Module 5- A Deep Dive into Matplotlib

Module 5 Syllabus	**Introduction,** Overview of Plots in Matplotlib**, Pyplot Basics:** Creating Figures, Closing Figures, Format Strings, Plotting, Plotting Using pandas DataFrames, Displaying Figures, Saving Figures**; Basic Text and Legend Functions:** Labels, Titles, Text, Annotations, Legends**; Basic Plots:**Bar Chart, Pie Chart, Stacked Bar Chart, Stacked Area Chart, Histogram, Box Plot, Scatter Plot, Bubble Plot**; Layouts:** Subplots, Tight Layout, Radar Charts, GridSpec**; Images:** Basic Image Operations, Writing Mathematical Expressions **Textbook 2: Chapter 3**

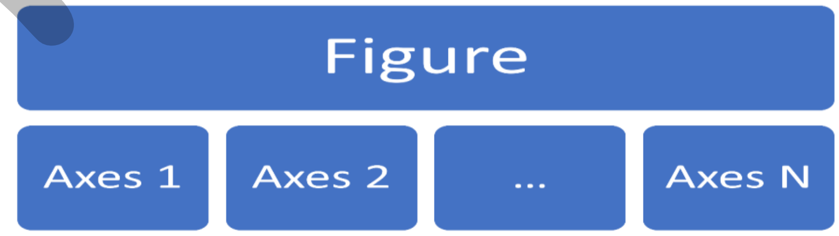
Handouts for Session 34: Introduction, Overview of Plots in Matplotlib

5.1 Introduction

- **Matplotlib** is **used for data science and machine learning visualizations**
- **John Hunter** an **American neurobiologist** began developing **Matplotlib** in **2003**
- It aimed to **emulate** the commands of the **MATLAB** software, which was the scientific standard back then.
- Several features, such as the global style of MATLAB, were introduced into Matplotlib to make the transition to Matplotlib easier for MATLAB users

5.2 Overview of Plots in Matplotlib

- Plots in Matplotlib have a hierarchical structure, Nests Python objects to create a tree-like structure.
- Each plot is encapsulated in a Figure object.
- This Figure is the top-level container of the visualization.
- Figure can have multiple axes, which are basically individual plots inside this top-level container.
- There are other Python objects can be customized - control axes, tick marks, legends, titles, text boxes, the grid etc.



A Figure contains at least one axes object

The two main components of a plot are as follows:

1. **Figure**
   - ✓ It is the outermost container that allows you to draw multiple plots within it.
   - ✓ It also allows to configure the Title.

2. Axes

- ✓ The axes are an actual plot, or subplot, depending on whether you want to plot single or multiple visualizations.
- ✓ Its sub-objects include the x-axis, y-axis, spines, and legends.

- Hierarchical structure allows to create a complex and customizable visualization.
- Matplotlib gives us the ability not only to display data, but also to design the whole Figure around it by adjusting the Grid, X and Y ticks, tick labels, and the Legend.

Components of Figure Object:

1. **Spines**: Lines connecting the axis tick marks
2. **Title**: Text label of the whole Figure object
3. **Legend**: Describes the content of the plot
4. **Grid**: Vertical and horizontal lines used as an extension of the tick marks
5. **X/Y axis label**: Text labels for the X and Y axes below the spines
6. **Minor tick**: Small value indicators between the major tick marks
7. **Minor tick label**: Text label that will be displayed at the minor ticks
8. **Major tick**: Major value indicators on the spines
9. **Major tick label**: Text label that will be displayed at the major ticks
10. **Line**: Plotting type that connects data points with a line
11. **Markers**: Plotting type that plots every data point with a defined marker

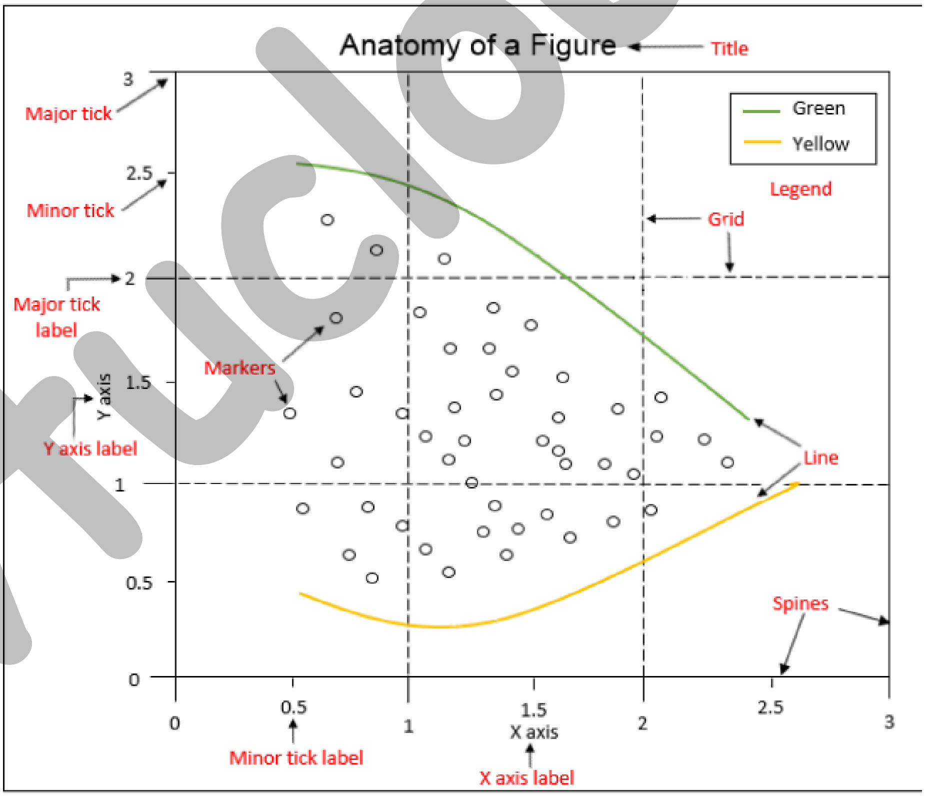


Figure 3.2: Anatomy of a Matplotlib Figure

Review Questions

1. What is Matplotlib?

2. What is a Figure?

3. What are the two main components of a Plot?

4. What are the Components of Figure Object?

Handouts for Session 35: Pyplot Basics: Creating Figures, Closing Figures, Format Strings, Plotting

5.3 pyplot basics

- **pyplot** provides **a simpler interface for creating visualizations** that allow the users to plot the data without explicitly configuring the Figure and Axes themselves.
- It is handy to use the alias **plt** to reference the imported submodule, as follows

import matplotlib.pyplot as plt

1. Creating Figures

- plt.figure() - creates a new Figure.
- This function returns a Figure instance, but it is also passed to the backend.
- Every Figure-related command that follows is applied to the current Figure and does not need to know the Figure instance.
- By default figure has a,

  width of 6.4 inches

  height of 4.8 inches

  dpi (dots per inch) - 100

- **To change the default values** of the Figure, use the **parameters figsize** and **dpi**.

Example:

```
#To change the width and the height
plt.figure(figsize=(10, 5))
#To change the dpi
plt.figure(dpi=300)
```

2. Closing Figures

- Figures that are no longer used should be closed by explicitly calling **plt.close(),** which also cleans up memory efficiently.
- If nothing is specified, the **plt.close()** command will **close the current Figure**.
- **To close a specific Figure**, you can either **provide a reference to a Figure instance** or **provide the Figure number**.
- To find the number of a Figure object, use the number attribute, as follows:

  plt.gcf().number

- **plt.close('all') -** command **closes all active Figures.**

The following example shows how a Figure can be created and closed.

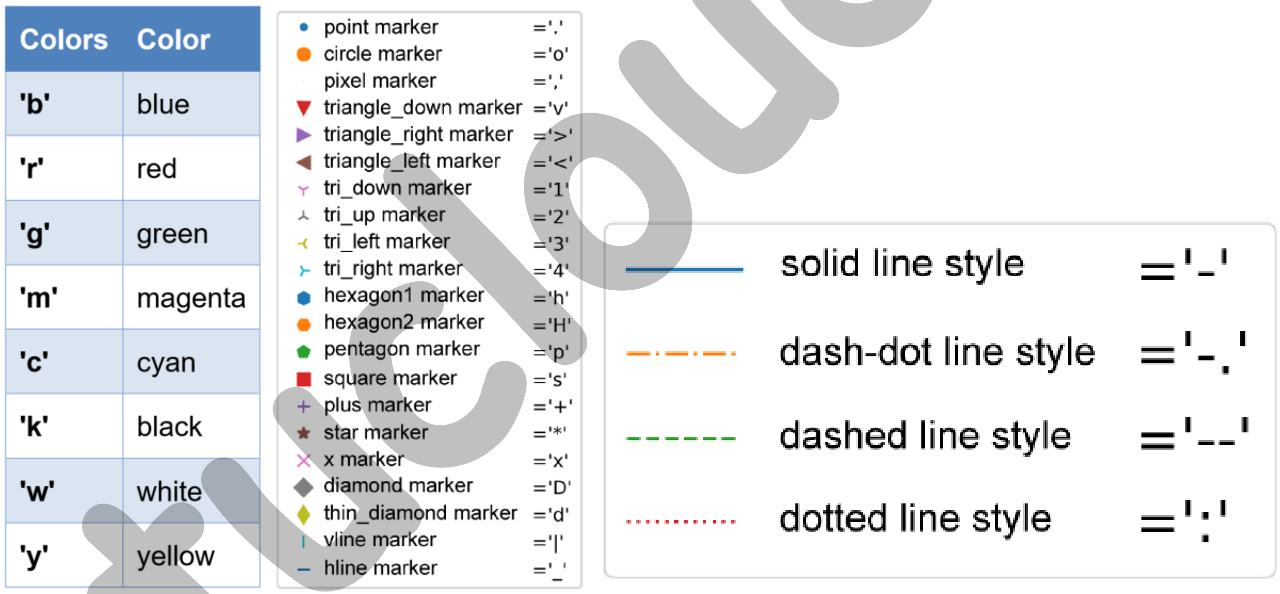
#Create Figure with Figure number 10
plt.figure(num=10)
#Close Figure with Figure number 10
plt.close(10)

3. Format Strings

- They are a neat way to specify **colors**, **marker types**, and **line styles**.
- A format string is specified as **[color][marker][line]**, where each item is optional.
- If the **color** argument is the only argument of the format string, you can use **matplotlib.colors**.
- Matplotlib recognizes the following formats, among others:

  RGB or RGBA float tuples (for example, (0.2, 0.4, 0.3) or (0.2, 0.4, 0.3, 0.5))
  RGB or RGBA hex strings (for example, '#0F0F0F' or '#0F0F0F0F')

The following table is an example of how a color, marker options and line style can be represented in one particular format:



Colors	Color
'b'	blue
'r'	red
'g'	green
'm'	magenta
'c'	cyan
'k'	black
'w'	white
'y'	yellow

●	point marker	='.'
●	circle marker	='o'
	pixel marker	=','
▼	triangle_down marker	='v'
▶	triangle_right marker	='>'
◀	triangle_left marker	='<'
⋎	tri_down marker	='1'
⋏	tri_up marker	='2'
≺	tri_left marker	='3'
≻	tri_right marker	='4'
⬢	hexagon1 marker	='h'
⬣	hexagon2 marker	='H'
⬟	pentagon marker	='p'
■	square marker	='s'
+	plus marker	='+'
★	star marker	='*'
×	x marker	='x'
◆	diamond marker	='D'
♦	thin_diamond marker	='d'
|	vline marker	='|'
–	hline marker	='_'

——	solid line style	='-'
-·-·-	dash-dot line style	='-.'
------	dashed line style	='--'
········	dotted line style	=':'

Color, Marker Options and Line styles specified in string format

4. Plotting

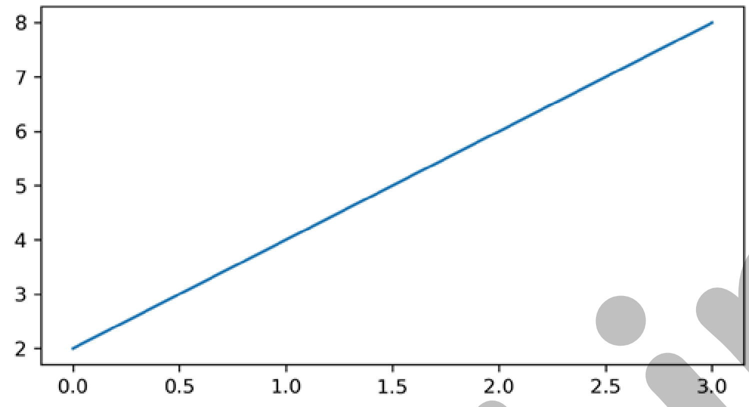
- To plot **data points as lines and/or markers** syntax for plot is,

  plt.plot([x], y, [fmt])

  where [ ] indicates that they are optional
- The function **returns a list of Line2D objects representing the plotted data.**
- By default, if you do not provide a format string (fmt), the data points will be connected with straight, solid lines.

Example:
plt.plot([0, 1, 2, 3], [2, 4, 6,8])

produces a plot, as shown in the following diagram.
Since x is optional and the default values are [0, …, N-1],
plt.plot([2, 4, 6, 8]) results in the same plot:

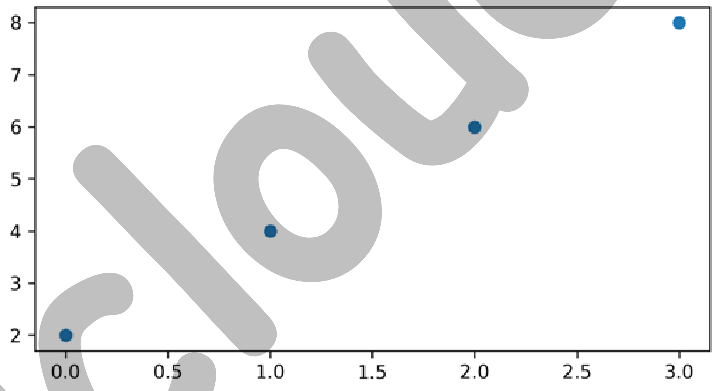


Plotting data points as lines

- To plot markers instead of lines, specify a format string with any marker type.

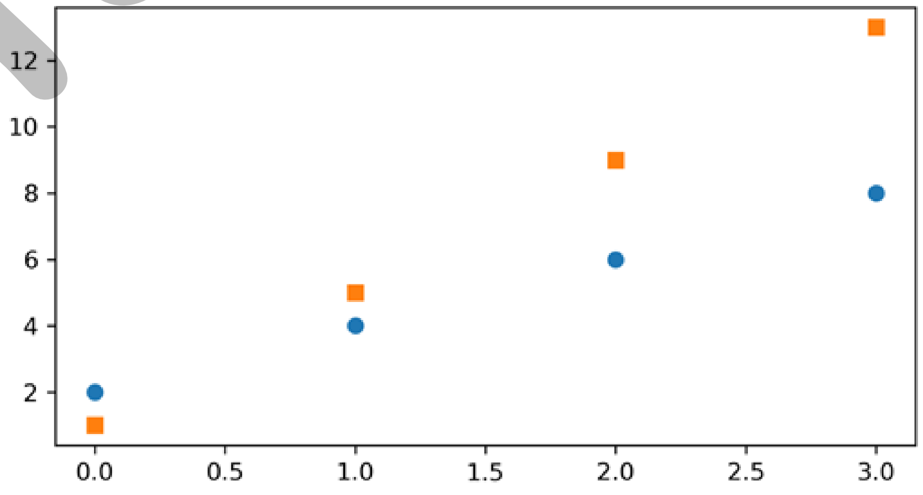
Example,

plt.plot([0, 1, 2, 3], [2, 4, 6, 8], 'o') displays data points as circles, as shown in the following diagram:



Plotting data points with markers (circles)

- To plot multiple data pairs, the syntax **plt.plot([x], y, [fmt], [x], y2, [fmt2], …)**
  Example:
- **plt.plot([2, 4, 6, 8], 'o', [1, 5, 9,13], 's')** results in the following diagram.



Plotting data points with multiple markers

Review Questions

1. How to create a new figure?

2. What is the default height, width and dpi of a Figure?

3. Why a figures has to be closed?

4. What is the way to specify colors, marker types, and line styles?

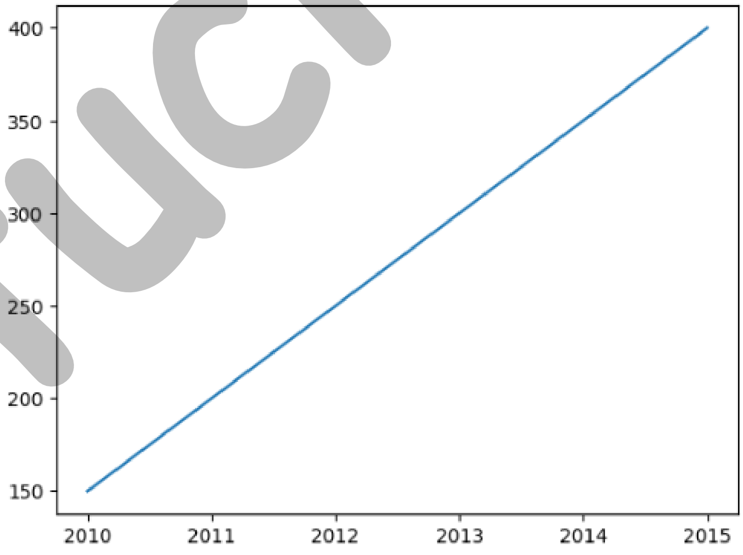
Handouts for Session 36: Plotting Using pandas DataFrames

5. Plotting Using pandas DataFrames

- Instead of providing x and y values, **pandas.DataFrame** can be provided the in the **data** parameter and give keys for x and y, as follows:

plt.plot('x_key', 'y_key', data=df)

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

data = {
  'Year': [2010, 2011, 2012, 2013, 2014, 2015],
  'Sales': [150, 200, 250, 300, 350, 400]
}

df = pd.DataFrame(data)
plt.plot('Year', 'Sales',data=df)
```



Plot using DataFrame

Ticks

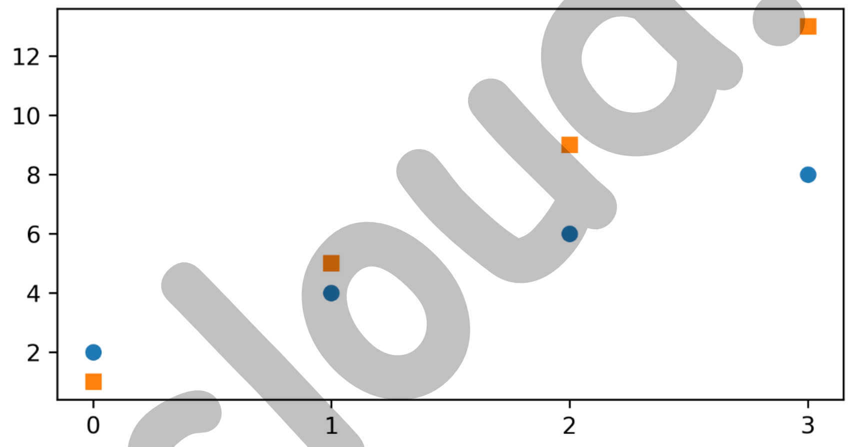
- Tick locations and labels can be set manually if Matplotlib's default isn't sufficient.

- Considering the previous plot, it might be preferable to only have ticks at multiples of ones at the x-axis. One way to accomplish this is to use plt.xticks() and plt.yticks() to either get or set the ticks manually.
- plt.xticks(ticks, [labels], [**kwargs]) sets the current tick locations and labels of the x-axis.

  Parameters:
  - **ticks**: **List of tick locations**; if an **empty list is passed, ticks will be disabled.**
  - **labels** (**optional**): You can optionally pass a list of labels for the specified locations.
  - ****kwargs (optional): matplotlib.text.Text()** properties can be used to customize the **appearance of the tick labels**. A quite useful property is rotation; this allows you to rotate the tick labels to use space more efficiently.

Example 1:

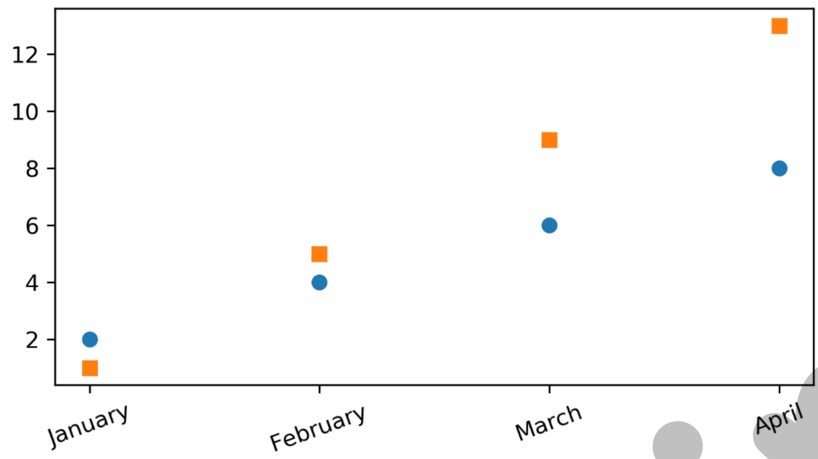
```
plt.figure(figsize=(6, 3))
plt.plot([2, 4, 6, 8], 'o', [1, 5, 9, 13], 's')
plt.xticks(ticks=np.arange(4))
```



Plot with custom ticks

Example 2:

