PER SCHOLAS

Getting Started With Data Visualization by Using Python

Lesson 2

Getting Started with Data Visualization by Using Python

Introduction

In this Lesson, we will introduce the Matplotlib library and the Pyplot module. This will be followed by an exploration and demonstration of graph or chart styling, and you will learn how to create professional looking line charts.

Learning Objectives

By the end of this lesson, learners will be able to:

- Describe fundamental concepts in data analytics.
- Choose the right visual representation for different types of data.
- Explain how to design visuals that effectively communicate your message and how to leverage interactivity to engage and empower your audience.
- Identify common types of data visualization and their uses.
- Effectively communicate insights derived from the line plot.
- Use basic plot elements like labels, titles, and legends.
- Customize line styles, markers, and colors in a visualization.
- Manipulate subplots using Python.
- Select appropriate line styles based on specific data and visualization goals.

Prerequisites

Learners need to be proficient in Python programming and Python Pandas. You must have Python and Jupyter Notebook installed and configured on your machine, or you must be able to access a cloud-based notebook.

PER SCHOLAS

Table of Contents

PER SCHOLAS

Python: Your Gateway to Data Visualization

Python provides various libraries that come with different features for visualizing data. All these libraries come with different features and can support various types of graphs. Some Python popular libraries for Data Visualization are:

- **Matplotlib**
- **Seaborn**
- **Plotly**
- **PyWaffle**
- **Bokeh**
- **Altair**
- **ggplot**
- **Holoviews**
- **Folium**

Overview of Matplotlib (1 of 2)

- **Matplotlib** is a widely-used Python library **for creating various charts and plots**. It was designed to closely resemble MATLAB, a proprietary programming language developed in the 1980s.

- **Matplotlib** integrates well with other libraries and frameworks such as NumPy, Pandas, Seaborn, and Plotly.

- By leveraging the power of Matplotlib, you can transform your data into compelling visualizations that inform, engage, and empower effective communication.

PER SCHOLAS

Overview of Matplotlib (2 of 2)

Matplotlib makes easy things easy, and hard things possible.

Matplotlib:

- Creates publication quality plots.
- Makes interactive figures that can zoom, pan, and update.
- Customizes visual style and layout.
- Exports to many file formats.
- Embeds in JupyterLab and Graphical User Interfaces.
- Uses a rich array of third-party packages built on Matplotlib.

Click here for learn more about Matplotlib plotting.

Key Features of Matplotlib

- **Extensive Customization:** Matplotlib offers extensive customization options, allowing users to control every aspect of their visualizations. From modifying colors, labels, and legends, to adjusting axes and grids, users can tailor their plots to meet specific requirements.

- **Publication-Quality Visuals:** Matplotlib produces high-quality, publication-ready visualizations suitable for research papers, reports, and presentations.

- **Open-source and free:** Being free and open-source software, Matplotlib is readily available for anyone to use and contribute to.

PER SCHOLAS

Installing Matplotlib

Open a Terminal or Command Prompt: Depending on your operating system (Windows, macOS, Linux), open a terminal or command prompt window. **Use the following command to install Matplotlib**

```
pip install matplotlib
```

For Jupyter NoteBook and Google Colab Users:

```
!pip install matplotlib
```

For Conda Users: Users who prefer to use Conda to install Matplotlib, can use the command below:

```
conda install matplotlib
```

PER SCHOLAS

Essential Packages of Matplotlib

This set of code imports essential libraries for data analysis and visualization in Python:

```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline
```

Overview of Matplotlib's Pyplot Module

- Pyplot is a Matplotlib module that provides simple functions for adding plot elements such as lines, images, text, etc. to the axes in the current figure.
- Most of the Matplotlib utilities lie under the **pyplot** submodule, and are usually imported under the **'plt'** alias:

```
import matplotlib.pyplot as plt
```

- **The plt.plot() function is used to draw points (markers) in a diagram.**
  - By default, the **plot()** function draws a line from point to point.
  - The function takes parameters for specifying points in the diagram.
  - Parameter 1 is an array containing the points on the x-axis.
  - Parameter 2 is an array containing the points on the y-axis.
  - If we need to plot a line from (1, 3) to (8, 10), we have to pass two arrays [1, 8] and [3, 10] to the plot function.
- **If you are using Matplotlib from within a script, the plt.show() function is your friend. The plt.show() starts an event loop, looks for all currently active figure objects, and opens one or more interactive windows that display your figure or figures.**

PER SCHOLAS

Matplotlib - functions()

These are the functions or actions that the objects in Matplotlib can perform such as plotting data, adding labels, adding title, saving figures, etc.

1. plot(): Plots data on the axes.
2. show(): Displays your figure or figures.
3. scatter(): Creates a scatter plot of data points.
4. bar(): Generates vertical bar plots.
5. barh(): Generates horizontal bar plots.
6. hist(): Creates histograms of data.
7. imshow(): Displays images.
8. contour(): Draws contour lines.
9. annotate(): Adds annotations (text labels) to a plot.
10. savefig(): Saves the current figure to a file.

Functions can be called using the dot notation such as plt.plot(x, y) or fig.savefig('filename.png'), where plt is the pyplot module and fig is a figure object1..

Click here for learn more about Matplotlib functions.

PER SCHOLAS

Matplotlib - Properties

These are the characteristics or properties of the objects in Matplotlib. They can be set or modified using keyword arguments.

