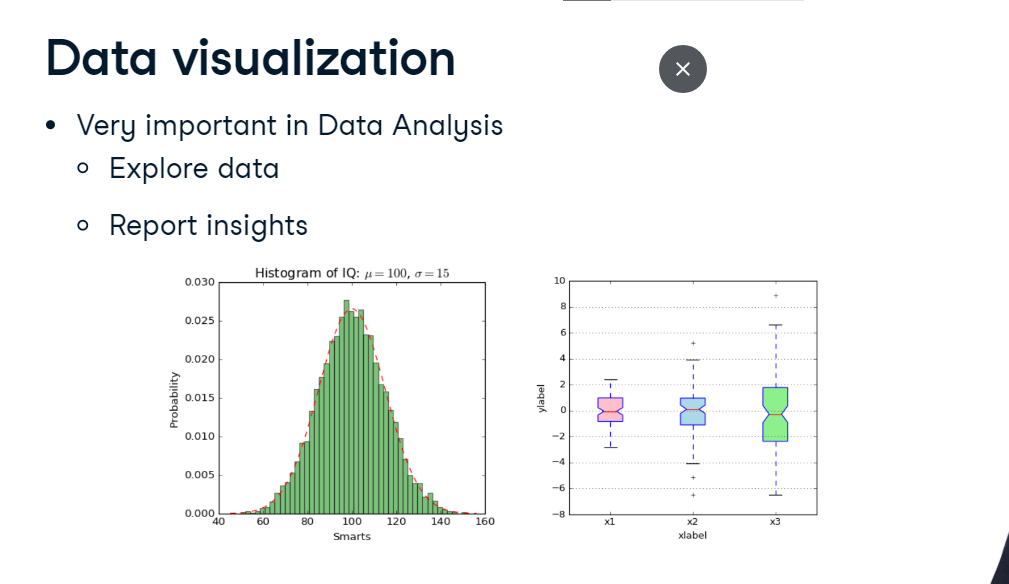
**Basic plots with Matplotlib**

Hi! My name is Hugo, and I'm a data scientist and educator at DataCamp. I'm also the host of the weekly podcast DataFramed, which you need to check out to stay up to date with everything that's happening in data science. In this intermediate Python course, you're going to take your Python skills to the next level, specifically for data science.

You will learn how to visualize data and to store data in new data structures. Along the way, you will master control structures, which you will need to customize the flow of your scripts and algorithms. These are the types of things data scientists use every day. We'll finish this chapter with a case study, where you'll blend together everything you've learned to solve an exciting challenge.

**Data visualization**

This first chapter is about data visualization, which is a very important part of data analysis. First of all, you will use it to explore your dataset. The better you understand your data, the better you'll be able to extract insights. And once you've found those insights, again, you'll need visualization to be able to share your valuable insights with other people.



**Beautiful plot**

As an example, have a look at this beautiful plot. It's made by the late, the great Swedish professor Hans Rosling. His talks about global development have been viewed millions of times. And what makes them so intriguing, is that by making beautiful plots, he allows the data to tell their own story. Here we see a bubble chart, where each bubble represents a country. The bigger the bubble, the bigger the country's population, so the two biggest bubbles here are China and India. There are 2 axes.

1. 1 Source: GapMinder, Wealth and Health of Nations

**Beautiful plot**

The horizontal axis shows the GDP per capita, in US dollars.

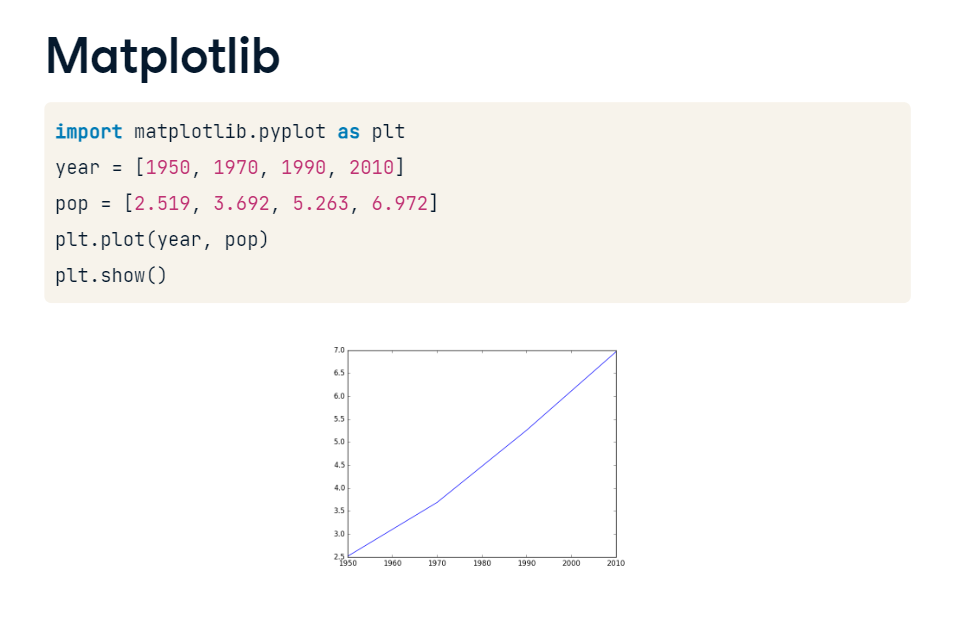
1. 1 Source: GapMinder, Wealth and Health of Nations

