Hard Additional Requirement 2

The aim of this additional requirment is to analyse some other data sets.

The data set being analysed in this notebook is data that NASA has collected: https://data.nasa.gov/Space-Science/Meteorite-Landings/gh4g-9sfh (https://data.nasa.gov/Space-Science/Meteorite-Landings/gh4g-9sfh). A discription of each of the columns is available at: https://www.kaggle.com/nasa/meteorite-landings (https://www.kaggle.com/nasa/meteorite-landings)

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patch
from ipywidgets import *
%matplotlib inline
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.axes3d import Axes3D, get_test_data
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
from matplotlib import cm
from operator import itemgetter

import pandas as pd
import numpy as np
import plotly
import plotly.graph_objs as go
import plotly.plotly as py

from ipywidgets import widgets
from IPython.display import display
from plotly.graph_objs import *
from plotly.widgets import GraphWidget
from plotly.offline import download_plotlyjs, init_notebook_mode, plot,
iplot
plotly.offline.init_notebook_mode(connected=True)

import cufflinks as cf
init_notebook_mode(connected=True)

from scipy.misc import imread
```

```
/cs/home/ea50/.local/lib/python3.5/site-packages/IPython/html.py:14: Shim
Warning:

The `IPython.html` package has been deprecated since IPython 4.0. You sho
uld import from `notebook` instead. `IPython.html.widgets` has moved to `
ipywidgets`.

/cs/home/ea50/.local/lib/python3.5/site-packages/IPython/utils/traitlets.
py:5: UserWarning:

IPython.utils.traitlets has moved to a top-level traitlets package.
```

First the data needs to be read in.

In [2]:
```
df=pd.read_csv("../data/Meteorite_Landings.csv")
```

Secondly the refining of the data is done, so any duplicate rows, or row which have a 'None' value is dropped.

In [3]:
```
refinedNASA = df.copy()
refinedNASA = refinedNASA.dropna()
refinedNASA = refinedNASA.drop_duplicates()
```

Then the only two columns which have a limited number of possible values are checked, and any row which has invalid values also gets dropped. These possible values was specified in the links up above in the explanation of the NASA data.

In [4]:
```
#possible valid values
fallarray = ['Fell', 'Found']
nametypearray = ['Valid', 'Relict']

#keeps track fo which rows are invalid so need to be removed
rowsToRemove = []

#for each column the values are all checked
rCounter = 0
falls = refinedNASA['fall']
for fa in falls:
    flag = 1
    for f in fallarray:
                if f == fa:
                    flag = 0
    if flag == 1:
        rowsToRemove.append(rCounter)
    rCounter = rCounter + 1

rCounter = 0
types = refinedNASA['nametype']
for typ in types:
    flag = 1
    for name in nametypearray:
                if name == typ:
                    flag = 0
    if flag == 1:
        rowsToRemove.append(rCounter)
    rCounter = rCounter + 1

#list is reversed so that when the rows are getting removed it doesn't i
nterfere with the indexes of other rows
rowsToRemove.sort(reverse=True)

for i in rowsToRemove:
    refinedNASA = refinedNASA.drop(refinedNASA.index[[i]])