1. **Figure:** The top-level container for all elements of a plot. It represents the entire figure or window where one or more plots can be drawn.
2. **Axes:** The area within the figure where data is plotted. Each figure can contain one or more axes objects, which represent individual plots or subplots.
3. **Axis:** The x-axis and y-axis of a plot. Axes objects contain Axis objects responsible for rendering ticks, tick labels, and axis labels.
4. **Line:** The line or marker elements representing data points in a plot. These can be customized with properties like color, linewidth, line style, etc.
5. **Marker:** Symbols used to indicate data points on a plot. Markers can be customized in terms of shape, size, color, etc.
6. **Title:** A text element that provides a title for a plot.
7. **Labels:** Text elements that label the x-axis and y-axis.
8. **Legend:** A box containing labels describing the meaning of the different elements in a plot.

Click here for learn more about Matplotlib properties

PER SCHOLAS

Matplotlib - Attributes

Attributes are the variables or data that the objects in Matplotlib store such as the x and y data, the figure number, the axes title, etc.

- **Figure attributes:**
  - figsize: Specifies the width and height of the figure in inches.
  - dpi: Sets the dots per inch (resolution) of the figure.
  - facecolor: Sets the background color of the figure.
  - subplotpars: Controls the spacing between subplots.
- **Axes attributes:**
  - title: The title of the plot.
  - xlabel, ylabel: Labels for the x-axis and y-axis, respectively.
  - xlim, ylim: Sets the limits for the x-axis and y-axis, respectively.
  - xscale, yscale: Sets the scale of the x-axis and y-axis, respectively (linear, log, etc.).
- **Line attributes:**
  - color: Sets the color of the line or marker.
  - linestyle: Sets the style of the line (solid, dashed, etc.).
  - linewidth: Sets the width of the line.
  - marker: Sets the marker symbol.
  - markersize: Sets the size of the markers.

Approaches To Create Plots (1 of 2)

In Matplotlib, there are two primary ways to create plots:

1. **State-based approach (pyplot interface).**
2. **Object-oriented approach.**

State-based approach (pyplot interface):

- This is the familiar MATLAB-like interface where you implicitly create and manipulate figures and axes. Functions like **plt.plot(), plt.title(), plt.xlabel()**, etc., operate on the current axes (which is automatically created if it doesn't exist) and figure.
- In this approach, Matplotlib maintains an internal state that tracks the current figure and axes. If a plot function is called (e.g., **plt.plot()**), Matplotlib automatically creates a figure and axes if they do not exist, and then performs the plotting operation on the current axes.
- Functions like plt.plot(), plt.title(), plt.xlabel(), etc., directly operate on the current axes and figure.
- This approach is convenient for quick and simple plotting tasks, as it requires less code.

Approaches To Create Plots (2 of 2)

Object-oriented approach:

- This approach involves explicitly creating figures and axes objects and calling methods on them to plot data and customize the plot. It provides more control and flexibility over the plot's appearance and layout.

- In this approach, you explicitly create a figure (fig) using **plt.figure()** and then add **axes (ax)** to this figure using methods like **fig.add_subplot()** or **fig.add_axes()**.

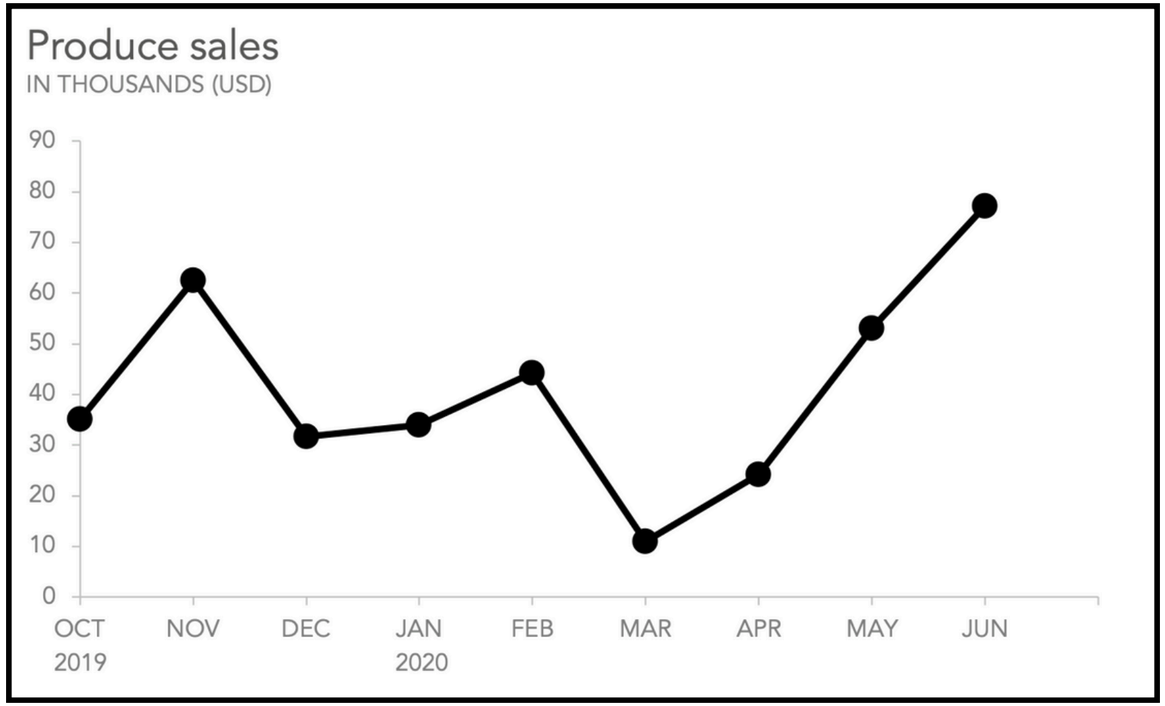
- You then call plotting methods (**ax.plot(), ax.scatter(),** etc.) and customization methods (**ax.set_title(), ax.set_xlabel(), ax.set_ylabel(),** etc.) directly on the axes object (ax).

