Lesson # & Title	Focused Area(s)	Lesson Objectives	Graded Tasks (%)
LESSON 3 Visualising data using Python Matplotlib	Topic 1: Customising plots Topic 2: Plotting 2D arraysstatistics Topic 3: Statistical plots	• Understand how to use Python to visualise data by introducing the matplotlib library • Learn the different classes in the library like pyplot API and pylab • The different elements used to create a 2D plot • How to create sublots to make it easy to understand the relations of different data • Learn how to create different types of plots, especially for statistical purposes like a histogram.	

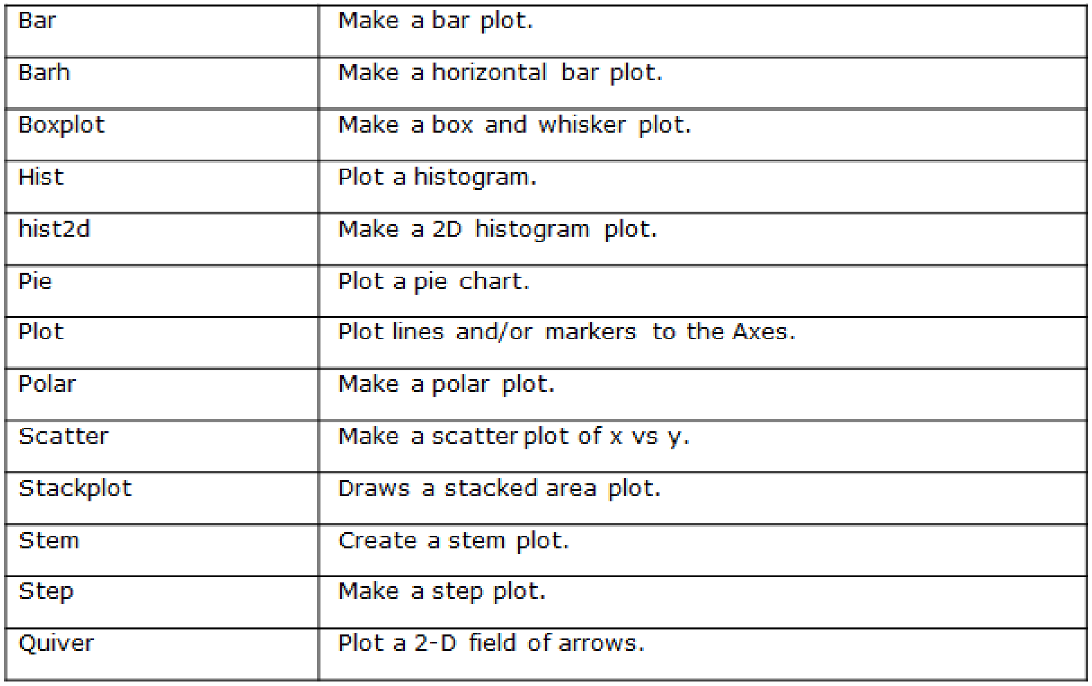
Why This Lesson

MATPLOTLIB is an important library used for data analytics. Visualising data can give the analyst information on the nature of the data and the relationships that can be found from the different data. In addition, the distribution of the data, how data skew, and even how data is clean sometimes can be figured from the plot. Matplotlib contains classes that help to create a figure, canvas, plot some lines, then add some details to help the analyst to study the plot.

TOPIC 1: Customizing Plots

1.1 Introduction to Matplotlib

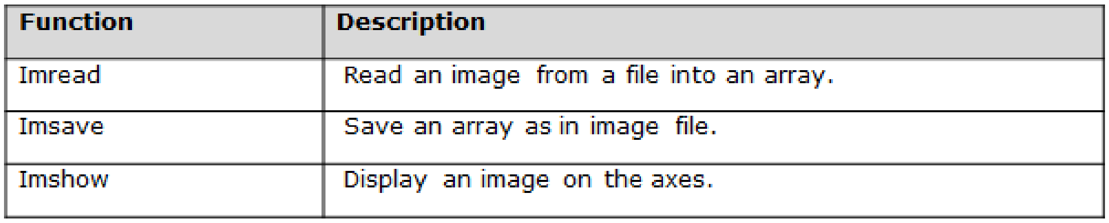
Matplotlib contains different classes like pyplot which contains a group of commands style functions to create different figures and create different types with different customisations. In Figure 1 you can find the different plot types that can be found in this library.



