Introduction to MATPLOTLIB (http://matplotlib.org/)

One of the most popular uses for Python is data analysis. Naturally, data scientists want a way to visualize their data. Either they are wanting to see it for themselves to get a better grasp of the data, or they want to display the data to convey their results to someone. With Matplotlib, arguably the most popular graphing and data visualization module for Python, this is very simplistic to do. In this tutorial, I will be covering all of what I consider to be the basic necessities for Matplotlib.

In order to get the Matplotlib, you should first head to Matplotlib.org and download the version that matches your version of Python. From there, it'd be wise to go ahead and make sure you have pyparsing, dateutil, six, numpy, etc. You can get all of these as well, if you are on a Windows machine by heading to: http://www.lfd.uci.edu/~gohlke/pythonlibs/#matplotlib

Once you have Matplotlib installed, be sure to open up a terminal or a script, type:

```
import matplotlib
```

Make sure there are no errors on the import. If there are, read the error. Most often, either the bit version does not match (64 bit vs 32 bit), or you are missing a package like dateutil or pyparsing.

Once you can successfully import matplotlib, then you are ready to continue.

Legends, Titles, and Labels with Matplotlib

In this tutorial, we're going to cover legends, titles, and labels within Matplotlib. A lot of times, graphs can be self-explanatory, but having a title to the graph, labels on the axis, and a legend that explains what each line is can be necessary.

To start:

```
import matplotlib.pyplot as plt


x = [1,2,3]

y = [5,7,4]


x2 = [1,2,3]

y2 = [10,14,12]
```

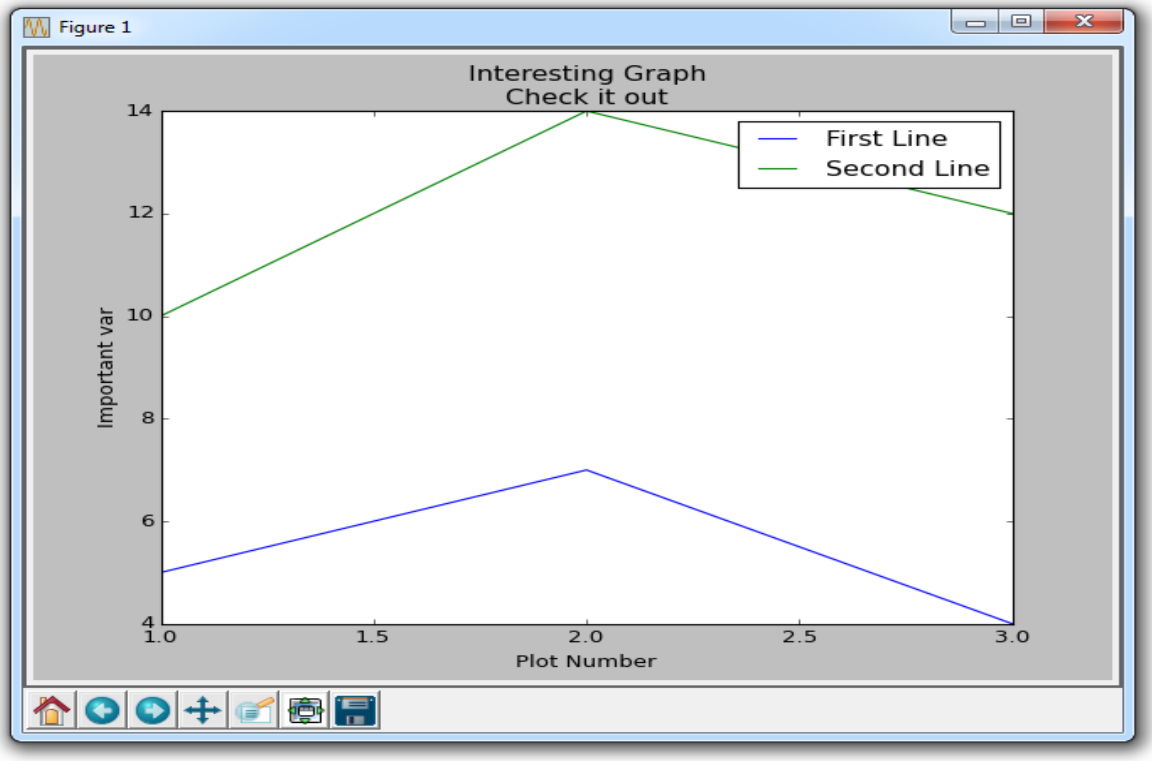
This way, we have two lines that we can plot. Next:

```
plt.plot(x, y, label='First Line')

plt.plot(x2, y2, label='Second Line')
```

