Python_intro_week4

November 22, 2019

Week 4

This week we will examining how to import and export data, using native python commands, pandas, astropy and sunpy.

We will also take a look at some useful features of pandas, astropy and sunpy that may help you will future analysis.

1 Importing & Exporting data

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Reading and writing data is a common task, whether its from/to an excel table, an image file, etc. Usually you will find it easier to work with high level tools given in the add-on modules (e.g. pandas, astropy), as they have been specifically designed to make the task easier.

However, it is insightful to see how the basic process work.

1.1 Reading in files

To open a file for reading or writing, the basic function is *open*

[2]:
```
file=open('datasets/gas_experiment.csv') # requires path to file
```

The default option is to open the file as read-only.

Here we have created a file object.

We can then iterate over the lines, for example:

[19]:
```
lines=[line.rstrip() for line in file]
print(lines)
```

When a file is opened with *open*, it is important to explicitly close it.

[3]:
```
file.close()
```

An easier way to keep this clean is by using the *with* statement.

In the following example we will use a *csv.reader* to read in our data and also use the *with* statement. When the code exits the block of code associated with the *with* statement, the file is automatically closed.

[3]:
```python
import csv  # import csv module

with open('gas_experiment.csv',newline='') as csvfile:
    experimentResults = csv.reader(csvfile, delimiter=',', quotechar='|')
    for row in experimentResults:
        print(row)
        print(row[1])
```

```
['pressure', ' temperature', ' volume']
 temperature
['84087     ', ' 293', ' 0.001']
 293
['168174', ' 294', ' 0.001']
 294
['252261', ' 295', ' 0.001']
 295
['336348', ' 296', ' 0.001']
 296
['420435', ' 297', ' 0.001']
 297
['504522', ' 298', ' 0.001']
 298
```

1.2 Writing to file

To write results to a csv file you can use the csv package.

A file first has to be created with the open function, however, this time we have to tell the function we want to write to the file.

[24]:
```python
import csv

with open('new_file.csv',mode='w') as nfile:    # mode='w' means write.
    file_writer=csv.writer(nfile,delimiter=',')

    file_writer.writerow(['Name','Age','Height',"Gender"])
    file_writer.writerow(['jeff',21,1.34,'Male'])
```

1.3 Reading in an image

Images can be imported with various modules, such as OpenCV, PIL. However, matplotlib has a function to import PNG images.

[28]:
```
import matplotlib.image as mpimg

img=mpimg.imread('datasets/crab_neb.png')
plt.figure(figsize=(13,13))
plt.imshow(img)
plt.title("Crab Nebula (Hubble Space Telescope)")
plt.show()
```

2 A very brief introduction to pandas

Pandas is an often used tool for data analysis. It contains its own data structures and manipulation tools that streamline the process of working with data. It is especially beneficial if your data is in tabular form. I only give a very basic insight here into the functionality that pandas has to offer, you can find detailed tutorials on-line, e.g. tutorial .

As with *numpy*, we have to import pandas

[3]:
```
import pandas as pd
```

Pandas has two main data types, which are

- series - a one-dimensional array-like object, containing a sequence of values and their data labels (or index)
- data frame - a rectangular table of data containing an ordered collection of columns.

2.1 Data series

Here we will create a basic data series.

[33]:
```
example=pd.Series([1,9,5,70]) # these are the values
print(example)
```

```
0     1
1     9
2     5
3    70
dtype: int64
```

Each observation in the data series comes with a unique identifier. This can be changed if desired.

[26]:
```
example2=pd.Series([1,9,5,70],index=['c','d','f','e'])
print(example2)
print(example2['f'])
```

```
c     1
d     9
f     5
e    70
dtype: int64
5
```

2.2 Data frame

As mentioned, the data frame is a table that contains multiple columns. The columns are the variables of interest and the rows of the table are the observations. Note in the example below that each row has a unique ID (the same as with the series).

The data frame is able to contain mixed data types.

[5]:
```
data={'colour':['blue','blue','red','green','red','blue'],
     'size':[10,30,15,20,50,41],
```

```
        'Voltage':[0.1,0.4,1.,0.5,0.31,0.2]
      }
frame=pd.DataFrame(data)
frame
```

[5]:

```
   Voltage colour  size
0     0.10   blue    10
1     0.40   blue    30
2     1.00    red    15
3     0.50  green    20
4     0.31    red    50
5     0.20   blue    41
```

You can select individual columns by referring to the column title

[6]:

```
frame['colour']
```

[6]:

```
0     blue
1     blue
2      red
3    green
4      red
5     blue
Name: colour, dtype: object
```

A subset of the data frame can be accessed by giving multiple columns in a list.

[9]:

```
frame[['colour','size']]
```

[9]:

```
  colour  size
0   blue    10
1   blue    30
2    red    15
3  green    20
4    red    50
5   blue    41
```

Row access can be achieved by giving the row values via slicing:

[10]:

```
frame[2:4]
```

[10]:

```
   Voltage colour  size
2      1.0    red    15
3      0.5  green    20
```

We can also create subsets of the data with relational operators:

[12]:
```
new_df=frame[ frame['colour']=='blue' ]
new_df
```

[12]:

	Voltage	colour	size
0	0.1	blue	10
1	0.4	blue	30
5	0.2	blue	41

Pandas also works with matplotlib to provide some direct plotting functionality. It is recommened that you read the docs to find out all that is possible.

Here is brief example that counts all the unique enteries in the a particular column of the data frame:

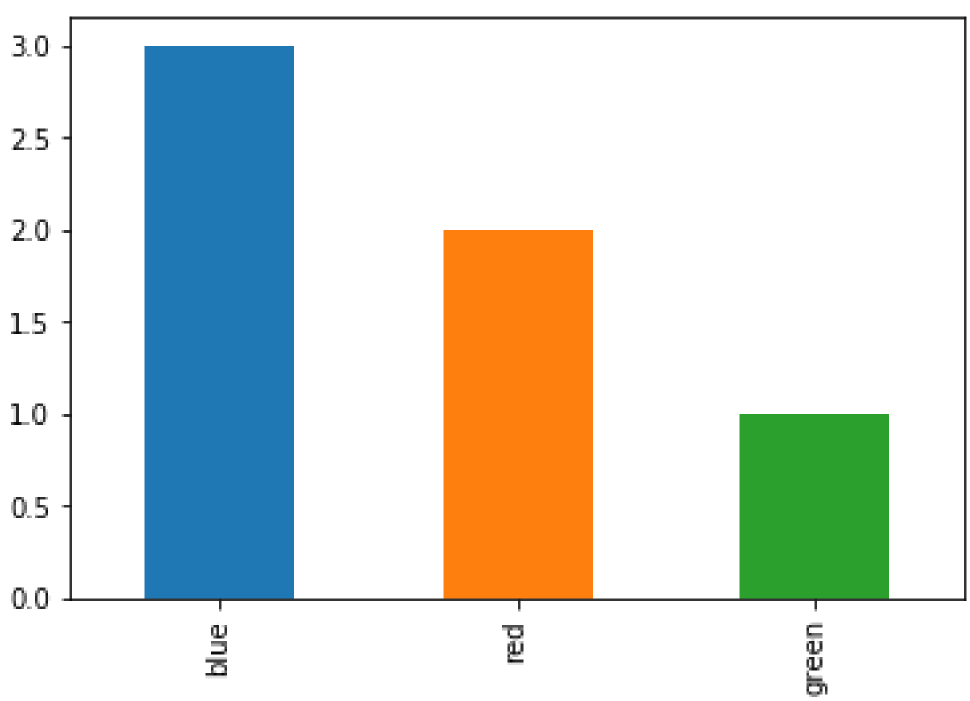
[15]:
```
frame['colour'].value_counts()
```

