**Python NumPy Tutorial – Learn NumPy Arrays With Examples**

you have learned about Arrays in Python and its various fundamentals like functions, lists vs arrays along with its creation. But, those were just the basics and  being the most sought-after skill in the programming domain today, there’s obviously so much more to learn. In this python numpy tutorial, you will understand each aspect of Numpy in the following sequence:

* + [What Is a Python NumPy Array?](https://www.edureka.co/blog/python-numpy-tutorial/" \l "WhatIsNumpy)
  + [NumPy Arrays v/s List](https://www.edureka.co/blog/python-numpy-tutorial/#NumpyVsList)
  + [NumPy Operations](https://www.edureka.co/blog/python-numpy-tutorial/#NumpyOperations)
  + [NumPy Special Functions](https://www.edureka.co/blog/python-numpy-tutorial/#SpecialFunctions)

 So, let’s get started! :-)

## ****What is a Python NumPy?****

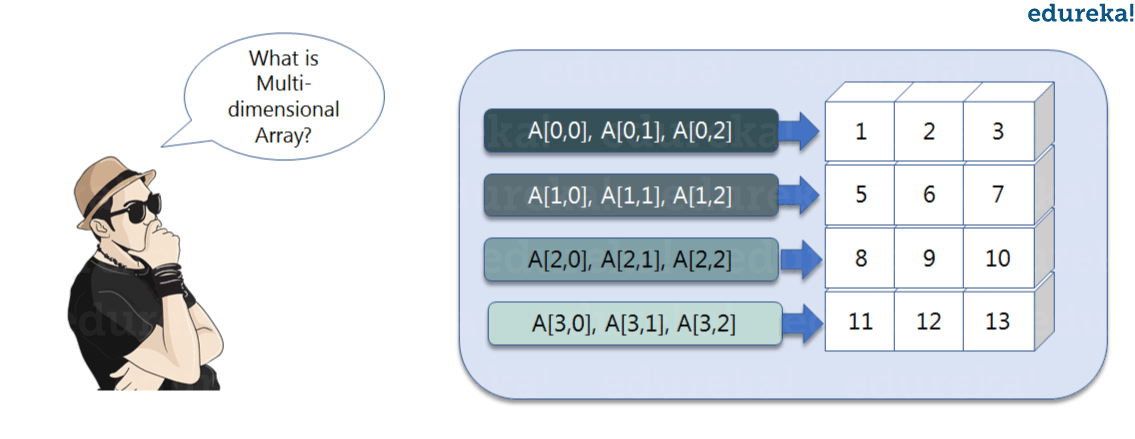
NumPy is a Python package which stands for ‘Numerical Python’. It is the core library for scientific computing, which contains a powerful n-dimensional array object, provide tools for integrating C, C++ etc. It is also useful in linear algebra, random number capability etc. NumPy array can also be used as an efficient multi-dimensional container for generic data. Now, let me tell you what exactly is a python numpy array.

**NumPy Array:**Numpy array is a powerful N-dimensional array object which is in the form of rows and columns. We can initialize numpy arrays from nested Python lists and access it elements. In order to perform these numpy operations, the next question which will come in your mind is:

### ****How do I install NumPy?****

To install Python NumPy, go to your command prompt and type “pip install numpy”. Once the installation is completed, go to your IDE (For example: PyCharm) and simply import it by typing: “import numpy as np”

Moving ahead in python numpy tutorial, let us understand what exactly is a multi-dimensional numPy array.



Here, I have different elements that are stored in their respective memory locations. It is said to be two dimensional because it has rows as well as columns. In the above image, we have 3 columns and 4 rows available.

Let us see how it is implemented in PyCharm:

### ****Single-dimensional Numpy Array:****

|  |  |
| --- | --- |
| 1  2  3 | import numpy as np  a=np.array([1,2,3])  print(a) |

Output – [1 2 3]

### ****Multi-dimensional Array:****

|  |  |
| --- | --- |
| 1  2 | a=np.array([(1,2,3),(4,5,6)])  print(a) |

O/P – [[ 1 2 3]  
[4 5 6]]

Many of you must be wondering that why do we use python numpy if we already have python list? So, let us understand with some examples in this python numpy tutorial.

## ****Python NumPy Array v/s List****

We use python numpy array instead of a list because of the below three reasons:

1. Less Memory
2. Fast
3. Convenient
4. The very first reason to choose python numpy array is that it occupies less memory as compared to list. Then, it is pretty fast in terms of execution and at the same time it is very convenient to work with numpy. So these are the major advantages that python numpy array has over list. Don’t worry, I am going to prove the above points one by one practically in PyCharm. Consider the below example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | import numpy as np    import time  import sys  S= range(1000)  print(sys.getsizeof(5)\*len(S))    D= np.arange(1000)  print(D.size\*D.itemsize) |

1. O/P –  14000
2. 4000
3. he above output shows that the memory allocated by list (denoted by S) is 14000 whereas the memory allocated by the numpy array is just 4000. From this, you can conclude that there is a major difference between the two and this makes python numpy array as the preferred choice over list.
4. Next, let’s talk how python numpy array is faster and more convenient when compared to list.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | import time  import sys    SIZE = 1000000    L1= range(SIZE)  L2= range(SIZE)  A1= np.arange(SIZE)  A2=np.arange(SIZE)    start= time.time()  result=[(x,y) for x,y in zip(L1,L2)]  print((time.time()-start)\*1000)    start=time.time()  result= A1+A2  print((time.time()-start)\*1000) |

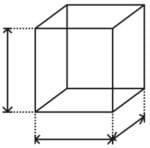
1. O/P – 380.9998035430908  
   49.99995231628418

In the above code, we have defined two lists and two numpy arrays. Then, we have compared the time taken in order to find the sum of lists and sum of numpy arrays both. If you see the output of the above program, there is a significant change in the two values. List took 380ms whereas the numpy array took almost 49ms. Hence, numpy array is faster than list. Now, if you noticed we had run a ‘for’ loop for a list which returns the concatenation of both the lists whereas for numpy arrays, we have just added the two array by simply printing A1+A2. That’s why working with numpy is much easier and convenient when compared to the lists.

Therefore, the above examples proves the point as to why you should go for python numpy array rather than a list!

Moving forward in python numpy tutorial, let’s focus on some of its operations.

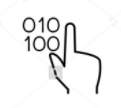
## ****Python NumPy Operations****

* **ndim**:  
  You can find the dimension of the array, whether it is a two-dimensional array or a single dimensional array. So, let us see this practically how we can find the dimensions. In the below code, with the help of ‘ndim’ function, I can find whether the array is of single dimension or multi dimension.

|  |  |
| --- | --- |
| 1  2  3 | import numpy as np  a = np.array([(1,2,3),(4,5,6)])  print(a.ndim) |

Output – 2

Since the output is 2, it is a two-dimensional array (multi dimension).

* **itemsize**:  
  You can calculate the byte size of each element. In the below code, I have defined a single dimensional array and with the help of ‘itemsize’ function, we can find the size of each element.

|  |  |
| --- | --- |
| 1  2  3 | import numpy as np  a = np.array([(1,2,3)])  print(a.itemsize) |

Output – 4

So every element occupies 4 byte in the above numpy array.

* **dtype:**  
  You can find the data type of the elements that are stored in an array. So, if you want to know the data type of a particular element, you can use ‘dtype’ function which will print the datatype along with the size. In the below code, I have defined an array where I have used the same function.



|  |  |
| --- | --- |
| 1  2  3 | import numpy as np  a = np.array([(1,2,3)])  print(a.dtype) |

Output – int32

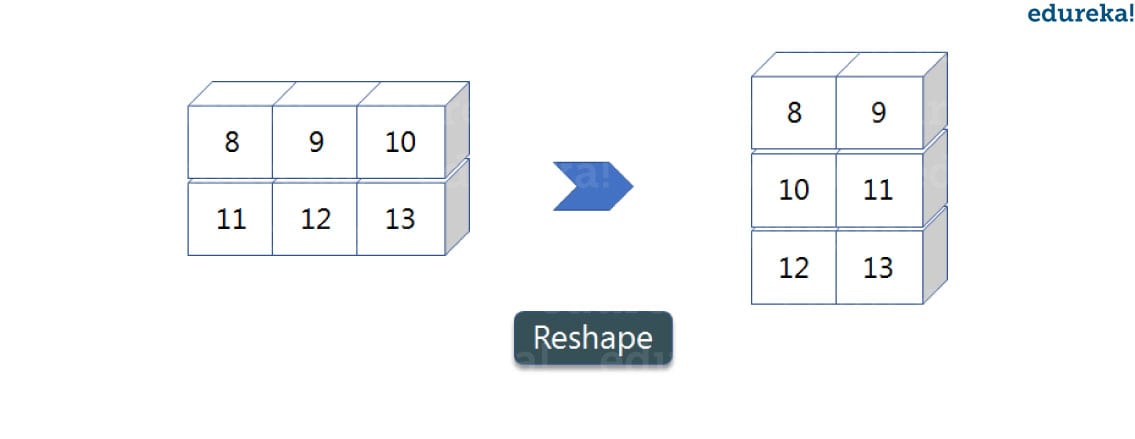
As you can see, the data type of the array is integer 32 bits. Similarly, you can find the size and shape of the array using ‘size’ and ‘shape’ function respectively.

|  |  |
| --- | --- |
| 1  2  3  4 | import numpy as np  a = np.array([(1,2,3,4,5,6)])  print(a.size)  print(a.shape) |

Output – 6 (1,6)

Next, let us move forward and see what are the other operations that you can perform with python numpy module. We can also perform reshape as well as slicing operation using python numpy operation. But, what exactly is reshape and slicing? So let me explain this one by one in this python numpy tutorial.