Bar	Make a bar plot.
Barh	Make a horizontal bar plot.
Boxplot	Make a box and whisker plot.
Hist	Plot a histogram.
hist2d	Make a 2D histogram plot.
Pie	Plot a pie chart.
Plot	Plot lines and/or markers to the Axes.
Polar	Make a polar plot.
Scatter	Make a scatter plot of x vs y.
Stackplot	Draws a stacked area plot.
Stem	Create a stem plot.
Step	Make a step plot.
Quiver	Plot a 2-D field of arrows.

Figure 1 Plots Types

There are also different image functions used read the images, show images and also save these images as shown in Figure 2.



Function	**Description**
Imread	Read an image from a file into an array.
Imsave	Save an array as in image file.
Imshow	Display an image on the axes.

Figure 2 Image functions

There are also functions that are used to deal with the axis for each plot. Such as how to add a title for the plot, and x label, and y label for the plot. More axis functions can be found in Figure 3.

Function	Description
Axes	Add axes to the figure.
Text	Add text to the axes.
Title	Set a title of the current axes.
Xlabel	Set the x axis label of the current axis.
Xlim	Get or set the x limits of the current axes.
Xscale	Set the scaling of the x-axis.
Xticks	Get or set the x-limits of the current tick locations and labels.
Ylabel	Set the y axis label of the current axis.
Ylim	Get or set the y-limits of the current axes.
Yscale	Set the scaling of the y-axis.
Yticks	Get or set the y-limits of the current tick locations and labels.

Figure 3 Axis functions

Moreover, there are functions used to work with the figures such as "show" to display the figure, and "savefig" to save the figure in the workspace. Figure 4 explains the different functions used under this category.

Function	**Description**
Figtext	Add text to figure.
Figure	Creates a new figure.
Show	Display a figure.
Savefig	Save the current figure.
Close	Close a figure window.

Figure 4 Figure functions

1.2 Simple Plot

To start the first plotting you need first to learn how to call the matblotlib and the pyplot as follows:

```
# pyplot is imported under the shortcut <plt>
import matplotlib.pyplot as plt
```

In this line, you import the library and use plt as an alias for the library. In the next example, NumPy will be used to create an array and then plot the sin function of the created array as follows:

```python
import numpy as np

# define the x range to be from 0 to 2 pi
pi = 3.14
x=np.arange(0, pi*2, 0.05)

# define the y to be from sin(x) using np
y=np.sin(x)
```

To visualize sin(x), the plot method will be used, where the default is the line plot as follows:

```python
plt.plot(x,y)
plt.xlabel("Angle")
plt.ylabel("Sine")
plt.title('Sine wave')
plt.show()
```

In the example above, the plot(x,y) is used to plot the sine wave considering the x-axis to be an angle, and the y-axis to be the sine wave. Plt. show is used here to show the plot in the output as shown in Figure 5.

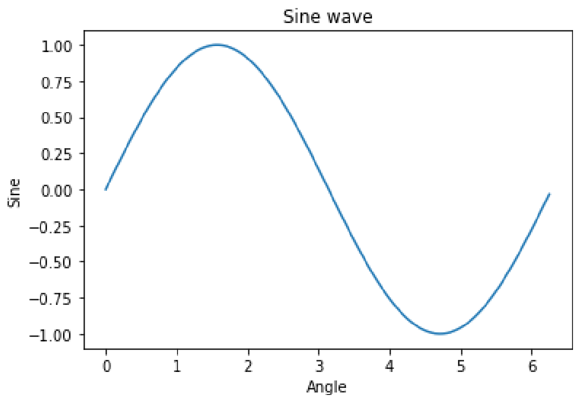


Figure 5 Simple plot for the sine wave

TOPIC 2: Plotting 2D interface.

