National University of Computer and Emerging Sciences



Lab Manual 10

Al2002-Artificial Intelligence Lab

|  |  |
| --- | --- |
| Course Instructor | Dr. Hajra Waheed |
| Lab Instructor (s) | Hamza Ayub  Muhammad Ahmed |
| Sections | B1 - B2 |
| Semester | Spring 2023 |

Department of Computer Science

FAST-NU, Lahore, Pakistan

Table of Contents

[1](#_1fob9te) Objectives 3

[2](#_3znysh7) Task Distribution 3

[3](#_2et92p0) Matplotlib 3

[3.1](#_tyjcwt) Installation of Matplotlib 3

[3.2](#_3dy6vkm) Pyplot 4

[Example 4](#_1t3h5sf)

[3.3](#_4d34og8) Plotting x and y points 5

[Example 5](#_2s8eyo1)

[3.4](#_17dp8vu) Plotting Without Line 6

[3.5](#_3rdcrjn) Multiple Points 6

[3.6](#_26in1rg) Default X-Points 7

[3.7](#_lnxbz9) Create Labels and title for a Plot 8

[3.8](#_35nkun2) Add Grid Lines to a Plot 8

[3.9](#_1ksv4uv) Display Multiple Plots 9

[3.10](#_44sinio) Creating Scatter Plots 10

[3.11](#_2jxsxqh) Creating Bars 11

[3.12](#_z337ya) Histogram 12

[3.12.1](#_3j2qqm3) Create Histogram 13

[3.13](#_1y810tw) Creating Pie Charts 14

[3.14](#_4i7ojhp) Box Plot 15

# Objectives

After performing this lab, students shall be able to understand the following Python concepts and applications:

* Matplotlib Introduction
* Application of Matplotlib
* Min-Max Algorithm (Game theory)

# Task Distribution

|  |  |
| --- | --- |
| **Total Time** | **170 Minutes** |
| Matplotlib Introduction/Installation Guide | 20 Minutes |
| Application of Matplotlib | 25 Minutes |
| Min-Max Algorithm | 25 Minutes |
| Exercise | 90 Minutes |
| Online Submission | 10 Minutes |

# Matplotlib

Matplotlib is a low-level graph plotting library in python that serves as a visualization utility. Matplotlib was created by John D. Hunter. Matplotlib is open source and we can use it freely. Matplotlib is mostly written in python, a few segments are written in C, Objective-C and JavaScript for Platform compatibility.

## Installation of Matplotlib

If you have Python and PIP already installed on a system, then installation of Matplotlib is very easy.

Install it using this command:

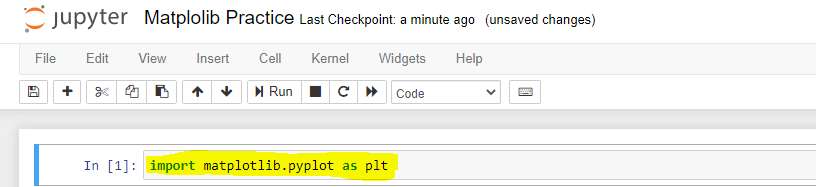


pip install matplotlib

But mostly distribution like Anaconda, Spyder have pre-installed matplotlib.

## Pyplot

Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:



import matplotlib.pyplot as plt

Now the Pyplot package can be referred to as plt.

### Example

Draw a line in a diagram from position (0,0) to position (6,250):

|  |  |
| --- | --- |
| **Code** | **Output** |
| import matplotlib.pyplot as plt import numpy as np  xpoints=np.array([0, 6]) ypoints=np.array([0, 250])  plt.plot(xpoints, ypoints) plt.show() |  |

## Plotting x and y points

The 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.

### Example

Draw a line in a diagram from position (1, 3) to position (8, 10):

|  |  |
| --- | --- |
| **Code** | **Output** |
| 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() |  |

There are many types of single lines/multiple lines that can be drawn, explore other types at: <https://www.w3schools.com/python/matplotlib_line.asp>

## Plotting Without Line

To plot only the markers, you can use *shortcut string notation* parameter 'o', which means 'rings'.

**Example**

Draw two points in the diagram, one at position (1, 3) and one in position (8, 10):

|  |  |
| --- | --- |
| **Code** | **Output** |
| import matplotlib.pyplot as plt import numpy as np  xpoints = np.array([1, 8]) ypoints = np.array([3, 10])  plt.plot(xpoints, ypoints, 'o') plt.show() |  |

There can be different type of markers, you can explore at: <https://www.w3schools.com/python/matplotlib_markers.asp>

## Multiple Points

You can plot as many points as you like, just make sure you have the same number of points in both axis.

