Astro Data Science Seminar
Lab 4 - 2016 May 10
Data wrangling, file Input/Output

Ask any programmer, scientist, or data analyst: one of the most frustrating parts of research is dealing with other people's data. Today we'll look at a few helpful tools to read in data, and a couple ways to write it too.

Part 1
Update your Fork (again)

I know this is getting routine, but just incase you've forgotten there are 3 main ways to update your Fork of the class repository:

1. Update it the classic way: https://help.github.com/articles/syncing-a-fork/
2. do a *reverse* Pull Request, thus pulling the latest version from the "upstream master" (aka, the class repo) back to your fork
    - careful to not just submit a Pull Request as normal. Look for the "Try switching the base" link to quickly reverse the Pull Request
3. Brute force, delete your Fork, re-fork, the re-clone on your workstation.
    - While not graceful, this is commonly done...

And once again,
Be sure to "Duplicate" the lab4-example.ipynb file, and make one called lab4-YOURNAME.ipynb

Part 2
Reading Files

Note: these examples are pretty "easy", in that they don't have large portions of missing data, binary data files (e.g. IDL save files), weird structured data (e.g. XML files), or strange formats (e.g. HEALpix). You *will* encounter strange formats. The way to succeed is to beg/borrow/steal solutions to get the data to read.

Tip 1: Microsoft Excel, for all it's shortcomings, is one of the most advance data wrangling environment ever built. There's nothing wrong with opening your datafile in Excel, rearranging it, and then exporting it to something Python can read (e.g. a `.csv` file).

Tip 2: If there are strange values, weird delimiters, missing data, extra spaces... you can sometimes just open files in TextEdit/NotePad and do a search & replace.

2.1: Read a simple text file

- We've done this before, using simple `.csv` files (e.g. Lab 1) with `np.loadtxt`
  - This is an important basic tool
- Now try reading in another simple text file using `np.loadtxt`
  - The file is: `data/2mass_photometry.tbl`
  - It contains data from the 2MASS database
  - Read in columns for RA, Dec, j_m (J mag), and k_m (K mag)
- You'll run in to problems... why? Look at the contents of the file!

2.2: Read a text file using Pandas

- Could teach an entire course on Data Analysis with Pandas...
- This package includes a ton of VERY helpful routines used in data science and data analysis
  - bonus: most are in C, much faster than if written in native Python code.
- make a plot of the infrared color magnitude diagram (J-K, J) for this field
  - remember, flip the Y-axis for magnitudes!
- Bonus: only plot stars within 0.25 deg from the field center
  - Center: RA=132.825deg, Dec=11.80deg
  - This is the famous open cluster M67

2.3: Read an astronomy image (FITS file)

- FITS stands for Flexible Image Transport System, the long standing default in astronomy
- You may see `.fits` and `.fit` files interchangeably
- Need a special reader to parse them. In Python, the standard is part of `astropy`
  - `astropy.io.fits`
- Make a plot of the light curve (time, PDCsap_flux)
  - Can you spot the transiting exoplanet, Kepler 7b?
- Bonus: When was this image taken? Get data from the Header!

2.4: Read a FITS table

- The problem with FITS: it can be either an image or a data file. You don't know until you investigate it.
- FITS tables are very flexible, can contain (almost) any kind of data. Very handy!
- Like images, can contain many extensions with different kinds of data.
- `astropy` makes this simple-ish again!

<u>Part 3</u>
Writing Files

3.1: Save the image of the planetary nebula to a file

- use a `.png` extension.
- Try a `.pdf` extension. Python/Matplotlib is smart and knows how to write many file types

- This is a helpful snippet of code, reading `.fits` images and spitting out `.png` thumbnails. You may need it again someday to quickly look at data!

3.2: Save some data to an output file

- There are a *lot* of ways to save output data
  - What's most important is that you remember what data is in which column. I suggest using headers or named columns when possible! "Pythonic" code (and data) is readable!
- Select stars where their coordinates are within 0.25deg of the field center
  - Center: RA=132.825deg, Dec=11.80deg
- Save JUST these stars to an output text file
  - I suggest CSV for ease of reading.
- I suggest trying to use Pandas for this!
  - make a new DataFrame using only the columns you need
  - select only the stars within 0.25deg of field center
  - use the `.to_csv()` method to output it.