' ,'tab:red']

plt.xlabel("Cities")
plt.ylabel("Polution Weighted PM(2.5)")
plt.bar(x,y,color=bar_colors,label=bar_labels)
plt.title("TOP 5 MOST POLLUTED CITIES IN INDIA")
plt.legend(title="color")

plt.show()
```

Output:



Description :-

(Here I use tab for rgb color)

plt.xlabel() **:-**Sets the label for the x-axis of the plot.

plt.ylabel() :- Sets the label for the y-axis of the plot.

plt.bar() :-Creates a vertical bar chart using given data and optional styling like color and label.

plt.title() :-Adds a title to the chart.

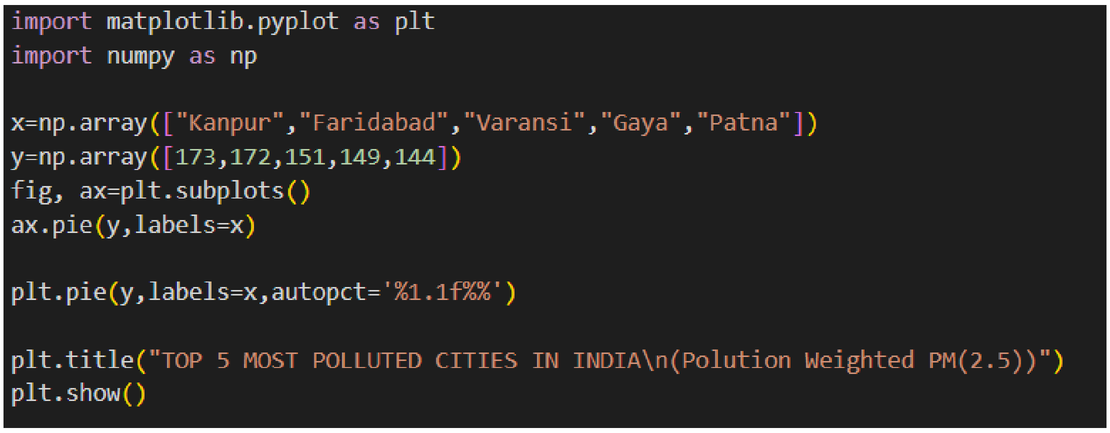
plt.legend() :- Displays a legend to explain the meaning of visual elements (like colors or labels).

plt.show() :-Renders and displays the final plot window.

PIE CHART:-

A **pie chart** shows **proportions or percentages** of different categories as slices of a circle. It's great for visualizing **parts of a whole**.

Code snippet:-



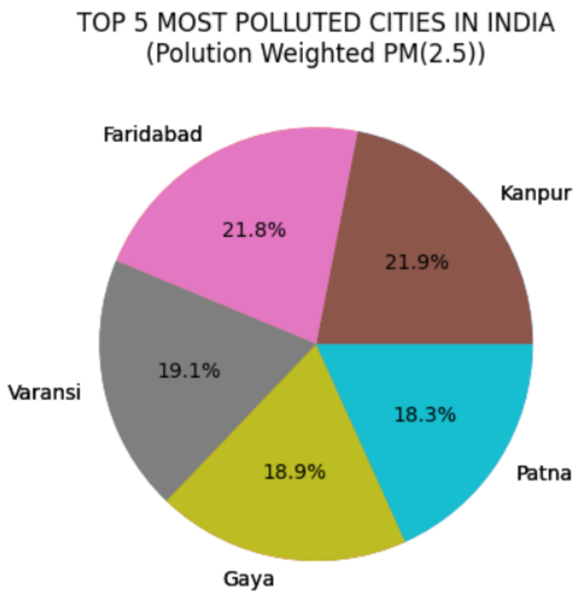
```
import matplotlib.pyplot as plt
import numpy as np

x=np.array(["Kanpur","Faridabad","Varansi","Gaya","Patna"])
y=np.array([173,172,151,149,144])
fig, ax=plt.subplots()
ax.pie(y,labels=x)

plt.pie(y,labels=x,autopct='%1.1f%%')

