Data Visualization

Data visualization is the process of **representing data in a visual form** like charts, graphs, or maps instead of raw tables or numbers.

- It helps humans **see patterns, trends, and insights** quickly.
- Instead of reading hundreds of rows of numbers, you can understand the data **at a glance**.

Type	Example Use Case
Bar chart	Compare sales across months
Line chart	Show stock price trends over time
Scatter plot	Analyse relationship between two variables
Pie chart	Show percentage distribution of categories
Heatmap	Show correlation between variables

Advantages of Data Visualization:

1. Understand even complex data verry quickly.
2. Used to spot trends and patterns in data.
3. Better decision making – make informed decisions.
4. Handles big data well

Note:

1. In interviews instead of telling by word tell by visualizing.
2. Visualization can lie if it is not used properly. (If there is bad scaling and all)

Matplotlib

Matplotlib is a tool in Python that helps you draw graphs and charts to see and understand data better.

- It's hard to understand trends just by looking at numbers.
- But if you draw a chart, like a bar graph or line plot, it becomes easy to see patterns.

Seaborn

What is Seaborn?

- Seaborn is a Python data visualization library built on top of matplotlib.
- It is designed specifically for statistical and analytical plots.
- Works seamlessly with pandas DataFrames.
- Makes charts more aesthetic by default (better colors, styles, and themes).

2️⃣ Why use Seaborn instead of Matplotlib?

Feature	Matplotlib	Seaborn	Advantage
Syntax	More verbose, need more code	Simple, concise	Faster coding for common plots
Style	Basic, needs customization	Beautiful default themes	Charts look professional without extra work
Multiple lines / categories	Need to manually separate columns	Use `hue` or `style`	Easier to compare categories
Statistical plots	Limited (you have to calculate stats first)	Built-in statistical plotting	Can show confidence intervals, regression lines automatically
DataFrames	Works with arrays	Works directly with pandas DataFrames	Less preprocessing, directly pass DataFrame
Facets / subplots	Harder to manage multiple small charts	`FacetGrid` or `col=` makes it easy	Quick multi-chart visualization

Line chart

A **line chart** is a graph that displays data points connected by straight lines.

It's mainly used to:

- Show trends over time (time series data).
- Compare multiple series.
- Identify patterns, rises, or drops in data.

A line chart is used **when you want to show trends or changes over an ordered variable**, usually **time**.

Common Scenarios:

Situation	Example
Trend over time	Monthly sales, website visitors over a year, stock prices
Comparing multiple series	Sales of multiple products across months
Showing continuous data	Temperature changes during the day, heart rate over time
Highlighting growth or decline	Revenue growth, user engagement, stock performance

Key hint:
If your **x-axis represents an ordered variable** (usually time, but can be a sequence like days, age, or experiment steps), a line chart is usually appropriate.

2. How to Know You Should Use a Line Chart

Ask yourself these questions:

1. **Is my data continuous or sequential?**
   - Example: Daily temperature → Yes, use line chart.
   - Example: Categories like "Fruits: Apple, Banana, Mango" → No, better use bar chart.
2. **Do I want to show trends over time or progression?**
   - Example: Monthly sales trend → Line chart.
   - Example: Total sales per region → Bar chart (categorical).
3. **Do I want to compare multiple series over the same sequence?**

- Example: Sales of three products over 6 months → Line chart with multiple lines.

4. **Do I want to highlight peaks, drops, or patterns?**

- Line charts make it easy to see **highs, lows, or consistent growth**.

Examples of Wrong vs Right Usage

Wrong: Showing sales by product category with a line chart.

- Reason: Products are discrete categories, not a sequence → Use a bar chart.

Right: Showing sales for the same product every month → Line chart.

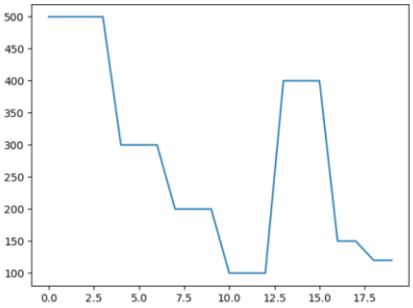
Right: Comparing website visitors across multiple months for multiple websites → Line chart with multiple lines.

---

1. **Simple line Chart Creation:**

- Here we have to give column name inside bracket from the data frame

```
plt.plot(df["Price"])
plt.show()
```

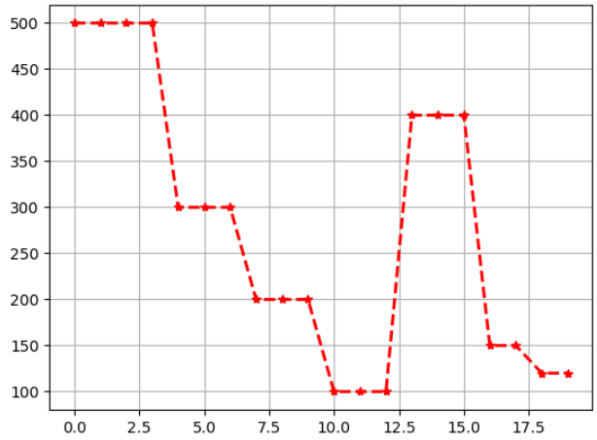


This is how Price column is varying

---

2. **Plot With marker and colour**

```
plt.plot(df["Price"],color='red',marker="*",linestyle="--",linewidth="2")
plt.grid()
plt.show()
```



- **color** makes the line color in line chart to specified one.
- **marker** makes the marks the point wherever data point is there
- **linestyle** is used to change the style of the line
- **linewidth** is used to se the width of the line
- **plt.grid()** is used to show grid line

Example Problems:

	Month	Visitors	Signups	Revenue
0	January	1200	50	2000
1	February	1500	65	2500
2	March	1700	80	3000
3	April	1600	70	2800
4	May	1800	90	3500
5	June	2000	100	4000
6	July	2200	120	4500
7	August	2100	110	4300
8	September	2300	130	4800
9	October	2500	150	5200
10	November	2400	140	5000
11	December	2600	160	5500

Problem 1:

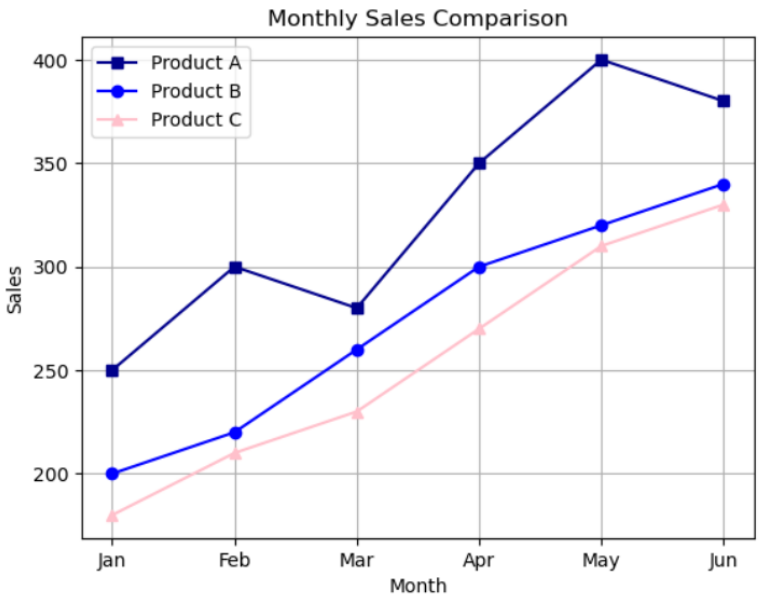
➔ **Plot a line chart for Month vs Visitors.**