#CSV file for the refined Data is made
refinedNASA.to_csv("refinedNASA.csv")
```

Finding all the possible values for each column and their occurances, all except 'name' and 'id'.

In [5]:
```
nametype = refinedNASA.groupby('nametype')
nametype.size().sort_values()
```

Out[5]:
```
nametype
Relict       21
Valid     38095
dtype: int64
```

In [6]:
```
recclass = refinedNASA.groupby('recclass')
recclass.size().sort_values()
```

Out[6]:
```
recclass
H6                         1
R                          1
H3.7-6                     1
H3.7/3.8                   1
Pallasite?                 1
H3.8-4                     1
H3.8-5                     1
H3.8-6                     1
H3.8/3.9                   1
H4(?)                      1
Lunar (bas. breccia)       1
H4-melt breccia            1
Mesosiderite?              1
H5-7                       1
Mesosiderite-C             1
Howardite-an               1
L5-7                       1
Mesosiderite-B             1
L~3                        1
Iron, IAB-sHL-an           1
L4-melt breccia            1
Lunar (norite)             1
L4-an                      1
H3.5-4                     1
H3.4/3.5                   1
LL3.05                     1
Fusion crust               1
H(?)4                      1
H(L)3                      1
H(L)3-an                   1
                        ...
LL3                       88
Iron, ungrouped          105
H~5                      106
Mesosiderite             107
Iron, IIAB               111
Eucrite                  115
CR2                      116
H5/6                     166
Eucrite-pmict            169
Diogenite                178
Howardite                179
CV3                      184
LL4                      198
E3                       205
Ureilite                 214
LL                       223
L3                       268
Iron, IIIAB              270
CO3                      308
H3                       313
CM2                      330
H4/5                     395
L4                       939
LL6                     1660
LL5                     2199
L5                      3264
H4                      3880
H6                      3898
H5                      6243
L6                      7519
dtype: int64
```

In [7]:
```
fall = refinedNASA.groupby('fall')
fall.size().sort_values()
```

Out[7]:
```
fall
Fell      1065
Found    37051
dtype: int64
```

In [8]:
```
year = refinedNASA.groupby('year')
year.size().sort_values()
```

Out[8]:
```
year
01/01/1583 12:00:00 AM       1
01/01/1787 12:00:00 AM       1
01/01/1790 12:00:00 AM       1
01/01/1792 12:00:00 AM       1
