

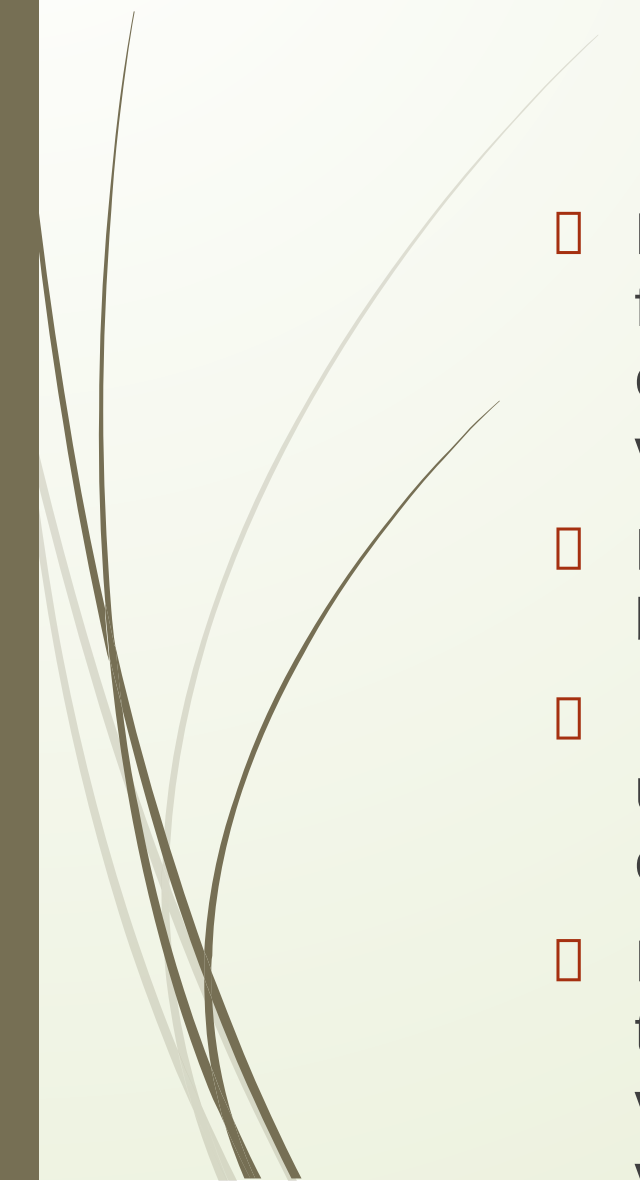


# Data Visualization

Data Science with Python



# What is Data Visualization?

- 
- ❑ Data Visualization is basically putting the analyzed data in the form of visuals i.e - graphs, images. These visualizations make it easy for humans to understand the analyzed trends through visuals.
  - ❑ Data Visualization is very important when it comes to analyzing big datasets.
  - ❑ When data scientists analyze complex datasets they also need to understand the insights collected. Data Visualization will make it easier for them to understand through graphs and charts.
  - ❑ Data visualization is a way of presenting complex data in a form that is graphical and easy to understand. When analyzing large volumes of data and making data-driven decisions, data visualization is crucial.



# The advantages and benefits of effective data visualization

- ❑ Data visualization allows you to:
- ❑ Get an initial understanding of your data by making trends, patterns, and outliers easily visible .
- ❑ Comprehend large volumes of data quickly and efficiently
- ❑ Communicate insights and findings to non-data experts, making your data accessible and actionable
- ❑ Tell a meaningful and impactful , highlighting only the most relevant information for a given context.
- ❑ an increased ability to act on findings quickly and, therefore, achieve success with greater speed and less mistakes.

# When should you visualize your data?

data visualization which takes usually comprises the final step in the data analysis process.

## THE DATA ANALYSIS PROCESS

**Step 1:**

Define the question

**Step 2:**

Collect the data

**Step 3:**


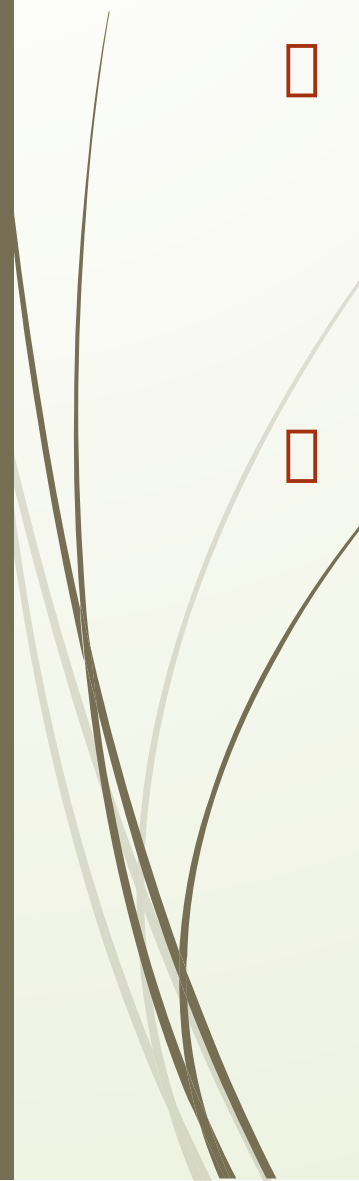
Clean the data

**Step 4:**

Analyze the data

**Step 5:**

Visualize and share your findings

- 
- 
- ❑ **Visualize the data and share your findings:** Translate your key insights into visual format (e.g. graphs, charts, or heatmaps) and present them to the relevant audience(s).
  - ❑ By using visual elements like **charts, graphs, and maps**, data visualization techniques provide an accessible way to see and **understand trends, outliers, and patterns in data.**


# What is data visualization used for?

**Convey changes over time:** For example, a line graph could be used to present how the value of Bitcoin changed over a certain time period.

- **Determine the frequency of events:** You could use a histogram to visualize the frequency distribution of a single event over a certain time period (e.g. number of internet users per year from 2007 to 2021).
- **Highlight interesting relationships or correlations between variables:** If you wanted to highlight the relationship between two variables (e.g. marketing spend and revenue, or hours of weekly exercise vs. cardiovascular fitness), you could use a scatter plot to see, at a glance, if one increases as the other decreases (or vice versa).
- **Examine a network:** If you want to understand what's going on within a certain network (for example, your entire customer base), network visualizations can help you to identify (and depict) meaningful connections and clusters within your network of interest.
- **Analyze value and risk:** in order to figure out which opportunities or strategies are worth pursuing, data visualizations—such as a color-coded system—could help you to categorize and identify, at a glance, which items are feasible.



# How to visualize your data: Different types of data visualization

- There are many different options when it comes to visualizing your data.
  - The visualization you choose depends on the type of data you're working with and what you want to convey or highlight.
  - It's also important to consider the complexity of your data and how many different variables are involved.
- 



# Five data visualization categories


- When considering the different types of data viz,
- **Temporal data visualizations** are linear and one-dimensional. Examples include scatterplots, timelines, and line graphs.
- **Hierarchical visualizations** organize groups within larger groups, and are often used to display clusters of information. Examples include tree diagrams, ring charts.
- **Network visualizations** show the relationships and connections between multiple datasets. Examples include matrix charts, word clouds, and node-link diagrams.
- **Multidimensional or 3D visualizations** are used to depict two or more variables. Examples include pie charts, Venn diagrams, stacked bar graphs, and histograms.
- **Geospatial visualizations** convey various data points in relation to physical, real-world locations (for example, voting patterns across a certain country). Examples include heat maps, cartograms, and density maps.





# Python Libraries for Data Visualization

- ❑ Python offers numerous libraries for data visualization, each with unique features and advantages. Below are some of the most popular libraries:
- ❑ Here are some of the most popular ones:
- ❑ **Matplotlib**
- ❑ **Seaborn**
- ❑ **Pandas**
- ❑ **Plotly** Plotly is an interactive visualization library.
- ❑ **Plotnine**
- ❑ **Altair**
- ❑ **Bokeh**
- ❑ **Pygal**
- ❑ **Geoplotlib**

