**ELMA EDISON**

**A00279906**

**Programming for Data Analytics – (AL\_KDATA\_9\_1)**

**11-12-2020**

**Data Analytics Python Libraries**

**1.Introduction**

Python being an open source universally useful language, is regularly utilized for aspects other than data science and data analysis. There are few libraries which offer users the required functionality while crunching data. Some of the major Python libraries used to work with data are NumPy for scientific computing, Pandas for data analysis and manipulation and Matplotlib for plotting and visualization of data. I had already installed Python on my laptop and for using the libraries, Integrated Development Environments (IDEs) are required. PyCharm is one of the most widely used IDEs for Python. I downloaded PyCharm, opened a new project and just tried out some basic python programs. The screenshot (Figure 1.1) given below is the PyCharm user interface and the opened file is the main python file. New projects can be created and in which different python files can be created. I created the project called 'firstProject' in which a python file called 'test' is created.

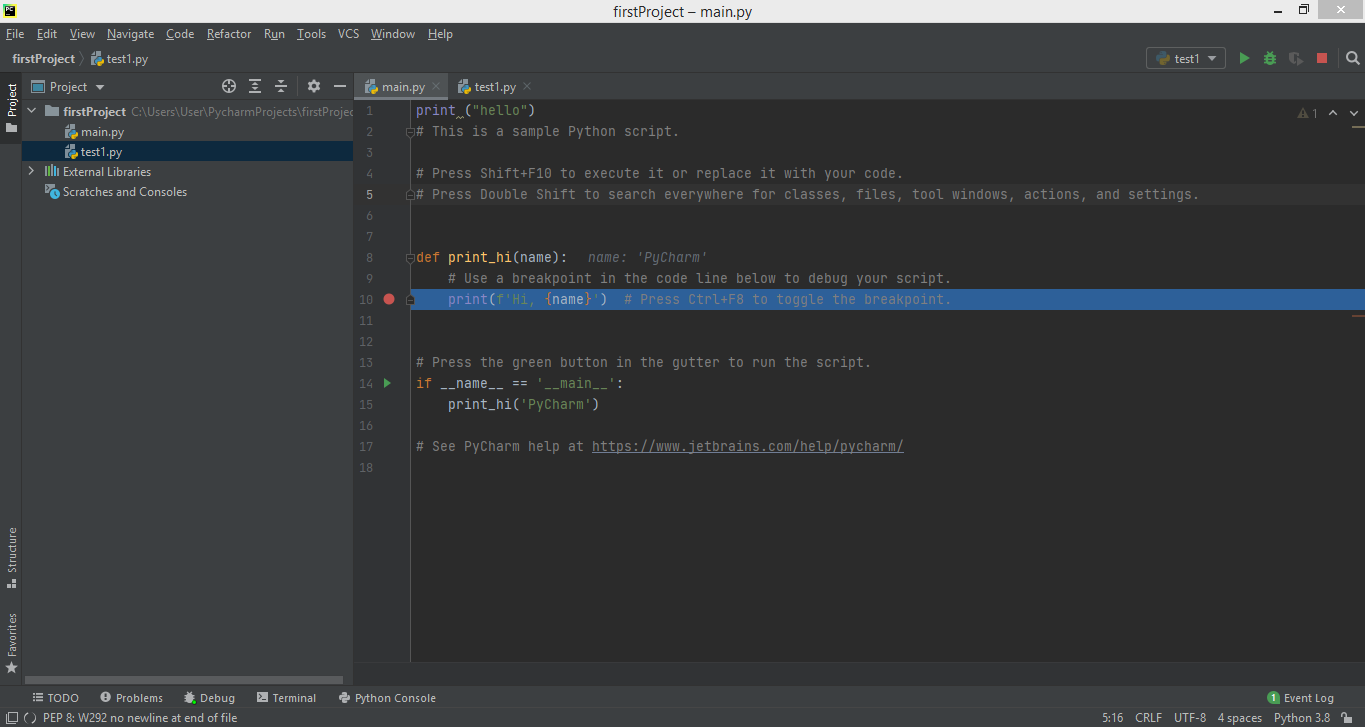


Figure 1.1

**2.Numpy**

NumPy represents Numerical Python and its most important feature is the n-dimensional array. NumPy has also simple linear algebra functions such as Fourier transforms, complex random number functionalities and methods for integration with other low-level languages such as C, C++ and Fortran. I installed the library NumPy into PyCharm and tried out a basic simple program shown in Figure 2.1.

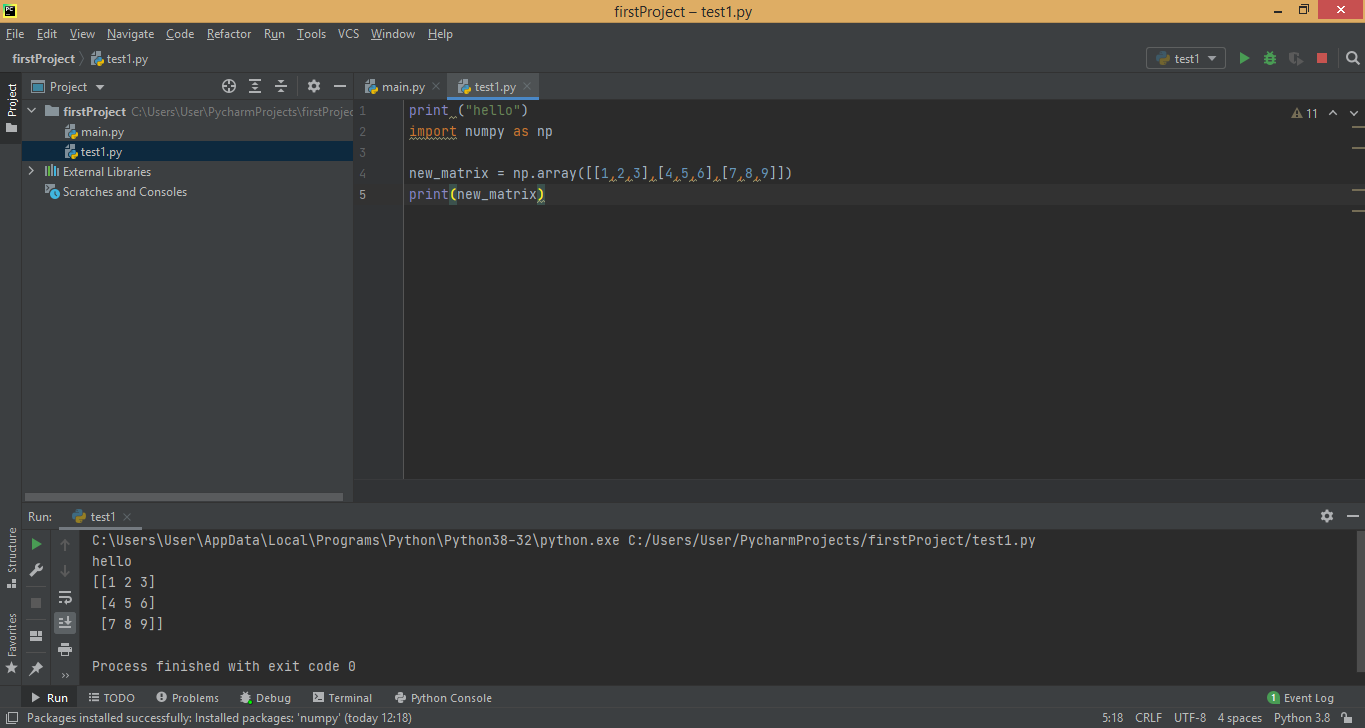


Figure 2.1

NumPy is faster as it uses fixed type. The binary values are going to be stored in Int32 (4 bytes) and so is faster to read less bytes of memory. To import the entire library, "import numpy as np" is used. Functions or methods can be referred by prefixing each call with np. as in the above example (Figure 2.1). Here I created an array matrix using the 'array' function and then used print() function to print the matrix. Numpy array comprises of elements of a single data type and is an incredible N-dimensional array object in the form of rows and columns. The type object can be accessed via the .dtype attribute. Sometimes an array's elements are initially unknown, but the size will be known. Therefore, NumPy provides many functions for building arrays with the initial content of the placeholder. These reduce the need to expand arrays, a costly process. Some functions are shown below in Figure 2.2.

Here in this figure, the function 'zeros' is used to create an array full of zeros and function 'ones' for an array full of ones. The function 'empty' is used to create an array who has random initial content and depends on memory state. The arange function is used to create number sequences and returns an array. Due to the finite floating point precision and as arange() is used for floating point arguments, it is usually not possible to predict the number of elements obtained. For this reason, the linspace() function, which receives the number of elements we like as an argument is typically easier to use.

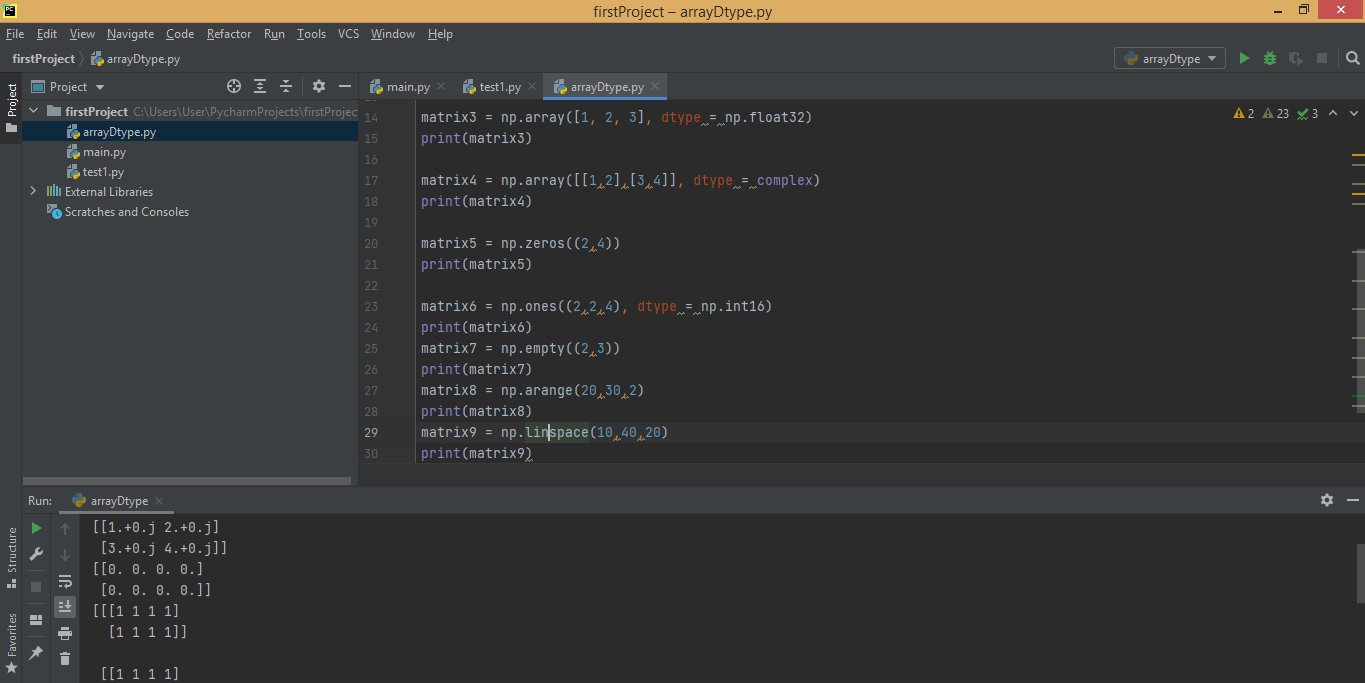


Figure 2.2

To inspect arrays, different NumPy operations can be used such as 'ndim' for knowing the dimension of the array, 'itemsize' to calculate the byte size of each element, 'dtype' to find the datatype of each elements stored in an array. An example is given below in Figure 2.3. Here I created an array with 2 rows and 3 columns and then applied different numpy operations to know the details of the array like its dimension, byte size, datatype etc.