[15]:
```
blue     3
red      2
green    1
Name: colour, dtype: int64
```

Then adding the plot command, and specifying a bar chart:

[14]:
```
frame['colour'].value_counts().plot(kind='bar')
```

[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x11fda5eb8>
```



2.3 Reading in external data

Often you will not be building the data frame yourself, but reading in data from an external source. To copy this example you will need to download the file *gas_experiments.csv* from Blackboard.

Lets try reading in our csv file with pandas:

[17]:
```
df=pd.read_csv('datasets/gas_experiment.csv') #Location of file on your computer
df
```

[17]:

	pressure	temperature	volume
0	84087	293.0	0.001
1	168174	294.0	0.001
2	252261	295.0	0.001
3	336348	296.0	0.001
4	420435	297.0	0.001
5	504522	298.0	0.001
6	600000	NaN	0.001

Easy enough!

If you open up the file in a text editor or excel, you will see it is quite simple, i.e., only contains column headers and the column data. You may have more complicated situations where the file contains additional unwanted information at the beginning or end. Using a few extra keywords in *read_csv* you can read in a clean table.

You may have noticed that the last value in the temperature column is given as NaN. This means that there was no data in this column for measurement 7, it's what is known as a missing value (for obvious reasons!).

Missing values are common, especially in large data sets. Before we can do any analysis on the data, we have to remove or replace missing values. We will just remove them.

If there are a small number of values, you can remove them with the *drop* method.

[19]:
```
new_df=df.drop(6)
new_df
```

[19]:

	pressure	temperature	volume
0	84087	293.0	0.001
1	168174	294.0	0.001
2	252261	295.0	0.001
3	336348	296.0	0.001
4	420435	297.0	0.001
5	504522	298.0	0.001

However, for a larger table you might want to use *dropna*:

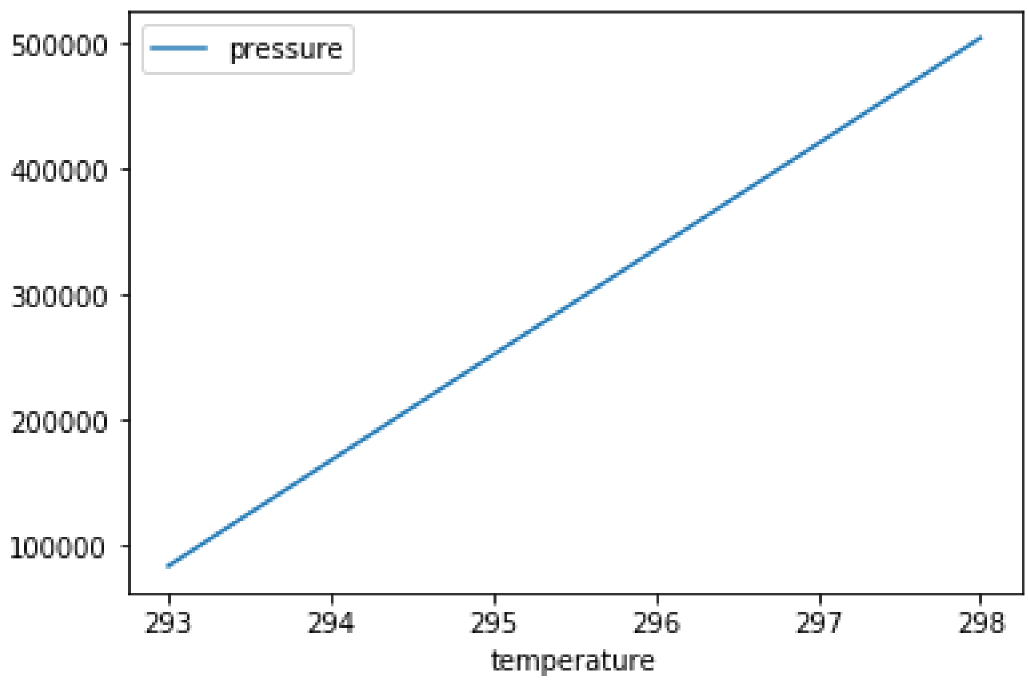
[21]:
```
new_df=df.dropna()
new_df
```

[21]:

	pressure	temperature	volume
0	84087	293.0	0.001
1	168174	294.0	0.001
2	252261	295.0	0.001
3	336348	296.0	0.001
4	420435	297.0	0.001
5	504522	298.0	0.001

[22]:
```
new_df.plot('temperature','pressure')
```

[22]: <matplotlib.axes._subplots.AxesSubplot at 0x11ffcefd0>



Note to self. To read files from google sheets:

Copy the URL from the Address Bar:

google_sheet_url = 'https://docs.google.com/spreadsheets/d/19nK-I3FIgLLCK9XHSKNOPYknuq4b8-qnAuAyKUegoNQ/edit#gid=280140380'

Replace "edit#gid" text in the google_sheet_url variable above with "export?format=csv&gid"

3 Astropy

Astropy is a package that provides tool and functionality for performing common astronomical tasks in Python.

As with all packages/modules introduced in this course, we will only cover a few features. A detailed description of astropy and the range of functions can be found at: http://docs.astropy.org/en/stable/index.html

3.1 Reading in fits files

Astronomical data is typically stored in *fits* files. In order to read them into Python we need to load part of the *astropy* package, which contains the necessary functions.

To load the *astropy.io* module

[6]:
```
from astropy.io import fits
```

The first step is to define where the data file is, so Python knows where to find it:

[ ]:
```
file='sun_data/....fits'
```

We then call a function to open the file:

[ ]:
```
hdul=fits.open(file)
```

What this command does is open an HDU (Header Data Unit). This is the highest level component of the FITS file structure, consisting of a header and (typically) a data array or table.

We can explore the HDU with an actual fits file. I have based the following loosely on the astropy tutorial.

First, let us download a fits file:

[7]:
```
from astropy.utils.data import download_file
image_file = download_file('http://data.astropy.org/tutorials/FITS-images/
↪HorseHead.fits', cache=True )
```

We can now open the fits file and use the *info* method to see what the file contains.

[8]:
```
hdu_list = fits.open(image_file)
hdu_list.info()
```