➔ **Identify the months where visitors decreased compared to the previous month.**

```python
import matplotlib.pyplot as plt

plt.plot(df["Month"], df["Visitors"], color='darkblue', marker="*", linewidth=2)
plt.grid(True)
plt.title("Monthly Visitors")
plt.xlabel("Months")
plt.ylabel("Visitors")
plt.xticks(rotation=45)  # optional for better readability
plt.show()
```

For Multiple Lines:

```python
plt.plot(df["Month"],df["Product_A"],color="darkblue",label='Product A',marker ='s')
plt.plot(df["Month"],df["Product_B"],color="blue",label='Product B',marker = 'o')
plt.plot(df["Month"],df["Product_C"],color="pink",label='Product C',marker = '^')

plt.title('Monthly Sales Comparison')
plt.xlabel('Month')
plt.ylabel('Sales')
plt.legend()
plt.grid()
plt.show()
```



- Marker = ‘s’ → square
- Marker = ‘o’ → circle
- Marker = ‘^’ → Triangle
- Apart from that you can use * etc also.

5. Common Tips / Edge Cases

1. If x-axis is **dates**, convert to `datetime` for proper formatting.
2. Missing values (`NaN`) are ignored in Matplotlib; Pandas can interpolate.
3. For many series, use different colors and markers for clarity.
4. Always label axes and add a legend if plotting multiple lines.
5. For large datasets, consider reducing markers or using transparency (`alpha`) to avoid clutter.

Some Scenarios

1️⃣ Case 1: You give only y-values

```python
import matplotlib.pyplot as plt

y = [2, 4, 6, 8, 10]
plt.plot(y)
plt.show()
```

👉 Here, matplotlib will **automatically create the x-axis** as `[0, 1, 2, 3, 4]`.
So yes, you don't *need* to give the x-axis explicitly.

2️⃣ Case 2: You give both x and y values

```python
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
plt.plot(x, y)
plt.show()
```

👉 Now, it uses your given **x values** instead of default `[0, 1, 2, ...]`.

When to give x-axis?

- If your x values are just positions (0,1,2,3) → you can skip it.
- If your x values are meaningful (like dates, time, categories) → you should provide them.

🔹 Example with Dates on X-axis:

```python
import matplotlib.pyplot as plt

days = ["Mon", "Tue", "Wed", "Thu", "Fri"]
sales = [10, 20, 15, 25, 30]

plt.plot(days, sales, marker='o')
plt.xlabel("Day")
plt.ylabel("Sales")
plt.title("Sales Over Days")
plt.show()
```

👉 Here, giving x-axis (`days`) makes the chart **readable**.

Line Chart – Using Pandas

Syntax:

```
df.plot(kind="line")
```

- `df` → your pandas DataFrame
- `kind="line"` → specifies a line chart (default is line if omitted)
- You can optionally specify:
  - `x=` → column to use as x-axis
  - `y=` → column(s) to plot on y-axis
  - `title=` → chart title
  - `marker=` → marker style on points

Default Behaviour of df.plot()

`df.plot()`	DataFrame **index** (0, 1, 2, ... if not set)	All **numeric columns**
`df.plot(x="col_name")`	Column specified in `x=`	All numeric columns except x-column
`df.plot(x="col_name", y=["col1", "col2"])`	Column specified in `x=`	Only columns in `y=`

✅ Key points:

- Non-numeric columns **cannot be plotted on y-axis** unless you encode them.
- If x-axis is non-numeric (like dates, months), you **must specify** `x=`.

3️⃣ Single Line Chart Example

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

df = pd.DataFrame({
    "Month": ["Jan", "Feb", "Mar", "Apr", "May", "Jun"],
    "Sales": [100, 120, 140, 160, 180, 200]
})

df.plot(x="Month", y="Sales", kind="line", marker="o", title="Monthly Sales")
plt.ylabel("Sales")
plt.show()
```

What's happening:

- `Month` → x-axis
- `Sales` → y-axis
- `marker="o"` → adds dots for each point

4️⃣ Multiple Lines in One Chart

```python
df = pd.DataFrame({
    "Month": ["Jan", "Feb", "Mar", "Apr", "May", "Jun"],
    "Product_A": [120, 150, 170, 200, 220, 250],
    "Product_B": [100, 130, 160, 180, 210, 230],
    "Product_C": [90, 110, 140, 160, 190, 210]
})

df.plot(x="Month", y=["Product_A", "Product_B", "Product_C"], kind="line", marker="o")
plt.title("Monthly Sales by Product")
plt.ylabel("Sales")
plt.show()
```

Interview points:

- Multiple y-columns → multiple lines
- Automatically generates **legend**
- Markers and title improve readability

Interview Questions

🔹 Basic Questions

1. **How do you create a simple line chart in matplotlib?**
   - (Expected: plt.plot(y) or plt.plot(x, y))
2. **What happens if you don't provide the x-axis values in plt.plot()?**
   - (Expected: It automatically uses 0, 1, 2, ... as x-axis).
3. **How do you add labels and a title to a chart?**
   - (Expected: plt.xlabel(), plt.ylabel(), plt.title()).
4. **How do you display multiple line charts in the same plot?**
   - (Expected: Call plt.plot() multiple times with labels).
5. **How do you rotate x-axis labels (e.g., for dates or long text)?**
   - (Expected: plt.xticks(rotation=45)).
6. **How do you limit the range of x-axis or y-axis?**
   - (Expected: plt.xlim(min, max), plt.ylim(min, max)).
7. **How do you annotate a specific point in a line chart?**
   - (Expected: plt.annotate("text", (x, y))).
8. **You have sales data for multiple products across months. How will you plot them in one chart to compare trends?**
   - (Expected: Multiple lines + legend).
9. **If one line has much larger values than another, how will you show both clearly?**
   - (Expected: Use twin axes: ax.twinx()).
10. **If your dataset has thousands of points and the line chart looks messy, what will you do?**
    - (Expected: Downsample, smooth line, or aggregate data before plotting).
11. **How will you save a matplotlib line chart as an image file?**
    - (Expected: plt.savefig("chart.png")).

18. Difference between plt.plot() and df.plot() (pandas)?

- plt.plot() is matplotlib's function.
- df.plot() is pandas' wrapper built on top of matplotlib (less code).

Bar Chart

- A bar plot (or bar chart) is used to compare categories or groups using rectangular bars.
- Each bar's height (or length) shows the value or count of that category.

You use a bar plot when:

- You want to compare values across different categories.
- You have categorical data on one axis and numerical values on the other.

Examples:

- Comparing sales by product category
- Comparing number of students by department
- Comparing profit by region

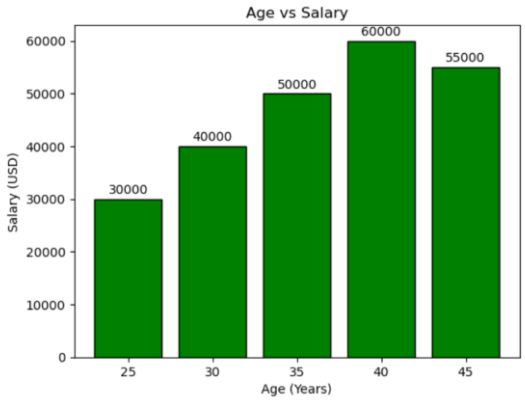
What is a Barh Plot?

- A barh plot is the horizontal version of a bar plot.
  It means the bars go sideways instead of up and down.
- Use barh when:
- Category names are long, so it's easier to read horizontally.
- You want to compare many categories clearly.
- ✅ Example: Comparing top 10 countries by population — names fit better horizontally.

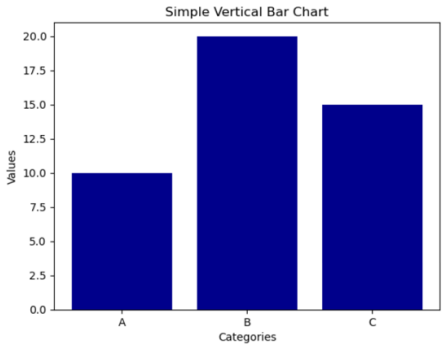
1. Main Function: plt.bar ()

- Used for **Vertical Bar Chart**