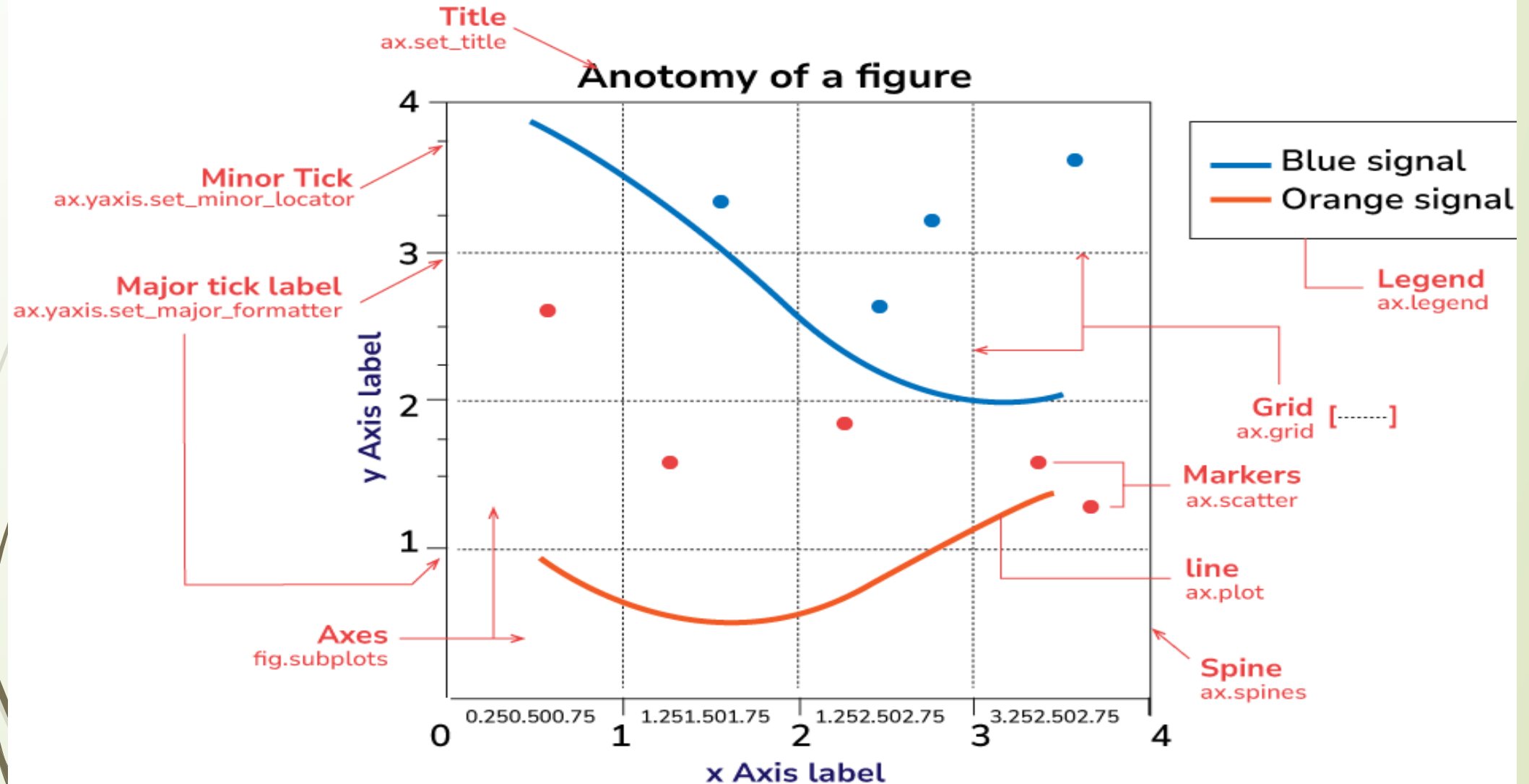


# Getting Started – Data Visualization with Matplotlib

- ❑ Matplotlib is a great way to begin visualizing data in Python, essential for data visualization in data science.
- ❑ Matplotlib is a powerful and versatile open-source plotting library for Python, designed to help users visualize data in a variety of formats.
- ❑ It is a versatile library that designed to help users visualize data in a variety of formats.
- ❑ Well-suited for creating a wide range of static, animated, and interactive plots.
- ❑ **If you want to convert your boring data into interactive plots and graphs, Matplotlib is the tool for you.**

# Components or Parts of Matplotlib Figure

## Matplotlib





The parts of a Matplotlib figure include (as shown in the figure above):

- ❑ **Figure:** The overarching container that holds all plot elements, acting as the canvas for visualizations.
- ❑ **Axes:** The areas within the figure where data is plotted; each figure can contain multiple axes.
- ❑ **Axis:** Represents the x-axis and y-axis, defining limits, tick locations, and labels for data interpretation.
- ❑ **Lines and Markers:** Lines connect data points to show trends, while markers denote individual data points in plots like scatter plots.
- ❑ **Title and Labels:** The title provides context for the plot, while axis labels describe what data is being represented on each axis.

# Matplotlib Pyplot

- ❑ Pyplot is a submodule within Matplotlib that provides a MATLAB-like interface for making plots. It simplifies the process of adding plot elements such as lines, images, and text to the axes of the current figure.
- ❑ The `plot()` method is one of the most essential functions in Pyplot used to create line graphs, which are the foundation of most visualizations. This method takes at least two arguments: the x-values and the y-values. Syntax:
- ❑ **Steps to Use Pyplot:**
  - ❑ **Import Matplotlib:** Start by importing `matplotlib.pyplot` as `plt`.
  - ❑ **Create Data:** Prepare your data in the form of lists or arrays.
  - ❑ **Plot Data:** Use `plt.plot()` to create the plot.
  - ❑ **Customize Plot:** Add titles, labels, and other elements using methods like `plt.title()`, `plt.xlabel()`, and `plt.ylabel()`.
  - ❑ **Display Plot:** Use `plt.show()` to display the plot.

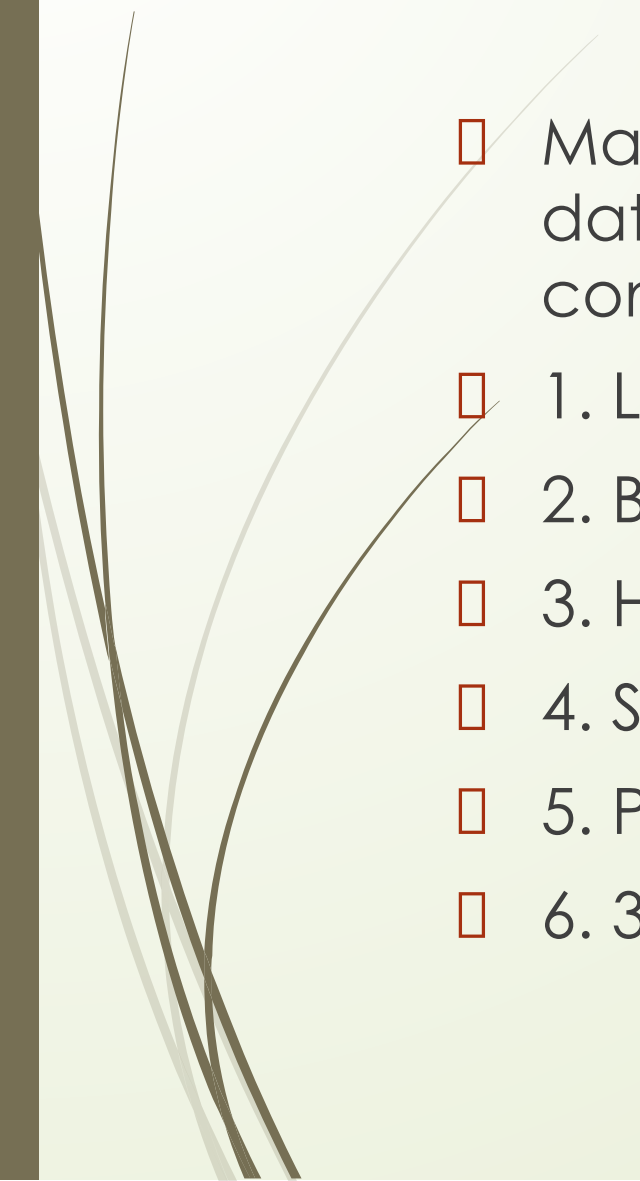


# Key Uses of Matplotlib:

- ❑ **Basic Plots:** Line plots, bar charts, histograms, scatter plots, etc.
- ❑ **Statistical Visualization:** Box plots, error bars, and density plots.
- ❑ **Customization:** Control over colors, labels, gridlines, and styles.
- ❑ **Subplots & Layouts:** Create multiple plots in a single figure.
- ❑ **3D Plotting:** Surface plots and 3D scatter plots using `mpl_toolkits.mplot3d`.
- ❑ **Animations & Interactive Plots:** Dynamic visualizations with `FuncAnimation`.
- ❑ **Integration:** Works well with Pandas, NumPy and Jupyter Notebooks.



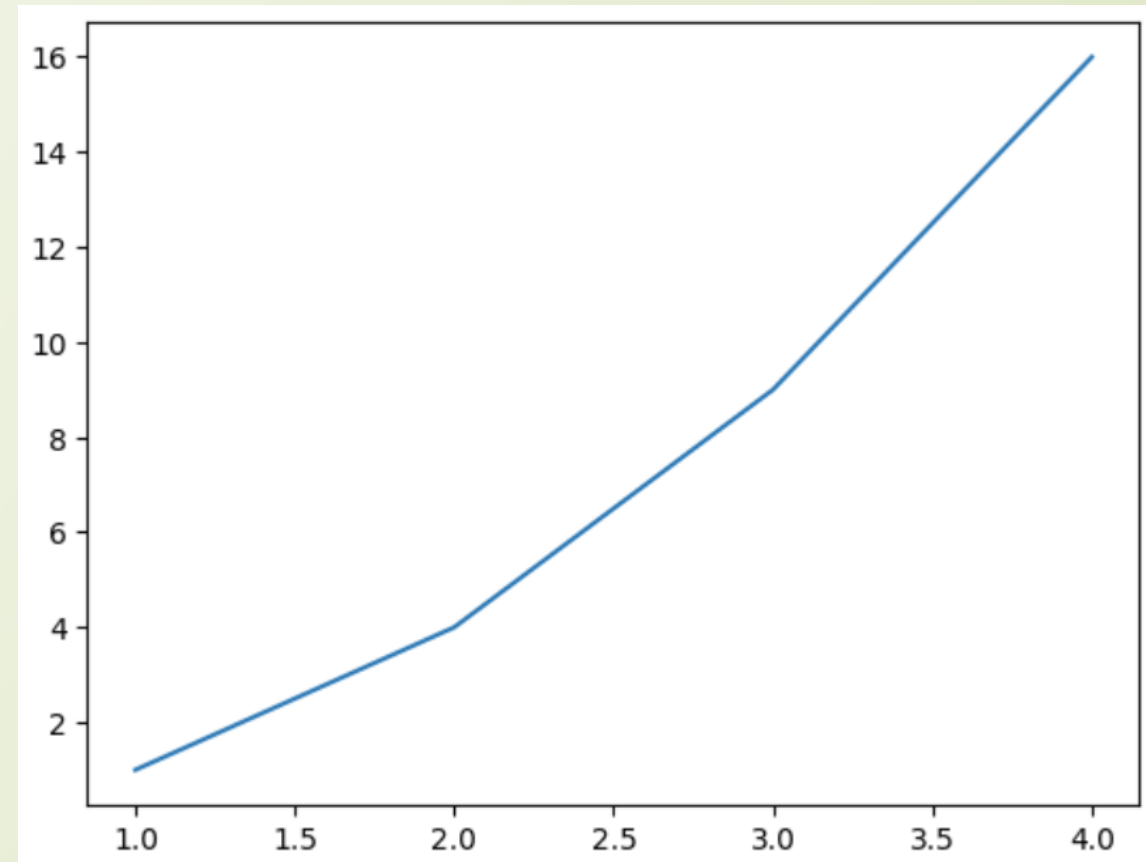
# Different Types of Plots in Matplotlib

- Matplotlib offers a wide range of plot types to suit various data visualization needs. Here are some of the most commonly used types of plots in Matplotlib:
  - 1. Line Graph
  - 2. Bar Chart
  - 3. Histogram
  - 4. Scatter Plot
  - 5. Pie Chart
  - 6. 3D Plot
- 