```
plt.figure(figsize=(6, 3))
plt.plot([2, 4, 6, 8], 'o', [1, 5, 9, 13], 's')
plt.xticks(ticks=np.arange(4),
labels=['January', 'February', 'March', 'April'],rotation=20)
```



Plot with custom tick labels

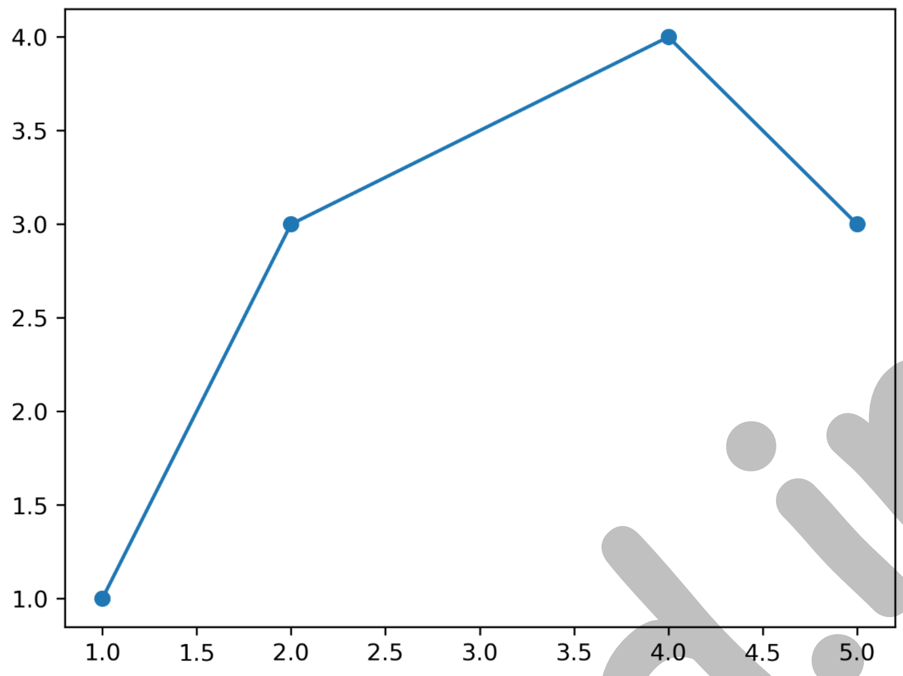
6. Displaying Figures

- **plt.show()** is **used to display a Figure or multiple Figures**.
- To display Figures within a Jupyter Notebook, simply set the **%matplotlib inline** command at the beginning of the code.
- If you forget to use plt.show(), the plot won't show up.
  Note: In a typical script run outside of an interactive environment, you would need to call plt.show() to display the plot.

7. Saving Figures

- The **plt.savefig(fname)** saves the current Figure.
- There are some useful optional parameters you can specify, such as dpi, format, or transparent.

```
plt.figure()

plt.plot([1, 2, 4, 5], [1, 3, 4, 3], '-o')

#bbox_inches='tight' removes the outer white margins

plt.savefig('lineplot.png', dpi=300, bbox_inches='tight')
```



Saved Figure

Review Questions:

1. What is Dataframe?
2. How data frame can be passed as a parameter for plotting?

Handouts for Session 37: Displaying Figures, Saving Figures Basic Text and Legend Functions: Labels, Titles, Text, Annotations

5.4 Basic Text and Legend Functions

- All of the functions we discuss in this topic, except for the legend, create and return a matplotlib.text.Text() instance.

1. Labels

- Matplotlib provides a few label functions those are used for setting labels to the x- and y-axes.
- The **plt.xlabel()** and **plt.ylabel()** functions **sets the label for the current axes**.
- The **set_xlabel()** and **set_ylabel()** functions **sets the label for specified axes.**

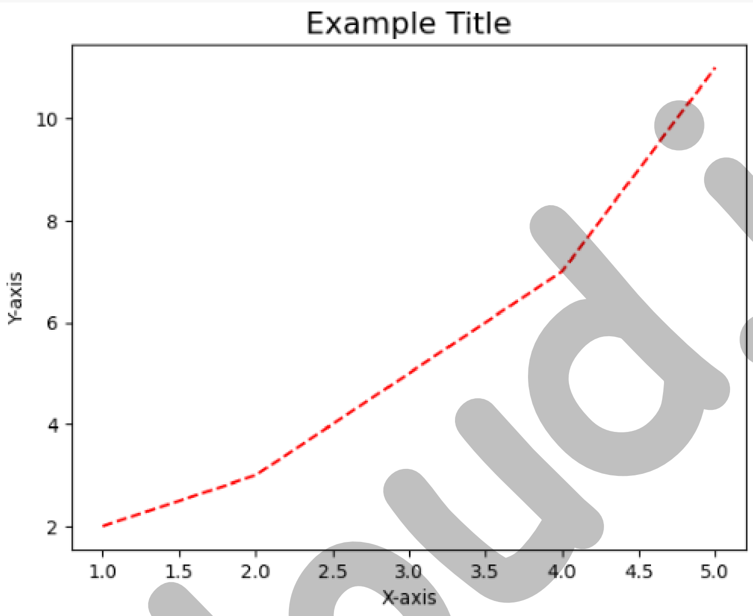
2. Titles

- A **title** describes a particular chart/graph.
- The titles are placed above the axes in the center, left edge, or right edge.
- There are two options for titles – you can either set the **Figure title** or the title of an **Axes**.
- The **suptitle()** function **sets the title for the current and specified Figure**.
- The **title()** function **sets the title for the current and specified axes**.

Example:

```
# Data points
x = [1, 2, 3, 4, 5]
```

```
y = [2, 3, 5, 7, 11]

# Plot with format string: 'r--' means red color, dashed line
plt.plot(x, y, 'r--')
plt.xlabel('X-axis') #sets label for x-axis of current plot(AXES)
plt.ylabel('Y-axis') #sets label for y-axis of current plot(AXES)

plt.title('Example Title',fontsize=16) #Title for current plot
plt.show()
```



3. Text

- There are two options for **text** – you can either add text to a Figure or text to an Axes.
- The **figtext(x, y, text)** and **text(x, y, text)** functions add text at locations **x** or **y** for a Figure.
- **Text is placed at an arbitrary position on the Axes (plot).**

Example:

```
ax.text(4, 6, 'Text in Data Coords', bbox={'facecolor': 'yellow', 'alpha':0.5, 'pad':10})
```

- This creates a yellow text box with the text **Text in Data Coords**.
- Text can be used to provide additional textual information to a visualization.

4. Annotations

- **Annotations** are used to annotate some features of the plot.
- In annotations, there are two locations to consider:
  - ✓ the annotated location, **xy**
  - ✓ text **xytext**.
- It is useful to specify the parameter **arrowprops**, which results in an arrow pointing to the annotated location.

Example:

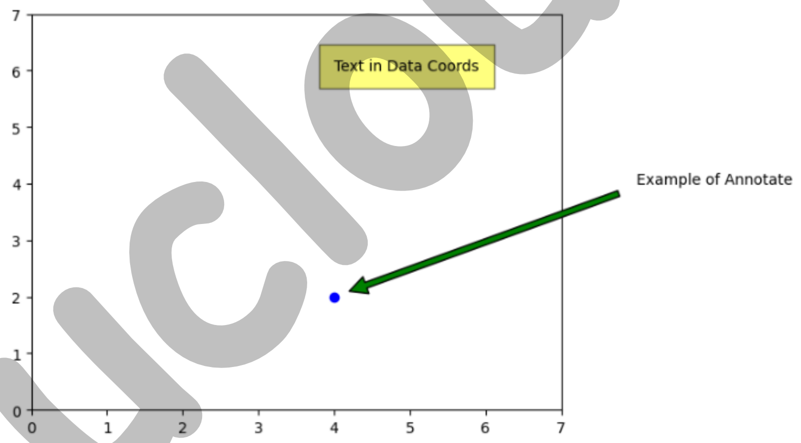
```
ax.annotate('Example of Annotate', xy=(4,2), xytext=(8,4),
```

arrowprops=dict(facecolor='green', shrink=0.05))

The above lines of code creates a green arrow pointing to the data coordinates (4, 2) with the text Example of Annotate at data coordinates (8, 4):

Example code:

```
# Create a figure and an axis
fig, ax = plt.subplots()


# Add text with a bounding box
ax.text(4, 6, 'Text in Data Coords',
      bbox={'facecolor': 'yellow', 'alpha': 0.5, 'pad': 10})
plt.plot(4, 2, 'bo')
ax.annotate('Example of Annotate', xy=(4,2), xytext=(8,4),
arrowprops=dict(facecolor='green', shrink=0.05))

# Set limits for x and y axis, needed because text is always placed # at arbitrary position
ax.set_xlim(0, 7)
ax.set_ylim(0, 7)

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

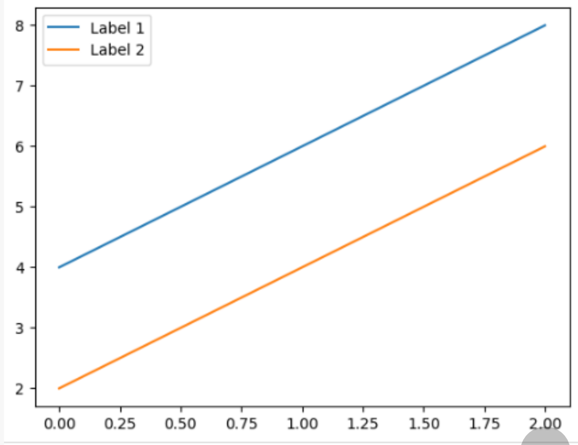


5. Legends

- Legend describes the content of the plot.
- To add a **legend** to your Axes, it is necessary to specify the **label** parameter at the time of plot creation.
- Calling **plt.legend()** for the current Axes or **Axes.legend()** for a specific Axes will add the legend.
- The **loc** parameter specifies the location of the legend.

Example:

```
plt.plot([4, 6, 8], label='Label 1')
plt.plot([2, 4, 6], label='Label 2')
plt.legend()
```



Review Questions:
1. What are basic Text functions?
2. What does Title indicate?
3. Why annotations are used?
4. How to display figures?
5. How to save the current figure?

5.5 Visualizing Stock Trends by Using a Line Plot

- Let's look at the following scenario: you are interested in investing in stocks. You downloaded the stock prices for the "big five": Amazon, Google, Apple, Facebook, and Microsoft. You want to visualize the closing prices in dollars to identify trends.
- Import the necessary modules and enable plotting within a Jupyter Notebook.
- Use pandas to read the datasets (GOOGL_data.csv, FB_data.csv, AAPL_data.csv, AMZN_data.csv, and MSFT_data.csv) located in the Datasets folder.
- The read_csv() function reads a .csv file into a DataFrame.
- Use Matplotlib to create a line chart visualizing the closing prices for the past 5 years (whole data sequence) for all five companies. Add labels, titles, and a legend to make the visualization self-explanatory. Use plt.grid() to add a grid to your plot. If necessary, adjust the ticks in order to make them readable.

Solution:

```
# Import statements
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
%matplotlib inline
# load datasets
google = pd.read_csv('GOOGL_data.csv')
facebook = pd.read_csv('FB_data.csv')
apple = pd.read_csv('AAPL_data.csv')
amazon = pd.read_csv(‘AMZN_data.csv')
```

```
microsoft = pd.read_csv('MSFT_data.csv')
# Create figure
plt.figure(figsize=(16, 8), dpi=300)
# Plot data
plt.plot('date', 'close', data=google, label='Google')
plt.plot('date', 'close', data=facebook, label='Facebook')
plt.plot('date', 'close', data=apple, label='Apple')
plt.plot('date', 'close', data=amazon, label='Amazon')
plt.plot('date', 'close', data=microsoft, label='Microsoft')
# Specify ticks for x and y axis
plt.xticks(np.arange(0, 1260, 40), rotation=70)
plt.yticks(np.arange(0, 1450, 100))
# Add title and label for y-axis
plt.title('Stock trend', fontsize=16)
plt.ylabel('Closing price in $', fontsize=14)
# Add grid
plt.grid()
# Add legend
plt.legend()
# Show plot
plt.show()
```

Output:

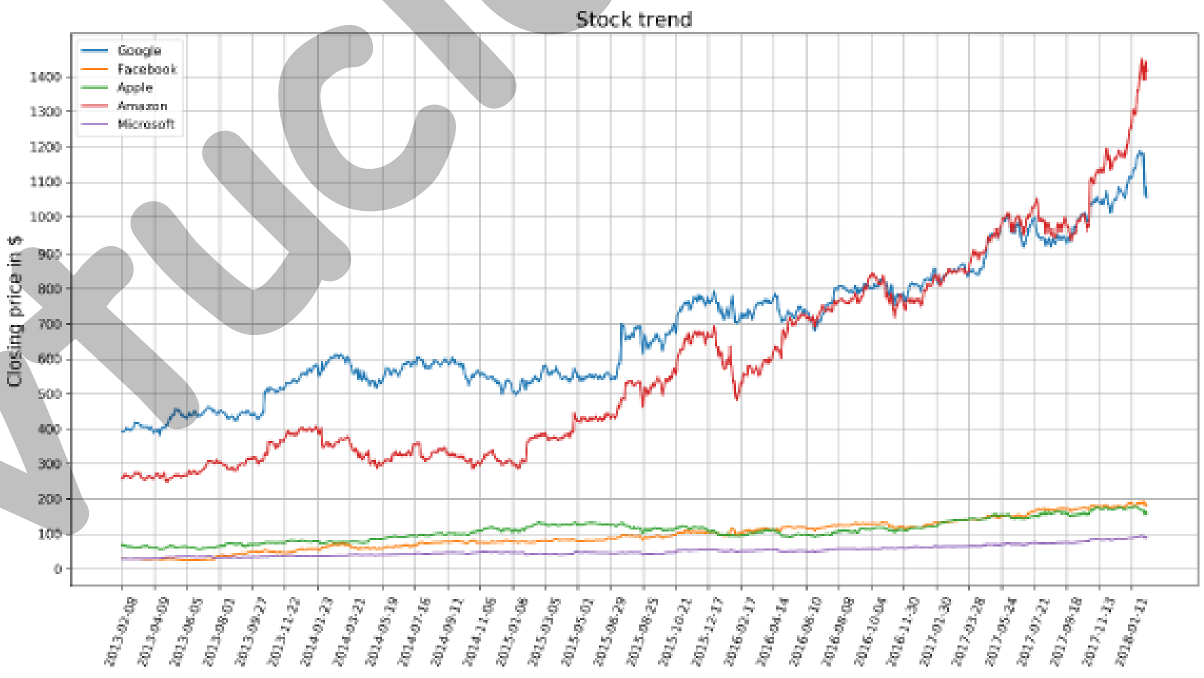


Figure 3.15: Visualization of stock trends of five companies

Handouts for Session 38: Legends, Basic Plots: Bar Chart, Pie Chart, Stacked Bar Chart

5.6 Basic Plots

In this section, we are going to go through the different types of simple plots. This includes bar charts, pie charts, stacked bar, and area charts, histograms, box plots, scatter plots and bubble plots.

5.6.1 Bar Chart

- The **plt.bar(x, height, [width])** creates a vertical bar plot. For horizontal bars, use the **plt.barh()** function.
- Important parameters:

  x: Specifies the x coordinates of the bars
  height: Specifies the height of the bars
  width (optional): Specifies the width of all bars; the default is 0.8
  Ex: plt.bar(['A', 'B', 'C', 'D'], [20, 25, 40, 10])

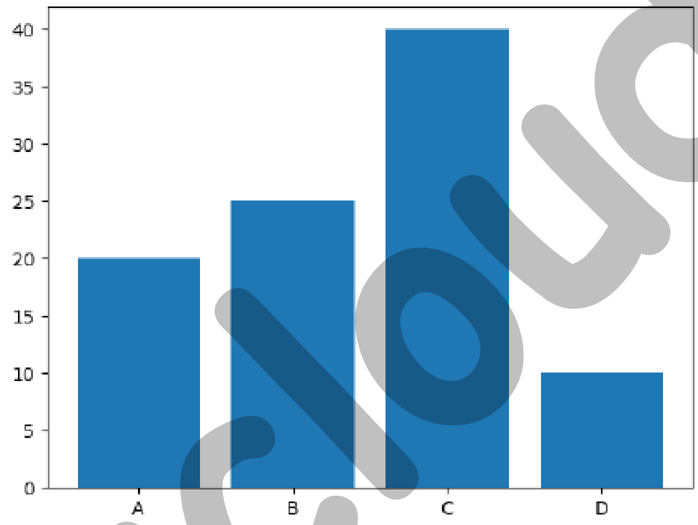


Figure 3.16: A simple bar chart

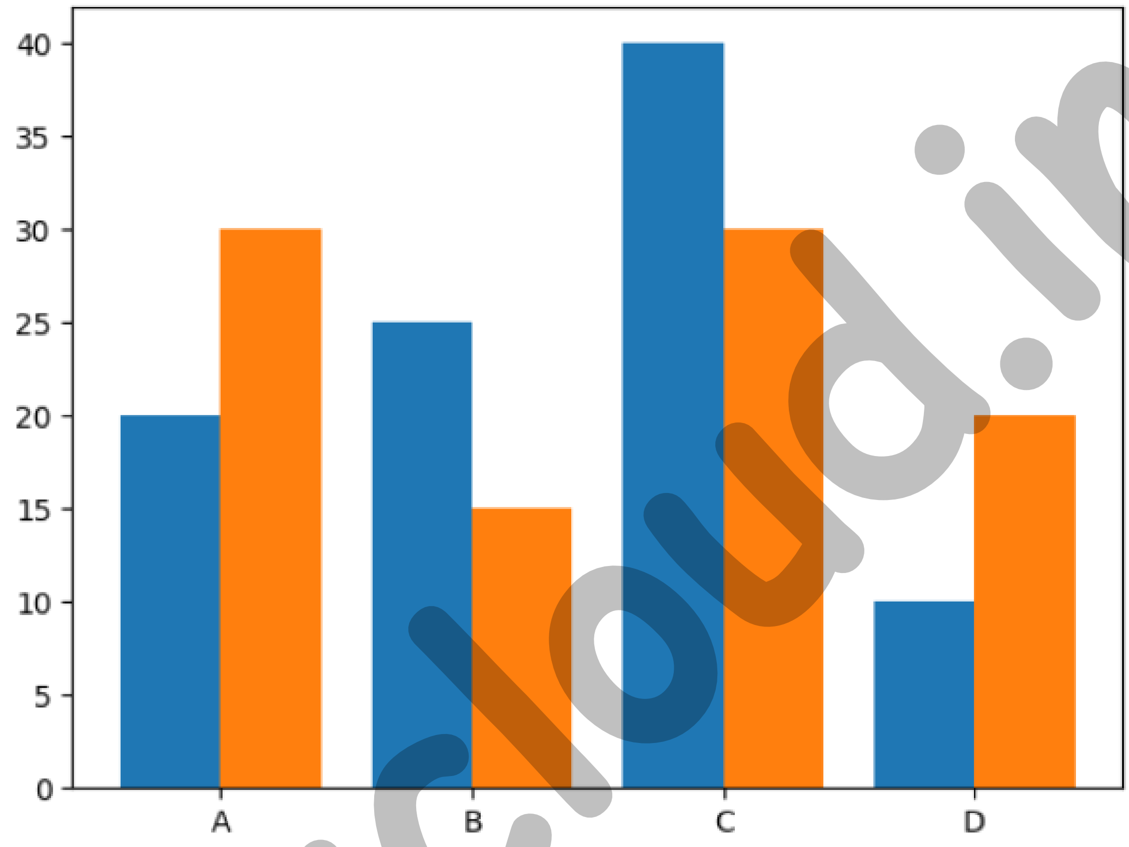
- If you want to have subcategories, you have to use the **plt.bar()** function multiple times with shifted x-coordinates.
- This is done in the following example and illustrated in the figure that follows.
- The **arange() function** is a method in the NumPy package that returns evenly spaced values within a given interval.
- The **gca() function** helps in getting the instance of current axes on any current Figure.
- The **set_xticklabels() function** is used to set the x-tick labels with the list of given string labels.