```python
import matplotlib.pyplot as plt

# Sample Data
ages = ['25', '30', '35', '40', '45']
salaries = [30000, 40000, 50000, 60000, 55000]

# Plot Column Bar Chart
plt.bar(ages, salaries, color='green', edgecolor='black')

# Add Labels and Title
plt.xlabel('Age (Years)')
plt.ylabel('Salary (USD)')
plt.title('Age vs Salary')

# Add Data Labels on top of each bar
for i, salary in enumerate(salaries):
    plt.text(i, salary + 1000, str(salary), ha='center')

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



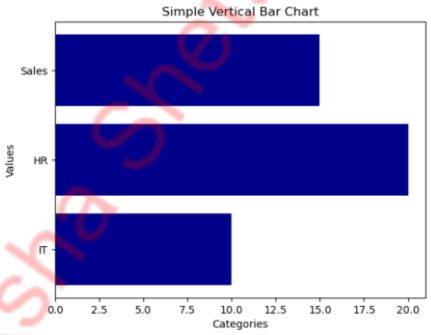
```python
import matplotlib.pyplot as plt

categories = ['A', 'B', 'C']
values = [10, 20, 15]

plt.bar(categories, values,color='darkblue')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Simple Vertical Bar Chart')
plt.show()
```



2. Horizontal Bar Chart: plt.barh()

- Useful when category labels are long

```python
import matplotlib.pyplot as plt

categories = ['IT', 'HR', 'Sales']
values = [10, 20, 15]

plt.barh(categories, values,color='darkblue')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Simple Vertical Bar Chart')
plt.show()
```

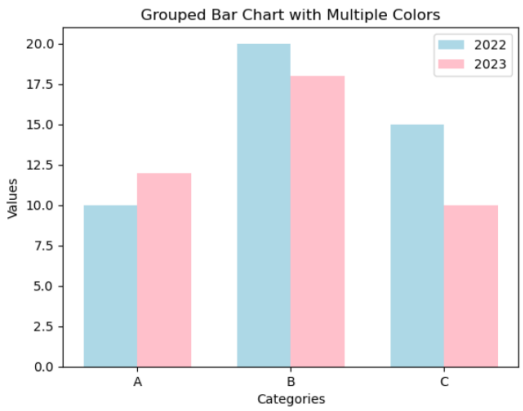


3. Grouped Bar Chart (Comparison of Multiple Series)

- Useful to compare multiple sets of values side by side

Example:

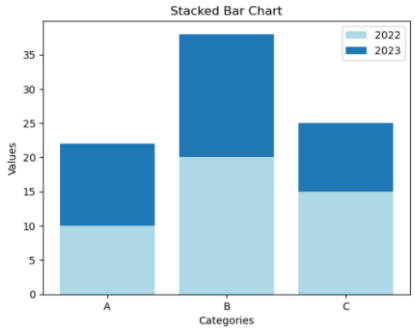
```python
import matplotlib.pyplot as plt
import numpy as np

categories = ['A', 'B', 'C']
values1 = [10, 20, 15]
values2 = [12, 18, 10]

x = np.arange(len(categories))  # [0,1,2]
width = 0.35

# Assign different colors to each bar in values1
colors1 = ['lightblue']
colors2 = ['pink']

plt.bar(x - width/2, values1, width=width, label='2022', color=colors1)
plt.bar(x + width/2, values2, width=width, label='2023', color=colors2)

plt.xticks(x, categories)
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Grouped Bar Chart with Multiple Colors')
plt.legend()
plt.show()
```



4. Stacked Bar Chart

- Bars are stacked vertically to show total and part-wise contribution

```
plt.bar(categories, values1, label='2022',color='lightblue')
plt.bar(categories, values2, bottom=values1, label='2023')

plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Stacked Bar Chart')
plt.legend()
plt.show()
```



Histogram

A **histogram** is a type of **bar chart** that shows the **distribution of numerical (continuous) data**.

- **It divides the data into intervals** called **bins**.
- Each bin shows **how many values fall into that range** (frequency).
- Taller bars = more values in that range.
- Shorter bars = fewer values.

Example:
You have ages of 100 people. Some are 18–25, some 26–35, some 36–45, etc.
A histogram will show how many people are in each age group.

💡 **Key idea:** Histograms are for **numerical data**, not categorical data.

2️⃣ How it Looks

- X-axis → ranges of values (bins)
- Y-axis → count/frequency of values in each range

Why we use it:

- To **understand data distribution**: normal, skewed, uniform, etc.
- To **detect outliers** or unusual patterns.
- To **compare different groups** easily.

When to Use a Histogram

- When you have **numerical data** and want to understand **its distribution**.
- When you want to **identify patterns** like clustering, gaps, or outliers.
- During **EDA (Exploratory Data Analysis)** to get insights before deeper analysis.
- Useful in **data science, business analysis, research, and reporting**.

