Exercise 1

Simple data processing with Python and Pandas

Prior Knowledge
Unix Command Line Shell
Simple Python

Learning Objectives
First steps with Pandas
Understand the Jupyter Notebook model

Software Requirements
(see separate document for installation of these)

- Python 3.7
- Jupyter notebooks
- (This exercise should also work using Python 3.6)

Ensuring we have the pre-requisite code installed.

1. Before we can use *pandas* we need to make sure we have it installed. You can install it using the graphical user interface **Anaconda Navigator** or you can install from the command-line (terminal window) using **conda** or **pip.** I recommend setting up a new conda environment (where we will store all of the correct versions of libraries needed for this course and which therefore won't cause any compatibility conflicts with other versions you may have installed). If using the command line, create a new python3 environment called bigdata:

   ```
   conda create –n bigdata python=3
   ```

   Activate this environment:

   ```
   conda activate bigdata
   ```

   Install the required libraries and their dependencies

   ```
   conda install pandas
   conda install matplotlib
   conda install jupyter
   ```

   You will see something like this appear in the terminal window.

```
The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    intel-openmp-2020.0        |              166         896 KB
    mkl-service-2.3.0          |   py38hfbe908c_0          46 KB
    mkl_fft-1.0.15             |   py38h5e564d8_0         138 KB
    mkl_random-1.1.0           |   py38h6440ff4_0           5 KB
    numpy-1.18.1               |   py38h7241aed_0           5 KB
    numpy-base-1.18.1          |   py38h6575580_1         3.9 MB
    pandas-1.0.1               |   py38h6c726b0_0         7.9 MB
    python-dateutil-2.8.1      |            py_0         224 KB
    pytz-2019.3                |            py_0         231 KB
    six-1.14.0                 |          py38_0          26 KB
    ------------------------------------------------------------
                                           Total:        13.3 MB

The following NEW packages will be INSTALLED:

  blas               pkgs/main/osx-64::blas-1.0-mkl
  intel-openmp       pkgs/main/osx-64::intel-openmp-2020.0-166
  libgfortran        pkgs/main/osx-64::libgfortran-3.0.1-h93005f0_2
  mkl                pkgs/main/osx-64::mkl-2019.4-233
  mkl-service        pkgs/main/osx-64::mkl-service-2.3.0-py38hfbe908c_0
  mkl_fft            pkgs/main/osx-64::mkl_fft-1.0.15-py38h5e564d8_0
  mkl_random         pkgs/main/osx-64::mkl_random-1.1.0-py38h6440ff4_0
  numpy              pkgs/main/osx-64::numpy-1.18.1-py38h7241aed_0
  numpy-base         pkgs/main/osx-64::numpy-base-1.18.1-py38h6575580_1
  pandas             pkgs/main/osx-64::pandas-1.0.1-py38h6c726b0_0
  python-dateutil    pkgs/main/noarch::python-dateutil-2.8.1-py_0
  pytz               pkgs/main/noarch::pytz-2019.3-py_0
  six                pkgs/main/osx-64::six-1.14.0-py38_0


Proceed ([y]/n)? y
```

Type `y` to proceed. This should ensure that pandas, matplotlib and any dependencies are downloaded and installed in your new environment.

Downloading our sample data

2. Let's make a directory to store our code.

   ```
   mkdir hyg
   cd hyg
   ```

3. Now let's download some star data.
   This data is found at:
   http://www.astronexus.com/hyg

   You can either download the data by going to that website and finding HYG3.0 and downloading into the newly created directory, or you can use a command line and type:

   ```
   wget http://www.astronexus.com/files/downloads/hygdata_v3.csv.gz
   ```

4. Either way that you downloaded it, you now need to uncompress it:

   ```
   gunzip hygdata_v3.csv.gz
   ```

5. Check it's the right size:
   `ls -l ~/hyg`

   You should see:
   `-rw-rw-r-- 1 big big 33449663 Apr 21  2015 hygdata_v3.csv`

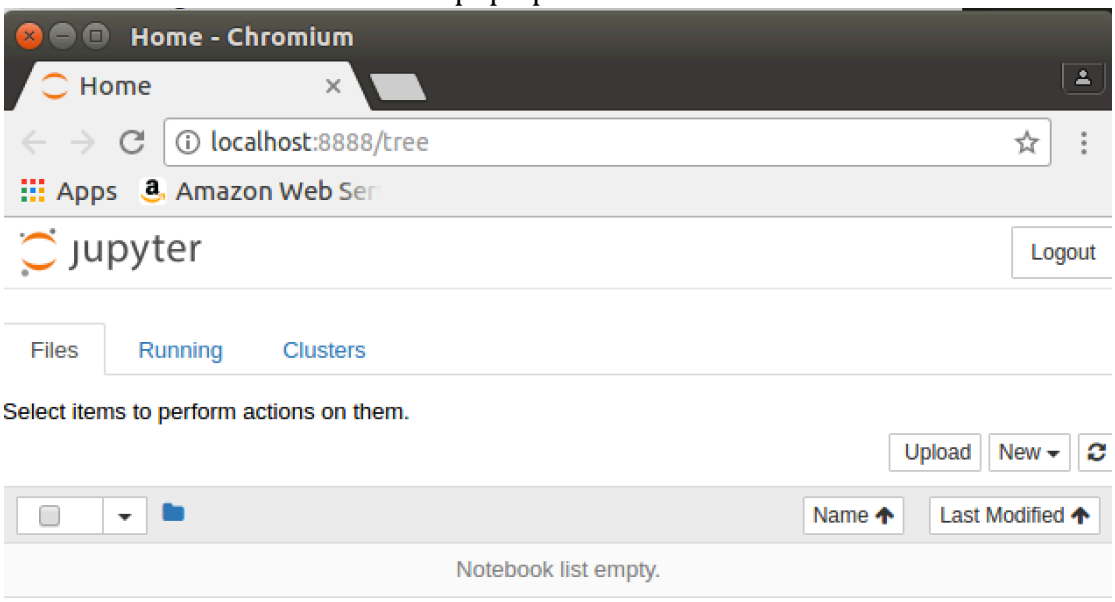
6. To start Jupyter, type (from the same command line that is in the hyg directory):

```
jupyter notebook
```

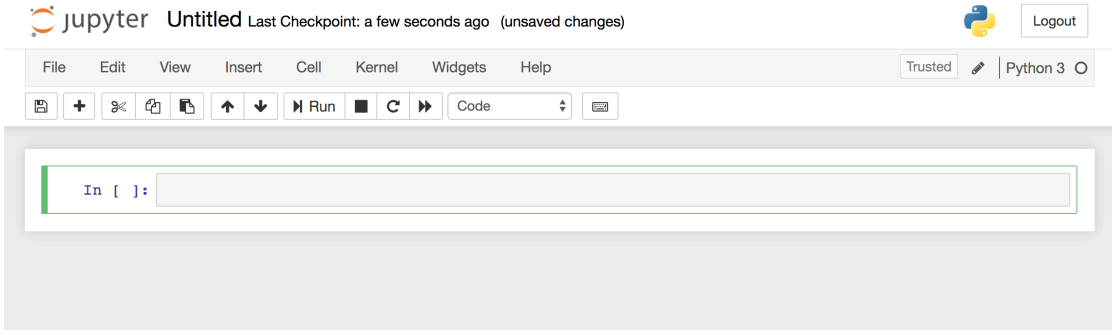
7. In the terminal window you will see

```
[I 13:53:23.865 NotebookApp] Serving notebooks from local
directory: /home/oxclo/pse
[I 13:53:23.866 NotebookApp] 0 active kernels
[I 13:53:23.866 NotebookApp] The Jupyter Notebook is running at:
http://localhost:8888/?token=fd655aab32ed4840ceb47b8b7392b1243a27f5
6350888a91
[I 13:53:23.866 NotebookApp] Use Control-C to stop this server and
shut down all kernels (twice to skip confirmation).
[C 13:53:23.868 NotebookApp]

    Copy/paste this URL into your browser when you connect for the
first time,
    to login with a token:

http://localhost:8888/?token=fd655aab32ed4840ceb47b8b7392b1243a27f5
6350888a91
```

8. And then a browser window will pop up.



9. Use the **New** button to create a new Python3 notebook:



10. Click on the name of the notebook (currently "Untitled") and rename it to **hyg**

11. Now type the following into the **Cell** (next to the words **In [ ]:**)
You don't need to type in the comments but it will help you to remember what each bit does!