Example:

```
import matplotlib.pyplot as plt
import numpy as np
labels = ['A', 'B', 'C', 'D']
x = np.arange(len(labels))
width = 0.4
```

```
plt.bar(x - width / 2, [20, 25, 40, 10], width=width)
plt.bar(x + width / 2, [30, 15, 30, 20], width=width)
# Ticks and tick labels must be set manually
plt.xticks(x)
ax = plt.gca()
ax.set_xticklabels(labels)
```

Output:



5.6.1.1 Activity: Creating a Bar Plot for Movie Comparison

In this activity, we will create visually appealing bar plots. We will use a bar plot to compare movie scores. You are given five movies with scores from Rotten Tomatoes. The Tomatometer is the percentage of approved Tomatometer critics who have given a positive review for the movie. The Audience Score is the percentage of users who have given a score of 3.5 or higher out of 5. Compare these two scores among the five movies.

The following are the steps to perform:

1. Import the necessary modules and enable plotting within a Jupyter Notebook.
2. Use pandas to read the data located in the Datasets subfolder.
3. Use Matplotlib to create a visually appealing bar plot comparing the two scores for all five movies.
4. Use the movie titles as labels for the x-axis. Use percentages at intervals of 20 for the y-axis and minor ticks at intervals of 5. Add a legend and a suitable title to the plot.
5. Use functions that are required to explicitly specify the axes. To get the reference to the current axes, use ax = plt.gca(). To add minor y-ticks, use Axes.set_yticks([ticks], minor=True). To add

a horizontal grid for major ticks, use Axes.yaxis.grid(which='major'), and to add a dashed horizontal grid for minor ticks, use Axes.yaxis.grid(which='minor', linestyle='--').

Solution:

```
# Import statements
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
# Load dataset
movie_scores = pd.read_csv('movie_scores.csv')
# Create figure
plt.figure(figsize=(10, 5), dpi=300)
# Create bar plot
pos = np.arange(len(movie_scores['MovieTitle']))
width = 0.3
plt.bar(pos-width / 2, movie_scores['Tomatometer'], width, label='Tomatometer')
plt.bar(pos + width / 2, movie_scores['AudienceScore'],width, label='Audience
Score')
# Specify ticks
plt.xticks(pos, rotation=10)
plt.yticks(np.arange(0, 101, 20))
# Get current Axes for setting tick labels and horizontal grid
ax = plt.gca()
# Set tick labels
ax.set_xticklabels(movie_scores['MovieTitle'])
ax.set_yticklabels(['0%', '20%', '40%', '60%', '80%', '100%'])
# Add minor ticks for y-axis in the interval of 5
ax.set_yticks(np.arange(0, 100, 5), minor=True)
# Add major horizontal grid with solid lines
ax.yaxis.grid(which='major')
# Add minor horizontal grid with dashed lines
ax.yaxis.grid(which='minor', linestyle='--')
# Add title
plt.title('Movie comparison')
# Add legend
plt.legend()
# Show plot
plt.show()
```

The expected output is as follows:

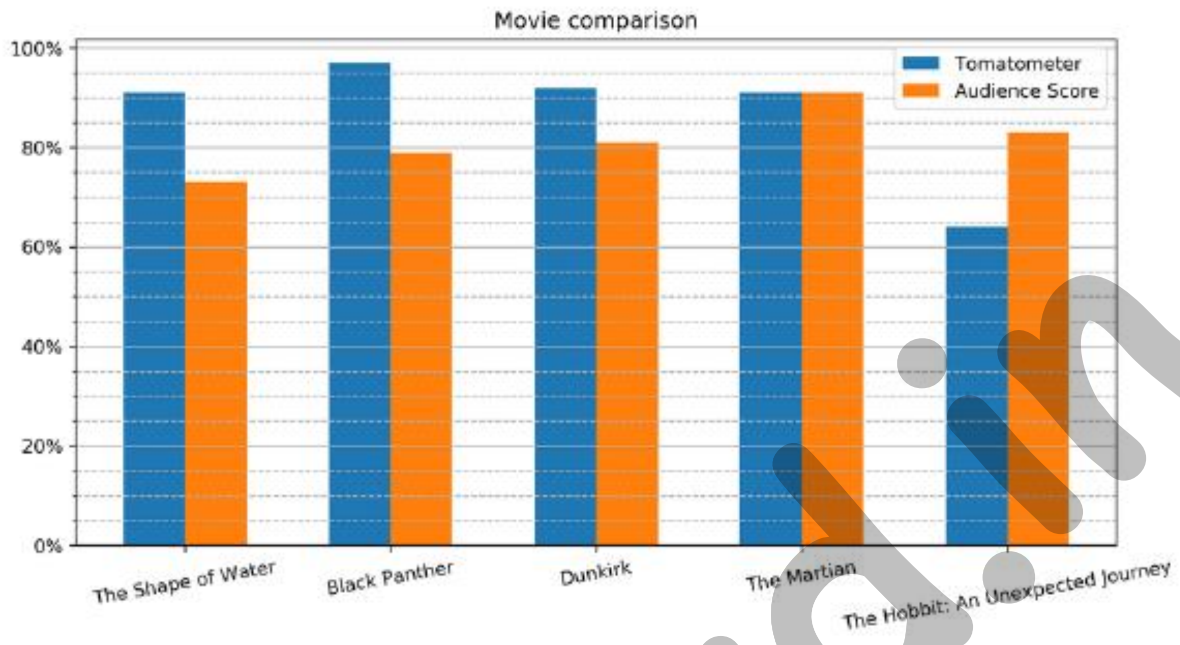


Figure 3.18: Bar plot comparing scores of five movies

5.6.2 Pie Chart

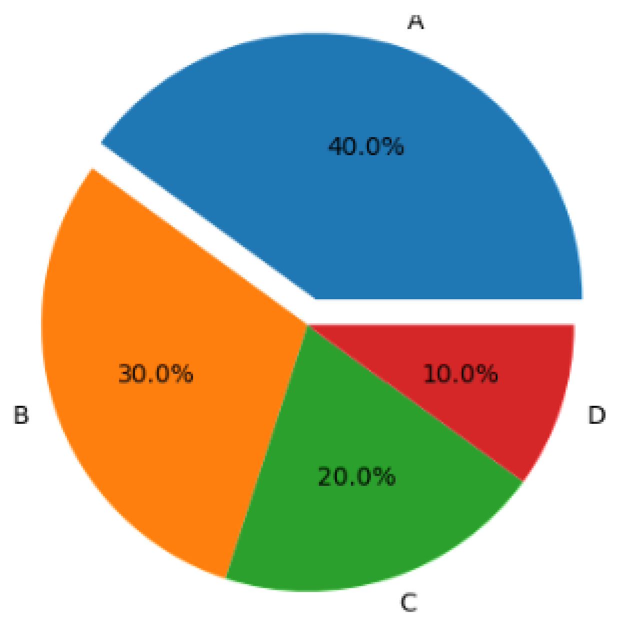
- The **plt.pie(x, [explode], [labels], [autopct])** function creates a pie chart.

Important parameters:

- **x:** Specifies the slice sizes.
- **explode (optional):** Specifies the fraction of the radius offset for each slice. The explode-array must have the same length as the x-array.
- **labels (optional):** Specifies the labels for each slice.
- Shows percentages inside the slices according to the specified format string. **autopct (optional): Example: '%1.1f%%'.**

Example:

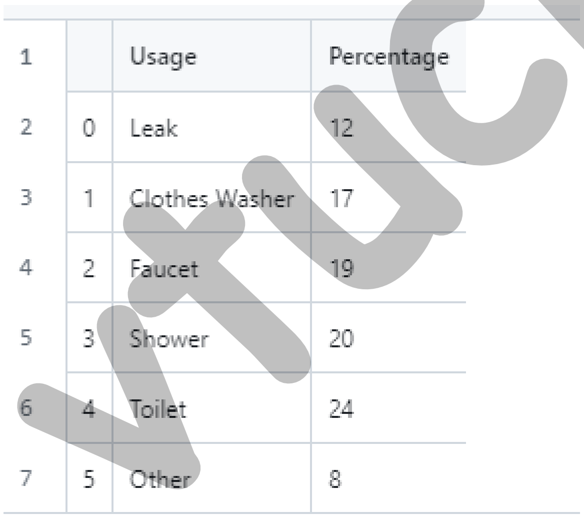
```
import matplotlib.pyplot as plt
plt.pie([0.4, 0.3, 0.2, 0.1], explode=(0.1, 0, 0, 0), labels=['A', 'B', 'C', 'D'],autopct='%1.1f%%')
```



5.6.2.1 Activity: Creating a Pie Chart for Water Usage

- In this exercise, we will use a pie chart to visualize water usage. There has been a shortage of water in your locality in the past few weeks. To understand the reason behind it, generate a visual representation of water usage using pie charts. Use pandas to read the data. Use a pie chart to visualize water usage. Highlight one usage of your choice using the explode parameter. Show the percentages for each slice and add a title

Dataset:



1		Usage	Percentage
2	0	Leak	12
3	1	Clothes Washer	17
4	2	Faucet	19
5	3	Shower	20
6	4	Toilet	24
7	5	Other	8

```
# Import statements

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline
```

```
# Load dataset

data = pd.read_csv('water_usage.csv’)

# Create figure

plt.figure(figsize=(8, 8), dpi=300)

# Create pie plot

plt.pie('Percentage', explode=(0, 0, 0.1, 0, 0, 0), labels='Usage', data=data,
autopct='%.0f%%')

# Add title

plt.title('Water usage')

# Show plot

plt.show()
```

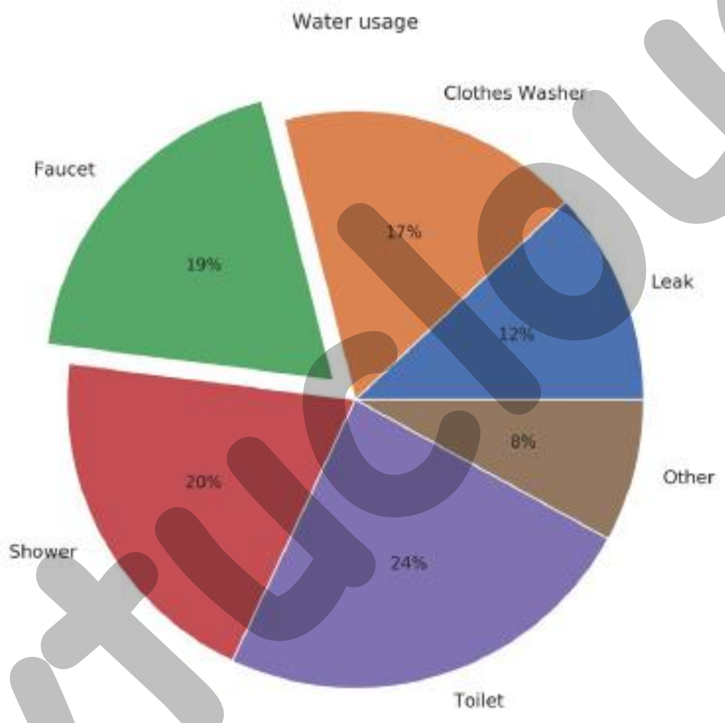


Figure 3.20: Pie chart for water usage

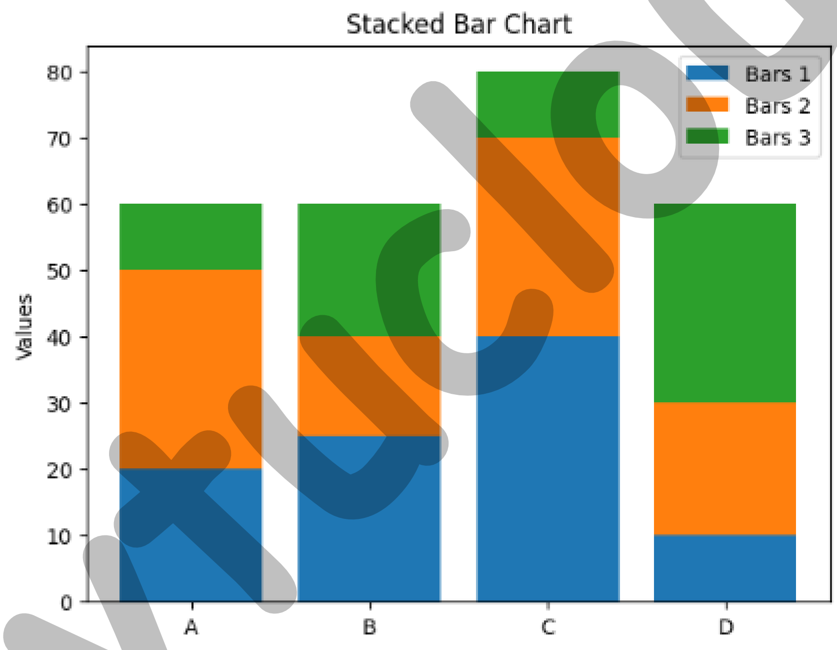
5.6.3 Stacked Bar Chart

- A stacked bar chart uses the same plt.bar function as bar charts.
- For each stacked bar, the plt.bar function must be called, and the bottom parameter must be specified, starting with the second stacked bar.

plt.bar(x, bars1)
plt.bar(x, bars2, bottom=bars1)
plt.bar(x, bars3, bottom=np.add(bars1, bars2))

Example:

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

labels = ['A', 'B', 'C', 'D']
x = np.arange(len(labels))
bars1 = [20, 25, 40, 10]
bars2 = [30, 15, 30, 20]
bars3 = [10, 20, 10, 30]

plt.bar(x, bars1, label='Bars 1')
plt.bar(x, bars2, bottom=bars1, label='Bars 2')
plt.bar(x, bars3, bottom=np.add(bars1, bars2), label='Bars 3')

plt.xticks(x, labels)
plt.ylabel('Values')
plt.title('Stacked Bar Chart')
plt.legend()

plt.show()
```



5.6.3.1 Activity - Creating a Stacked Bar Plot to Visualize Restaurant Performance

- You are the owner of a restaurant and, due to a new law, you have to introduce a No Smoking Day. To make as few losses as possible, you want to visualize how many sales are made every day, categorized by smokers and non-smokers.

- Use the dataset tips from Seaborn, which contains multiple entries of restaurant bills, and create a matrix where the elements contain the sum of the total bills for each day and smokers/non-smokers
- Import all the necessary dependencies and load the tips dataset. Note that we have to import the Seaborn library to load the dataset.
- Use the given dataset and create a matrix where the elements contain the sum of the total bills for each day and split according to smokers/non-smokers.
- Create a stacked bar plot, stacking the summed total bills separated according to smoker and non-smoker for each day.
- Add a legend, labels, and a title.

Solution:

```
# Import statements
import pandas as sb
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
# Load dataset
bills = sns.load_dataset('tips')
print(bills)
days = ['Thur', 'Fri', 'Sat', 'Sun']
days_range = np.arange(len(days))
smoker = ['Yes', 'No']
bills_by_days = [bills[bills['day'] == day] for day in days]
print('bills by day',bills_by_days)
bills_by_days_smoker =  [[bills_by_days[day][bills_by_days[day]['smoker'] == s]
                for s in smoker] for day in days_range]
print('Bills by days smoker',bills_by_days_smoker)
total_by_days_smoker =[[bills_by_days_smoker[day][s]['total_bill'].sum()
              for s in range(len(smoker))] for day in days_range]
print('Total bills by smoker',total_by_days_smoker)
totals = np.asarray(total_by_days_smoker)
# Create figure
plt.figure(figsize=(10, 5), dpi=300)
# Create stacked bar plot
plt.bar(days_range, totals[:, 0], label='Smoker')
plt.bar(days_range, totals[:, 1], bottom=totals[:, 0], label='Non-smoker')
# Add legend
plt.legend()
# Add labels and title
plt.xticks(days_range)
ax = plt.gca()
ax.set_xticklabels(days)
ax.yaxis.grid()
plt.ylabel('Daily total sales in $')
plt.title('Restaurant performance')
```

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

Output:

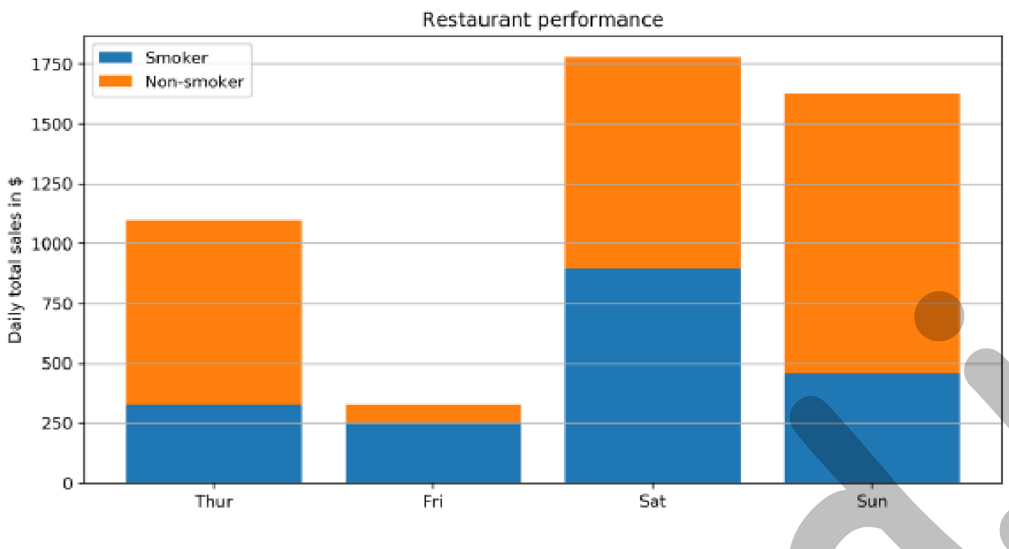


Figure 3.22: Stacked bar chart showing the performance of a restaurant on different days

Questions:

1. Why legends are used?
2.Define Bar chart
3.List the Different Parameters in Bar Chart.
4.Define Pie Chart
5.List the parameters involved in pie chart
6.Define Stacked Bar Chart
7.List the difference parameters involved in Stacked bar chart

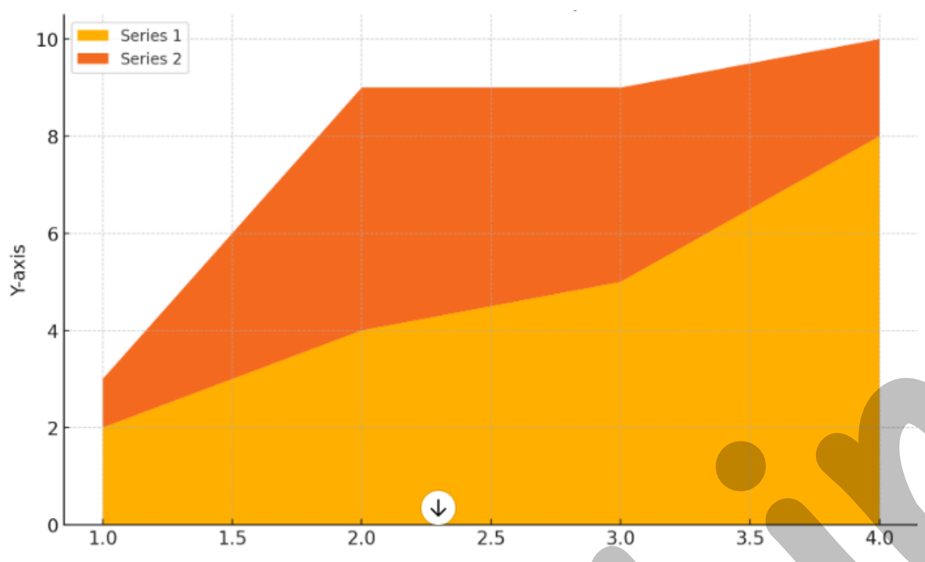
Handouts for Session 39: Stacked Area Chart, Histogram, Box Plot, Scatter Plot, Bubble Plot, Layouts:Subplots

5.6.4 Stacked Area Chart

- plt.stackplot(x, y) creates a stacked area plot.

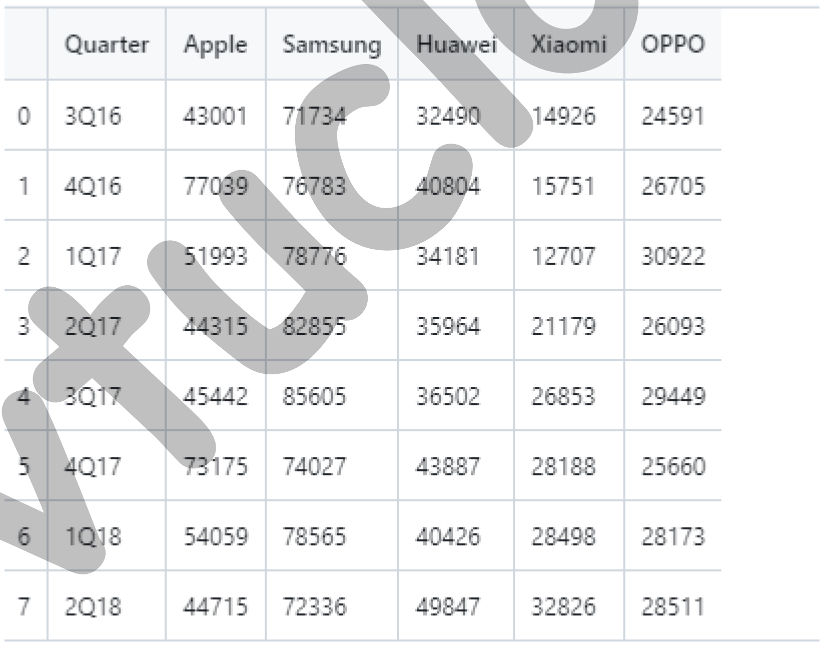
Important parameters:

- **x:** Specifies the x-values of the data series.
- **y:** Specifies the y-values of the data series.
- For multiple series, either as a 2D array or any number of 1D arrays, call the following function:
- **plt.stackplot(x, y1, y2, y3, …).**
- **labels (optional): Specifies the labels as a list or tuple for each data series.**
- **plt.stackplot([1, 2, 3, 4], [2, 4, 5, 8], [1, 5, 4, 2])**



5.6.4.1 Activity - Comparing Smartphone Sales Units Using a Stacked Area Chart

- Let's look at the following scenario: you want to invest in one of the five biggest smartphone manufacturers. Looking at the quarterly sales units as part of a whole may be a good indicator of which company to invest in:
- Import the necessary modules and enable plotting within a Jupyter Notebook.
- Use pandas to read the smartphone_sales.csv dataset
- Create a visually appealing stacked area chart. Add a legend, labels, and a title.
- After executing the preceding steps, the expected output should be as follows:



	Quarter	Apple	Samsung	Huawei	Xiaomi	OPPO
0	3Q16	43001	71734	32490	14926	24591
1	4Q16	77039	76783	40804	15751	26705
2	1Q17	51993	78776	34181	12707	30922
3	2Q17	44315	82855	35964	21179	26093
4	3Q17	45442	85605	36502	26853	29449
5	4Q17	73175	74027	43887	28188	25660
6	1Q18	54059	78565	40426	28498	28173
7	2Q18	44715	72336	49847	32826	28511