- This approach provides more control and flexibility over the plot's appearance and layout, making it suitable for more complex or customized plots.

PER SCHOLAS

Overview of Line Plot

Line plot use lines to connect **different data points** or **series of data points**. They are helpful when creating a graph that presents trends and patterns in data, such as time series data. Some examples are changes in weather, stock prices, sales, etc.



Example 1A: Line Plot graph using matplotlib.pyplot

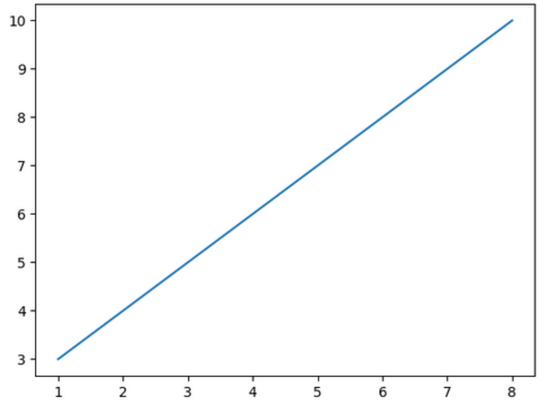
In this example, we will use the **State-based approach**:

Example One: With Line

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

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints)
plt.show()
```

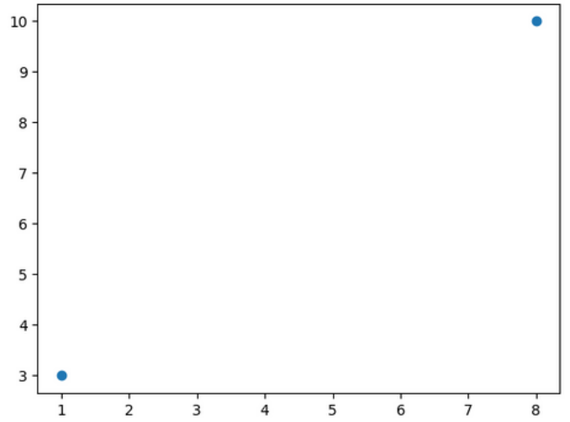


Example Two: Without Line

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

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints, '0')
plt.show()
```



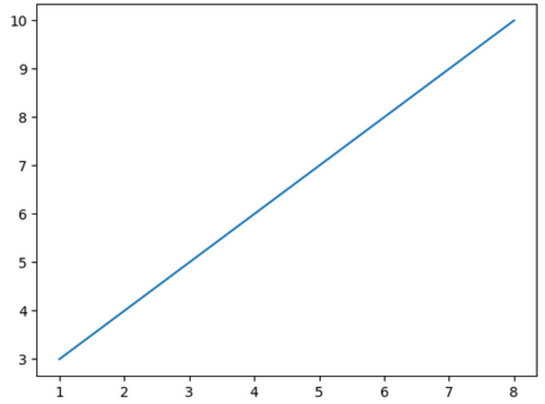
PER SCHOLAS

Example 1B: Line Plot graph using matplotlib.pyplot

In this example, we will use the **Object-oriented approach:**

Example One: With Line

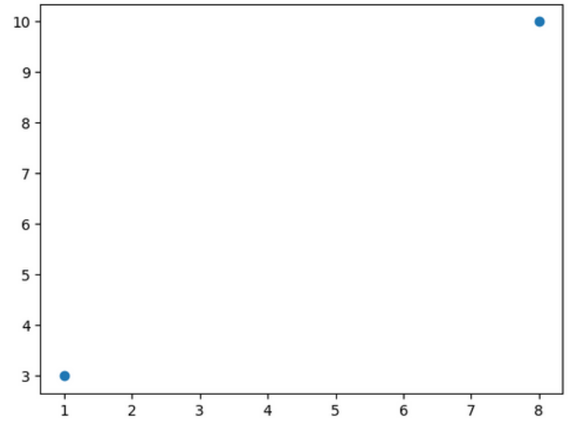
```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([1, 8])
ypoints = np.array([3, 10])
# Using object-oriented approach
fig, ax = plt.subplots() # Create figure and axes
objects
ax.plot(xpoints, ypoints) # Plot data
plt.show() # Display the plot
```