```
Filename: /Users/richardmorton/.astropy/cache/download/py3/2c9202ae878ecfcb60878
ceb63837f5f
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     161   (891, 893)   int16
  1  er.mask       1 TableHDU        25   1600R x 4C   [F6.2, F6.2, F6.2, F6.2]
```

We see that the file contains two extensions, the primary extension indexed as 0 and another which we are told is a table.

For the Primary extension, we can see by looking at the *Dimensions* and *Format* that we are dealing with an image.

The cards column gives use the number of tags in the fits header.

The fits header contains a standard set of information that describes the observation of the data and also provides information about the data product.

[15]:
```
hdu_list[0].header
```

[15]:
```
SIMPLE  =                    T /FITS: Compliance
BITPIX  =                   16 /FITS: I*2 Data
NAXIS   =                    2 /FITS: 2-D Image Data
NAXIS1  =                  891 /FITS: X Dimension
NAXIS2  =                  893 /FITS: Y Dimension
EXTEND  =                    T /FITS: File can contain extensions
DATE    = '2014-01-09        '  /FITS: Creation Date
ORIGIN  = 'STScI/MAST'         /GSSS: STScI Digitized Sky Survey
SURVEY  = 'SERC-ER '           /GSSS: Sky Survey
REGION  = 'ER768   '           /GSSS: Region Name
PLATEID = 'A0JP    '           /GSSS: Plate ID
SCANNUM = '01      '           /GSSS: Scan Number
DSCNDNUM= '00      '           /GSSS: Descendant Number
TELESCID=                    4 /GSSS: Telescope ID
BANDPASS=                   36 /GSSS: Bandpass Code
COPYRGHT= 'AAO/ROE '           /GSSS: Copyright Holder
SITELAT =              -31.277 /Observatory: Latitude
SITELONG=              210.934 /Observatory: Longitude
TELESCOP= 'UK Schmidt - Doubl' /Observatory: Telescope
INSTRUME= 'Photographic Plate' /Detector: Photographic Plate
EMULSION= 'IIIaF   '           /Detector: Emulsion
FILTER  = 'OG590   '           /Detector: Filter
PLTSCALE=                67.20 /Detector: Plate Scale arcsec per mm
PLTSIZEX=              355.000 /Detector: Plate X Dimension mm
PLTSIZEY=              355.000 /Detector: Plate Y Dimension mm
PLATERA =        85.5994550000 /Observation: Field centre RA degrees
PLATEDEC=       -4.94660910000 /Observation: Field centre Dec degrees
PLTLABEL= 'OR14052 '           /Observation: Plate Label
DATE-OBS= '1990-12-22T13:49:00' /Observation: Date/Time
EXPOSURE=                 65.0 /Observation: Exposure Minutes
PLTGRADE= 'AD2     '           /Observation: Plate Grade
OBSHA   =             0.158333 /Observation: Hour Angle
OBSZD   =              26.3715 /Observation: Zenith Distance
AIRMASS =              1.11587 /Observation: Airmass
REFBETA =        66.3196420000 /Observation: Refraction Coeff
REFBETAP=     -0.0820000000000 /Observation: Refraction Coeff
REFK1   =        6423.52290000 /Observation: Refraction Coeff
REFK2   =       -102122.550000 /Observation: Refraction Coeff
```

```
CNPIX1  =                12237 /Scan: X Corner
CNPIX2  =                19965 /Scan: Y Corner
XPIXELS =                23040 /Scan: X Dimension
YPIXELS =                23040 /Scan: Y Dimension
XPIXELSZ=              15.0295 /Scan: Pixel Size microns
YPIXELSZ=              15.0000 /Scan: Pixel Size microns
PPO1    =       -3069417.00000 /Scan: Orientation Coeff
PPO2    =       0.000000000000 /Scan: Orientation Coeff
PPO3    =        177500.000000 /Scan: Orientation Coeff
PPO4    =       0.000000000000 /Scan: Orientation Coeff
PPO5    =        3069417.00000 /Scan: Orientation Coeff
PPO6    =        177500.000000 /Scan: Orientation Coeff
PLTRAH  =                    5 /Astrometry: Plate Centre H
PLTRAM  =                   42 /Astrometry: Plate Centre M
PLTRAS  =                23.86 /Astrometry: Plate Centre S
PLTDECSN= '-       '           /Astrometry: Plate Centre +/-
PLTDECD =                    4 /Astrometry: Plate Centre D
PLTDECM =                   56 /Astrometry: Plate Centre M
PLTDECS =                 47.9 /Astrometry: Plate Centre S
EQUINOX =               2000.0 /Astrometry: Equinox
AMDX1   =        67.1550859799 /Astrometry: GSC1 Coeff
AMDX2   =      0.0431478884485 /Astrometry: GSC1 Coeff
AMDX3   =       -292.435619180 /Astrometry: GSC1 Coeff
AMDX4   =  -2.68934864702E-005 /Astrometry: GSC1 Coeff
AMDX5   =   1.99133423290E-005 /Astrometry: GSC1 Coeff
AMDX6   =  -2.37011931379E-006 /Astrometry: GSC1 Coeff
AMDX7   =       0.000000000000 /Astrometry: GSC1 Coeff
AMDX8   =   2.21426387429E-006 /Astrometry: GSC1 Coeff
AMDX9   =  -8.12841581455E-008 /Astrometry: GSC1 Coeff
AMDX10  =   2.48169090021E-006 /Astrometry: GSC1 Coeff
AMDX11  =   2.77618933926E-008 /Astrometry: GSC1 Coeff
AMDX12  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDX13  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDX14  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDX15  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDX16  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDX17  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDX18  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDX19  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDX20  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDY1   =        67.1593591466 /Astrometry: GSC1 Coeff
AMDY2   =     -0.0471363749174 /Astrometry: GSC1 Coeff
AMDY3   =        316.004963520 /Astrometry: GSC1 Coeff
AMDY4   =   2.86798151430E-005 /Astrometry: GSC1 Coeff
AMDY5   =  -2.00968236347E-005 /Astrometry: GSC1 Coeff
AMDY6   =   2.27840393227E-005 /Astrometry: GSC1 Coeff
AMDY7   =       0.000000000000 /Astrometry: GSC1 Coeff
```