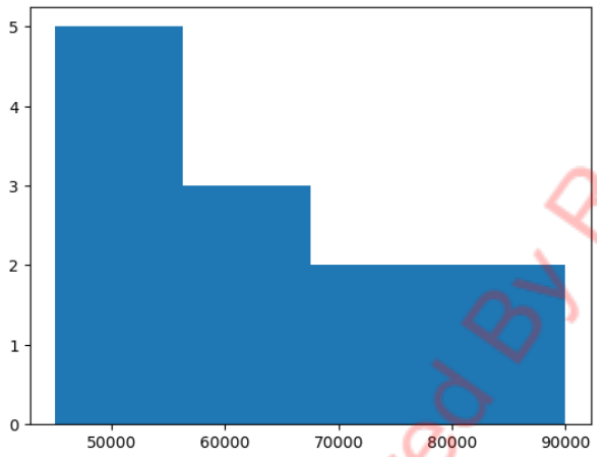
Examples:

	Age	Salary	Department	Experience
0	25	50000	HR	2
1	30	60000	IT	5
2	22	45000	Finance	1
3	35	80000	IT	7
4	28	52000	HR	3
5	40	90000	Finance	10
6	23	48000	IT	2
7	31	65000	HR	4
8	27	55000	Finance	3
9	29	58000	IT	5
10	34	72000	HR	6
11	32	68000	Finance	8

Example 1:

```
import matplotlib.pyplot as plt
plt.hist(df["Salary"],bins=4)
```

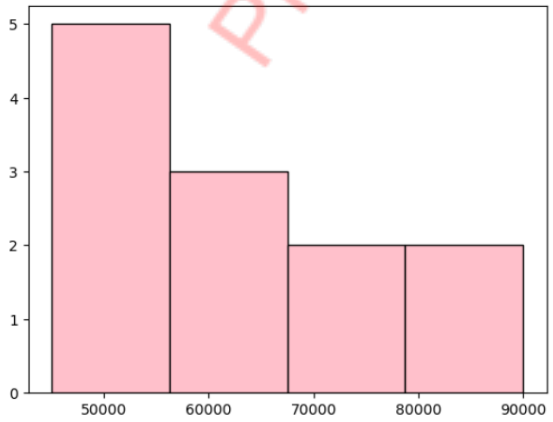
```
(array([5., 3., 2., 2.]),
 array([45000., 56250., 67500., 78750., 90000.]),
 <BarContainer object of 4 artists>)
```



Example 2:

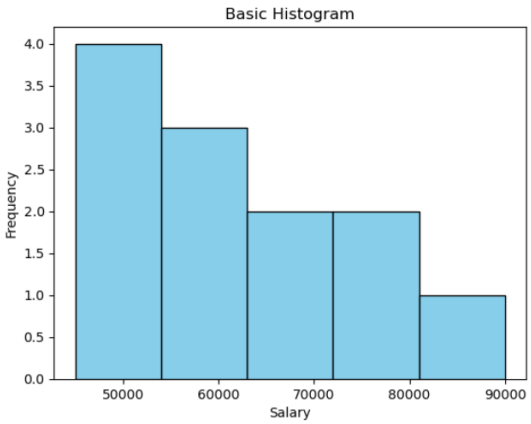
```
plt.hist(df["Salary"],bins=4,color='pink',edgecolor='black')
```

```
(array([5., 3., 2., 2.]),
 array([45000., 56250., 67500., 78750., 90000.]),
 <BarContainer object of 4 artists>)
```



Example 3:

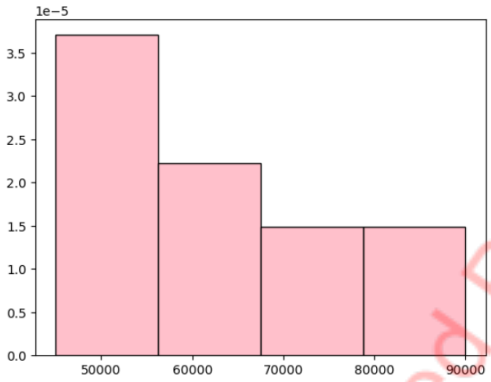
```
plt.hist(df["Salary"], bins=5, color='skyblue', edgecolor='black')
plt.title("Basic Histogram")
plt.xlabel("Salary")
plt.ylabel("Frequency")
plt.show()
```



Example 4:

```
plt.hist(df["Salary"],bins=4,color='pink',edgecolor='black',density=True)
```

```
(array([3.70370370e-05, 2.22222222e-05, 1.48148148e-05, 1.48148148e-05]),
 array([45000., 56250., 67500., 78750., 90000.]),
 <BarContainer object of 4 artists>)
```

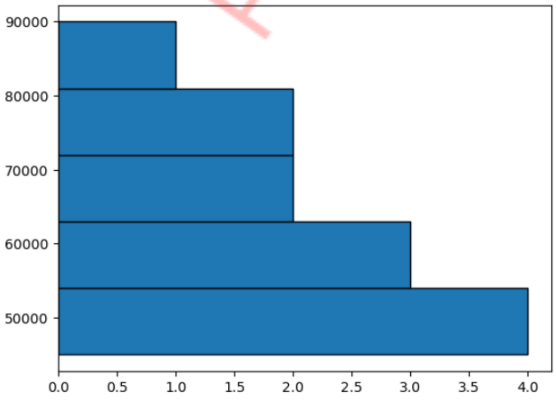


- If we use density = True it shows Probability instead of count.

Example 5:

```
plt.hist(df["Salary"], bins=5, orientation='horizontal',edgecolor='black')
```

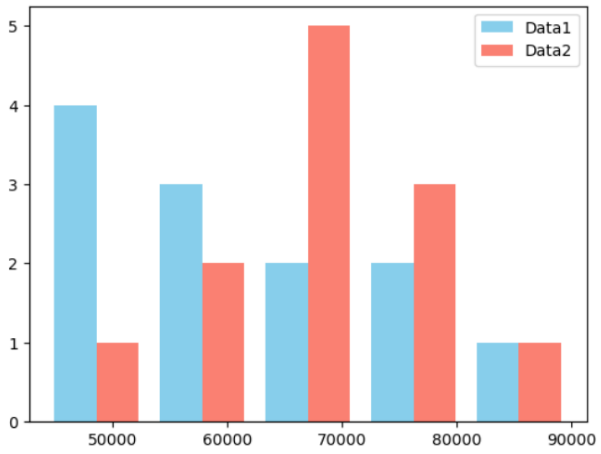
```
(array([4., 3., 2., 2., 1.]),
 array([45000., 54000., 63000., 72000., 81000., 90000.]),
 <BarContainer object of 5 artists>)
```



Example 6: For Multiple Columns

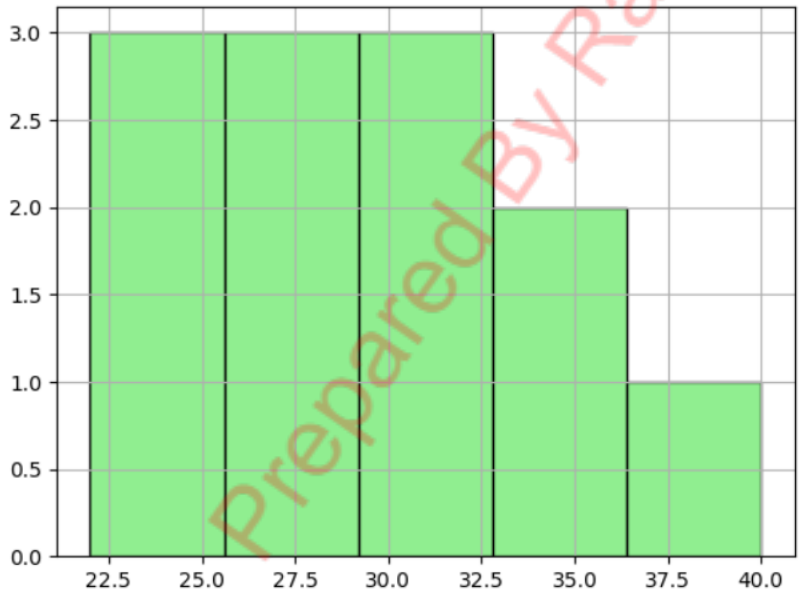
```
data2 = [10, 14, 13, 18, 17, 16, 20]
plt.hist([df["Salary"], df["Salary1"]], bins=5, color=['skyblue','salmon'], label=['Data1','Data2'])
plt.legend()
```

```
<matplotlib.legend.Legend at 0x215a59353d0>
```



