Data Importing and Wrangling

December 1, 2017

0.0.1 SNe Data

Start by importing some raw SNe to wrangle. This data was aquired from the database found at https://sne.space. This CSV file contains information on where in the sky the SNe were observed (RA and Dec), on what day the optical magnitute reached its maximum, and the measured redshift (distance).

Steps: 1. Because the neutrino data is in terms of Modified Julian Day (MJD) we will convert the SNe Gregorian dates to MJD 2. The RA's and Dec's need to be converted to radians 3. There are multiple values for some of the RA's, Dec's, and redshit measurements. Some of these are to different significant figures, but since they are relatively close I take the mean of the measurements. 4. Finally, we output data by filtering the correct SNe Types and dates corresponding to the neutrino data

In [1]:
```
import pandas as pd
import numpy as np

from astropy.time import Time
from astropy import units as u
from astropy.coordinates import Angle
```

In [2]:
```
cd ../data
```

```
/Users/nicholassenno/DataScience/sne_nu/data
```

In [3]:
```
# read in the SNe data

sne_data = pd.read_csv('raw_sne_data.csv',index_col='Name')
sne_data.head()
```

Out[3]:

Name	Max Date	R.A.	Dec.	z	Type
SN2011ep	2011/07/08	17:03:41.78	+32:45:52.6,+32:45:52.60	0.28,0.28	Ic
PTF11ixk	2011/07/23	13:21:45.03	+31:14:04.6	0.021	Ic
PTF11izq	2011/07/25	13:47:30.11	+40:04:32.5	0.062	Ib
PTF11ilr	2011/07/29	23:07:32.50	+15:20:23.0	NaN	Ib
SN2011ee	2011/07/31	23:27:57.34	+08:46:38.1,+08:46:38.10,+08:46:38.0	0.03	Ic

In [4]:
```
# convert the Gregorian dates to MJD using Time from astropy.time package
# add a new column to the df with MJD dates
```

```
sne_data['Max Date (MJD)'] = Time([date for date in \
                                   sne_data['Max Date'].str.replace('/','-')]).mjd + 0.5
```

In [5]:
```
sne_data.head()
```

Out[5]:

Name	Max Date	R.A.	Dec.	z	Type	Max Date (MJD)
SN2011ep	2011/07/08	17:03:41.78	+32:45:52.6,+32:45:52.60	0.28,0.28	Ic	55750.5
PTF11ixk	2011/07/23	13:21:45.03	+31:14:04.6	0.021	Ic	55765.5
PTF11izq	2011/07/25	13:47:30.11	+40:04:32.5	0.062	Ib	55767.5
PTF11ilr	2011/07/29	23:07:32.50	+15:20:23.0	NaN	Ib	55771.5
SN2011ee	2011/07/31	23:27:57.34	+08:46:38.1,+08:46:38.10,+08:46:38.0	0.03	Ic	55773.5

In [6]:
```
# Define function to calculate the average values of R.A. and Dec.
# ang_unit is a kwarg which describes how the angular data is presented
#
#   R.A. -- u.hourangle
#   Dec. -- u.degree


def get_average_angle(ang, ang_unit = u.degree) :
    return np.mean([y for y in map(lambda x : Angle(x,ang_unit).rad,ang.split(','))])

# Define function to calculate the average value of redshift (z)
# Most values are strings that are separated by ','
# However, NaNs are floats. A -1 is returned to signal NA data

def get_average_z(z) :
    return np.mean(list(map(float,z.split(',')))) if type(z) is str else -1
```

In [7]:
```
# Clean the R.A., Dec., and z values with one list comprehension
# Create new columns for R.A. and Dec. in rad
# Replace the z column with the new values

sne_data['R.A. (rad)'], sne_data['Dec. (rad)'], \
    sne_data['z'] = list(zip(* [(get_average_angle(row['R.A.'],u.hourangle), \
        get_average_angle(row['Dec.']), get_average_z(row['z'])) \
            for _, row in sne_data.iterrows()]))
```

In [8]:
```
sne_data.head(10)
```

Out[8]:

Name	Max Date	R.A.	Dec.	z	Type	Max Date (MJD)
SN2011ep	2011/07/08	17:03:41.78	+32:45:52.6,+32:45:52.60	0.28,0.28	Ic	55750.5
PTF11ixk	2011/07/23	13:21:45.03	+31:14:04.6	0.021	Ic	55765.5
PTF11izq	2011/07/25	13:47:30.11	+40:04:32.5	0.062	Ib	55767.5

```
PTF11ilr  2011/07/29  23:07:32.50                          +15:20:23.0
SN2011ee  2011/07/31  23:27:57.34  +08:46:38.1,+08:46:38.10,+08:46:38.0
PTF11kaa  2011/08/02  17:26:24.17                          +46:51:29.6
SN2011gd  2011/08/17  16:34:25.67  +21:32:28.4,+21:32:28.39,+21:32:28.3
PTF11klg  2011/09/06  22:07:09.92                          +06:29:08.7
PTF11kmb  2011/09/16  22:22:53.61                          +36:17:36.5
SN2011fl  2011/09/25  00:47:19.93              +27:49:35.5,+27:49:35.51
```

Name	z	Type	Max Date (MJD)	R.A. (rad)	Dec. (rad)
SN2011ep	0.280000	Ic	55750.5	4.466718	0.571850
PTF11ixk	0.021000	Ic	55765.5	3.498297	0.545147
PTF11izq	0.062000	Ib	55767.5	3.610658	0.699453
PTF11ilr	-1.000000	Ib	55771.5	6.054293	0.267729
SN2011ee	0.030000	Ic	55773.5	6.143366	0.153192
PTF11kaa	0.040000	Ib	55775.5	4.565794	0.817830
SN2011gd	0.009800	Ib	55790.5	4.339010	0.375965
PTF11klg	0.026522	Ic	55810.5	5.790851	0.113198
PTF11kmb	0.017000	Ib-Ca	55820.5	5.859478	0.633441
SN2011fl	0.015800	Ib	55829.5	0.206526	0.485665

We would also like to filter out only Type Ib/c SNe. Start by examining all the unique Types.

In [9]:
```
bad_types = ['SLSN-II','II']
```

In [10]:
```
unique_types = sne_data.Type.unique()
print(unique_types)

right_type = [sne_type not in bad_types for sne_type in sne_data.Type]
```

```
['Ic' 'Ib' 'Ib-Ca' 'SLSN-I' 'Ic BL' 'Ibn' 'Ib/c' 'Ic?' 'SLSN-II' 'IIb/Ib'
 'Ic-lum?' 'II' 'Ib/IIb' 'Ib/c-BL' 'Ic Pec']
```

Most of these types are acceptable (I had already excluded Type Ia SNe from sne.space). However, we would like to remove the Type II and Type SLSN II.

Based on the neutrino data, we only want SNe for which neutrinos are within the 99% Poisson confidence interval assuming an average delay of 13 days between core-collapse and maximum brightness (see text for details).

In [11]:
```
min_mjd = 55750.5
max_mjd = 56068.5

in_time_window = (sne_data['Max Date (MJD)'] >= min_mjd) & \
    (max_mjd >= sne_data['Max Date (MJD)'])
```

Write the cleaned data to a file 'cleaned_sne_data.csv'. Note that we have only used some of the columns, and changed their order to help with readability.

In [12]:
```
sne_data.loc[in_time_window*right_type, \
    ['Max Date (MJD)','R.A. (rad)','Dec. (rad)','z','Type']].sort_values( \
        'Max Date (MJD)').to_csv('cleaned_sne_data.csv')
```