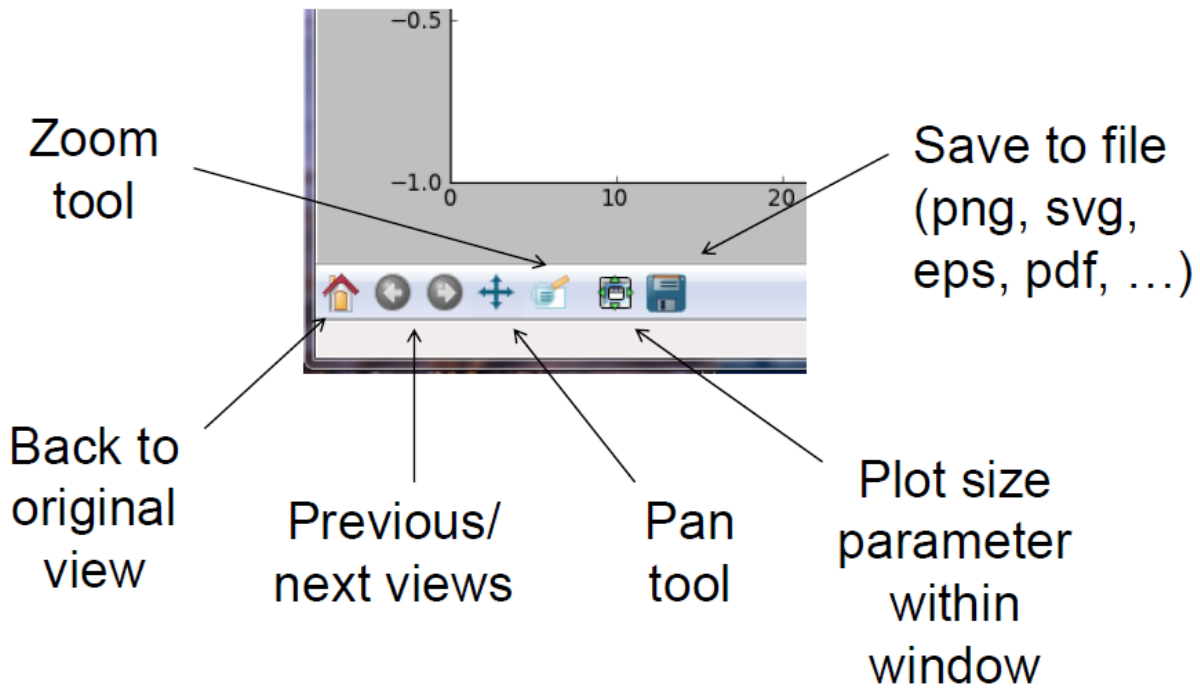
Here, we plot as we've seen already, only this time we add another parameter "label." This allows us to assign a name to the line, which we can later show in the legend. The rest of our code:

```
plt.xlabel('Plot Number')

plt.ylabel('Important var')

plt.title('Interesting Graph\nCheck it out')

plt.legend()

plt.show()
```

With plt.xlabel and plt.ylabel, we can assign labels to those respective axis. Next, we can assign the plot's title with plt.title, and then we can invoke the default legend with plt.legend(). The resulting graph:



Matplotlib Menu Bar

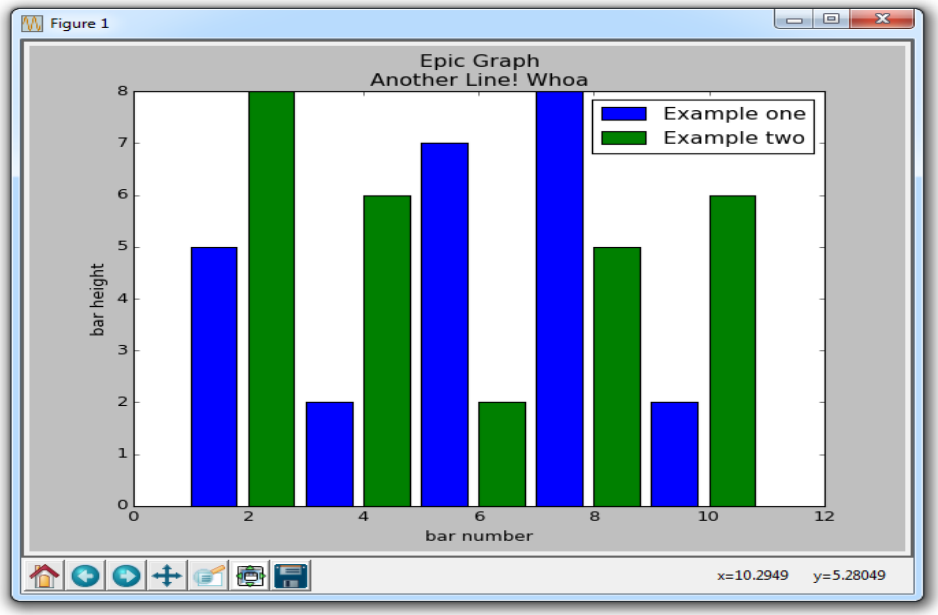


Bar Charts and Histograms with Matplotlib

In this tutorial, we cover bar charts and histograms with Matplotlib. First, let's cover a bar chart.

```
import matplotlib.pyplot as plt

          plt.bar([1,3,5,7,9],[5,2,7,8,2], label="Example one")


plt.bar([2,4,6,8,10],[8,6,2,5,6], label="Example two", color='g')

plt.legend()

plt.xlabel('bar number')

plt.ylabel('bar height')


plt.title('Epic Graph\nAnother Line! Whoa')


plt.show()
```

The plt.bar creates the bar chart for us. If you do not explicitly choose a color, then, despite doing multiple plots, all bars will look the same. This gives us a change to cover a new Matplotlib customization option, however. You can use color to color just about any kind of plot, using colors like g for green, b for blue, r for red, and so on.