```
# Import statements
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
%matplotlib inline
# Load dataset
sales = pd.read_csv('smartphone_sales.csv')
# Create figure
plt.figure(figsize=(10, 6), dpi=300)
# Extracting labels from the DataFrame columns, starting from the third column
labels = sales.columns[2:]
# Create stacked area chart
plt.stackplot('Quarter', 'Apple', 'Samsung', 'Huawei', 'Xiaomi', 'OPPO', data=sales,
labels=labels)
# Add legend
plt.legend()
# Add labels and title
plt.xlabel('Quarters')
plt.ylabel('Sales units in thousands')
plt.title('Smartphone sales units')
# Show plot
plt.show()
```

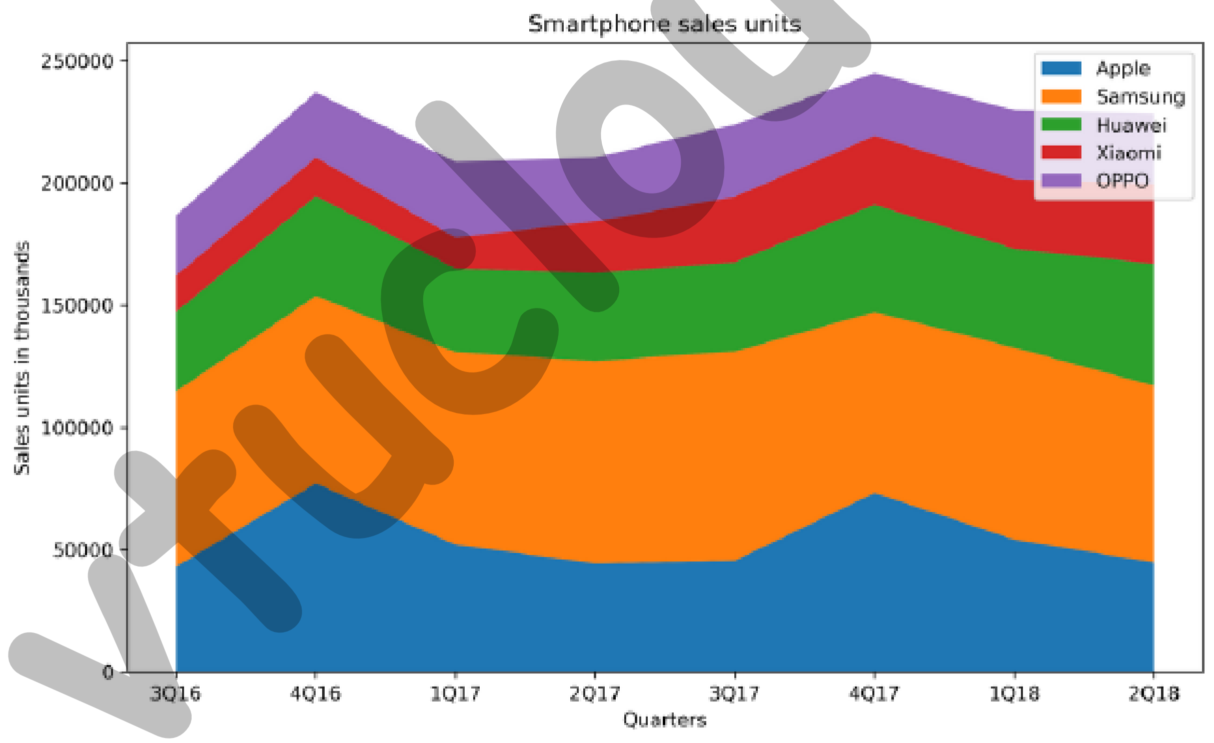


Figure 3.24: Stacked area chart comparing sales units of different smartphone manufacturers

5.6.5 Histogram

- A histogram visualizes the distribution of a single numerical variable. Each bar represents the frequency for a certain interval. The plt.hist(x) function creates a histogram.

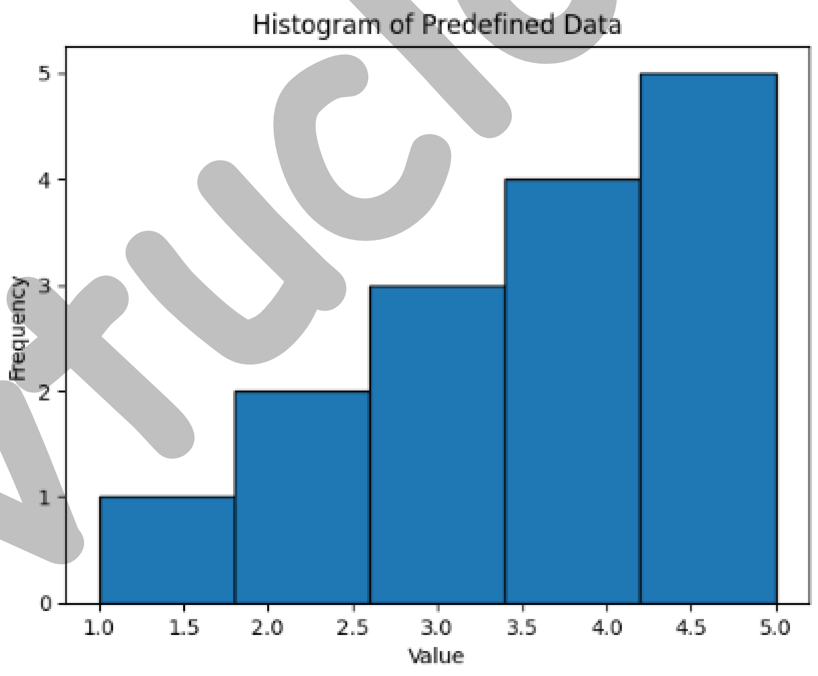
Important parameters:

- x: Specifies the input values.

- bins: (optional): Specifies the number of bins as an integer or specifies the bin edges as a list.
- range: (optional): Specifies the lower and upper range of the bins as a tuple.
- density: (optional): If true, the histogram represents a probability density.
  Syntax: plt.hist(x, bins=30, density=True)

Example 1

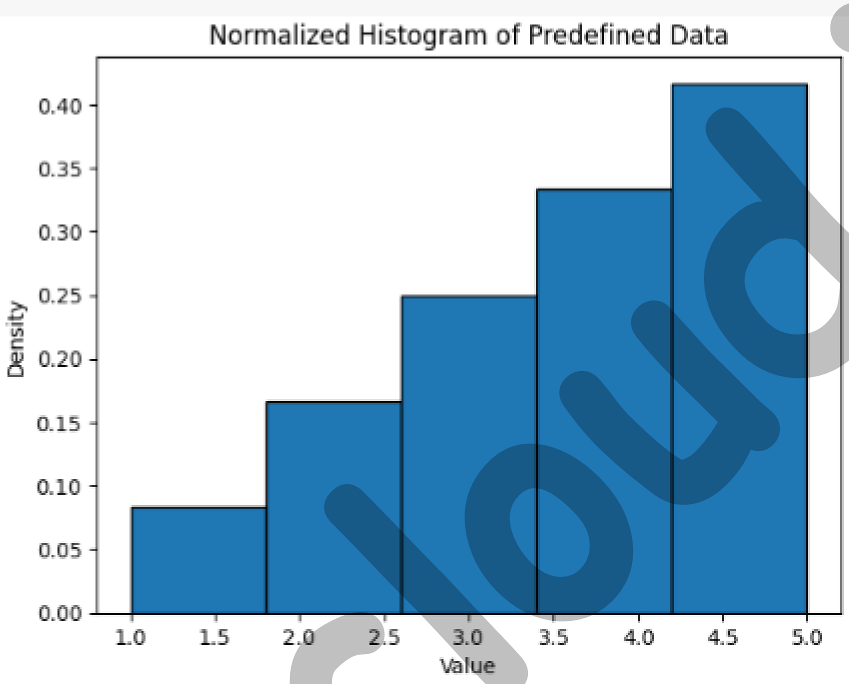
```
import matplotlib.pyplot as plt

# Define the data
data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5]

# Create the histogram
plt.hist(data, bins=5, edgecolor='black')
#Range of data: 1 to 5
#Width of each bin: (5 - 1) / 5 = 4 / 5 = 0.8

# Adding labels and title
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram of Predefined Data')

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



Example 2:

```
import matplotlib.pyplot as plt

# Define the data
data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5]
```

```
# Create the histogram with density=True
plt.hist(data, bins=5, edgecolor='black', density=True)

# Adding labels and title
plt.xlabel('Value')
plt.ylabel('Density')
plt.title('Normalized Histogram of Predefined Data')

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

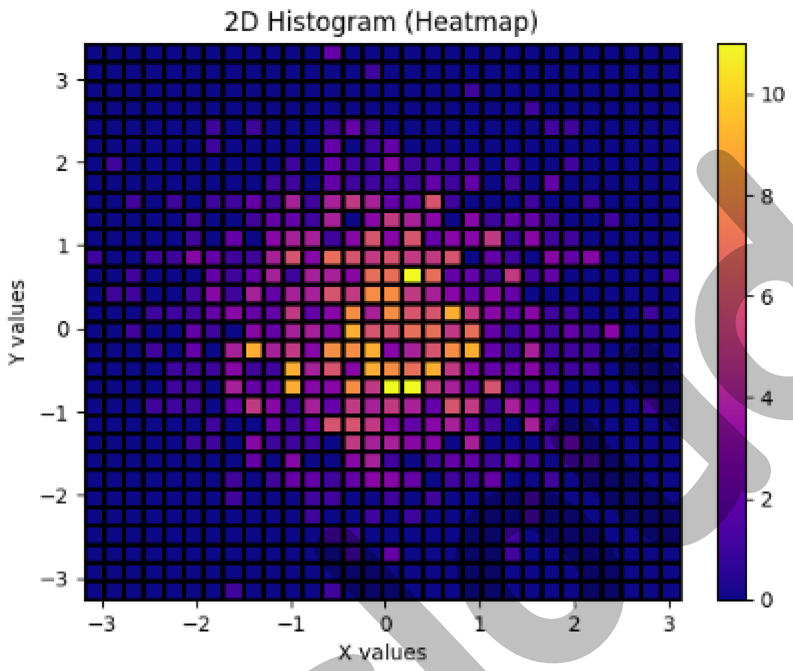


- **plt.hist2d(x, y)** creates a 2D histogram.2D histograms can be used to visualize the frequency of two-dimensional data.
- The data is plotted on the xy-plane and the frequency is indicated by the color.
- Histograms are a good way to visualize an estimated density of your data.

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

# Generate some random data
x = np.random.randn(1000)
y = np.random.randn(1000)

# Create a 2D histogram (heatmap)
plt.hist2d(x, y, bins=30,edgecolor='black', cmap='plasma')

# Adding labels and title
```

```
plt.xlabel('X values')
plt.ylabel('Y values')
plt.title('2D Histogram (Heatmap)')

# Display the plot
plt.colorbar()
plt.show()
```

Output:



5.6.6 Box Plot

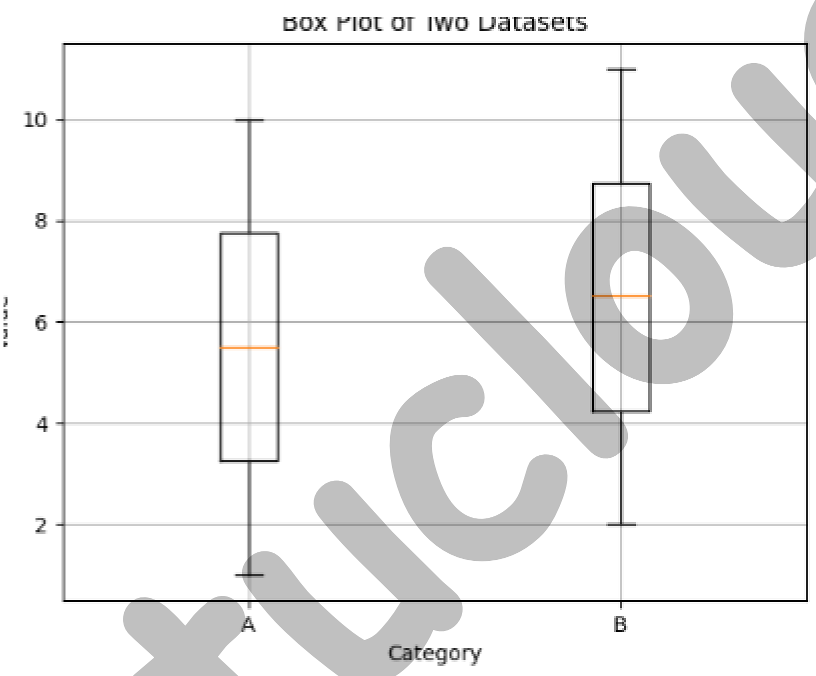
- The box plot shows multiple statistical measurements. The box extends from the lower to the upper quartile values of the data, thereby allowing us to visualize the interquartile range. The plt.boxplot(x) function creates a box plot.

Important parameters:

- x: Specifies the input data. It specifies either a 1D array for a single box, or a sequence of arrays for multiple boxes.
- notch: (optional) If true, notches will be added to the plot to indicate the confidence interval around the median.
- labels: (optional) Specifies the labels as a sequence.
- showfliers: (optional) By default, it is true, and outliers are plotted beyond the caps.
- showmeans: (optional) If true, arithmetic means are shown.
- plt.boxplot([x1, x2], labels=['A', 'B'])

Example 1:

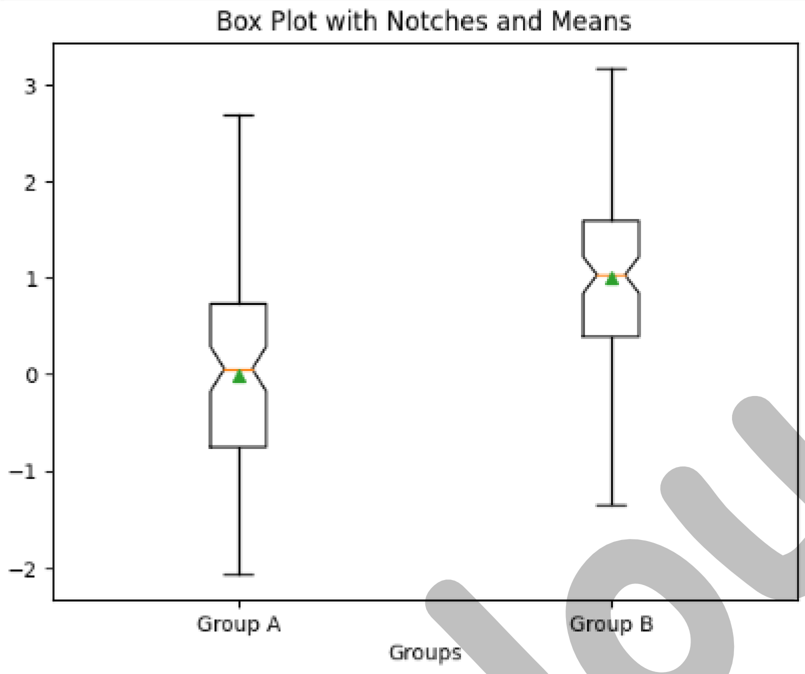
```
# Define the data for two categories
x1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
x2 = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

# Create the box plot
plt.boxplot([x1, x2], labels=['A', 'B'])

# Adding labels and title
plt.xlabel('Category')
plt.ylabel('Value')
plt.title('Box Plot of Two Datasets')
# Add grid
plt.grid(True)
# Display the plot
plt.show()
```



Example 2:

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

# Generate some sample data
data1 = np.random.normal(loc=0, scale=1, size=100)
data2 = np.random.normal(loc=1, scale=1, size=100)

# Create a box plot with specified parameters
plt.boxplot([data1, data2], notch=True, labels=['Group A', 'Group B'], showfliers=True, showmeans=True)
```

```
# Adding labels and title
plt.xlabel('Groups')
plt.ylabel('Values')
plt.title('Box Plot with Notches and Means')

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



5.6.6.1 Activity - Using a Histogram and a Box Plot to Visualize Intelligence Quotient

- Visualize the intelligence quotient (IQ) of 100 applicants using histogram and box plots. 100 people have come for an interview in a company. To place an individual applicant in the overall group, a histogram and a box plot shall be used.
- Import the necessary modules and enable plotting within a Jupyter Notebook.
- Use the IQ scores to create the plots
- Plot a histogram with 10 bins for the given IQ scores. IQ scores are normally distributed with a mean of 100 and a standard deviation of 15. Visualize the mean as a vertical solid red line, and the standard deviation using dashed vertical lines. Add labels and a title.
- Create a box plot to visualize the same IQ scores. Add labels and a title.
- Create a box plot for each of the IQ scores of the different test groups. Add labels and a title