plt.title("TOP 5 MOST POLLUTED CITIES IN INDIA\n(Polution Weighted PM(2.5))")
plt.show()
```

Output:-



Description:-

plt.subplots():- Creates a figure and a set of subplots (axes) for plotting.

ax.pie():- Draws a pie chart on the specified axes.

plt.pie():-Draws a pie chart directly on the current figure.

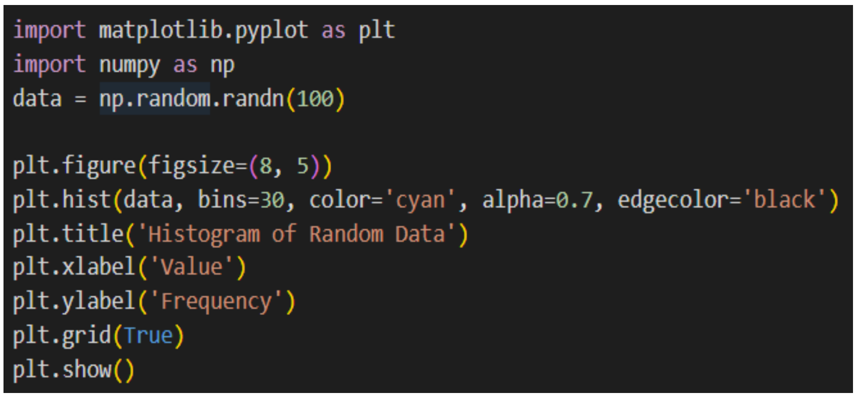
plt.title():-Adds a title to the entire plot.

plt.show():- Displays the plot window.

HISTOGRAM:-

A **histogram** shows the **distribution of numerical data** by dividing it into **equal intervals (bins)** and displaying how many values fall into each bin.

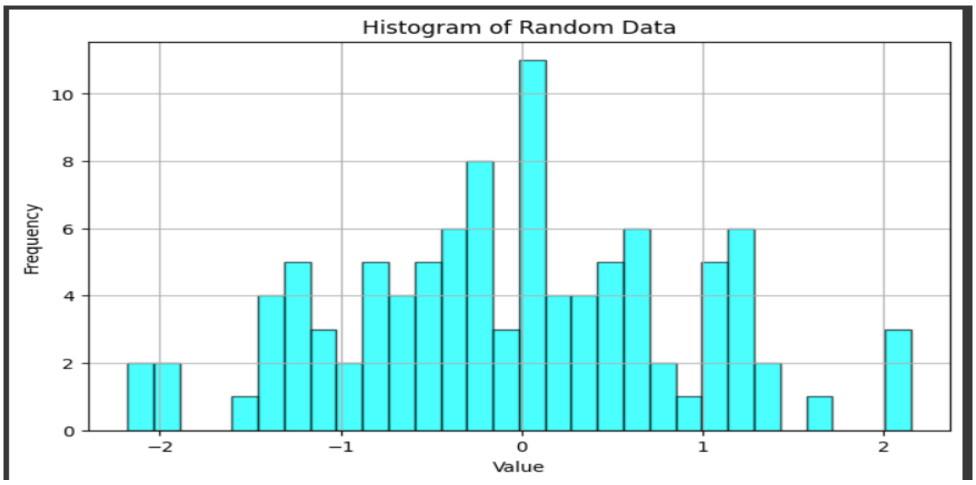
Code snippet:-



```
import matplotlib.pyplot as plt
import numpy as np
data = np.random.randn(100)

plt.figure(figsize=(8, 5))
plt.hist(data, bins=30, color='cyan', alpha=0.7, edgecolor='black')
plt.title('Histogram of Random Data')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

Output:-



Description:-

np.random.randn() :- Generates an array of random numbers from a standard normal distribution.

plt.figure() :- Creates a new figure for plotting with optional size settings.

