Visualization of Earthquake Data

Visualization is often an important part of any data analysis project. A popular Python module used for visualization is called matplotlib so we'll start there. Specifically, we will be working with a part of matplotlib called pyplot[1]. So the first thing we want to do is import pyplot. I do it like this …

```python
import matplotlib.pyplot as plt
```

I'm going to work with data from the "selected" list created in the previous "data selection" document. Each item in the "selected" list consists of two elements: (1) a tuple that contains location information (latitude,longitude) as floats and (2) a dictionary that contains information about each earthquake event with these keys: Magnitude, Type, datetime, Magnitude Type, and Depth. In the previous document, we focused on selecting data records based on location, date, and magnitude. To follow along with my code, you must have a "selected" list. In this section, I'm using the entire data set … nothing is excluded.

The simplest thing to do in pyplot is a scatter plot … so let's do that. It will show us the location of every event in our data set.

Here, we establish separate lists of latitude (y-axis values) and longitude (x-axis values). Then we call plt.scatter() and plt.show(). This is as simple as it gets.

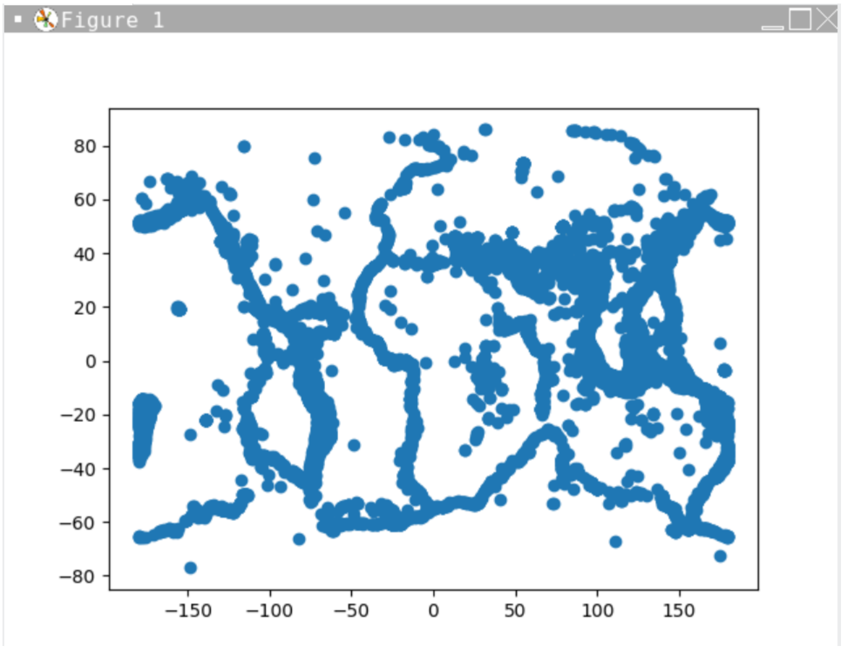
```python
# gather plot data
lats = [lat for (lat,lng), _ in selected ] # y-axis
lngs = [lng for (lat,lng), _ in selected ] # x-axis
# generate the graphic with default values
plt.scatter(lngs, lats)
# display the scatter plot
plt.show()
```

Notice that the result is quite bear. There's no way to know what the graphic means.

- No title.
- No axis labels.
- No units.

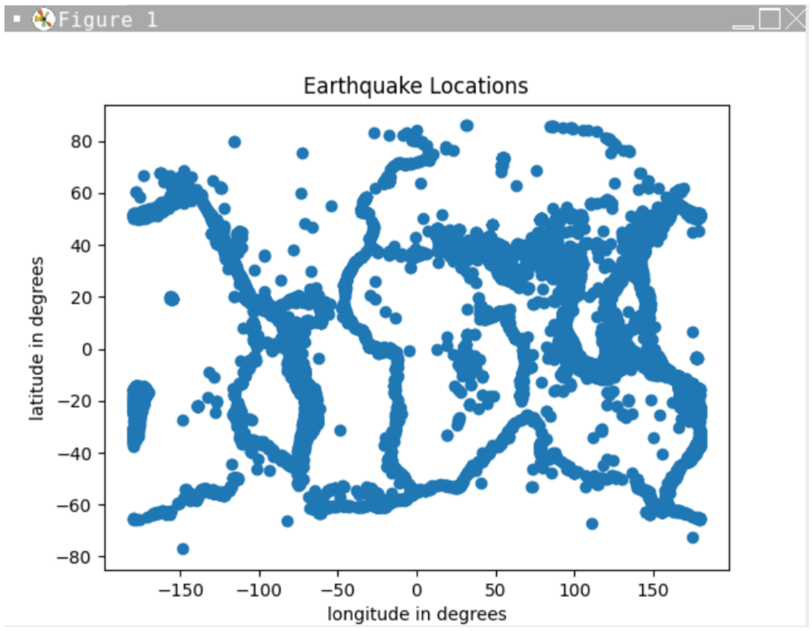
Let's fix that.