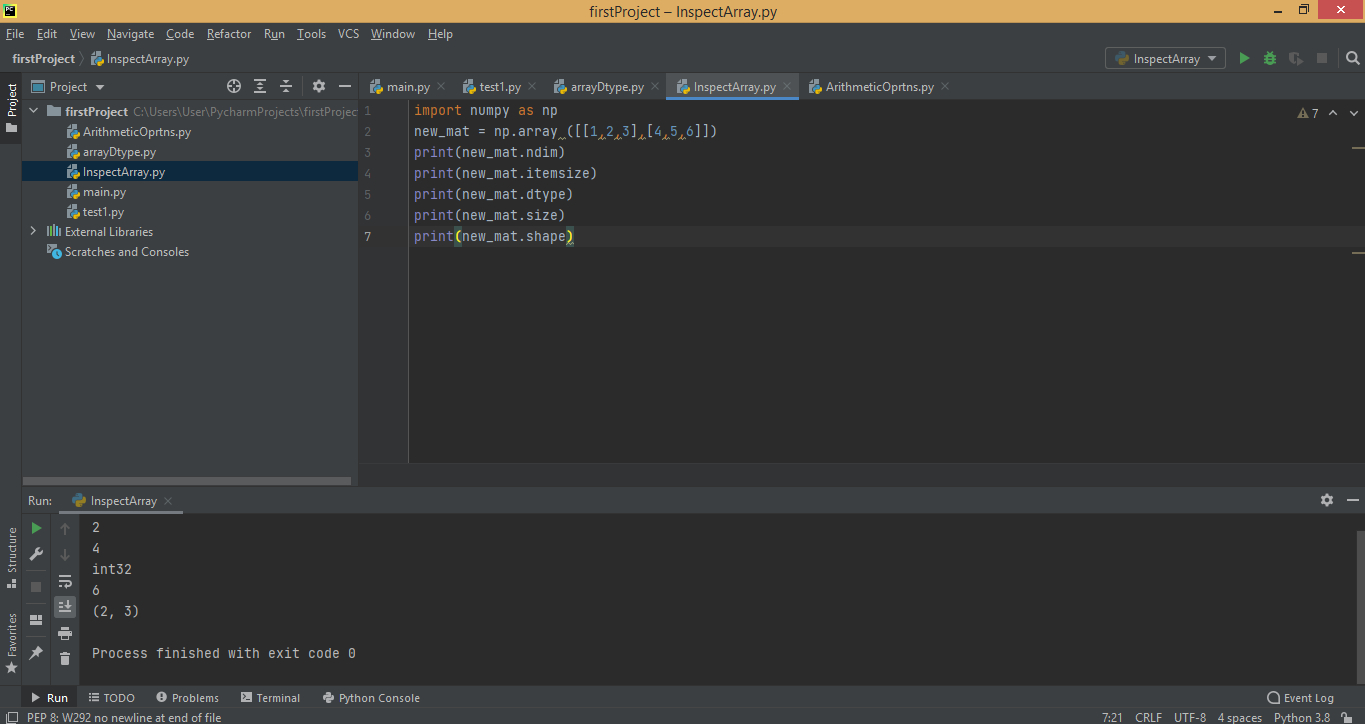


Figure 2.3

Arithmetic operations can be done using functions such as add(), subtract(), multiply(), divide(), exp(), sqrt(), sin() etc. An example of few operations are shown below in Figure 2.4. Here, I created two arrays that is first one with 'array' function and second one with 'arrange' function. Then added and subtracted the two arrays using add() and subtract() functions respectively. After that I took the square root and sine of the first array using sqrt() and sin() functions respectively.

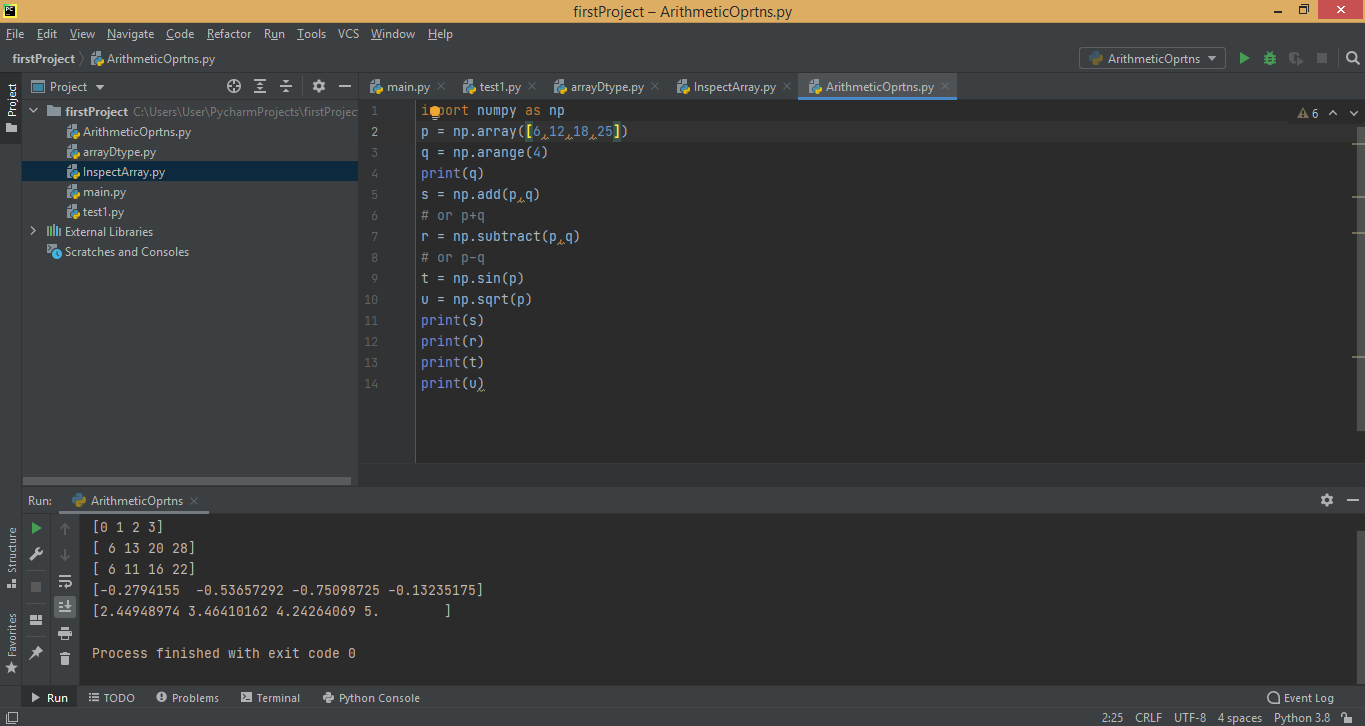


Figure 2.4

There are some aggregate functions in numpy such as sum(), min(), max(), mean(), median() etc to calculate the total sum of elements, minimum value of the elements in the array, maximum value of the elements in the array, mean of the array, median of the array respectively. An example using some aggregate functions is given below in Figure 2.5. In this example, I have compared two arrays as a whole by using '==' condition to check if two arrays are equal and then used all() function to check if its True or False. I also have used greater() function to check elementwise if array elements of one array is greater than the other array.

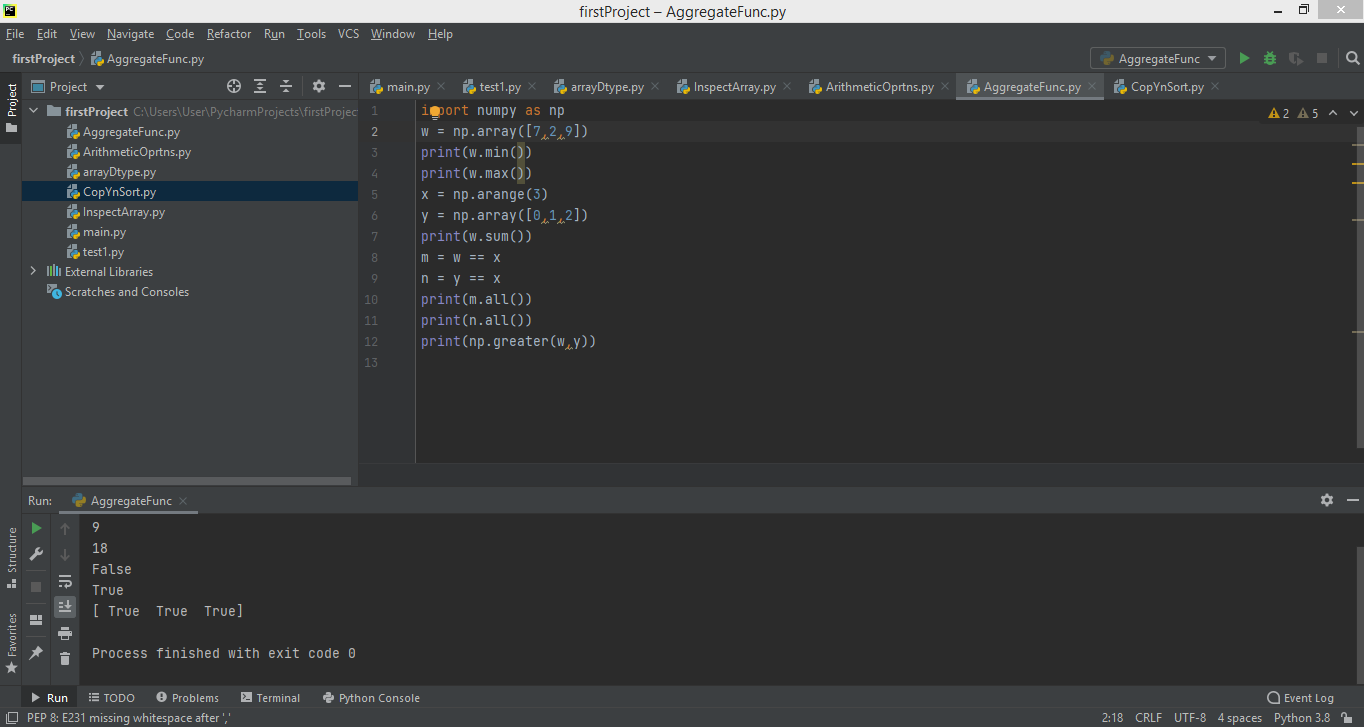


Figure 2.5

Sorting of arrays is another important function done using numpy. The sort() function is used to sort arrays and an example with this function is shown below in Figure 2.6. In this example, I created an array with 2 rows and 2 columns and I used sort function to sort the particular array. I also have used copy() function to duplicate a copy of the same array.

