Exploring Data Through Plotting

Realizing that there’s more to the solar system than just the Sun, you have commandeered the *Cassini* spacecraft and performed a stellar occultation of the rings of Saturn. Stellar occultations work by watching a star pass behind a region of space and measuring how much the star dims. The more it dims, the denser the material in that region of space must be. Let’s find a curious feature within the rings and make a plot of it.

To start on this task, create a new Python file. Remember to give it a descriptive name.

The occultation data is stored in a file named alpleo9.csv. This is a simple ascii file in the ubiquitous format called Comma-Separated Values, a common way of storing any data, especially with different columns for different kind of data. CSV files have the major advantage that they are human readable for a quick glance to check if the file is the right one, plus they are very easy to read in for analysis.

To read a CSV file with Pandas, there are several tools available:

* numpy.loadtxt and numpy.genfromtxt (adding dealing with missing data points to loadtxt)
* the Python “csv” module
* the “pandas” module and it’s very convenient “read\_csv” function.
* Use pandas to read the CSV file into a pandas dataframe.

Now that we have a pandas dataframe…

Python dictionaries are pairs of *keys* and *values*. A key is a string that identifies the entry within the dictionary. A value can be any Python data type. Dictionaries are created like this:

my\_dict = {‘name’: ‘Morgan’, ‘age’: 25, ‘favorite\_numbers’: [0,1,5]}

and then accessed like this:

>>>print(my\_dict[‘name’])

Morgan

>>>print(my\_dict[‘favorite\_numbers’])

[0,1,5]

You can retrieve a list of all keys in a given dictionary using the keys() method on the list:

>>>print(my\_dict.keys())

dict\_keys([‘name’, ‘age’, ‘favorite\_numbers’])

Two keys interest us right now in the dictionary we just loaded from the pickle file. They are ‘data’ and ‘radius’. The unit for data is counts and the unit for radius is kilometers.

* Copy the values associated with the ‘data’ and ‘radius’ keys into two separate variables for easy use.

Now that we have the values we’re interested in, it’s time to make a preliminary plot. To do this, we need to import another external package, matplotlib. This is a very large package, however, and we don’t need it all. In Python, you can easily import just the pieces you need using the from command. To import only the pyplot component of matplotlib, use the line:

from matplotlib import pyplot

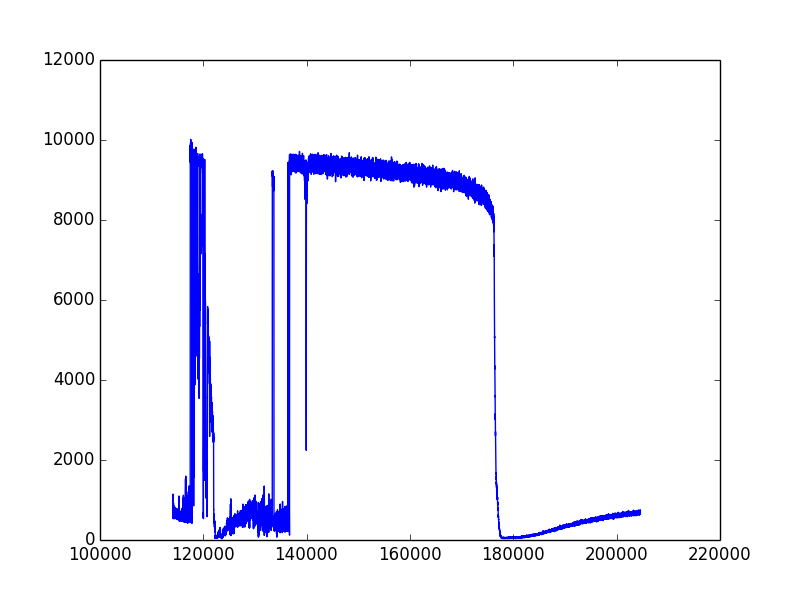
Remember to place all imports at the top of your program.