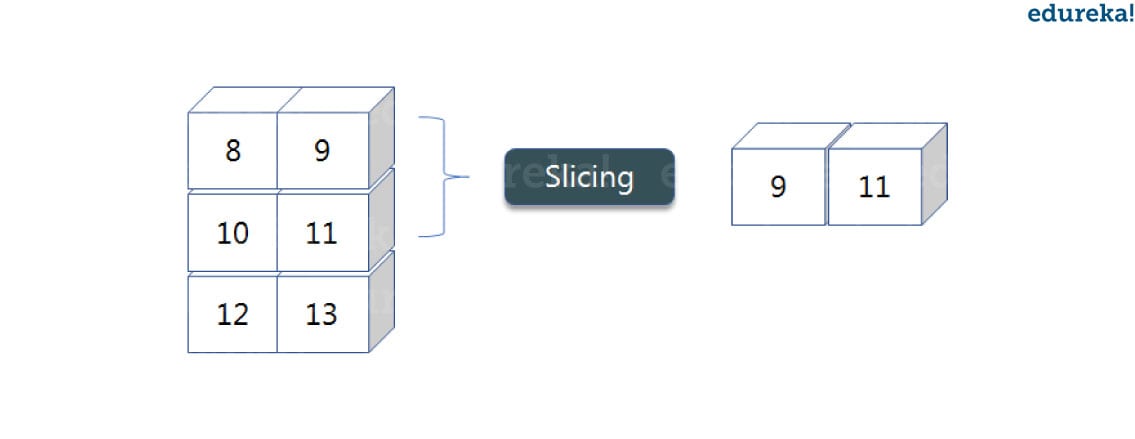
**reshape:**  
Reshape is when you change the number of rows and columns which gives a new view to an object. Now, let us take an example to reshape the below array:



* As you can see in the above image, we have 3 columns and 2 rows which has converted into 2 columns and 3 rows. Let me show you practically how it’s done.

|  |  |
| --- | --- |
| 1  2  3  4  5 | import numpy as np  a = np.array([(8,9,10),(11,12,13)])  print(a)  a=a.reshape(3,2)  print(a) |

* Output – [[ 8 9 10] [11 12 13]] [[ 8 9] [10 11] [12 13]]
* **slicing:**  
  As you can see the ‘reshape’ function has showed its magic. Now, let’s take another operation i.e Slicing. Slicing is basically extracting particular set of elements from an array. This slicing operation is pretty much similar to the one which is there in the list as well. Consider the following example:



Before getting into the above example, let’s see a simple one. We have an array and we need a particular element (say 3) out of a given array. Let’s consider the below example:

|  |  |
| --- | --- |
| 1  2  3 | import numpy as np  a=np.array([(1,2,3,4),(3,4,5,6)])  print(a[0,2]) |

Output – 3

Here, the array(1,2,3,4) is your index 0 and (3,4,5,6) is index 1 of the python numpy array. Therefore, we have printed the second element from the zerothindex.  
Taking one step forward, let’s say we need the 2nd element from the zeroth and first index of the array. Let’s see how you can perform this operation:

|  |  |
| --- | --- |
| 1  2  3 | import numpy as np  a=np.array([(1,2,3,4),(3,4,5,6)])  print(a[0:,2]) |

Output – [3 5]

Here colon represents all the rows, including zero. Now to get the 2nd element, we’ll call index 2 from both of the rows which gives us the value 3 and 5 respectively.

Next, just to remove the confusion, let’s say we have one more row and we don’t want to get its 2nd element printed just as the image above. What we can do in such case?  
Consider the below code:

|  |  |
| --- | --- |
| 1  2  3 | import numpy as np  a=np.array([(8,9),(10,11),(12,13)])  print(a[0:2,1]) |

Output – [9 11]

As you can see in the above code, only 9 and 11 gets printed. Now when I have written 0:2, this does not include the second index of the third row of an array. Therefore, only 9 and 11 gets printed else you will get all the elements i.e [9 11 13].

* **linspace**   
  This is another operation in python numpy which returns evenly spaced numbers over a specified interval. Consider the below example:

|  |  |
| --- | --- |
| 1  2  3 | import numpy as np  a=np.linspace(1,3,10)  print(a) |

Output – [ 1. 1.22222222 1.44444444 1.66666667 1.88888889 2.11111111 2.33333333 2.55555556 2.77777778 3. ]

As you can see in the result, it has printed 10 values between 1 to 3.

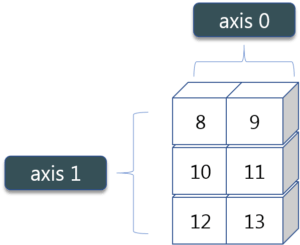
* **max/ min**

Next, we have some more operations in numpy such as to find the minimum, maximum as well the sum of the numpy array. Let’s go ahead in python numpy tutorial and execute it practically.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | import numpy as np    a= np.array([1,2,3])  print(a.min())  print(a.max())  print(a.sum()) |

Output – 1 3 6

You must be finding these pretty basic, but with the help of this knowledge you can perform a lot bigger tasks as well. Now, lets understand the concept of **axis** in python numpy.

As you can see in the figure, we have a numpy array 2\*3. Here the rows are called as axis 1 and the columns are called as axis 0. Now you must be wondering what is the use of these axis?

Suppose you want to calculate the sum of all the columns, then you can make use of axis. Let me show

* you practically, how you can implement axis in your PyCharm:

|  |  |
| --- | --- |
| 1  2 | a= np.array([(1,2,3),(3,4,5)])  print(a.sum(axis=0)) |

Output – [4 6 8]

Therefore, the sum of all the columns are added where 1+3=4, 2+4=6 and 3+5=8. Similarly, if you replace the axis by 1, then it will print [6 12] where all the rows get added.

* **Square Root & Standard Deviation**

There are various mathematical functions that can be performed using python numpy. You can find the square root, standard deviation of the array. So, let’s implement these operations:

|  |  |
| --- | --- |
| 1  2  3  4 | import numpy as np  a=np.array([(1,2,3),(3,4,5,)])  print(np.sqrt(a))  print(np.std(a)) |

Output – [[ 1. 1.41421356 1.73205081]  
[ 1.73205081 2. 2.23606798]]  
1.29099444874

As you can see the output above, the square root of all the elements are printed. Also, the standard deviation is printed for the above array i.e how much each element varies from the mean value of the python numpy array.

* **AdditionOperation**  
  You can perform more operations on numpy array i.e addition, subtraction,multiplication and division of the two matrices. Let me go ahead in python numpy tutorial, and show it to you practically:

|  |  |
| --- | --- |
| 1  2  3  4 | import numpy as np  x= np.array([(1,2,3),(3,4,5)])  y= np.array([(1,2,3),(3,4,5)])  print(x+y) |

Output – [[ 2 4 6] [ 6 8 10]]

This is extremely simple! Right? Similarly, we can perform other operations such as subtraction, multiplication and division. Consider the below example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | import numpy as np  x= np.array([(1,2,3),(3,4,5)])  y= np.array([(1,2,3),(3,4,5)])  print(x-y)  print(x\*y)  print(x/y) |

Output – [[0 0 0] [0 0 0]]  
[[ 1 4 9] [ 9 16 25]]  
[[ 1. 1. 1.] [ 1. 1. 1.]]