**Example**

Draw a line in a diagram from position (1, 3) to (2, 8) then to (6, 1) and finally to position (8, 10):

|  |  |
| --- | --- |
| **Code** | **Output** |
| import matplotlib.pyplot as plt import numpy as np  xpoints = np.array([1, 2, 6, 8]) ypoints = np.array([3, 8, 1, 10])  plt.plot(xpoints, ypoints) plt.show() |  |

## Default X-Points

If we do not specify the points in the x-axis, they will get the default values 0, 1, 2, 3, (etc. depending on the length of the y-points.

So, if we take the same example as above, and leave out the x-points, the diagram will look like this:

**Example**

Plotting without x-points:

|  |  |
| --- | --- |
| **Code** | **Output** |
| import matplotlib.pyplot as plt import numpy as np  ypoints = np.array([3, 8, 1, 10, 5, 7])  plt.plot(ypoints) plt.show() |  |

The x-points in the example above are [0, 1, 2, 3, 4, 5] by default.

## Create Labels and title for a Plot

With Pyplot, you can use the xlabel() and ylabel() functions to set a label for the x- and y-axis.

**Example**

Add labels to the x- and y-axis:

|  |  |
| --- | --- |
| **Code** | **Output** |
| 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, 270,  280, 290, 300, 310, 320, 330])  plt.plot(x, y)  plt.title("Sports Watch Data") plt.xlabel("Average Pulse") plt.ylabel("Calorie Burnage")  plt.show() |  |

## Add Grid Lines to a Plot

With Pyplot, you can use the grid() function to add grid lines to the plot.

**Example**

Add grid lines to the plot:

|  |  |
| --- | --- |
| **Code** | **Output** |
| 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, 270,   280, 290, 300, 310, 320, 330])  plt.title("Sports Watch Data") plt.xlabel("Average Pulse") plt.ylabel("Calorie Burnage")  plt.plot(x, y)  **plt.grid()**  plt.show() |  |

Different type of grid can be generated, for more details see: <https://www.w3schools.com/python/matplotlib_grid.asp>

## Display Multiple Plots

With the subplots() function you can draw multiple plots in one figure:

**Example**

Draw 2 plots:

|  |  |
| --- | --- |
| **Code** | **Output** |
| import matplotlib.pyplot as plt import numpy as np  #plot 1: x = np.array([0, 1, 2, 3]) y = np.array([3, 8, 1, 10])  plt.subplot(1, 2, 1) plt.plot(x,y)  #plot 2: x = np.array([0, 1, 2, 3]) y = np.array([10, 20, 30, 40])  plt.subplot(1, 2, 2) plt.plot(x,y)  plt.show() |  |

There different ways to plot multiple plots: <https://www.w3schools.com/python/matplotlib_subplots.asp>

## Creating Scatter Plots

With Pyplot, you can use the scatter() function to draw a scatter plot.

The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

**Example:**

|  |  |
| --- | --- |
| **Code** | **Output** |
| import matplotlib.pyplot as plt import numpy as np  x=np.array([5,7,8,7,2,17,2,9,  4,11,12,9,6])  y=np.array([99,86,87,88,111,  86,103,87,94,78,77,85,86])  plt.scatter(x,y) plt.show() |  |

**Explanation of above plot:**

The observation in the example above is the result of 13 cars passing by. The X-axis shows how old the car is. The Y-axis shows the speed of the car when it passes. Are there any relationships between the observations? It seems that the newer the car, the faster it drives, but that could be a coincidence, after all we only registered 13 cars.

There are different type of scatter graphs that can be created (kindly see the link given, as all examples will make the manual lengthy): <https://www.w3schools.com/python/matplotlib_scatter.asp>

## Creating Bars

With Pyplot, you can use the bar() function to draw bar graphs:

**Example**

Draw 4 bars:

|  |  |
| --- | --- |
| **Code** | **Output** |
| import matplotlib.pyplot as plt import numpy as np  x = np.array(["A", "B", "C", "D"]) y = np.array([3, 8, 1, 10])  plt.bar(x,y) plt.show() |  |

The bar() function takes arguments that describes the layout of the bars.

The categories and their values represented by the *first*and *second*argument as arrays.

|  |  |
| --- | --- |
| import matplotlib.pyplot as plt  x = ["APPLES", "BANANAS"] y = [400, 350] plt.bar(x, y) |  |

## Histogram

A histogram is a graph showing *frequency* distributions. It is a graph showing the number of observations within each given interval. Example: Say you ask for the height of 250 people; you might end up with a histogram like this:

|  |  |
| --- | --- |
| You can read from the histogram that there are approximately:  2 people from 140 to 145cm 5 people from 145 to 150cm 15 people from 151 to 156cm 31 people from 157 to 162cm 46 people from 163 to 168cm 53 people from 168 to 173cm 45 people from 173 to 178cm 28 people from 179 to 184cm 21 people from 185 to 190cm 4 people from 190 to 195cm |  |

### Create Histogram

In Matplotlib, we use the hist() function to create histograms.