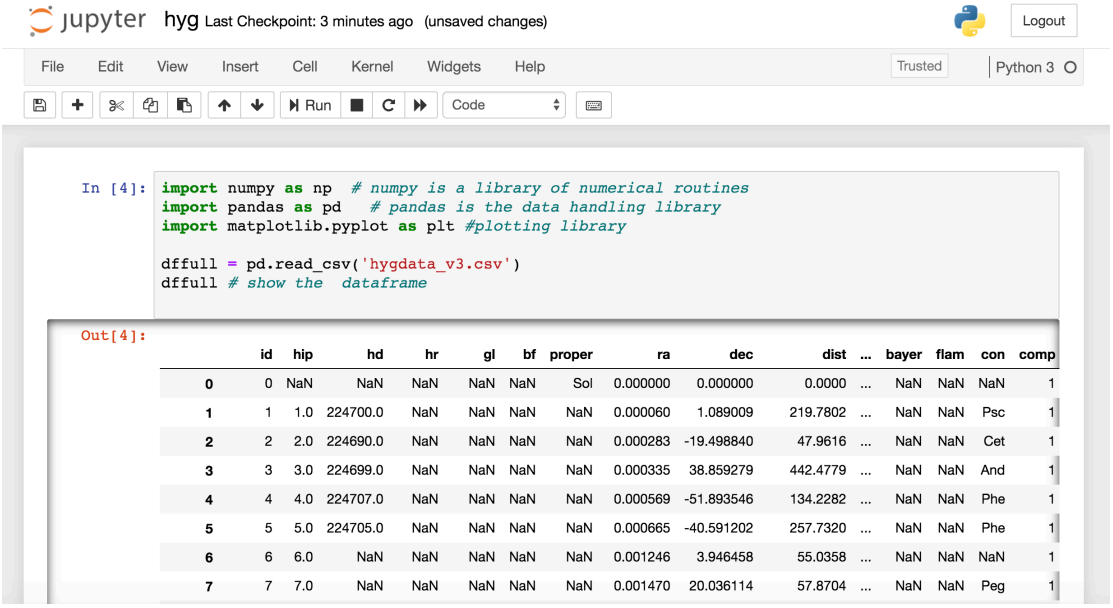
```
import numpy as np  # numpy is a library of numerical routines
import pandas as pd   # pandas is the data handling library
import matplotlib.pyplot as plt #plotting library

dffull = pd.read_csv('hygdata_v3.csv')
dffull # show the dataframe
```

This is creating a DataFrame. This is an object offered by the pandas library that helps deal with tabular data. It is very good at dealing with data that naturally falls into rows and columns and also that has missing elements.

12. Now click on the Run icon 

(or use the useful shortcut ***Shift-Enter*** or ***Ctrl-Enter***)