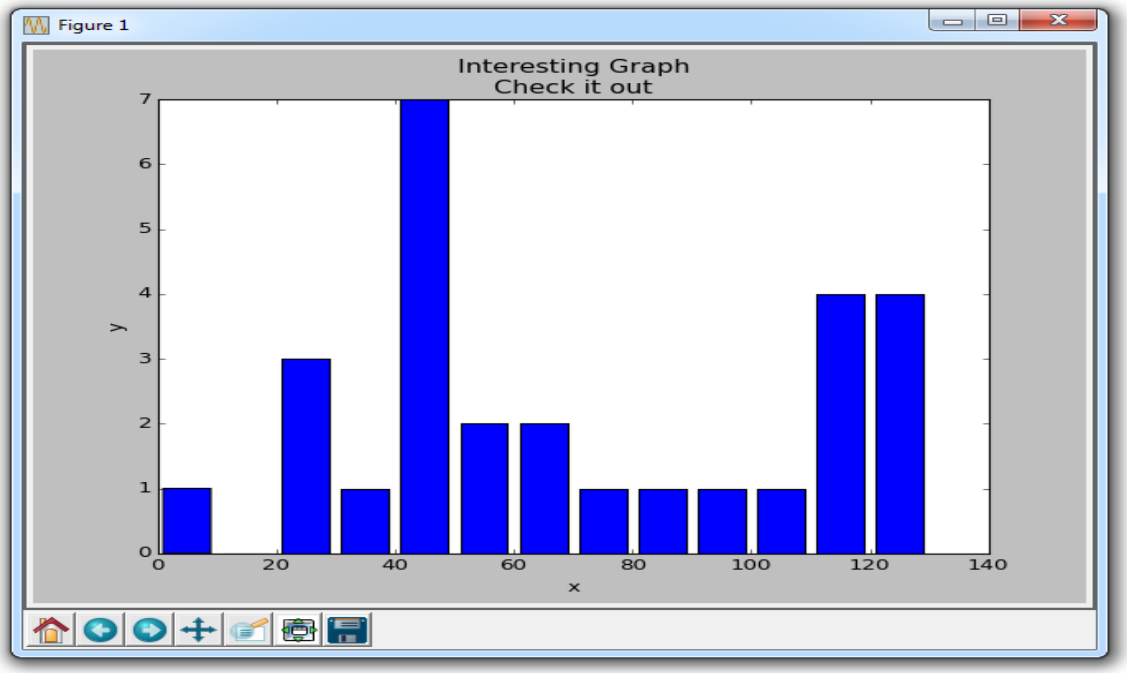


Next, we can cover histograms. Very much like a bar chart, histograms tend to show distribution by grouping segments together. Examples of this might be age groups, or scores on a test. Rather than showing every single age a group might be, maybe you just show people from 20-25, 25-30... and so on. Here's an example:

```
import matplotlib.pyplot as plt


population_ages = [22,55,62,45,21,22,34,42,42,4,99,102,110,120,121,122,130
,111,115,112,80,75,65,54,44,43,42,48]


bins = [0,10,20,30,40,50,60,70,80,90,100,110,120,130]


plt.hist(population_ages, bins, histtype='bar', rwidth=0.8)
```

```
plt.xlabel('x')

plt.ylabel('y')

plt.title('Interesting Graph\nCheck it out')

plt.legend()

plt.show()
```

The resulting graph is:



For plt.hist, you first put in all of the values, then you specify into what "bins" or containers you will place the data into. In our case, we are plotting a bunch of ages, and we want to display them in terms of increments of 10 years. We give the bars a width of 0.8, but you can choose something else if you want to make the bars thicker, or thinner.

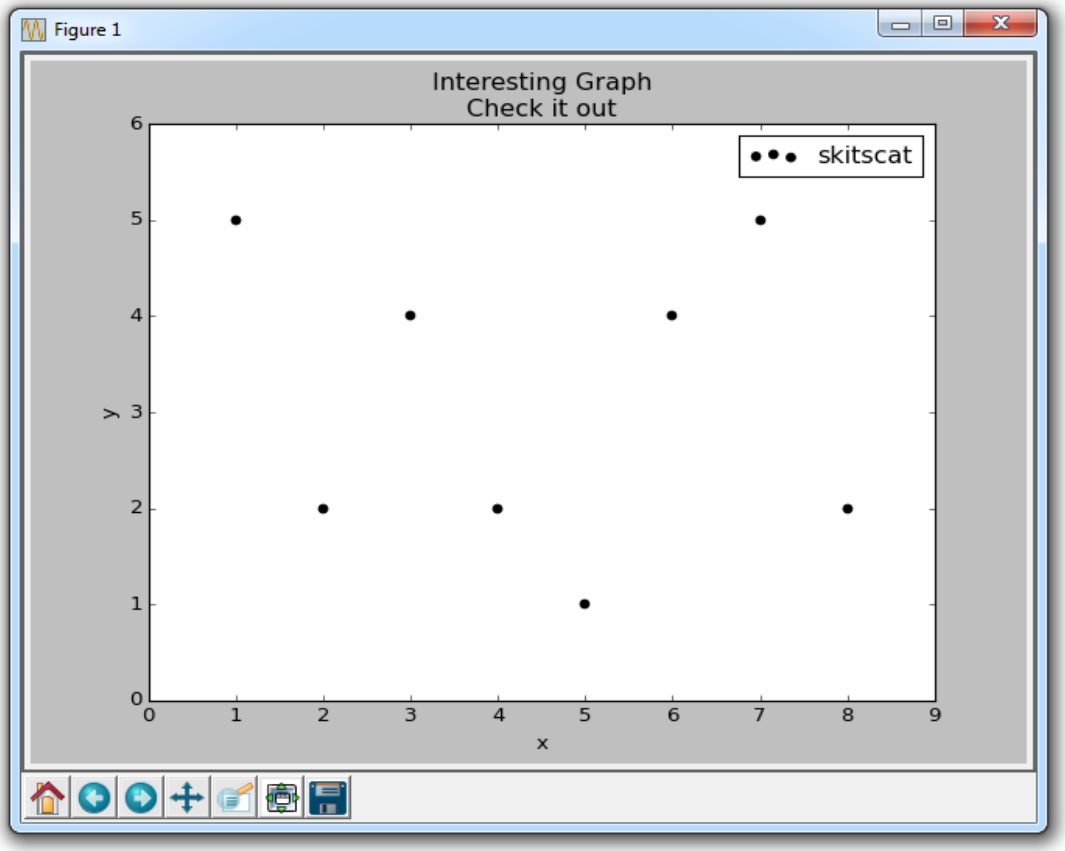
Scatter Plots with Matplotlib

Next up, we cover scatter plots! The idea of scatter plots is usually to compare two variables, or three if you are plotting in 3 dimensions, looking for correlation or groups.