Example Two: Without Line

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

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])
# Using object-oriented approach
fig, ax = plt.subplots() # Create figure and axes objects
ax.plot(xpoints, ypoints, 'o') # Plot data
plt.show() # Display the plot
```



PER SCHOLAS

Matplotlib Pyplot: Plotting Line Styles

- Plotting line styles in Matplotlib refers to the ability to customize the appearance of lines in plots. Matplotlib offers various line styles, including solid lines, dashed lines, dotted lines, and more. These styles can be adjusted to control the visual representation of data in plots, allowing users to convey information effectively.
- We can use the **`linestyle` keyword argument**, or shorter **`ls,`** to change the style of the plotted line:

Basic Line Styles:

- **'-' (solid): The default line style, represented by a continuous solid line.**
- **'--' (dashed): A line with evenly spaced dashes.**
- **':' (dotted): A line with evenly spaced dots.**
- **dashdot: A line with alternating dashes and dots.**

Click here to access additional comprehensive information regarding Line Styling.

Click here for the example - Guided Lab - 344.2.1 - Plotting Line Styles.

PER SCHOLAS

Matplotlib Pyplot: Markers

- You can use the keyword argument **marker** to emphasize each data point with a specified marker.

Marker Reference: You can choose any of these markers:

'o' Circle	'H' Hexagon
'*' Star	'h' Hexagon
'.' Point	'v' Triangle Down
',' Pixel	'^' Triangle Up
'x' X	'<' Triangle Left
'X' X (filled)	'>' Triangle Right
'+' Plus	'1' Tri Down
'P' Plus (filled)	'2' Tri Up
's' Square	'3' Tri Left
'D' Diamond	'4' Tri Right
'd' Diamond (thin)	'|' Vline
'p' Pentagon	'_' Hline

PER SCHOLAS

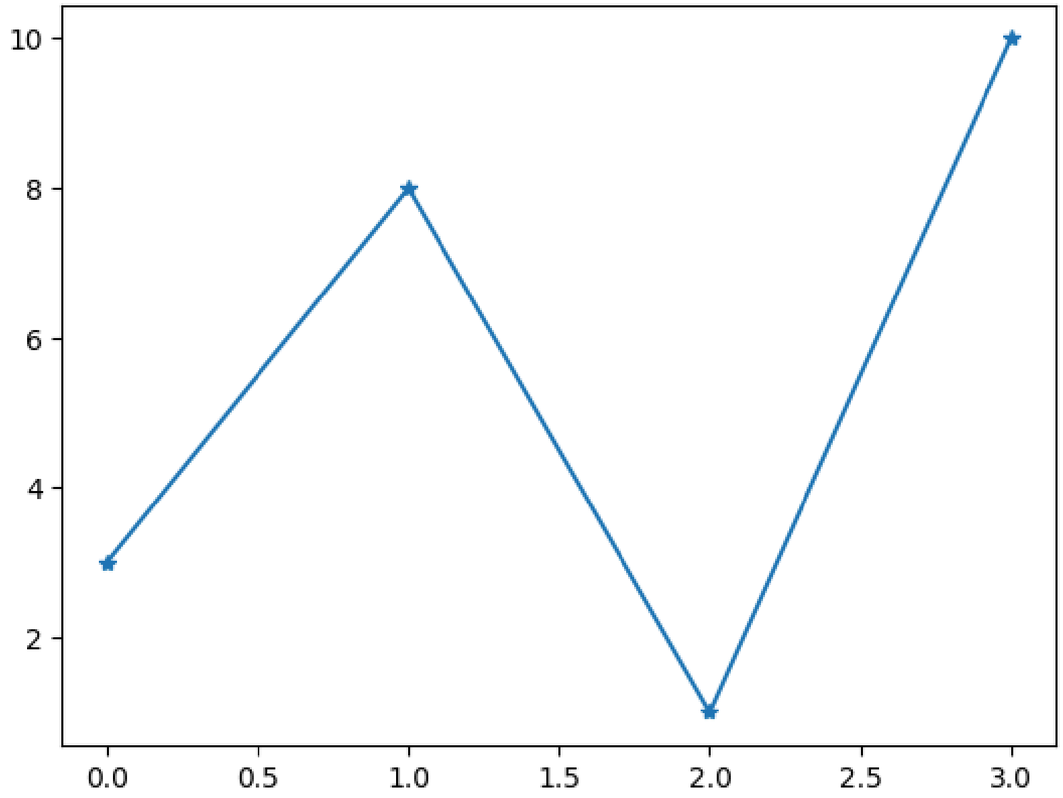
Example: Matplotlib Pyplot: Markers

This Python code uses the Matplotlib library to create a simple line plot, and marks each point on the plot with an **asterisk (*)**.