* **Vertical & Horizontal Stacking**  
  Next, if you want to concatenate two arrays and not just add them, you can perform it using two ways – *vertical stacking* and *horizontal stacking*. Let me show it one by one in this python numpy tutorial.

|  |  |
| --- | --- |
| 1  2  3  4  5 | import numpy as np  x= np.array([(1,2,3),(3,4,5)])  y= np.array([(1,2,3),(3,4,5)])  print(np.vstack((x,y)))  print(np.hstack((x,y))) |

Output – [[1 2 3] [3 4 5] [1 2 3] [3 4 5]]  
[[1 2 3 1 2 3] [3 4 5 3 4 5]]

* **ravel**  
  There is one more operation where you can convert one numpy array into a single column i.e *ravel*. Let me show how it is implemented practically:

|  |  |
| --- | --- |
| 1  2  3 | import numpy as np  x= np.array([(1,2,3),(3,4,5)])  print(x.ravel()) |

Output – [ 1 2 3 3 4 5]

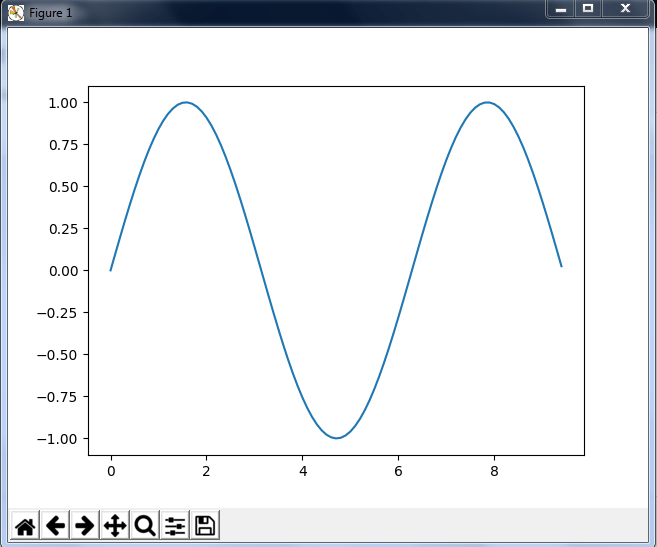
Let’s move forward in python numpy tutorial, and look at some of its special functions.

## ****Python Numpy Special Functions****

* There are various special functions available in numpy such as sine, cosine, tan, log etc. First, let’s begin with sine function where we will learn to plot its graph. For that, we need to import a module called matplotlib. To understand the basics and practical implementations of this module, you can refer **[Matplotlib Tutorial](https://www.edureka.co/blog/python-matplotlib-tutorial/" \t "_blank)**. Moving ahead with python numpy tutorial, let’s see how these graphs are plotted.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | import numpy as np  import matplotlib.pyplot as plt  x= np.arange(0,3\*np.pi,0.1)  y=np.sin(x)  plt.plot(x,y)  plt.show() |

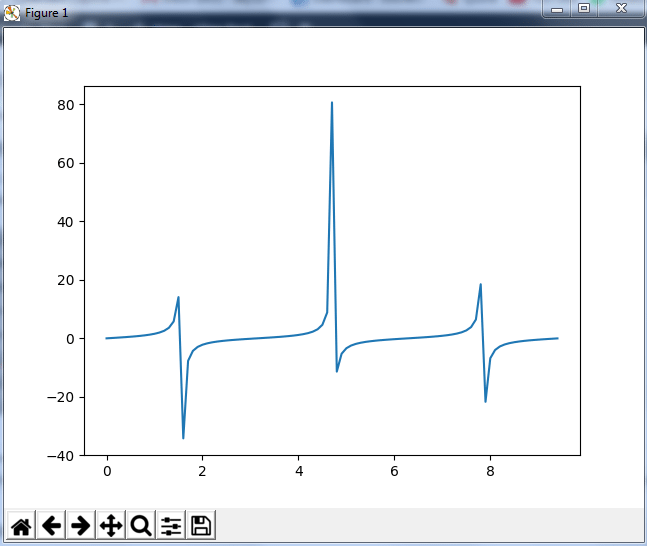
* Output –



Similarly, you can plot a graph for any trigonometric function such as cos, tan etc. Let me show you one more example where you can plot a graph of another function, let’s say ***tan****.*

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | import numpy as np  import matplotlib.pyplot as plt  x= np.arange(0,3\*np.pi,0.1)  y=np.tan(x)  plt.plot(x,y)  plt.show() |

Output –



Moving forward with python numpy tutorial, let’s see some other special functionality in numpy array such as exponential and logarithmic function. Now in exponential, the *e*value is somewhere equal to 2.7 and in log, it is actually *log base 10*.  When we talk about natural log i.e log base e, it is referred as Ln. So let’s see how it is implemented practically:

|  |  |
| --- | --- |
| 1  2 | a= np.array([1,2,3])  print(np.exp(a)) |

Output – [ 2.71828183   7.3890561   20.08553692]

As you can see the above output, the exponential values are printed i.e *e* raise to the power 1 is *e,* which gives the result as 2.718… Similarly, *e* raise to the power of 2 gives the value somewhere near 7.38 and so on. Next, in order to calculate log, let’s see how you can implement it:

|  |  |
| --- | --- |
| 1  2  3  4 | import numpy as np  import matplotlib.pyplot as plt  a= np.array([1,2,3])  print(np.log(a)) |

Output – [ 0.          0.69314718  1.09861229]

Here, we have calculated natural log which gives the value as displayed above. Now, if we want log base 10 instead of Ln or natural log, you can follow the below code:

|  |  |
| --- | --- |
| 1  2  3  4 | import numpy as np  import matplotlib.pyplot as plt  a= np.array([1,2,3])  print(np.log10(a)) |

Output – [ 0.        0.30103      0.47712125]

By this, we come to the end of this python numpy tutorial. We have covered all the basics of python numpy, so you can start practicing now. The more you practice, the more you will learn.

# Matplotlib Tutorial – Python Matplotlib Library with Examples

 I discussed about a numerical library of python called [***Python NumPy***](https://www.edureka.co/blog/python-numpy-tutorial/). In this blog, I will be talking about another library, **Python Matplotlib**. matplotlib.pyplot is a python package used for 2D graphics. Learning to use this library efficiently is also an essential part of [***Python Certification curriculum***](https://www.edureka.co/python-programming-certification-training). Below is the sequence in which I will be covering all the topics of python matplotlib:

* [What Is Python Matplotlib](https://www.edureka.co/blog/python-matplotlib-tutorial/" \l "matplotlib)?
* [Types Of Plots](https://www.edureka.co/blog/python-matplotlib-tutorial/#types)–  [Bar Graph](https://www.edureka.co/blog/python-matplotlib-tutorial/#Bargraph)  
  – [Histogram](https://www.edureka.co/blog/python-matplotlib-tutorial/#Histogram)  
  – [Scatter Plot](https://www.edureka.co/blog/python-matplotlib-tutorial/#Scatter)  
  – [Area Plot](https://www.edureka.co/blog/python-matplotlib-tutorial/#AreaPlot)  
  – [Pie Chart](https://www.edureka.co/blog/python-matplotlib-tutorial/#PiePlot)
* [Working With Multiple Plots](https://www.edureka.co/blog/python-matplotlib-tutorial/#multiplePlots)

## ****What Is Python Matplotlib?****

**matplotlib.pyplot** is a plotting library used for 2D graphics in python programming language. It can be used in python scripts, shell, web application servers and other graphical user interface toolkits.

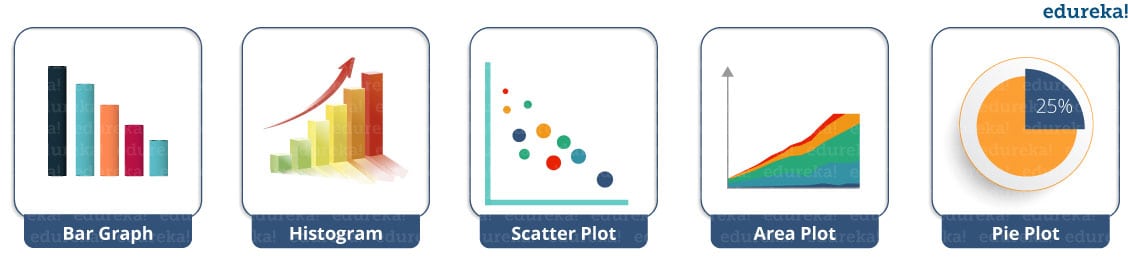
There are several toolkits which are available that extend python matplotlib functionality. Some of them are separate downloads, others can be shipped with the matplotlib source code but have external dependencies.

* + **Basemap**: It is a map plotting toolkit with various map projections, coastlines and political boundaries.
  + **Cartopy**: It is a mapping library featuring object-oriented map projection definitions, and arbitrary point, line, polygon and image transformation capabilities.
  + **Excel tools**: Matplotlib provides utilities for exchanging data with Microsoft Excel.
  + **Mplot3d**: It is used for 3-D plots.
  + **Natgrid**: It is an interface to the natgrid library for irregular gridding of the spaced data.

Next, let us move forward in this blog and explore different types of plots available in python matplotlib.