01/01/1793 12:00:00 AM       1
01/01/1794 12:00:00 AM       1
01/01/1797 12:00:00 AM       1
01/01/1801 12:00:00 AM       1
01/01/1806 12:00:00 AM       1
01/01/1809 12:00:00 AM       1
01/01/1785 12:00:00 AM       1
01/01/1817 12:00:00 AM       1
01/01/1821 12:00:00 AM       1
01/01/1832 12:00:00 AM       1
01/01/1833 12:00:00 AM       1
12/24/1399 12:00:00 AM       1
01/01/2101 12:00:00 AM       1
01/08/0601 12:00:00 AM       1
12/22/1575 12:00:00 AM       1
12/23/1490 12:00:00 AM       1
12/23/1491 12:00:00 AM       1
01/01/1820 12:00:00 AM       1
01/01/1781 12:00:00 AM       1
12/28/0860 12:00:00 AM       1
01/01/1775 12:00:00 AM       1
01/01/1600 12:00:00 AM       1
01/01/1623 12:00:00 AM       1
01/01/1628 12:00:00 AM       1
01/01/1632 12:00:00 AM       1
01/01/1637 12:00:00 AM       1
                          ...
01/01/2004 12:00:00 AM     366
01/01/1992 12:00:00 AM     372
01/01/1985 12:00:00 AM     377
01/01/1984 12:00:00 AM     402
01/01/1977 12:00:00 AM     420
01/01/1981 12:00:00 AM     458
01/01/1995 12:00:00 AM     484
01/01/1996 12:00:00 AM     573
01/01/1974 12:00:00 AM     691
01/01/2011 12:00:00 AM     713
01/01/1994 12:00:00 AM     717
01/01/1991 12:00:00 AM     869
01/01/1987 12:00:00 AM     914
01/01/2008 12:00:00 AM     936
01/01/1993 12:00:00 AM     976
01/01/2010 12:00:00 AM    1005
01/01/2007 12:00:00 AM    1038
01/01/2002 12:00:00 AM    1066
01/01/2001 12:00:00 AM    1339
01/01/1986 12:00:00 AM    1374
01/01/2009 12:00:00 AM    1496
01/01/2000 12:00:00 AM    1502
01/01/1997 12:00:00 AM    1505
01/01/1990 12:00:00 AM    1506
01/01/1999 12:00:00 AM    1592
01/01/2006 12:00:00 AM    1616
01/01/2003 12:00:00 AM    1754
01/01/1998 12:00:00 AM    2147
01/01/1988 12:00:00 AM    2295
01/01/1979 12:00:00 AM    3045
dtype: int64
```

In [9]:
```
reclat = refinedNASA.groupby('reclat')
reclat.size().sort_values()
```

Out[9]:
```
reclat
 18.178330       1
 19.680200       1
 19.681020       1
 19.681070       1
 19.682480       1
 19.683640       1
 19.685780       1
 19.685980       1
 19.686500       1
 19.678970       1
 19.686750       1
 19.693620       1
 19.694180       1
 19.696000       1
 19.697300       1
 19.699920       1
 19.701280       1
 19.702050       1
 19.703400       1
 19.692720       1
 19.678270       1
 19.677420       1
 19.677090       1
 19.663930       1
 19.663950       1
 19.664530       1
 19.664650       1
 19.665800       1
 19.666070       1
 19.667170       1
                ...
-72.989720      31
-72.773610      31
-82.500000      32
-25.233330      32
 29.916670      33