The vertical axis shows life expectancy. We clearly see that people live longer in countries with a higher GDP per capita. Still, there is a huge difference in life expectancy between countries on the same income level. Now why am I telling you this? Well, because by the end of this chapter, you'll be able to build this beautiful plot yourself!

1. 1 Source: GapMinder, Wealth and Health of Nations

**Matplotlib**

There are many visualization packages in python, but the mother of them all, is matplotlib. You will need its subpackage pyplot. By convention, this subpackage is imported as plt, like this. For our first example, let's try to gain some insights in the evolution of the world population. I have a list with years here, year, and a list with corresponding populations, expressed in billions, pop. In the year 1970, for example, 3.7 billion people lived on planet Earth. To plot this data as a line chart, we call plt-dot-plot and use our two lists as arguments. The first argument corresponds to the horizontal axis, and the second one to the vertical axis. You might think that a plot will pop up right now, but Python's pretty lazy. It will wait for the show function to actually display the plot. This is because you might want to add some extra ingredients to your plot before actually displaying it, such as titles and label customizations. I'll talk about that some more later on. Just remember this: the plot function tells Python what to plot and how to plot it. show actually displays the plot.



**Matplotlib**

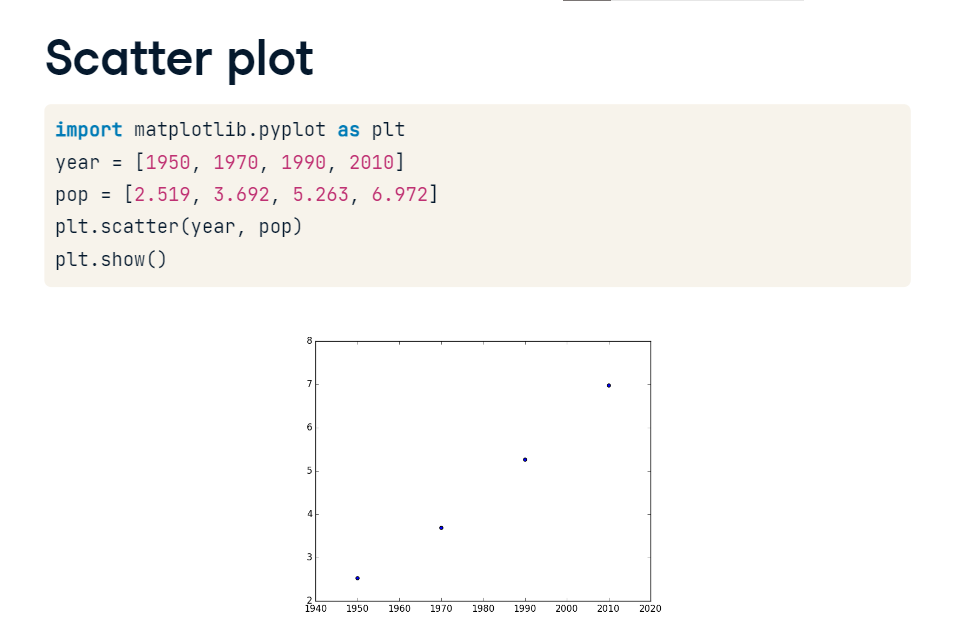
When we look at our plot, we see that the years are indeed shown on the horizontal axis, and the populations on the vertical axis.