Histogram Using Pandas

```
df['Age'].hist(bins=5, color='lightgreen', edgecolor='black')
plt.show()
```

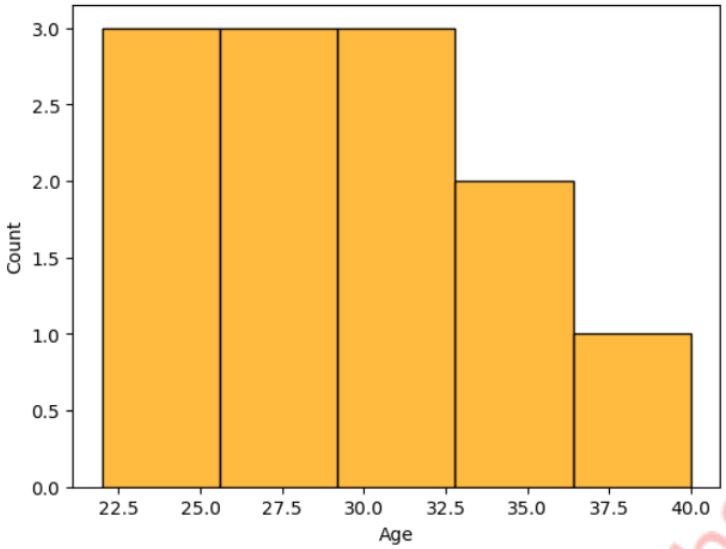


Pandas also allows:

- `df['column'].plot(kind='hist', bins=10)`
- Quick **data exploration**: `df['column'].hist()` immediately shows distribution.

Histogram Using Seaborn

```python
import seaborn as sns

sns.histplot(x=df['Age'], bins=5, kde=False, color='orange')
plt.show()
```



- `kde=True` adds a **smooth density line** over histogram.
- `hue` can separate categories:

```python
df['Gender'] = ['M','F','M','F','M','F','M','F','M','F','M','F']
sns.histplot(data=df, x='Age', hue='Gender', bins=5, multiple='dodge')
```

Tips

- Histogram is not a bar chart — bar chart is categorical; histogram is numerical/frequency-based.
- Always label axes and title your chart.
- Explain why you chose bins or colours if asked.
- Mention outlier detection and distribution understanding.

COUNT PLOT

- A count plot is a type of bar chart that shows the number of times each category appears in your data.
- Count plot is used to count and display the frequency of categories in a column.

Why is count plot used?

We use count plot when we want to:

- See how many times each category occurs in a dataset.
- Compare category frequencies easily.
- Understand data distribution for categorical variables (like gender, product type, or department).
- It's super useful in Exploratory Data Analysis (EDA) — when you first start exploring your data.

Examples:

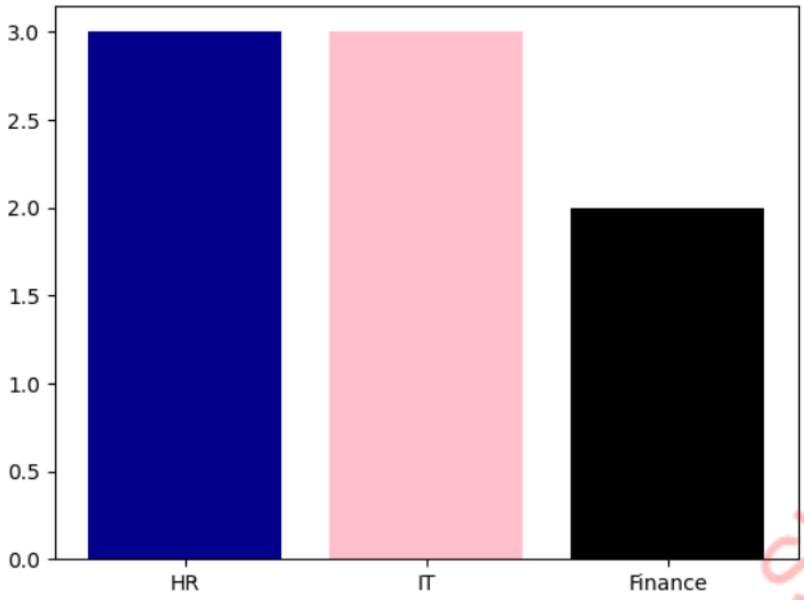
	Name	Department	Gender	Experience
0	Asha	HR	F	2
1	Ravi	IT	M	5
2	Meena	IT	F	3
3	Arjun	HR	M	4
4	Kiran	Finance	M	6
5	Priya	Finance	F	1
6	Manoj	IT	M	7
7	Divya	HR	F	2

```
import matplotlib.pyplot as plt

count = df['Department'].value_counts()
count
```

```
Department
HR         3
IT         3
Finance    2
Name: count, dtype: int64
```

```
plt.bar(count.index, count.values, color=["darkblue","pink","black"])
```

```
<BarContainer object of 3 artists>
```



Bivariate Analysis

Here we are going to analyse the relationship between 2 features.

Scatter Plot

A **scatter plot** is used to **visualize the relationship between two numerical variables**.
Each point represents an observation one variable on the X-axis and another on the Y-axis.

Example use:

- Compare height vs weight
- See if advertising spend increases sales
- Check correlation between age and income

Use Case:

- "We use a scatter plot when we want to understand the relationship between two continuous numerical variables.
- For example, I can use it to check how salary changes with age.
- If the points move upward in a pattern, it means there's a positive correlation.
- Scatter plots are great for spotting trends, correlations, and outliers."