## ****Python Matplotlib : Types of Plots****

There are various plots which can be created using python matplotlib. Some of them are listed below:

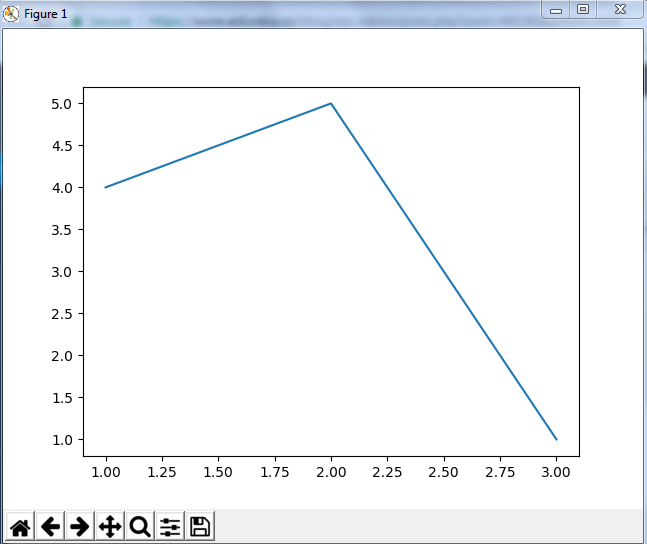


I will demonstrate each one of them in detail.

But before that, let me show you very basic codes in python matplotlib in order to generate a simple graph.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | from matplotlib import pyplot as plt     #Plotting to our canvas     plt.plot([1,2,3],[4,5,1])     #Showing what we plotted     plt.show() |

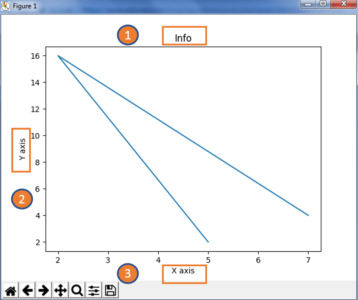
Output –



So, with three lines of code, you can generate a basic graph using python matplotlib. Simple, isn’t it?  
Let us see how can we add title, labels to our graph created by python matplotlib library to bring in more meaning to it. Consider the below example:

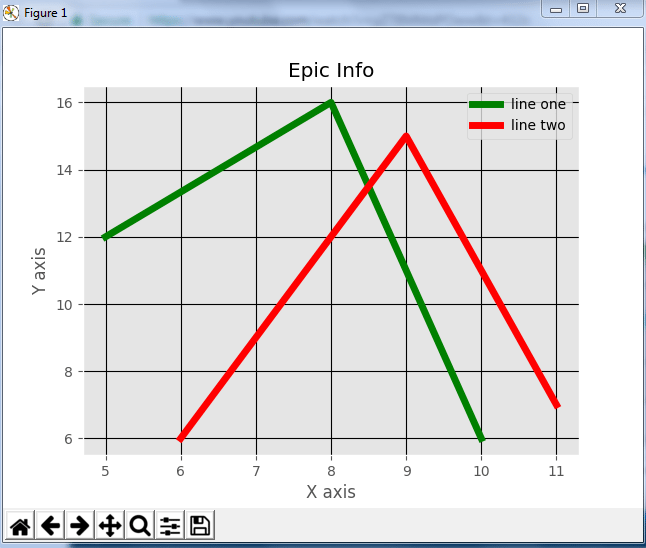
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | from matplotlib import pyplot as plt    x = [5,2,7]  y = [2,16,4]  plt.plot(x,y)  plt.title('Info')  plt.ylabel('Y axis')  plt.xlabel('X axis')  plt.show() |

Output –



You can even try many styling techniques to create a better graph. What if you want to change the width or color of a particular line or what if you want to have some grid lines, there you need styling! So, let me show you how to add style to a graph using python matplotlib. First, you need to import the style package from python matplotlib library and then use styling functions as shown in below code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | from matplotlib import pyplot as plt  from matplotlib import style    style.use('ggplot')  x = [5,8,10]  y = [12,16,6]  x2 = [6,9,11]  y2 = [6,15,7]  plt.plot(x,y,'g',label='line one', linewidth=5)  plt.plot(x2,y2,'c',label='line two',linewidth=5)  plt.title('Epic Info')  plt.ylabel('Y axis')  plt.xlabel('X axis')  plt.legend()  plt.grid(True,color='k')  plt.show() |

Output –  
X

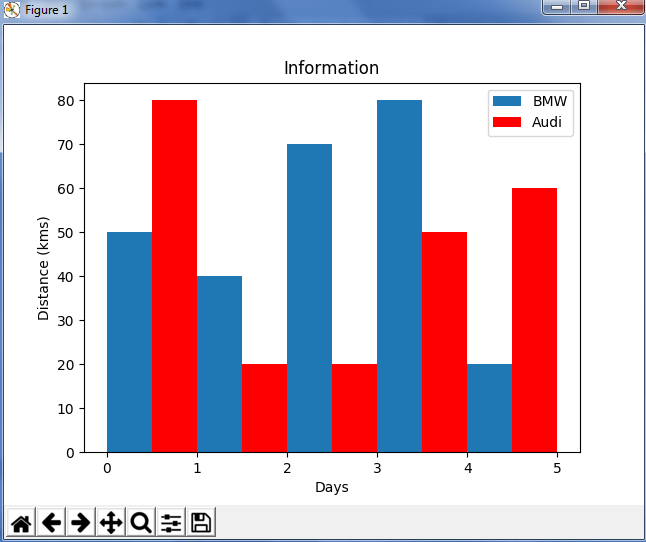
Next in this python matplotlib blog, we will understand different kinds of plots. Let’s start with bar graph!

### ****Python Matplotlib: Bar Graph****

First, let us understand why do we need a bar graph. A bar graph uses bars to compare data among different categories. It is well suited when you want to measure the changes over a period of time. It can be represented horizontally or vertically. Also, the important thing to keep in mind is that longer the bar, greater is the value. Now, let us practically implement it using python matplotlib.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | from matplotlib import pyplot as plt    plt.bar([0.25,1.25,2.25,3.25,4.25],[50,40,70,80,20],  label="BMW",width=.5)  plt.bar([.75,1.75,2.75,3.75,4.75],[80,20,20,50,60],  label="Audi", color='r',width=.5)  plt.legend()  plt.xlabel('Days')  plt.ylabel('Distance (kms)')  plt.title('Information')  plt.show() |

Output –



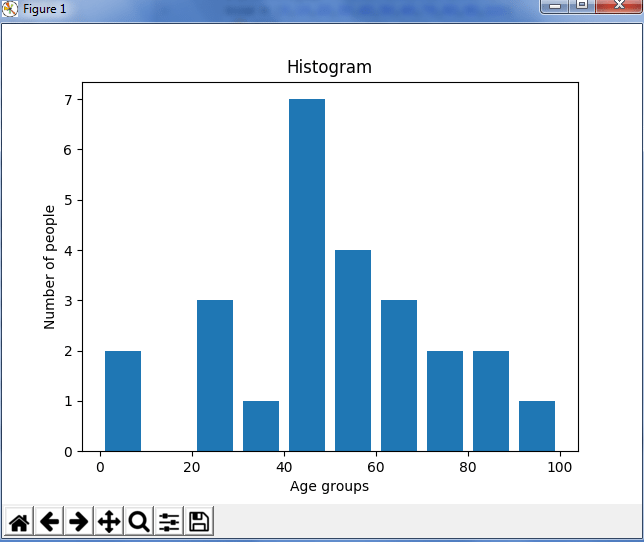
In the above plot, I have displayed the comparison between the distance covered by two cars BMW and Audi over a period of 5 days. Next, let us move on to another kind of plot using python matplotlib – Histogram.

### ****Python Matplotlib – Histogram****

Let me first tell you the difference between a bar graph and a histogram. Histograms are used to show a distribution whereas a bar chart is used to compare different entities. Histograms are useful when you have arrays or a very long list. Let’s consider an example where I have to plot the age of population with respect to bin. Now, bin refers to the range of values that are divided into series of intervals. Bins are usually created of the same size. In the below code, I have created the bins in the interval of 10 which means the first bin contains elements from 0 to 9, then 10 to 19 and so on.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | import matplotlib.pyplot as plt  population\_age = [22,55,62,45,21,22,34,42,42,4,2,102,95,85,55,110,120,70,65,55,111,115,80,75,65,54,44,43,42,48]  bins = [0,10,20,30,40,50,60,70,80,90,100]  plt.hist(population\_age, bins, histtype='bar', rwidth=0.8)  plt.xlabel('age groups')  plt.ylabel('Number of people')  plt.title('Histogram')  plt.show() |

Output –



As you can see in the above plot, we got age groups with respect to the bins. Our biggest age group is between 40 and 50.

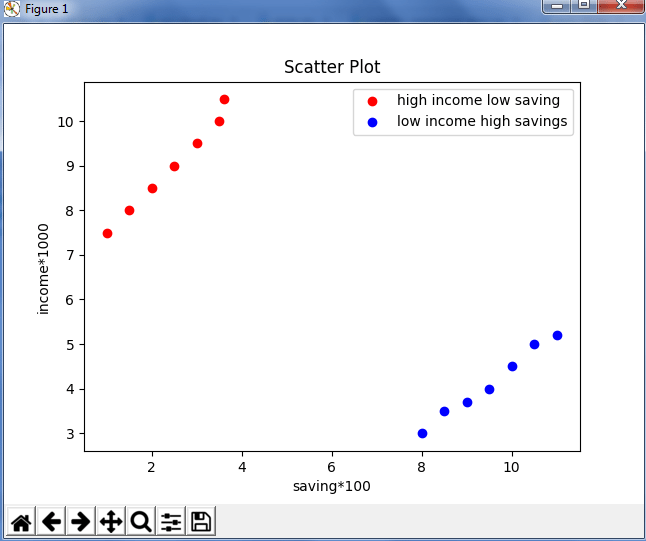
### ****Python Matplotlib : Scatter Plot****

Usually we need scatter plots in order to compare variables, for example, how much one variable is affected by another variable to build a relation out of it. The data is displayed as a collection of points, each having the value of one variable which determines the position on the horizontal axis and the value of other variable determines the position on the vertical axis.

