DATA VISUALIZATION IN PYTHON

Data visualization is the technique to present the data in a pictorial or graphical format. It enables stakeholders and decision makers to analyze data visually. The data in a graphical format allows them to identify new trends and patterns easily.

Data visualization involves exploring data through visual representations. It's closely associated with data mining, which uses code to explore the patterns and connections in a data set. A data set can be just a small list of numbers that fits in one line of code or many gigabytes of data.

Making beautiful representations of data is about more than pretty pictures. When you have a simple, visually appealing representation of a data set, its meaning becomes clear to viewers. People will see patterns and significance in your data sets that they never knew existed.

"Visualization gives you answers to questions you didn't know you had."
Ben Shneiderman

Plotting a Simple Line Graph

Let's plot a **Simple Line Graph** using ***matplotlib***, and then customize it to create a more informative visualization of our data.

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy.

We'll use the square number sequence 1, 4, 9, 16, 25 as the data for the graph.

We first import the **pyplot** module using the alias **plt** so we don't have to type pyplot repeatedly. **pyplot** contains a number of functions that help generate charts and plots.

We create a **list** to hold the squares and then pass it to the **plot()** function, which will try to plot the numbers in a meaningful way.

plt.show() opens matplotlib's viewer and displays the plot, as shown in Figure 1.

```
import matplotlib.pyplot as plt
squares = [1, 4, 9, 16, 25]
plt.plot(squares)
plt.show()
```

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

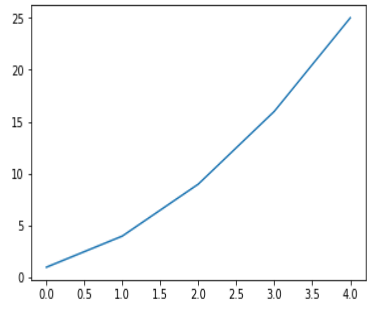


Figure 1. Simple Line Graph

Changing the Label Type and Graph Thickness

Matplotlib allows you to adjust every feature of a visualization. We'll use a few of the available customizations to improve the readability of this plot.

The **linewidth** parameter controls the thickness of the line that **plot()** generates. The **title()** function sets a title for the chart. The **fontsize** parameters, which appear repeatedly throughout the code, control the size of the text on the chart.

The **xlabel()** and **ylabel()** functions allow you to set a title for each of the axes and the function **tick_params()** styles the tick marks.

The arguments shown here affect the tick marks on **both** the **x** and **y** axes and set the font size of the tick mark labels to 14 (**labelsize=14**).

As you can see in Figure 2, the resulting chart is much easier to read. The label type is bigger, and the line graph is thicker.

```
import matplotlib.pyplot as plt
squares = [1, 4, 9, 16, 25]
plt.plot(squares, linewidth=5)

# Set chart title and label axes.
plt.title("Square Numbers", fontsize=24)
plt.xlabel("Value", fontsize=14)
plt.ylabel("Square of Value", fontsize=14)

# Set size of tick labels.
plt.tick_params(axis='both', labelsize=14)
plt.show()
```

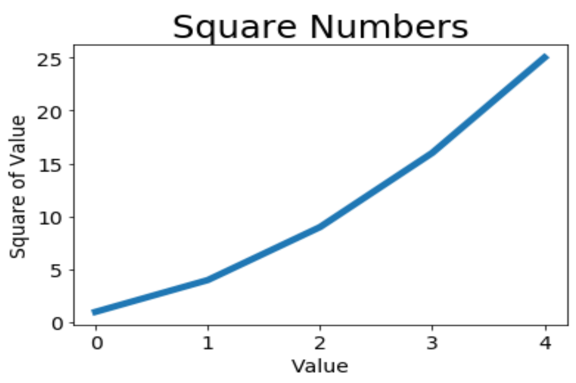


Figure 2. The chart is much easier to read now

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Correcting the Plot

Notice at the end of the graph that the square of 4 is shown as 25! Let's fix that.

When you give **plot()** a sequence of numbers, it assumes the first data point corresponds to an x-coordinate value of 0, but our first point corresponds to an x-value of 1. We can override the default behavior by giving **plot()** both the input and output values used to calculate the squares.