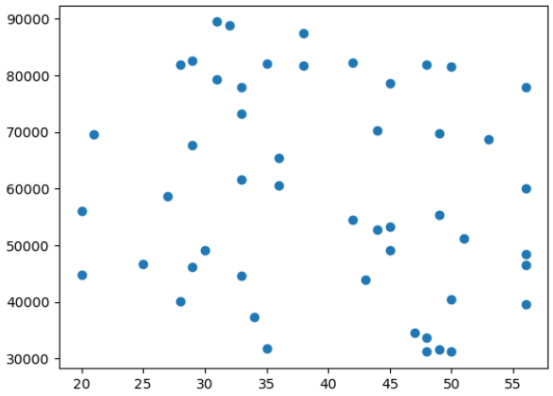
Examples:

	Age	Salary	Experience	Department	Performance
0	29	82568	10	HR	4.543308
1	56	60091	6	IT	2.461460
2	35	31681	1	IT	1.875077
3	20	56030	3	IT	4.009985
4	48	31159	8	Sales	1.427518

Example 1:

```
import matplotlib.pyplot as plt

plt.scatter(df['Age'], df['Salary'])
```

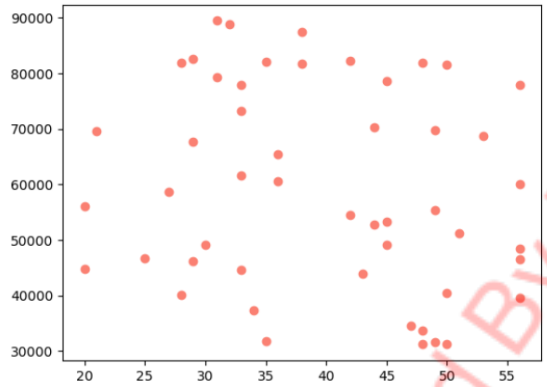
```
<matplotlib.collections.PathCollection at 0x19745751390>
```



Example 2:

```
plt.scatter(df["Age"],df["Salary"],color='salmon')
```

```
<matplotlib.collections.PathCollection at 0x1974b051b50>
```

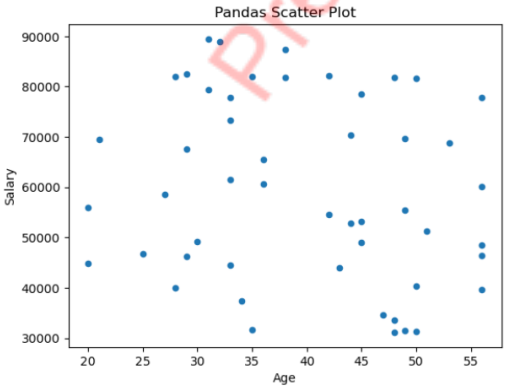


Using Pandas:

Example 1:

```
df.plot.scatter(x='Age', y='Salary', title='Pandas Scatter Plot')
```

```
<Axes: title={'center': 'Pandas Scatter Plot'}, xlabel='Age', ylabel='Salary'>
```



Box Plot

A Box Plot (also called a Box-and-Whisker Plot) is a graph that shows the spread and summary of a dataset using five key numbers:

1. Minimum
2. First Quartile (Q1) – 25% of the data lies below this value
3. Median (Q2) – middle value (50%)
4. Third Quartile (Q3) – 75% of the data lies below this value
5. Maximum

So, it basically tells us:

- How the data is spread out
- Where the middle (median) lies
- If there are outliers (unusual extreme values)

Why Box Plot is Used

We use a box plot when we want to:

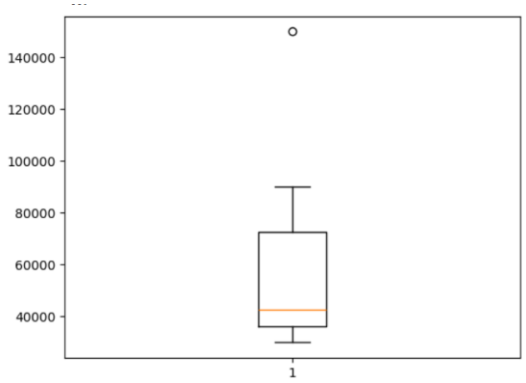
- Understand data distribution quickly
- Compare distributions across groups (like comparing experience of employees in different departments)
- Detect outliers (data points that are too high or low)
- See how spread out the values are
- It's very common in data analysis and data science for understanding numerical data.
- You can draw a box plot only if you have numeric data with enough variety and no missing values.
- Optionally, you can group by a category to compare distributions.

Examples:

	Name	Department	Gender	Experience	Salary	Age
0	Alice	HR	F	2	30000	25
1	Ravi	IT	M	5	50000	28
2	Meena	IT	F	3	45000	26
3	Arjun	HR	M	4	40000	30
4	Kiran	Finance	M	6	80000	35
5	Priya	Finance	F	1	32000	24
6	Manoj	IT	M	7	90000	40
7	Divya	HR	F	2	35000	27
8	Rohit	IT	M	15	150000	50
9	Sneha	Finance	F	3	40000	29

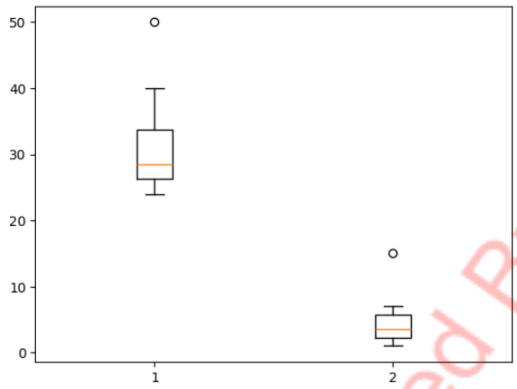
Scenario 1: To pass one column / one value

```
plt.boxplot(x = df["Salary"])
```



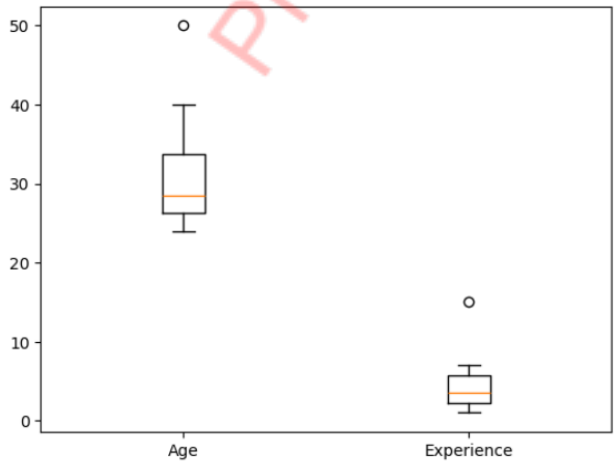
Scenario 2: To pass multiple columns or values we have to pass in the form of list.

```
plt.boxplot([df["Age"], df["Experience"]])
```



Scenario 3: Labelling

```
plt.boxplot([df["Age"], df["Experience"]],labels=["Age","Experience"])
```



Scenario 4:

```
Hr_Salary = df[df["Department"] == "HR"]["Salary"]
Hr_Salary
```

```
0    30000
3    40000
7    35000
Name: Salary, dtype: int64
```