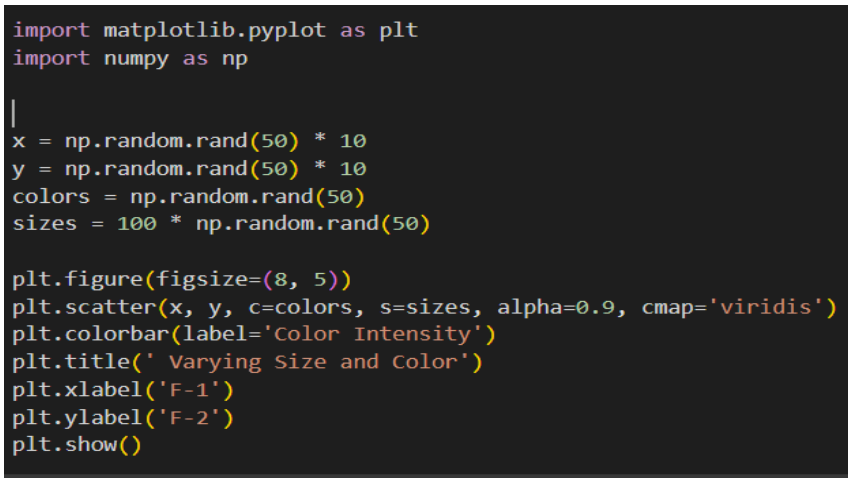
plt.hist() :-Plots a histogram to show the distribution of data.

plt.grid() :- Displays a grid on the plot.

SCATTER PLOT:-

A **scatter plot** shows the **relationship between two numerical variables** using dots. It's used to find trends, patterns, or correlations.

Code snippet:-

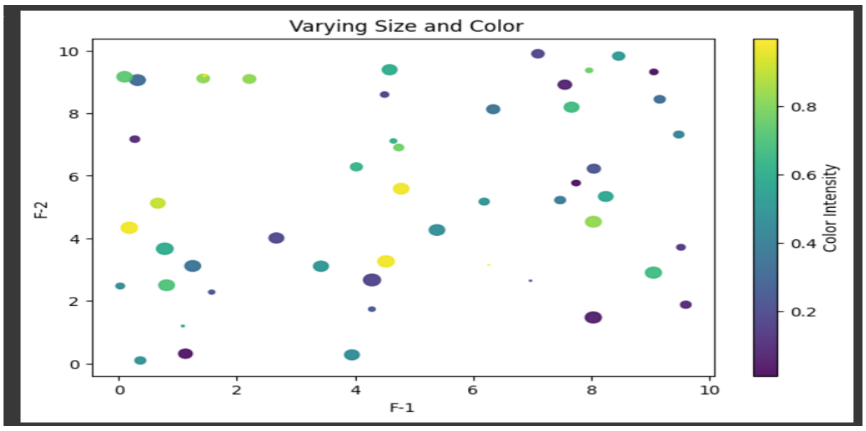


```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.rand(50) * 10
y = np.random.rand(50) * 10
colors = np.random.rand(50)
sizes = 100 * np.random.rand(50)

plt.figure(figsize=(8, 5))
plt.scatter(x, y, c=colors, s=sizes, alpha=0.9, cmap='viridis')
plt.colorbar(label='Color Intensity')
plt.title(' Varying Size and Color')
plt.xlabel('F-1')
plt.ylabel('F-2')
plt.show()
```

Output:-



Description:-

np.random.rand() :- Generates random numbers from a uniform distribution.

plt.figure() :- Initializes a new figure for plotting, optionally setting its size.

plt.scatter() :- Creates a scatter plot with optional color, size, and transparency.

plt.colorbar() :- Adds a color scale legend to the plot.

SEABORN:-

Seaborn is a **high-level data visualization library** built on Matplotlib. It's designed to make creating **attractive and informative statistical graphics** much easier and quicker.Seaborn is like having pre-mixed, beautiful paint colors and ready-made templates for common paintings.

To import **Seaborn**, just use this line at the top of your Python script :-

- import seaborn as sns

Now, we will see the list of different plots in each category:-

Categorical Plots

- barplot()
- countplot()
- boxplot()
- violinplot()
- stripplot()
- swarmplot()
- pointplot()

Distribution Plots

- histplot()
- kdeplot()
- ecdfplot()
- rugplot()

Relational Plots

- scatterplot()
- lineplot()
- relplot()

Regression Plots

- regplot()
- lmplot()