Now **plot()** will graph the data correctly because we've provided both the input and output values, so it doesn't have to assume how the output numbers were generated. The resulting plot, shown in Figure 3, is correct.

```
import matplotlib.pyplot as plt
input_values = [1, 2, 3, 4, 5]
squares = [1, 4, 9, 16, 25]
plt.plot(input_values, squares, linewidth=5)

# Set chart title and label axes.
plt.title("Square Numbers", fontsize=24)
plt.xlabel("Value", fontsize=14)
plt.ylabel("Square of Value", fontsize=14)

# Set size of tick labels.
plt.tick_params(axis='both', labelsize=14)
plt.show()
```

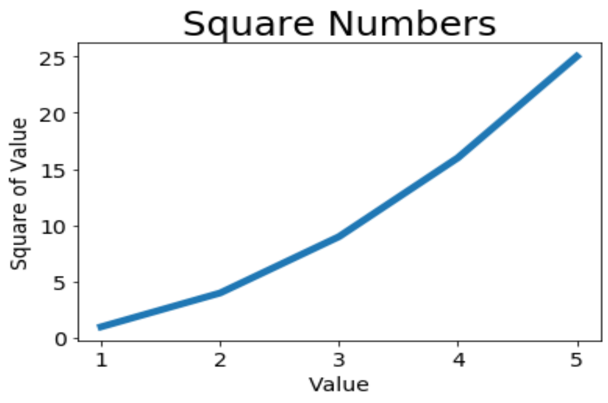


Figure 3. The data is now plotted correctly

Plotting and Styling Individual Points with scatter()

Sometimes it's useful to be able to plot and style individual points based on certain characteristics. For example, you might plot small values in one color and larger values in a different color. You could also plot a large data set with one set of styling options and then emphasize individual points by replotting them with different options.

To plot a single point, use the **scatter()** function. Pass the single (x, y) values of the point of interest to **scatter()**, and it should plot those values.

```
import matplotlib.pyplot as plt
plt.scatter(2, 4)
plt.show()
```

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

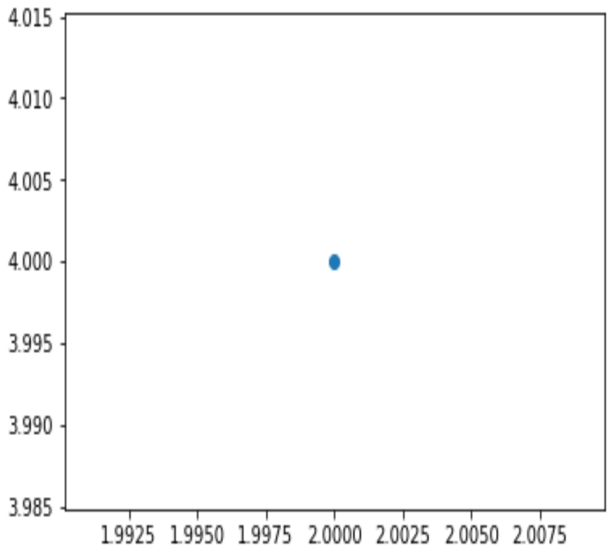


Figure 4. Plotting a single point

Plotting and Styling Individual Points with scatter()

Let’s style the output to make it more interesting. We’ll add a title, label the axes, and make sure all the text is large enough to read.

We call **scatter()** and use the **s** argument to set the size of the dots used to draw the graph. When you run, you should see a single point in the middle of the chart, as shown in Figure 5.

```
import matplotlib.pyplot as plt
plt.scatter(2, 4, s=200)
# Set chart title and label axes.
plt.title("Square Numbers", fontsize=24)
plt.xlabel("Value", fontsize=14)
plt.ylabel("Square of Value", fontsize=14)
# Set size of tick labels.
plt.tick_params(axis='both', which='major', labelsize=14)
plt.show()
```

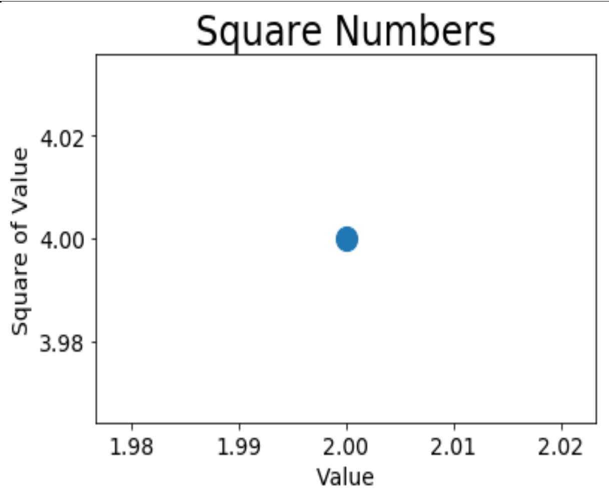


Figure 5. Plotting a single point

Plotting a Series of Points with scatter()

To plot a series of points, we can pass **scatter()** separate lists of **x** and **y** values.