There are four data points, and Python draws a line between them. In 1950, the world population was around 2 point 5 billion. In 2010, it was around 7 billion. So the world population has almost tripled in sixty years. What if the population keeps on growing like that? Will the world become over populated? You'll find out in the exercises.

**Scatter plot**

Let me first introduce you to another type of plot: the scatter plot. To create it, we can start from the code from before. This time, though, you change the plot function to scatter.

The resulting scatter plot simply plots all the individual data points; Python doesn't connect the dots with a line. For many applications, the scatter plot is often a better choice than the line plot, so remember this scatter function well. You could also say that this is a more -honest- way of plotting your data, because you can clearly see that the plot is based on just four data points.



**Histogram**

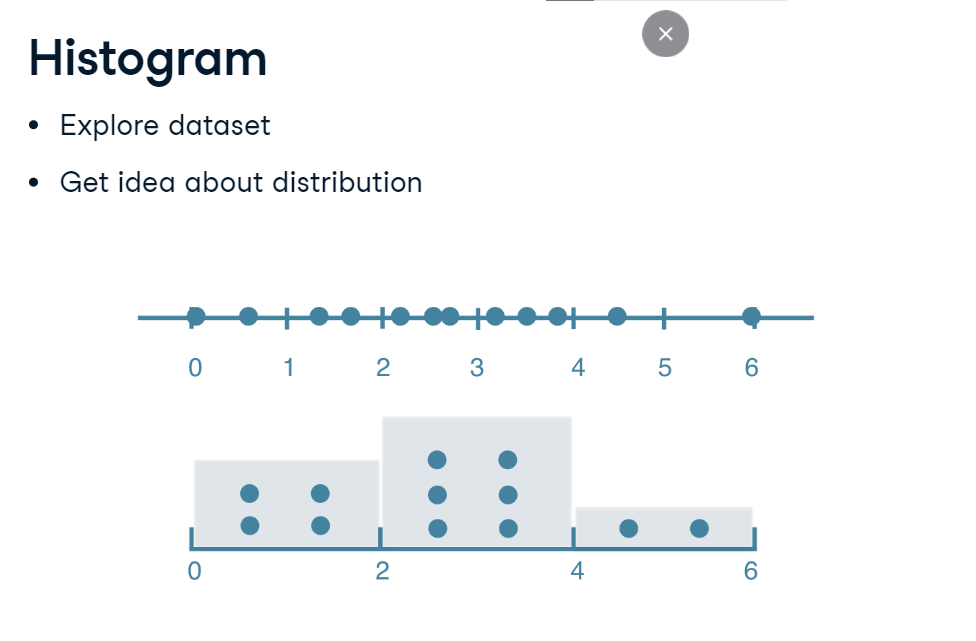
The histogram is a type of visualization that's very useful to explore your data. It can help you to get an idea about the distribution of your variables. To see how it works, imagine 12 values between 0 and 6.

I've put them along a number line here. To build a histogram for these values, you can divide the line into equal chunks, called bins.

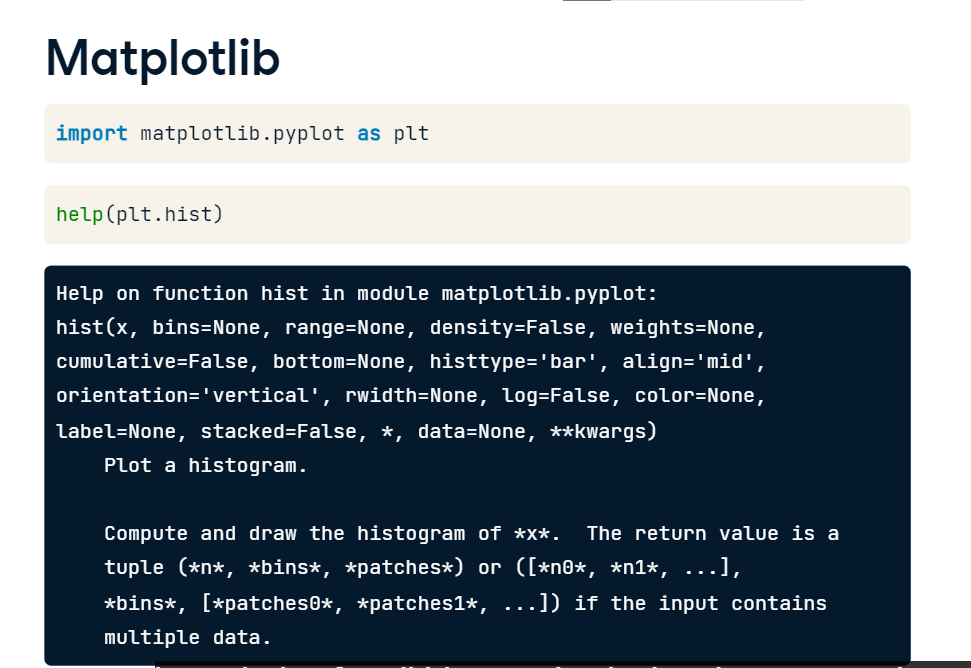
Suppose you go for 3 bins, that each have a width of 2. Next, you count how many data points sit inside each bin.

