Episode 8
Matplotlib, SciPy, and Pandas

Now that we understand ndarrays, we can start using other packages that utilize them. In particular, we're going to look at Matplotlib, SciPy, and Pandas. Matplotlib is a package that can make a wide variety of plots and graphs. SciPy contains many useful mathematical functions as well as a number of subpackages that provide specialized capabilities. And Pandas a Python data analysis library.

We will start with Matplotlib. The following code makes a sample plot.

```
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-4.,4.,401)
y=1./(np.pi*(1.+x**2))
plt.title("Y versus X")
plt.plot(x,y)
plt.show()
```

We import the packages we need. Then we use a NumPy function called `linspace` to generate 401 points from -4.0 to 4.0 *inclusive*. (We give `linspace` endpoints and not a range.) We compute a function of those points and then make a simple line plot. The `plt.show()` is required if you run a script, but if you type directly into a iPython console it will show the plot when it's created.

To make a fancier plot follow the example here:

```
import numpy as np
import matplotlib.pyplot as plt
x1 = np.linspace(0.0, 5.0)
x2 = np.linspace(0.0, 2.0)
y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)
y2 = np.cos(2 * np.pi * x2)
plt.subplot(2, 1, 1)
plt.plot(x1, y1, 'yo-')
plt.title('A tale of 2 subplots')
plt.ylabel('Damped oscillation')
plt.subplot(2, 1, 2)
plt.plot(x2, y2, 'r.-')
plt.xlabel('time (s)')
plt.ylabel('Undamped')
plt.show()
```

This code sets up two independent variables x1 and x2, and a function defined on each one, y1 and y2. Subplots are used to place multiple plots on the same graph. For this example we set up two rows and one column (2,1,..). The first plot is plot 1 (notice that we count from 1 here) and the second is plot 2. In the first call to `plot` we plot y1 versus x1 using yellow circles jointed by lines. In the second call we use red dots joined by lines. The title must be drawn as part of the first subplot to place it at the top.

Now let's look at an example of a two-dimensional plot.

```
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-2.*np.pi,2.*np.pi,100)
y=np.linspace(-4.*np.pi,4.*np.pi,200)
X,Y=np.meshgrid(x,y)
Z=np.sin(X)+np.cos(Y)
plt.contour(X,Y,Z)
```

A meshgrid is a set of tuples, one for each grid point, for which each point has both an `x` and a `y` value. Thus `y` values are replicated along `X` and `x` values are copied along `Y`. By this means, the computation of `Z` has the values of both `x` and `y` at each point.

For filled contours use
```
plt.contourf(X,Y,Z)
```

Now we will generate a surface plot of the same function. We need to add an import:

```
from mpl_toolkits.mplot3d import Axes3D
```

The `figure` function creates a new figure. Without it, the old figure will be overwritten.

```
fig=plt.figure()
my_axes=fig.add_subplot(111,projection='3d')
my_axes.plot_surface(X,Y,Z)
```

We can create many more types of graphs with Matplotlib. Students should look at the gallery and example code at http://matplotlib.org.

Now we'll look at SciPy. This is a large collection of modules and subpackages and we will not study them in detail. More information is at http://scipy.org (information about affiliated projects, including NumPy, Pandas, and others, is also at that URL). Particularly useful is https://docs.scipy.org/doc/scipy/reference/.

Our example uses scipy.linalg to solve a system of linear equations.

$x + 2y + 3z = 11$

$4x + 5y + 6z = 12$

$7x + 8y + 9z = 13$

In matrix form this is

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 11 \\ 12 \\ 13 \end{bmatrix}$$

The code to solve this is simple:

```
from scipy import linalg
A=np.array([[1,2,3],[4,5,6],[7,8,9]])
B=np.array([11,12,13])
X=linalg.solve(A,B)
print X
```

We can find many other useful mathematical algorithms in SciPy, including special functions, numerical integration, signal processing, and optimization packages. However, we leave these as an exercise for the student and continue to Pandas.