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

The **x_values** list contains the numbers to be squared, and **y_values** contains the square of each number. When these lists are passed to **scatter()**, matplotlib reads one value from each list as it plots each point. The points to be plotted are (1, 1), (2, 4), (3, 9), (4, 16), and (5, 25);

the result is shown in Figure 6.

```
import matplotlib.pyplot as plt
x_values = [1, 2, 3, 4, 5]
y_values = [1, 4, 9, 16, 25]
plt.scatter(x_values, y_values, s=100)
# Set chart title and label axes.
plt.title("Square Numbers", fontsize=24)
plt.xlabel("Value", fontsize=14)
plt.ylabel("Square of Value", fontsize=14)
# Set size of tick labels.
plt.tick_params(axis='both', which='major', labelsize=14)
plt.show()
```

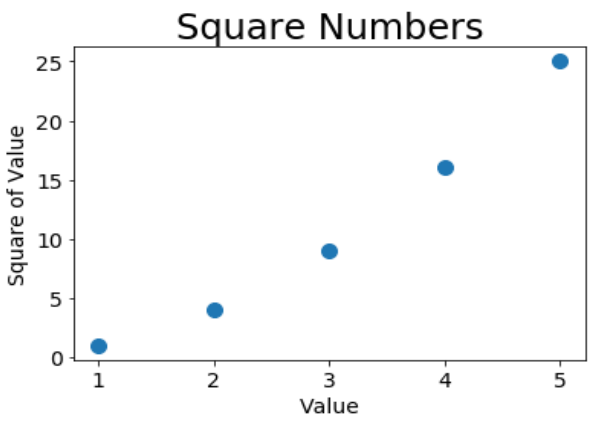


Figure 6. A scatter plot with multiple points

Calculating Data Automatically

Writing out lists by hand can be inefficient, especially when we have many points. Rather than passing our points in a list, let's use a loop in Python to do the calculations for us. Here's how this would look with 1000 points.

We start with a list of x-values containing the numbers 1 through 1000.

Next, a list comprehension generates the y-values by looping through the x-values, squaring each number, and storing the results in **y_values**.

We then pass the input and output lists to **scatter()**.

Because this is a large data set, we use a smaller point size and we use the **axis()** function to specify the range of each axis.

The **axis()** function requires four values: the minimum and maximum values for the x-axis and the y-axis.

Here, we run the x-axis from 0 to 1100 and the y-axis from 0 to 1100000. Figure 7 shows the result.

```
import matplotlib.pyplot as plt
x_values = list(range(1, 1001))
y_values = [x**2 for x in x_values]
plt.scatter(x_values, y_values, s=40)
# Set chart title and label axes.
plt.title("Square Numbers", fontsize=24)
plt.xlabel("Value", fontsize=14)
```

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
plt.ylabel("Square of Value", fontsize=14)
# Set the range for each axis.
plt.axis([0, 1100, 0, 1100000])
plt.show()
```

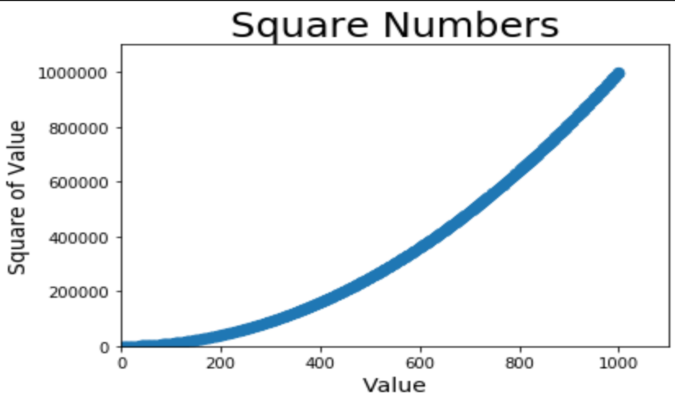


Figure 7. Python can plot 1000 points as easily as it plots 5 points

Removing Outlines from Data Points

matplotlib lets you color points individually in a scatter plot. The default—blue dots with a black outline—works well for plots with a few points. But when plotting many points, the black outlines can blend together. To remove the outlines around points, pass the argument **edgecolor='none'** when you call **scatter()**:

Run program using this call, and you should see only solid blue points in your plot.