Some sample code for a scatter plot:

```
import matplotlib.pyplot as plt


x = [1,2,3,4,5,6,7,8]

y = [5,2,4,2,1,4,5,2]


plt.scatter(x,y, label='skitscat', color='k', s=25, marker="o")


plt.xlabel('x')

plt.ylabel('y')

plt.title('Interesting Graph\nCheck it out')

plt.legend()

plt.show()
```

The result:



The plt.scatter allows us to not only plot on x and y, but it also lets us decide on the color, size, and type of marker we use. There are a bunch of marker options, see the Matplotlib Marker Documentation for all of your choices.

Markers Shapes and Colors

http://matplotlib.org/api/markers_api.html

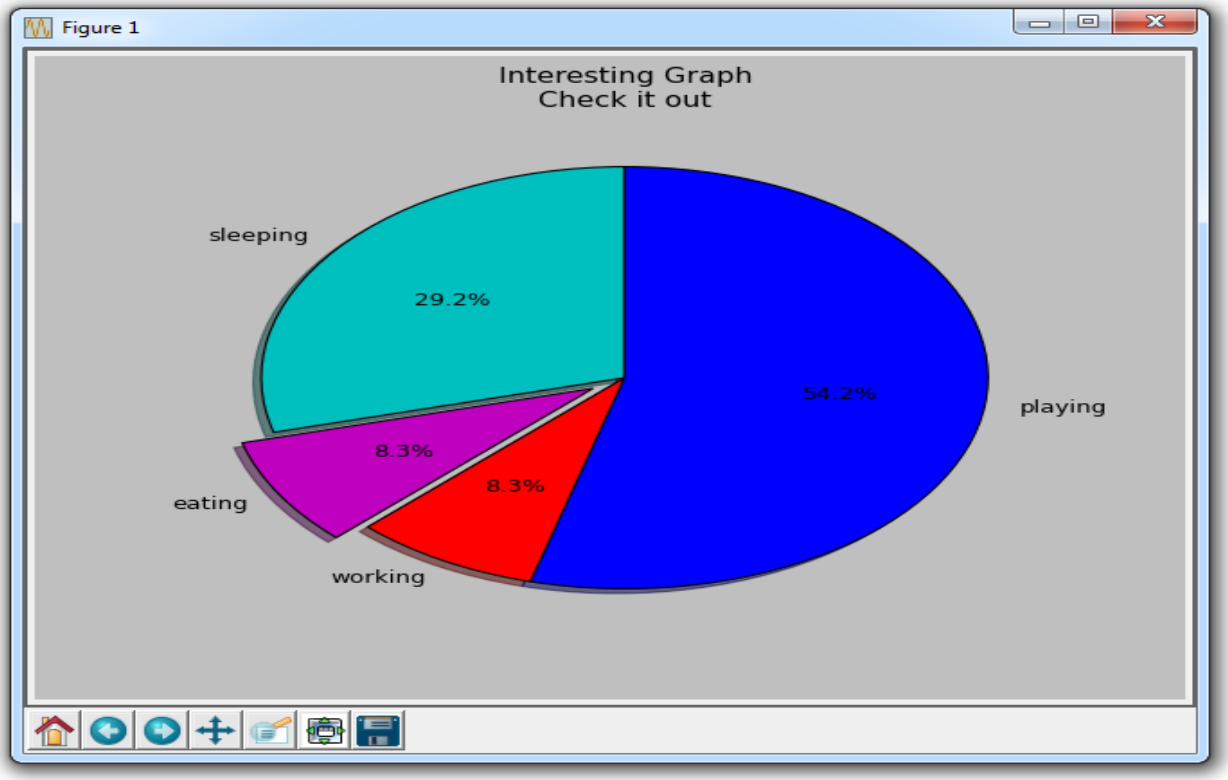
http://matplotlib.org/api/colors_api.html