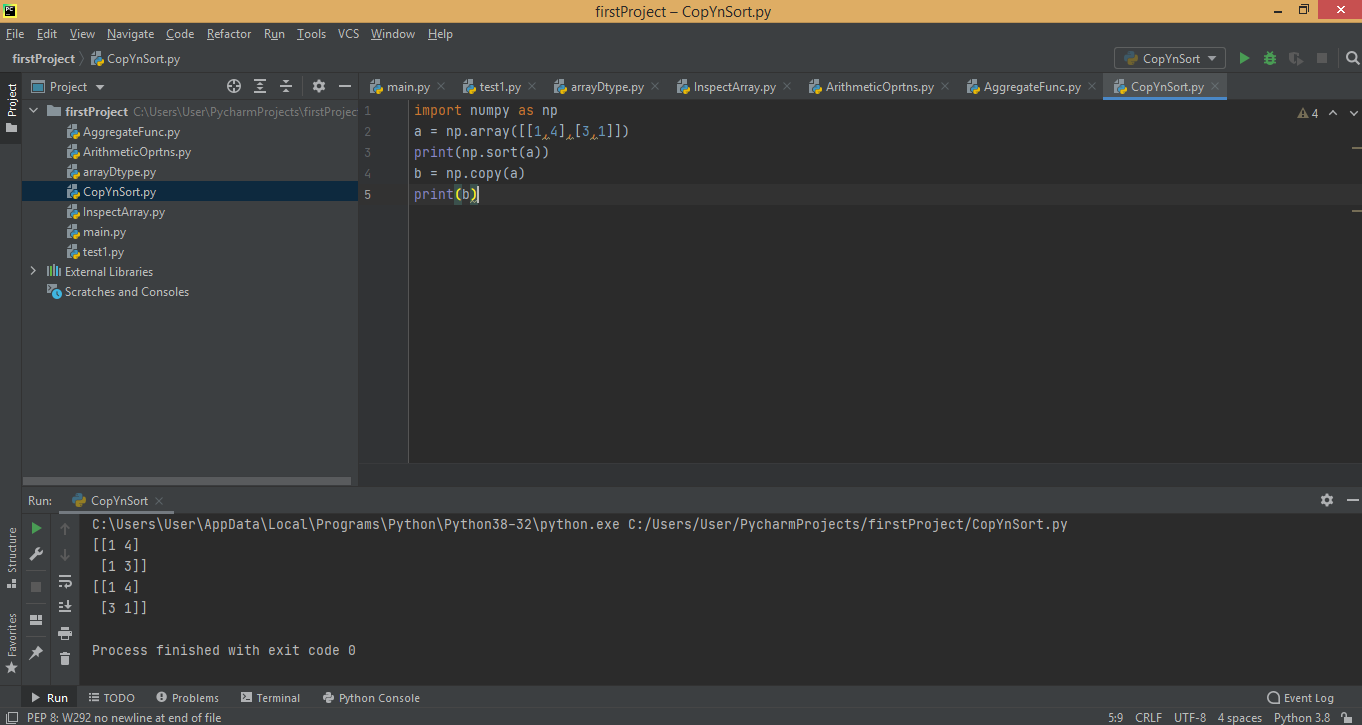


Figure 2.6

**3.MatPlotlib**

MatPlotlib is one of the most widely used library for plotting and visualising data using numerous variety of graphs. It is known that human brain can process information quickly when it is in pictorial or graphical form and so data visualisation is one of the key steps in data analytics. It helps for quicker interpretation of data and adjustment of different variables to see their effect. And to visualise data, matplotlib python package is used. There are different types of plots in MatPlotlib such as Bar graphs, Histograms, Scatter plot, Pie plot, Hexagonal Bin plot and Area plot.

I went through few tutorials and tried out some examples. A simple example is given below in Figure 3.1, where I imported the pyplot from matplotlib as plt and used the pyplot by giving x and y axes to plot the graph in the canvas and plt.show() to show what graph is obtained. The graph is shown in Figure 3.2.

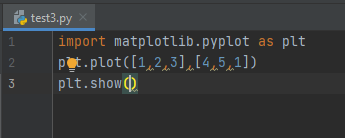


Figure 3.1

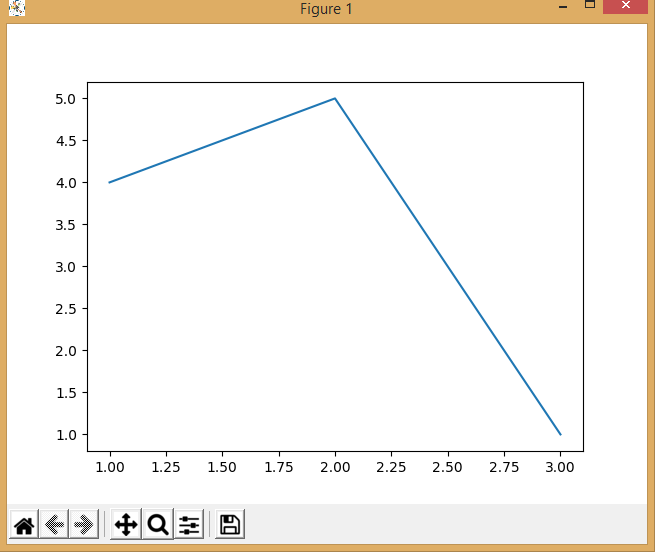


Figure 3.2

In the above example, the graph actually doesn't make sense as it doesn't have a title, x and y labels. So I tried out another example to plot a graph with title, x and y labels which is shown below in Figure 3.3 and the output graph is shown in Figure 3.4.

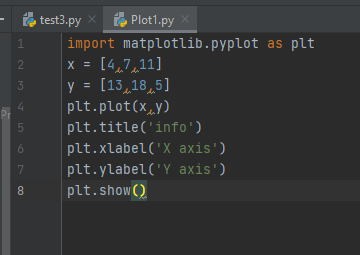


Figure 3.3

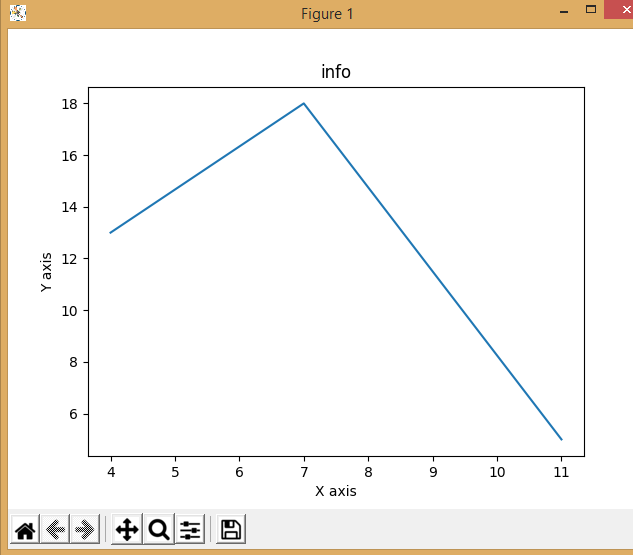


Figure 3.4

To visualise the graph in a better way or to add style to the graph that is to change the width of the line or add grid lines or change the colour of the line, first we have to import 'style' from matplotlib. An example is given in the Figure 3.5 where I imported style and then called the function where I have used ggplot for plotting. Here I have used 4 variables x, y, x2 and y2. Then plt.plot() is used to plot the graph where apart from x and y axes, I have given the color, label and linewidth for the curve. Title, x and y labels are also given. I have also included the legend() function to add legend to the graph and so line one and line two is shown in the top right corner. In addition to that, I have also included grid lines which are black (color is denoted by k).

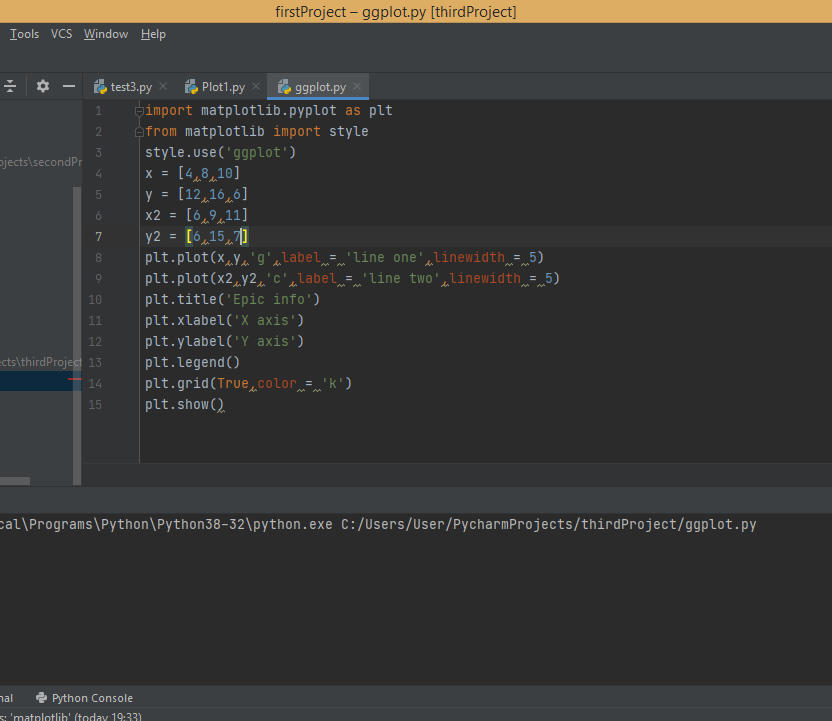


Figure 3.5

The output obtained is easily visualisable using plt.show() function and is shown in Figure 3.6. Here line one is represented by the color 'green' and line two is represented by the color 'blue'. The title is given as 'Epic info' and x-y labels are also given.



Figure 3.6

The examples explained above are of simple basic linear graphs. Now to compare things between groups, we need a bar graph which are very well suited to measure changes over time. Bar graph has categorical variables. Given below is an example of bar graph in Figure 3.7.

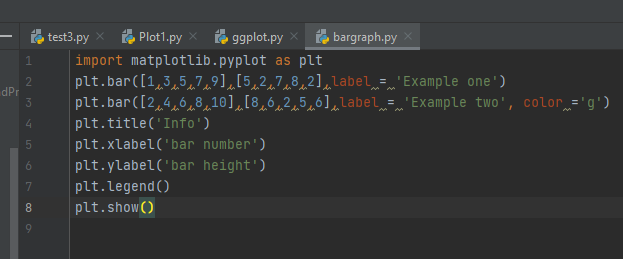


Figure 3.7

Here, firstly I imported pyplot from matplotlib as plt. And then using plt.bar() function, gave the x-y values, label and color for the bar. I used legend() function to add legend to the bar graph which is shown on the top-right corner. Then as always, it is a must to mention the title, x and y labels and show the bar graph using plt.show() function. The Figure 3.8 is the ouput bar graph obtained with blue and green colours.