```
import matplotlib.pyplot as plt
x_values = list(range(1, 1001))
y_values = [x**2 for x in x_values]
plt.scatter(x_values, y_values, edgecolor='none', s=40)
# Set chart title and label axes.
plt.title("Square Numbers", fontsize=24)
plt.xlabel("Value", fontsize=14)
plt.ylabel("Square of Value", fontsize=14)
# Set the range for each axis.
plt.axis([0, 1100, 0, 1100000])
plt.show()
```

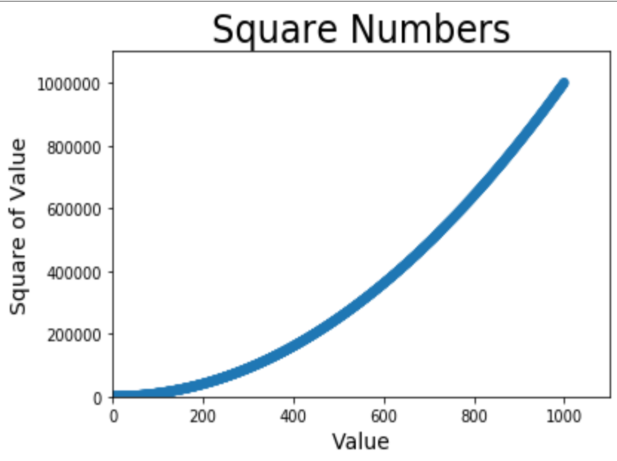


Figure 8.

Defining Custom Colors

To change the color of the points, pass **c** to **scatter()** with the name of a color to use.

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

You can also define custom colors using the RGB color model. To define a color, pass the **c** argument a tuple with three decimal values (one each for red, green, and blue), using values between 0 and 1. Values closer to 0 produce dark colors, and values closer to 1 produce lighter colors.

```
import matplotlib.pyplot as plt
x_values = list(range(1, 1001))
y_values = [x**2 for x in x_values]
plt.scatter(x_values, y_values, c='red', edgecolor='none', s=40)
# Set chart title and label axes.
plt.title("Square Numbers", fontsize=24)
plt.xlabel("Value", fontsize=14)
plt.ylabel("Square of Value", fontsize=14)
# Set the range for each axis.
plt.axis([0, 1100, 0, 1100000])
plt.show()
```

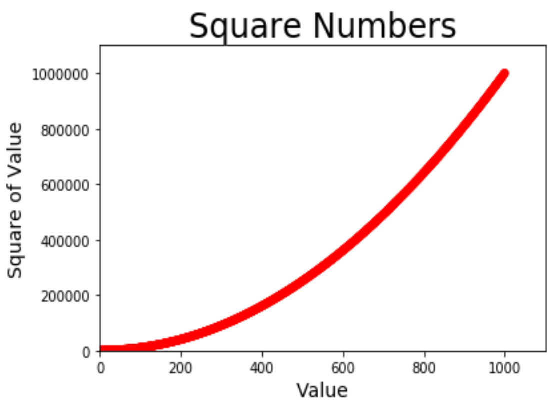


Figure 9.

RGB

plt.scatter(x_values, y_values, c=(0, 0, 0.8), edgecolor='none', s=40)

Using a Colormap

A **colormap** is a series of colors in a gradient that moves from a starting to ending color. Colormaps are used in visualizations to emphasize a pattern in the data. For example, you might make low values a light color and high values a darker color. The **pyplot** module includes a set of built-in colormaps. To use one of these colormaps, you need to specify how **pyplot** should assign a color to each point in the data set.

We pass the list of y-values to **c** and then tell **pyplot** which colormap to use through the **cmap** argument. This code colors the points with lower y-values light blue and the points with larger y-values dark blue. The resulting plot is shown in Figure 10.

```
import matplotlib.pyplot as plt
x_values = list(range(1001))
y_values = [x**2 for x in x_values]
plt.scatter(x_values, y_values, c=y_values, cmap=plt.cm.Blues,
edgecolor='none', s=40)
# Set chart title and label axes.
plt.title("Square Numbers", fontsize=24)
plt.xlabel("Value", fontsize=14)
plt.ylabel("Square of Value", fontsize=14)
# Set the range for each axis.
plt.axis([0, 1100, 0, 1100000])
```

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
    plt.show()
```