* Import pyplot

Now it’s time to make the plot! In pyplot, plots are first created and configured before finally being displayed to the user. The plot(*x*,*y*) function in pyplot will produce a simple plot and the show() function will display it to the user. A good thing to know about plots is that they interrupt the execution of your program. This means that while a plot window is up, Python will just sit there waiting for you to close it before moving on.

* Make and display a plot of radius vs data

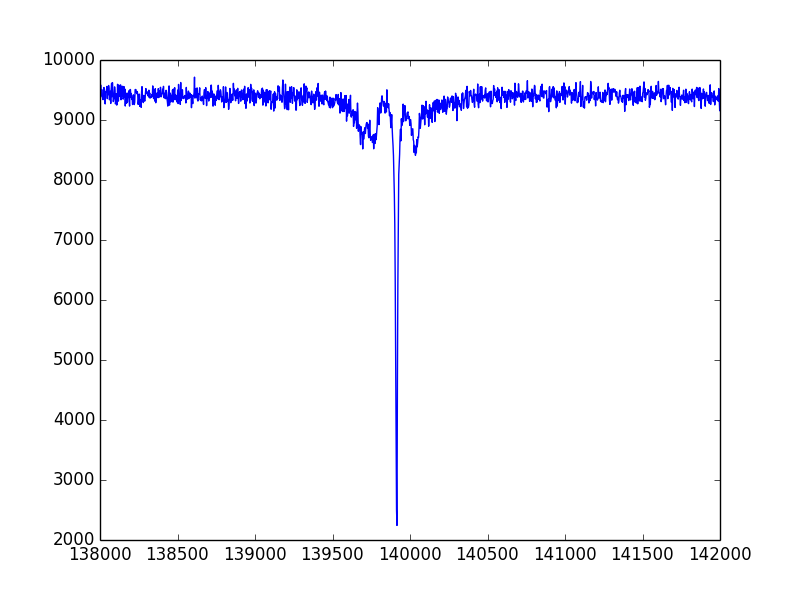
If you’ve made it successfully to this point, you should have a plot that looks like this:



Doesn’t look like much, eh? What you’re looking at is an observation of three regions of the rings: the Cassini Division, the A ring, and the F ring. Let’s take a closer look at the F ring, one of the most turbulent regions in the solar system. The F ring is centered around 140,000 km from Saturn.

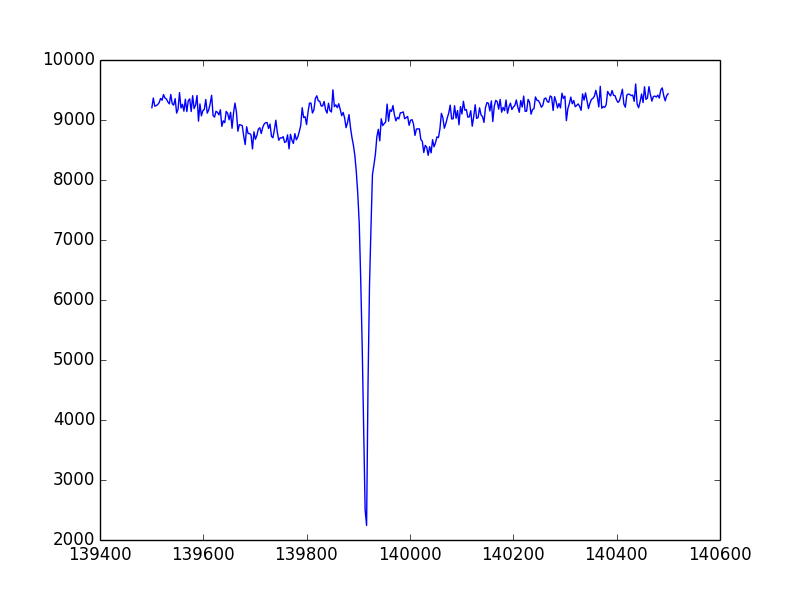
* Plot the region between *r* = 138,000 km and *r* = 142,000 km in your data

This should look something like this:



The dips near the center of the plot represent the core of the F ring. The large peak in the middle, however, is something different. Let’s make a plot of it.