```
#Mark each point with a circle:
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = '*')
plt.show()
```



Click here for more example - Guided lab - 344.2.2 - Matplotlib Pyplot Markers and LineStyle.

Matplotlib Pyplot: “Labels” and “Title”

- As we discussed in previous lectures, visualization (graphs) should be self-explanatory, but having a title on the graph, labels on the axis, and a legend that explains what each line is can be necessary.
    - With Pyplot, you can use the **xlabel()** and **ylabel()** functions to set a **label** for the x- and y-axis.
    - With Pyplot, you can use the **title()** function to set a title for the plot.

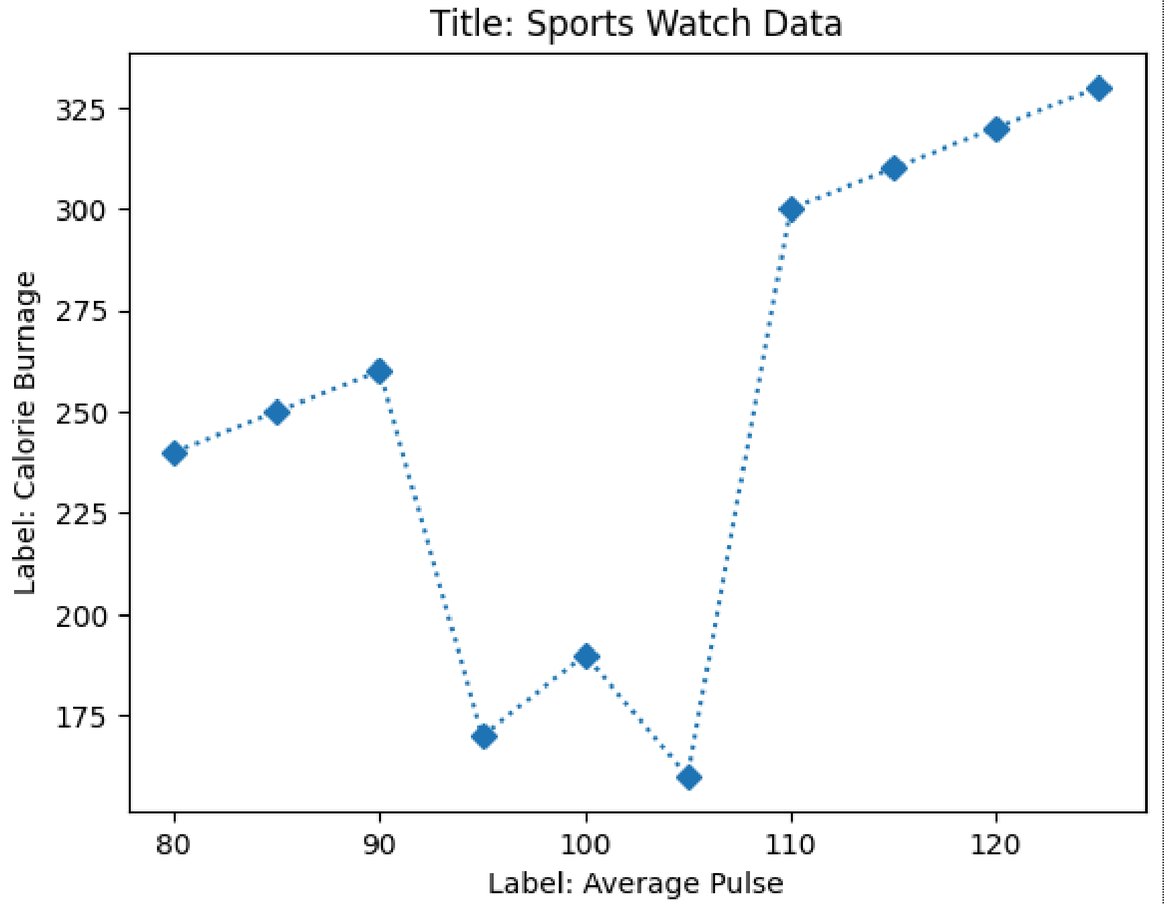
Example 1A- Matplotlib Pyplot: “Labels” and “Title”

In this example, we will create a simple line plot visualizing sports watch data. We will add a **title** and **label** to x-axis and y-axis, and mark each point on the plot with a ``diamond.``

In this example we will use **State-based approach:**

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([80, 85, 90, 95, 100, 105, 110,
115, 120, 125])
y = np.array([240, 250, 260, 170, 190, 160, 300,
310, 320, 330])

plt.plot(x, y, marker = 'D', linestyle='dotted')
plt.title("Title: Sports Watch Data")
plt.xlabel("Label: Average Pulse" )
plt.ylabel("Label: Calorie Burnage")
plt.show()
```



PER SCHOLAS

Example 1B- Matplotlib Pyplot: “Labels” and “Title”

In this example, we will create a simple line plot visualizing sports watch data. We will add a **title** and **label** to x-axis and y-axis, and mark each point on the plot with a ``diamond.``

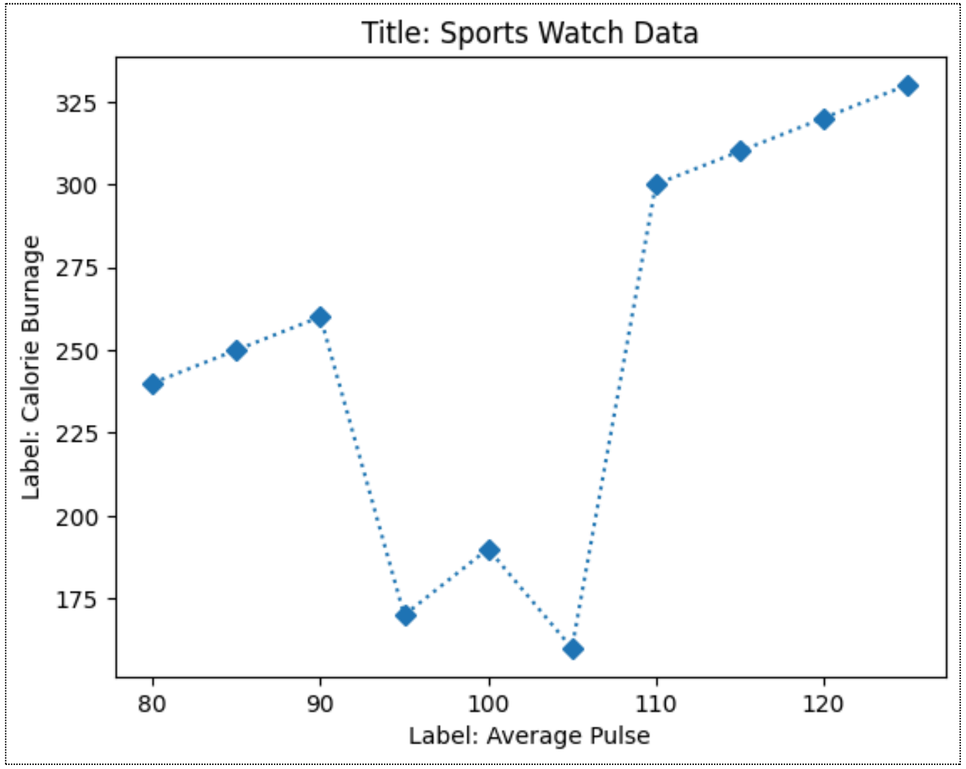
In this example we will use Object-oriented approach:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 170, 190, 160, 300, 310, 320, 330])