Solution:

```
# IQ samples
    iq_scores = [126, 89, 90, 101, 102, 74, 93, 101, 66,120, 108, 97, 98, 105, 119, 92, 113,
    81,104, 108, 83, 102, 105, 111, 102, 107, 103,89, 89, 110, 71, 110, 120, 85, 111, 83,
    122,120, 102, 84, 118, 100, 100, 114, 81, 109, 69,97, 95, 106, 116, 109, 114, 98, 90, 92,
    98,91, 81, 85, 86, 102, 93, 112, 76, 89, 110,75, 100, 90, 96, 94, 107, 108, 95, 96, 96,114,
    93, 95, 117, 141, 115, 95, 86, 100, 121,103, 66, 99, 96, 111, 110, 105, 110, 91, 112,102,
    112, 75]
    group_a = [118, 103, 125, 107, 111, 96, 104, 97, 96,114, 96, 75, 114, 107, 87, 117, 117,
    114,117, 112, 107, 133, 94, 91, 118, 110, 117,86, 143, 83, 106, 86, 98, 126, 109, 91,
    112,120, 108, 111, 107, 98, 89, 113, 117, 81, 113, 112, 84, 115, 96, 93, 128, 115, 138, 121,
    87,112, 110, 79, 100, 84, 115, 93, 108, 130, 107,106, 106, 101, 117, 93, 94, 103, 112, 98,
    103,70, 139, 94, 110, 105, 122, 94, 94, 105, 129, 110, 112, 97, 109, 121, 106, 118, 131, 88,
    122,125, 93, 78]
    group_b = [126, 89, 90, 101, 102, 74, 93, 101, 66,120, 108, 97, 98, 105, 119, 92, 113, 81,
    104, 108, 83, 102, 105, 111, 102, 107, 103,89, 89, 110, 71, 110, 120, 85, 111, 83,122, 120,
    102, 84, 118, 100, 100, 114, 81,109, 69, 97, 95, 106, 116, 109, 114, 98, 90, 92, 98, 91, 81,
    85, 86, 102, 93, 112, 76, 89, 110, 75, 100, 90, 96, 94, 107, 108, 95, 96, 96, 114, 93, 95, 117,
    141, 115, 95,86, 100, 121, 103, 66, 99, 96, 111, 110, 105, 110, 91, 112, 102, 112, 75]

    group_c = [108, 89, 114, 116, 126, 104, 113, 96, 69, 121,109, 102, 107, 122, 104, 107, 108,
    137, 107, 116,98, 132, 108, 114, 82, 93, 89, 90, 86, 91,99, 98, 83, 93, 114, 96, 95, 113, 103,
    81,107, 85, 116, 85, 107, 125, 126, 123, 122, 124,115, 114, 93, 93, 114, 107, 107, 84, 131,
    91,108, 127, 112, 106, 115, 82, 90, 117, 108, 115,113, 108, 104, 103, 90, 110, 114, 92, 101,
    72,109, 94, 122, 90, 102, 86, 119, 103, 110, 96,90, 110, 96, 69, 85, 102, 69, 96, 101, 90]

    group_d = [93, 99, 91, 110, 80, 113, 111, 115, 98, 74,96, 80, 83, 102, 60, 91, 82, 90, 97,
    101,89, 89, 117, 91, 104, 104, 102, 128, 106, 111, 79, 92, 97, 101, 106, 110, 93, 93, 106,
    108, 85, 83, 108, 94, 79, 87, 113, 112, 111, 111, 79, 116, 104, 84, 116, 111, 103, 103, 112,
    68,54, 80, 86, 119, 81, 84, 91, 96, 116, 125,99, 58, 102, 77, 98, 100, 90, 106, 109, 114,102,
    102, 112, 103, 98, 96, 85, 97, 110, 131,92, 79, 115, 122, 95, 105, 74, 85, 85, 95]

    # Import statements
    import numpy as np
    import matplotlib.pyplot as plt
    %matplotlib inline
    #Plot a histogram with 10 bins for the given IQ scores. IQ scores are normally distributed
    with a mean of 100 and a standard deviation of 15. Visualize the mean as a vertical solid
    red line, and the standard deviation using dashed vertical lines. Add labels and a title

    # Create figure
    plt.figure(figsize=(6, 4), dpi=150)
    # Create histogram
    plt.hist(iq_scores, bins=10)
    plt.axvline(x=100, color='r')
    plt.axvline(x=115, color='r', linestyle= '--')
    plt.axvline(x=85, color='r', linestyle= '--')
    # Add labels and title
    plt.xlabel('IQ score')
    plt.ylabel('Frequency')
    plt.title('IQ scores for a test group of a hundred adults')
    # Show plot
    plt.show()
    # Create figure
    plt.figure(figsize=(6, 4), dpi=150)
    # Create histogram
    plt.boxplot(iq_scores)
    # Add labels and title
    ax = plt.gca()
    ax.set_xticklabels(['Test group'])
    plt.ylabel('IQ score')
    plt.title('IQ scores for a test group of a hundred adults')
    # Show plot
    plt.show()
    # Create figure
    plt.figure(figsize=(6, 4), dpi=150)
    # Create histogram
    plt.boxplot([group_a, group_b, group_c, group_d])
    # Add labels and title
    ax = plt.gca()
    ax.set_xticklabels(['Group A', 'Group B', 'Group C', 'Group D'])
    plt.ylabel('IQ score')
    plt.title('IQ scores for different test groups')
    # Show plot
    plt.show()
```

Output:

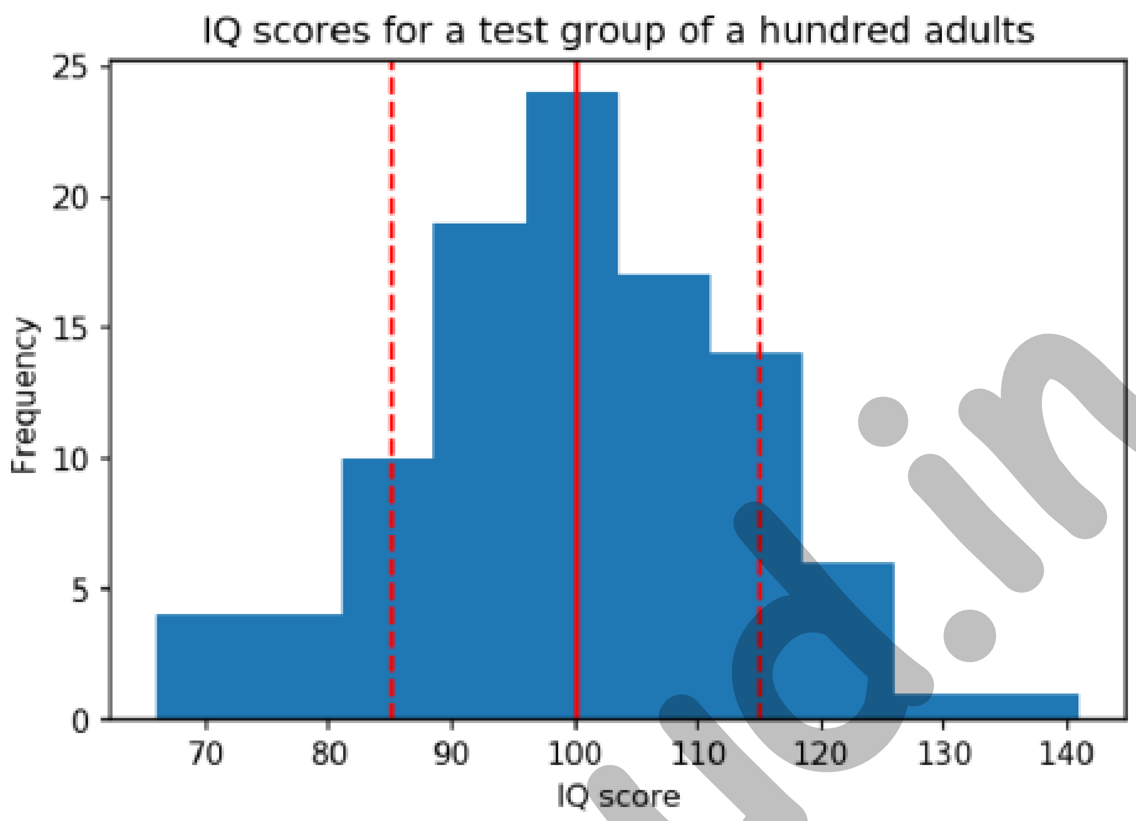


Figure 3.54: Histogram for an IQ test

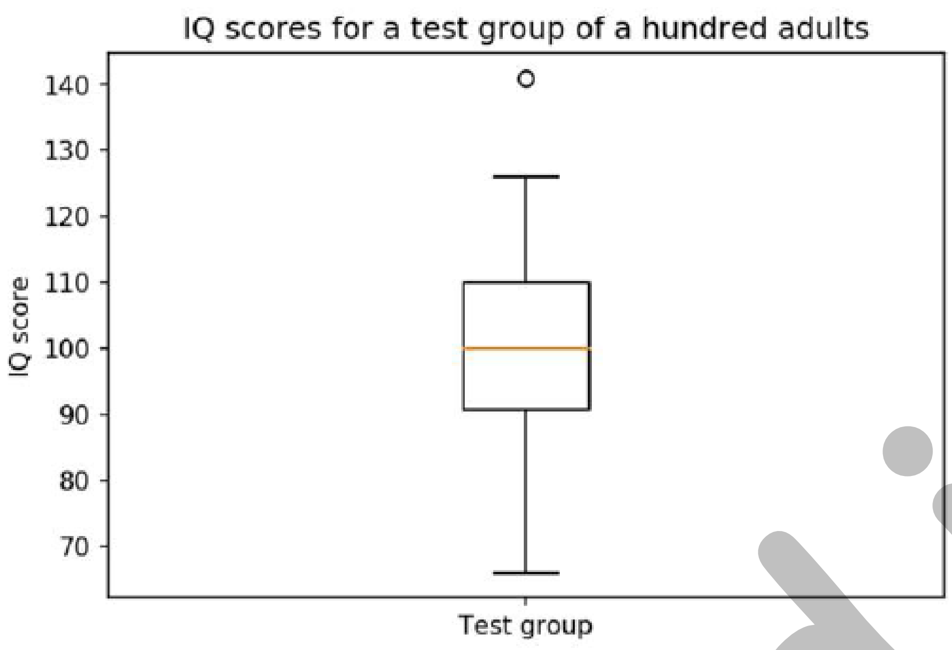


Figure 3.29: Box plot for IQ scores

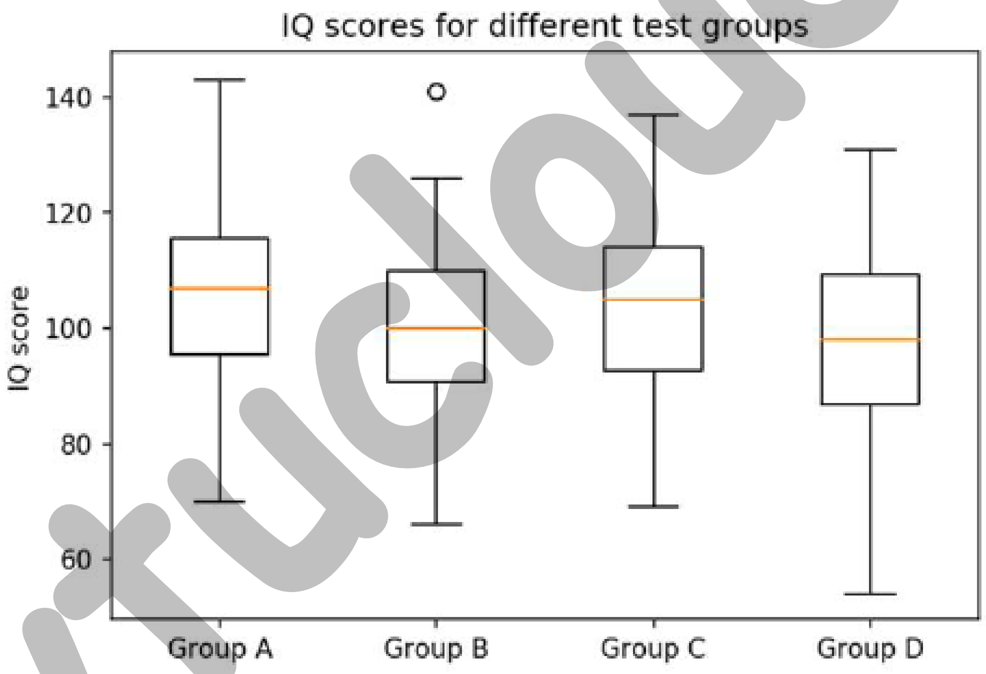


Figure 3.30: Box plot for IQ scores of different test groups

5.6.7 Scatter Plot

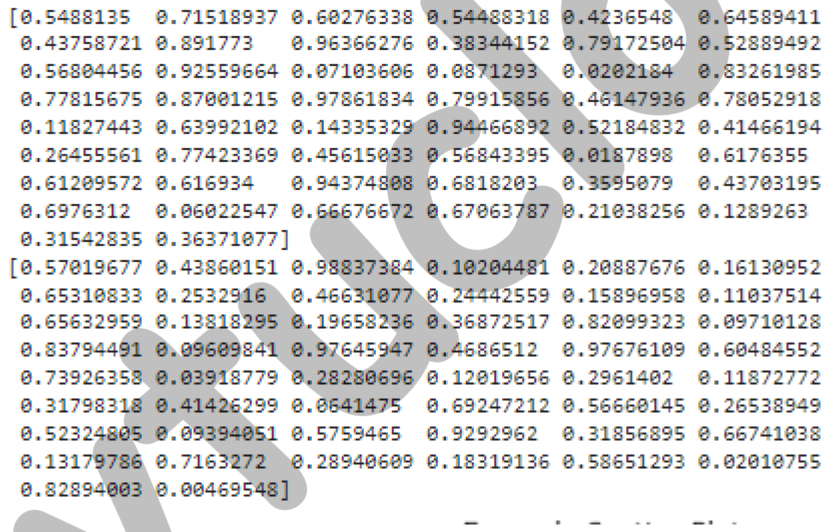
- Scatter plots show data points for two numerical variables, displaying a variable on both axes. plt.scatter(x, y) creates a scatter plot of y versus x, with optionally varying marker size and/or color.
- Important parameters:
- x, y: Specifies the data positions.

- s: (optional) Specifies the marker size in points squared.
- c:(optional) Specifies the marker color. If a sequence of numbers is specified, the numbers will be mapped to the colors of the color map.
- **plt.scatter(x, y)**

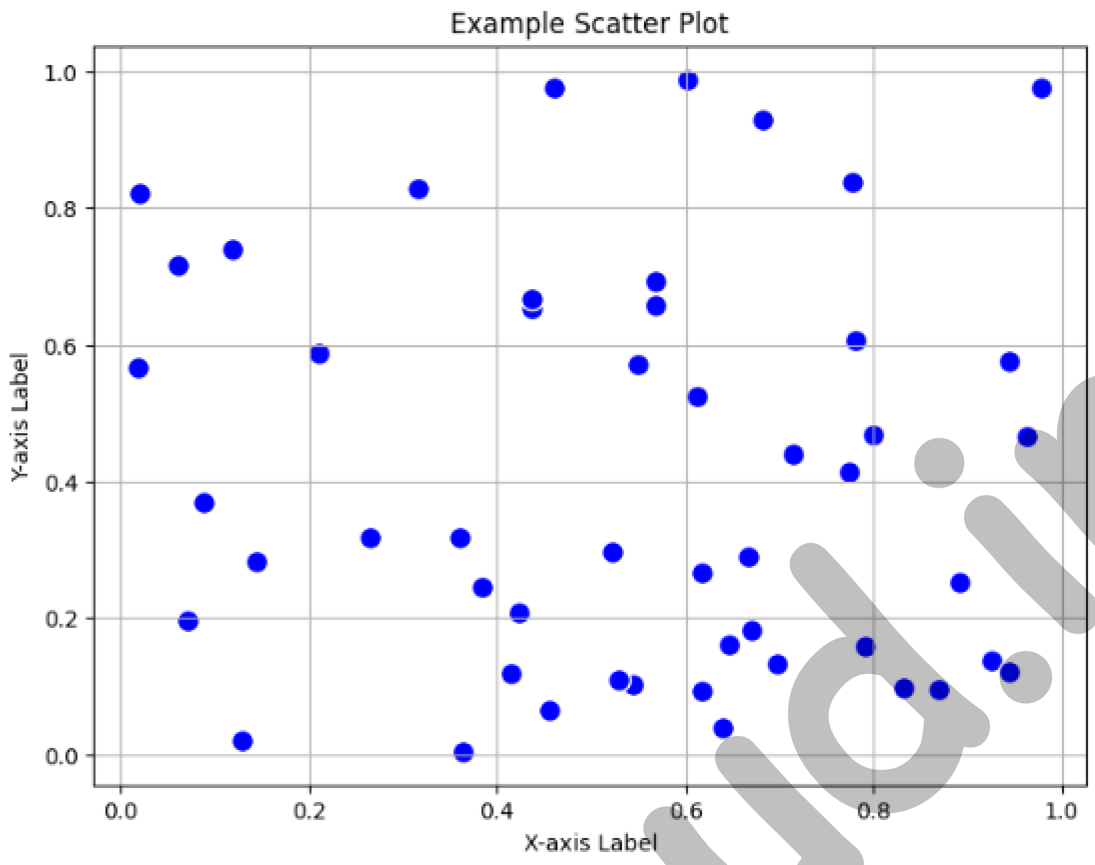
Example:

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

# Generating some random data
x = np.random.rand(50)
y = np.random.rand(50)
print(x)
print(y)
# Creating the scatter plot
plt.figure(figsize=(8, 6))
scatter=plt.scatter(x, y, c='blue', alpha=1, edgecolors='w', s=100)
plt.title('Example Scatter Plot')
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')
plt.grid(True)
plt.show()
```

Output:



```
[0.5488135  0.71518937 0.60276338 0.54488318 0.4236548  0.64589411
 0.43758721 0.891773   0.96366276 0.38344152 0.79172504 0.52889492
 0.56804456 0.92559664 0.07103606 0.0871293  0.0202184  0.83261985
 0.77815675 0.87001215 0.97861834 0.79915856 0.46147936 0.78052918
 0.11827443 0.63992102 0.14335329 0.94466892 0.52184832 0.41466194
 0.26455561 0.77423369 0.45615033 0.56843395 0.0187898  0.6176355
 0.61209572 0.616934   0.94374808 0.6818203  0.3595079  0.43703195
 0.6976312  0.06022547 0.66676672 0.67063787 0.21038256 0.1289263
 0.31542835 0.36371077]
[0.57019677 0.43860151 0.98837384 0.10204481 0.20887676 0.16130952
 0.65310833 0.2532916  0.46631077 0.24442559 0.15896958 0.11037514
 0.65632959 0.13818295 0.19658236 0.36872517 0.82099323 0.09710128
 0.83794491 0.09609841 0.97645947 0.4686512  0.97676109 0.60484552
 0.73926358 0.03918779 0.28280696 0.12019656 0.2961402  0.11872772
 0.31798318 0.41426299 0.0641475  0.69247212 0.56660145 0.26538949
 0.52324805 0.09394051 0.5759465  0.9292962  0.31856895 0.66741038
 0.13179786 0.7163272  0.28940609 0.18319136 0.58651293 0.02010755
 0.82894003 0.00469548]
```



Example 2:

```
import pandas as pd
import numpy as np

# Sample data
data = pd.DataFrame({
    'Maximum longevity (yrs)': [12, 14, np.nan, 10, np.inf, 20],
    'Body mass (g)': [200, np.nan, 350, 180, 210, np.inf]
})

print("Original DataFrame:")
print(data)
# Column names
longevity = 'Maximum longevity (yrs)'
mass = 'Body mass (g)'


# Filtering the DataFrame
filtered_data = data[np.isfinite(data[longevity]) & np.isfinite(data[mass])]

print("\nFiltered DataFrame:")
print(filtered_data)
```

```
Original DataFrame:
   Maximum longevity (yrs)  Body mass (g)
0                     12.0          200.0
1                     14.0            NaN
2                      NaN          350.0
3                     10.0          180.0
4                      inf          210.0
5                     20.0            inf