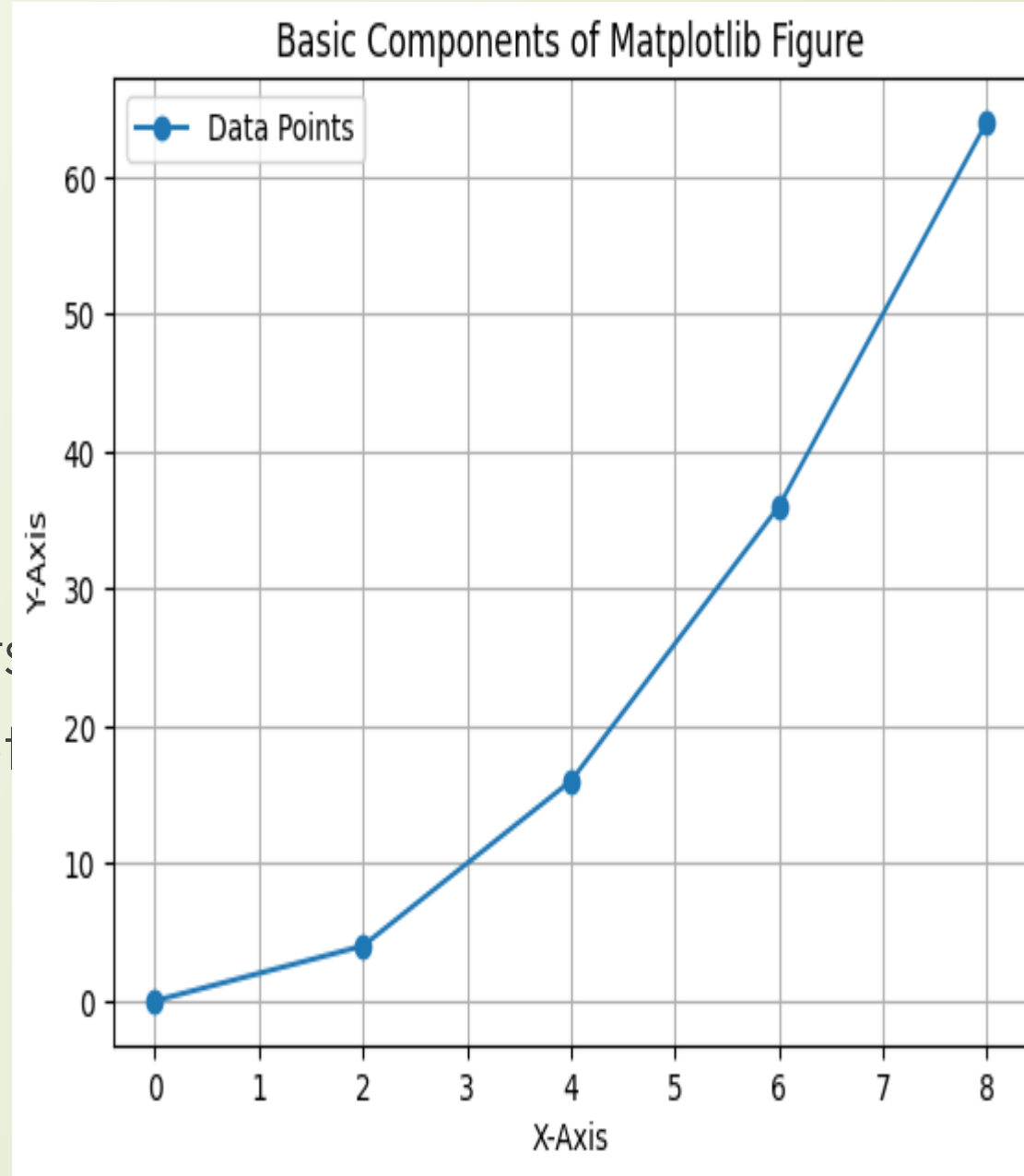
# Simple Example

- ❑ `import matplotlib.pyplot as plt`
- ❑ `x=[1, 2, 3, 4]`
- ❑ `y=[1, 4, 9, 16]`
- ❑ `plt.plot(x,y)`
- ❑ `plt.show()`



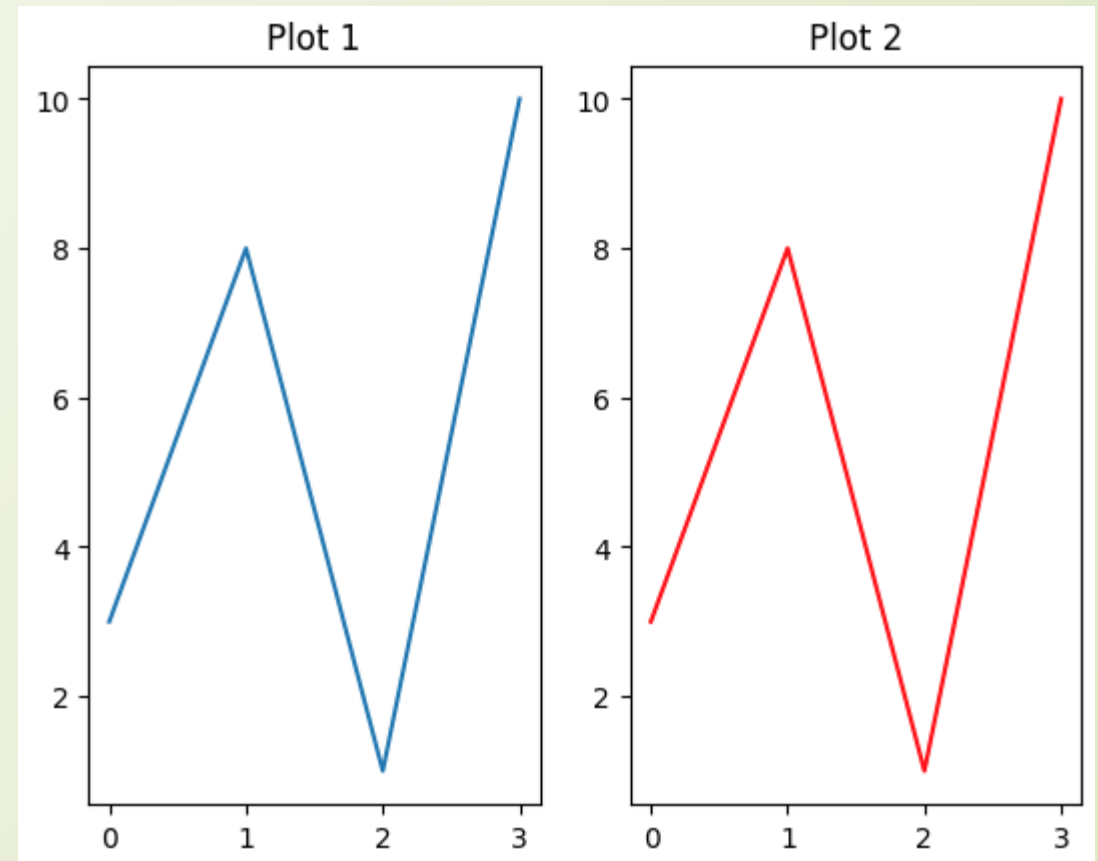
# Example:

- ❑ `import matplotlib.pyplot as plt`
- ❑ `x = [0, 2, 4, 6, 8]`
- ❑ `y = [0, 4, 16, 36, 64]`
- ❑ `fig, ax = plt.subplots()`
- ❑ `ax.plot(x, y, marker='o', label="Data Points")`
- ❑ `ax.set_title("Basic Components of Matplotlib Figure")`
- ❑ `ax.set_xlabel("X-Axis")`
- ❑ `ax.set_ylabel("Y-Axis")`
- ❑ `plt.show()`



# Creating Multiple Plots with subplots()

- ❑ The `subplots()` function in Matplotlib allows plotting multiple plots using the same data or axes. For example, setting `nrows=1` and `ncols=2` creates two subplots that share the y-axis.
- ❑ `import matplotlib.pyplot as plt`
- ❑ `import numpy as np`
- ❑ `x = np.array([0, 1, 2, 3])`
- ❑ `y = np.array([3, 8, 1, 10])`
- ❑ `fig, ax = plt.subplots(1, 2)`
- ❑ `ax[0].plot(x, y)`
- ❑ `ax[0].set_title('Plot 1')`
- ❑ `ax[1].plot(x, y, 'r')`
- ❑ `ax[1].set_title('Plot 2')`
- ❑ `plt.show()`

