Lab # 02

National University of Computer & Emerging Sciences

Lab Manual

CS461: Artificial Intelligence Lab

Course Instructor	Dr. Hafeez-Ur-Rahman
Lab Instructor	Muhammad Yousaf
Semester	Spring 2021

1.2. Python

The objective of this session is to get exposure to Python programming within PyScripter environment, and write some simple code to access data and plot it using some key Python libraries.

Learning outcomes

1) Write simple Python code inside PyScripter/Sublime to create random numbers and frequency histogram using the numpy library.
2) Create graphs in Python using the matplotlib library.
3) Understand and work with numeric data in the library pandas.

1.2.1. Introduction

When it comes to programming, adopting a language depends on what you want to accomplish. For example, if you want to write a code to solve complex numerical equations, you may use C++ or Fortran. Similarly, if you are interested in statistics, R may be a good choice. Accordingly, when it comes to implementation of pattern recognition or quick scripting for artificial intelligence problems, python is widely used as it is a higher level language that is open source, cross-platform, and is easy to learn and code. Additionally, standard libraries for clustering, optimization, and classification are available for direct use in Python. You can find more about python at http://www.python.org/doc/ and http://www.diveintopython.org.

Now, you need an "environment" to write your Python code. There are many other independent softwares, such as IDLE, Spider, Sublime, and PyScripter, to write and run a Python code. In this class, we are going to use PyScripter as this is one of the most famous python IDE available. **Getting PyScripter** (Purdue students can skip the download step and just open the program from the Start menu). **Download** PyScripter from https://sourceforge.net/projects/pyscripter and install it on your computer.

Start/open PyScripter by double clicking the icon or from the start menu. The PyScripter window, shown on the next page, allows you to write the Python code, run the program, and look at the results. The PyScripter window has four sections as shown in the Figure 1.2. (Note: your background color in PyScripter can be white or black; it does not matter)

1.2.2 Python Libraries

A library is basically a collection of standard code or sub-routines that are frequently used in programming. A library helps a coder (you) to accomplish a programming task in a few lines of code by borrowing all the lines stored in a function within a library. One of the greatest strengths of Python is its large library that provide access to tools for numerical computations as well as

GIS applications. Some common libraries that we will be using in this class are briefly introduced below, along with a sample code that you can try in your PyScripter.

1.2.2.1. NumPy

If you are familiar with Matlab, numpy provides Matlab like functionality within Python. Using NumPy you can solve multi-dimensional arrays and matrices, and perform simple to complex mathematical operations including statistics. Now, lets see how to use numpy in PyScripter to generate 1000 random numbers from a normal distribution with mean = 100 and standard deviation = 15.

Write the first line of code as below. (Note: All code in this tutorial will be highlighted to distinguish it from the regular text)

```
import numpy as np
```

Using this single statement, we have imported or borrowed the *numpy* library and referenced it as np. This is how we will import any library. You can use any name to reference a library. In this case we used np as it is an abbreviation of numpy. You can use nmy, npy or anything. You get the point! Use something logical!

Now we are going to define two variable for the mean and standard deviation, and assign them the value of 100 and 15, respectively. Again, you can pick any name for defining a variable, but it is a good practice to use something that is self-explanatory. We use greek letters and for mean and standard deviation, respectively. Lets write our second line of code to define these two variables as below.

```
mu, sigma = 100, 15
```

Here we defined both variables in a single statement. We could also do mu = 100 on one line and sigma = 15 on another line. Once we have the mean and standard deviation, we can then generate values using the following statement.

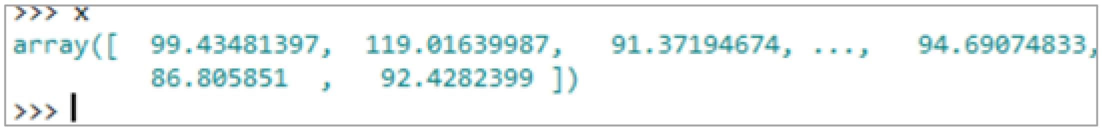
```
x = mu + sigma*np.random.randn(1000)
```

The randn function above generates random numbers from a standard normal distribution (mean = 0 and variance = 1), which is then scaled to our mu and sigma, and stored in variable *x*. Basically, *x* is an array to store 1000 random values from the specified normal distribution.

Q. What is an array?

Now run the code by clicking on the green triangle button ▶ in the toolbar (ctrl + F9). In the lower left corner, you will see “Running” and then "Ready" or “Script run OK”. This means the code ran successfully, and you can now see the results.

We defined an array named x to store our random values. You can see the results by calling this x array in Python Interpreter window. Simply type x in the Python interpreter window, and press *Enter*. You should then see all the random values as shown below.



```
>>> x
array([  99.43481397,  119.01639987,   91.37194674, ...,   94.69074833,
         86.805851  ,   92.4282399 ])
>>>
```

Congratulations! You just finished your first code in Python! Let's experiment with another library in Python.

