Experiment No.: 05

Title: Demonstration of plotting the data using python.

Objectives:

1. To learn about plotting the data.
2. To use matplotlib for data visualization/plotting.

Theory:

Data visualization is a very important part of data analysis. We can use it to explore our data. If we understand our data well, we'll have a better chance to find some insights. Finally, when we find any insights, we can use visualizations again to be able to share our findings with other people.

Basic Visualization Rules

Before we look at some kinds of plots, we'll introduce some basic rules. Those rules help us make nice and informative plots instead of confusing ones.

- The first step is to choose the appropriate plot type. If there are various options, we can try to compare them, and choose the one that fits our model the best.
- Second, when we choose your type of plot, one of the most important things is to label your axis. If we don't do this, the plot is not informative enough. When there are no axis labels, we can try to look at the code to see what data is used and if we're lucky we'll understand the plot. But what if we have just the plot as an image? What if we show this plot to your boss who doesn't know how to make plots in Python?
- Third, we can add a title to make our plot more informative.
- Fourth, add labels for different categories when needed.
- Five, optionally we can add a text or an arrow at interesting data points.
- Six, in some cases we can use some sizes and colors of the data to make the plot more informative.

Types of Visualizations and Examples with Matplotlib

There are many types of visualizations. Some of the most famous are: **line plot, scatter plot, histogram, box plot, bar chart, and pie chart**. But among so many options how do we choose the right visualization? First, we need to make some exploratory data analysis. After we know the shape of the data, the data types, and other useful statistical information, it will be easier to pick the right visualization type.

There are many visualization packages in Python. One of the most famous is Matplotlib. It can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, and web application servers.
Some basic functions of the **matplotlib.pyplot** subpackage are included in the examples below. Assume that the **matplotlib.pyplot** subpackage is imported with an alias **plt**.

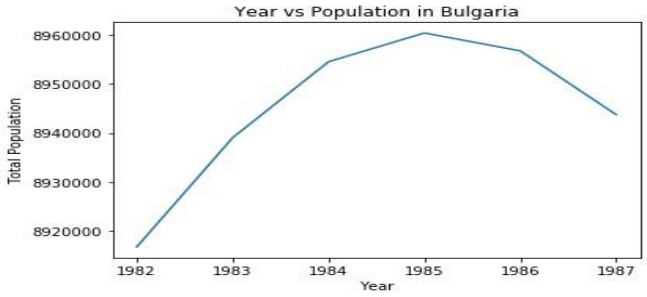
plt.title("My Title") will add a title "My Title" to your plot
plt.xlabel("Year") will add a label "Year" to your x-axis

plt.ylabel("Population") will add a label "Population" to your y-axis
plt.xticks([1, 2, 3, 4, 5]) set the numbers on the x-axis to be 1, 2, 3, 4, 5. We can also pass and labels as a second argument. For, example, if we use this code plt.xticks([1, 2, 3, 4, 5], ["1M", "2M", "3M", "4M", "5M"]), it will set the labels 1M, 2M, 3M, 4M, 5M on the x-axis.
plt.yticks() - works the same as plt.xticks(), but for the y-axis.

Line Plot:

It is a type of plot which displays information as a series of data points called "markers" connected by straight lines. In this type of plot, we need the measurement points to be ordered (typically by their x-axis values). This type of plot is often used to visualize a trend in data over intervals of time - a time series.
To make a line plot with Matplotlib, we call plt.plot(). The first argument is used for the data on the horizontal axis, and the second is used for the data on the vertical axis. This function generates your plot, but it doesn't display it. To display the plot, we need to call the plt.show() function. This is nice because we might want to add some additional customizations to our plot before we display it. For example, we might want to add labels to the axis and title for the plot.

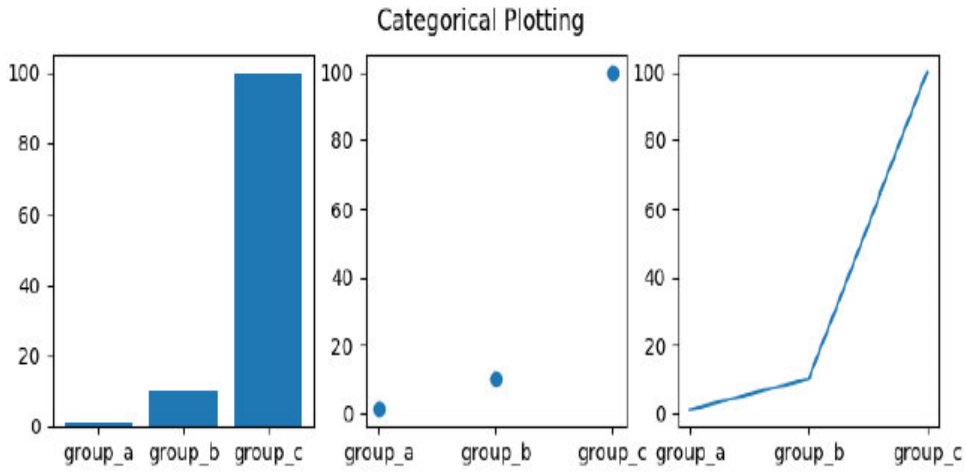
```
import matplotlib.pyplot as plt
years = [1983, 1984, 1985, 1986, 1987]
total_populations = [8939007, 8954518, 8960387, 8956741, 8943721]
plt.plot(years, total_populations)
plt.title("Year vs Population in Bulgaria")
plt.xlabel("Year")
plt.ylabel("Total Population")
plt.show()
```



Subplot

Many plot types can be combined in one figure to create powerful and flexible representations of data.For E.g.

```
names = ['group_a', 'group_b', 'group_c']
values = [1, 10, 100]
plt.figure(figsize=(9, 3))
plt.subplot(131)
plt.bar(names, values)
plt.subplot(132)
plt.scatter(names, values)
plt.subplot(133)
plt.plot(names, values)
plt.suptitle('Categorical Plotting')
plt.show()
```



The **subplot()** command specifies numrows, numcols, plot_numberwhere plot_number ranges from **1** to **numrows*numcols**. The commas in the subplotcommand are optional if **numrows*numcols<10**. So subplot(211) is identical tosubplot(2, 1, 1).

You can create an arbitrary number of subplots and axes. If you want to place an axes manually, i.e., not on a rectangular grid, use the axes() command, which allows youto specify the location as axes([left, bottom, width, height]) where all valuesare in fractional (0 to 1) coordinates.

You can clear the current figure with **clf()** and the current axes with **cla()**.If you are making lots of figures, you need to be aware of one more thing: the memoryrequired for a figure is not completely released until the figure is explicitly closed with**close()**. Deleting all references to the figure, and/or using the window manager to killthe window in which the figure appears on the screen, is not enough, because pyplotmaintains internal references until close() is called.