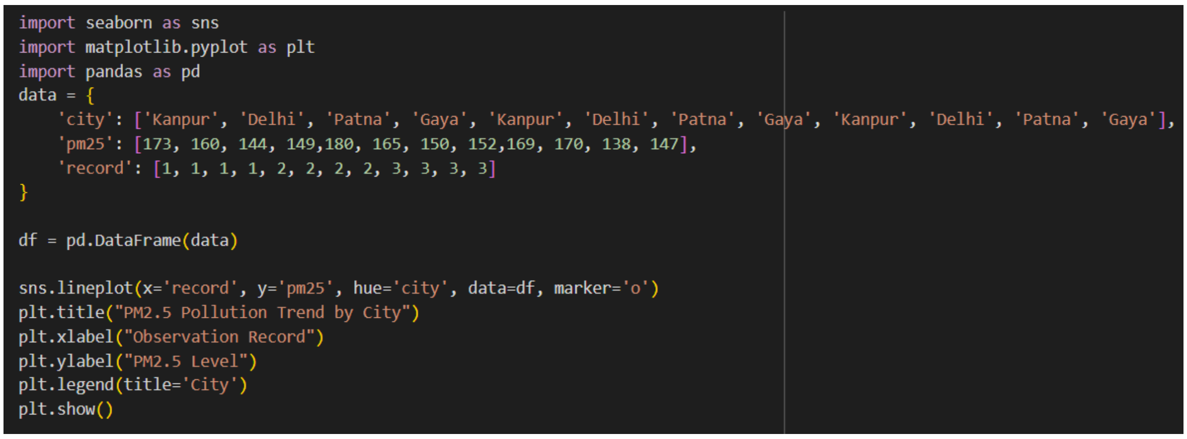
Matrix Plots

- heatmap()
- clustermap()

Here, I will provide some examples:-

LINEPLOT:-

Code:-

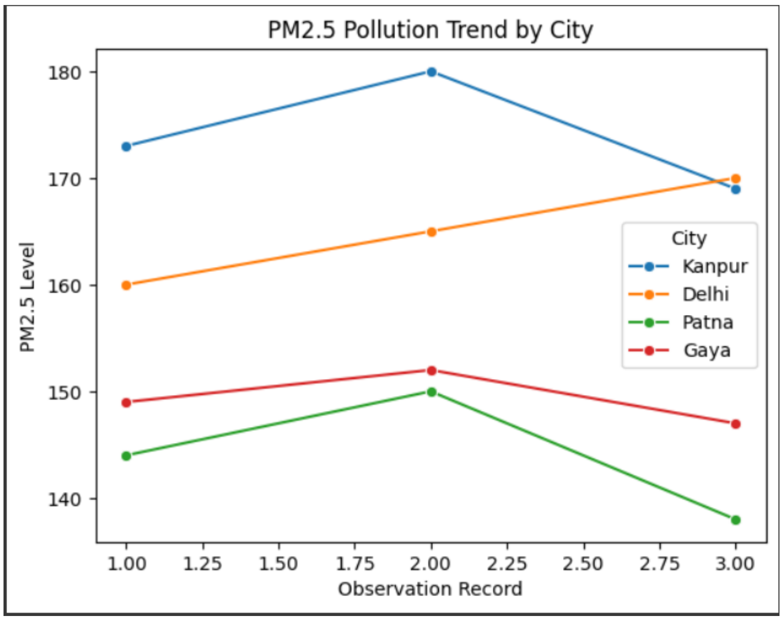


```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
data = {
    'city': ['Kanpur', 'Delhi', 'Patna', 'Gaya', 'Kanpur', 'Delhi', 'Patna', 'Gaya', 'Kanpur', 'Delhi', 'Patna', 'Gaya'],
    'pm25': [173, 160, 144, 149,180, 165, 150, 152,169, 170, 138, 147],
    'record': [1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3]
}

df = pd.DataFrame(data)

sns.lineplot(x='record', y='pm25', hue='city', data=df, marker='o')
plt.title("PM2.5 Pollution Trend by City")
plt.xlabel("Observation Record")
plt.ylabel("PM2.5 Level")
plt.legend(title='City')
plt.show()
```