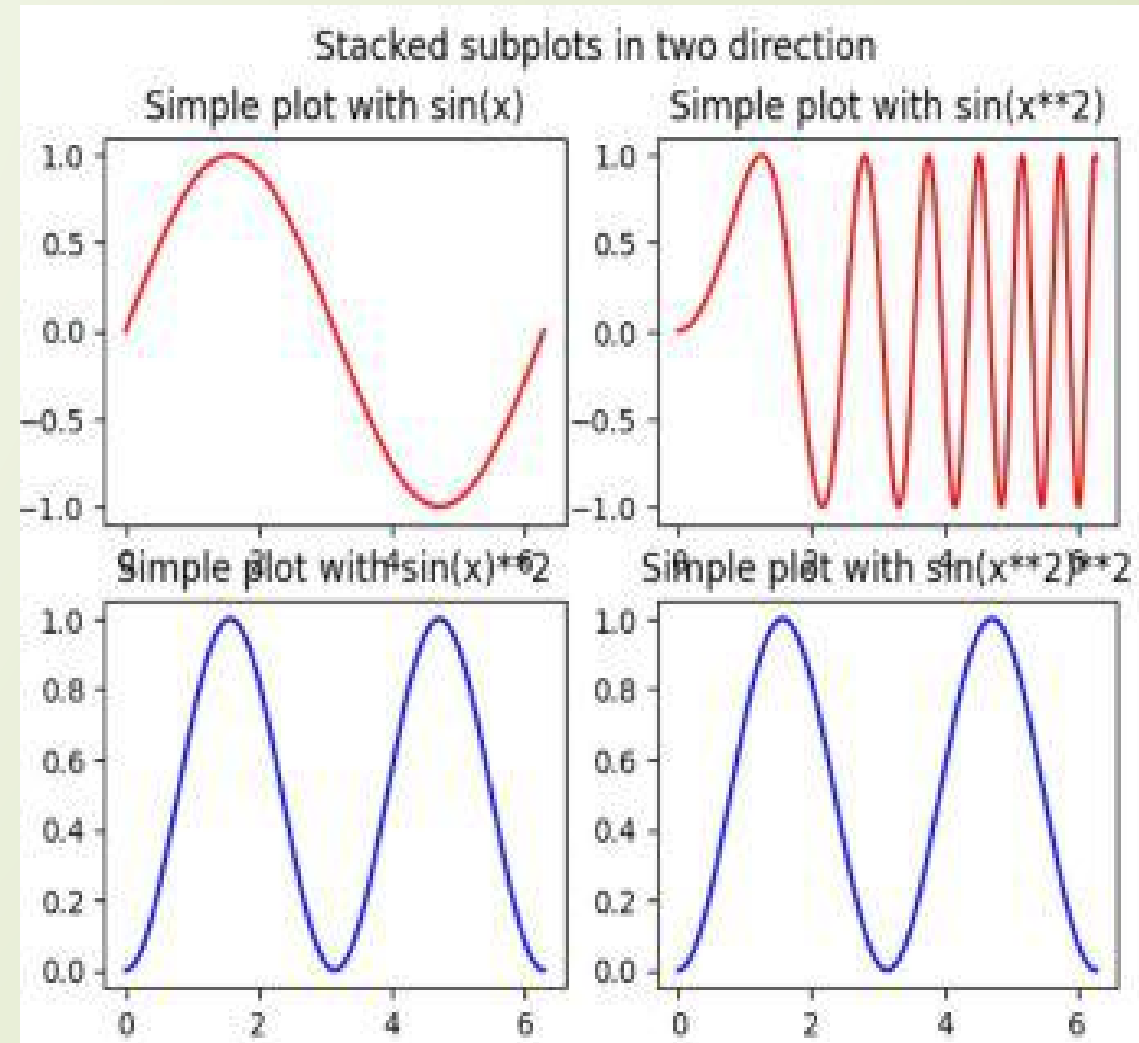


# Stacking Subplots in Two Directions

- You can stack subplots vertically and horizontally by adjusting the `nrows` and `ncols` parameters in `subplots()`. This example demonstrates a 2×2 grid layout.
  - # Implementation of matplotlib function
  - `import numpy as np`
  - `import matplotlib.pyplot as plt`
  - # First create some toy data:
  - `x = np.linspace(0, 2 * np.pi, 400)`
  - `y1 = np.sin(x)`
  - `y2 = np.sin(x**2)`
  - `y3 = y1**2`
  - `y4 = y2**2`
- ```
fig, ax = plt.subplots(nrows=2, ncols=2)
ax[0, 0].plot(x, y1, c='red')
ax[0, 1].plot(x, y2, c='red')
ax[1, 0].plot(x, y3, c='blue')
ax[1, 1].plot(x, y4, c='blue')
ax[0, 0].set_title('Simple plot with sin(x)')
ax[0, 1].set_title('Simple plot with sin(x**2)')
ax[1, 0].set_title('Simple plot with sin(x)**2')
ax[1, 1].set_title('Simple plot with sin(x**2)**2')
fig.suptitle('Stacked subplots in two direction')
plt.show()
```

# Cont...

- ❑ `fig, ax = plt.subplots(nrows=2, ncols=2)`
- ❑ `ax[0, 0].plot(x, y1, c='red')`
- ❑ `ax[0, 1].plot(x, y2, c='red')`
- ❑ `ax[1, 0].plot(x, y3, c='blue')`
- ❑ `ax[1, 1].plot(x, y3, c='blue')`
- ❑ `ax[0, 0].set_title('Simple plot with sin(x)')`
- ❑ `ax[0, 1].set_title('Simple plot with sin(x**2)')`
- ❑ `ax[1, 0].set_title('Simple plot with sin(x)**2')`
- ❑ `ax[1, 1].set_title('Simple plot with sin(x**2)**2')`
- ❑ `fig.suptitle('Stacked subplots in two direction')`
- ❑ `plt.show()`





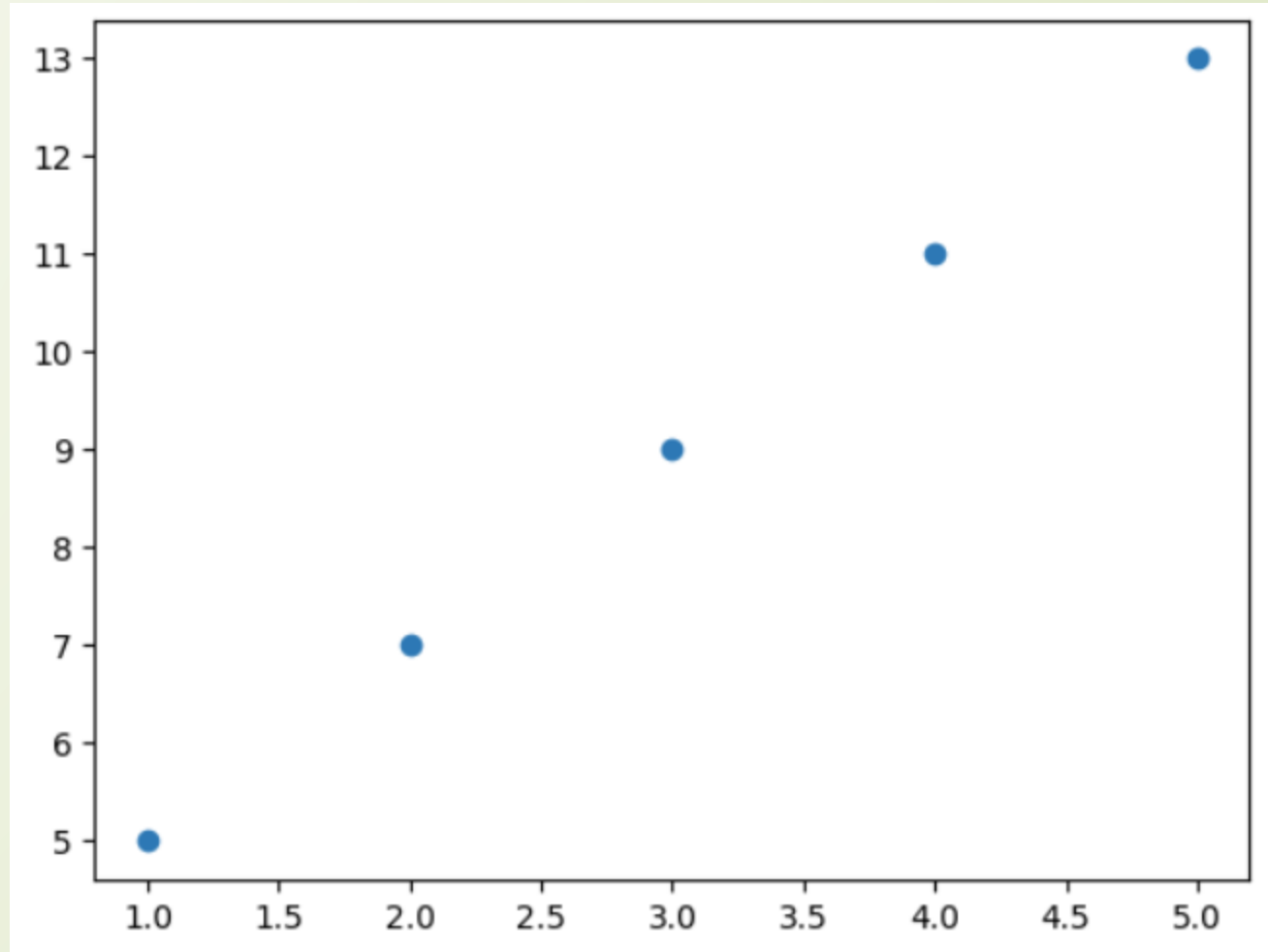
# Five common types of data visualization (and when to use them)

## ❑ Scatterplots

- ❑ Scatterplots (or scatter graphs) visualize the relationship between two variables. One variable is shown on the x-axis, and the other on the y-axis, with each data point depicted as a single “dot” or item on the graph. This creates a “scatter” effect, hence the name.
- ❑ Scatterplots are best used for large datasets when there’s no temporal element. For example, if you wanted to visualize the relationship between a person’s height and weight, or between how many carats a diamond measures and its monetary value, you could easily visualize this using a scatterplot.
- ❑ It’s important to bear in mind that scatterplots simply describe the correlation between two variables; they don’t infer any kind of cause-and-effect relationship.

# Using scatter() for Scatter Plots

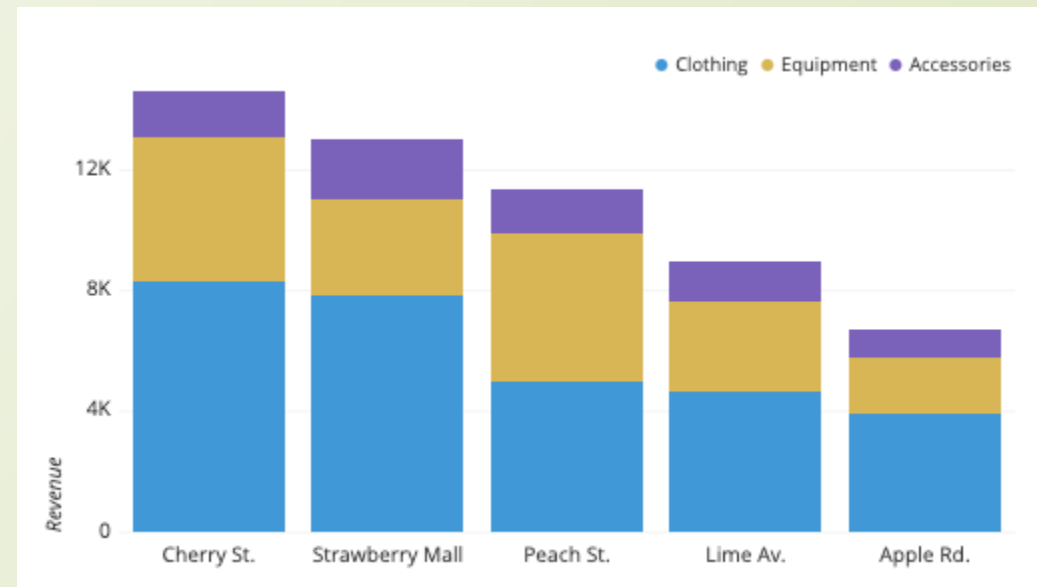
- ❑ `import matplotlib.pyplot as plt`
- ❑ `x = [1, 2, 3, 4, 5]`
- ❑ `y = [5, 7, 9, 11, 13]`
- ❑ `plt.scatter(x, y)`
- ❑ `plt.show()`