Visualisation in Python can be done using different libraries such as Matplotlib and seaborn. Matplotlib as introduced earlier is a highly customizable plotting library. Seaborn is built based on Matblotlib which adds more visualisations and is simpler. With the variety of different classes and methods in the Matplotlib library direct it to be object-oriented where the developer or analyst can create figure objects. Objects then will be used to call methods that help to build the figure such as axes, then plots, or subplots, and finally visualise the data with different styles.

2.1 Create a figure

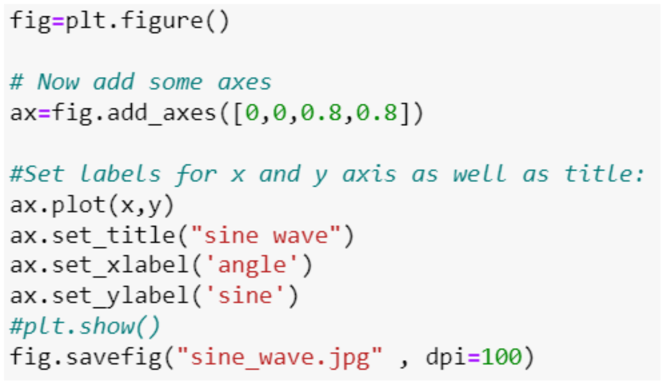
Using the plt alias defined earlier, the figure is created as:

fig = plt.figure()

Now, it is important to create a canvas that is used to put the plotting on. The canvas here is called the axes. Axes method is called using the fig variable which is initialised based on the figure. The axes method should be identified with the four attributes, left, bottom; the initial point on the canvas, and width and height of the canvas on the figure. The values of these attributes range between 0 and 1. For example:

ax=fig.add_axes([0,0,0.8,0.8])

ax is a variable used to define the canvas, the left and bottom are 0s, and the width and height are 0.8 each. The full code can be as:



```
fig=plt.figure()

# Now add some axes
ax=fig.add_axes([0,0,0.8,0.8])

#Set labels for x and y axis as well as title:
ax.plot(x,y)
ax.set_title("sine wave")
ax.set_xlabel('angle')
ax.set_ylabel('sine')
#plt.show()
fig.savefig("sine_wave.jpg" , dpi=100)
```

The canvas ax calls the different methods to create the plot and add the details. After creating the figure, it can be saved using the savfig method as depicted in the example above. Axes take different methods such as:

a. Legend

   Declaration: ax.legend()

   Parameters: Handel, Lable, Location

   Legends are used to identify the different lines or types that can exist in the figure. The important parameter here is the location of the legend in the plot, it can be in the upper left, upper right, or best which will put the legend in an appropriate position.

b. Plot

   This methods take x and y arrays as input and also can take additional options like colour and marker as shown in the figures below.

Character	Color
'b'	Blue
'g'	Green
'r'	Red
'c'	Cyan
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

Figure 6 Color options

Character	Description
'.'	Point marker
'o'	Circle marker
'x'	X marker
'D'	Diamond marker
'H'	Hexagon marker
's'	Square marker
'+'	Plus marker

Figure 7 Marker options

The code example for using these methods is:

```
y = [1, 4, 9, 16, 25,36,49, 64]
x1 = [1, 16, 30, 42,55, 68, 77,88]
x2 = [1,6,12,18,28, 40, 52, 65]

fig=plt.figure()
ax=fig.add_axes([0,0,1,1])

l1=ax.plot(x1,y,'ys-') # solid line with yellow colour and square marker

l2=ax.plot(x2,y,'go--') # dash line with green colour and circle marker
l4 = ax.plot(y, x2,'cs-')

l3= ax.plot(9*np.sqrt(y), y)



ax.legend(labels=('tv', 'Smartphone','Unknown','test'), loc=2) # legend placed at lower right

ax.set_title("Advertisement effect on sales")
ax.set_xlabel('medium')
ax.set_ylabel('sales')

plt.show()
fig.savefig("advertis.png")
```