Output:-



Description:-

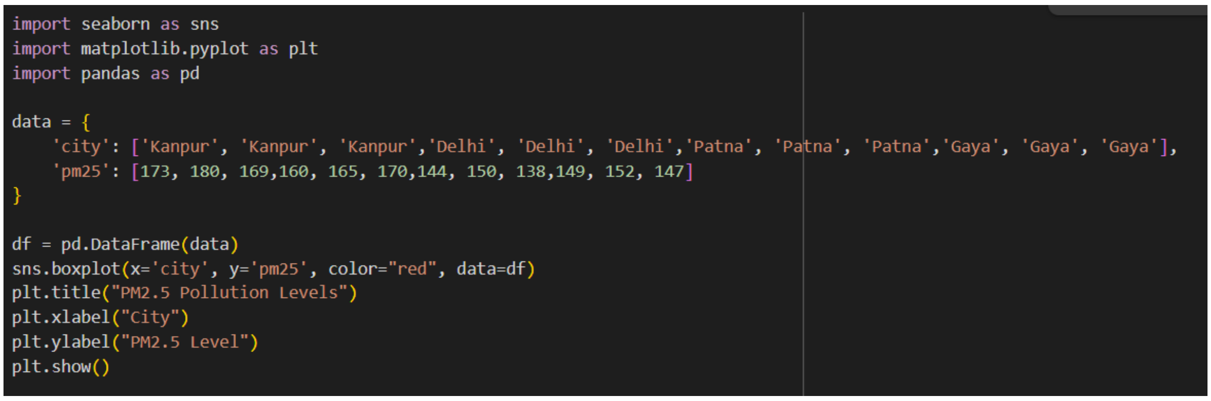
pd.DataFrame() :- Converts a dictionary into a DataFrame (tabular format).

sns.lineplot() :- Creates a line plot for visualizing trends over a continuous variable.

BOX PLOT:-

A **box plot in Seaborn** is used to visualize the **distribution, spread, and outliers** of data across different categories.

Code:-

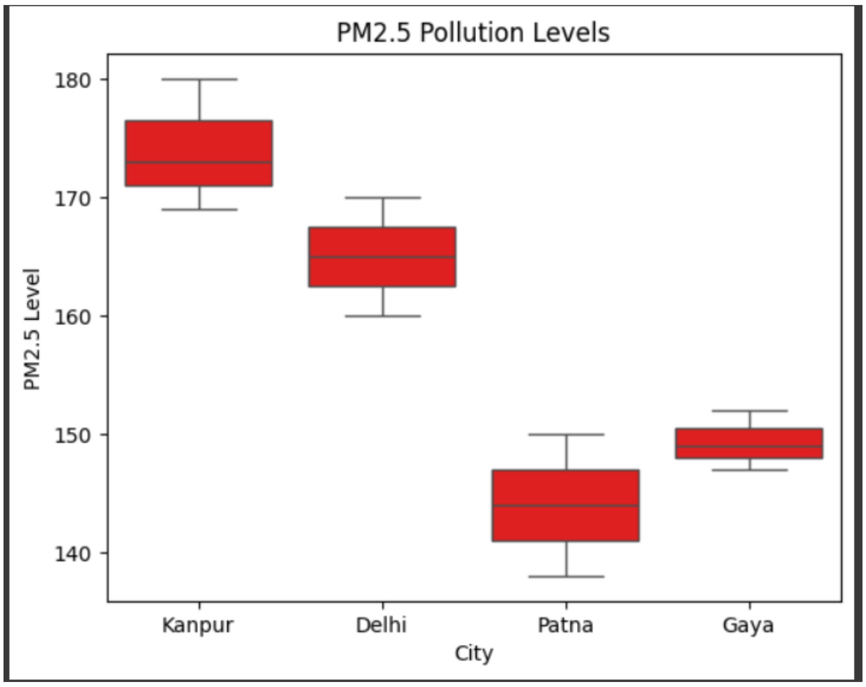


```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

data = {
    'city': ['Kanpur', 'Kanpur', 'Kanpur','Delhi', 'Delhi', 'Delhi','Patna', 'Patna', 'Patna','Gaya', 'Gaya', 'Gaya'],
    'pm25': [173, 180, 169,160, 165, 170,144, 150, 138,149, 152, 147]
}

df = pd.DataFrame(data)
sns.boxplot(x='city', y='pm25', color="red", data=df)
plt.title("PM2.5 Pollution Levels")
plt.xlabel("City")
plt.ylabel("PM2.5 Level")
plt.show()
```