Consider the below example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | import matplotlib.pyplot as plt  x = [1,1.5,2,2.5,3,3.5,3.6]  y = [7.5,8,8.5,9,9.5,10,10.5]    x1=[8,8.5,9,9.5,10,10.5,11]  y1=[3,3.5,3.7,4,4.5,5,5.2]    plt.scatter(x,y, label='high income low saving',color='r')  plt.scatter(x1,y1,label='low income high savings',color='b')  plt.xlabel('saving\*100')  plt.ylabel('income\*1000')  plt.title('Scatter Plot')  plt.legend()  plt.show() |

Output –



As you can see in the above graph, I have plotted two scatter plots based on the inputs specified in the above code. The data is displayed as a collection of points having ‘high income low salary’ and ‘low income high salary’.

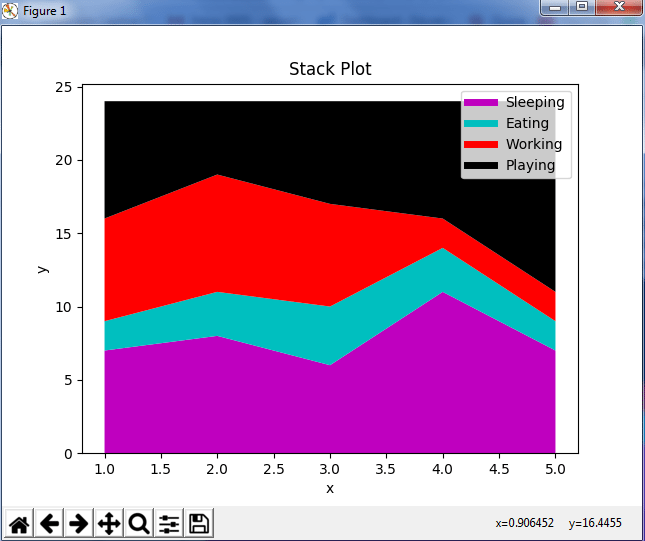
Next, let us understand area plot or you can also say Stack plot using python matplotlib.

### ****Python Matplotlib : Area Plot****

Area plots are pretty much similar to the line plot. They are also known as stack plots. These plots can be used to track changes over time for two or more related groups that make up one whole category. For example, let’s compile the work done during a day into categories, say sleeping, eating, working and playing. Consider the below code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | import matplotlib.pyplot as plt  days = [1,2,3,4,5]     sleeping =[7,8,6,11,7]   eating = [2,3,4,3,2]   working =[7,8,7,2,2]   playing = [8,5,7,8,13]     plt.plot([],[],color='m', label='Sleeping', linewidth=5)   plt.plot([],[],color='c', label='Eating', linewidth=5)   plt.plot([],[],color='r', label='Working', linewidth=5)   plt.plot([],[],color='k', label='Playing', linewidth=5)     plt.stackplot(days, sleeping,eating,working,playing, colors=['m','c','r','k'])     plt.xlabel('x')   plt.ylabel('y')   plt.title('Stack Plot')   plt.legend()   plt.show() |

Output –

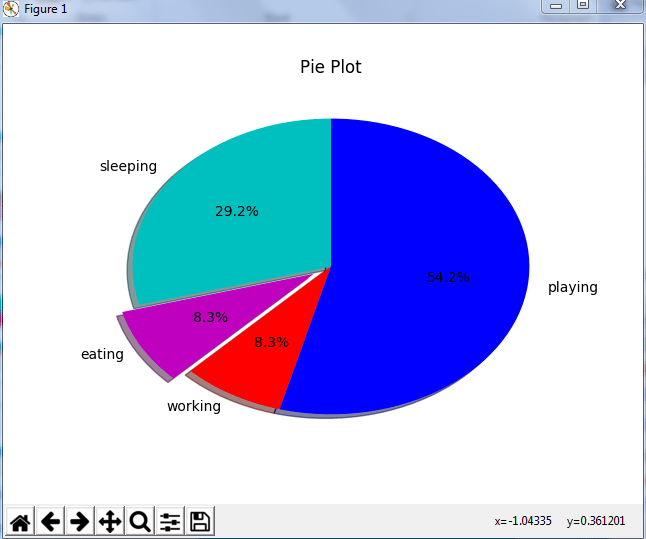


As we can see in the above image, we have time spent based on the categories. Therefore, area plot or stack plot is used to show trends over time, among different attributes. Next, let us move to our last yet most frequently used plot – Pie chart.

### ****Python Matplotlib : Pie Chart****

A pie chart refers to a circular graph which is broken down into segments i.e. slices of pie. It is basically used to show the percentage or proportional data where each slice of pie represents a category. Let’s have a look at the below example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | import matplotlib.pyplot as plt    days = [1,2,3,4,5]    sleeping =[7,8,6,11,7]  eating = [2,3,4,3,2]  working =[7,8,7,2,2]  playing = [8,5,7,8,13]  slices = [7,2,2,13]  activities = ['sleeping','eating','working','playing']  cols = ['c','m','r','b']    plt.pie(slices,    labels=activities,    colors=cols,    startangle=90,    shadow= True,    explode=(0,0.1,0,0),    autopct='%1.1f%%')    plt.title('Pie Plot')  plt.show() |

Output –

In the above pie chart, I have divided the circle into 4 sectors or slices which represents the respective category (playing, sleeping, eating and working) along with the percentage they hold. Now, if you have noticed these slices adds up to 24 hrs, but the calculation of pie slices is done automatically for you. In this way, pie charts are really useful as you don’t have to be the one who calculates the percentage or the slice of the pie.

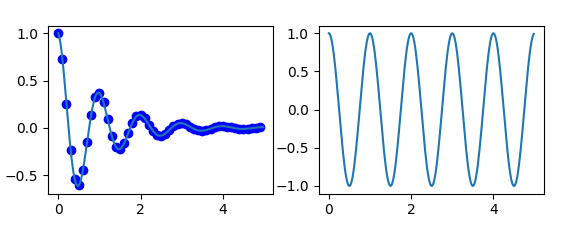
Next in python matplotlib, let’s understand how to work with multiple plots.

## ****Python Matplotlib : Working With Multiple Plots****

I have discussed about multiple types of plots in python matplotlib such as bar plot, scatter plot, pie plot, area plot etc. Now, let me show you how to handle multiple plots. For this, I have to import numpy module which I discussed in my previous blog on [Python Numpy](https://www.edureka.co/blog/python-numpy-tutorial/). Let me implement it practically, consider the below example.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | import numpy as np  import matplotlib.pyplot as plt    def f(t):      return np.exp(-t) \* np.cos(2\*np.pi\*t)  t1 = np.arange(0.0, 5.0, 0.1)  t2 = np.arange(0.0, 5.0, 0.02)  plt.subplot(221)  plt.plot(t1, f(t1), 'bo', t2, f(t2))  plt.subplot(222)  plt.plot(t2, np.cos(2\*np.pi\*t2))  plt.show() |

Output -



The code is pretty much similar to the previous examples that you have seen but there is one new concept here i.e. subplot. The subplot() command specifies numrow, numcol, fignum which ranges from 1 to numrows\*numcols. The commas in this command are optional if numrows\*numcols<10. So subplot (221) is identical to subplot (2,2,1). Therefore, subplots helps us to plot multiple graphs in which you can define it by aligning vertically or horizontally. In the above example, I have aligned it horizontally.

Apart from these, python matplotlib has some disadvantages. Some of them are listed below:

* They are heavily reliant on other packages, such as NumPy.
* It only works for python, so it is hard or impossible to be used in languages other than python. (But it can be used from Julia via PyPlot package).

We have come to an end of this python matplotlib tutorial. I have covered all the basics of matplotlib, so you can start practicing now. I hope you guys are clear about each and every aspect that I have discussed above. After this python matplotlib blog, I will be coming up with more blogs on python class, scikit learn and array. Stay tuned!