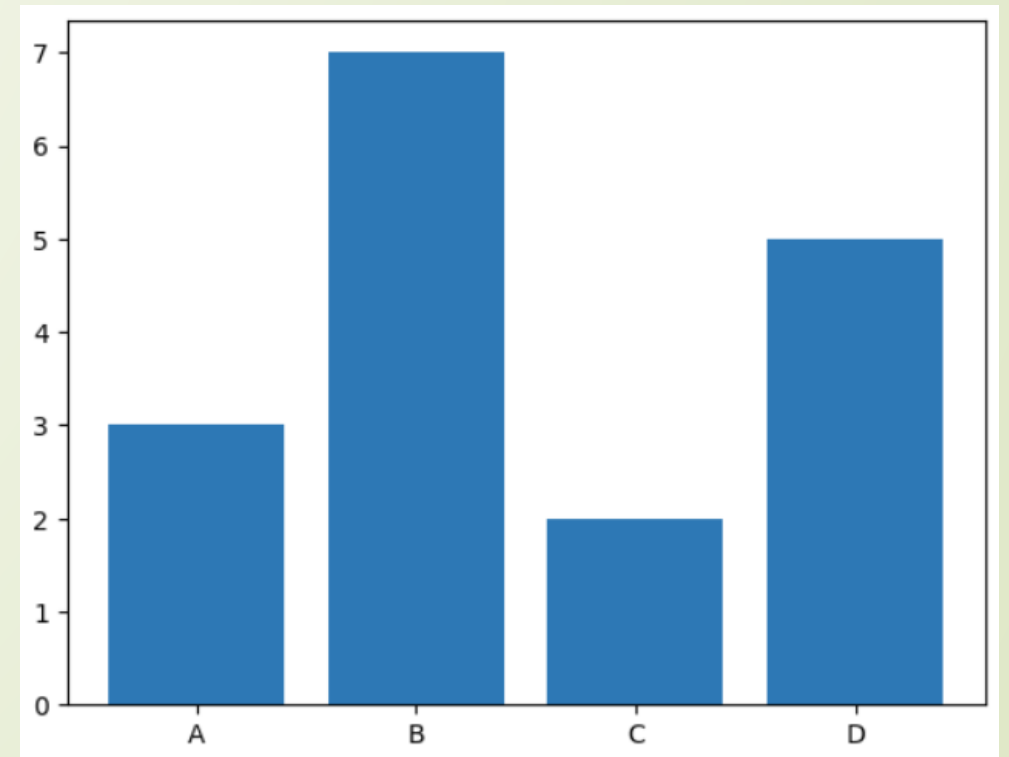
# Bar charts

- ❑ Bar charts are used to plot categorical data against discrete values. Categorical data refers to data that is not numeric, and it's often used to describe certain traits or characteristics.
- ❑ Some examples of categorical data include things like education level (e.g. high school, undergrad, or post-grad) and age group (e.g. under 30, under 40, under 50, or 50 and over).
- ❑ Discrete values are those which can only take on certain values
- ❑ So, with a bar chart, you have your categorical data on the x-axis plotted against your discrete values on the y-axis. The height of the bars is directly proportional to the values they represent, making it easy to compare your data at a glance.



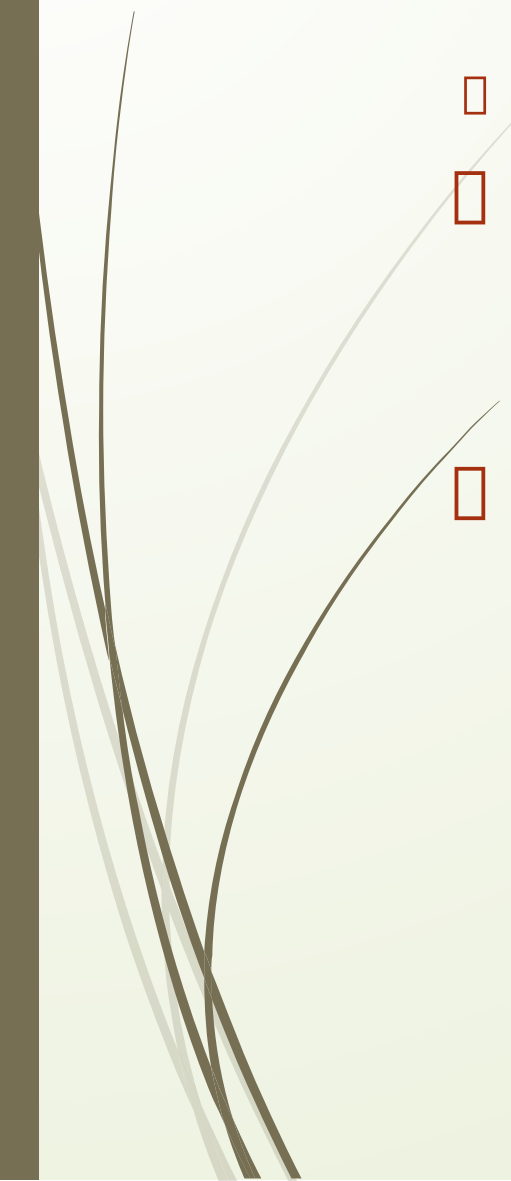
# Plotting Bar Charts Using bar()

- ❑ Bar charts are useful for comparing quantities. Use the bar() method to create a vertical or horizontal bar chart by specifying the x and y values.
- ❑ `import matplotlib.pyplot as plt`
- ❑ `categories = ['A', 'B', 'C', 'D']`
- ❑ `values = [3, 7, 2, 5]`
- ❑ `plt.bar(categories, values)`
- ❑ `plt.show()`



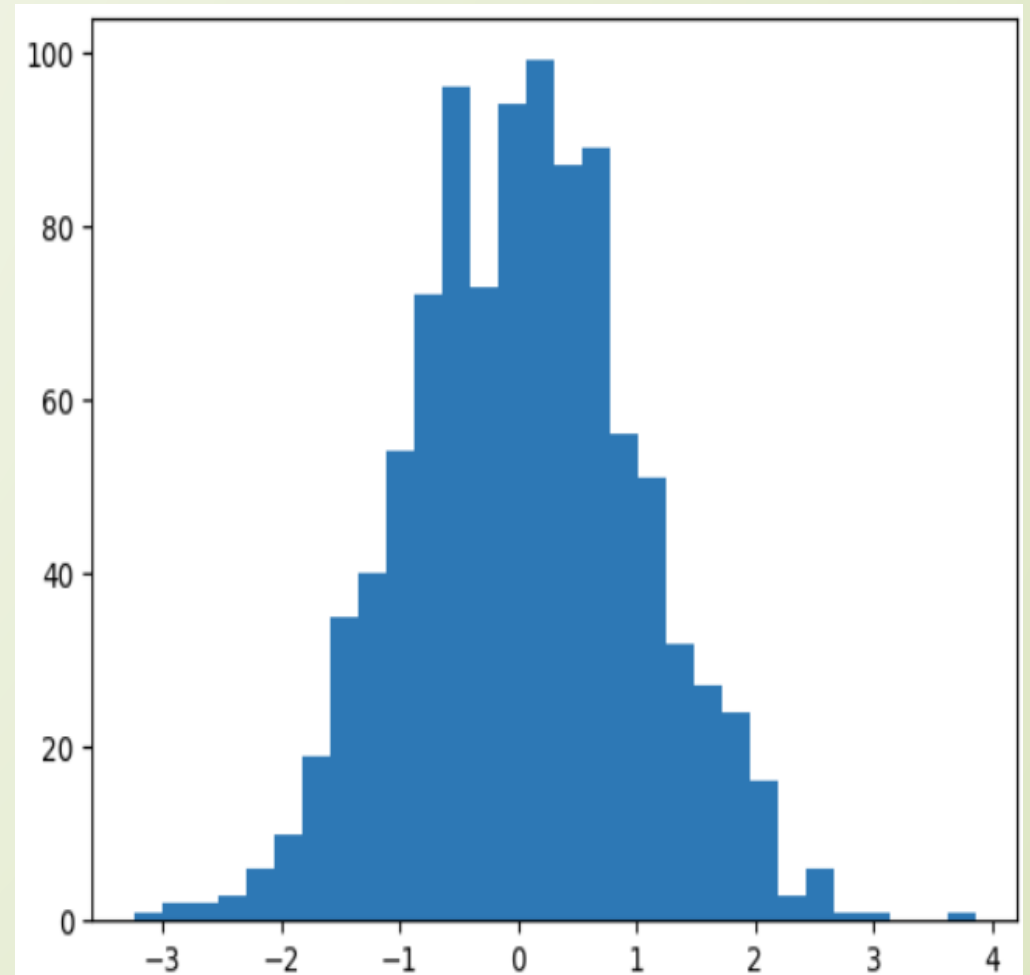


# Histogram Plot

- A histogram is used to display frequency distributions in a bar graph.
  - In this example, we'll combine matplotlib's histogram and subplot capabilities by creating a plot containing five bar graphs.
  - The areas in the bar graph will be proportional to the frequency of a random variable, and the widths of each bar graph will be equal to the class interval.
- 

# Working with Histograms Using hist()

- Histograms are useful for visualizing the distribution of data.
- `import matplotlib.pyplot as plt`
- `import numpy as np`
- `data = np.random.randn(1000)`
- `plt.hist(data, bins=30)`
- `plt.show()`