Output:-

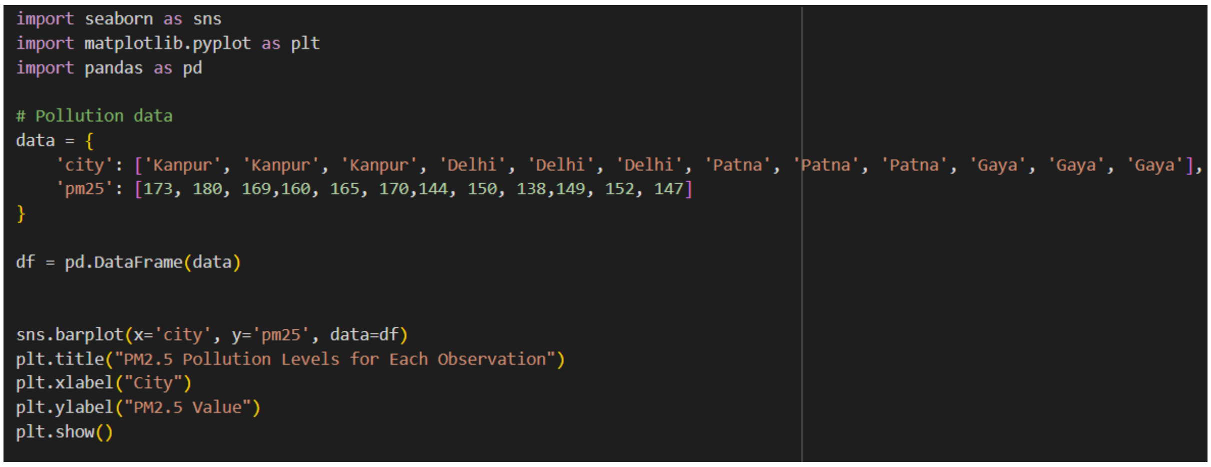


Description:-

sns.boxplot() :- Draws a box plot for visualizing the distribution and outliers

BAR PLOT:-

Code:-



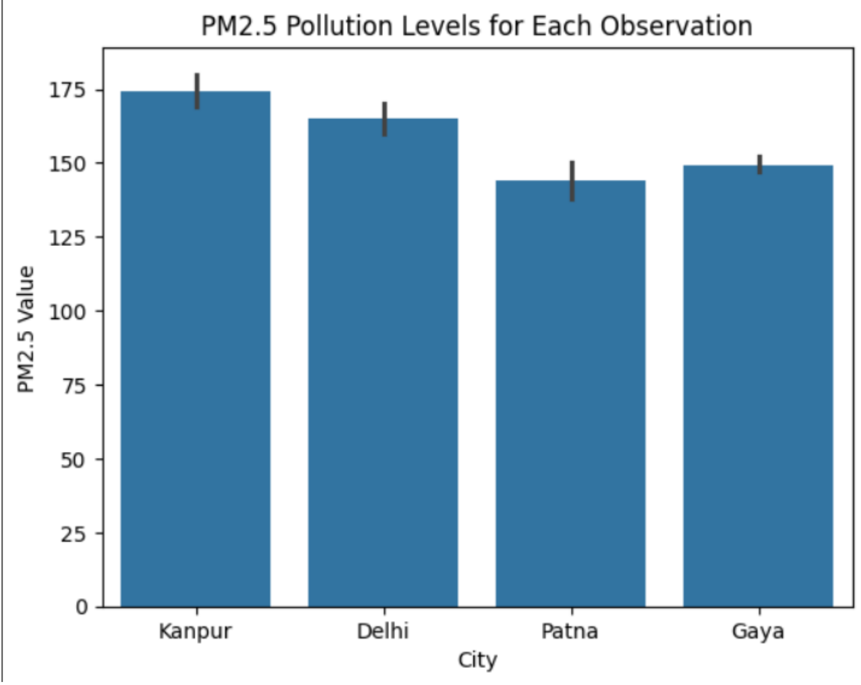
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Pollution data
data = {
    'city': ['Kanpur', 'Kanpur', 'Kanpur', 'Delhi', 'Delhi', 'Delhi', 'Patna', 'Patna', 'Patna', 'Gaya', 'Gaya', 'Gaya'],
    'pm25': [173, 180, 169,160, 165, 170,144, 150, 138,149, 152, 147]
}