```
AMDY8   =   2.23885090381E-006 /Astrometry: GSC1 Coeff
AMDY9   =  -2.28360163464E-008 /Astrometry: GSC1 Coeff
AMDY10  =   2.44828851495E-006 /Astrometry: GSC1 Coeff
AMDY11  =  -5.76717487998E-008 /Astrometry: GSC1 Coeff
AMDY12  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDY13  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDY14  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDY15  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDY16  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDY17  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDY18  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDY19  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDY20  =       0.000000000000 /Astrometry: GSC1 Coeff
AMDREX1 =        67.1532034737 /Astrometry: GSC2 Coeff
AMDREX2 =      0.0434354199559 /Astrometry: GSC2 Coeff
AMDREX3 =       -292.435438892 /Astrometry: GSC2 Coeff
AMDREX4 =   4.60919247070E-006 /Astrometry: GSC2 Coeff
AMDREX5 =  -3.21138058537E-006 /Astrometry: GSC2 Coeff
AMDREX6 =   7.23651736725E-006 /Astrometry: GSC2 Coeff
AMDREX7 =       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX8 =       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX9 =       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX10=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX11=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX12=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX13=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX14=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX15=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX16=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX17=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX18=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX19=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREX20=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY1 =        67.1522589487 /Astrometry: GSC2 Coeff
AMDREY2 =     -0.0481758265285 /Astrometry: GSC2 Coeff
AMDREY3 =        315.995683716 /Astrometry: GSC2 Coeff
AMDREY4 =  -7.47397531230E-006 /Astrometry: GSC2 Coeff
AMDREY5 =   9.55221105409E-007 /Astrometry: GSC2 Coeff
AMDREY6 =   7.60954485251E-006 /Astrometry: GSC2 Coeff
AMDREY7 =       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY8 =       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY9 =       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY10=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY11=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY12=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY13=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY14=       0.000000000000 /Astrometry: GSC2 Coeff
```

```
AMDREY15=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY16=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY17=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY18=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY19=       0.000000000000 /Astrometry: GSC2 Coeff
AMDREY20=       0.000000000000 /Astrometry: GSC2 Coeff
ASTRMASK= 'er.mask '           /Astrometry: GSC2 Mask
WCSAXES =                    2 /GetImage: Number WCS axes
WCSNAME = 'DSS                ' /GetImage: Local WCS approximation from full plat
RADESYS = 'ICRS               ' /GetImage: GSC-II calibration using ICRS system
CTYPE1  = 'RA---TAN           ' /GetImage: RA-Gnomic projection
CRPIX1  =           446.000000 /GetImage: X reference pixel
CRVAL1  =            85.274970 /GetImage: RA of reference pixel
CUNIT1  = 'deg                ' /GetImage: degrees
CTYPE2  = 'DEC--TAN           ' /GetImage: Dec-Gnomic projection
CRPIX2  =           447.000000 /GetImage: Y reference pixel
CRVAL2  =            -2.458265 /GetImage: Dec of reference pixel
CUNIT2  = 'deg                ' /Getimage: degrees
CD1_1   =        -0.0002802651 /GetImage: rotation matrix coefficient
CD1_2   =         0.0000003159 /GetImage: rotation matrix coefficient
CD2_1   =         0.0000002767 /GetImage: rotation matrix coefficient
CD2_2   =         0.0002798187 /GetImage: rotation matrix coefficient
OBJECT  = 'data               ' /GetImage: Requested Object Name
DATAMIN =                 3759 /GetImage: Minimum returned pixel value
DATAMAX =                22918 /GetImage: Maximum returned pixel value
OBJCTRA = '05 41 06.000       ' /GetImage: Requested Right Ascension (J2000)
OBJCTDEC= '-02 27 30.00       ' /GetImage: Requested Declination (J2000)
OBJCTX  =             12682.48 /GetImage: Requested X on plate (pixels)
OBJCTY  =             20411.37 /GetImage: Requested Y on plate (pixels)
```

The header can be accessed like a dictionary, in the sense it has keys and values. For example, we can find the date that the image was taken.

[16]:
```
head=hdu_list[0].header
head['DATE']
```

[16]:
```
'2014-01-09'
```

The data in the extension is accessed with .data:

[18]:
```
imageData=hdu_list[0].data
```

The data can be found to be returned as a *numpy* array. And looking at the shape of the data, we can see it corresponds to the information given with *.info* .

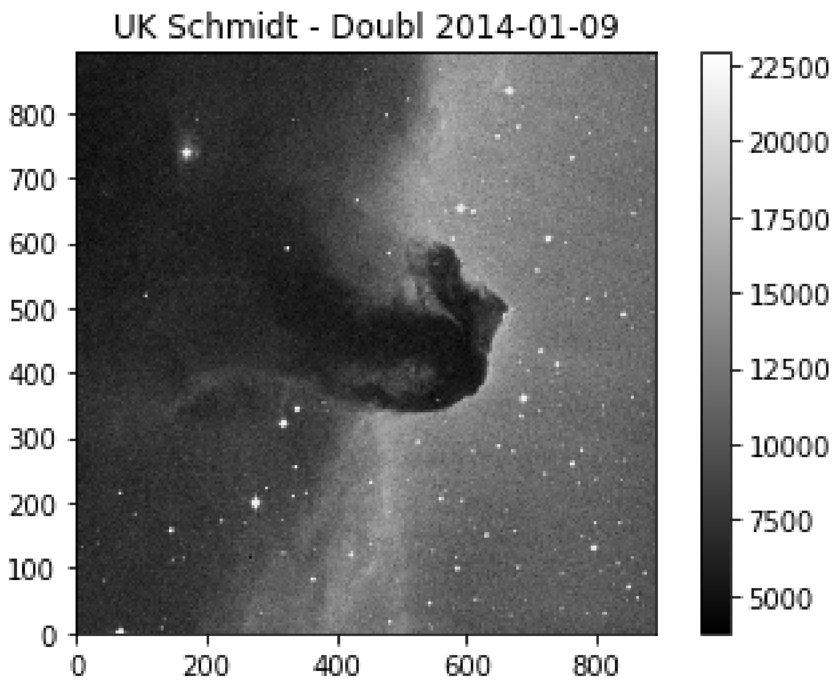
[22]:
```
print(type(imageData))
print(imageData.shape)
```

```
<class 'numpy.ndarray'>
(893, 891)
```

This means we can display the data simply, e.g. using imshow.

[29]:
```
plt.imshow(imageData,cmap='gray',origin='lower')
plt.title(head['telescop']+' '+head['date'])
plt.colorbar()
```

[29]:
```
<matplotlib.colorbar.Colorbar at 0x128617828>
```



3.2 Astropy units & constants

Astropy has functionality such that numbers can be associated with units, which means you can create a calculation that also outputs the correct units! Not only does this provide you with the relevant units, it also provides a mechanism to check you have implemented your calculation correctly - wrong units means a mistake.

The astropy units can be found by loading in the units module:

[1]:
```
from astropy import units as u
```

The units can then be applied to a range of python objects:

[6]:
```
print(15*u.meter)
print(np.arange(10)*u.second)
print(type(15*u.meter))
```

```
15.0 m
[0. 1. 2. 3. 4. 5. 6. 7. 8. 9.] s
<class 'astropy.units.quantity.Quantity'>
```

You can see that when we check the type of the value and unit, it forms a *quantity* object.

You can assign the quantity to a variable and access methods and attributes of quantity. For example, you can get back the value or the unit:

[7]:
```
dist=23*u.meter
print(dist.value)
print(dist.unit)
```