* Plot the region [137000,140000] km

Here’s our strange feature, a clump of material known as a *kitten*:

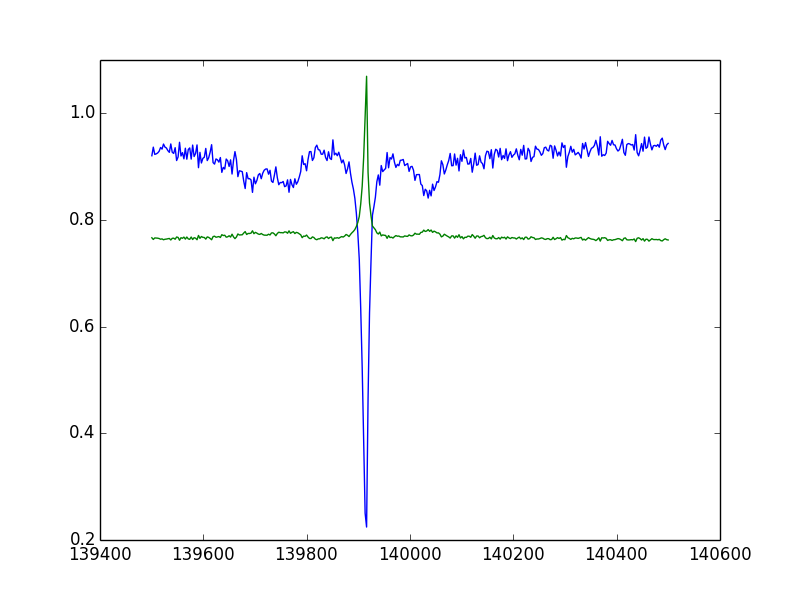
The counts in this plot measure the amount of light that filters through the rings at any given point, but counts aren’t linearly related to density. As a proxy for density, astronomers often use *optical depth*. The data you unpacked from alpleo9.p contained a key corresponding to the optical depth. Store that data in a new variable and add it to your plot. To add more than one line to a plot, call the plot() function multiple times before the show() function.

* Plot the optical depth vs ring radius on the same plot as the counts.

Do you see two lines? If not, print out the first few elements of the data and optical depth to see how similar they are. Not too similar, huh? There are two ways to address this problem, the easy way and the hard way. Like with many things in life, the hard solution is the better one, but for today let’s go with the easy one. Divide the large data by some number so it’s similar in size with the smaller one.

* Replot the two data sets on the same plot

You should get something like this:



This plot is missing a bunch of important stuff, though. At minimum, we need a title and axis labels, so let’s add those first. An axis label is especially vital now that we’ve altered the data. Calling pyplot.title(string) after pyplot.plot() but before pyplot.show() will let you set the title. Use xlabel(*string*) and ylabel(*string*) from pyplot to do the same for the axis labels.

Sometimes it is also useful to label a location on the plot. You can use pyplot.text(x,y,*string*) to place a text label anywhere on the plot. The coordinates x and y are specified in the units on the axes. Find the minimum number of counts in this region and label it on the plot.

Since we have two different lines on our plot, we want to add a legend. You can use the command pyplot.legend(*names*) to produce a legend. The array names should contain strings with what you wish to label each line.

Although matplotlib does an admirable job of automatically choosing the *x* and *y* axis ranges, sometimes we want to alter them. The pyplot functions xlim() and ylim() can do just that. Give your plot a little more room in the vertical dimension and ensure that the horizontal axis only spans the data region.