1.2.2.2. Matplotlib

matplotlib is a plotting library for the Python programming language and it can be used to make plots in PyScripter. Lets use this library to create a histogram from the x array we just created. Import the pyplot sub-library from matplotlib and name it as plt. Write the following code in your pyscripter window as shown below. (Note: you can either write this statement after your last statement, but it is a good practice to have all imports on the top so you know what libraries are available in the code.)

```
import matplotlib.pyplot as plt
```

Now lets create the histogram of x by using plt. Write the following code in your pyscripter window. This will generate the histogram by dividing the array into 30 bins. Bins size is a parameter so you can choose some other value if you wish.

```
plt.hist(x,bins=30)
```

After creating the histogram, lets plot the histogram by using the following code.

```
plt.show()
```

Now run the code, and you should see something similar to the following histogram.

If you want, you can play around with the bin size to see how the plot changes.

Lets use pyplot to create scatter plot between two random variables. Define two random variables *a* and *b* as shown below. Remember we used *randn* earlier to create random numbers from a standard normal distribution. The random function will generate purely random numbers. The number inside the parenthesis defines the size.

```
a = np.random.random(100)
```

```
b = np.random.random(100)
```

Now plot a and b on a scatter plot by using the code below. After you type plt, you will see a list of associated functions so pick scatter and provide a and b as two variables as shown below.

```
plt.scatter(a,b)
```

Then show the plot.

```
plt.show()
```

You are ready to run the code at this point. After you run the code, you will first see the histogram, which you have to close, and then the scatter plot will be displayed as shown below.

How can you avoid the histogram to show and let the program only show the scatter plot? Think of #

1.2.2.3. Pandas

Python Data Analysis Library, or Pandas, is a Python package providing fast, flexible, and expressive data structures designed to make working with is “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.

As usual, lets first import the *pandas* library as *pd* <all import statements will be at the beginning of your code>

```
import pandas as pd
```

Now read the csv file from your working directory and store it in data as shown below (note the forward slash in the directory name). *usecols* will store the data from each column in the respective array.

```
data = pd.read_csv('C:/temp/wabash.csv', usecols=['datetime', 'discharge', 'stage'])
```

The above statement reads the CSV file and stores the data in three arrays: time stamp in ‘datetime’, flow values in ‘discharge’, and stage in ‘stage’. Because the datetime has both date and time in hh:ss format, it is stored as a string (or text) instead of date. Lets convert this array from string to date by using the statement below. This statement will convert the string to a datetime format array and stored it in timestamp series.

```
timestamp = pd.to_datetime(data.datetime)
```

Now we can plot information from data by using the statement below.

```
plt.plot(data.stage,data.discharge)
```

The scatter function we used earlier gave us points. The plot function will give us a line. Lets also define our x and y axes labels, and provide a title for the plot as shown below. The functions we are using here are self-explanatory.

```
plt.xlabel("Stage(ft)")
```

```
plt.ylabel("Discharge(cfs)")
```

```
plt.title("Stage Discharge Curve for Wabash")
```

Finally, show the plot by using the statement below.

```
plt.show()
```

Run your code. If you want you can tell the program to skip the earlier plots by commenting the related statements using #. The stage discharge curve is shown below. Now write some python code on your own to create a discharge time series as shown below. Make sure it has axes and plot titles.

2.0. What is PROLOG?

- Prolog = Programmation en Logique (Programming in Logic).
- Prolog is a declarative programming language unlike most common programming languages.
- In a declarative language
  - o the programmer specifies a goal to be achieved
  - o the Prolog system works out how to achieve it
- relational databases owe something to Prolog

- traditional programming languages are said to be **procedural**
- procedural programmer must specify in detail how to solve a problem:

  mix ingredients;
  beat until smooth;

  bake for 20 minutes in a moderate oven;
  remove tin from oven;

  put on bench;
  close oven;
  turn off oven;

- in purely declarative languages, the programmer only states what the problem is and leaves the rest to the language system
- We'll see specific, simple examples of cases where Prolog fits really well shortly.

Applications of PROLOG

Some applications of Prolog are:

- intelligent data base retrieval
- natural language understanding
- expert systems
- specification language
- machine learning
- robot planning
- automated reasoning
- problem solving

Relations

- Prolog programs specify *relationships* among objects and properties of objects.
- When we say, "John owns the book", we are declaring the ownership relationship between two objects: John and the book.
- When we ask, "Does John own the book?" we are trying to find out about a relationship.
- Relationships can also rules such as:

  Two people are sisters **if**

they are both female **and**

they have the same parents.

- A rule allows us to find out about a relationship even if the relationship isn't explicitly stated as a fact.

A little more on being sisters

- As usual in programming, you need to be a bit careful how you phrase things:
- The following would be better:

A and B are sisters **if**

A and B are both female **and**

they have the same father **and**

they have the same mother **and**

A is not the same as B

Programming in prolog

- declare facts describing explicit relationships between objects and properties objects might have (e.g. Jameel likes pizza, grass has_colour green, Tiger is_a_dog, Chutki taught Jamila Swedish)
- define rules defining implicit relationships between objects (e.g. the sister rule above)

  and/or rules defining implicit object properties (e.g. X is a parent if there is a Y such that Y is a child of X).

One then uses the system by:

- asking questions above relationships between objects, and/or about object properties (e.g. does Jameel like pizza? is Jan a parent?)