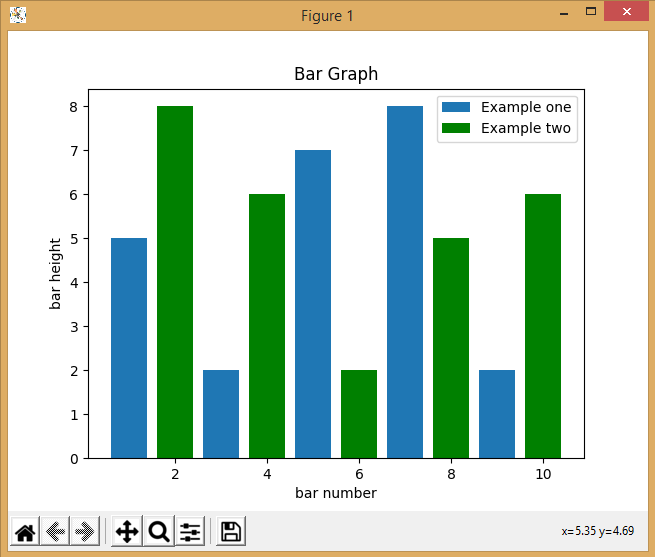


Figure 3.8

If we have a categorical value, then we can use bar graph. But if we have quantitative variables that is for example age group, then we should use histogram. I have tried out an example shown in Figure 3.9 using histogram for population ages. Here I have created two lists that is one is population\_ages and another is bins. Now since it is histogram, we use plt.hist() to plot the histogram where variables are passed, the histtype is chosen as 'bar' and rwidth is set to 0.8. Then title, x and y labels are also set and plt.show() is used to show the histogram obtained (shown below in Figure 3.10.

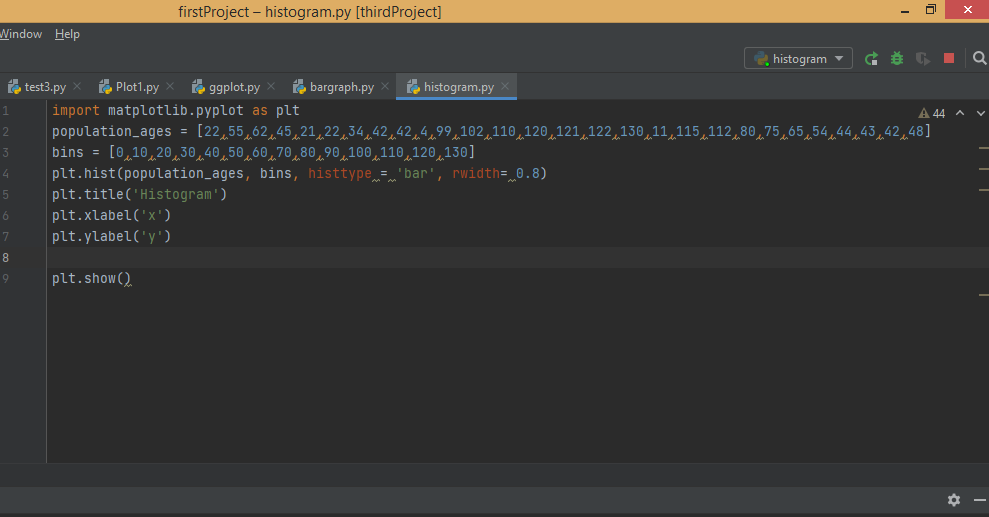


Figure 3.9

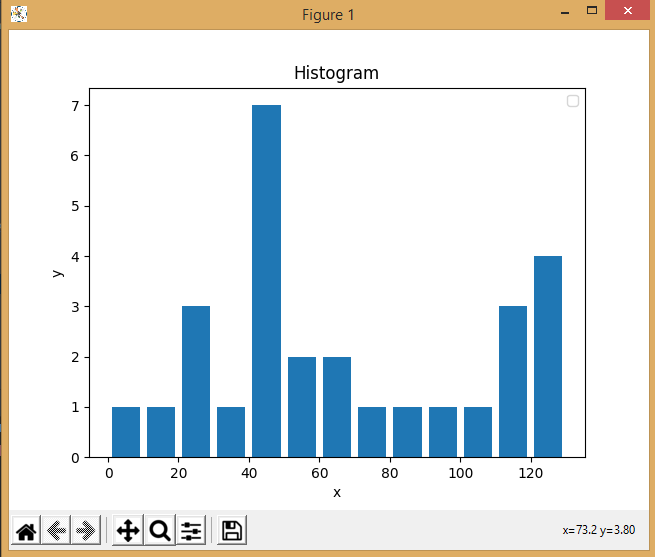


Figure 3.10

Scatterplots are used to find out the correlation between 2 or 3 variables(if it is a 3-D) that is to find out how much two variables are related to each other. An example is given below in Figure 3.11 where two variables are plotted using Scatterplot. Here the difference is that two variables are plotted using plt.scatter with label set as 'skitscat' and color is set as black (denoted by k).

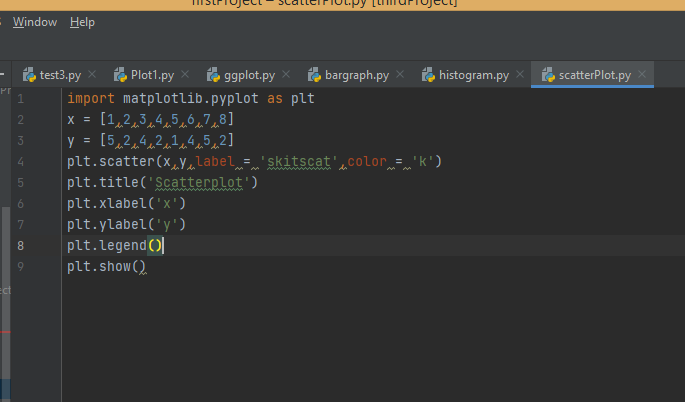


Figure 3.11

The output showing the scatter plot is given in Figure 3.12, where the title, x and y labels with skitscat label for the plot is also shown.

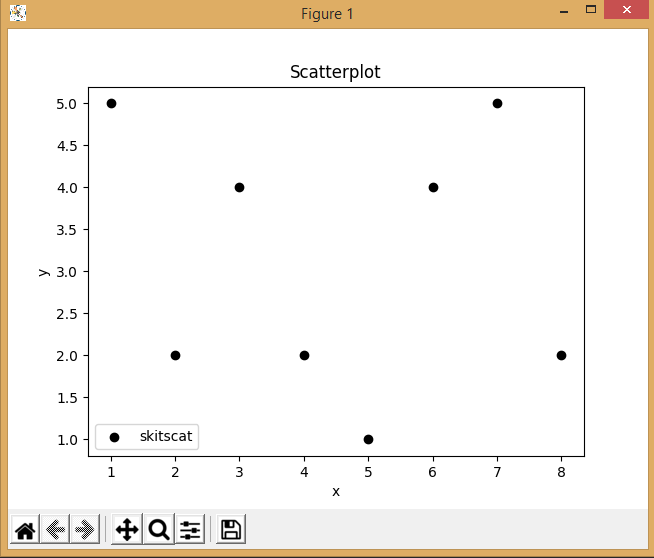


Figure 3.12

Area plot or Stack plot are a lot similar to line graphs. It is used when there is need to track changes over time for one or more related groups that make up a whole category. An example of how you spend your day by sleeping, eating, working and playing is shown below in Figure 3.13. Here five lists are created and for plotting, each of them are given empty labels so that they are depicted in the Area graph. Color and linewidth are also given to make the graph look much better and colorful. The output of the Stack Plot is obtained and shown in Figure 3.14.

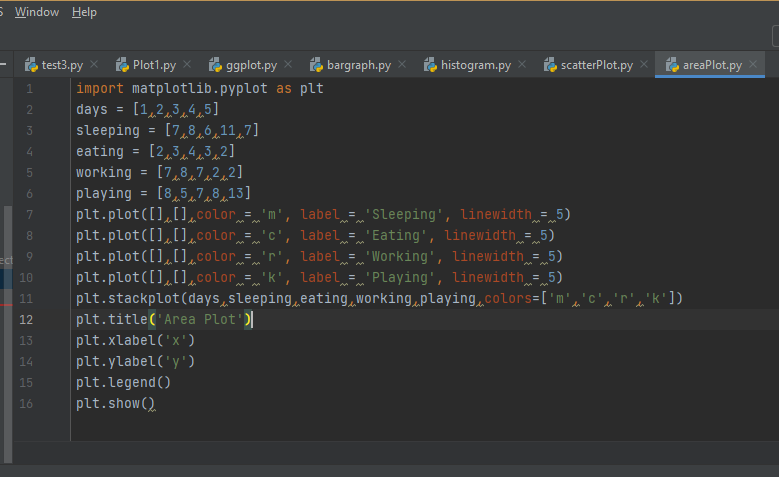


Figure 3.13

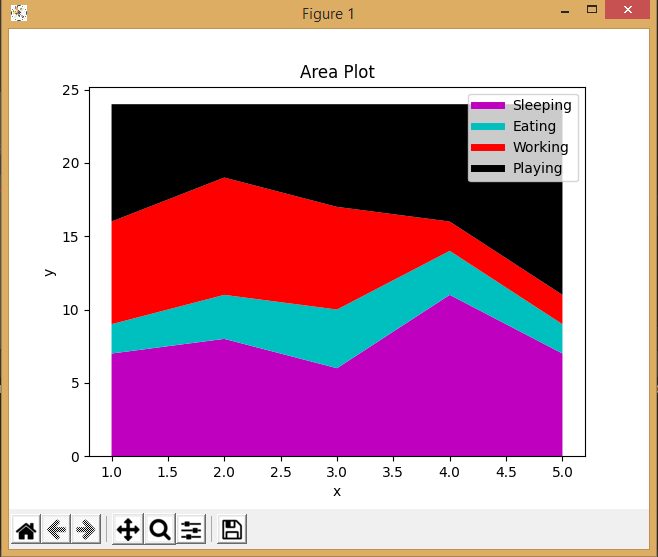


Figure 3.14

Pie charts are for a point in time and are used to show paths to the whole and often a percentage share. They are similar to Stack plots. An example of the pie chart is given below in Figure 3.15. To plot a pie chart, plt.pie() is used where slices, labels, colors of each slices, startangle, shadow, explode, autopct are all given. Slices are sizes relevant for each part and then labels, colors. Startangle is where the pie chart should start and so I gave 90 which will be a vertical line. I have used shadow and to pull a slice a bit, I used the explode parameter. Since I have given 0.1 to second slice, eating is pulled out a bit. To overlay the percentage on graph, autopct() is used. The output of the Pie chart is obtained and shown in Figure 3.16.