# Pie chart

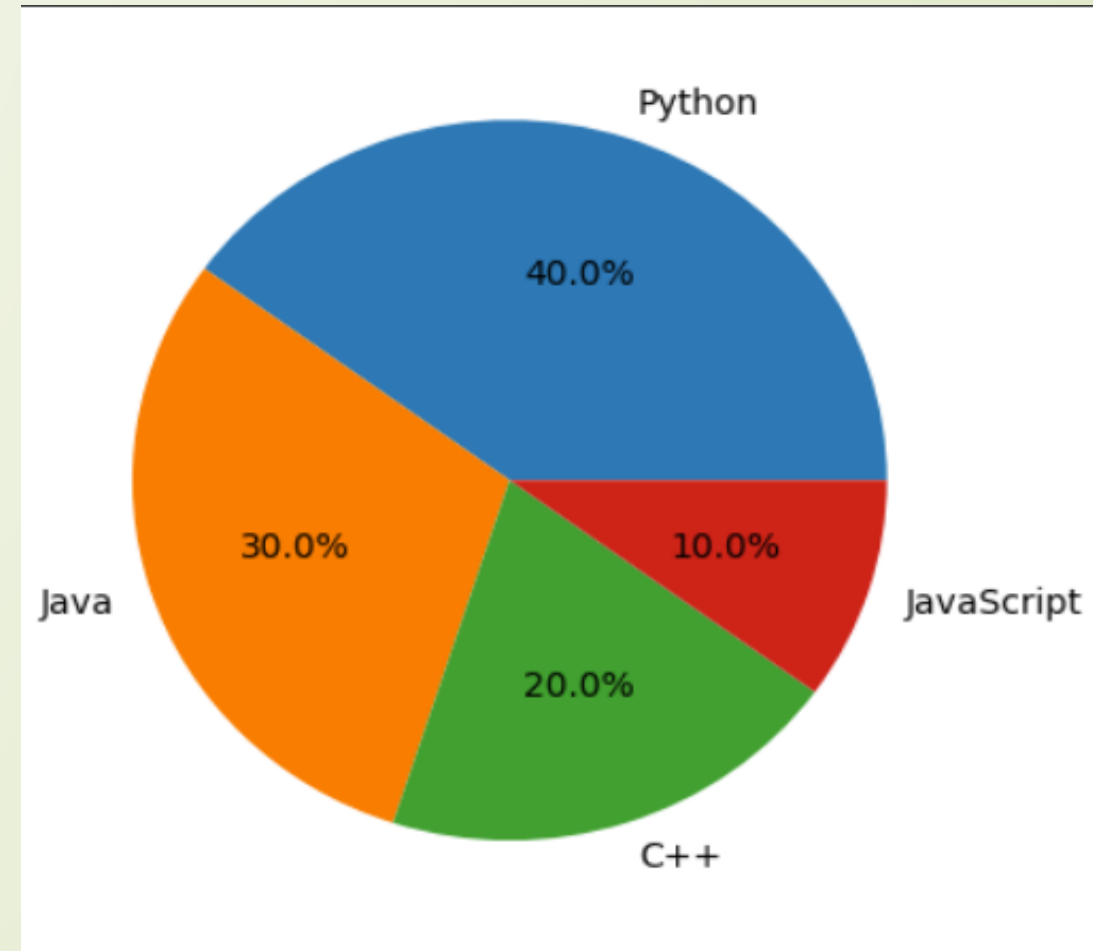
- ❑ A **Pie Chart** is a circular statistical plot that can display only one series of data. The area of the chart is the total percentage of the given data.
- ❑ The area of slices of the pie represents the percentage of the parts of the data. The slices of pie are called wedges. The area of the wedge is determined by the length of the arc of the wedge.
- ❑ The area of a wedge represents the relative percentage of that part with respect to whole data. Pie charts are commonly used in business presentations like sales, operations, survey results, resources, etc as they provide a quick summary.

# Creating Pie Chart

- ❑ Matplotlib API has `pie()` function in its `pyplot` module which create a pie chart representing the data in an array.
- ❑ **Syntax:** `matplotlib.pyplot.pie(data, explode=None, labels=None, colors=None, autopct=None, shadow=False)`
- ❑ **Parameters:**
  - data** represents the array of data values to be plotted, the fractional area of each slice is represented by **`data/sum(data)`**. If `sum(data)<1`, then the data values returns the fractional area directly.
  - labels** is a list of sequence of strings which sets the label of each wedge.
  - color** attribute is used to provide color to the wedges.
  - autopct** is a string used to label the wedge with their numerical value.
  - shadow** is used to create shadow of wedge.

# Using pie() for Pie Charts

- ❑ `import matplotlib.pyplot as plt`
- ❑ `labels = ['Python', 'Java', 'C++', 'JavaScript']`
- ❑ `sizes = [40, 30, 20, 10]`
- ❑ `plt.pie(sizes, labels=labels, autopct='%1.1f%%')`
- ❑ `plt.show()`





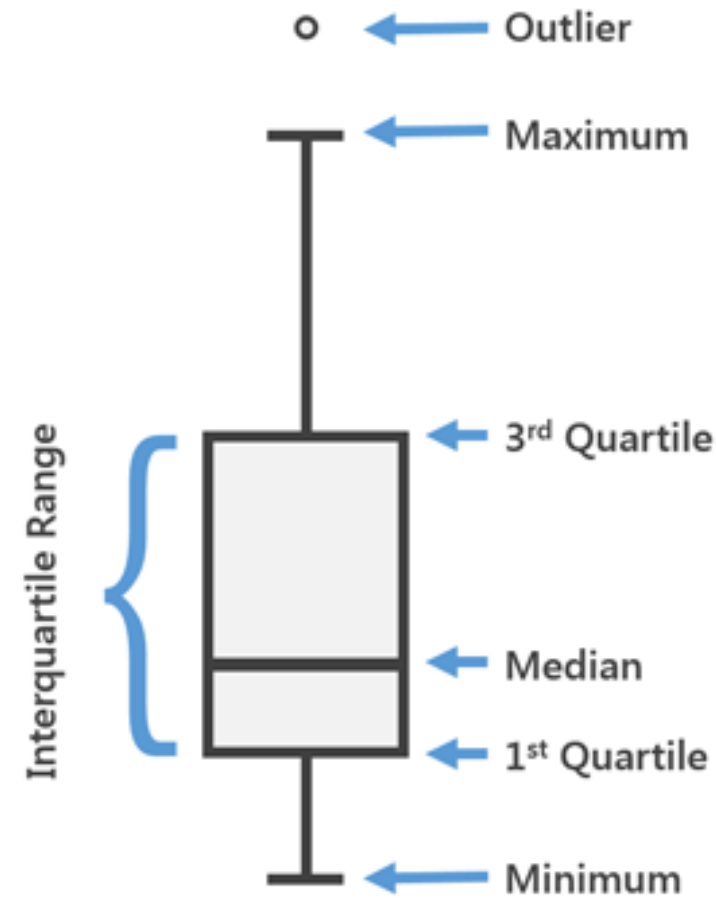


# Boxplot

- ❑ Boxplot is available in the **Seaborn** library.
- ❑ Here x is considered as the dependent variable and y is considered as the independent variable. These box plots come under **univariate analysis**, which means that we are exploring data only with one variable.
- ❑ Box plot is a graphical representation of the distribution of a dataset. It displays key summary statistics such as the median, quartiles, and potential outliers in a concise and visual manner.

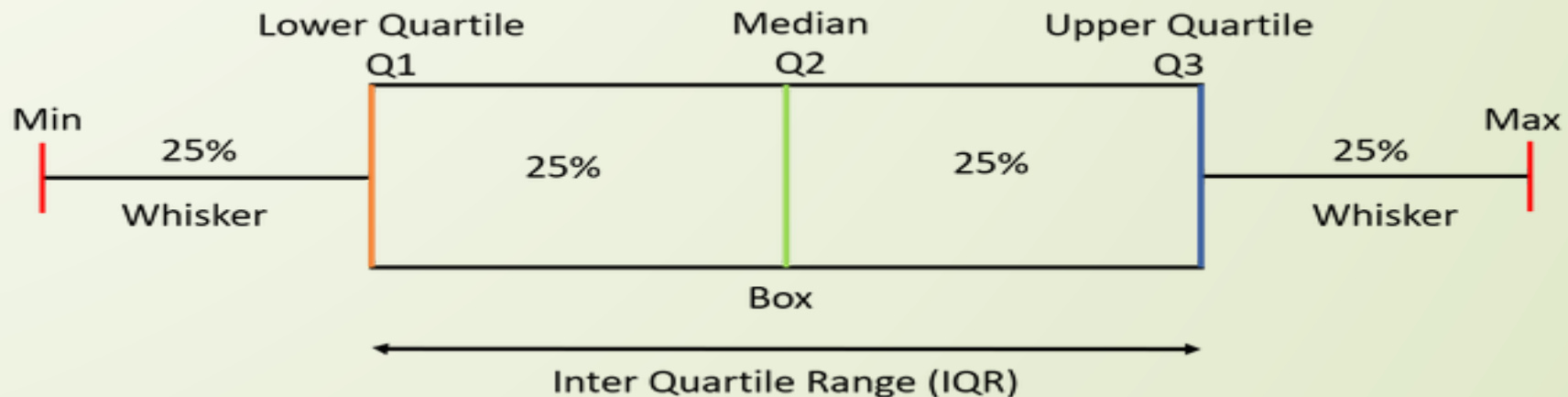
# Box and Whisker Plot

- This plot can be used to obtain more **statistical details** about the data.
- The straight lines at the maximum and minimum are also called **whiskers**.
- Points that lie outside the whiskers will be considered as an outlier.
- The box plot also gives us a description of the **25th, 50th, 75th quartiles**.
- With the help of a box plot, we can also determine the **Interquartile range(IQR)** where maximum details of the data will be present. Therefore, it can also give us a clear idea about the outliers in the dataset.



# Elements of Box Plot

- ❑ A box plot gives a five-number summary of a set of data which is-
- ❑ Minimum – It is the minimum value in the dataset excluding the outliers.
- ❑ First Quartile (Q1) – 25% of the data lies below the First (lower) Quartile.
- ❑ Median (Q2) – It is the mid-point of the dataset. Half of the values lie below it and half above.
- ❑ Third Quartile (Q3) – 75% of the data lies below the Third (Upper) Quartile.
- ❑ Maximum – It is the maximum value in the dataset excluding the outliers.



# Cont..

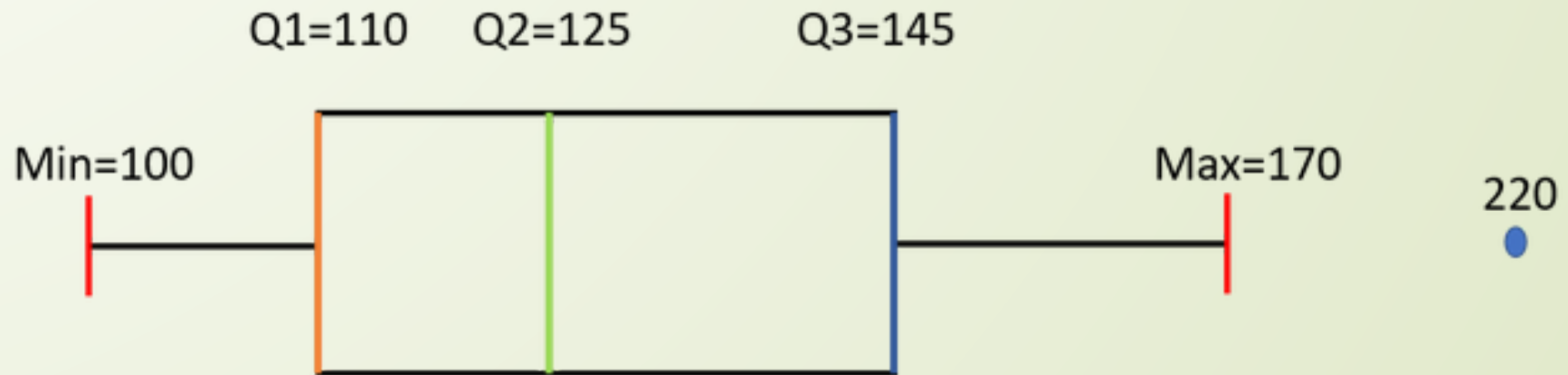
- ❑ The area inside the box (50% of the data) is known as the Inter Quartile Range. The IQR is calculated as –
- ❑  $IQR = Q3 - Q1$
- ❑ Outliers are the data points below and above the lower and upper limit. The lower and upper limit is calculated as –
- ❑ Lower Limit =  $Q1 - 1.5 * IQR$
- ❑ Upper Limit =  $Q3 + 1.5 * IQR$
- ❑ The values below and above these limits are considered outliers and the minimum and maximum values are calculated from the points which lie under the lower and upper limit.