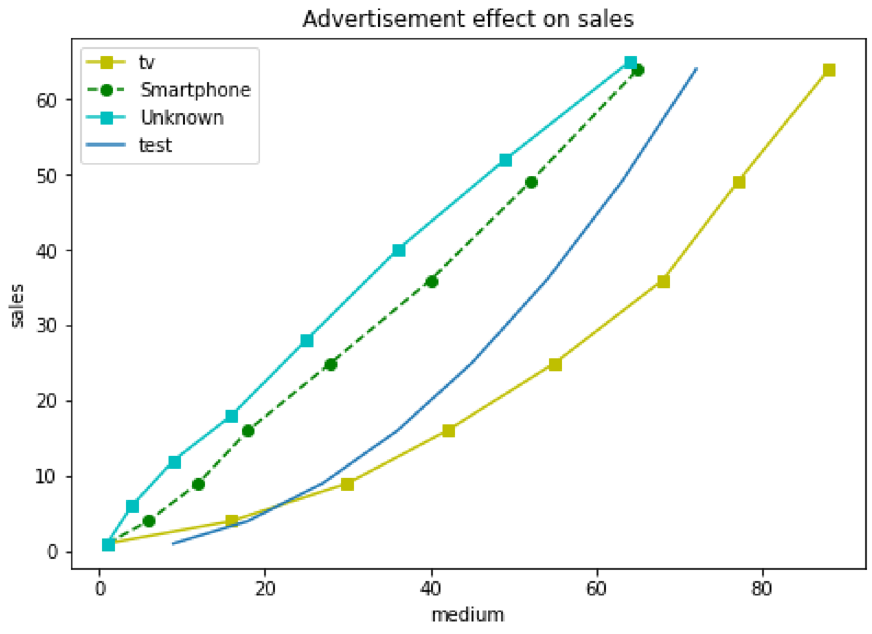


Figure 8 Axes Interface

2.2 Create multiple plots

In the example above, the canvas contains just one plot. Multiple plots can be also defined in one figure using a function called subplot(). This function works in a grid design. First, identify the number of plots as rows and columns, then define the position of each subplot in this grid. The following example explains how this function works and the output of this example is shown in Figure 9.

```
# plot a line, implicitly creating a subplot(111)
plt.plot([1,2,3])
# now create a subplot which represents the top plot of a grid with 2 rows and 1 column.

plt.subplot(311)
plt.plot(range(12))
plt.subplot(312, facecolor='c') # creates 2nd subplot with yellow background
plt.plot(range(12))
plt.subplot(313, facecolor='y') # creates 2nd subplot with yellow background
plt.plot(range(12))
```

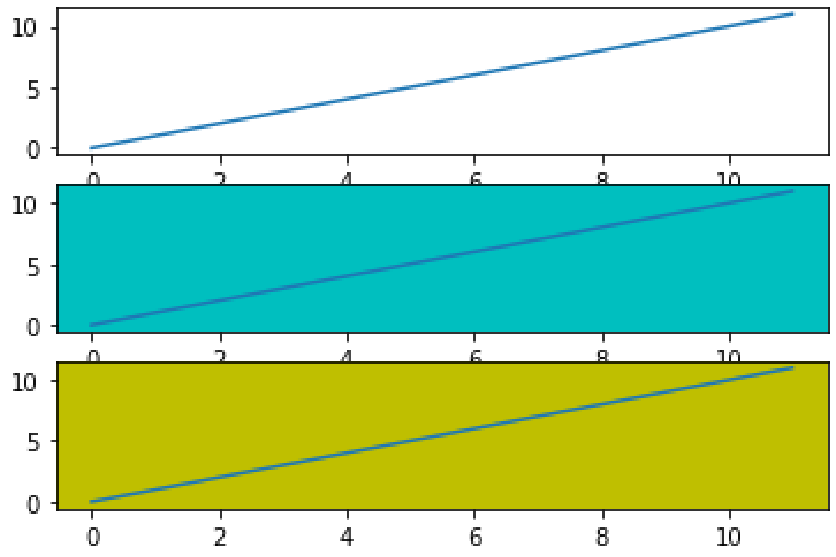


Figure 9 Multiple plots

TOPIC 3: Plot Types

There are different types of plots that are used for visualising the data like line, bar, and column. In addition, some types are used for statistical purposes like histograms and pie charts.