-72.778610      35
-83.250000      35
-72.983889      37
-72.782500      39
-72.779170      40
 34.083330      40
-72.998890      41
-72.778330      52
-72.775000      57
-72.983890      67
-72.778890      69
-72.954880      74
-85.633330     108
-24.850000     178
-85.666670     185
-86.716670     217
-86.366670     226
-84.216670     263
-76.183330     539
-76.716670     680
-79.683330    1130
-72.000000    1506
-84.000000    3040
-71.500000    4760
 0.000000     6410
dtype: int64
```

In [10]:
```
reclong = refinedNASA.groupby('reclong')
reclong.size().sort_values()
```

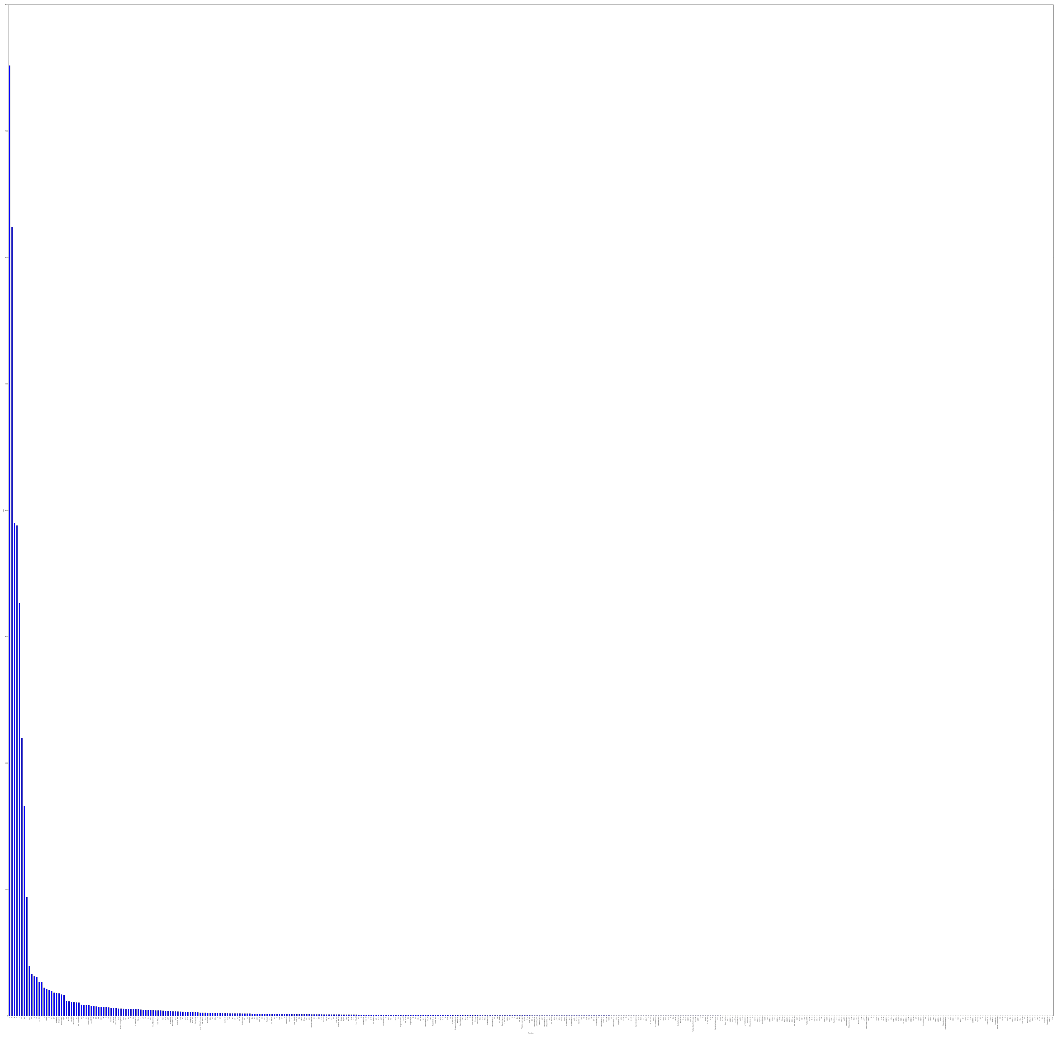
Out[10]:
```
reclong
 57.128720       1
 156.376960      1
 156.377890      1
 156.383440      1
 156.383980      1
 156.384060      1
 156.386150      1
 156.387370      1
 156.387400      1
 156.387420      1
 156.387810      1
 156.376940      1
 156.389140      1
 156.390240      1
 156.391450      1
 156.392460      1
 156.393020      1
 156.393690      1
 156.394680      1
 156.395670      1
 156.396530      1
 156.397840      1
 156.398280      1
 156.390150      1
 156.399260      1
 156.375810      1
 156.375320      1
 156.346820      1
 156.350000      1
 156.356920      1
                ...
 156.383330     26
 161.083330     29
 75.246389      30
 161.500000     30
-5.583330       31
 75.340000      31
-69.716670      32
 75.200000      32
 155.500000     32
 75.187220      33
 157.000000     34
-103.500000     35
 75.246390      42
 75.313610      42
 159.333330     44
 160.473280     74
-68.700000     105
-70.533330     178
 175.000000    185
-141.500000    217
-70.000000     228
 160.500000    263
 155.750000    473
 157.166670    542
 159.666670    637
 159.750000    657
 26.000000    1506
 168.000000   3040
 35.666670    4984
 0.000000     6186
dtype: int64
```

In [11]:
```
GeoLocation = refinedNASA.groupby('GeoLocation')
GeoLocation.size().sort_values()
```

Out[11]:
```
GeoLocation
(-1.002780, 37.150280)          1
(19.891870, 56.330470)          1
(19.892400, 56.667680)          1
(19.896680, 56.331020)          1
(19.899300, 56.172670)          1
(19.902650, 55.655383)          1
(19.905380, 56.654700)          1
(19.906370, 56.493600)          1
(19.908030, 55.908600)          1
(19.911170, 11.875170)          1
(19.911670, 56.335900)          1
(19.913920, 56.999920)          1
(19.914480, 55.662780)          1
(19.914550, 55.664470)          1
(19.891520, 56.328530)          1
(19.919330, 11.881830)          1
(19.924150, 56.336830)          1
(19.924880, 56.404880)          1
(19.928850, 55.668880)          1