df = pd.DataFrame(data)


sns.barplot(x='city', y='pm25', data=df)
plt.title("PM2.5 Pollution Levels for Each Observation")
plt.xlabel("City")
plt.ylabel("PM2.5 Value")
plt.show()
```

Output:-



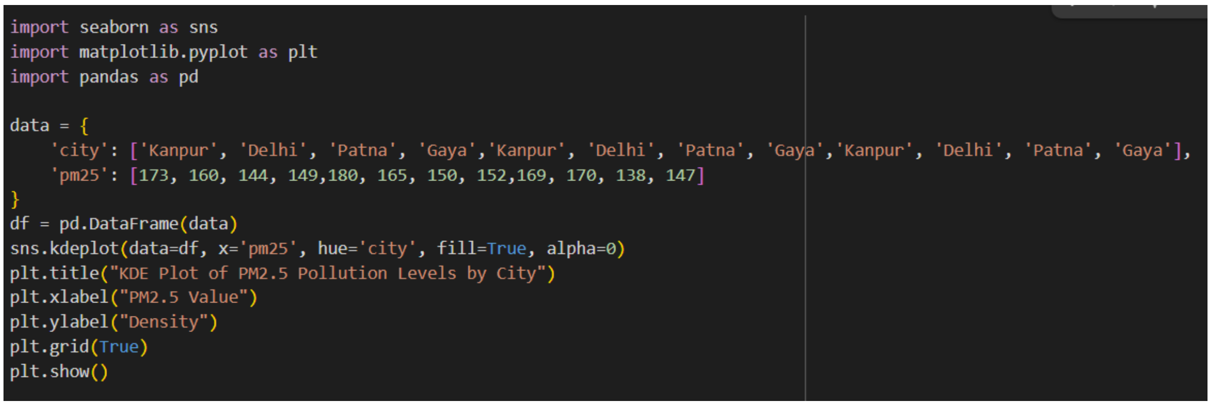
Description:-

sns.barplot() :- Creates a bar plot that shows the average of a numeric variable for each category.

KDE PLOT :-

Instead of using discrete bars like a histogram, a KDE plot uses a smooth, continuous curve to represent the distribution of the data.

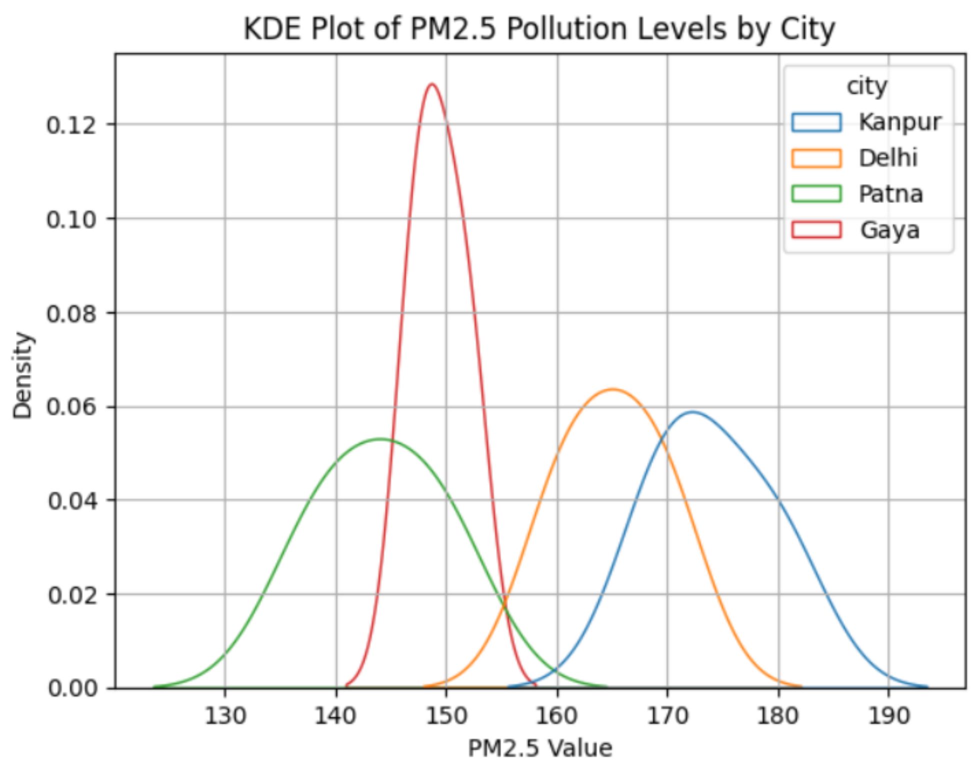
Code:-



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

data = {
    'city': ['Kanpur', 'Delhi', 'Patna', 'Gaya','Kanpur', 'Delhi', 'Patna', 'Gaya','Kanpur', 'Delhi', 'Patna', 'Gaya'],
    'pm25': [173, 160, 144, 149,180, 165, 150, 152,169, 170, 138, 147]
}
df = pd.DataFrame(data)
sns.kdeplot(data=df, x='pm25', hue='city', fill=True, alpha=0)
plt.title("KDE Plot of PM2.5 Pollution Levels by City")
plt.xlabel("PM2.5 Value")
plt.ylabel("Density")
plt.grid(True)
plt.show()
```