The hist() function will use an array of numbers to create a histogram, the array is sent into the function as an argument. For simplicity we use NumPy to randomly generate an array with 250 values, where the values will concentrate around 170, and the standard deviation is 10.

|  |  |
| --- | --- |
| **Code** | **Output** |
| import matplotlib.pyplot as plt import numpy as np  x = np.random.normal(170, 10, 250)  plt.hist(x) plt.show() |  |

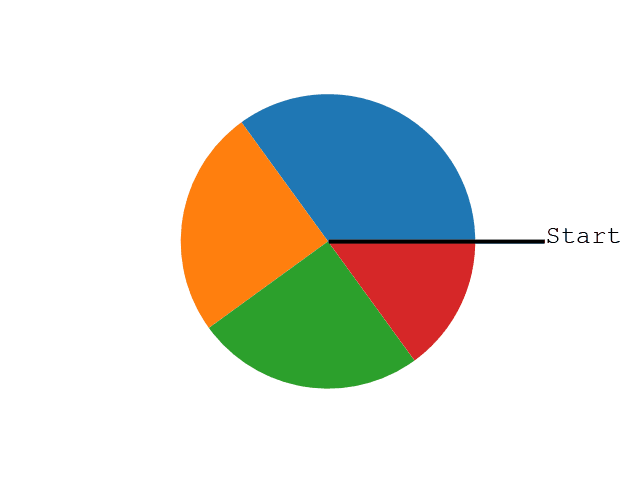
## Creating Pie Charts

With Pyplot, you can use the pie() function to draw pie charts:

|  |  |
| --- | --- |
| **Code** | **Output** |
| import matplotlib.pyplot as plt import numpy as np  y = np.array([35, 25, 25, 15]) mylabels = ["Apples","Bananas","Cherries","Dates"]  plt.pie(y, labels = mylabels)  plt.legend() plt.show() |  |

As you can see the pie chart draws one piece (called a wedge) for each value in the array (in this case [35, 25, 25, 15]).

By default, the plotting of the first wedge starts from the x-axis and move *counterclockwise*:

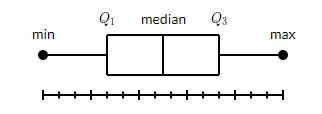


**Note:** The size of each wedge is determined by comparing the value with all the other values, by using this formula:

The value divided by the sum of all values: x/sum(x)

## Box Plot

A box plot which is also known as a whisker plot displays a summary of a set of data containing the minimum, first quartile, median, third quartile, and maximum. In a box plot, we draw a box from the first quartile to the third quartile. A vertical line goes through the box at the median. The whiskers go from each quartile to the minimum or maximum.



The image is taken from: <https://www.tutorialspoint.com/matplotlib/matplotlib_box_plot.htm>

**Example 1:** Draw a box-and-whisker plot for the data set {3, 7, 8, 5, 12, 14, 21, 13, 18}.

Minimum: 3, *Q*1: 6, Median: 12, *Q*3: 16, and Maximum: 21.

|  |  |
| --- | --- |
| **Code** | **Output** |
| import matplotlib.pyplot as plt  data = [3, 7, 8, 5, 12, 14, 21, 13, 18]  plt.boxplot(data)  plt.show() |  |

## Sea Born

Seaborn is a visualization library that is built on top of Matplotlib. It provides data visualizations that are typically more aesthetic and statistically sophisticated.

For example, simply using the Seaborn set() method can dramatically improve the appearance of your Matplotlib plots.

**Seaborn** is also known for making correlation heatmaps, which can be used to identify variable dependence. To generate one, first we need to calculate the correlation between a set of numerical columns.

First, import Seaborn as sns and reformat all of the figures we generated. At the top of your script, use the following code to import library.

**import seaborn as sns**

We can set annot for annotate to true to see the correlation values:

**sns.heatmap(corr, annot=True)**

In Seaborn tool, **pairplot** method is used which allows you to generate a matrix of distributions and scatter plots for a set of numerical features.

Seaborn helps you explore and understand your data. Its plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them.

Please visit [this](https://towardsdatascience.com/exploratory-data-analysis-8fc1cb20fd15) site for further details of seaborn.

<https://towardsdatascience.com/exploratory-data-analysis-8fc1cb20fd15>