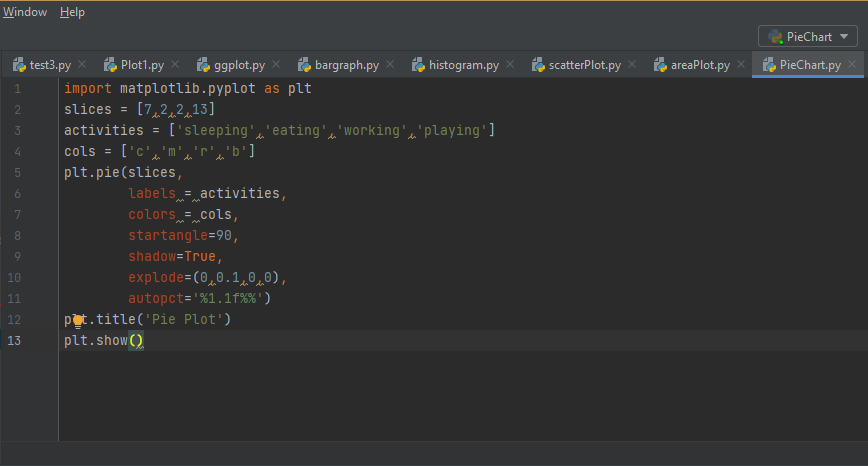


Figure 3.15

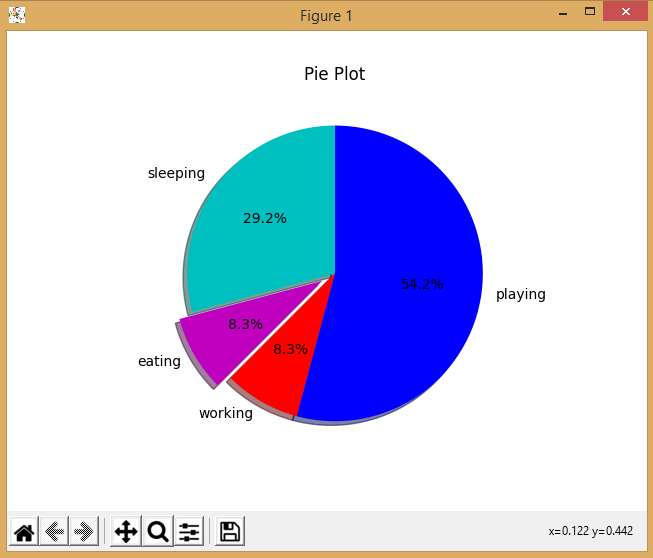


Figure 3.16

I have tried out another example to plot multiple graphs. In the Figure 3.17 given below, I have used the numpy package to create arrays i.e., np.arange() to create 2 arrays and then to obtain multiple plots, subplot() function is used. In that 2 is denoted for two plots, 1 is denoted for horizontal plot as I want only 1 horizontal plot. And 2 vertical plots are there and so the first graph being the first in position, it is denoted by 1. Similarly 212 for the second plot. The output of multiple plot is obtained and shown in Figure 3.18.

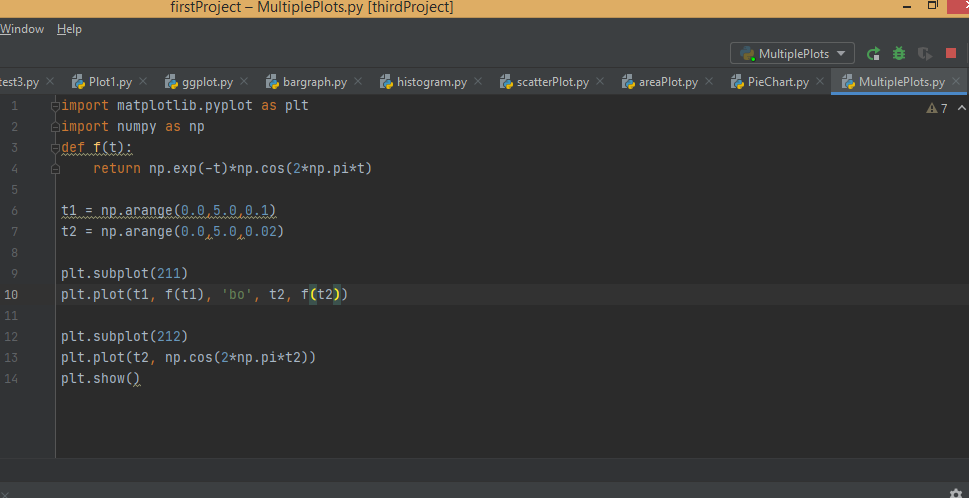


Figure 3.17

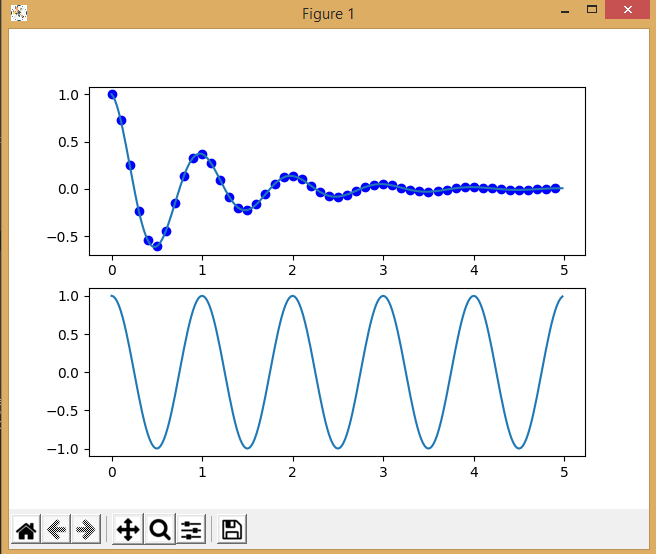


Figure 3.18

**4.Pandas**

Pandas is used for data manipulation, analysis and cleaning. It is well suited for different kind of datas such as tabular data, arbitrary matrix data, ordered and unordered timeseries data, unlabelled data, observational and statistical data. It is built on NumPy and the Pandas data structures are DataFrame and Series.

A DataFrame is a two-dimensional, theoretically heterogeneous tabular data that is mutable in size. It also has access labels, which are rows and columns on which the arithmetic operations coincide. It can be thought as a dictionary like container for series objects. The series is a one-dimensional array that can carry any form of data. The access labels are generally referred to as index here. The Pandas series is nothing more than an Excel sheet column. An example showing series and dataframe is given below in Figure 4.1. To create a Series, values and index needs to be given. After that I created a date time index from '20201202' to 10 periods and then created a DataFrame using random values with the date time index. I also created one more DataFrame by passing a dictionary.

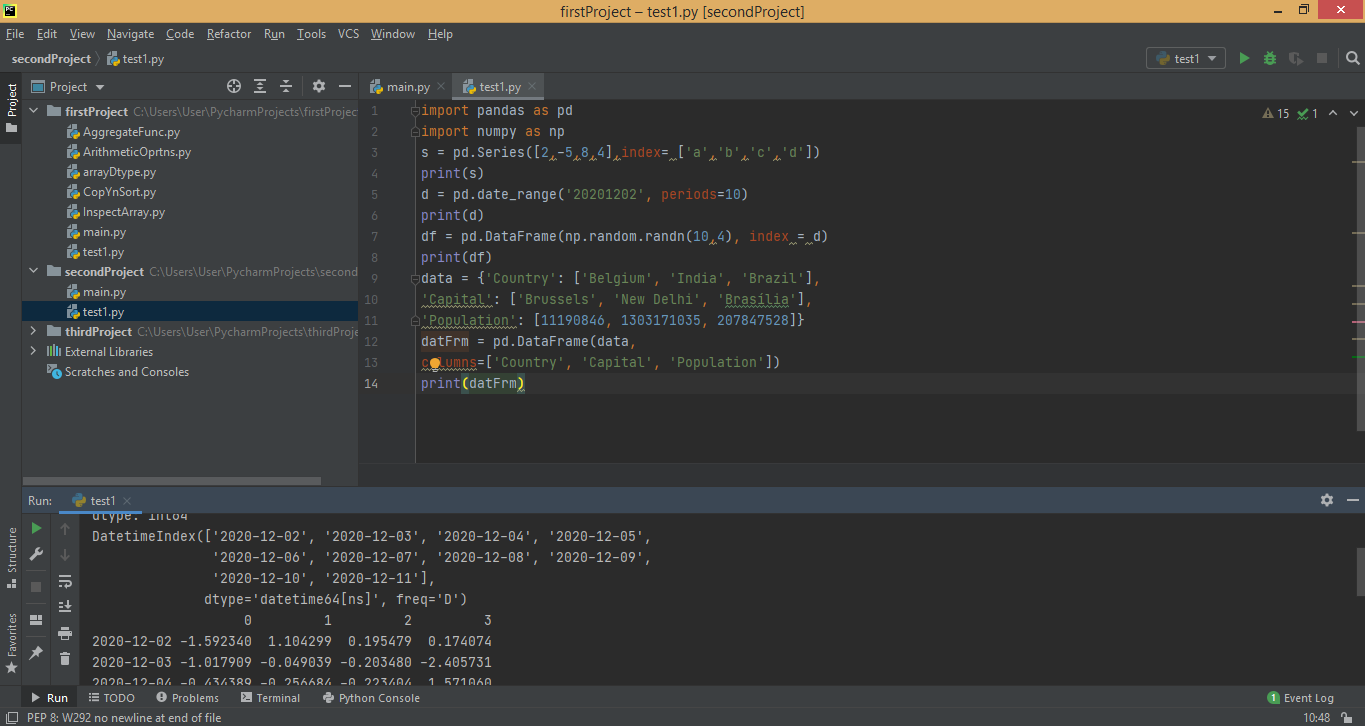


Figure 4.1

To view the data that is to retrieve Series or DataFrame information, various functions can be used. The function head() can be used to view the first five values/rows inside a DataFrame and tail() function is used to view the last five rows. To get all the values of index that is to get a description of index, 'index' function is used. The function 'columns' is used to get all the columns in a DataFrame. To view the summary statistics which includes count, mean, minimum, maximum, standard deviation etc, 'describe' function is used. The figure (Figure 4.2) given below is the screenshot of an example showing how to retrieve series or dataframe informations. Here I created a series and a DataFrame with random values and then tried out few functions.