(19.929430, 55.923470)          1
(19.930650, 56.083870)          1
(19.933330, 51.216670)          1
(19.936450, 56.360580)          1
(19.937030, 56.358020)          1
(19.937500, 56.474000)          1
(19.937550, 56.474030)          1
(19.937870, 56.395550)          1
(19.938400, 56.789020)          1
(19.938430, 56.422420)          1
(19.922820, 56.351170)          1
                              ...
(-80.066670, 156.383330)       26
(-72.983889, 75.246389)        27
(34.083330, -103.500000)       27
(-73.083330, 75.200000)        28
(-84.283330, 161.083330)       29
(-84.266670, 161.500000)       30
(29.916670, -5.583330)         31
(-82.500000, 155.500000)       32
(-25.233330, -69.716670)       32
(-72.998890, 75.187220)        32
(-83.250000, 157.000000)       34
(-72.983890, 75.246390)        38
(-72.778890, 75.313610)        39
(-76.716670, 159.333330)       42
(-72.954880, 160.473280)       74
(-85.633330, -68.700000)      105
(-24.850000, -70.533330)      178
(-85.666670, 175.000000)      185
(-86.716670, -141.500000)     217
(0.000000, 35.666670)         223
(-86.366670, -70.000000)      226
(-84.216670, 160.500000)      263
(-79.683330, 155.750000)      473
(-76.183330, 157.166670)      539
(-76.716670, 159.666670)      637
(-79.683330, 159.750000)      657
(-72.000000, 26.000000)      1505
(-84.000000, 168.000000)     3040
(-71.500000, 35.666670)      4760
(0.000000, 0.000000)         6186
dtype: int64
```

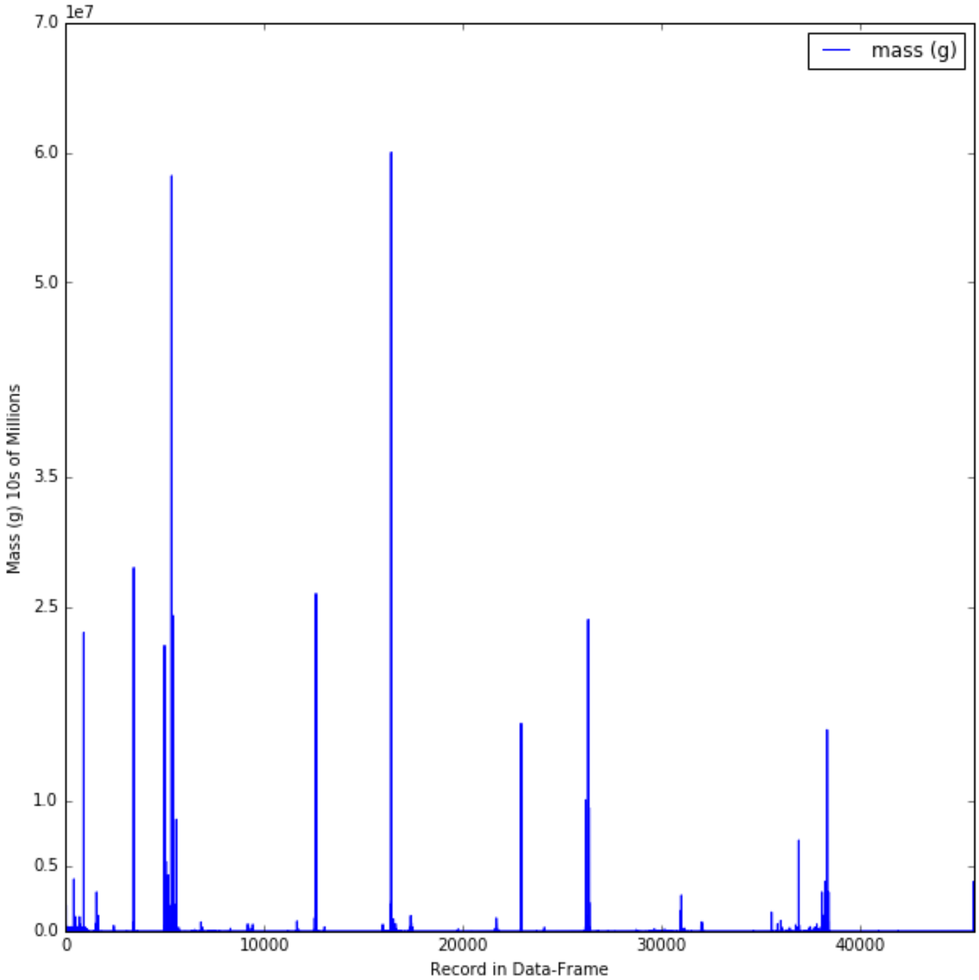
This bar plot shows all the possible clas of metorite, 'recclass', and the number of records each one has. Double-click on the figure to see in detail.

In [12]:
```
ax = refinedNASA['recclass'].value_counts().plot(kind="bar", figsize=(15
5, 150))
ax.set_xlabel("Recclass")
ax.set_ylabel("Count")
plt.show()
```



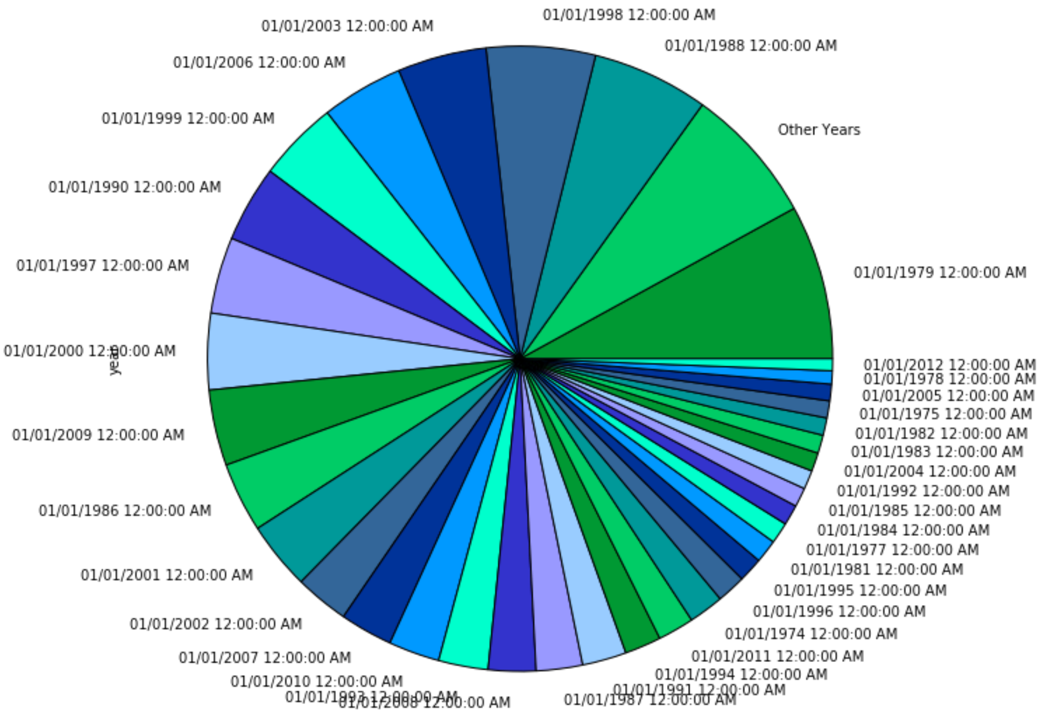
This line plot shows the masses, in grams, of each meteorite recorded.

In [13]:
```
ax = refinedNASA[['mass (g)']].plot(kind="line", figsize=(10,10))
ax.set_xlabel("Record in Data-Frame")
ax.set_ylabel("Mass (g) 10s of Millions")
#setting the y axis to have more helpful axis values
ax.set_yticks([0, 5000000, 10000000,25000000,35000000, 50000000,60000000
, 70000000])
plt.show()
```



In this next plot, a pie chart was used to show the number of metorites recorded in each year, the years which had less that 200 metorites were grouped together, to make the chart easier to see and to show all the labels clearly.

In [14]:
```
colors = ['#009933', '#00cc66', '#009999', '#336699', '#003399', '#0099f
f', '#00ffcc', '#3333cc', '#9999ff', '#99ccff']