3.1 Bar Chart

If your data is a categorical data, then you need to use the bar chart. It is a rectangular shape, you can define the width of each bar and the height will be according to the y-axis value. The position of the bar on the x-axis can be defined as either centred on the x-value or aligned with the value. The declarations and options of the bar function are explained below.

```
ax.bar(x, height, width, bottom, align)
```

x	sequence of scalars representing the x coordinates of the bars. align controls if x is the bar center (default) or left edge.
height	scalar or sequence of scalars representing the height(s) of the bars
width	scalar or array-like, optional. the width(s) of the bars default 0.8
bottom	scalar or array-like, optional. the y coordinate(s) of the bars default None
align	{'center', 'edge'}, optional, default 'center'

Let's see the example of using bar() function.

```
fig=plt.figure()
ax=fig.add_axes([0,0,1,1])

langs=['C', 'C++', 'Java', 'Python', 'PHP']

students=[23,17,35,29,12]

ax.bar(langs,students,width=.6, bottom=0, align='edge')

plt.show()
```

The output of running this part of the code is shown in Figure 10.

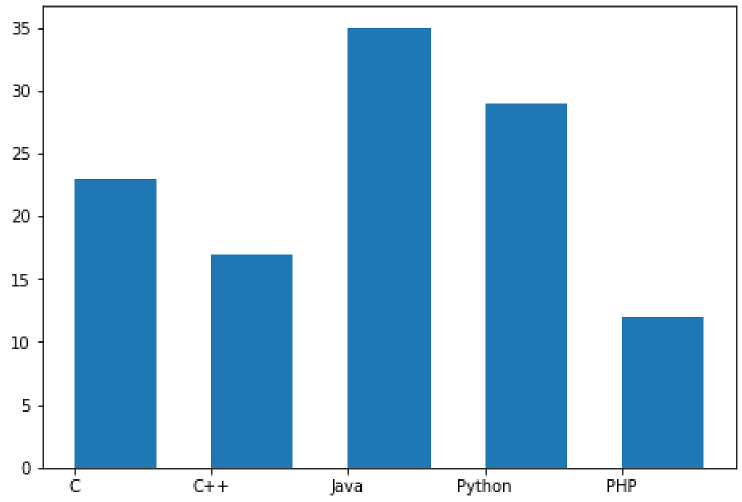
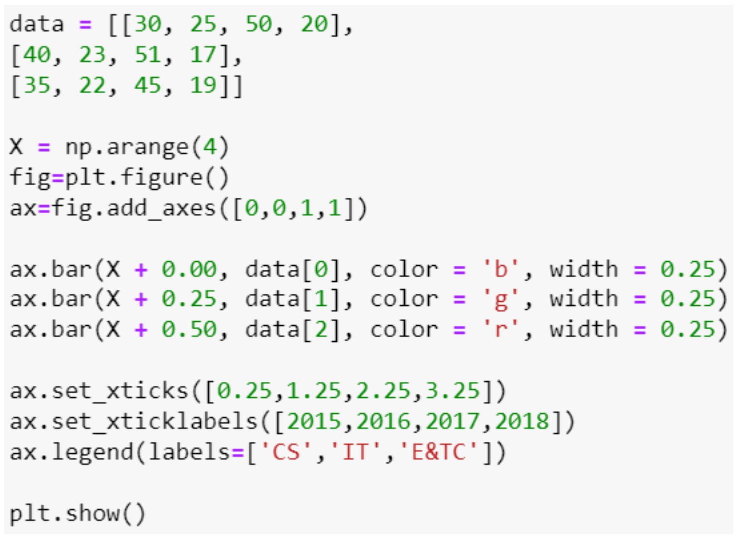


Figure 10 Bar chart

Try to add the labels and the title of this chart.