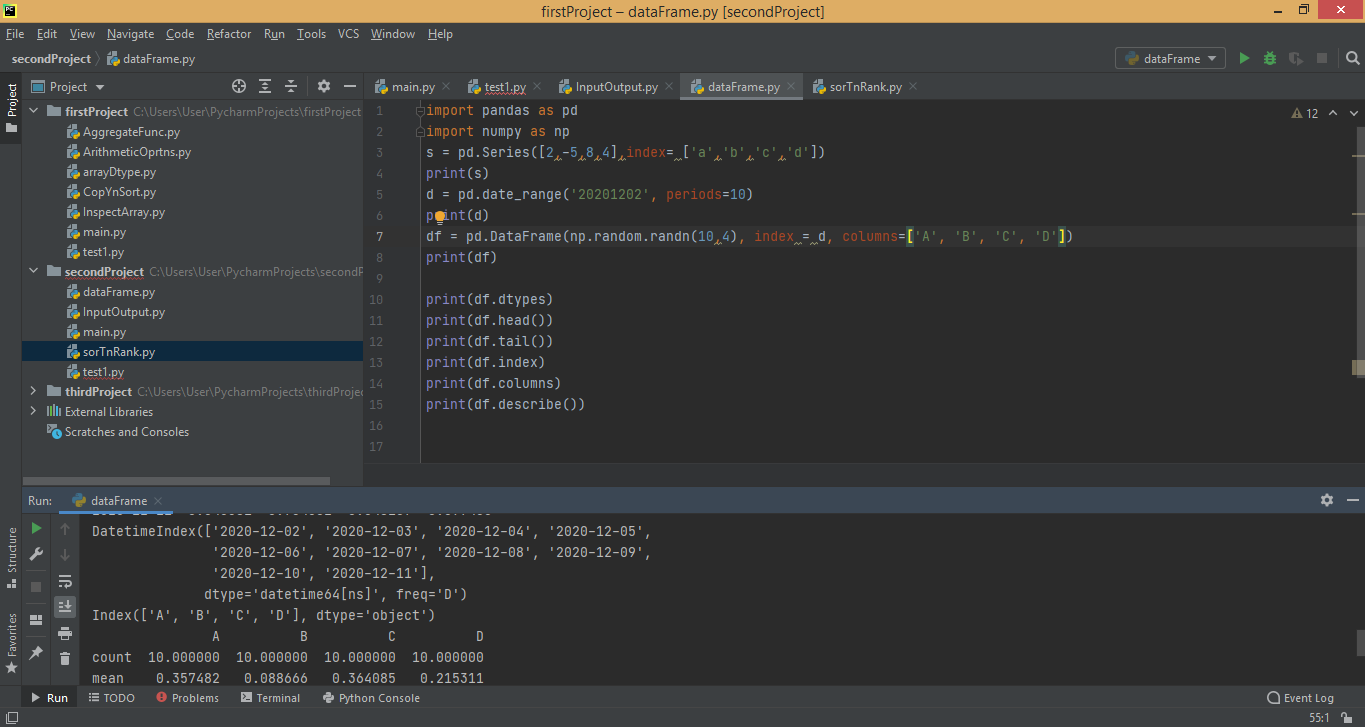


Figure 4.2

Sorting can be done using Pandas that is the data can be sorted using axis. The function sort\_index() will sort the the data by labels along an axis and the function sort\_values() will sort by the values along an axis. The ascending parameter is used to sort in ascending order. In the Figure 4.3 given below, a DataFrame with random values is taken and is sorted by labels and values. Also another DataFrame is created by passing a dictionary. NaN (not a number ) is a numeric datatype to represent undefined or unpresentable values and in the example below, it is given to the last row. The function rank() is used to assign ranks to entries. There are various parameters for rank() function such as axis, method, na\_option, ascending, pct. The default rank() function is used without using any parameter. Records having same values can be ranked using the highest rank by setting method = 'max'. For records with NaN values, it can be placed at the bottom by choosing na\_option = 'bottom'. To express the ranking in percentile rank, pct can be set as True.

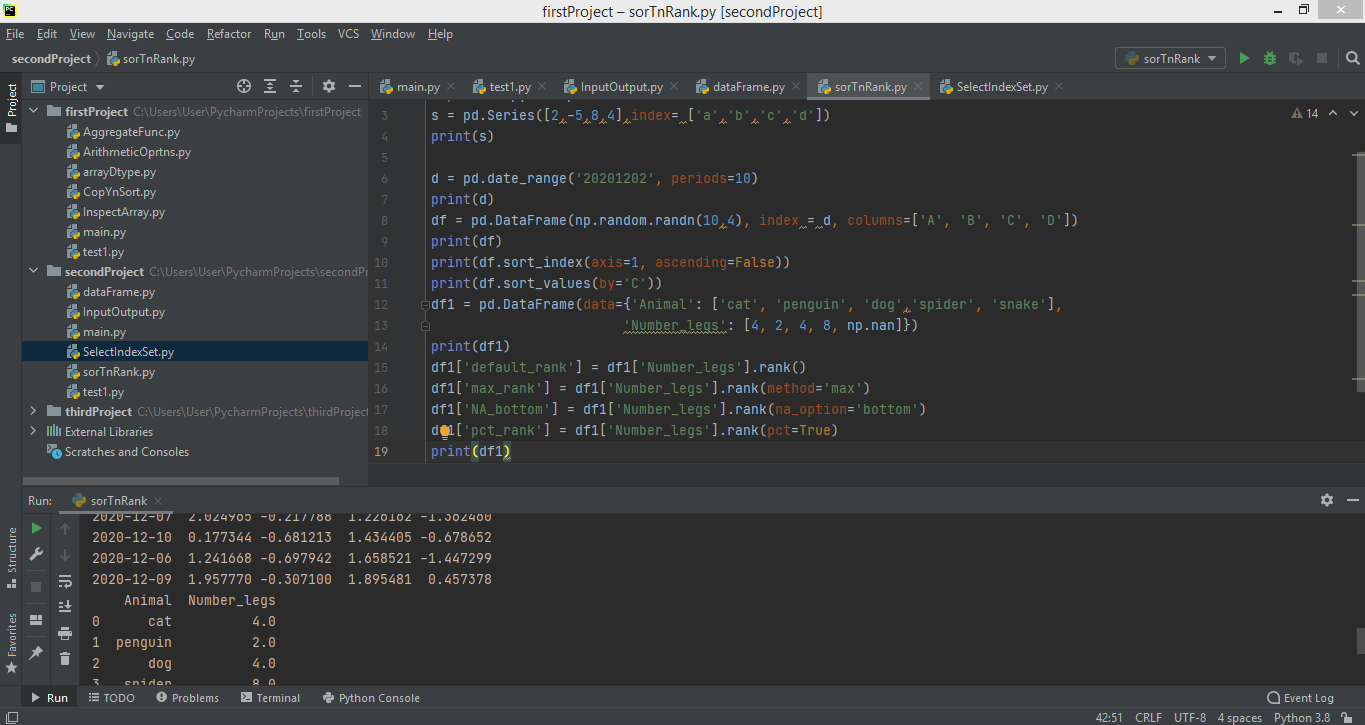


Figure 4.3

Selection can be done using Pandas where it is possible to select particular value, column, etc. An example is given below in the Figure 4.4 where selection of a particular column from a dataframe and a particular value from a series are done. Slicing can be done by indicating the start and end values. Datas can be selected using labels where labels are required to be passed in loc(). For selecting values by position, iloc() can be used. And for data on a multi-axis by label, column names can be passed. Label slicing can be done by giving the start and end points as well as passing labels. To get scalar values fastly, at() can be used instead of loc(). For obtaining a value from the position of a dataframe, iloc() is used. Boolean indexing is also done by giving conditions like less than or greater than etc to obtain the required values. I have tried using all these selection functions in the example given below in Figure 4.4.

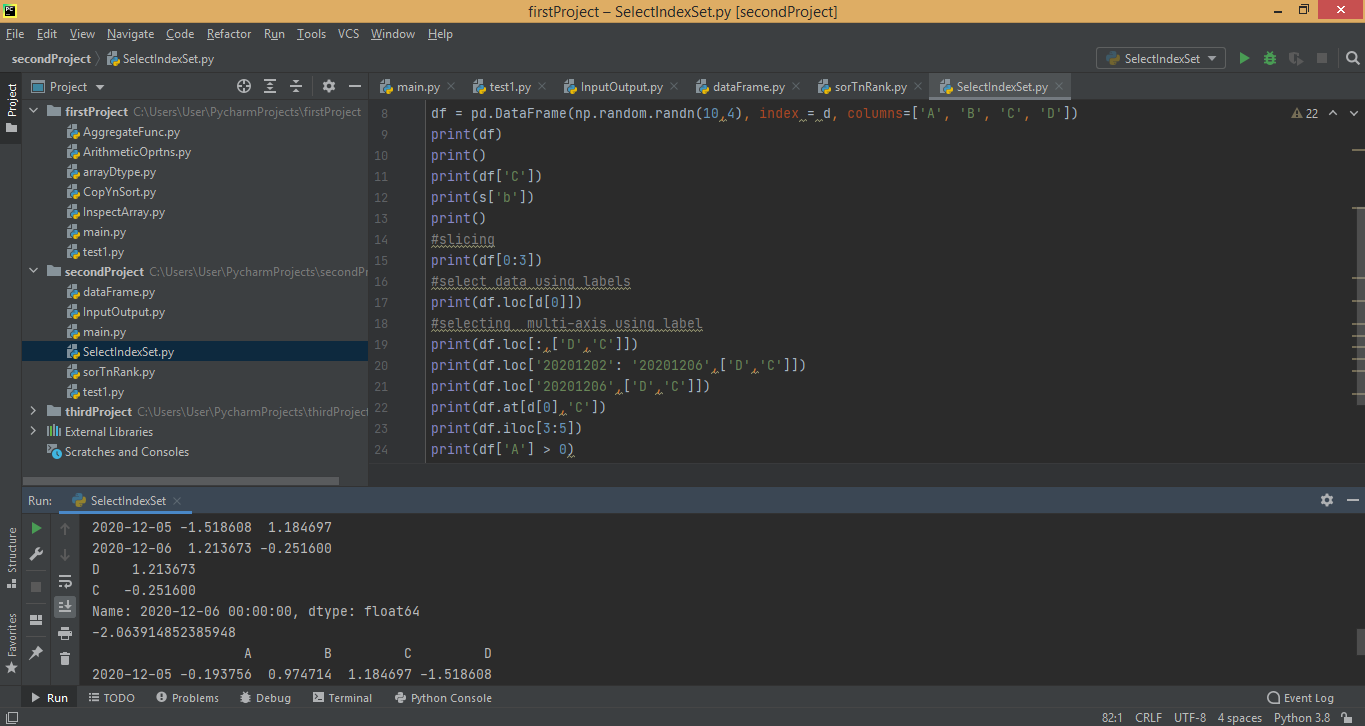


Figure 4.4

Data Allignment can be done either by introducing NaN values in the indices that do not overlap or with the help of fill methods. An example is given below in Figure 4.5 where I created two series, one having four indices and other with 3 indices. By adding up the two series a and b, NaN value gets introduced in the index which was missing. Another way to do internal data alignment is by using arithmetic operations with Fill methods where a is added to b and the missing index gets filled with the value we give. The arithmetic operations used are add for addition, sub for subtraction, mul for multiplication and div for division.