data = refinedNASA.copy()
possibleYears = data.year.unique()
#keeping trach of all the rows to remove
rowsToRemove = []

years = data['year']
for possible in possibleYears:
    rCounter = 0
    flag = 1
    tempRows = []
    for year in years:
                if possible == year:
                    flag = flag +1
                    if rCounter not in tempRows:
                        tempRows.append(rCounter)
                rCounter = rCounter + 1
    #if there were less than 200 occurances of this year then it gets ad
ded to the remove list
    if flag < 200:
        #makes sure the row doesn't get added twice to the remove list
        for num in tempRows:
            if num not in rowsToRemove:
                rowsToRemove.append(num)

#keeps track of how many lines to re-add once they've been dropped
linesToAdd = len(rowsToRemove)
rowsToRemove.sort(reverse=True)
for i in rowsToRemove:
    data = data.drop(data.index[[i]])

#adds the same number of rows as deleted all with the same value for 'ye
ar'
for i in range(0, linesToAdd):
    line = pd.DataFrame([[1,1,1,1,1,1, "Other Years" ,1,1,1]], columns=[
'name', 'id', 'nametype', 'recclass', 'mass (g)', 'fall', 'year', 'recla
t', 'reclong', 'GeoLocation'])
    data = data.append(line, ignore_index=True)

data['year'].value_counts().plot(kind="pie", colors = colors, figsize=(1
0,10))
plt.show()
```



This interactive widget allows the user to look at the locations where Meteorites have fallen. To make it actually usable, the list of meteorites had to be limited, otherwise the list of them takes too long to generate and can sometimes crash the notebook due to the sheer number of records. The axes are as accurate as they could be made in accordance with the numbers on the map background image. The tab for the options available takes a bit of time to open, but if you give it a minute it will open.

In [17]:
```
#the image used as the background
img = imread("world-latitudes.jpg")
name = list(refinedNASA.name.unique())
#limits the list to every 5th element, this can be changed to the full l
ist, but it takes a while to collect all the data
names = name[::5]
#the function to make it interactive
def update(Meteorite = names):
    fig = plt.figure(figsize=(15,12))
    ax = fig.add_subplot(1, 1, 1)
    #gets the data for the Meteorite the user has selected
    data = refinedNASA[(((refinedNASA['name'] == Meteorite)))]
    #gets the 'GeoLocation' for the record
    geo = data.GeoLocation.iloc[0]
    numbers = geo.split("(")
    numbers = numbers[1].split(")")
    numbers = numbers[0].split(", ")
    #separates the two values fromt he data and stores them ready to be
used
    latitude = numbers[0]
    longitude = numbers[1]
    #sets the values for the y axis to be as close to the onces on the b
ackground image
    ax.set_yticks([-80,-60, -40, 0, 40,60, 80])
    ax.imshow(img, extent=[(-175),175,(-90),90])
    #drawing the two lines for the lat value and long value
    ax.set_xlabel("Longitude")
    ax.set_ylabel("Latitude")
    ax.axhline(y=(float(latitude)), color='red')
    ax.axvline(x=(float(longitude)), color='red')
    fig.canvas.draw()

interact(update);
```