# Create figure and axes objects
fig, ax = plt.subplots()

# Plot the data using ax.plot()
ax.plot(x, y, marker='D', linestyle='dotted')

# Set title, xlabel, and ylabel using ax.set_XXX() methods
ax.set_title("Title: Sports Watch Data")
ax.set_xlabel("Label: Average Pulse")
ax.set_ylabel("Label: Calorie Burnage")
plt.show()
```



PER SCHOLAS

Matplotlib Pyplot: Color

- Matplotlib offers **Keyword Arguments (kwargs)** named **'color'** to specify colors for plot elements such as **labels**, **title**, **lines**, **markers**, and **text**.
- Basic Syntax:

```
plt.plot(x, y, color='blue') # Using predefined color names
plt.plot(x, y, color=(0.1, 0.2, 0.5)) # Using RGB tuples
plt.plot(x, y, color='#FF5733') # Using hex strings
plt.xlabel('X-axis', color='green') # Green label
plt.title('My Plot', color='green')  # Green title
```

Click here for list of colors.

Click here for the example: Guided LAB - 344.2.3 - Matplotlib Pyplot - Define Custom Color

PER SCHOLAS

Matplotlib Pyplot: Fonts

- To modify fonts in your plot, you can use the following methods:

1 - **Keyword Arguments:** Apply font properties directly to individual plot elements."

- `plt.xlabel('X-axis', fontweight='bold', fontsize=14) # Bold and larger label`
- `plt.title('My Plot', fontfamily='monospace') # Monospace font for title`

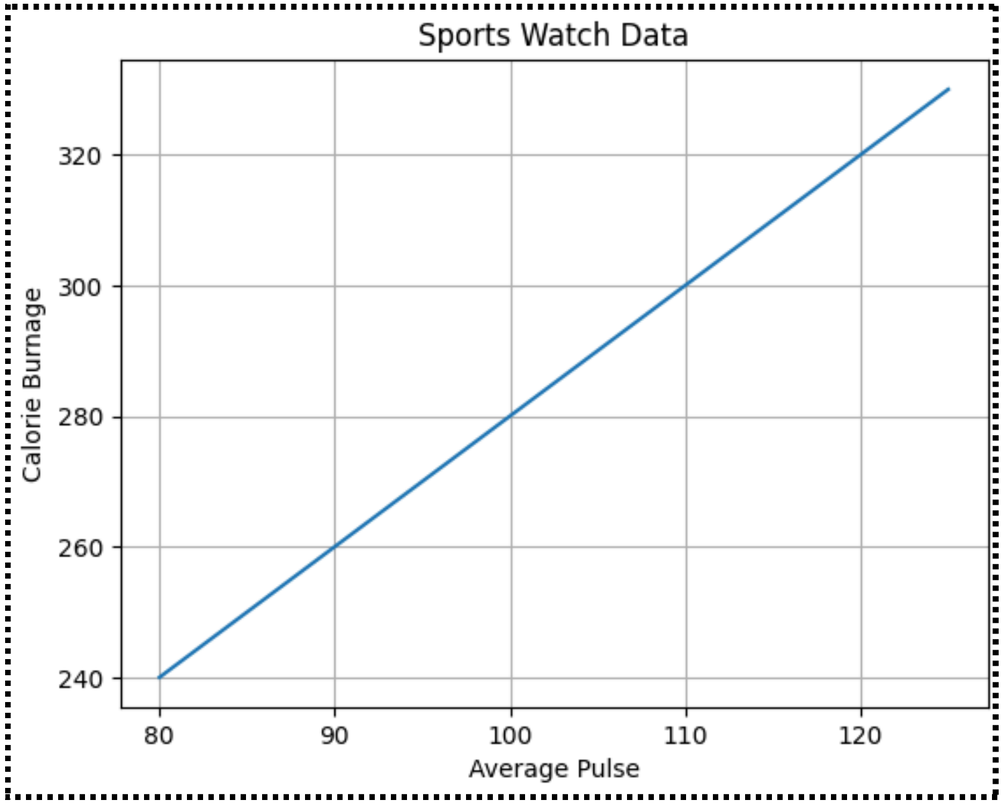
2 - plt.fontdict: We can set custom fonts for different text elements such as title, labels, and tick labels using the **fontdict** parameter in various Matplotlib functions.

Click here for the example - GLAB - 344.2.4 - Matplotlib Pyplot - Fonts Styling.

Matplotlib Pyplot: Grid Lines

- With Pyplot, you can use the **grid()** function to add grid lines to the plot.
  - The default color of the grid lines can be overridden by color argument
    - grid (color=”blue”)

Click here for the examples: Guided lab - 344.2.5 - Add grid lines in a Graph.



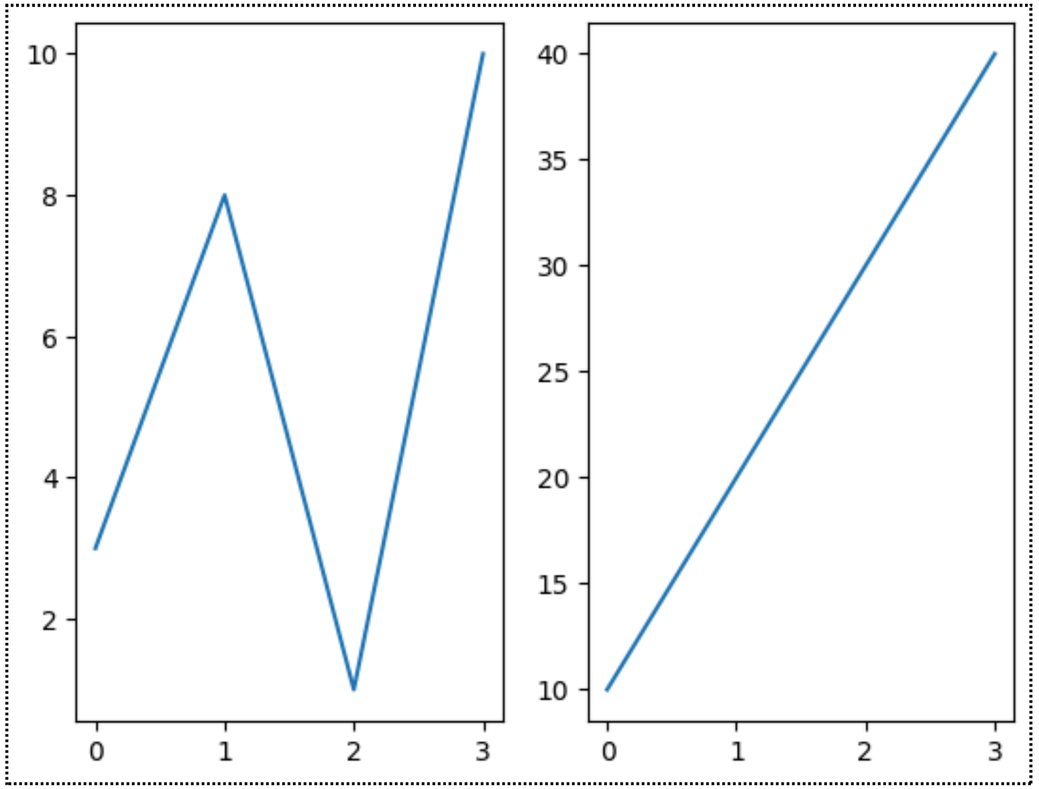
Matplotlib Pyplot: Subplots

Subplots are essentially multiple plots within a single figure. They allow you to organize and display multiple plots in a grid-like layout.

- With the **subplot()** function, you can draw multiple plots in one figure.
- The **subplot()** function returns a figure object and a multidimensional array of axes objects.

  `plt.subplot(subplot(nrows, ncols, index).`

- The **plt.tight_layout()** function is used to automatically adjust the layout of subplots within a figure.



Click here for the subplot examples: Guided Lab - 344.2.6 - Matplotlib Pyplot: Subplots

PER SCHOLAS

Matplotlib Pyplot: Control the properties of the figure

Figure size (figsize) determines the size of the figure in inches. This gives the amount of space the axes (and other elements) have inside the figure. The default figure size is (6.4, 4.8) inches in Matplotlib. A larger figure size will allow for longer texts, more axes, or more tick labels to be shown.