Output:-



Description:-

pd.DataFrame() :- Creates a DataFrame from a dictionary.

sns.kdeplot() :- Draws a Kernel Density Estimate plot to show the distribution of a numeric variable.

(fill=True) :- Fills the area under each KDE curve

Comparison Of Matplotlib And Seaborn:-

Matplotlib

- **Nature:** It's a low-level, comprehensive plotting library. Think of it as the fundamental building block for almost any kind of plot you can imagine. Seaborn is actually built on top of Matplotlib.
- **Control & Customization:** Offers fine-grained control over every single element of a plot. You can manually adjust colors, line styles, marker types, fonts, axes limits, tick locations, legends, annotations, and more. This makes it incredibly powerful for creating highly customized or publication-quality figures that meet specific design requirements.
- **Code Verbosity:** For many standard statistical plots, Matplotlib often requires more lines of code to achieve a polished look. You might need to set up the figure, axes, plot the data, add labels, and then adjust styles step-by-step.
- **Default Aesthetics:** Matplotlib's default plot aesthetics are generally functional but can appear basic or less visually appealing compared to Seaborn's. Achieving a "pretty" plot often requires manual customization.
- **Data Handling:** Primarily works with **NumPy arrays and Python lists**. While it can plot Pandas DataFrames, you often need to explicitly specify columns as x and y values.

Advantages of Matplotlib :-

- ❖ Fully customizable.
- ❖ Works for any type of plot.
- ❖ Great for fine-tuning visuals.
- ❖ More control for publications.

Disadvantages of Matplotlib :-

- Requires more code.
- Looks basic without styling.
- Not ideal for quick, statistical plots.

Seaborn

- **Nature:** It's a high-level, statistical data visualization library built as an abstraction layer on top of Matplotlib. Its primary goal is to make it easier to create informative and attractive statistical graphics.
- **Control & Customization:** Provides a higher-level interface, simplifying the process of creating complex statistical plots. While it offers good customization options (e.g., color palettes, themes, faceting), it doesn't offer the same granular control as raw Matplotlib.
- **Code Verbosity:** Requires significantly less code for common statistical visualizations. Many complex plots can be generated with a single function call.
- **Default Aesthetics:** Comes with beautiful, built-in themes and color palettes. Plots generated by Seaborn are generally aesthetically pleasing "out of the box," reducing the need for extensive manual styling.
- **Data Handling:** Designed to work seamlessly with Pandas DataFrames. You can often pass DataFrame column names directly to the plotting functions, which simplifies data mapping.

Advantages of Seaborn :-

- Beautiful plots by default.
- Easier for statistical visualization.
- Less code, more functionality.

- Works directly with Pandas DataFrames.

<u>Disadvantages of Seaborn :-</u>

- Less control for deep customization.
- Slower for large datasets.
- Depends on Matplotlib underneath.
- Limited to built-in plot types.