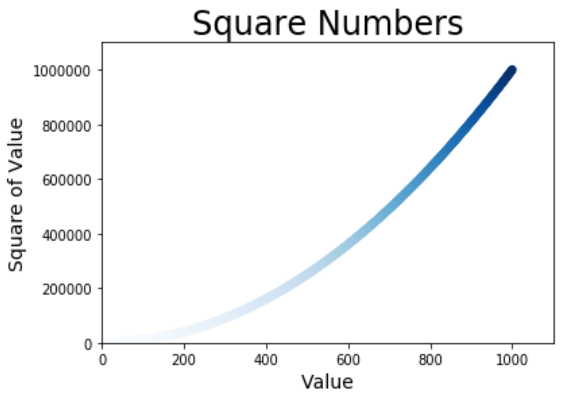
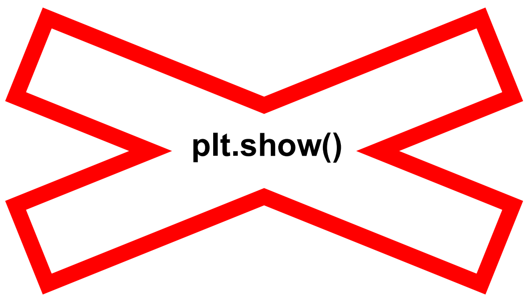


Figure 10. A plot using the Blues colormap

Saving Your Plots Automatically



plt.savefig('squares_plot.png', bbox_inches='tight')

If you want your program to automatically save the plot to a file, you can replace the call to **plt.show()** with a call to **plt.savefig().**

The first argument is a filename for the plot image, which will be saved in the same directory as your python program .

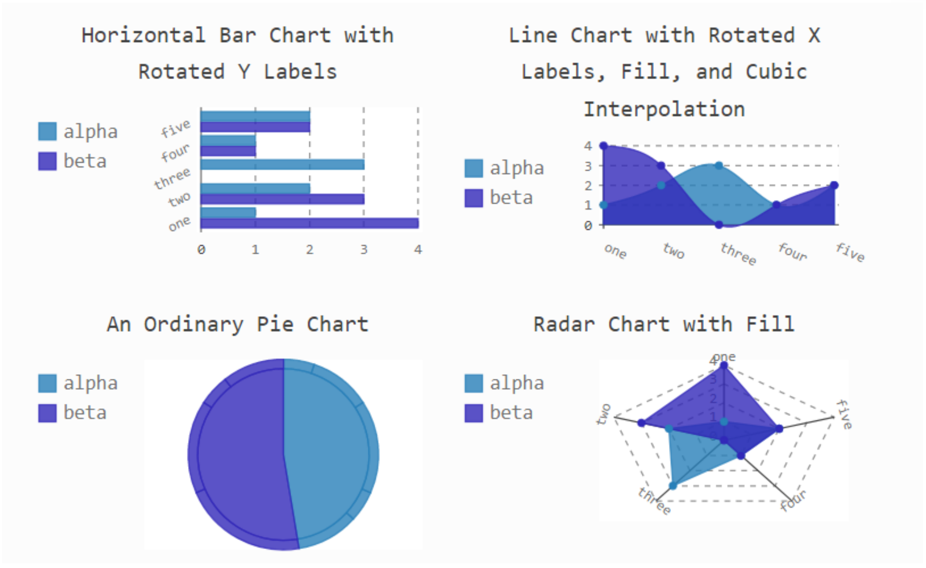
The second argument trims extra whitespace from the plot.

If you want the extra whitespace around the plot, you can omit this argument.

Pygal

In this section we'll use the Python visualization package Pygal to produce scalable vector graphics files. These are useful in visualizations that are presented on differently sized screens because they scale automatically to fit the viewer's screen. If you plan to use your visualizations online, consider using Pygal so your work will look good on any device people use to view your visualizations.

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



Making a Histogram

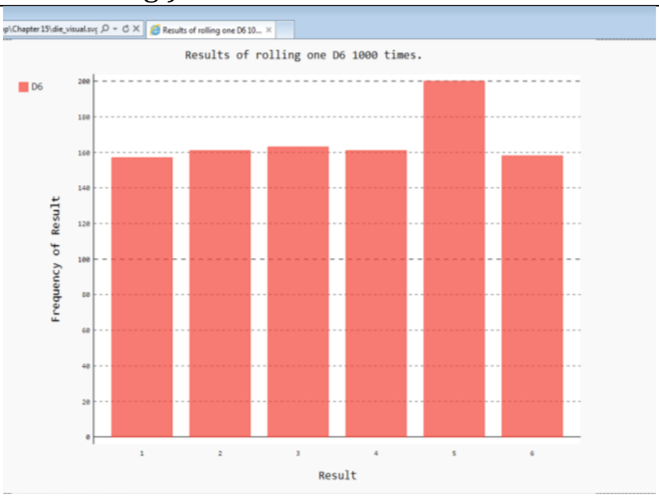
With a list of frequencies, we can make a histogram of the results. A histogram is a bar chart showing how often certain results occur.