plt.figure() allows you to explicitly control the properties of the figure before plotting anything on it. We can also specify various parameters when calling **plt.figure()** to customize the figure, such as its size, aspect ratio, and background color.

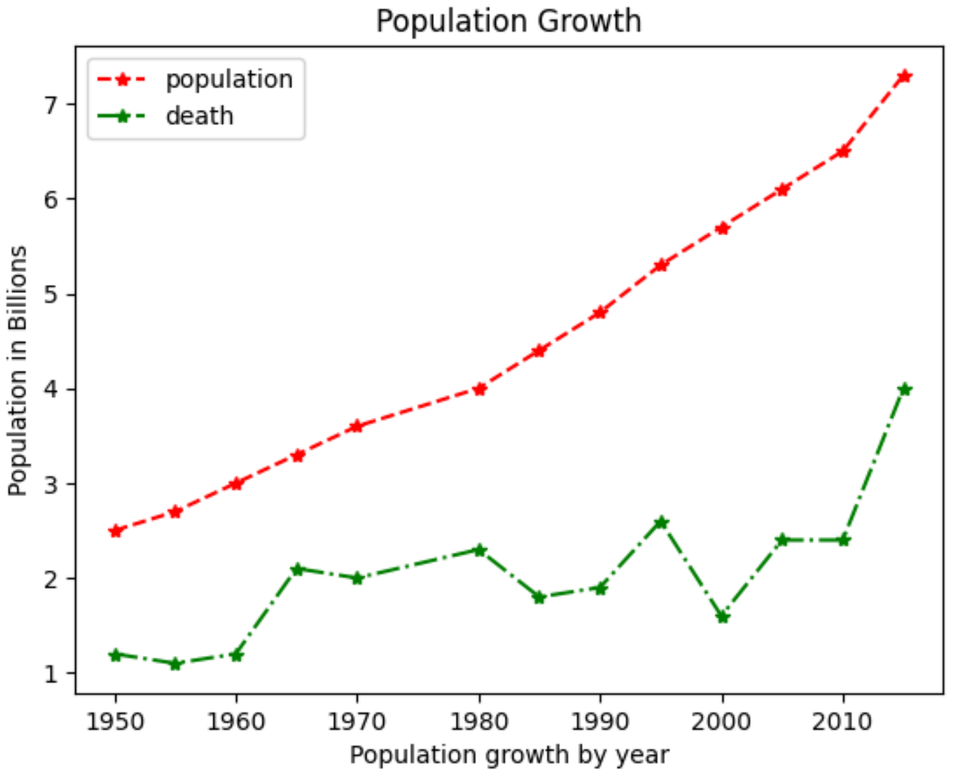
Click here for the examples: Guided Lab - 344.2.7 - Matplotlib Pyplot: Control the Properties of the Figure

PER SCHOLAS

Matplotlib Pyplot: Multiple lines and legend

Multiple Lines: If we want to create a single figure with multiple lines, we can simply call the **plot(x,y)** function multiple times.

Legend: When multiple lines are displayed within a single axes, it is often helpful to create a plot legend that labels each line type. Matplotlib provides a built-in method for quickly creating such a legend using the **plt.legend()** function. You can specify labels for each line by using the **label** keyword argument in the **plot()** function.



Click here for the subplot examples: Guided LAB - 344.2.8 - Matplotlib Pyplot: Multiple lines and line styling.ipynb

PER SCHOLAS

Hands-On Activity: Reading Data from CSV

Complete the GLAB - 344.2.9 - Exploratory Data Analysis on CSV data - Basic insights from the Data. You can find this lab on Canvas under the Assignment section.

PER SCHOLAS

Tips: How To Select Appropriate Line Styles (1 of 2)

Choosing the right line style for your data goes beyond aesthetics; it plays a crucial role in effectively communicating the trends and characteristics within your visualization. Here are some key principles to guide your selection:

Highlighting Relationships:

- **Distinguish data sets:** Use different styles for multiple data sets or groups to make them visually distinct and improve readability. For example, use solid lines for primary data, dashed lines for secondary data, and dotted lines for reference lines.
- **Compare trends:** Emphasize similarities or differences in trends with contrasting styles. Use solid lines for consistent trends, dashed lines for fluctuating trends, and dash-dot lines for complex or uncertain trends.
- **Show connections:** If you want to highlight a relationship between data sets, consider using similar styles with variations in color or thickness.

Tips: How To Select Appropriate Line Styles (2 of 2)

Clarity and Emphasis:

- **Avoid clutter:** Keep the number of line styles manageable, especially for complex plots. Too many styles can overwhelm viewers and obscure the message.
- **Prioritize clarity:** To make your visualization clear and easy to understand, use plain and simple line styles for the data that matters most. You can use more fancy or attention-grabbing styles for less important data.
- **Emphasize key points:** Use bolder lines or distinct styles to draw attention to specific trends or outliers that deserve deeper analysis.

PER SCHOLAS

Knowledge Check

1) What is Matplotlib?

2) What are some of the features of Matplotlib?

3) What are some of the major components of any graph or plot?