Pie Charts with Matplotlib

Pie charts are a lot like the stack plots, only they are for a certain point in time. Typically, a Pie Chart is used to show parts to the whole, and often a % share. Luckily for us, Matplotlib handles the sizes of the slices and everything, we just feed it the numbers.

```
import matplotlib.pyplot as plt


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('Interesting Graph\nCheck it out')

plt.show()
```



Within the plt.pie, we specify the "slices," which are the relevant sizes for each part. Then, we specify the color list for the corresponding slices. Next, we can optionally specify the "Start angle" for the graph. This lets you start the line where you want. In our case, we chose a 90 degree angle for the pie chart, which means the first division will be a verticle line. Next, we can optionally add a shadow to the plot for a bit of character, and then we can even use "explode" to pull out a slice a bit.

We have four total slices, so, with explode, if we didn't want to pull out any slices at all, we would do 0,0,0,0. If we wanted to pull out the first slice a bit, we would do 0.1,0,0,0. Finally, we do autopct to optionally overlay the percentages on to the graph itself.

Using Files with Matplotlib

Great, so we're Matplotlib wizards now, and we're ready to journey into the real world and plot stuff! One of the more popular file types that you'll first start using is CSVs. Eventually, you'll probably find

that people stop using CSV files and use either databases. For now, let's just cover CSV. There are obviously many ways to read files in Python. You can use Python's CSV module that is a part of the standard library.

You can make use of Numpy's loadtxt as well, which we'll be using. Another fantastic choice is using **Pandas**! So there are many choices to consider, but, for now, we're going to use Numpy.

Depending on your goals and requirements, you might eventually wind up choosing something else. NumPy is very open-ended for data analysis, yet still very powerful. On the other hand,Pandas is going to be a great choice for most people, but it is less open-ended.

Loading Data from Files for Matplotlib

Many times, people want to graph data from a file. There are many types of files, and many ways you may extract data from a file to graph it. Here, we'll show a couple of ways one might do this. We'll show how to utilize NumPy, which is a third-party module, to load files.

Next, you're going to need some sample data! Either grab some that you'd like to use if you think you are going to be able to make the necessary edits, or use this sample data in "examples.csv" and put it in the same directory as your current script. The sample data looks like:

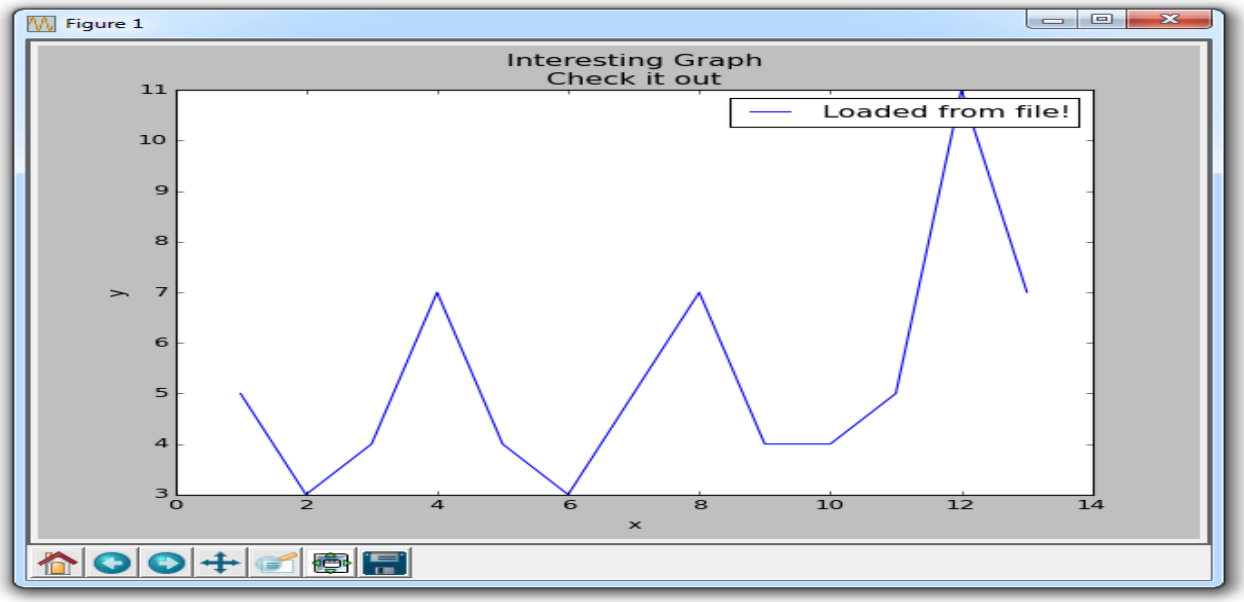
```
1,5
2,3
3,4
4,7
5,4
6,3
7,5
8,7
9,4
10,4
```