We make a bar chart by creating an instance of **pygal.Bar()**, which we store in **hist**. We then set the **title** attribute of **hist**, use the possible results of the labels for the x-axis, and add a title for each of the axes. We use **add()** to add a series of values to the chart. Finally, we render the chart to an **SVG** file, which expects a filename with the **.svg** extension.

The simplest way to look at the resulting histogram is in a web browser.

Notice that Pygal has made the chart interactive: hover your cursor over any bar in the chart and you'll see the data associated with it. This feature is particularly useful when plotting multiple data sets on the same chart.

```
import pygal
------------------------------------------------
# Visualize the results.
hist = pygal.Bar()
hist.title = "Results of rolling one D6 1000 times."
hist.x_labels = ['1', '2', '3', '4', '5', '6']
hist.x_title = "Result"
hist.y_title = "Frequency of Result"
hist.add('D6', frequencies)
hist.render_to_file('die_visual.svg')
```


Results of rolling one D6 10...
Results of rolling one D6 1000 times.
D6
200
180
160
140
120
100
80
60
40
20
0
Frequency of Result
1
2
3
4
5
6
Result

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Figure 11: A simple bar chart created with Pygal

Building a World Map

Pygal includes a **Worldmap** chart type to help map global data sets. As an example of how to use **Worldmap**, we'll create a simple map that highlights North America, Central America, and South America.

We make an instance of the **Worldmap** class and set the map's title attribute. Then we use the **add()** method, which takes in a label and a list of country codes for the countries we want to focus on. Each call to **add()** sets up a new color for the set of countries and adds that color to a key on the left of the graph with the label specified here. We want the entire region of North America represented in one color, so we place 'ca', 'mx', and 'us' in the list we pass to the first **add()** call to highlight Canada, Mexico, and the United States together. We then do the same for the countries in Central America and South America.

The method **render_to_file()** creates an **.svg** file containing the chart, which you can open in your browser. The output is a map highlighting North, Central, and South America in different colors, as shown in Figure 12.

```
import pygal
wm = pygal.maps.world.World()
wm.title = 'North, Central, and South America'
wm.add('North America', ['ca', 'mx', 'us'])
wm.add('Central America', ['bz', 'cr', 'gt', 'hn', 'ni', 'pa', 'sv'])
wm.add('South America', ['ar', 'bo', 'br', 'cl', 'co', 'ec', 'gf', 'gy', 'pe', 'py', 'sr', 'uy', 've'])
wm.render_to_file('americas.svg')
```

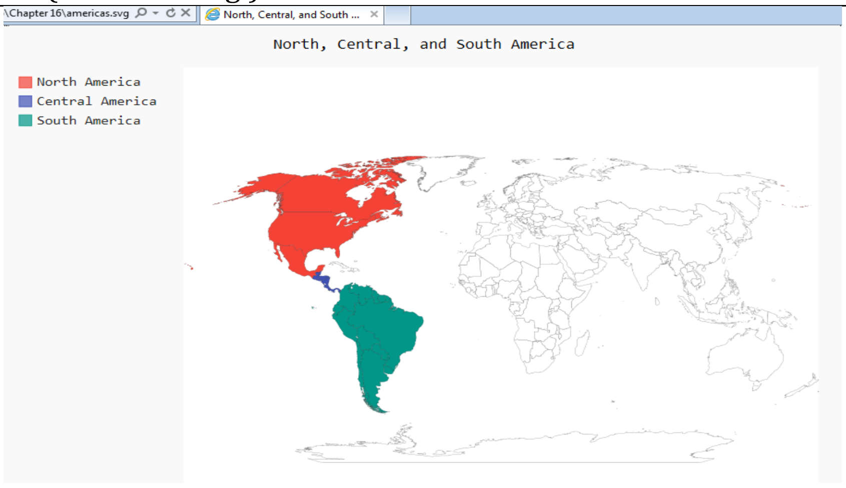


Figure 12. A simple instance of the Worldmap chart type

References

1. Link: [https://www.simplilearn.com/data-visualization-in-python-using-matplotlib-tutorial]
2. Python PYTHON CRASH COURSE by Eric Matthes

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.