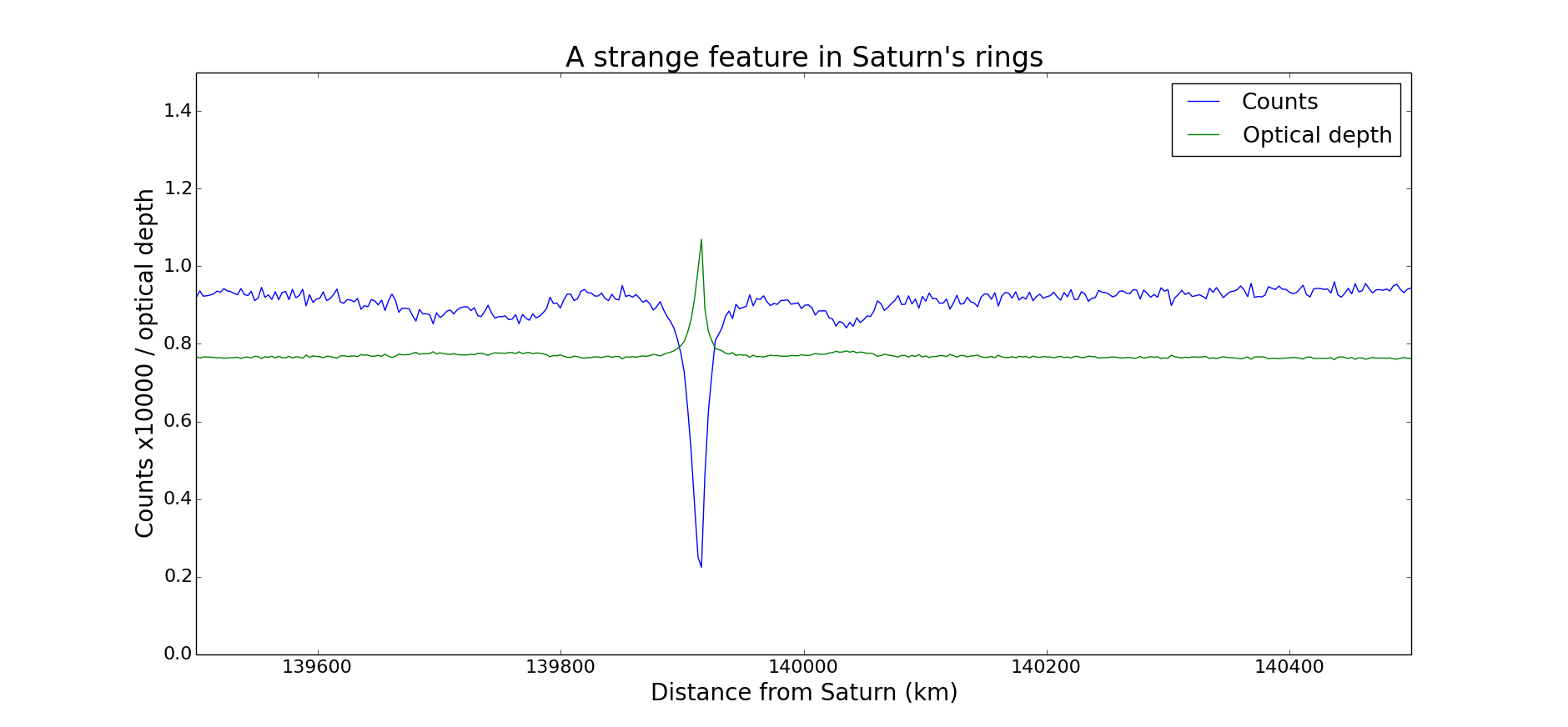
Finally, when you plan to share a plot with others, it is important that it’s large enough. Placing the following commands in front of pyplot.plot() will set the plot to be 20x10 inches with a base font size of 16:

rcParams.update({'font.size': 16})

rcParams.update({'figure.figsize': (20, 10)})

In order to use rcParams, you must import it from matplotlib. In addition, the title(), text(), xlabel(), and ylabel() functions in pyplot accept the parameter size, which allows you to set those specific fonts to be larger still. Make the title size 24, the axis labels size 20, and the text label size 18.

If you’ve made it this far, you should have a plot that looks like this:



Now that we’ve studies the F ring through stellar occultations, let’s take a look at what it looks like in an image. Find the image N1530370686.png, which was captured by the *Cassini* Imaging Science System. We can read this image into Python by using image.imread(*image*). In order to use this, you’ll need to import image from matplotlib. Don’t forget that you need pyplot.show() in order to make the plot visible!

* Display the image

Notice the thin line surrounding the main disk of the ring? That’s the F ring!