```
23.0
m
```

You can also change between SI and CGS units:

[8]:
```
print(dist.cgs)
```

```
2300.0 cm
```

As mentioned, you can do something more practical, like undertaking calculations:

[10]:
```
time=0.5*u.second
speed =dist/time
print(speed)
print(speed.to(u.kilometer/u.hour))  # convert to another system of units with␣
 ↪the .to method
```

```
46.0 m / s
165.6 km / h
```

As well as units, astropy also contains constants, which are potentially useful for your calculations. To get access to the constants you need to import the constants module.

[12]:
```
from astropy import constants as const

print(const.G)
```

```
  Name   = Gravitational constant
  Value  = 6.67408e-11
  Uncertainty  = 3.1e-15
  Unit  = m3 / (kg s2)
  Reference = CODATA 2014
```

[12]:
```
astropy.constants.codata2014.CODATA2014
```

For example, you might want to calculate the gravitional potential energy:

$$U = -\frac{GM_{\odot}m}{r}$$

for a human orbiting the Sun at 3 AU.

[16]:
```
mass_human=70*u.kilogram
U=-const.G*const.M_sun*mass_human/(3*u.AU)
print(U)
print(U.decompose()) # decompose to irreducible units
```

```
-3.0966236e+21 kg m3 / (AU s2)
-20699650239.07255 kg m2 / s2
```

4 Sunpy

Sunpy is solar specific package for downloading and manipulating data from solar space missions. However, I highlight that it isn't yet capable of performing all the neccessary calibration steps for certain instruments (e.g. NASA's Solar Dynamic Observatory).

The NASA recognised software is currently only distributed via solarsoft, which is IDL specific (IDL is currently the default language for solar data analysis - although things are slowly migrating to Python).

However, we can still access the most basic data product (typically called Level 1).

Before we get into the data, it is worthwhile providing an overview of the Solar Dynamic Observatory, just so we know what the instrument is and what is does.

4.1 Solar Dynamic Observatory

The Solar Dynamic Observatory (SDO) is a NASA mission which has been observing the Sun since 2010. It is made up of three instruments: * Helioseismic and Magnetic Imager (HMI) * Extreme Ultraviolet Variability Experiment (EVE) * Atmospheric Imaging Assembly (AIA)

HMI is an instrument designed to study oscillations and the magnetic field at the solar surface, or photosphere. HMI is one of three instruments on the Solar Dynamics Observatory; together, the suite of instruments observes the Sun nearly continuously and takes a terabyte of data a day. HMI observes the full solar disk at 6173 Å with a resolution of 1 arcsecond. (Description from `http://hmi.stanford.edu`).

The Atmospheric Imaging Assembly (AIA) for the SDO is designed to provide an unprecedented view of the solar corona, taking images that span at least 1.3 solar diameters in multiple wavelengths nearly simultaneously, at a resolution of about 1 arcsec and at a cadence of 10 seconds or better. The primary goal of the AIA Science Investigation is to use these data, together with data from other SDO instruments and from other observatories, to significantly improve our understanding of the physics behind the activity displayed by the Sun's atmosphere, which drives space weather in the heliosphere and in planetary environments. (Description from `https://aia.lmsal.com`).

We will not discuss EVE.

Needless to say, the data from HMI and AIA are very different and can be used in isolation or in unison, depending upon the goal of your study.

The first step is the learn how to access and view this data.

4.2 Downloading data

The data for NASA missions is stored at the Joint Science Operation Centre (JSOC) and can also be found on the Virtual Solar Observatory (VSO).

So, we use a tool in order to query these data bases - drms.

The first steps are to import drms and instantiate the drms client.

[40]:
```
import drms
c=drms.Client()
```

Data series can be accessed with the *series* method, with HMI series names containing "hmi" and AIA containing 'aia'

[39]:
```
c.series(r'aia')
```

[39]:
```
['aia.flatfield',
 'aia.lev1',
 'aia.lev1_euv_12s',
 'aia.lev1_uv_24s',
 'aia.lev1_vis_1h',
 'aia.master_pointing3h',
 'aia.response',
 'aia.temperature_summary_300s',
 'aia_test.lev1_12s4arc',
 'aia_test.master_pointing3h']
```

[41]:
```
c.series(r'hmi')
```

[41]:
```
['hmi.B_720s',
 'hmi.B_720s_dcon',
 'hmi.B_720s_dconS',
 'hmi.Bharp_720s',
 'hmi.Bharp_720s_nrt',
 'hmi.Ic_45s',
 'hmi.Ic_45s_dcon',
 'hmi.Ic_720s',
 'hmi.Ic_720s_dcon',
 'hmi.Ic_720s_dconS',
 'hmi.Ic_noLimbDark_720s',
 'hmi.Ld_45s_dcon',
```

```
'hmi.Ld_720s',
'hmi.Ld_720s_dcon',
'hmi.Ld_720s_dconS',
'hmi.Lw_45s',
'hmi.Lw_45s_dcon',
'hmi.Lw_720s',
'hmi.Lw_720s_dcon',
'hmi.Lw_720s_dconS',
'hmi.ME_720s_fd10',
'hmi.ME_720s_fd10_dcon',
'hmi.ME_720s_fd10_nrt',
'hmi.MEharp_720s',
'hmi.MEharp_720s_nrt',
'hmi.MHDcorona_daily_nrt',
'hmi.M_45s',
'hmi.M_45s_dcon',
'hmi.M_720s',
'hmi.M_720s_dcon',
'hmi.M_720s_dconS',
'hmi.Mharp_720s',
'hmi.Mharp_720s_nrt',
'hmi.Mrmap_latlon_720s',
'hmi.Mrmap_latlon_720s_nrt',
'hmi.Mrmap_lowres_latlon_720s',
'hmi.S_720s',
'hmi.S_720s_dcon',
'hmi.S_720s_dconS',
'hmi.TDKernels',
'hmi.V_45s',
'hmi.V_45s_dcon',
'hmi.V_720s',
'hmi.V_720s_dcon',
'hmi.V_720s_dconS',
'hmi.V_avg120',
'hmi.V_sht_2drls',
'hmi.V_sht_2drls_asym',
'hmi.V_sht_gf_gaps_retile',
'hmi.V_sht_gf_retile',
'hmi.V_sht_modes_asym',
'hmi.V_sht_modes_asym_archive',
'hmi.V_sht_pow',
'hmi.b_135s',
'hmi.b_720s_e15w1332_cea',
'hmi.b_720s_e15w1332_cutout',
'hmi.b_90s',
'hmi.b_synoptic',
'hmi.b_synoptic_small',
```