# Python Pandas Tutorial : Learn Pandas for Data Analysis

In this blog, we will be discussing data analysis using Pandas in Python. Today, Python certification is a hot skill in the industry that surpassed PHP in 2017 and C# in 2018 in terms of overall popularity and use. Before talking about Pandas, one must understand the concept of Numpy arrays. Why? Because Pandas is an open source software library which is built on top of [NumPy](https://www.edureka.co/blog/python-numpy-tutorial/" \t "_blank). In this Python Pandas Tutorial, I will take you through the following topics, which will serve as fundamentals for the upcoming blogs:

* [What is Pandas?](https://www.edureka.co/blog/python-pandas-tutorial/#WhatIsPandas)
* [Pandas Operation](https://www.edureka.co/blog/python-pandas-tutorial/#PandasOperations)
  + [Slicing the data frame](https://www.edureka.co/blog/python-pandas-tutorial/#Slicing)
  + [Merging & Joining](https://www.edureka.co/blog/python-pandas-tutorial/#Merging&Joining)
  + [Concatenation](https://www.edureka.co/blog/python-pandas-tutorial/#Concatenation)
  + [Changing the index](https://www.edureka.co/blog/python-pandas-tutorial/#ChangeIndex)
  + [Change Column headers](https://www.edureka.co/blog/python-pandas-tutorial/#ChangeHeaders)
  + [Data munging](https://www.edureka.co/blog/python-pandas-tutorial/#Munging)
* [Use-Case: Analyze youth unemployment data](https://www.edureka.co/blog/python-pandas-tutorial/#Use-Case)

Let’s get started. :-)

## ****What is Python Pandas?****

Pandas is used for data manipulation, analysis and cleaning. Python pandas is well suited for different kinds of data, such as:

* Tabular data with heterogeneously-typed columns
* Ordered and unordered time series data
* Arbitrary matrix data with row & column labels
* Unlabelled data
* Any other form of observational or statistical data sets

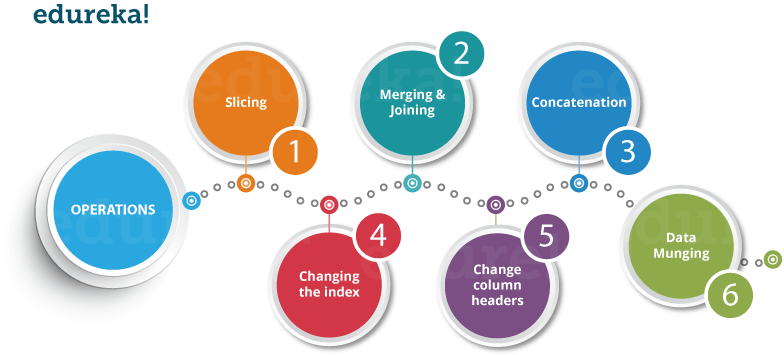
### ****How to install Pandas?****

To install Python Pandas, go to your command line/ terminal and type “pip install pandas” or else, if you have anaconda installed in your system, just type in “conda install pandas”. Once the installation is completed, go to your IDE (Jupyter, PyCharm etc.) and simply import it by typing: “import pandas as pd”

Moving ahead in Python pandas tutorial, let’s take a look at some of its operations:

## ****Python Pandas Operations****

Using Python pandas, you can perform a lot of operations with series, data frames, missing data, group by etc. Some of the common operations for data manipulation are listed below:



Now, let us understand all these operations one by one.

### ****Slicing the Data Frame****

In order to perform slicing on data, you need a data frame. Don’t worry, data frame is a 2-dimensional data structure and a most common pandas object. So first, let’s create a data frame.

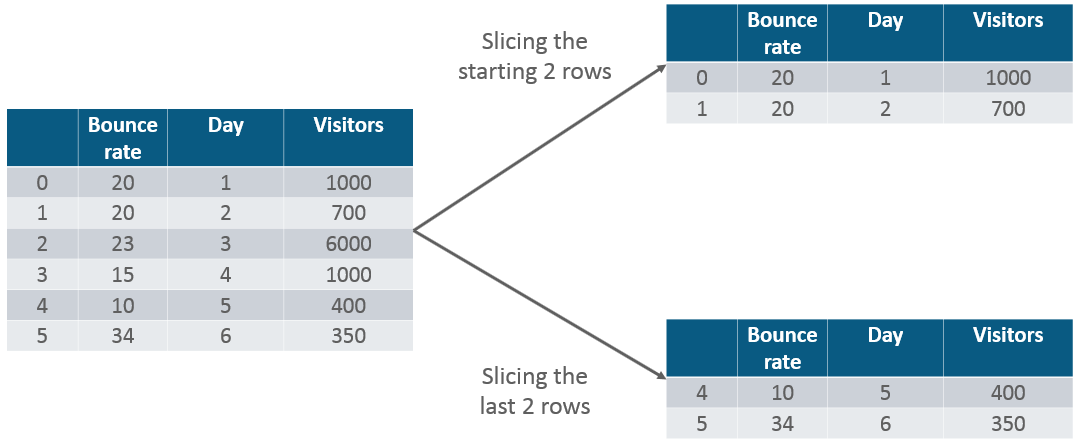
Refer the below code for its implementation in PyCharm:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import pandas as pd    XYZ\_web= {'Day':[1,2,3,4,5,6], "Visitors":[1000, 700,6000,1000,400,350], "Bounce\_Rate":[20,20, 23,15,10,34]}    df= pd.DataFrame(XYZ\_web)    print(df) |

#### ****Output:****

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | Bounce\_Rate Day Visitors  0     20          1   1000  1     20          2   700  2     23          3   6000  3     15          4   1000  4     10          5   400  5     34          6   350 |

The code above will convert a dictionary into a pandas Data Frame along with index to the left. Now, let us slice a particular column from this data frame. Refer the image below:



|  |  |
| --- | --- |
| 1 | print(df.head(2)) |

#### ****Output:****

|  |  |
| --- | --- |
| 1  2  3 | Bounce\_Rate Day Visitors  0      20         1   1000  1      20         2    700 |

Similarly, if you want the last two rows of the data, type in the below command:

|  |  |
| --- | --- |
| 1 | print(df.tail(2)) |

**Output:**

|  |  |
| --- | --- |
| 1  2  3 | Bounce\_Rate Day Visitors  4      10      5    400  5      34      6    350 |

Next in Python Pandas tutorial, let us perform merging and joining.

### ****Merging & Joining****

In merging, you can merge two data frames to form a single data frame. You can also decide which columns you want to make common. Let me implement that practically, first I will create three data frames, which has some key-value pairs and then merge the data frames together. Refer the code below:

|  |  |
| --- | --- |
| 1  2  3  4  5 | HPI   IND\_GDP Int\_Rate  0  80      50      2  1  90      45      1  2  70      45      2  3  60      67      3 |

#### ****Output:****

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | import pandas as pd    df1= pd.DataFrame({ "HPI":[80,90,70,60],"Int\_Rate":[2,1,2,3],"IND\_GDP":[50,45,45,67]}, index=[2001, 2002,2003,2004])    df2=pd.DataFrame({ "HPI":[80,90,70,60],"Int\_Rate":[2,1,2,3],"IND\_GDP":[50,45,45,67]}, index=[2005, 2006,2007,2008])    merged= pd.merge(df1,df2)    print(merged) |

As you can see above, the two data frames has merged into a single data frame. Now, you can also specify the column which you want to make common. For example, I want the “HPI” column to be common and for everything else, I want separate columns. So, let me implement that practically:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | df1 = pd.DataFrame({"HPI":[80,90,70,60],"Int\_Rate":[2,1,2,3], "IND\_GDP":[50,45,45,67]}, index=[2001, 2002,2003,2004])    df2 = pd.DataFrame({"HPI":[80,90,70,60],"Int\_Rate":[2,1,2,3],"IND\_GDP":[50,45,45,67]}, index=[2005, 2006,2007,2008])    merged= pd.merge(df1,df2,on ="HPI")    print(merged) |

#### ****Output:****

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | IND\_GDP  Int\_Rate  Low\_Tier\_HPI  Unemployment  2001     50      2         50.0            1.0  2002     45      1         NaN             NaN  2003     45      2         45.0            3.0  2004     67      3         67.0            5.0  2004     67      3         34.0            6.0 |

Next, let us understand **joining** in python pandas tutorial. It is yet another convenient method to combine two differently indexed dataframes into a single result dataframe. This is quite similar to the “merge” operation, except the joining operation will be on the “index” instead of  the “columns”. Let us implement it practically.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | df1 = pd.DataFrame({"Int\_Rate":[2,1,2,3], "IND\_GDP":[50,45,45,67]}, index=[2001, 2002,2003,2004])    df2 = pd.DataFrame({"Low\_Tier\_HPI":[50,45,67,34],"Unemployment":[1,3,5,6]}, index=[2001, 2003,2004,2004])    joined= df1.join(df2)  print(joined) |

#### ****Output:****

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | IND\_GDP  Int\_Rate Low\_Tier\_HPI  Unemployment  2001     50       2         50.0           1.0  2002     45       1         NaN            NaN  2003     45       2         45.0           3.0  2004     67       3         67.0           5.0  2004     67       3         34.0           6.0 |

As you can notice in the above output, in year 2002(index), there is no value attached to columns “low\_tier\_HPI” and “unemployment”, therefore it has printed NaN (Not a Number). Later in 2004, both the values are available, therefore it has printed the respective values.