Pandas is a package for data analytics. It is available within Anaconda and can be installed if it is not present in other installations of Python. It introduces a new data structure called a DataFrame. The DataFrame concept is borrowed from the R programming language. It can be conceptualized as a representation of a spreadsheet. It stores column names from a header, columns in the form of another data structure called a Series, and other information about the data. If the data are of appropriate types they can be extracted into NumPy arrays.

Our example uses another package called seaborn, a package based on Matplotlib for statistical visualizations. In the videos the version of Anaconda we used did not include it by default, so we demonstrated how to install it, but newer versions of Anaconda may include it by default. You can check whether it is included by examining the list of Installed packages in Environments through the Anaconda Navigator. You can also try to import it

```
import seaborn as sb
```

If this succeeds without generating an error, the package is installed. If it is not present you will need to install it.

We assume for the rest of the episode that you have imported the necessary packages with

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

as well as seaborn as sb. Seaborn includes a famous dataset from the 1930s about characteristics of various iris cultivars. We will load this dataset and use it to illustrate some of the capabilities of Pandas.

```
iris=sb.load_dataset('iris')
```

First let's look at the column headers and the first few lines of data.

```
iris.head()
```

To see the end of the data type

```
iris.tail(20)
```

This shows the last 20 lines (the default is 5 lines). The column names are stored as

```
iris.columns
```

while the data values are

```
iris.values
```

If we so wish, we could use the `values` attribute to extract the numerical data values into a NumPy array

```
data_vals=iris.values[:,:4].astype(float)
```

We used the NumPy built-in function `astype` to convert from a more generalized type "object" into an array of floating-point values.

We can summarize the data with

```
iris.describe()
```

If we want to extract data about the column labeled "sepal_width" we use

```
sepal_w=iris["sepal_width"]
```

We have shown how to extract the data into a NumPy array, but we can also use the Pandas built-in `iloc` to create a new DataFrame that is a subset of the original.

```
dset=iris.iloc[:,0:4]
dset.columns
```

Now let's compute some statistics about the data.

```
iris.mean()
```

This is over all the data, but we may want to group it by species.

```
iris.groupby('species').mean()
```

Finally, let's plot a histogram of data by species. Without Pandas this could require dozens of lines of NumPy and Matplotlib code. With Pandas it is a single statement.

```
iris.hist(by='species')
plt.show()
```

We hope this episode has inspired you to learn more about the important packages Matplotlib, SciPy, and Pandas. Now that we can manipulate data, we will next learn how to read and write it from and to files.

Tutorials for Matplotlib can be found at
http://www.scipy-lectures.org/intro/matplotlib/matplotlib.html
http://www.randalolson.com/2014/06/28/how-to-make-beautiful-data-visualizations-in-python-with-matplotlib/
Matplotlib provides a tutorial for a subset of Matplotlib and NumPy called pyplot at
https://matplotlib.org/users/pyplot_tutorial.html

Once you are comfortable with Matplotlib you may want to look at Plotly

https://plot.ly/python/

Anaconda users can install it with

```
conda install plotly
```

For more advanced 3D plots, Mayavi is a good option (Python 2.7 only)

http://www.scipy-lectures.org/packages/3d_plotting/index.html

Students of earth sciences might be interested in Basemap for plots

https://matplotlib.org/basemap/

```
conda install basemap
```

and xarray for working with data, particularly in NetCDF format, in a Pandas-like syntax.

```
conda install xarray
```

There are fewer tutorials for SciPy because it is a larger package with multiple subpackages. The standard tutorial for the major packages is at

https://docs.scipy.org/doc/scipy/reference/tutorial/

A good general site is

http://www.scipy-lectures.org/

They provide a tutorial at

http://www.scipy-lectures.org/intro/scipy.html

Another aspect of SciPy is the "scikits." These are generally less well developed and not as comprehensive as base SciPy packages, but can contain some useful functionality. The most popular is scikits-learn for machine learning

http://www.scipy-lectures.org/packages/scikit-learn/index.html

There are a number of Pandas tutorials. The official Pandas site provides a guide at

http://pandas.pydata.org/pandas-docs/stable/tutorials.html

A good introductory tutorial is at

http://synesthesiam.com/posts/an-introduction-to-pandas.html