4) What is the main objective of data visualization?

a) To complicate data analysis.

b) To communicate insights from data effectively.

c) To obscure data patterns.

d) To reduce the need for data exploration.

Knowledge Check

5) What is the purpose of the plt.plot() function in Matplotlib?

- a) To create scatter plots.
- b) To add labels to plots.
- c) To create line graphs.
- d) To customize plot colors.

6) Which function is used to add labels and titles to Matplotlib plots?

- a) plt.labels()
- b) plt.annotate()
- c) plt.title()
- d) plt.text()

Knowledge Check

7) Which kind of visualization should you use to analyze pass rates for multiple exams over time?

- A pie chart.
- A scatter plot.
- A line chart.
- A Map graph.

Summary

- Matplotlib library is used for creating static, animated, and interactive 2D plots or figures in Python. Matplotlib offers various plot types for different data representations.
- In order to be able to use Python's Data Visualization library, we need to import the pyplot module from Matplotlib library, using the following statement: *import matplotlib.pyplot as plt, where plt is an alias or an alternative name for matplotlib.pyplot*. You can keep any alias of your choice.
- The pyplot module of matplotlib contains a collection of functions that can be used to work on a plot. The plot() function of the pyplot module is used to create a figure. A figure is the overall window where the outputs of pyplot functions are plotted.
- plot() is provided with two parameters, which indicates values for x-axis and y-axis, respectively.
- The various components of a plot are: Title, Legend, Ticks, x label, ylabel • plt.plot() is used to build a plot, where plt is an alias. • plt.show() is used to display the figure, where plt is an alias.
- Use plt.plot(x, y) to connect data points with a line.

Summary

- **Labels and title:** Use plt.xlabel(), plt.ylabel(), and plt.title() to add labels and a title to your plot.
- **Legend:** Use plt.legend() to display labels for multiple datasets plotted on the same graph.
- **Line styles, markers, and colors**: Use arguments like linewidth, marker, and color within the plotting functions to customize the appearance of lines, markers, and bars.
- **Grid:** Use plt.grid(True) to add grid lines to your plot.
- **Display the plot:** Use plt.show() to display the generated plot in a separate window.
- You can set figure size and dpi by using figure() function.

References

https://aosabook.org/en/v2/matplotlib.html
https://matplotlib.org/stable/tutorials/pyplot.html
https://handsondataviz.org/map.html
https://en.wikipedia.org/wiki/Data_and_information_visualization#Techniques
https://thepythoncodingbook.com/basics-of-data-visualisation-in-python-using-matplotlib/
https://matplotlib.org/ https://www.datacamp.com/tutorial/matplotlib-tutorial-python

PER SCHOLAS

Questions?

PER SCHOLAS

UNLOCKING POTENTIAL

CHANGING THE FACE OF TECH