Filtered DataFrame:
   Maximum longevity (yrs)  Body mass (g)
0                     12.0          200.0
3                     10.0          180.0
```

5.6.7.1 Activity - Using a Scatter Plot to Visualize Correlation between Various Animals

- Use a scatter plot to show correlation within a dataset. You are given a dataset containing information about various animals. Visualize the correlation between the various animal attributes such as Maximum longevity in years and Body mass in grams.
- Import the necessary modules and enable plotting within the Jupyter Notebook
- Use pandas to read the data
- The given dataset is not complete. Filter the data so that you end up with samples containing a body mass and a maximum longevity. Sort the data according to the animal class; here, the isfinite() function (to check whether the number is finite or not) checks for the finiteness of the given element
- Create a scatter plot visualizing the correlation between the body mass and the maximum longevity. Use different colors to group data samples according to their class. Add a legend, labels, and a title. Use a log scale for both the x-axis and y-axis

Solution:

```
# Import statements
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
# Load dataset
data = pd.read_csv('anage_data.csv')
# Preprocessing
longevity = 'Maximum longevity (yrs)'
mass = 'Body mass (g)'
data = data[np.isfinite(data[longevity]) & np.isfinite(data[mass])]
# Sort according to class
amphibia = data[data['Class'] == 'Amphibia']
aves = data[data['Class'] == 'Aves']
mammalia = data[data['Class'] == 'Mammalia']
reptilia = data[data['Class'] == 'Reptilia']
# Create figure
plt.figure(figsize=(10, 6), dpi=300)
```

```
# Create scatter plot
plt.scatter(amphibia[mass], amphibia[longevity],label='Amphibia')
plt.scatter(aves[mass], aves[longevity], label='Aves')
plt.scatter(mammalia[mass], mammalia[longevity],label='Mammalia')
plt.scatter(reptilia[mass], reptilia[longevity],label='Reptilia')
# Add legend
plt.legend()
# Log scale
ax = plt.gca()
ax.set_xscale('log')
ax.set_yscale('log')
# Add labels
plt.xlabel('Body mass in grams')
plt.ylabel('Maximum longevity in years')
# Show plot
plt.show()
```

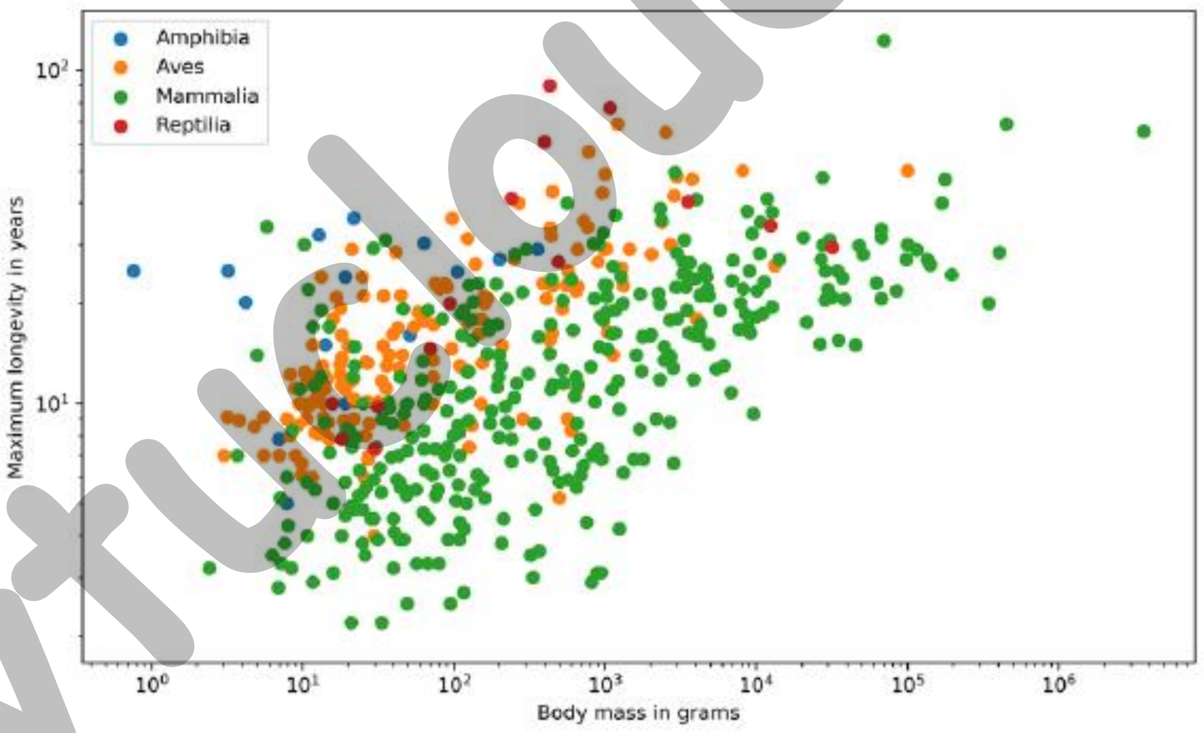


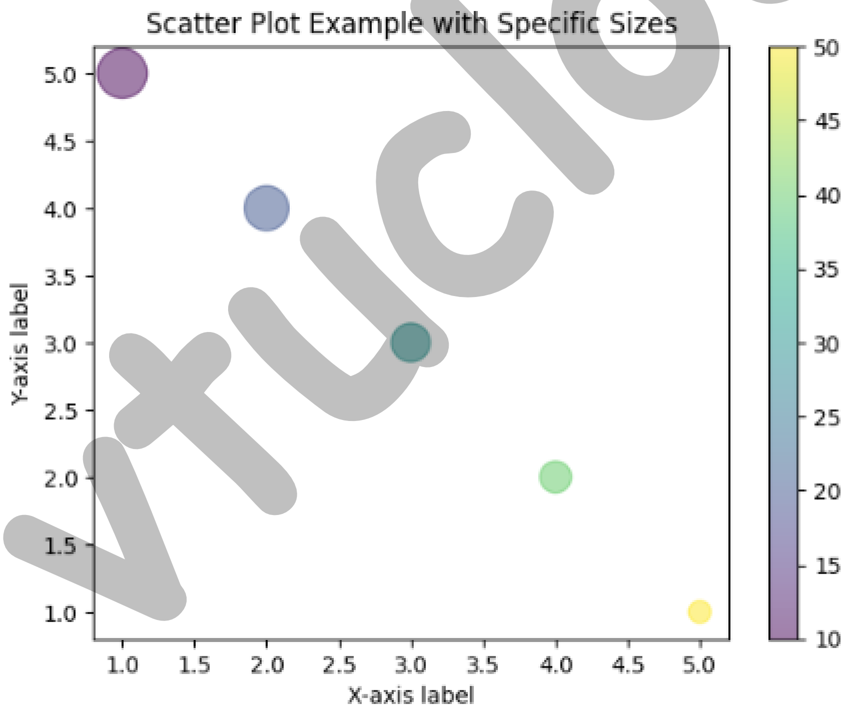
Figure 3.32: Scatter plot on animal statistics

5.6.8 Bubble Plot

- The plt.scatter function is used to create a bubble plot. To visualize a third or fourth variable, the parameters s (scale) and c (color) can be used.
- **plt.scatter(x, y, s=z*500, c=c, alpha=0.5)**
- **plt.colorbar()**

- The colorbar function adds a colorbar to the plot, which indicates the value of the color.

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

# Specific data
x = np.array([1, 2, 3, 4, 5])
y = np.array([5, 4, 3, 2, 1])
z = np.array([10, 8, 6, 4, 2])  # Relative sizes of the points
c = np.array([10, 20, 30, 40, 50])  # Colors of the points

# Create scatter plot
plt.scatter(x, y, s=z*500, c=c, alpha=0.5, cmap='viridis')

# Add color bar
plt.colorbar()

# Show plot
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Scatter Plot Example with Specific Sizes')
plt.show()
```



5.7 Layouts

- There are multiple ways to define a visualization layout in Matplotlib.
- By layout, we mean the arrangement of multiple Axes within a Figure.

- Subplots and tight layout to create visually appealing plots
- GridSpec, which offers a more flexible way to create multi-plots.

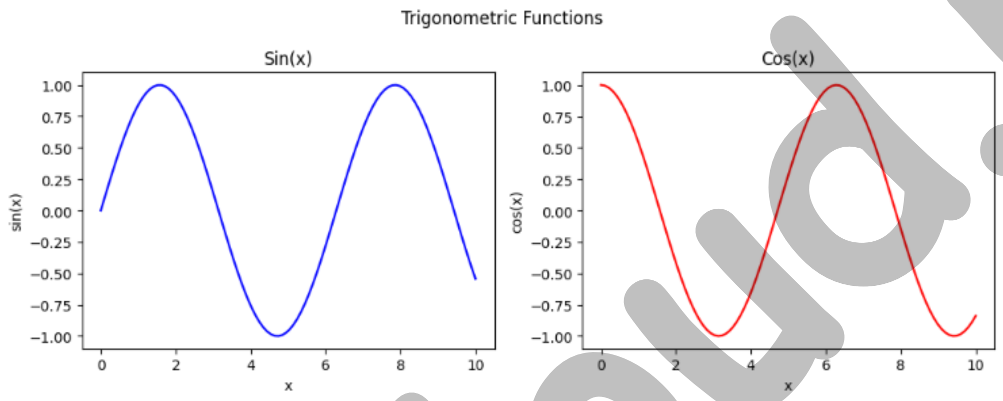
5.7.1 Subplots

- It is often useful to display several plots next to one another. Matplotlib offers the concept of subplots, which are multiple Axes within a Figure. These plots can be grids of plots, nested plots, and so on.
- The plt.subplots(, ncols) function creates a Figure and a set of subplots. nrows, ncols define the number of rows and columns of the subplots, respectively.
- The plt.subplot(nrows, ncols, index) function or, equivalently, plt.subplot(pos) adds a subplot to the current Figure. The index starts at 1. The plt.subplot(2, 2, 1) function is equivalent to plt. subplot(221).
- The Figure.subplots(nrows, ncols) function adds a set of subplots to the specified Figure
- The Figure.add_subplot(nrows, ncols, index) function or, equivalently, Figure.add_subplot(pos), adds a subplot to the specified Figure.
- To share the x-axis or y-axis, the parameters sharex and sharey must be set, respectively. The axis will have the same limits, ticks, and scale.
- plt.subplot and Figure.add_subplot have the option to set a projection. For a polar projection, either set the projection='polar' parameter or the parameter polar=True parameter.
- The plt.tight_layout() adjusts subplot parameters (primarily padding between the Figure edge and the edges of subplots, and padding between the edges of adjacent subplots) so that the subplots fit well in the Figure.If you do not use plt.tight_layout(), subplots might overlap.

Example 1:

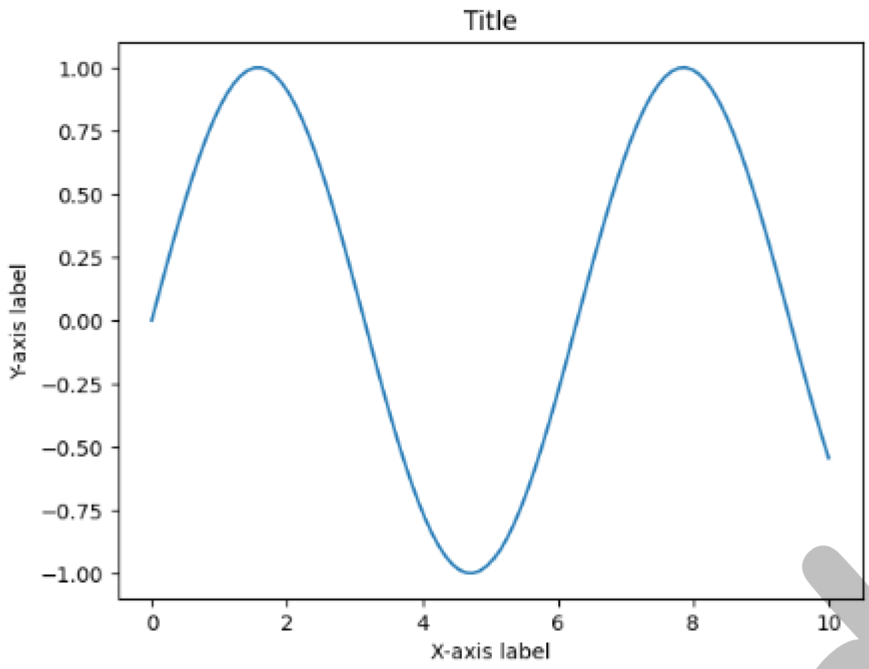
```
import matplotlib.pyplot as plt
import numpy as np

# Generate some example data
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)
# Create a figure with 1 row and 2 columns of subplots
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))

# Plot on the first subplot (left)
axes[0].plot(x, y1, color='blue')
axes[0].set_title('Sin(x)')
axes[0].set_xlabel('x')
axes[0].set_ylabel('sin(x)')

# Plot on the second subplot (right)
```

```
axes[1].plot(x, y2, color='red')
axes[1].set_title('Cos(x)')
axes[1].set_xlabel('x')
axes[1].set_ylabel('cos(x)')

# Add a title to the entire figure
fig.suptitle('Trigonometric Functions')

# Adjust layout to prevent overlap of titles and labels
plt.tight_layout()

# Display the figure
plt.show()
```



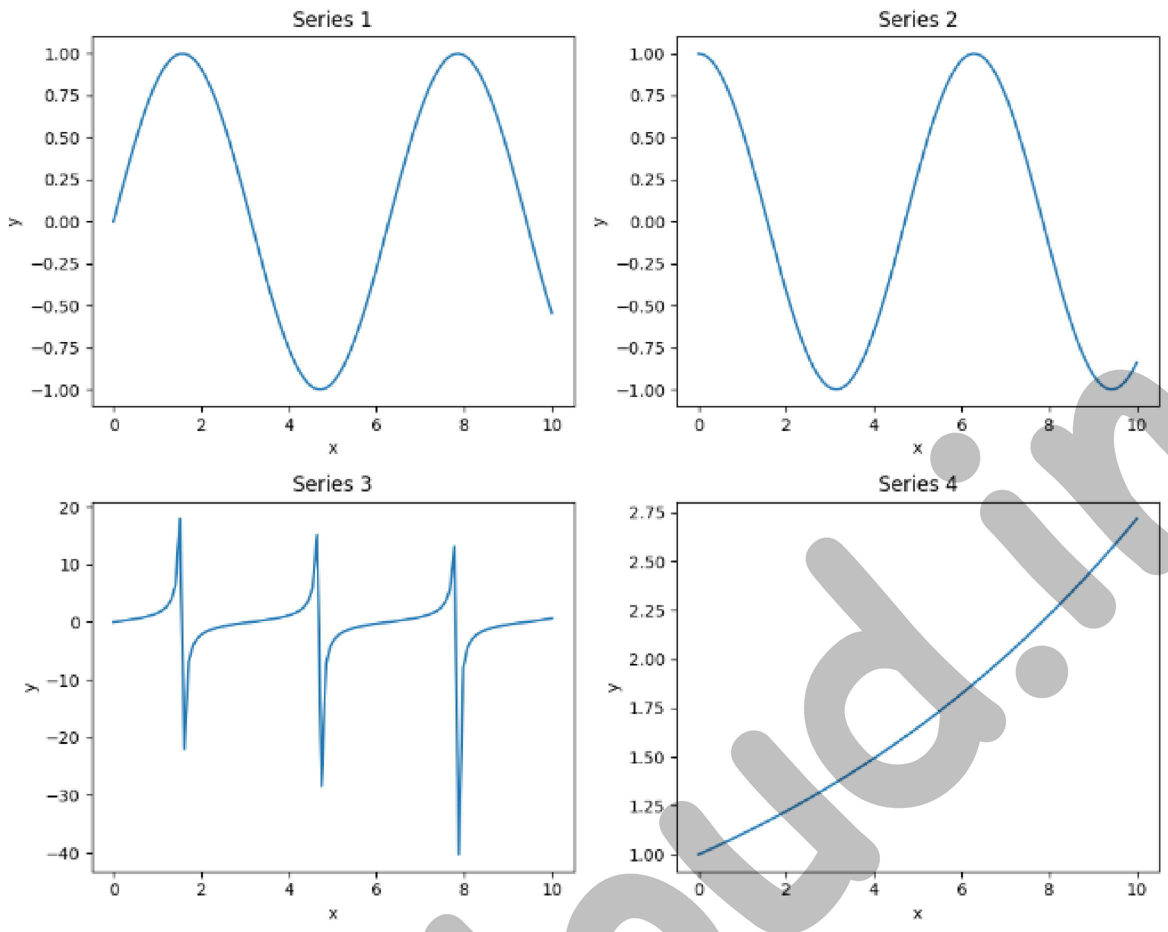
Example 2:

```
import matplotlib.pyplot as plt
fig = plt.figure()  # Create a figure object

# Add a subplot to the figure
ax = fig.add_subplot(111)
# Generate some example data
x = np.linspace(0, 10, 100)
y = np.sin(x)  # Example data for y, in this case, sin(x)
# Plot on the subplot
ax.plot(x,y)

# Customize the subplot
ax.set_title('Title')
ax.set_xlabel('X-axis label')
ax.set_ylabel('Y-axis label')

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



Example 3:

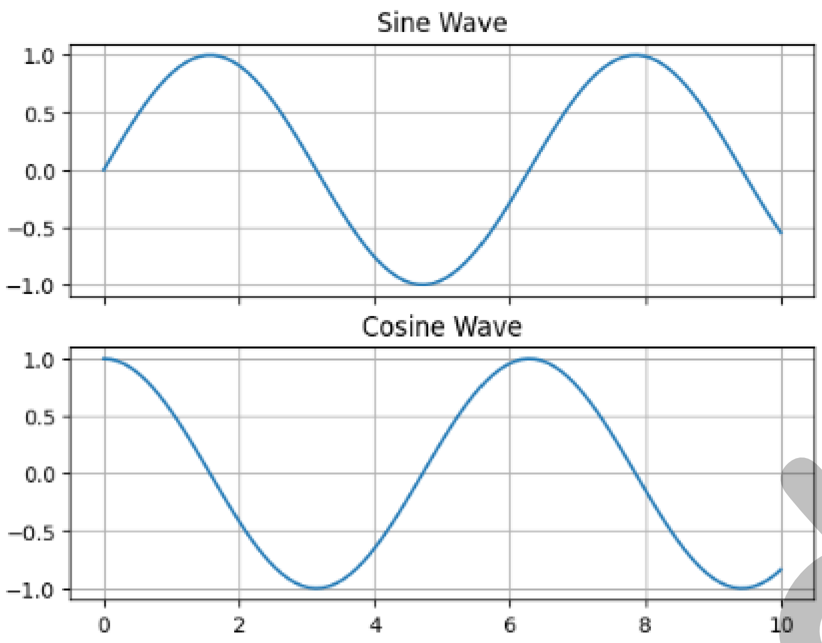
```
import matplotlib.pyplot as plt
import numpy as np

# Generate example data
x = np.linspace(0, 10, 100) #generates 100 evenly spaced numbers between 0 and 10
series = [np.sin(x), np.cos(x), np.tan(x), np.exp(x/10)]

# Create subplots using plt.subplots(2, 2)
fig, axes = plt.subplots(2, 2, figsize=(10, 8))

# Flatten the axes array
axes = axes.ravel()

# Plot each series on the respective subplot
for i, ax in enumerate(axes):
    ax.plot(x, series[i])
    ax.set_title(f'Series {i+1}')
    ax.set_xlabel('x')
    ax.set_ylabel('y')