# How to create a box plots?

- Here are the runs scored by a cricket team in a league of 12 matches – 100, 120, 110, 150, 110, 140, 130, 170, 120, 220, 140, 110.
- To draw a box plot for the given data first we need to arrange the data in ascending order and then find the minimum, first quartile, median, third quartile and the maximum.
- Ascending Order**
- 100, 110, 110, 110, 120, 120, 130, 140, 140, 150, 170, 220
- Median (Q2)** =  $(120+130)/2 = 125$ ; Since there were even values
- To find the First Quartile we take the first six values and find their median.
- Q1** =  $(110+110)/2 = 110$
- For the Third Quartile, we take the next six and find their median.
- Q3** =  $(140+150)/2 = 145$
- Note: If the total number of values is odd then we exclude the Median while calculating Q1 and Q3. Here since there were two central values we included them. Now, we need to calculate the Inter Quartile Range.
- IQR = Q3-Q1 = 145-110 = 35**
- We can now calculate the Upper and Lower Limits to find the minimum and maximum values and also the outliers if any.
- Lower Limit = Q1-1.5\*IQR = 110-1.5\*35 = 57.5**
- Upper Limit = Q3+1.5\*IQR = 145+1.5\*35 = 197.5**

# Cont..

- So, the minimum and maximum between the range  $[57.5, 197.5]$  for our given data are –
- Minimum = 100
- Maximum = 170
- The outliers which are outside this range are –
- Outliers = 220



# Bubble Chart

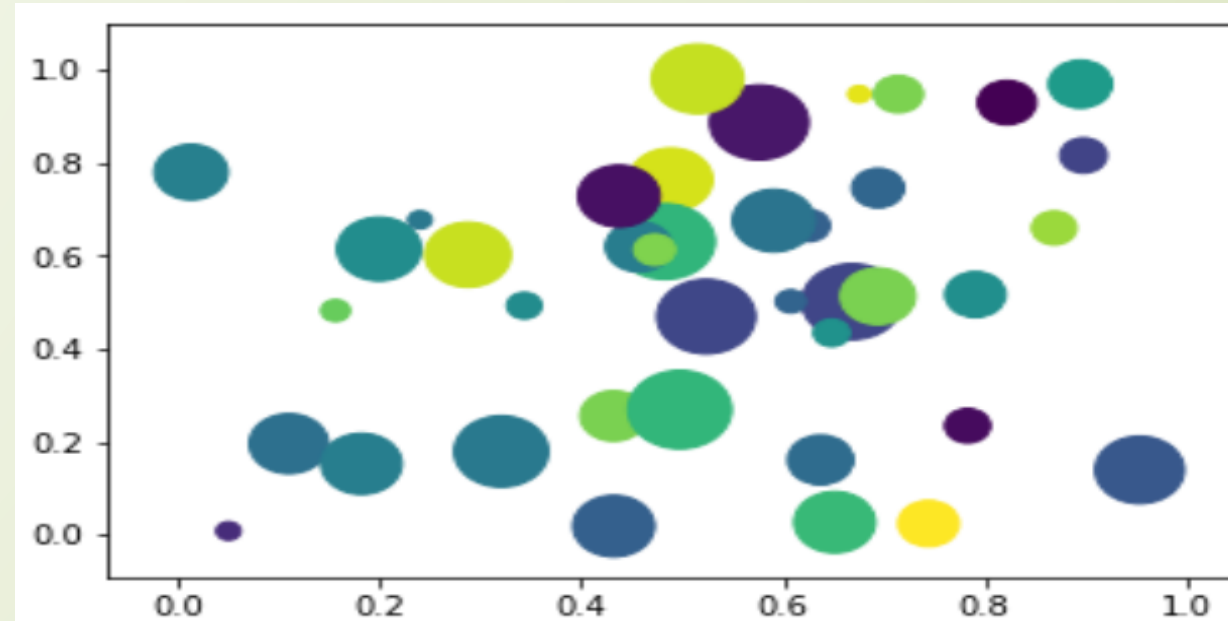
- ❑ The bubble chart in Plotly is created using the scatter plot. It can be created using the `scatter()` method of `plotly.express`.
- ❑ We use bubble charts to examine more than one variable together (**multivariate**).
- ❑ Bubble charts display data as a cluster of circles. The required data to create bubble chart needs

to have the xy coordinates, size of the bubble and the colour of the bubbles. The colours can be supplied by the library itself.

## Scaling the Size of Bubble Charts

To scale the bubble size, use the parameter `sizeref`. To calculate the values of `sizeref` use:

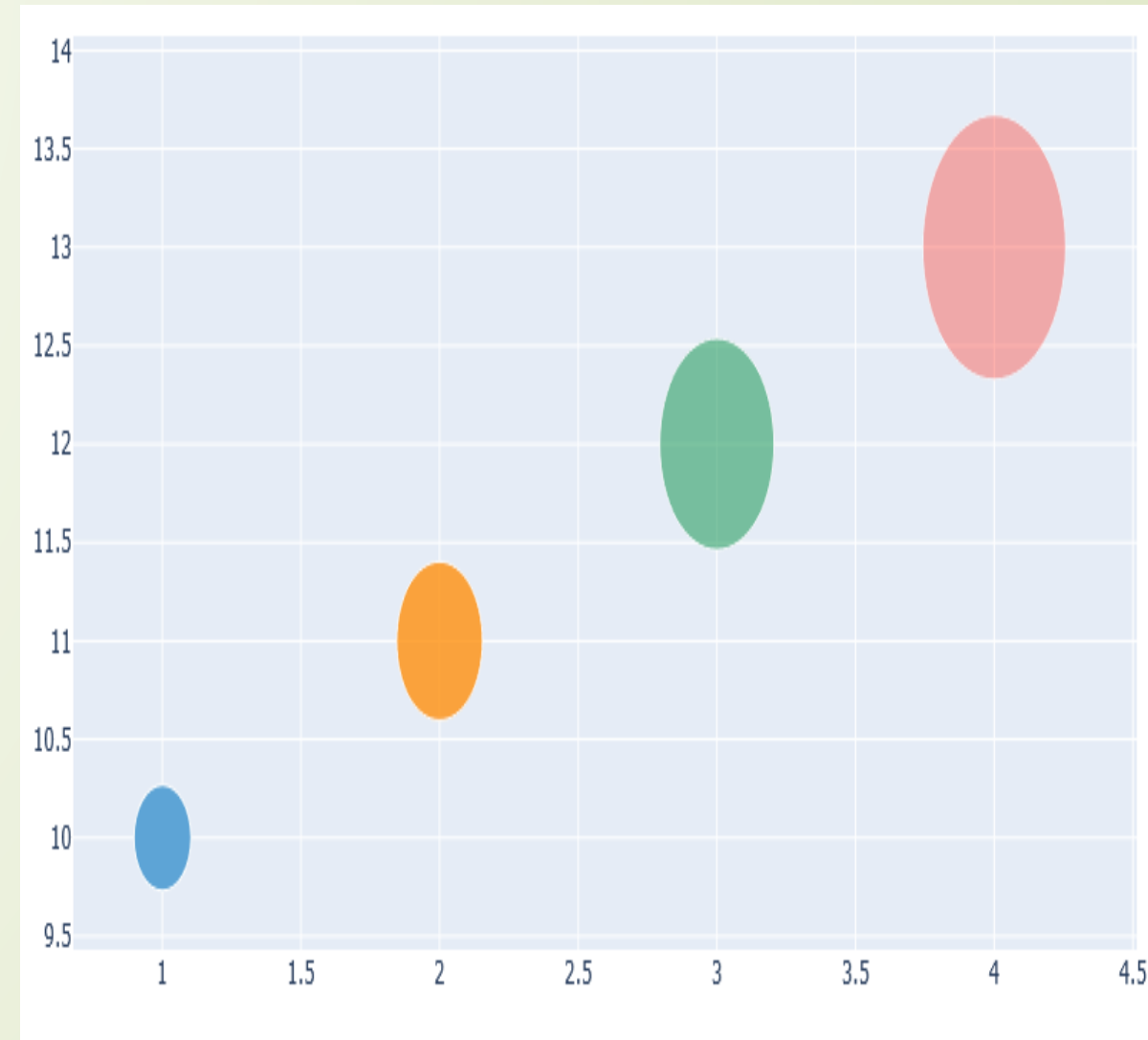
$$\text{sizeref} = 2. * \max(\text{array of size values}) / (\text{desired maximum marker size} ** 2).$$





# Simple Bubble chart

```
import plotly.graph_objects as go
fig = go.Figure(data=[go.Scatter(
    x=[1, 2, 3, 4], y=[10, 11, 12, 13],
    mode='markers',
    marker=dict(
        color=['rgb(93, 164, 214)', 'rgb(255, 144, 14)',
        'rgb(44, 160, 101)', 'rgb(255, 65, 54)'],
        opacity=[1, 0.8, 0.6, 0.4],
        size=[40, 60, 80, 100],
    )
)])
fig.show()
```