```
'hmi.bmap_lowres_latlon_720s',
'hmi.c_avg120',
'hmi.coefficients',
'hmi.eigenfunctions',
'hmi.flatfield',
'hmi.fsVbinned_nrt',
'hmi.fsi_phase_lon_lat',
'hmi.fsi_phase_lon_lat_5d',
'hmi.gcvbinned_nrt',
'hmi.hskernels',
'hmi.ic_nolimbdark_720s_nrt',
'hmi.ld_45s',
'hmi.leakage',
'hmi.lev1_cal',
'hmi.lev1_dcon',
'hmi.lookup_ChebyCoef_BNoise',
'hmi.lookup_corrected_expanded',
'hmi.lookup_expanded',
'hmi.m_720s_mod',
'hmi.m_720s_nrt',
'hmi.marmask_720s',
'hmi.marmask_720s_nrt',
'hmi.me_135s',
'hmi.me_720s_e15w1332',
'hmi.me_720s_e15w1332_harp',
'hmi.me_720s_fd10_harp',
'hmi.me_720s_fd10_harp_nrt',
'hmi.me_90s',
'hmi.meanpf_720s',
'hmi.mhdcorona',
'hmi.mhdcorona_daily',
'hmi.mldailysynframe_720s',
'hmi.mldailysynframe_720s_nrt',
'hmi.mldailysynframe_small_720s',
'hmi.mldailysynframe_small_720s_nrt',
'hmi.mlsynop_small_720s',
'hmi.mrdailysynframe_720s',
'hmi.mrdailysynframe_720s_nrt',
'hmi.mrdailysynframe_polfil_720s',
'hmi.mrdailysynframe_small_720s',
'hmi.mrdailysynframe_small_720s_nrt',
'hmi.mrsynop_small_720s',
'hmi.offpoint_flatfield',
'hmi.pfss_synframe',
'hmi.pfss_synop',
'hmi.polar_db',
'hmi.q_synframe',
```

```
'hmi.q_synop',
'hmi.rdMAI_fd05',
'hmi.rdMAI_fd15',
'hmi.rdMAI_fd30',
'hmi.rdVfitsc_fd05',
'hmi.rdVfitsc_fd15',
'hmi.rdVfitsc_fd30',
'hmi.rdVfitsf_fd05',
'hmi.rdVfitsf_fd15',
'hmi.rdVfitsf_fd30',
'hmi.rdVpspec_fd05',
'hmi.rdVpspec_fd15',
'hmi.rdVpspec_fd30',
'hmi.rdVtrack_fd05',
'hmi.rdVtrack_fd15',
'hmi.rdVtrack_fd30',
'hmi.rdvavgpspec_fd15',
'hmi.rdvavgpspec_fd30',
'hmi.rdvflows_fd15_frame',
'hmi.rdvflows_fd30_frame',
'hmi.s_135s',
'hmi.s_90s',
'hmi.sharp_720s',
'hmi.sharp_720s_nrt',
'hmi.sharp_cea_720s',
'hmi.sharp_cea_720s_nrt',
'hmi.synoptic_ml_720s',
'hmi.synoptic_ml_720s_nrt',
'hmi.synoptic_ml_small_720s_nrt',
'hmi.synoptic_mr_720s',
'hmi.synoptic_mr_720s_nrt',
'hmi.synoptic_mr_polfil_720s',
'hmi.synoptic_mr_small_720s_nrt',
'hmi.tdVinvrt_synopHC',
'hmi.tdVtimes_synopHC',
'hmi.tdVtrack_synopHC',
'hmi.tdpixlist',
'hmi.temperature_summary_300s',
'hmi.v_sht_72d',
'hmi.v_sht_gaps_72d',
'hmi.v_sht_gf_72d',
'hmi.v_sht_gf_gaps_72d',
'hmi.v_sht_modes',
'hmi.v_sht_modes_archive',
'hmi.v_sht_secs_72d',
'hmi.vw_V_sht_2drls',
'hmi.vw_V_sht_72d',
```