In order to plot different categories for each x-value, then you can add the bars and specify the position and the width of each bar as explained in the next code.



```
data = [[30, 25, 50, 20],
[40, 23, 51, 17],
[35, 22, 45, 19]]

X = np.arange(4)
fig=plt.figure()
ax=fig.add_axes([0,0,1,1])

ax.bar(X + 0.00, data[0], color = 'b', width = 0.25)
ax.bar(X + 0.25, data[1], color = 'g', width = 0.25)
ax.bar(X + 0.50, data[2], color = 'r', width = 0.25)

ax.set_xticks([0.25,1.25,2.25,3.25])
ax.set_xticklabels([2015,2016,2017,2018])
ax.legend(labels=['CS','IT','E&TC'])

plt.show()
```

As is clear in the above code, the developer needs to design the position of each bar. The offset here and the width are 0.25. Try to change the values and see the difference. The output of running this code is shown in Figure 11.

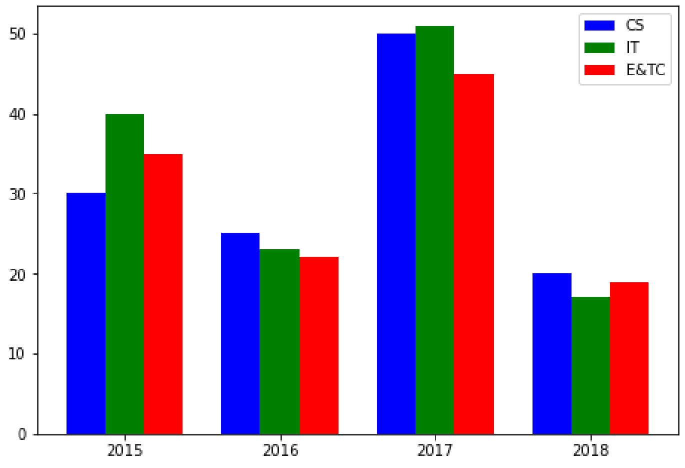


Figure 11 Bar chart with different categories

3.2 Histogram Chart

For numerical data, if you have a column of numerical data and you need to represent the distribution of these data, then use a histogram chart. There are different options used with this function such as "bin" which defined the range of the distribution. The declaration and different functions are explained in the following example.

```
# Marks obtained by students in a class.

fig,ax=plt.subplots(1,1)
a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])

ax.hist(a, bins = [0,50,75,100], histtype = 'step')

ax.set_title("histogram of result")
ax.set_xticks([0,25,50,75,100])
ax.set_xlabel('marks')
ax.set_ylabel('no. of students')

plt.show()
```

The output of running this code is obtained in Figure 12.

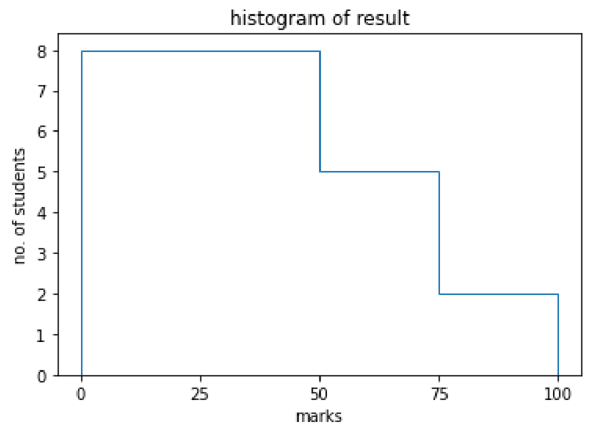


Figure 12 Histogram Chart

The output shows an empty shape as the histotype is steps. There are different types like bar which is the same plot of bar function. Try that out. The options used with hist() function are shown in Figure 13 below.

x	array or sequence of arrays
bins	integer or sequence or 'auto', optional
optional parameters	
range	The lower and upper range of the bins.
density	If True, the first element of the return tuple will be the counts normalized to form a probability density
cumulative	If True, then a histogram is computed where each bin gives the counts in that bin plus all bins for smaller values.
histtype	The type of histogram to draw. Default is 'bar' • 'bar' is a traditional bar-type histogram. If multiple data are given the bars are arranged side by side. • 'barstacked' is a bar-type histogram where multiple data are stacked on top of each other. • 'step' generates a lineplot that is by default unfilled. • 'stepfilled' generates a lineplot that is by default filled.