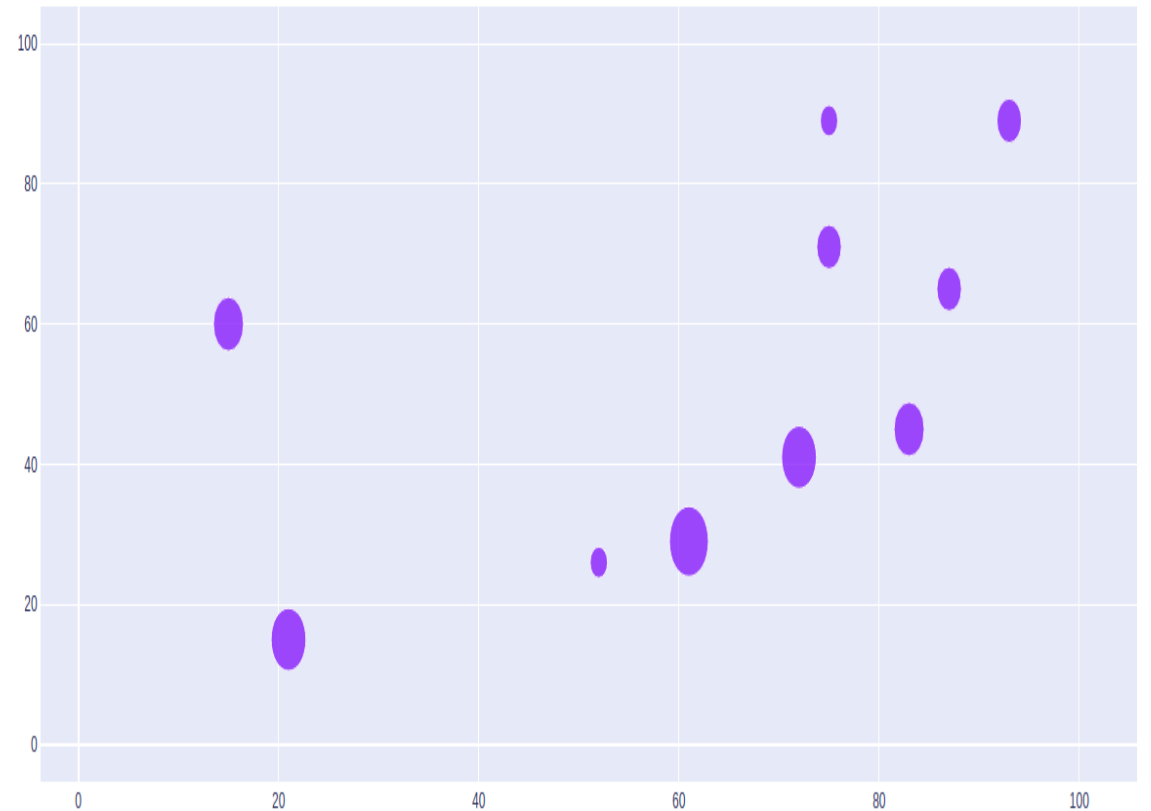


```
import plotly.graph_objects as px
import numpy as np
# creating random data through randint
# function of numpy.random
np.random.seed(42)

random_x= np.random.randint(1,101,100)
random_y= np.random.randint(1,101,100)

size = [20, 40, 60, 80, 100, 80, 60, 40, 20, 40]

plot = px.Figure(data=[px.Scatter(
    x = random_x,
    y = random_y,
    mode = 'markers',
    marker=dict(
        size=size,
        sizemode='area',
        sizeref=2.*max(size)/(40.**2),
        sizemin=4
    )
)])
plot.show()
```





# Area Chart



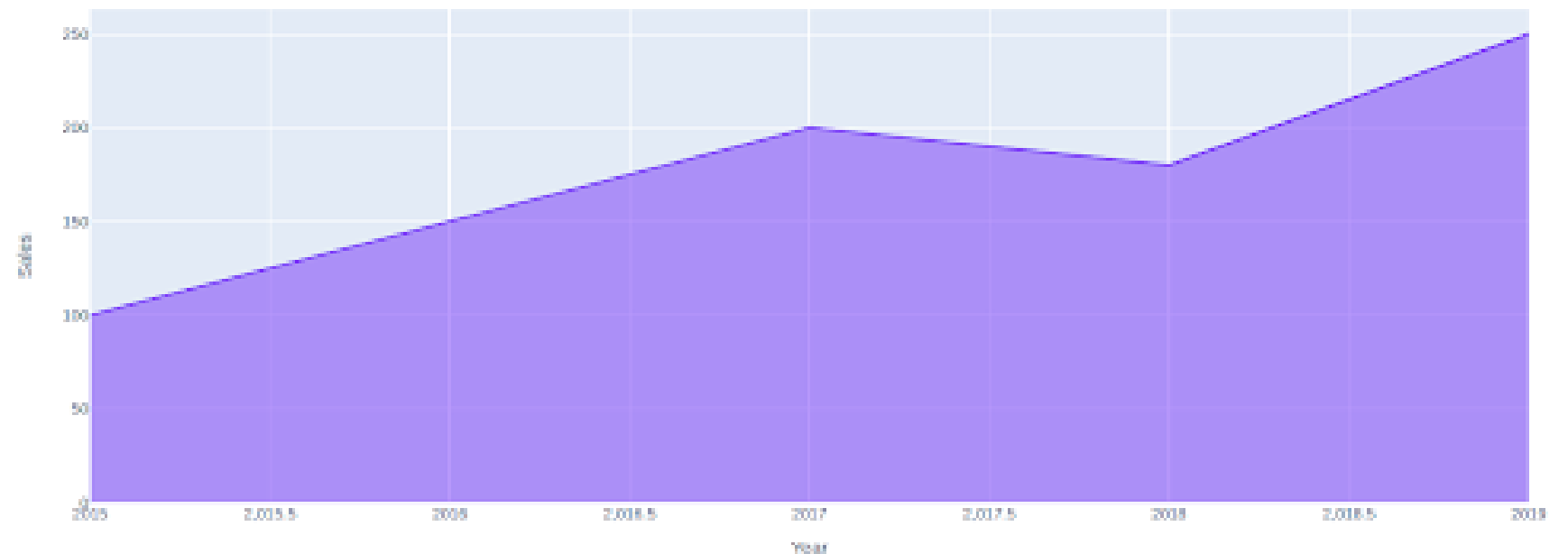
- An area chart is a type of data visualization that displays quantitative data over time or categories.
- It is similar to a line chart, but the area between the line and the x-axis is filled with color, visually representing the data's magnitude.
- Area charts are commonly used to show multiple variables' cumulative totals or compare the proportions of different categories.

# Area Plots

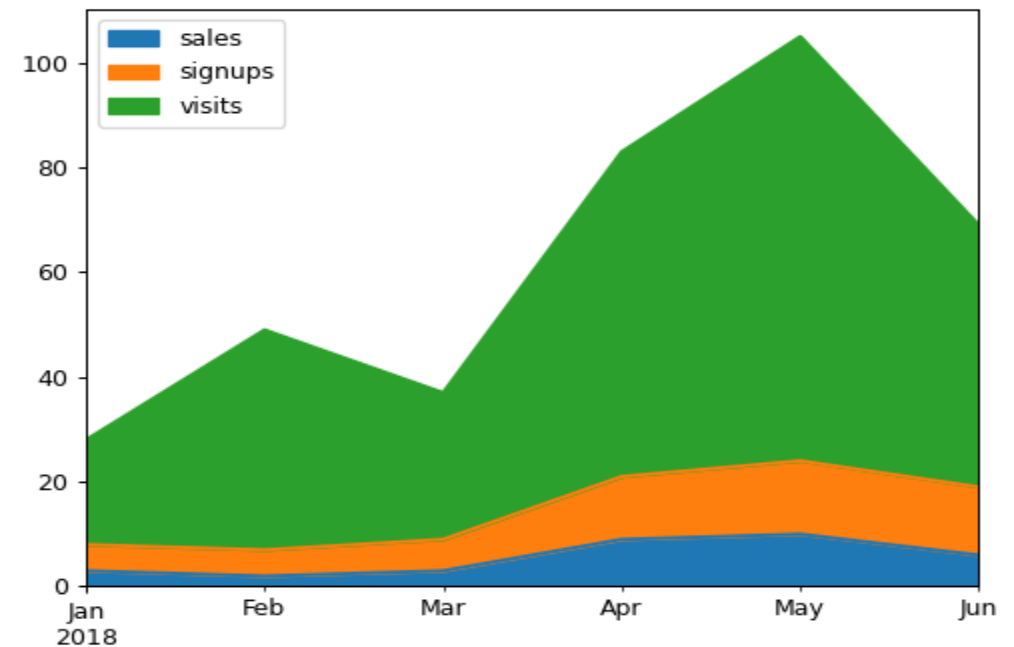
- ❑ An Area Plot is an extension of a Line Chart.
- ❑ An Area Plot is obtained by filling the region between the Line Chart and the axes with a color. When more than one Area Plot is shown in the same graph, each area plot is filled with a different color.
- ❑ Due to the color-fill effect of an area plot, the quantum and the trend of the variable is distinctly visible without making much effort.
- ❑ **Types of Area Plots:**
- ❑ **Overlapping Area plots:** In this scheme, multiple area plots are displayed in a graph with their areas overlapping. Drawing overlapping area plots require the color-fill effect to be transparent.
- ❑ **Stacked Area plots:** Multiple area plots stacked one on top of another or one below another.
- ❑ **Percentage based area plot:** An area plot drawn to plot variables with a maximum value of 100. Percentage based area plots can be drawn either with a stacked or with an overlapped scheme.

# Simple area chart

- ❑ `import pandas as pd`
- ❑ `import plotly.express as px`
- ❑ `data = {'Year': [2015, 2016, 2017, 2018, 2019],`
- ❑ `'Sales': [100, 150, 200, 180, 250]}`
- ❑ `df = pd.DataFrame(data)`
- ❑ `fig = px.area(df, x='Year', y='Sales')`
- ❑ `fig.show()`



```
>>> df = pd.DataFrame({  
...     'sales': [3, 2, 3, 9, 10, 6],  
...     'signups': [5, 5, 6, 12, 14, 13],  
...     'visits': [20, 42, 28, 62, 81, 50],  
... },  
index=pd.date_range(start='2018/01/01',  
end='2018/07/01', freq='ME'))  
  
>>> ax = df.plot.area()
```





# Customizing Stacked Area Charts

- Plotly provides a wide range of customization options for stacked area charts. We can customize the colors, labels, axes, and other visual elements to make the chart more visually appealing and informative.
- To customize the colors of the areas, we can use the ``color_discrete_sequence`` parameter in the ``px.area()`` function. This allows us to specify a list of colors for each category.



# Example stacked area chart

```
import pandas as pd
import plotly.express as px

# Create a dataframe with random values
data = pd.DataFrame({
    'Year': [2015, 2016, 2017, 2018, 2019],
    'Category A': [10, 20, 30, 40, 50],
    'Category B': [20, 30, 40, 50, 60],
    'Category C': [30, 40, 50, 60, 70]
})

# Create a stacked area chart
fig = px.area(data, x='Year', y=['Category A', 'Category B', 'Category C'],
              title='Stacked Area Chart',
              color_discrete_sequence=['#000000', '#FFFF00', '#800000'])
fig.show()
```