Save that, and then the code to plot from this data set:

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

```
import numpy as np


x, y = np.loadtxt('examples.csv', delimiter=',', unpack=True)

plt.plot(x,y, label='Loaded from file!')


plt.xlabel('x')

plt.ylabel('y')

plt.title('Interesting Graph\nCheck it out')

plt.legend()

plt.show()
```

The result should look like this:

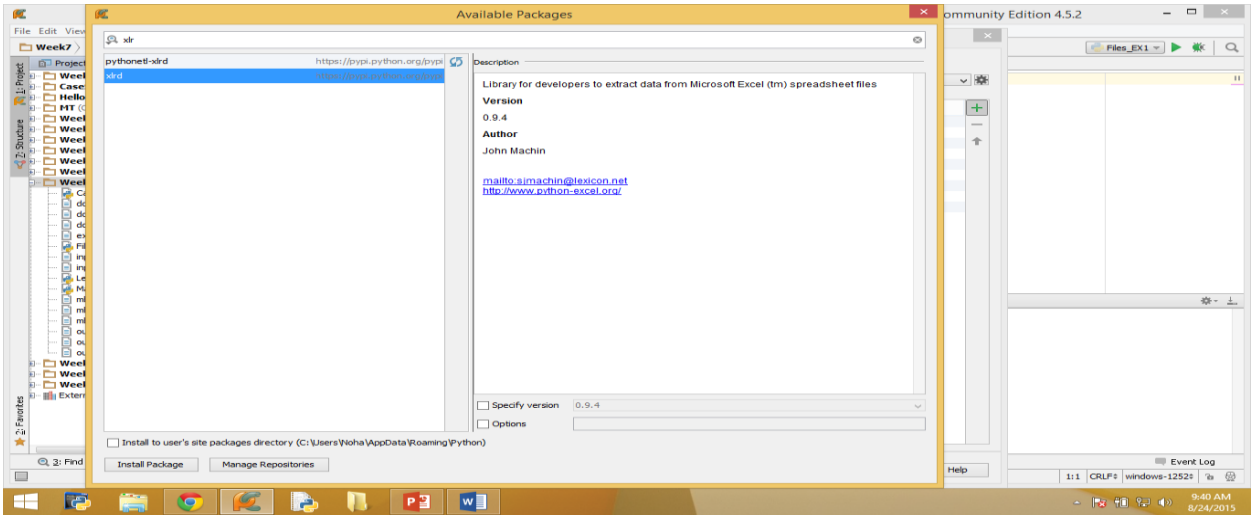


Here, our major new things are importing numpy, and then using numpy's loadtxt function.

Loadtxt can be used to load more than just .txt files. It's just load things with text, that's all. Here, we are unpacking the contents of exampleFile.csv, using the delimiter of a comma. It's important to note here that you MUST unpack the exact same number of columns that will come from the delimiter that you state. If not, you'll get an error.

Loading Data from Excel file for Matplotlib

You should download a package called xlrd to deal with Excel files, see below.



Acknowledgments

The materials used in this tutorial are obtained from http://pythonprogramming.net/matplotlib-python-3-basics-tutorial/

More Details

- http://matplotlib.sourceforge.net/users/screenshots.html
- http://matplotlib.sourceforge.net/gallery.html
- http://pythonprogramming.net/legends-titles-labels-matplotlib-tutorial/?completed=/matplotlib-python-3-basics-tutorial/