13. You should see:



	id	hip	hd	hr	gl	bf	proper	ra	dec	dist	...	bayer	flam	con	comp
0	0	NaN	NaN	NaN	NaN	NaN	Sol	0.000000	0.000000	0.0000	...	NaN	NaN	NaN	1
1	1	1.0	224700.0	NaN	NaN	NaN	NaN	0.000060	1.089009	219.7802	...	NaN	NaN	Psc	1
2	2	2.0	224690.0	NaN	NaN	NaN	NaN	0.000283	-19.498840	47.9616	...	NaN	NaN	Cet	1
3	3	3.0	224699.0	NaN	NaN	NaN	NaN	0.000335	38.859279	442.4779	...	NaN	NaN	And	1
4	4	4.0	224707.0	NaN	NaN	NaN	NaN	0.000569	-51.893546	134.2282	...	NaN	NaN	Phe	1
5	5	5.0	224705.0	NaN	NaN	NaN	NaN	0.000665	-40.591202	257.7320	...	NaN	NaN	Phe	1
6	6	6.0	NaN	NaN	NaN	NaN	NaN	0.001246	3.946458	55.0358	...	NaN	NaN	NaN	1
7	7	7.0	NaN	NaN	NaN	NaN	NaN	0.001470	20.036114	57.8704	...	NaN	NaN	Peg	1

14. Scroll down to the bottom of the table and you should see how many rows (stars) are in the catalogue. Note how the notebook automatically knows how to display pandas dataframes in an intelligent manner. Also note that you are not seeing all the rows or columns because there is too much data to display.

You can see the description of the columns here:
https://github.com/astronexus/HYG-Database/blob/master/README.md

15. Before we do any more data processing, let's configure Jupyter to do nice *tab completion*. In a new cell enter:

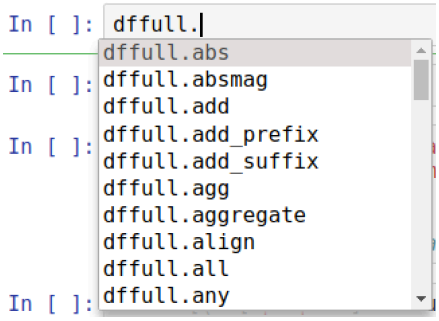
```
%config IPCompleter.greedy=True
```

Anything starting with % is a hint that this is for Jupyter not for Python.

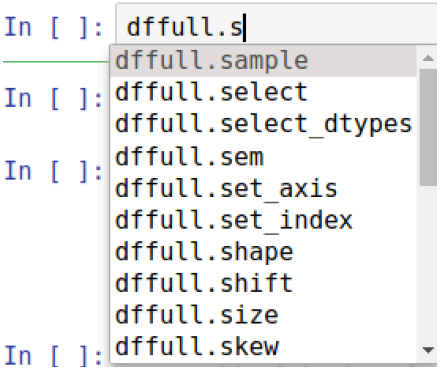
16. You can also just get that information (number of rows and columns) by using the dataframe shape. In the next cell type:

```
dffull.
```

Before typing anything else, hit the Tab key. You should see all possible options for syntax now appear in a little box like this:



Now type 's', and you should see just the operations starting with 's' appear:



Now move down using the down arrow and select 'shape' by hitting Enter.

Now hit **Shift-Enter** (same as Run icon)

You should see:

```
In [2]: dffull.shape
Out[2]: (119614, 37)

In [ ]:
```

Not all the columns are of interest to us. One simple approach is to create a new dataframe that only uses some of the columns from the old dataframe.

To do that, we can use the following syntax:

```
columns = ['id', 'gl', 'mag', 'absmag', 'proper', 'ra', 'dec',
   'dist','con', 'ci','lum']
   df = pd.DataFrame(dffull, columns=columns)
   df # show the resulting dataframe
```

17. Paste or type that into a new cell and execute it.

18. Now, lets identify the stars that have a 'proper' name.

19. There are a couple of ways we could do this. The first, simple one, is to just select that column, and then drop all NaN entries:

    Execute:

    ```
    df['proper'].dropna()
    ```

20. You should see something like:

```
In [21]: df['proper'].dropna()

Out[21]: 0                          Sol
         676                   Alpheratz
         744                        Caph
         1065                    Algenib
         2076                      Ankaa
         3172                     Shedir
         3413                     Diphda
         3759                  96 G. Psc
         3820          Van Maanen's Star
         4417                        Cih
         5436                     Mirach
         6672                    Ruchbah
         7574                   Achernar
         8884                   Sheratan
         9618                     Almaak
         9861                      Hamal
         10800                      Mira
         11734                   Polaris
         12082                 268 G. Cet
         13813                    Acamar
         14100                    Menkar
         14540                     Algol
         15471                  82 G. Eri
```