# Adjust layout to prevent overlap of titles and labels
plt.tight_layout()

# Display the figure
plt.show()
```



Example 4: sharex property

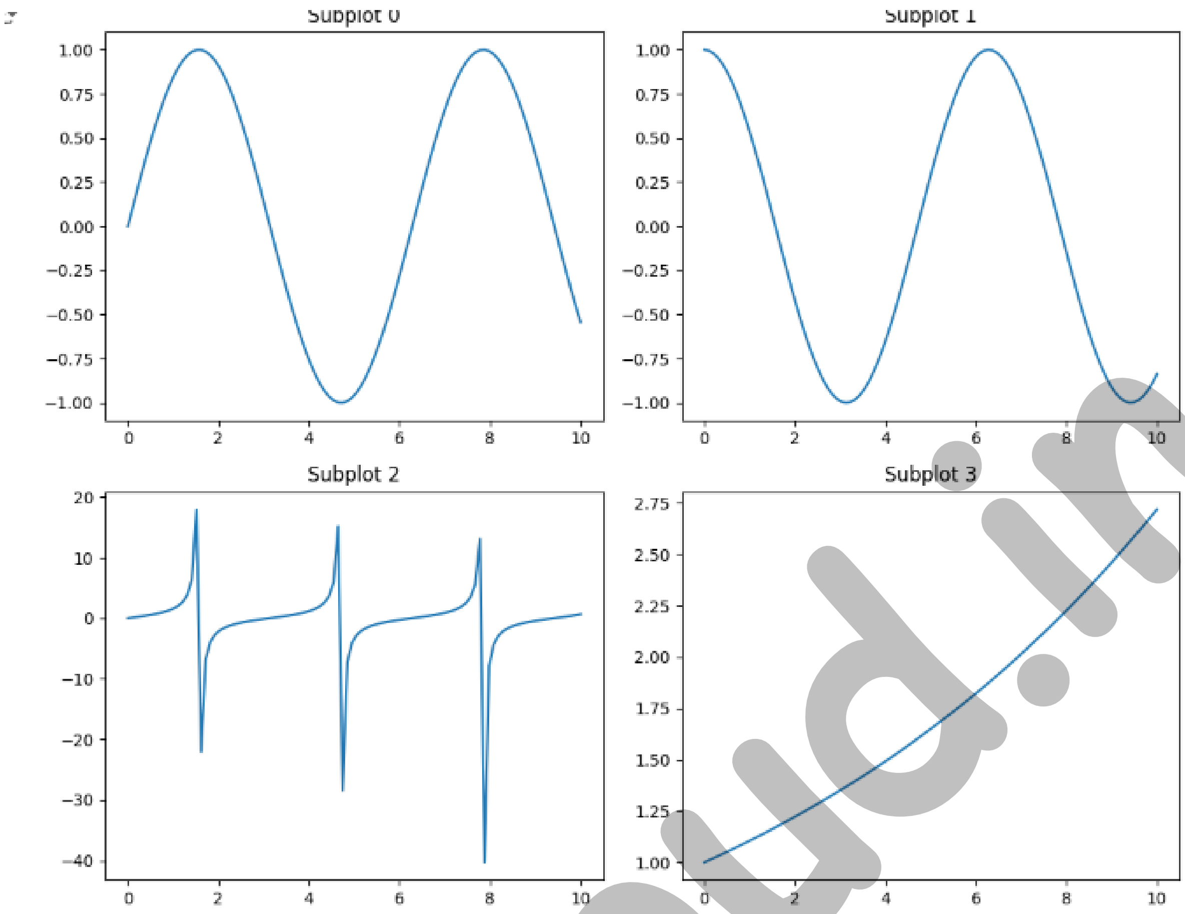
```
import matplotlib.pyplot as plt
import numpy as np

# Create some data
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

# Create a figure with two subplots sharing the x-axis
fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True)

# Plot data on the first subplot
ax1.plot(x, y1)
ax1.set_title('Sine Wave')
ax1.grid(True)

# Plot data on the second subplot
ax2.plot(x, y2)
ax2.set_title('Cosine Wave')
ax2.grid(True)

# Show the plot
```

```
plt.show()
```



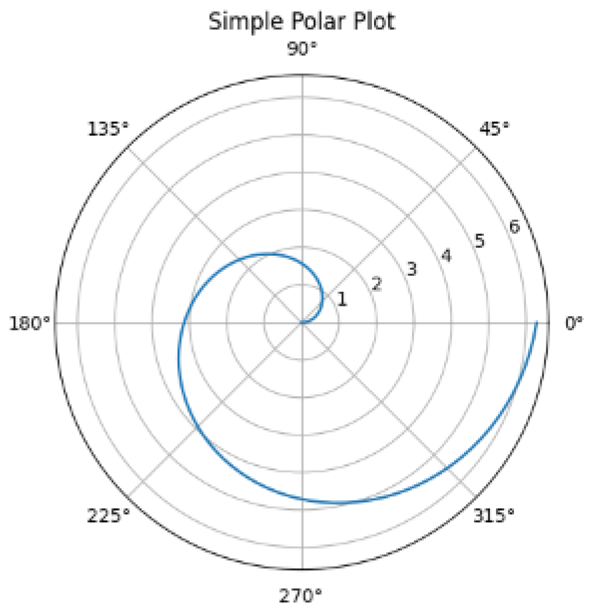
Example 5

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

# Generate example data
x = np.linspace(0, 10, 100)
series = [np.sin(x), np.cos(x), np.tan(x), np.exp(x/10)]

# Create subplots
fig, axes = plt.subplots(2, 2, figsize=(10, 8))

# Flatten the axes array
axes = axes.ravel()

# Plot each series on the respective subplot
for i, ax in enumerate(axes):
  print(i)
  print(ax)
  ax.plot(x, series[i])
  ax.set_title('Subplot ' + str(i))  # Set title for each subplot

# Adjust layout to prevent overlap of titles and labels
plt.tight_layout()

# Display the figure
plt.show()
```



Example 6: Polar Projection

```
import matplotlib.pyplot as plt
import numpy as np
#A polar plot represents data points in polar coordinates. It's useful for visualizing data that has a natural circular or angular structure, such as directions, periodic functions, or phase data.
# Create a polar subplot using projection='polar'
plt.subplot(111, projection='polar')

# Example data
theta = np.linspace(0, 2 * np.pi, 100) # Array of angles from 0 to 2*pi
print(theta)
r = theta #r is simply set to theta, which will result in a spiral plot.

plt.plot(theta, r)
plt.title('Simple Polar Plot')
plt.show()
```



Questions
1.Define Stacked Area chart
2.List the Different Parameters in Stacked Area chart.
3.Define Histogram
4.List the parameters involved in Histogram
5.Define Box Plot
6.List the difference parameters involved in Box Plot
7. Define Scatter Plot
8.Define Subplots
9.Define the different parameters involved in Subplots

Handouts for Session 40: Tight Layout, Radar Chart ,Grid Spec, Images: Basic Image Operations, Writing Mathematical Expressions

5.7.2 Tight Layout

The plt.tight_layout() adjusts subplot parameters (primarily padding between the Figure edge and the edges of subplots, and padding between the edges of adjacent subplots) so that the subplots fit well in the Figure.

If you do not use plt.tight_layout(), subplots might overlap:

```
fig, axes = plt.subplots(2, 2)
axes = axes.ravel()
for i, ax in enumerate(axes):
        ax.plot(series[i])
        ax.set_title('Subplot ' + str(i))
```

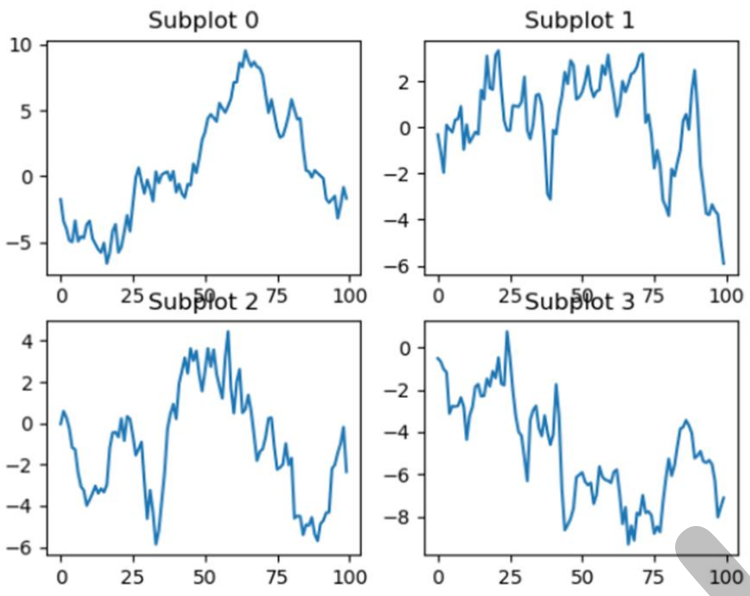


Figure 3.36: Subplots with no layout option

```
fig, axes = plt.subplots(2, 2)
axes = axes.ravel()
for i, ax in enumerate(axes):
        ax.plot(series[i])
        ax.set_title('Subplot ' + str(i))
plt.tight_layout()
```

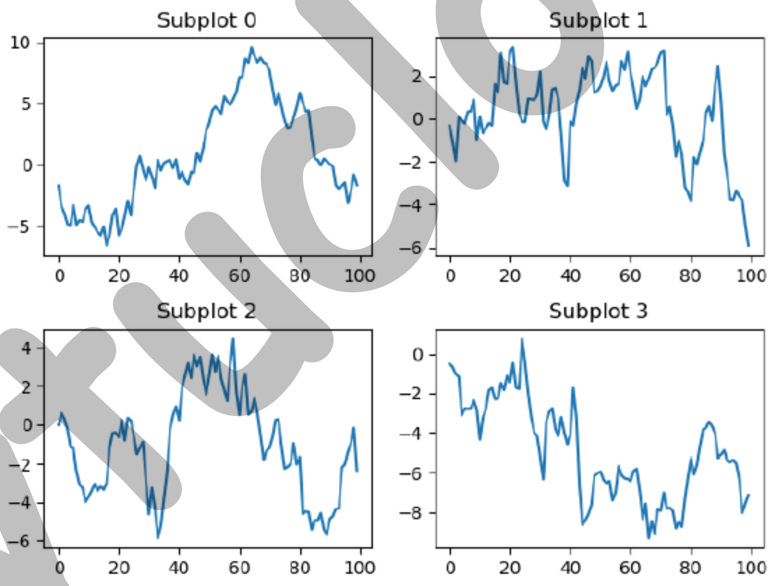


Figure 3.37: Subplots with a tight layout

5.7.3 Radar Chart

- Radar charts, also known as spider or web charts, visualize multiple variables, with each variable plotted on its own axis, resulting in a polygon.
- All axes are arranged radially, starting at the center with equal distance between each other, and have the same scale.

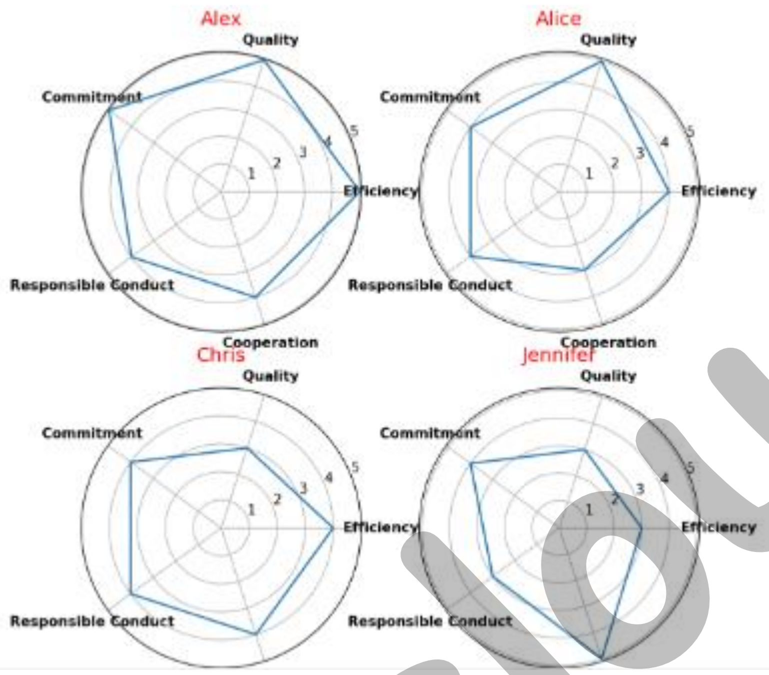
Activity: Working on Radar Charts

- As a manager of a team, you have to award a "Star Performer" trophy to an employee for the month of December. You come to the conclusion that the best way to understand the performance of your team members would be to visualize the performance of your team members in a radar chart
- Import the necessary modules and enable plotting within a Jupyter Notebook
- Create angle values and close the plot
- Create subplots with the polar projection. Set a tight layout so that nothing overlaps

```
# Import settings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# Sample data
# Attributes: Efficiency, Quality, Commitment, Responsible Conduct,Cooperation
data = pd.DataFrame({
'Employee': ['Alex', 'Alice', 'Chris', 'Jennifer'],
'Efficiency': [5, 4, 4, 3,],
'Quality': [5, 5, 3, 3],
'Commitment': [5, 4, 4, 4],
'Responsible Conduct': [4, 4, 4, 3],
'Cooperation': [4, 3, 4, 5]
})
print('data',data)
#This extracts the attribute names  except the employee
attributes = list(data.columns[1:])
print('attributes',attributes)
#This extracts the values corresponding to each attribute for all employees.
values = list(data.values[:, 1:])
print('values',values)
#This extracts the names of the employees.
employees = list(data.values[:, 0])
print('employees',employees)
#generates the angles for each attribute
angles = [n / float(len(attributes)) * 2 * np.pi for n in range(len(attributes))]
# to close the radar chart by appending the first angle to the end of the list of angles
angles += angles[:1]
values = np.asarray(values)
values = np.concatenate([values, values[:, 0:1]], axis=1)
# Create figure
plt.figure(figsize=(8, 8), dpi=150)
# Create subplots
for i in range(len(employees)):
   ax = plt.subplot(2, 2, i + 1, polar=True)
   ax.plot(angles, values[i])
   ax.set_yticks([1, 2, 3, 4, 5])
   ax.set_xticks(angles[:-1])  # Remove the duplicate last angle
```

```
    ax.set_xticklabels(attributes, fontsize=10, fontweight='bold')  # Adjust font size and weight
    ax.set_title(employees[i], fontsize=14, color='r')
# Show plot
plt.show()
```

```
data    Employee  Efficiency  Quality  Commitment  Responsible Conduct  Cooperation
0       Alex         5         5         5                  4             4
1      Alice         4         5         4                  4             3
2      Chris         4         3         4                  4             4
3   Jennifer         3         3         4                  3             5
attributes ['Efficiency', 'Quality', 'Commitment', 'Responsible Conduct', 'Cooperation']
values [array([5, 5, 5, 4, 4], dtype=object), array([4, 5, 4, 4, 3], dtype=object), array([4, 3, 4, 4, 4], dtype=object), array([3, 3, 4, 3, 5], dtype=object)]
employees ['Alex', 'Alice', 'Chris', 'Jennifer']
```



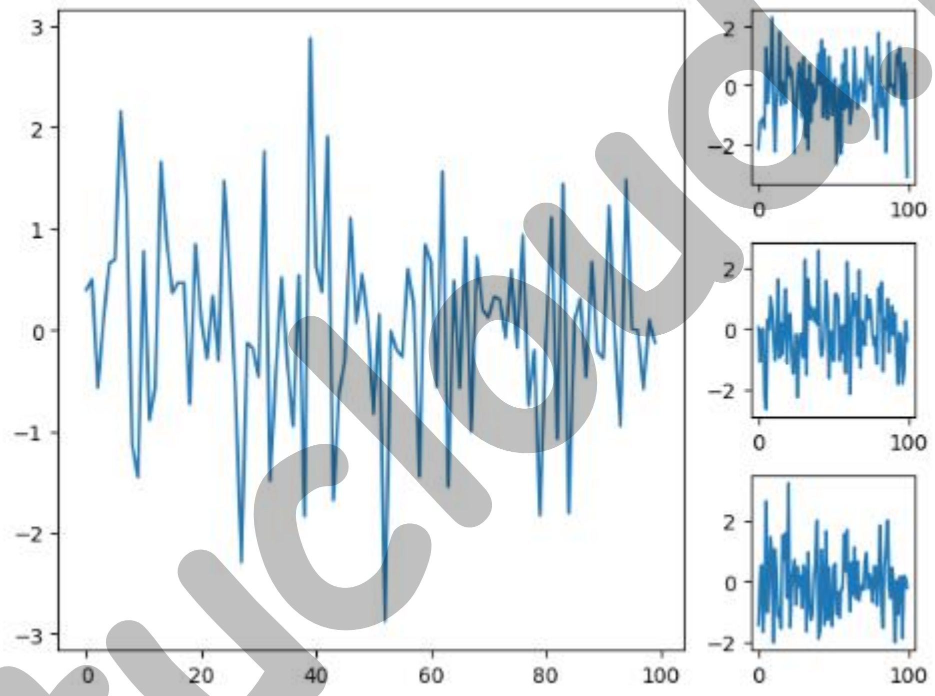
5.7.4 GridSpec

- The **matplotlib.gridspec.GridSpec(nrows, ncols)** function specifies the geometry of the grid in which a subplot will be placed.
- For example, you can specify a grid with three rows and four columns.
- As a next step, you have to define which elements of the gridspec are used by a subplot; elements of a gridspec are accessed in the same way as NumPy arrays.
- You could, for example, only use a single element of a gridspec for a subplot and therefore end up with 12 subplots in total.
- Another possibility, as shown in the following example, is to create a bigger subplot using 3x3 elements of the gridspec and another three subplots with a single element each.

Example:

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

# Dummy data for series (Replace this with your actual data)
```

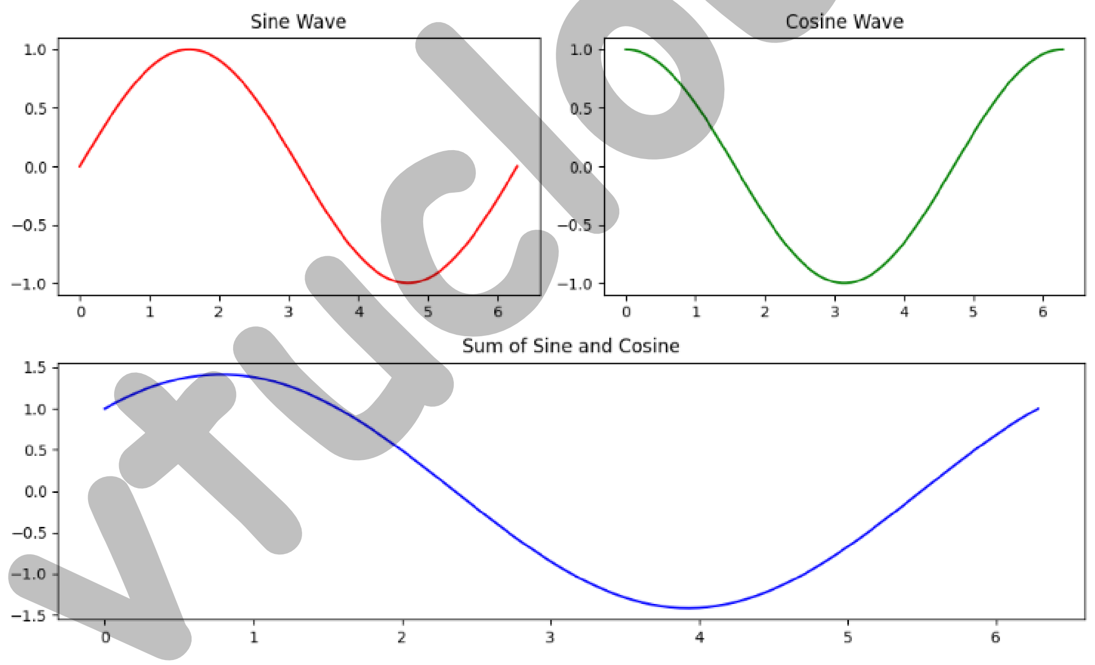
```
series = [np.random.randn(100) for _ in range(4)]

gs = gridspec.GridSpec(3, 4)
ax1 = plt.subplot(gs[:3, :3])
ax2 = plt.subplot(gs[0, 3])
ax3 = plt.subplot(gs[1, 3])
ax4 = plt.subplot(gs[2, 3])

ax1.plot(series[0])
ax2.plot(series[1])
ax3.plot(series[2])
ax4.plot(series[3])

plt.tight_layout()
plt.show()
```



Example 2:

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

# Sample data
x = np.linspace(0, 2 * np.pi, 100)
y1 = np.sin(x)
y2 = np.cos(x)
#create a GridSpec object, which is useful for creating subplots that span multiple rows and columns in a grid layout.
# Create a GridSpec object with 2 rows and 2 columns
gs = gridspec.GridSpec(2, 2)
```

```
# Create subplots using indexing from GridSpec
plt.figure(figsize=(10, 6))

# First subplot (top left)
ax1 = plt.subplot(gs[0, 0])
ax1.plot(x, y1, color='r')
ax1.set_title('Sine Wave')

# Second subplot (top right)
ax2 = plt.subplot(gs[0, 1])
ax2.plot(x, y2, color='g')
ax2.set_title('Cosine Wave')

# Third subplot (bottom, spanning two columns)
ax3 = plt.subplot(gs[1, :])
ax3.plot(x, y1 + y2, color='b')
ax3.set_title('Sum of Sine and Cosine')

# Adjust layout
plt.tight_layout()

# Show plot
plt.show()
```



5.8 Images

If you want to include images in your visualizations or work with image data, Matplotlib offers several functions for you. In this section, we will show you how to load, save, and plot images with Matplotlib.