```
 'hmi.vw_V_sht_gaps_72d',
 'hmi.vw_V_sht_gf_72d',
 'hmi.vw_V_sht_gf_gaps_72d',
 'hmi.vw_V_sht_modes',
 'hmi.vw_V_sht_modes_archive',
 'hmi.vw_V_sht_pow',
 'hmi.vw_v_45s',
 'hmi_test.qmap_test',
 'su_phil.hmi_M_1k_720s',
 'su_phil.hmi_M_remap_720s']
```

These can be filtered by using a portion of the string:

[44]:
```
c.series(r'hmi\.v_')
```

[44]:
```
['hmi.V_45s',
 'hmi.V_45s_dcon',
 'hmi.V_720s',
 'hmi.V_720s_dcon',
 'hmi.V_720s_dconS',
 'hmi.V_avg120',
 'hmi.V_sht_2drls',
 'hmi.V_sht_2drls_asym',
 'hmi.V_sht_gf_gaps_retile',
 'hmi.V_sht_gf_retile',
 'hmi.V_sht_modes_asym',
 'hmi.V_sht_modes_asym_archive',
 'hmi.V_sht_pow',
 'hmi.v_sht_72d',
 'hmi.v_sht_gaps_72d',
 'hmi.v_sht_gf_72d',
 'hmi.v_sht_gf_gaps_72d',
 'hmi.v_sht_modes',
 'hmi.v_sht_modes_archive',
 'hmi.v_sht_secs_72d']
```

DRMS records can be searched by creating a query. To do this, you need the series name and specific primekey's. To find the primekeys for each series, you can do the following:

[48]:
```
c.pkeys('hmi.m_45s')
```

[48]:
```
['T_REC', 'CAMERA']
```

You can also use the info method to find our more information about the series:

[53]:
```
ser_info=c.info('hmi.m_45s')
ser_info.segments
```

[53]:
```
                type  units protocol       dims         note
name
magnetogram   int   Gauss     fits  4096x4096  magnetogram
```

As you can see, all table structures returned by drms are in the form of Panda DataFrames.

We can now make a query to the JSOC. The following query provides the data series, then in sqaure brackets we have the data, how many days forward to look (e.g. 1 day) and at what intervals (6 hours).

[64]:
```python
k = c.query('hmi.m_45s[2014.10.22_TAI/1d@6h]', key='T_REC, CAMERA')
k
```

[64]:
```
                     T_REC  CAMERA
0  2014.10.22_00:00:00_TAI       2
1  2014.10.22_06:00:00_TAI       2
2  2014.10.22_12:00:00_TAI       2
3  2014.10.22_18:00:00_TAI       2
```

[65]:
```python
k,s= c.query('hmi.m_45s[2014.10.22_TAI/1d@6h]', key='T_REC,␣
↪CAMERA',seg='magnetogram')
```

[66]:
```python
s
```

[66]:
```
                             magnetogram
0   /SUM9/D624356059/S00008/magnetogram.fits
1   /SUM8/D624366826/S00008/magnetogram.fits
2  /SUM37/D624377627/S00008/magnetogram.fits
3  /SUM13/D624389154/S00008/magnetogram.fits
```

[67]:
```python
url = 'http://jsoc.stanford.edu'+s.magnetogram[0]
print(url)
```

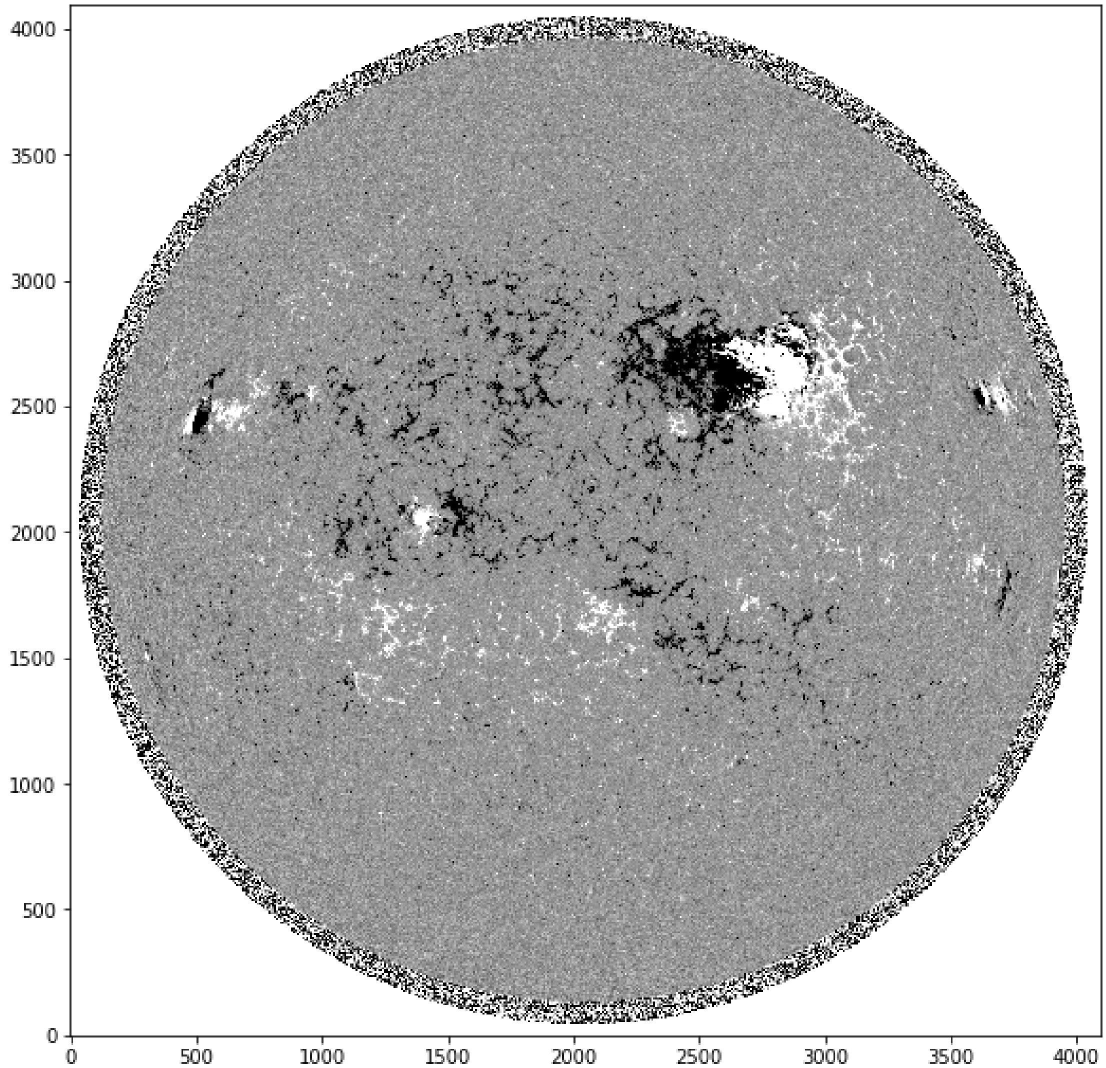
```
http://jsoc.stanford.edu/SUM9/D624356059/S00008/magnetogram.fits
```

[68]:
```python
from astropy.io import fits
data = fits.getdata(url)
print(data.shape, data.dtype)
```

```
Downloading http://jsoc.stanford.edu/SUM9/D624356059/S00008/magnetogram.fits
[Done]
(4096, 4096) float64
```

[106]:
```python
plt.figure(figsize=(10,10))
plt.imshow( np.clip(data,-100,100),origin='lower',cmap='Greys')
```

[106]: <matplotlib.image.AxesImage at 0x1a23fa5588>



Note that fits files accessed this way do not contain keyword data in their headers. In principle this isn't a problem as the Client.query() can be used to access all the keywords. See drms web page and github for examples.

The alternative is to send an export request to the JSOC Export Data webpage. Full details can be found on the [drms Tutorial web page] (https://drms.readthedocs.io/en/stable/tutorial.html).

It is also possible to download data with Fido in Sunpy. The Fido tool is built on top of drms, so functions similarly. You may find you have a personal preference.

In the following we access the VSO with Fido.

[89]:
```
import astropy.units as u
from sunpy.net import Fido, attrs as a
```

Next, we give the date, time, instrument and wavelength of the observation for the AIA instrument on-board SDO. We use the OR operator to access 2 wavelengths, 171 and 94 Angstroms.

```
[90]: result = Fido.search(a.Time('2012/03/04 00:00', '2012/03/04 00:02'),
                         a.Instrument('aia'),
                         a.Wavelength(171*u.angstrom) | a.Wavelength(94*u.angstrom))
      print(result)
```

```
Results from 2 Providers:

11 Results from the VSOClient:
   Start Time [1]       End Time [1]     Source …   Type   Wavelength [2]
                                                …            Angstrom
       str19               str19          str3  …   str8      float64
------------------- ------------------- ------ … -------- --------------
2012-03-04 00:00:00 2012-03-04 00:00:01    SDO … FULLDISK 171.0 .. 171.0
2012-03-04 00:00:12 2012-03-04 00:00:13    SDO … FULLDISK 171.0 .. 171.0
2012-03-04 00:00:24 2012-03-04 00:00:25    SDO … FULLDISK 171.0 .. 171.0
2012-03-04 00:00:36 2012-03-04 00:00:37    SDO … FULLDISK 171.0 .. 171.0
2012-03-04 00:00:48 2012-03-04 00:00:49    SDO … FULLDISK 171.0 .. 171.0
2012-03-04 00:01:00 2012-03-04 00:01:01    SDO … FULLDISK 171.0 .. 171.0
2012-03-04 00:01:12 2012-03-04 00:01:13    SDO … FULLDISK 171.0 .. 171.0
2012-03-04 00:01:24 2012-03-04 00:01:25    SDO … FULLDISK 171.0 .. 171.0
2012-03-04 00:01:36 2012-03-04 00:01:37    SDO … FULLDISK 171.0 .. 171.0
2012-03-04 00:01:48 2012-03-04 00:01:49    SDO … FULLDISK 171.0 .. 171.0
2012-03-04 00:02:00 2012-03-04 00:02:01    SDO … FULLDISK 171.0 .. 171.0