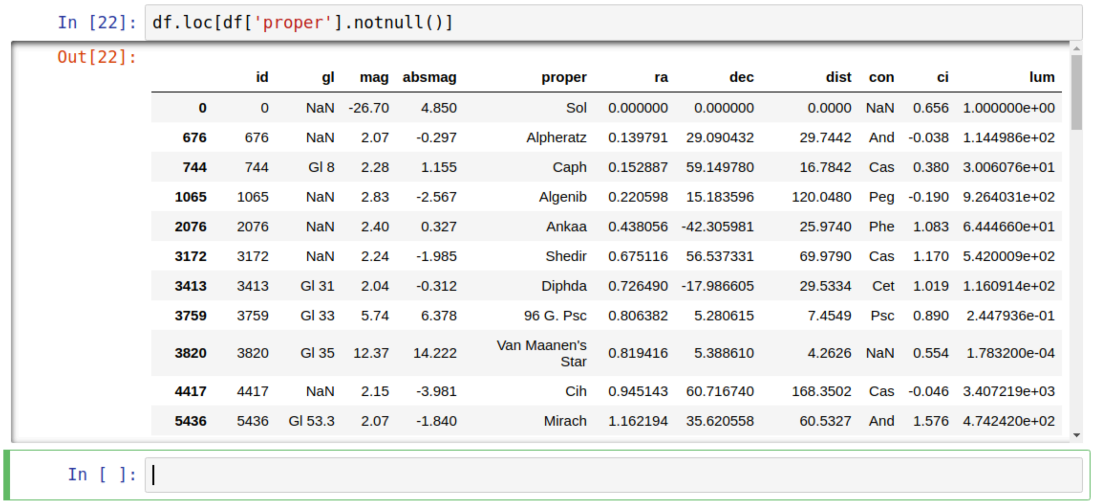
21. Notice that this no longer looks quite the same. This is because this has created a Series object instead of a DataFrame (Each column is effectively a Series, and we've extracted one column).

22. Suppose we want the whole DataFrame (all the columns), but only those with a 'proper' name. We can use a selection function to *locate* the right

rows:

```
df.loc[df['proper'].notnull()]
```

23. You should see:



In [22]:
```
df.loc[df['proper'].notnull()]
```

Out[22]:

	id	gl	mag	absmag	proper	ra	dec	dist	con	ci	lum
0	0	NaN	-26.70	4.850	Sol	0.000000	0.000000	0.0000	NaN	0.656	1.000000e+00
676	676	NaN	2.07	-0.297	Alpheratz	0.139791	29.090432	29.7442	And	-0.038	1.144986e+02
744	744	Gl 8	2.28	1.155	Caph	0.152887	59.149780	16.7842	Cas	0.380	3.006076e+01
1065	1065	NaN	2.83	-2.567	Algenib	0.220598	15.183596	120.0480	Peg	-0.190	9.264031e+02
2076	2076	NaN	2.40	0.327	Ankaa	0.438056	-42.305981	25.9740	Phe	1.083	6.444660e+01
3172	3172	NaN	2.24	-1.985	Shedir	0.675116	56.537331	69.9790	Cas	1.170	5.420009e+02
3413	3413	Gl 31	2.04	-0.312	Diphda	0.726490	-17.986605	29.5334	Cet	1.019	1.160914e+02
3759	3759	Gl 33	5.74	6.378	96 G. Psc	0.806382	5.280615	7.4549	Psc	0.890	2.447936e-01
3820	3820	Gl 35	12.37	14.222	Van Maanen's Star	0.819416	5.388610	4.2626	NaN	0.554	1.783200e-04
4417	4417	NaN	2.15	-3.981	Cih	0.945143	60.716740	168.3502	Cas	-0.046	3.407219e+03
5436	5436	Gl 53.3	2.07	-1.840	Mirach	1.162194	35.620558	60.5327	And	1.576	4.742420e+02

In [ ]:

Note that this is a "view" on the original dataframe `df` and has not actually changed it. In this view, every row is selected which meets the criteria (i.e. that the column *proper* is not null).