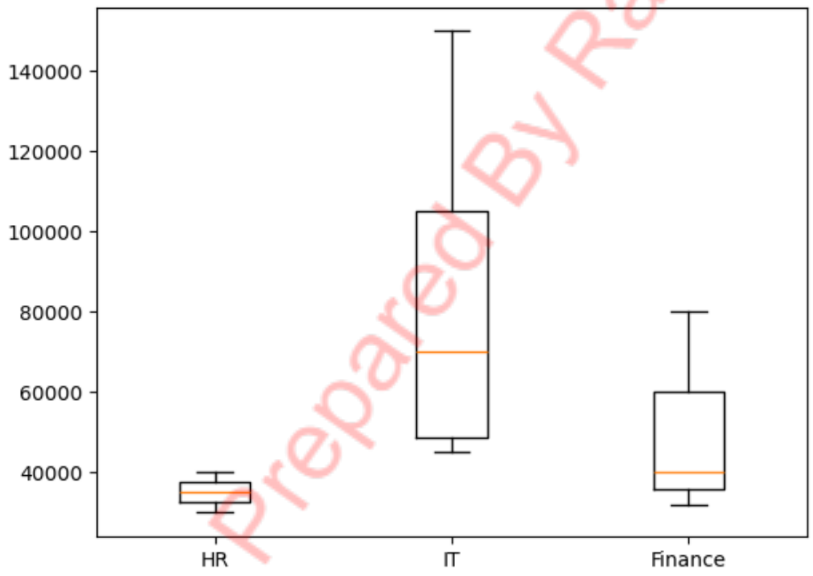
```
IT_Salary = df[df["Department"] == "IT"]["Salary"]
IT_Salary
```

```
1     50000
2     45000
6     90000
8    150000
Name: Salary, dtype: int64
```

```
Finance_Salary = df[df["Department"] == "Finance"]["Salary"]
Finance_Salary
```

```
4    80000
5    32000
9    40000
Name: Salary, dtype: int64
```

```
plt.boxplot([Hr_Salary, IT_Salary, Finance_Salary], labels=['HR', 'IT', 'Finance'])
```



Bubble Plot

- A Bubble Plot is like a scatter plot, but with an extra dimension of information shown by the size of the bubbles (circles).
- A bubble plot shows relationships between three numeric variables one on the X-axis, one on the Y-axis, and one as the bubble size.

hen to Use a Bubble Plot

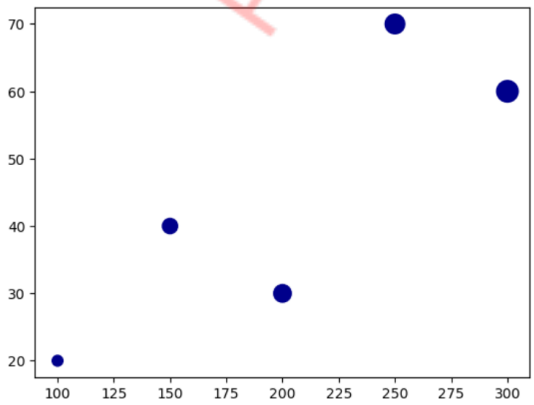
- Use a bubble plot when:
  1. You want to compare relationships between two numeric variables (like scatter plot)
  2. And also show a third numeric factor using the size of bubbles
- It's often used in:
- Business analysis → showing revenue vs profit vs market size
- Data analytics → comparing performance across categories
- Economics → GDP vs Life Expectancy vs Population

Examples:

	Company	Sales	Profit	MarketShare
0	A	100	20	30
1	B	150	40	60
2	C	200	30	80
3	D	250	70	100
4	E	300	60	120

Scenario 1:

```
plt.scatter(df["Sales"],df["Profit"],s=df["MarketShare"],color='darkblue')
```



In Matplotlib's **scatter plot**,
the parameter `s` stands for "**size of the markers (bubbles)**".

👉 In simple words:

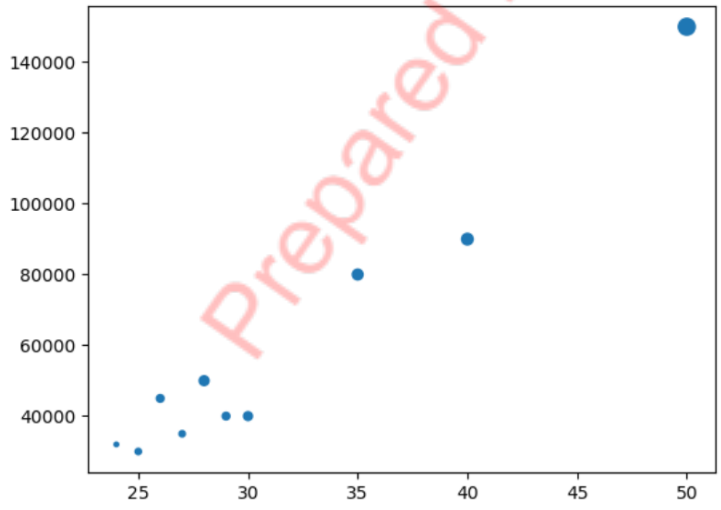
> `s=` controls **how big each dot or bubble** appears on the plot.

Scenario 2:

	Name	Department	Gender	Experience	Salary	Age
0	Alice	HR	F	2	30000	25
1	Ravi	IT	M	5	50000	28
2	Meena	IT	F	3	45000	26
3	Arjun	HR	M	4	40000	30
4	Kiran	Finance	M	6	80000	35
5	Priya	Finance	F	1	32000	24
6	Manoj	IT	M	7	90000	40
7	Divya	HR	F	2	35000	27
8	Rohit	IT	M	15	150000	50
9	Sneha	Finance	F	3	40000	29

```
plt.scatter(df["Age"],df["Salary"],s=df["Experience"]*6)
```

```
<matplotlib.collections.PathCollection at 0x1b3f0f96750>
```

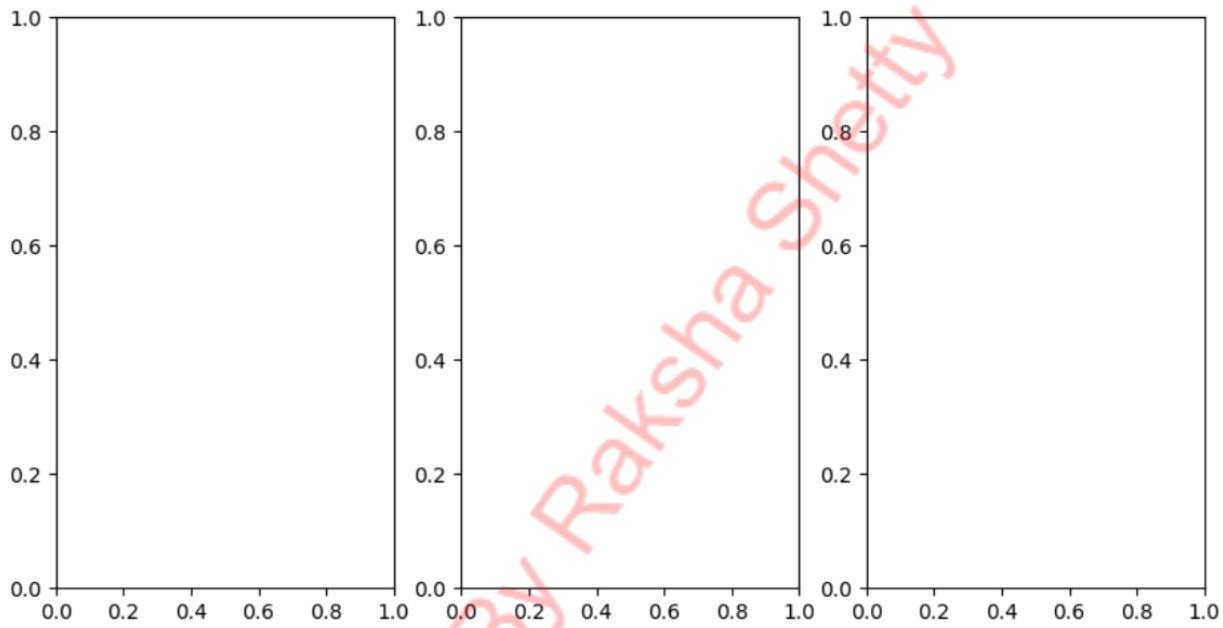


- I used *6 here because to make size of bubble big it was not visible that is why you can multiply anything.