10 Results from the VSOClient:
   Start Time [1]       End Time [1]     Source …   Type   Wavelength [2]
                                                …            Angstrom
       str19               str19          str3  …   str8      float64
------------------- ------------------- ------ … -------- --------------
2012-03-04 00:00:02 2012-03-04 00:00:03    SDO … FULLDISK   94.0 .. 94.0
2012-03-04 00:00:14 2012-03-04 00:00:15    SDO … FULLDISK   94.0 .. 94.0
2012-03-04 00:00:26 2012-03-04 00:00:27    SDO … FULLDISK   94.0 .. 94.0
2012-03-04 00:00:38 2012-03-04 00:00:39    SDO … FULLDISK   94.0 .. 94.0
2012-03-04 00:00:50 2012-03-04 00:00:51    SDO … FULLDISK   94.0 .. 94.0
2012-03-04 00:01:02 2012-03-04 00:01:03    SDO … FULLDISK   94.0 .. 94.0
2012-03-04 00:01:14 2012-03-04 00:01:15    SDO … FULLDISK   94.0 .. 94.0
2012-03-04 00:01:26 2012-03-04 00:01:27    SDO … FULLDISK   94.0 .. 94.0
2012-03-04 00:01:38 2012-03-04 00:01:39    SDO … FULLDISK   94.0 .. 94.0
2012-03-04 00:01:50 2012-03-04 00:01:51    SDO … FULLDISK   94.0 .. 94.0
```

To download the files we use Fido.fetch.

[94]:
```
result = Fido.search(a.Time('2012/03/04 00:00:00', '2012/03/04 00:00:30'),
                     a.Instrument('aia'),a.Wavelength(171*u.angstrom))

from os.path import expanduser
home = expanduser("~")              # provides home directory, e.g. '/Users/
 ↪richardmorton/'
path='~/analysis/sdo/'


downloaded_files = Fido.fetch(result,path=path) #download fits files to path
print(downloaded_files)
```

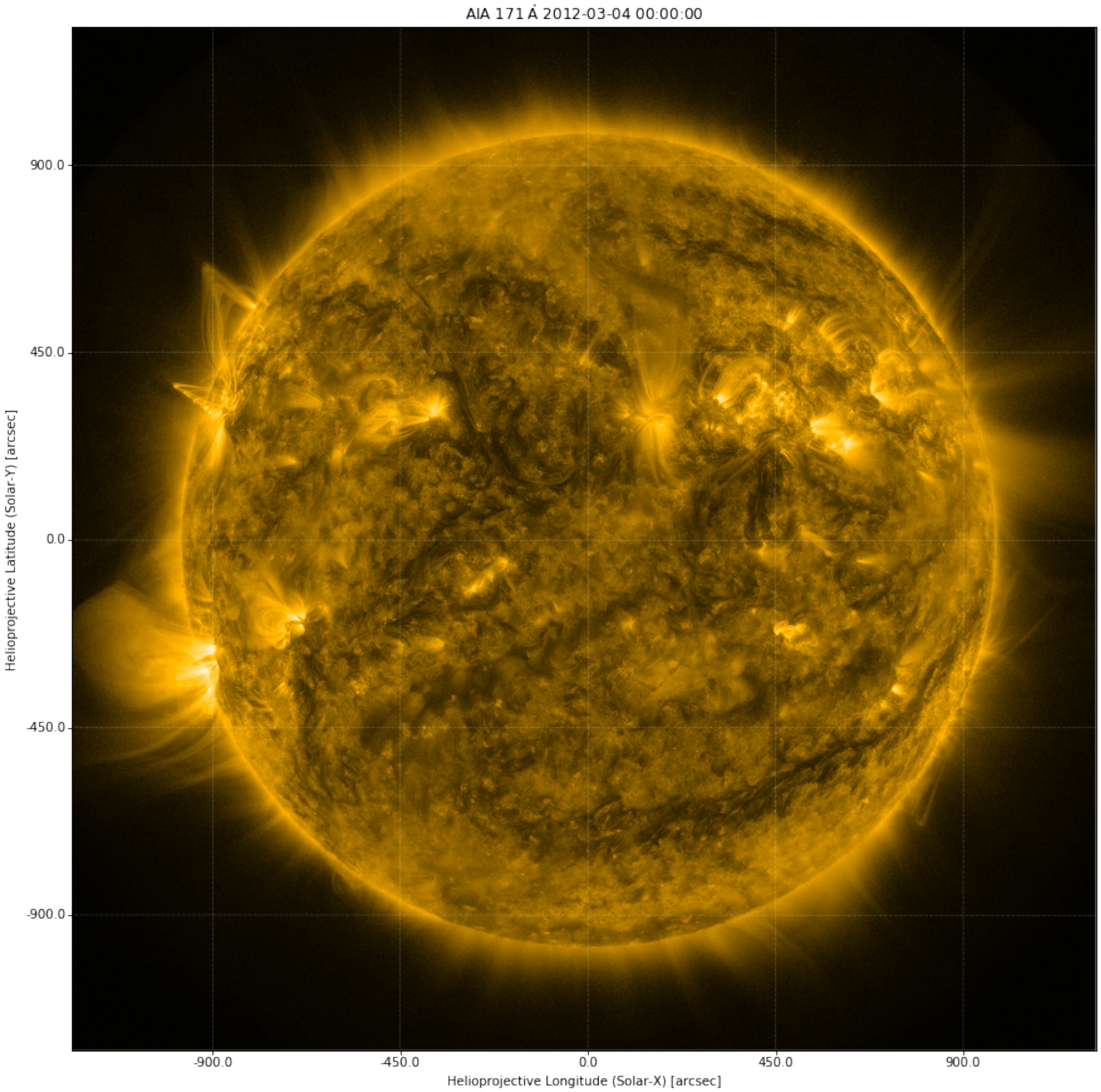
```
[
========================================
['/Users/richardmorton/analysis/sdo/aia_lev1_171a_2012_03_04t00_00_00_34z_image_
lev1.fits', '/Users/richardmorton/analysis/sdo/aia_lev1_171a_2012_03_04t00_00_12
_35z_image_lev1.fits', '/Users/richardmorton/analysis/sdo/aia_lev1_171a_2012_03_
04t00_00_24_34z_image_lev1.fits']
```

4.3 Plotting maps

Now that we have downloaded data, we can create a map object with the data. This helps us to show the image and includes some additional details about the data, e.g. date, correct solar coordinates.

[101]:
```
import sunpy

aia_map = sunpy.map.Map(downloaded_files[0])
fig = plt.figure(figsize=(15,15))
aia_map.plot()
plt.show()
```


AIA 171 Å 2012-03-04 00:00:00
900.0
450.0
0.0
-450.0
-900.0
Helioprojective Latitude (Solar-Y) [arcsec]
-900.0
-450.0
0.0
450.0
900.0
Helioprojective Longitude (Solar-X) [arcsec]