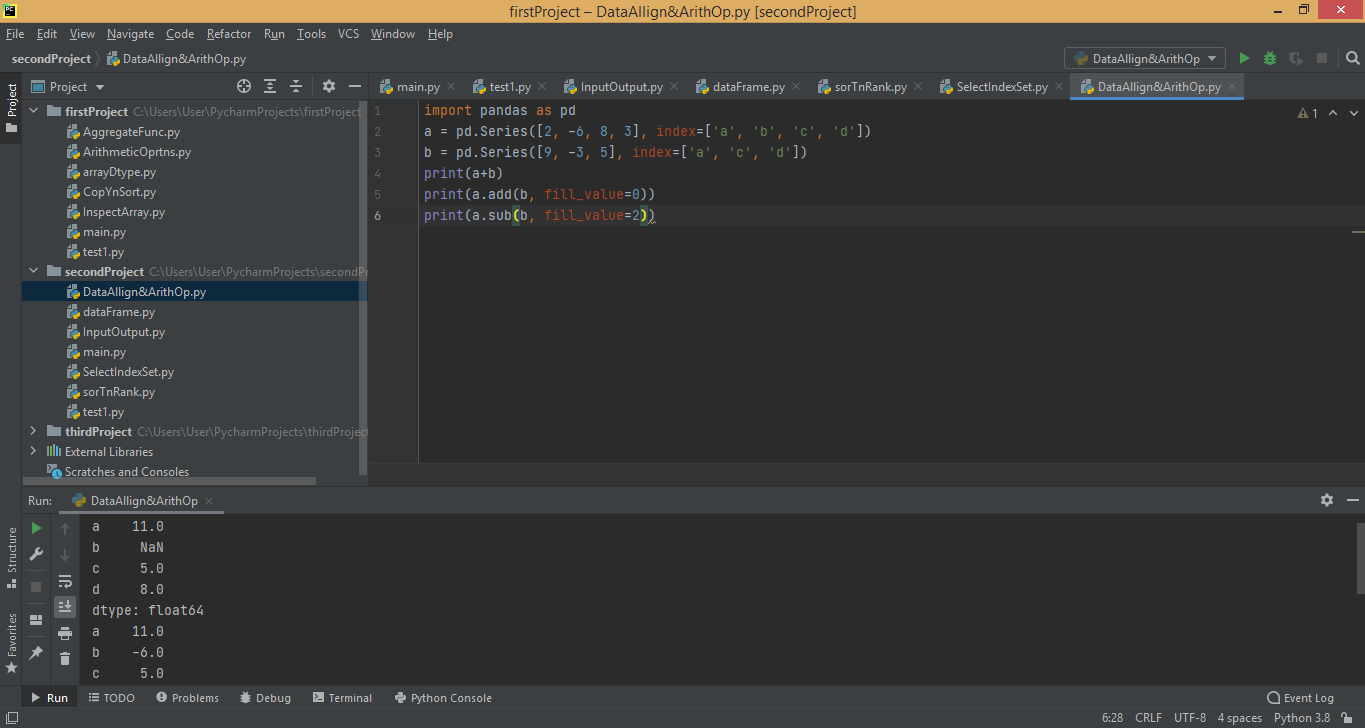


Figure 4.5

To drop certain values from a series or to drop a column from a dataframe, drop() function is used. Here in the example given below in Figure 4.6, I have created a series and a dataframe from which some values and columns are dropped respectively.

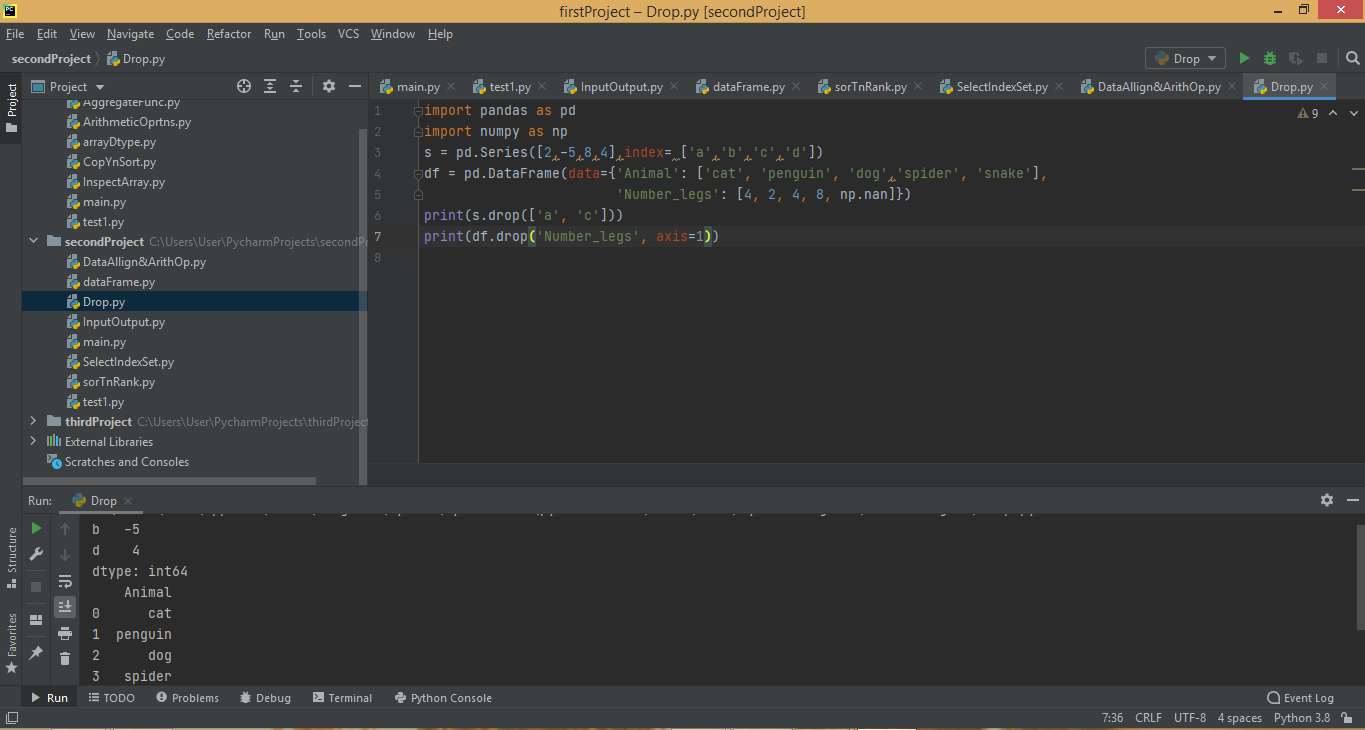


Figure 4.6

Input or Output functions are used to read input data and to write the output data. There can be a variety of file types like csv, excel, html, sql etc. There are different functions to read each one of them like read\_csv(), read\_excel(), read\_html(), read\_sql(), read\_sql\_table(), read\_sql\_query() and functions to write are to\_csv(), to\_excel(), to\_sql() etc. In the example given below in Figure 4.7, I used read\_csv() function where I gave the file path to read a 'zoo.csv' file. Then I used set\_index() function to set the index as the 'type' column and kept inplace as True. Then by using the to\_csv() function, I wrote it to a new csv file named 'newcsv'. Output is shown in the screenshot where all the columns are displayed.

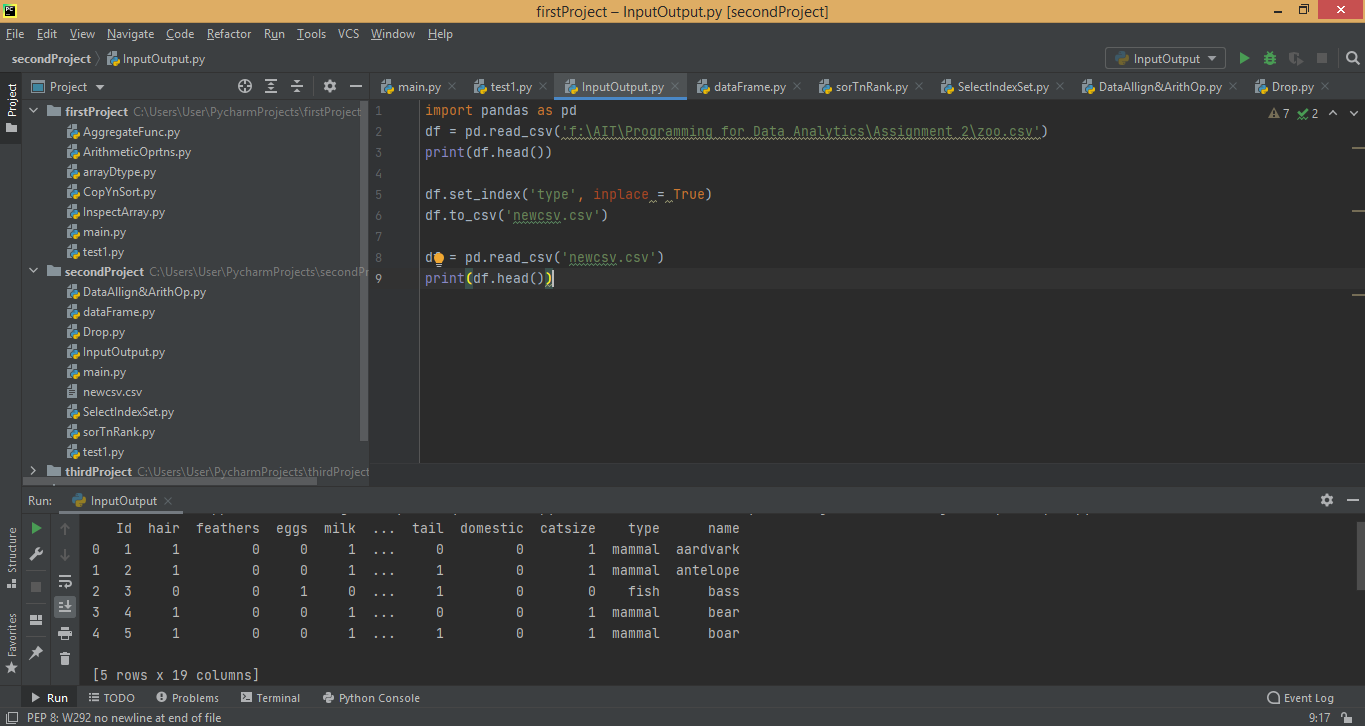


Figure 4.7

**5.Other Similar Libraries**

While learning the Numpy, MatPlotlib and Pandas, I came across few libraries that I think are similar and advanced packages.

**5.1.SciPy**

SciPy represents Scientific Python and is built on the NumPy. SciPy is one of the most valuable libraries for a number of significant-level science and engineering modules, such as the Linear Algebra, discrete Fourier Transform, Optimization and Sparse matrices. An example is shown below in Figure 5.1.1 where I used the same program I did for numpy. All basic arithmetic operations and scientific mathematical operations can be performed using scipy.