Figure 13 Hist() function options

3.3 Pie Chart

This chart is used to show the size of each item in the data and the value presented and the percentage of the whole pie. There are different options used with the pie() function as shown in Figure 14 below.

x	array-like. The wedge sizes.
labels	list. A sequence of strings providing the labels for each wedge
Colors	A sequence of matplotlibcolorargs through which the pie chart will cycle. If None, will use the colors in the currently active cycle.
Autopct	string, used to label the wedges with their numeric value. The label will be placed inside the wedge. The format string will be fmt%pct.

Figure 14 pie() function options

The declaration and different functions are explained in the following example.

```
fig=plt.figure()

ax=fig.add_axes([0,0,1,1])
ax.axis('equal')
langs=['C', 'C++', 'Java', 'Python', 'PHP']

students=[30,20,10,20,45]
ax.pie(students, labels=langs,autopct='%1.2f%%')

plt.show()
```

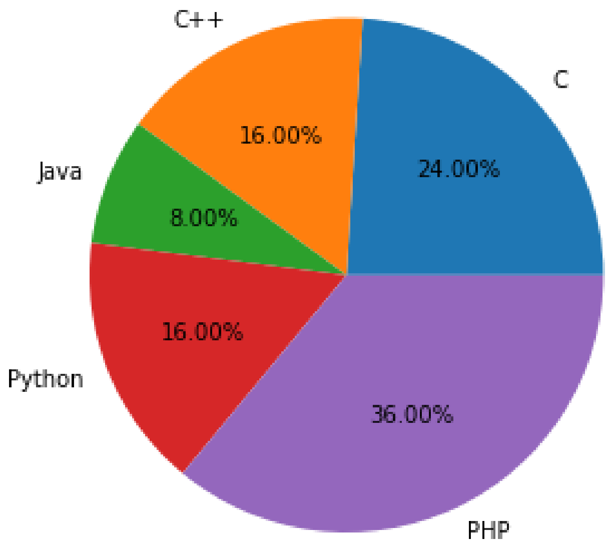


Figure 15 Pie Chart example

3.4 Scatter plot

To obtain the relation between the horizontal data and vertical data, scatter show the relation between them and how each data affects the other.

```
girls_grades = [89, 90, 70, 89, 100, 80, 90, 100, 80, 34]
boys_grades = [30, 29, 49, 48, 100, 68, 78, 45, 80, 30]
grades_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

fig=plt.figure()

ax=fig.add_axes([0,0,1,1])
ax.scatter(grades_range, girls_grades, color='y', marker='^')
ax.scatter(grades_range, boys_grades, color='b', marker='o')
ax.set_xlabel('Grades Range')
ax.set_ylabel('Grades Scored')
ax.set title('scatter plot')
```

Here we have the grades for the girls and the boys. From the scatter of the plot, you will be able to understand the relationship between both grades and who has the higher rate. The output is shown in Figure 16.

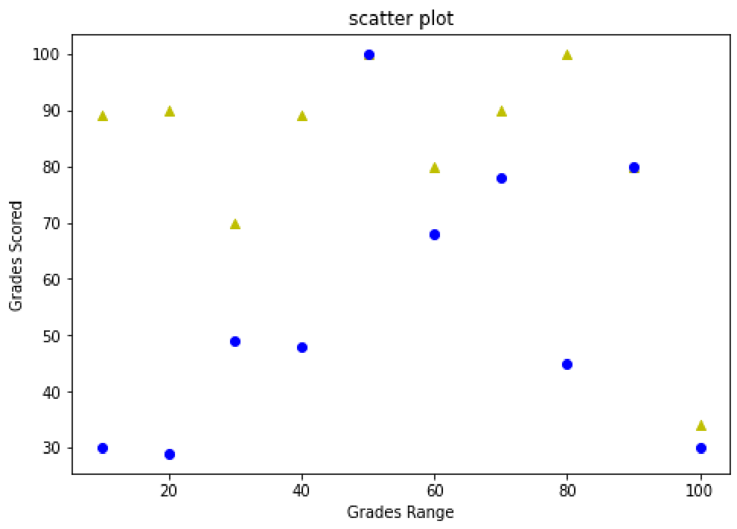


Figure 16 Scatter plot example

- end of lesson content –