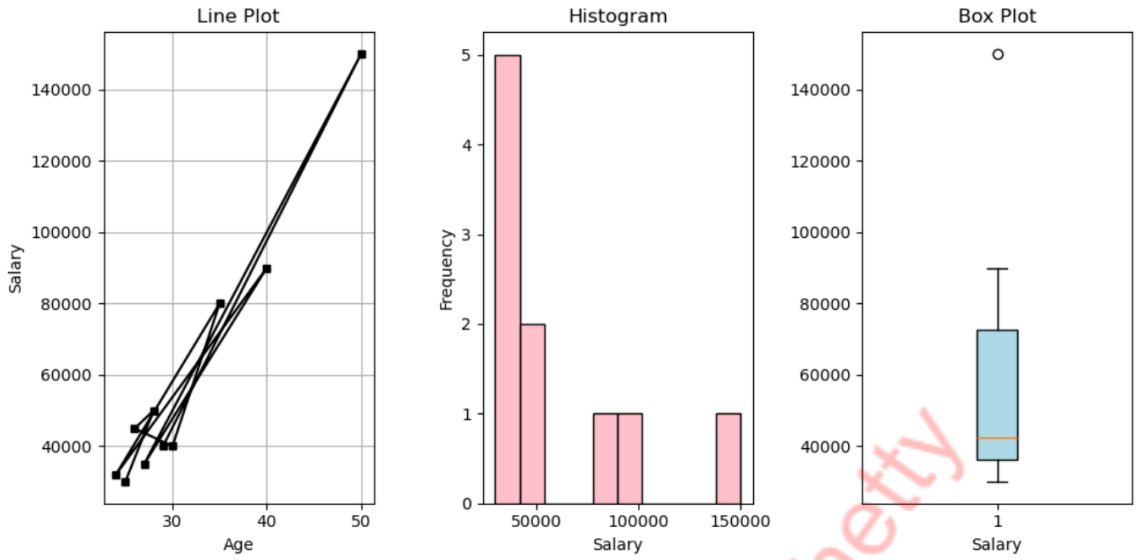
Object Oriented API

It is used for subplots or multiple plots in a particular figure.

```
fig, axs = plt.subplots(1,3,figsize=(10,5))

# “Create a single figure (fig) with 1 row and 3 side-by-side plots (axs), and make the total figure size 10 inches wide and 5 inches tall.”
# fig → the overall figure (like a big canvas)
# axs → the 3 small plot areas (axes) inside it
# (1, 3) → 1 row × 3 columns → 3 plots in a row
# figsize=(10, 5) → total width = 10 inches, height = 5 inches
```




```
fig, axs = plt.subplots(1,3,figsize=(10,5))

# “Create a single figure (fig) with 1 row and 3 side-by-side plots (axs), and make the total figure size 10 inches wide and 5 inches tall.”
# fig → the overall figure (like a big canvas)
# axs → the 3 small plot areas (axes) inside it
# (1, 3) → 1 row × 3 columns → 3 plots in a row
# figsize=(10, 5) → total width = 10 inches, height = 5 inches

# Line Plot
axs[0].plot(df["Age"], df["Salary"], color='black', marker='s', markersize=4)
axs[0].grid()
axs[0].set_title("Line Plot")
axs[0].set_xlabel("Age")
axs[0].set_ylabel("Salary")

# Histogram
axs[1].hist(df["Salary"], bins=10, color='pink',edgecolor='black')
axs[1].set_title("Histogram")
axs[1].set_xlabel("Salary")
axs[1].set_ylabel("Frequency")

# Box Plot
axs[2].boxplot(df["Salary"], patch_artist=True, boxprops=dict(facecolor='lightblue'))
axs[2].set_title("Box Plot")
axs[2].set_xlabel("Salary")

plt.tight_layout()   # Adjust spacing so titles and labels fit nicely
plt.show()
```



3. Understanding pyplot

- Whatis pyplot?
- Dierence between Matplotlib vs Pyplot
- How pyplot acts like a simple interface
- First basic plot: Plotting a simple line graph with plt.plot()
- Plot displaying: plt.show()
- Saving a plot: plt.savefig()

4. Plotting Line Graphs

- Plotting multiple lines in a single graph
- Setting X and Y labels
- Customizing line styles (dotted, dashed, solid)
- Plotting data from lists and NumPy arrays
- Plotting dates ortime series data

5. Adding Labels, Titles, and Legends

- Setting graph title: plt.title()
- Labeling X and Y axes: plt.xlabel(), plt.ylabel()
- Adding legends to graphs: plt.legend()
- Positioning legends (top, right, custom placement)
- Adding grids for better readability: plt.grid(True)

6. Working with Colors, Line Styles, and Markers

- Changing line colors manually (color argument)
- Using RGB, HEX, or named colors
- Setting line width (linewidth) and styles (linestyle)
- Customizing markers (dot, cross, square,triangle)
- Adding marker size (markersize) and colors (markerfacecolor, markeredgecolor)

7. Plotting Bar Charts, Pie Charts, and Histograms

Bar Charts
- Vertical and Horizontal bar charts
- Setting bar colors, edge colors
- Grouped bar charts and stacked bar charts
- Adding values on top of bars

Pie Charts
- Simple pie chart using plt.pie()
- Adding labels, percentages, shadows, explode
- Changing start angles and radius

Histograms
- Whatis a histogram? When to use it?
- Plotting a basic histogram
- Changing bins, range, and colors
- Understanding frequency distribution

8. Subplots and Layout Adjustments

- What are subplots?
- Creating multiple plots in one figure (plt.subplot())
- Using plt.subplots()for grid layout
- Adjusting space between plots: plt.tight_layout()
- Sharing axes, custom grid sizes
- Nested subplots (optional advanced)

9. Saving Figures (Simple Beginner Guide)

- Saving plots as images (PNG,JPG, SVG, PDF)
- plt.savefig('filename.png') usage
- Setting DPI (Dots per Inch)for better resolution
- Saving without extra white space (bbox_inches='tight')
- Saving dierent parts of figure

Seaborn

- **To install Seaborn:** pip install seaborn
- **To import:** import seaborn as sns

1. Seaborn comes with several built-in example datasets

```
import seaborn as sns

# List all available example datasets
print(sns.get_dataset_names())
```

Some of the datasets are:

```
'iris',
'mpg',
'penguins',
'planets',
'seaice',
'taxis',
'tips',
'titanic']
```

2. To load any datasets

```
penguins = sns.load_dataset('penguins')
penguins.head(6)
```