5.8.1 Basic Image operations

i)Loading Images

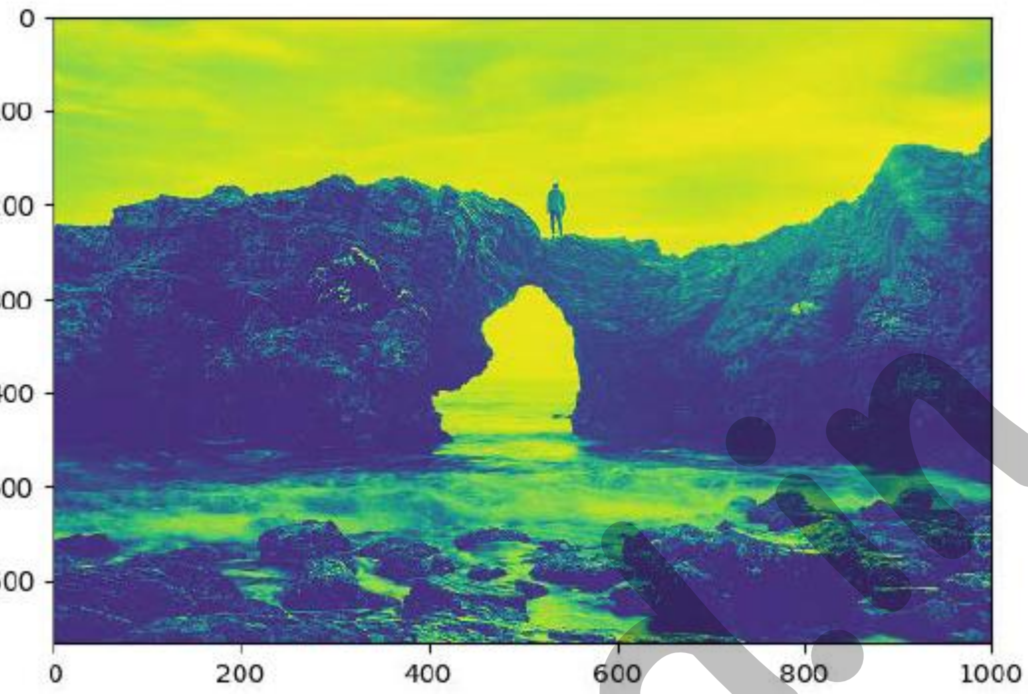
- If you encounter image formats that are not supported by Matplotlib, we recommend using the Pillow library to load the image.
- In Matplotlib, loading images is part of the image submodule. We use the alias mpimg for the submodule, as follows:
  import matplotlib.image as mpimg
- The mpimg.imread(fname) reads an image and returns it as a numpy.array object.
- For grayscale images, the returned array has a shape (height, width), for RGB images (height, width, 3), and for RGBA images (height, width, 4). The array values range from 0 to 255
- We can also load the image in the following manner
  img_filenames = os.listdir('images')
  imgs = [mpimg.imread(os.path.join('images', img_filename))for img_filename in img_filenames]
- The os.listdir() method in Python is used to get the list of all files and directories in the specified directory and then the os.path.join() function is used to join one or more path components intelligently.

ii) Saving Images

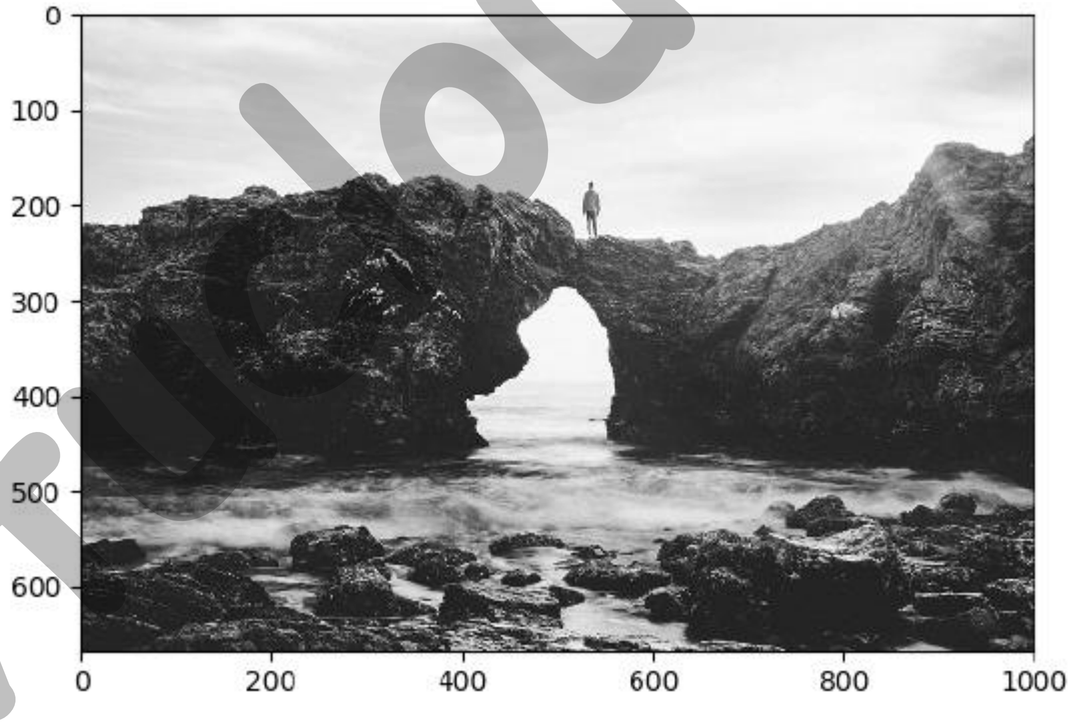
- The mpimg.imsave(fname, array) saves a numpy.array object as an image file.
- If the format parameter is not given, the format is deduced from the filename extension.
- With the optional parameters vmin and vmax, the color limits can be set manually.
- For a grayscale image, the default for the optional parameter, cmap, is 'viridis'; you might want to change it to 'gray'.

iii) Plotting a Single Image

- The plt.imshow(img) displays an image and returns an AxesImage object.
- For grayscale images with shape (height, width), the image array is visualized using a colormap. The default colormap is 'viridis'.
- To actually visualize a grayscale image, the colormap has to be set to 'gray' (that is, plt.imshow(img, cmap='gray').
- Values for grayscale, RGB, and RGBA images can be either float or uint8, and range from [0…1] or [0…255], respectively.
- To manually define the value range, the parameters vmin and vmax must be specified.
- A visualization of an RGB image is shown in the following figures:



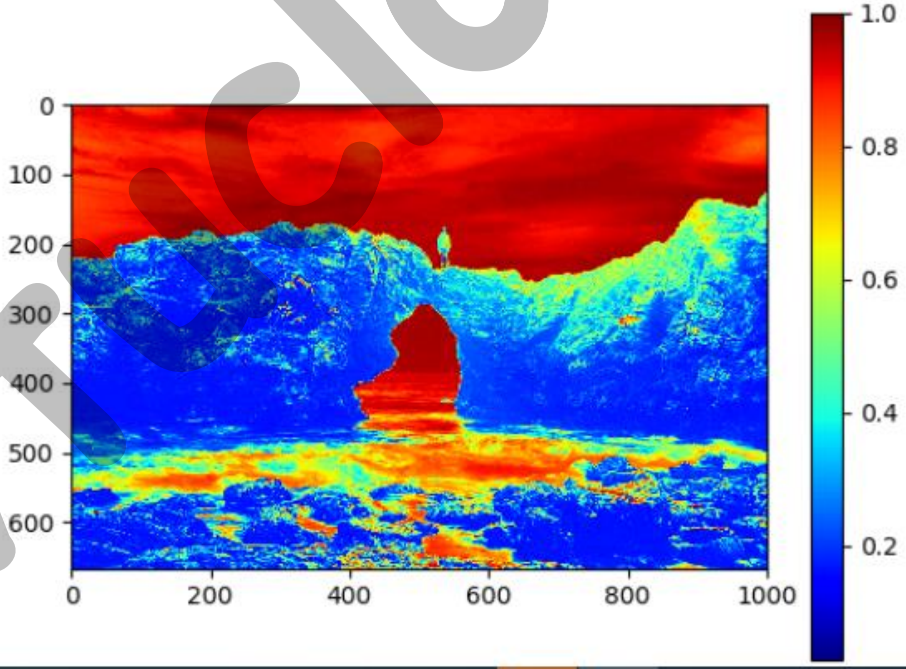
- The following figure shows a grayscale image with a gray colormap:



- The following figure shows an RGB image:

- Sometimes, it might be helpful to get an insight into the color values. We can simply add a color bar to the image plot. It is recommended to use a colormap with high contrast—for example, jet:

plt.imshow(img, cmap='jet')
plt.colorbar()



- Another way to get insight into the image values is to plot a histogram.
- To plot the histogram for an image array, the array has to be flattened using numpy.ravel:

```
plt.hist(img.ravel(), bins=256, range=(0, 1))
```

- The following diagram shows the output of the preceding code:

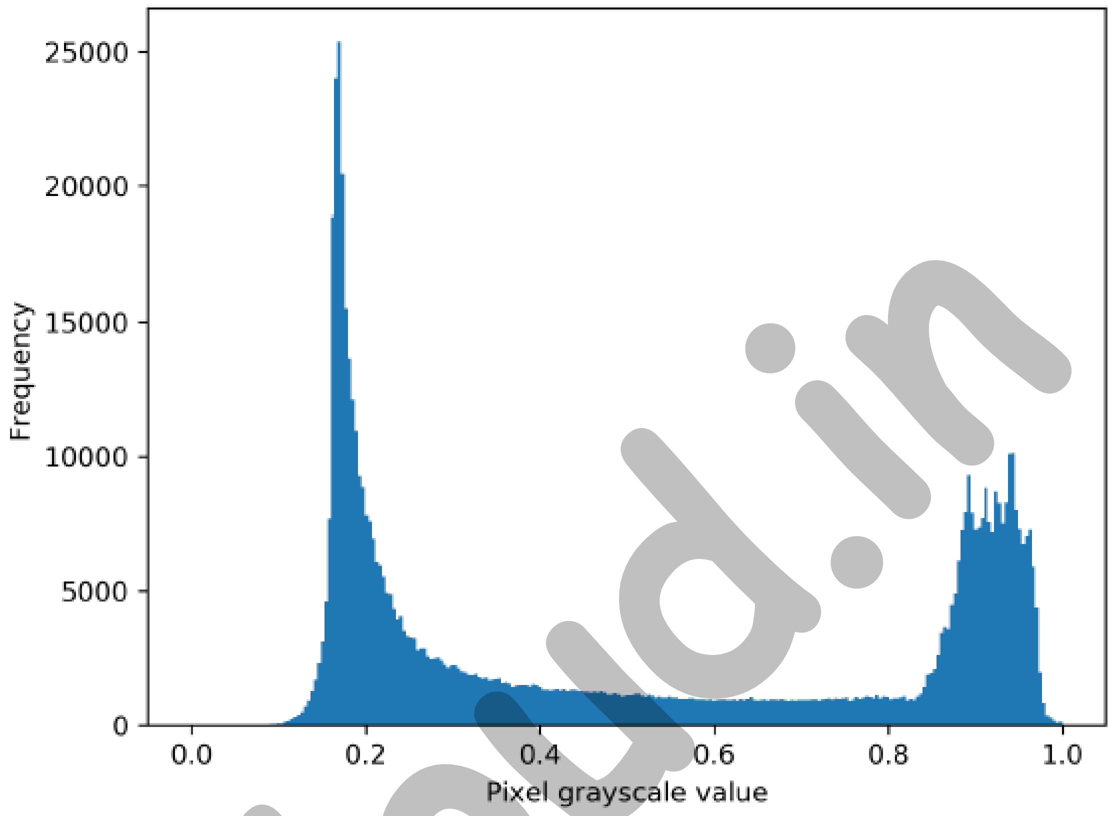


Figure 3.45: Histogram of image values

iv)Plotting Multiple Images in a Grid

To plot multiple images in a grid, we can simply use **plt.subplots** and plot an image per **Axes**:

```
fig, axes = plt.subplots(1, 2)
for i in range(2):
    axes[i].imshow(imgs[i])
```

The result of the preceding code is shown in the following diagram:

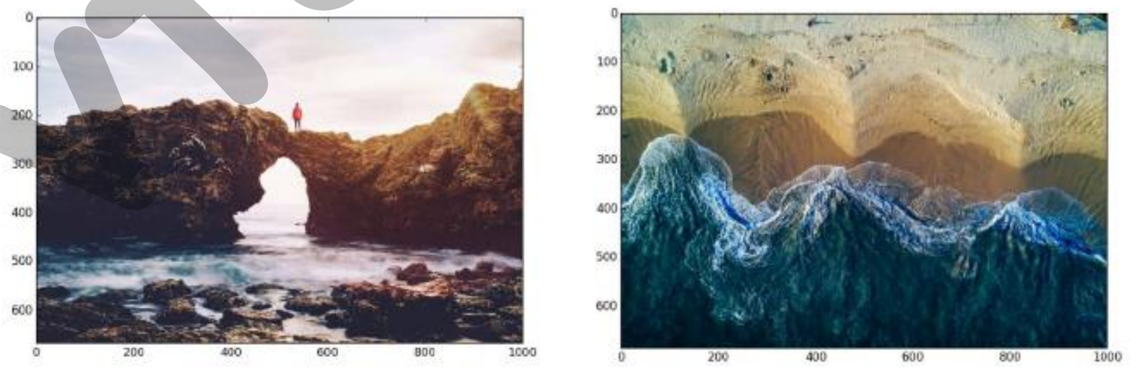


Figure 3.46: Multiple images within a grid

In some situations, it would be neat to remove the ticks and add labels. **axes.set_ xticks([])** and **axes.set_yticks([])** remove x-ticks and y-ticks, respectively. **axes.set_xlabel('label')** adds a label:

```
fig, axes = plt.subplots(1, 2)
labels = ['coast', 'beach']
for i in range(2):
    axes[i].imshow(imgs[i])
    axes[i].set_xticks([])
    axes[i].set_yticks([])
    axes[i].set_xlabel(labels[i])
```

The result of the preceding code is shown in the following diagram:

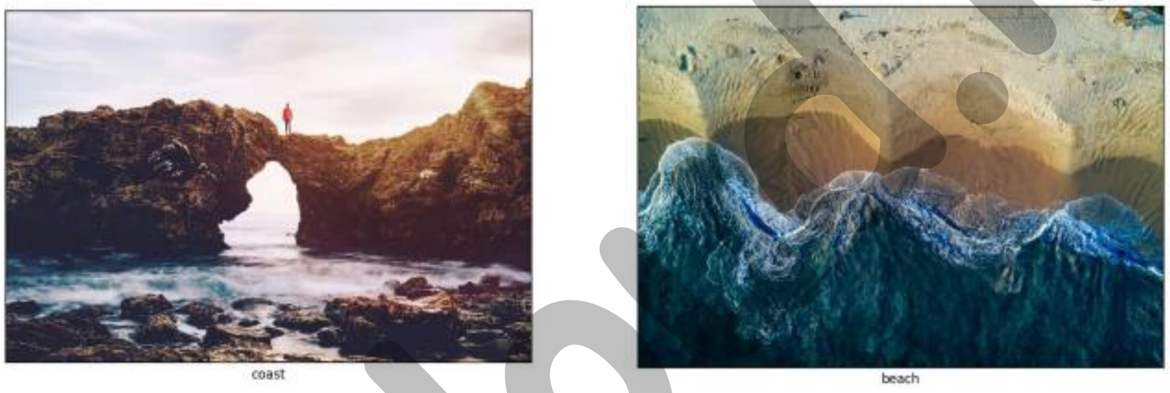


Figure 3.47: Multiple images with labels

Activity - Plotting Multiple Images in a Grid

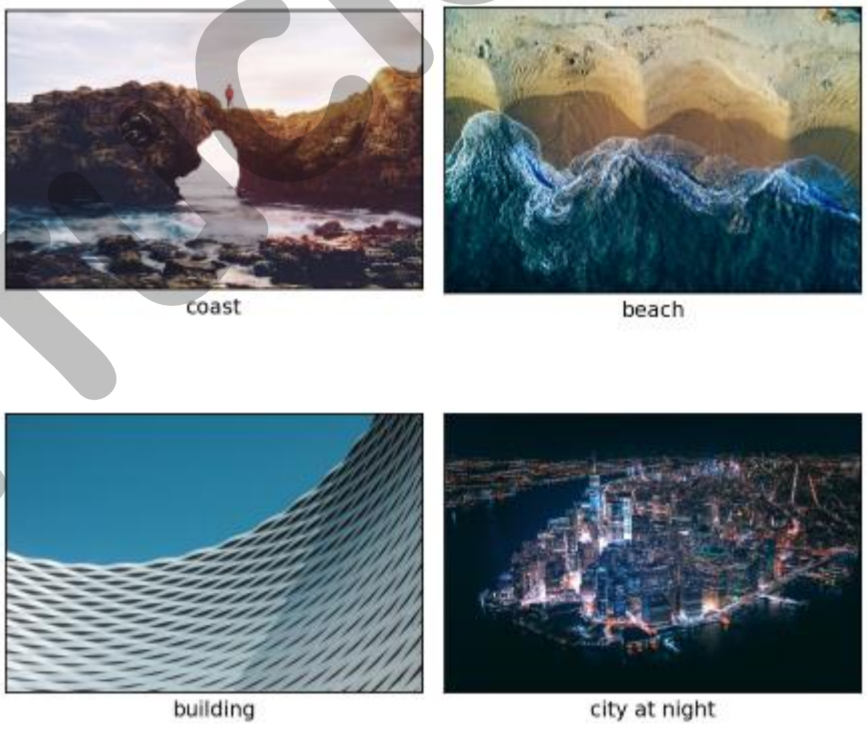
- Plot images in a grid. You are a developer in a social media company. Management has decided to add a feature that helps the customer to upload images in a 2x2 grid format. Develop some standard code to generate grid formatted images and add this new feature to your company's website.
- The following are the steps to perform:
- Import the necessary modules and enable plotting within a Jupyter Notebook.
- Load all four images from the Datasets subfolder.
- Visualize the images in a 2x2 grid. Remove the axes and give each image a label.
- The expected output should be as follows:

Solution:

```
# Import statements
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Load images
```

```
img_filenames = sorted(os.listdir('/content/drive/MyDrive/Colab
Notebooks/images'))
imgs = [mpimg.imread(os.path.join('/content/drive/MyDrive/Colab
Notebooks/images', img_filename)) for img_filename in img_filenames]

# Create subplot
fig, axes = plt.subplots(2, 2)
fig.set_size_inches(6, 6)
fig.set_dpi(150)
axes = axes.ravel()

# Specify labels
labels = ['coast', 'beach', 'building', 'city at night']

# Plot images
for i in range(len(imgs)):
    axes[i].imshow(imgs[i])
    axes[i].set_xticks([])
    axes[i].set_yticks([])
    axes[i].set_xlabel(labels[i])

plt.tight_layout()
plt.show()
```

Output:



5.9 Writing Mathematical Expressions

- In case you need to write mathematical expressions within the code, Matplotlib supports TeX, one of the most popular typesetting systems, especially for typesetting mathematical formulas.
- You can use it in any text by placing your mathematical expression in a pair of dollar signs.
- There is no need to have TeX installed since Matplotlib comes with its own parser.
- An example of this is given in the following code:

```
plt.xlabel(,$x$')
plt.ylabel('$\cos(x)$')
```

TeX examples:

- `'$\alpha_i>\beta_i$'` produces $\alpha_i > \beta_i$.
- `'$\sum_{i=0}^\infty x_i$'` produces $\sum_{i=0}^\infty x_i$.
- `'$\sqrt[3]{8}$'` produces $\sqrt[3]{8}$.
- `'$\frac{3 - \frac{x}{2}}{5}$'` produces $\frac{3-\frac{x}{2}}{5}$.

Questions:

1.Define Radar Chart.

2.list the parameters involved in Radar Chart

3.Define gridspec.

4.Define image

5.List the different image operations

6.State the syntax to save the image

7.List the different ways to write the mathematical expressions