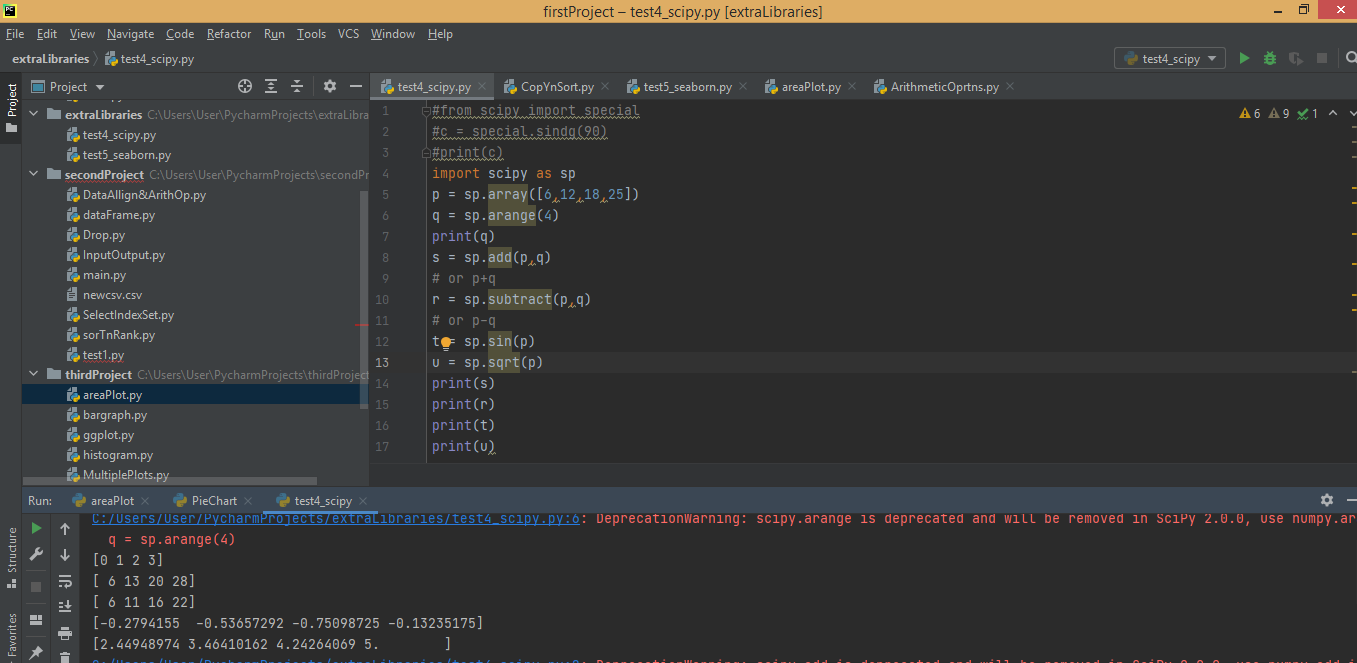


Figure 5.1.1

Another example of copying and sorting is done in the Figure 5.1.2 using scipy same as that of numpy. An example using aggregate functions is also done in the Figure 5.1.3.

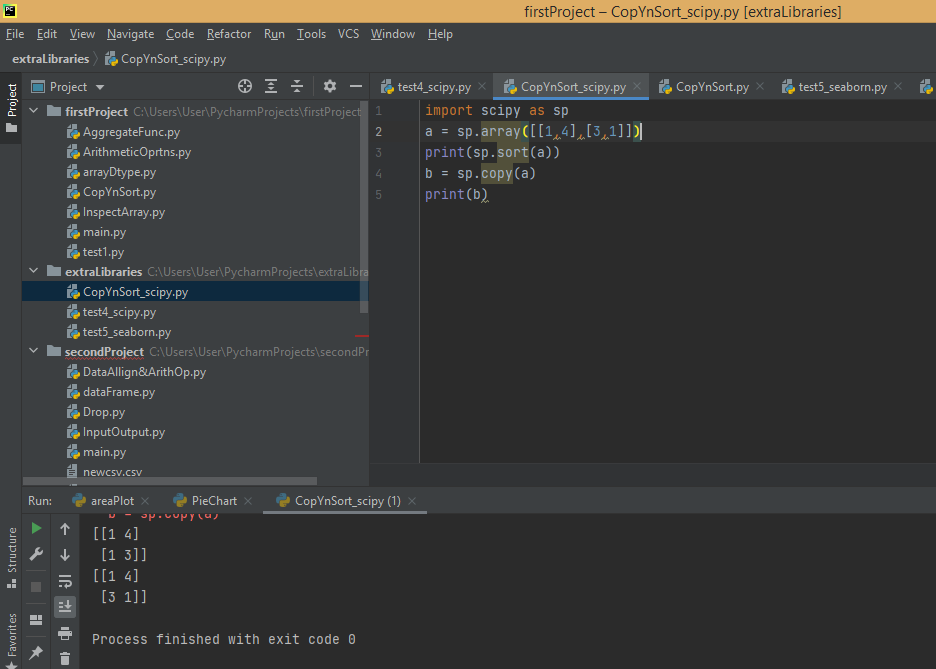


Figure 5.1.2

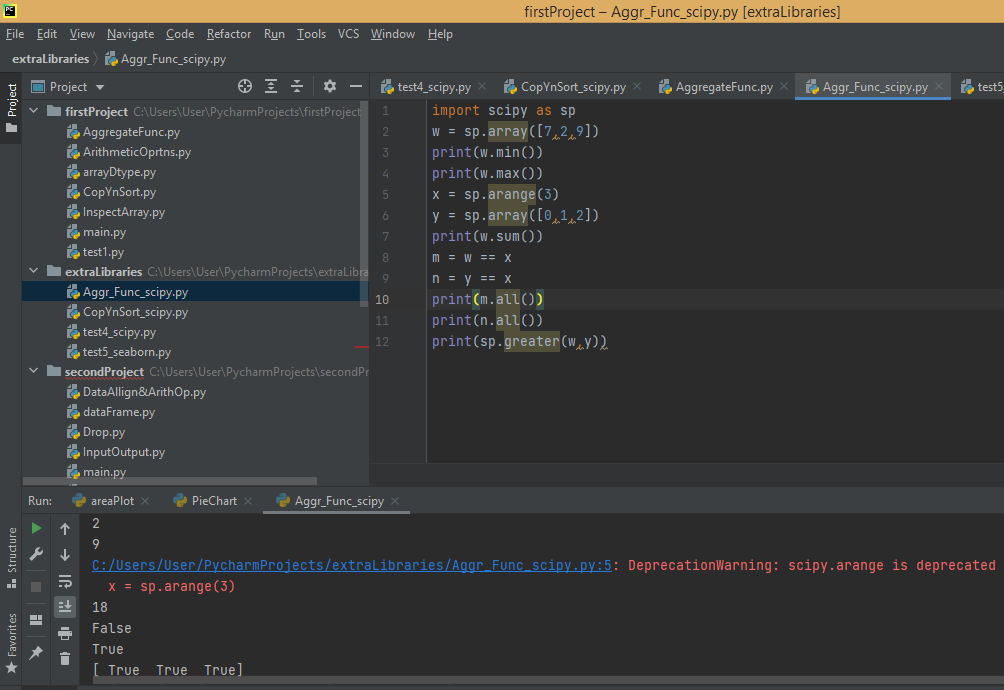


Figure 5.1.3

**5.2.Seaborn**

Seaborn is used for visualization of statistical data. It is a library for rendering attractive and informative statistical graphics in Python. It depends on Matplotlib and aims to make visualization a core part of data discovery and comprehension.

An example using seaborn is shown in the Figure 5.2.1. Firstly, I imported the seaborn as sns and imported pandas and matplotlib as well where I used pandas to read the csv file. Now to load the dataset in seaborn, sns.load\_dataset() is used. And then I used relplot() to show the relationship between x and y variables where I gave total\_bill as x and tip as y. This can be shown using hue, style and size parameters for different subsets of data. Finally to view the plot, show() function is used from the matplotlib package.

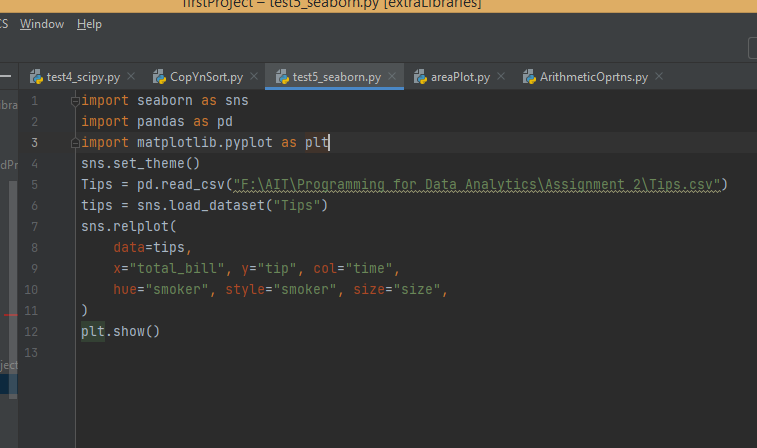


Figure 5.2.1



Figure 5.2.2

Another example that I tried is of heatmap which is shown in Figure 5.2.3. Here I used random.rand() function to create an array with random values and then used heatmap to plot it.

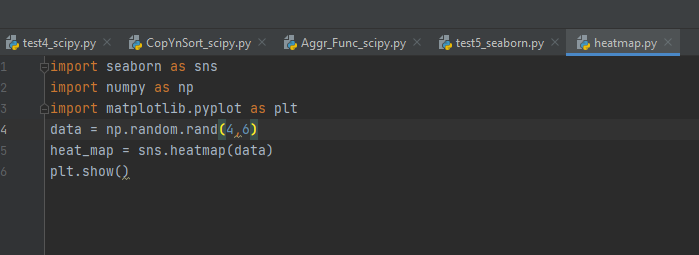


Figure 5.2.3

The output is shown in Figure 5.2.4.

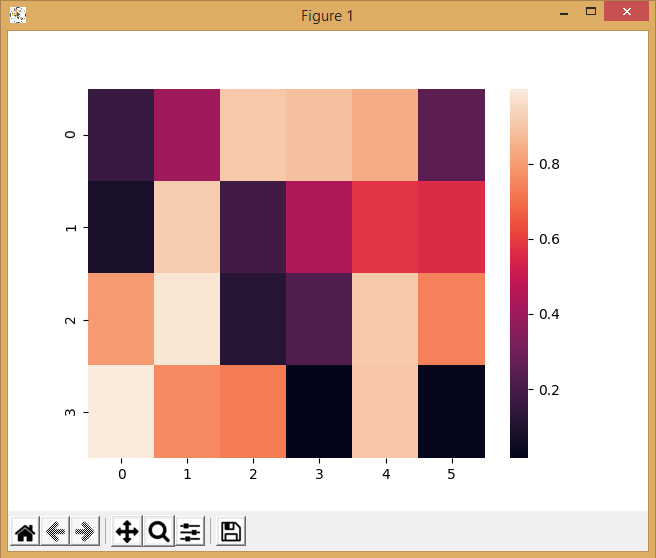


Figure 5.2.4

**6.Conclusion**

In this report, I have learned about few python libraries by watching some tutorials and done my own examples. Pycharm is a very user friendly editor and helped me understand the python libraries in detail. These python libraries play a key role in data analytics processes.

**References**

1. NumPy - Arithmetic Operations - Tutorialspoint [WWW Document], n.d. URL <https://www.tutorialspoint.com/numpy/numpy_arithmetic_operations.htm> (accessed 12.11.20).
2. Python Numpy Tutorial | Learn Numpy Arrays With Examples, 2017. . Edureka. URL <https://www.edureka.co/blog/python-numpy-tutorial/> (accessed 12.11.20).