There's 4 data points in the first bin,6 in the second bin and 2 in the third bin.

Finally, you draw a bar for each bin. The height of the bar corresponds to the number of data points that fall in this bin. The result is a histogram, which gives us a nice overview on how the 12 values are distributed. Most values are in the middle, but there are more values below 2 than there are above 4. Of course, we can use matplotlib to build histograms as well.

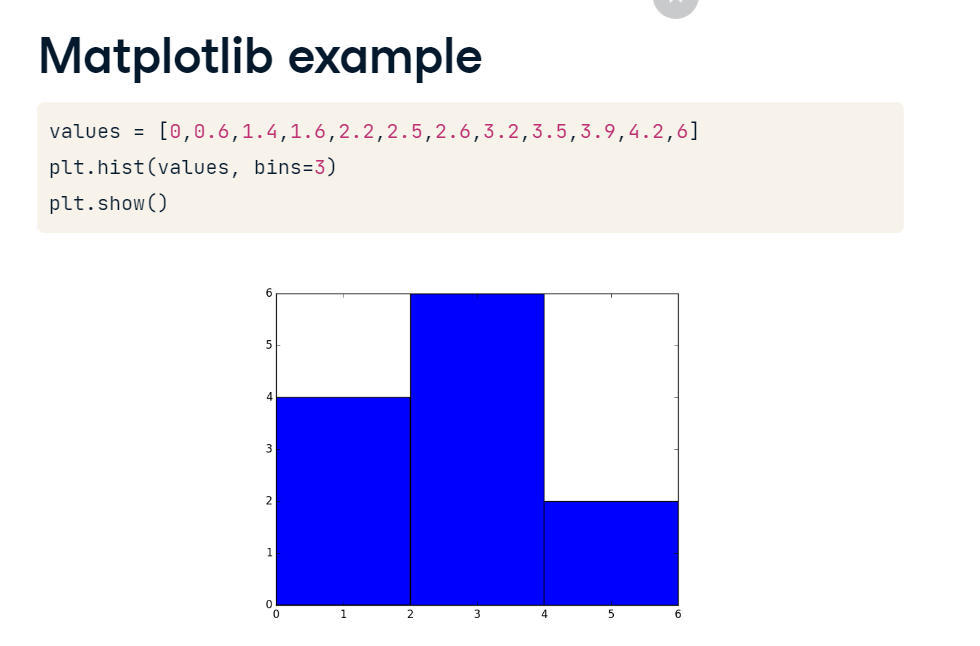


**Matplotlib**

As before, you should start by importing the pyplot package that's inside matplotlib. Next, you can use the hist function. Let's open up its documentation. There's a bunch of arguments you can specify, but the first two here are the most important ones. x should be a list of values you want to build a histogram for. You can use the second argument, bins, to tell Python into how many bins the data should be divided. Based on this number, hist will automatically find appropriate boundaries for all bins, and calculate how may values are in each one. If you don't specify the bins argument, it will by 10 by default. 

**Matplotlib example**

So to generate the histogram that you've seen before, let's start by building a list with the 12 values. Next, you simply call hist and pass this list as an input, so it's matched to the argument x. I also specified the bins argument to be 3, so that the values are divided in three bins. If you finally call the show function, you get a histogram. Histograms are really useful to give a bigger picture.



**Population pyramid**

As an example, have a look at this so-called population pyramid. The age distribution is shown, for both males and females, in the European Union. Notice that the histograms are flipped 90 degrees; the bins are horizontal now. The bins are largest for the ages 40 to 44, where there are 20 million males and 20 million females. They are the so called baby boomers. These are figures of the year 2010. What do you think will have changed in 2050?

Let's have a look. The distribution is flatter, and the baby boom generation has gotten older. With the blink of an eye, you can easily see how demographics will be changing over time. That's the true power of histograms at work here!