Moving ahead in Python pandas tutorial, let us understand how to concatenate two data data frames.

### ****Concatenation****

Concatenation basically glues the dataframes together. You can select the dimension on which you want to concatenate. For that, just use “pd.concat” and pass in the list of dataframes to concatenate together. Consider the below example.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | df1 = pd.DataFrame({"HPI":[80,90,70,60],"Int\_Rate":[2,1,2,3], "IND\_GDP":[50,45,45,67]}, index=[2001, 2002,2003,2004])    df2 = pd.DataFrame({"HPI":[80,90,70,60],"Int\_Rate":[2,1,2,3],"IND\_GDP":[50,45,45,67]}, index=[2005, 2006,2007,2008])    concat= pd.concat([df1,df2])    print(concat) |

#### ****Output:****

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | HPI  IND\_GDP Int\_Rate  2001    80    50       2  2002    90    45       1  2003    70    45       2  2004    60    67       3  2005    80    50       2  2006    90    45       1  2007    70    45       2  2008    60    67       3 |

As you can see above, the two dataframes are glued together in a single dataframe, where the index starts from 2001 all the way upto 2008. Next, you can also specify axis=1 in order to join, merge or cancatenate along the columns. Refer the code below:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | df1 = pd.DataFrame({"HPI":[80,90,70,60],"Int\_Rate":[2,1,2,3], "IND\_GDP":[50,45,45,67]}, index=[2001, 2002,2003,2004])    df2 = pd.DataFrame({"HPI":[80,90,70,60],"Int\_Rate":[2,1,2,3],"IND\_GDP":[50,45,45,67]}, index=[2005, 2006,2007,2008])    concat= pd.concat([df1,df2],axis=1)    print(concat) |

**Output:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | HPI  IND\_GDP  Int\_Rate HPI  IND\_GDP Int\_Rate  2001   80.0  50.0       2.0   NaN    NaN     NaN  2002   90.0  45.0       1.0   NaN    NaN     NaN  2003   70.0  45.0       2.0   NaN    NaN     NaN  2004   60.0  67.0       3.0   NaN    NaN     NaN  2005   NaN   NaN        NaN   80.0   50.0    2.0  2006   NaN   NaN        NaN   90.0   45.0    1.0  2007   NaN   NaN        NaN   70.0   45.0    2.0  2008   NaN   NaN        NaN   60.0   67.0    3.0 |

As you can above, there are bunch of missing values. This happens because the dataframes didn’t have values for all the indexes you want to concatenate on. Therefore, you should make sure that you have all the information lining up correctly when you join or concatenate on the axis.

### ****Change the index****

Next in python pandas tutorial, we’ll understand how to change the index values in a dataframe. For example, let us create a dataframe with some key value pairs in a dictionary and change the index values. Consider the example below:

Let us see how it actually happens:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import pandas as pd    df= pd.DataFrame({"Day":[1,2,3,4], "Visitors":[200, 100,230,300], "Bounce\_Rate":[20,45,60,10]})    df.set\_index("Day", inplace= True)    print(df) |

**Output:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Bounce\_Rate  Visitors  Day  1      20           200  2      45           100  3      60           230  4      10           300 |

As you can notice in the output above, the index value has been changed with respect to the “Day” column.

### ****Change the Column Headers****

Let us now change the headers of column in this python pandas tutorial. Let us take the same example, where I will change the column header from “Visitors” to “Users”. So, let me implement it practically.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import pandas as pd    df = pd.DataFrame({"Day":[1,2,3,4], "Visitors":[200, 100,230,300], "Bounce\_Rate":[20,45,60,10]})    df = df.rename(columns={"Visitors":"Users"})    print(df) |

#### ****Output:****

|  |  |
| --- | --- |
| 1  2  3  4  5 | Bounce\_Rate  Day  Users  0    20         1    200  1    45         2    100  2    60         3    230  3    10         4    300 |

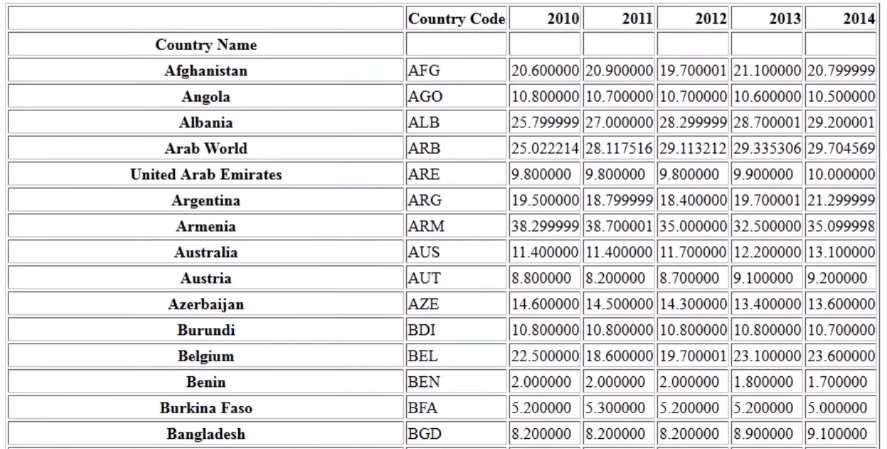
As you see above, column header “Visitors” has been changed to “Users”. Next in python pandas tutorial, let us perform data munging.

### ****Data Munging****

In Data munging, you can convert a particular data into a different format. For example, if you have a .csv file, you can convert it into .html or any other data format as well. So, let me implement this practically.

|  |  |
| --- | --- |
| 1  2  3  4  5 | import pandas as pd    country= pd.read\_csv("D:UsersAayushiDownloadsworld-bank-youth-unemploymentAPI\_ILO\_country\_YU.csv",index\_col=0)    country.to\_html('edu.html') |

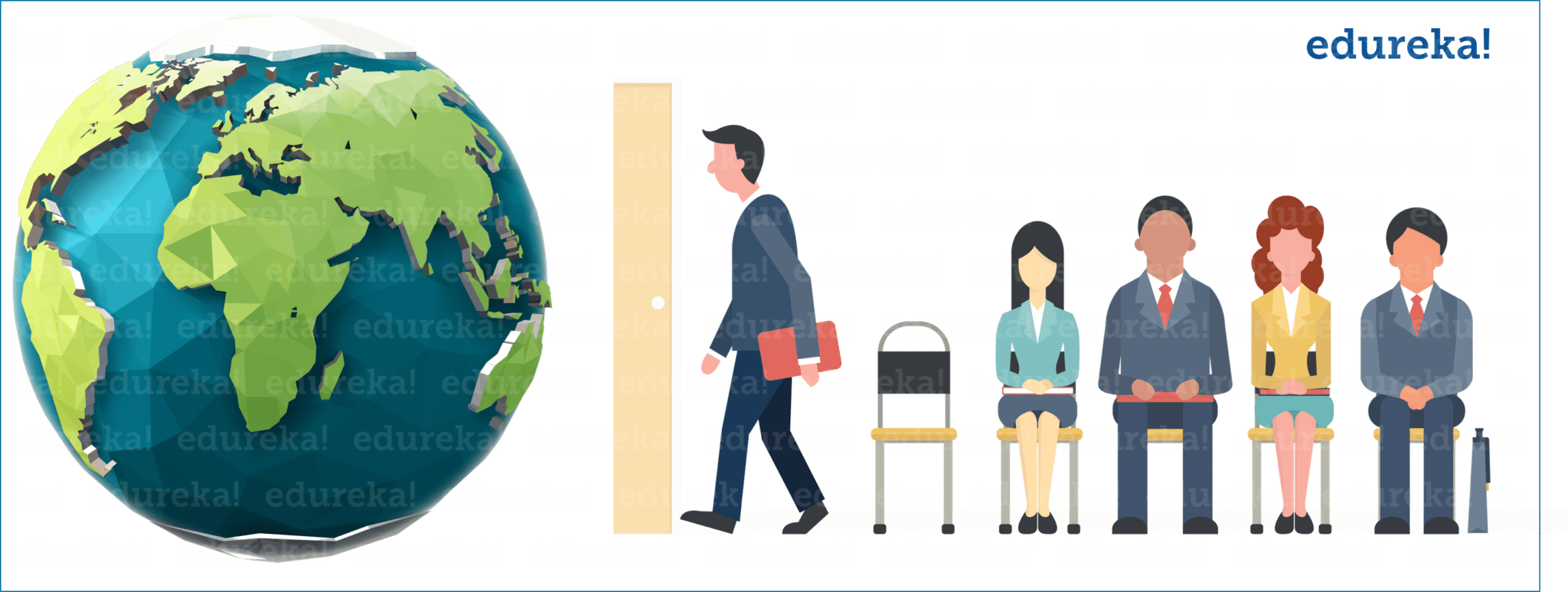
Once you run this code, a HTML file will be created named “edu.html”. You can directly copy the path of the file and paste it in your browser which displays the data in a HTML format. Refer the below screenshot:



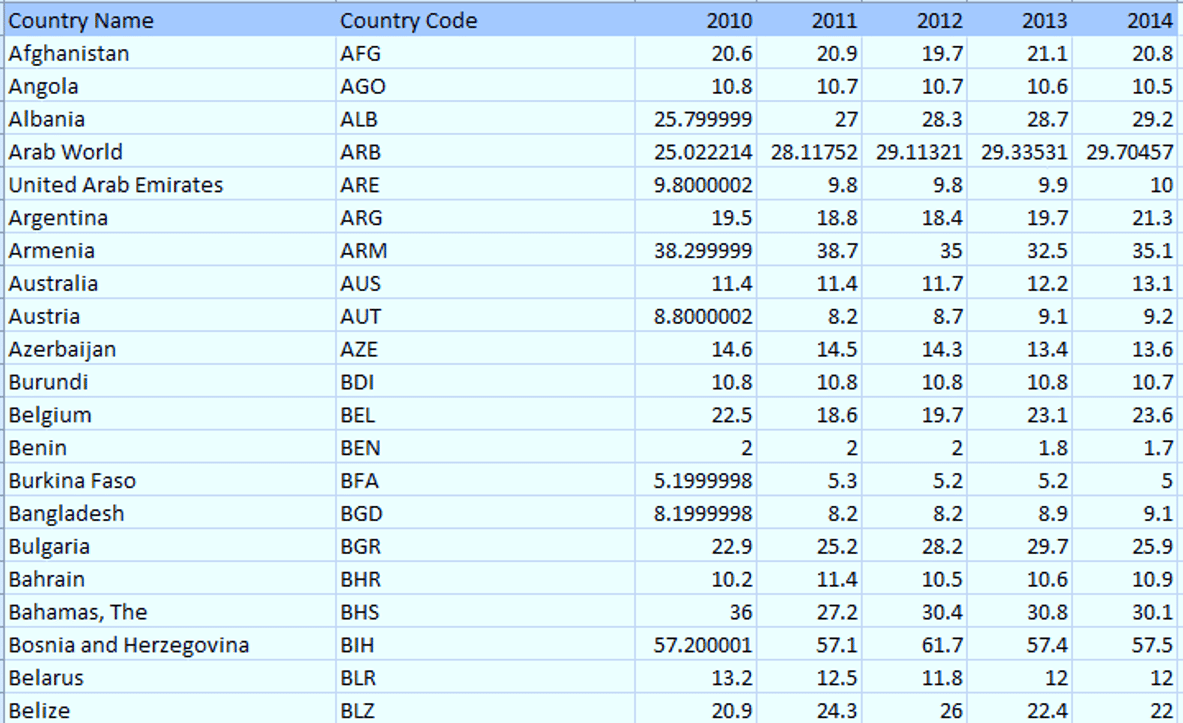
Next in python pandas tutorial, let’s have a look at a use-case which talks about the global youth unemployment.

## ****Python Pandas Tutorial: Use Case to Analyze Youth Unemployment Data****

**Problem Statement**: You are given a dataset whichcomprises of the percentage of unemployed youth globally from 2010 to 2014. You have to use this dataset and find the change in the percentage of youth for every country from 2010-2011.



First, let us understand the dataset which contains the columns as Country Name, Country Code and the year from 2010 to 2014.  Now using pandas, we will use “pd.read\_csv” to read the .csv file format file.   
Refer the screenshot below:



Let us move ahead and perform data analysis in which we are going to find out the percentage change in the unemployed youth between 2010 to 2011. Then we will visualize the same using [Matplotlib](https://www.edureka.co/blog/python-matplotlib-tutorial/" \l "matplotlib" \t "_blank) library, which is a powerful library for visualization in Python. It can be used in Python scripts, shell, web application servers and other GUI toolkits. You can use read more here: [Matplotlib Tutorial.](https://www.edureka.co/blog/python-matplotlib-tutorial/" \t "_blank)

Now, let us implement the code in PyCharm:

import pandas as pd

import matplotlib.pyplot as plt

from matplotlib import style

style.use (‘fivethirtyeight’)

counntry=pd.read\_csv(“D:UsersAayushiDownloadsworld-bank-youth-umemploymentAPI\_

df=country.head(5)

df=df.set\_index([“Country Code”])

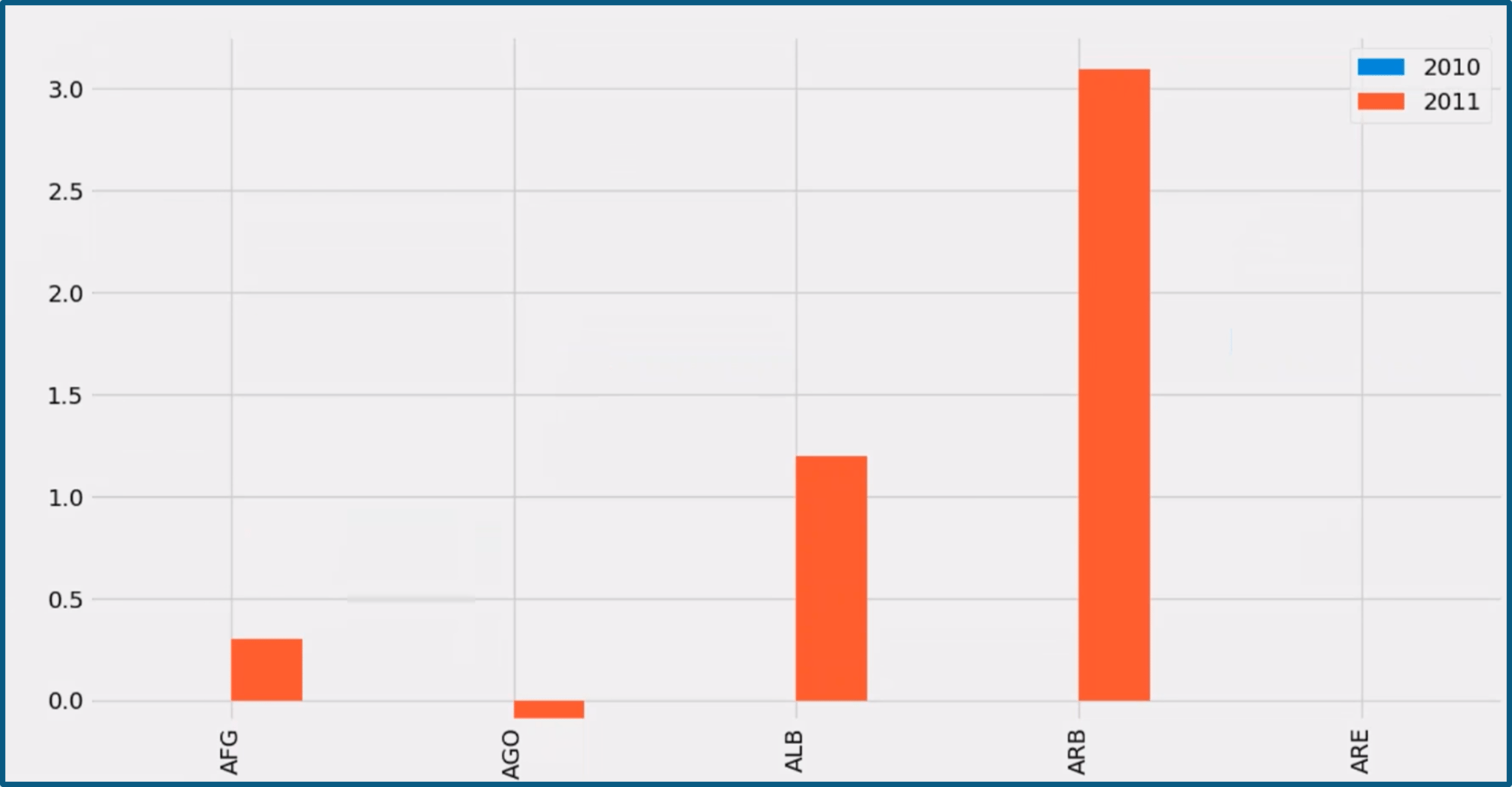
sd=sd.reindex(columns=[ ‘2010’,’2011’])

db=sd.diff(axis=1)

db.plot(kind=”bar”)

plt.show()

As you can see above, I have performed the analysis on the top 5 rows of the country dataframe. Next, I have defined a index value to be “Country Code” and then re-index the column to 2010 and 2011. Then, we have one more dataframe db, which prints the difference between the two columns or the percentage change of unemployed youth from 2010 to 2011. Finally, I have plotted a barplot using Matplotlib library in Python.



Now if you noticed in the above plot, in Afghanistan(AFG) between 2010 to 2011, there has been a rise in unemployed youth of approx. 0.25%. Then in Angola(AGO), there is a negative trend which means that the percentage of unemployed youth has been reduced. Similarly, you can perform analysis on different sets of data.