24. You can sort the data based on a column using the following syntax, e.g. to identify the stars by distance.

```
df.sort_values('dist', ascending=False)
```

25. If you just want to see the first 10 rows of a DataFrame you can use:

```
df.head(n=10)
```

26. Use those to identify the five furthest "proper named" stars. What do you think of the data?

27. You can select on multiple criteria at once, e.g.:

```
df.loc[(df['proper'].notnull()) & (df['dist']<100000)].sort_values(
  'dist', ascending=False)
```

28. Identify the Gliese catalog identifier of the three least luminescent stars.

Visualisation

29. We can do some simple graphing of the data in Jupyter very easily. MatPlotlib is a simple graphing package for Python. We have already installed it and imported it. This next line tells Jupyter to automatically plot diagrams made by matplotlib directly in the notebook.

```
%matplotlib notebook
```

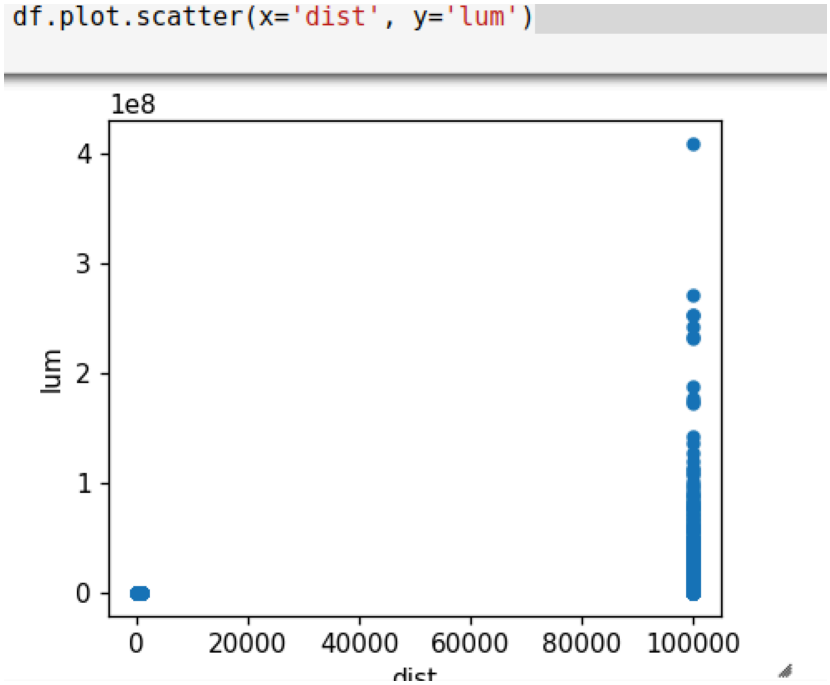
30. Any pandas dataframe or series is automatically plottable by matplotlib (although you may not get anything useful!).

31. Try it:
```
df.plot()
```

32. For something more useful, let's plot a scatter graph of luminosity vs distance:

```
df.plot.scatter(x='dist', y='lum')
```

33. You should see:



34. Once again, it looks like the data is incorrect and therefore not useful (see the comment in the documentation under the distance attribute).

35. Redo the graph this time filtering out any distance >= 100,000.

36. This still isn't much use. Now try making the scales logarithmic by adding the parameters `logx=True, logy=True` to the plot. You may also need to reset the x or y range on the graph e.g.,

```
df_filtered.plot.scatter(x='dist',y='lum',logy=True,
logx=True,xlim=(1,1000))
```

37. Is there anything meaningful about the resulting graph?

38. Extension:

    Explore the data further using the matplotlib to identify any interesting correlations between the data.

39. Before finishing, close the Jupyter browser windows and then stop the Jupyter server by using Ctrl-C on the window, and then **y**.