Adding labels is easily accomplished like this …



[1] find documentation here: https://matplotlib.org/stable/api/pyplot_summary.html

```
plt.scatter(lngs, lats)
plt.xlabel('longitude in degrees')
plt.ylabel('latitude in degrees')
plt.title('Earthquake Locations')
plt.show()
```

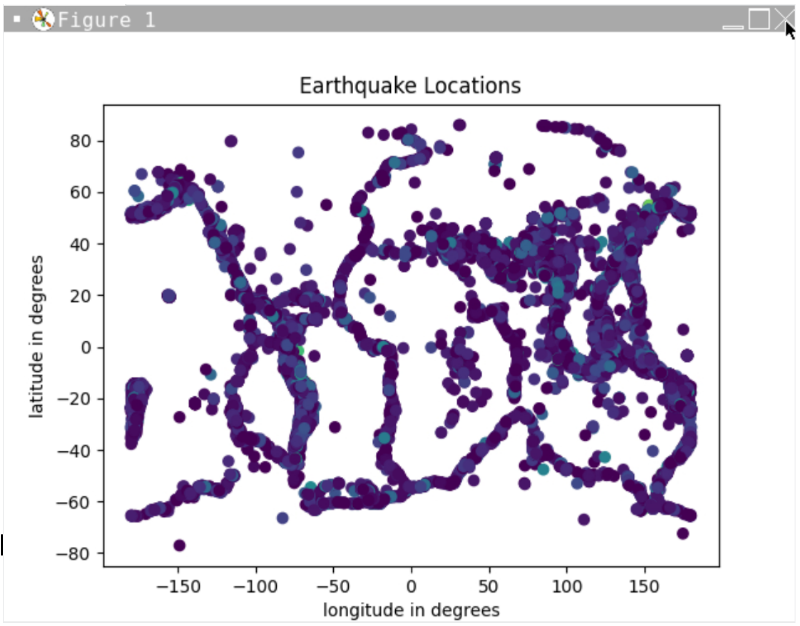


Now we have some labels and the graphic is looking a bit more professional. But there's still room for improvement. Some earthquakes are more severe than others. I can use pyplot to create a color scale that reflects earthquake magnitude. The first step is to create a list of magnitudes from our selected list. Then we assign the list of magnitudes to the plot colors and specify the range of colors we want to use. The process is called "color mapping" and pyplot provides many predefined color maps to choose from. In the code below, I use a color map called 'viridis'.

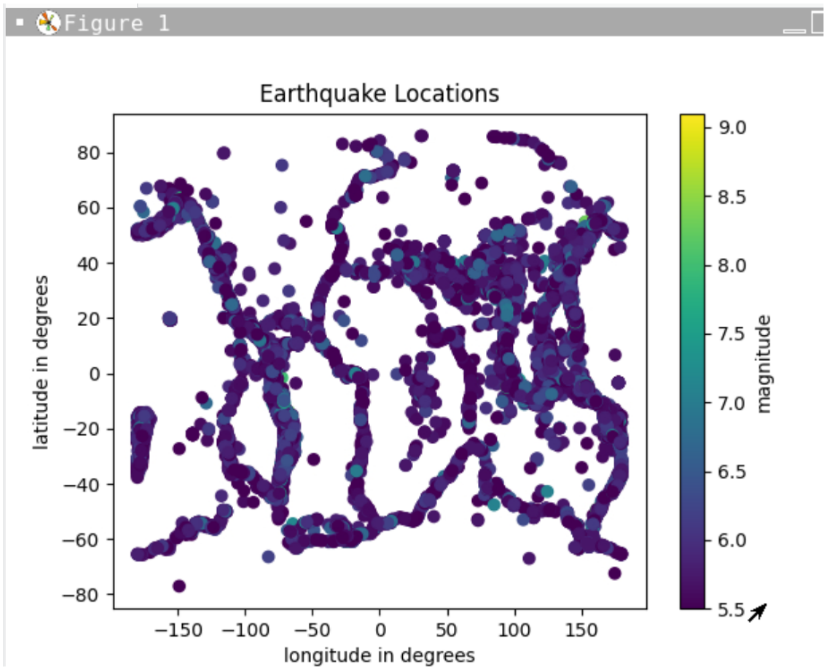
```
# gather plot data
lats = [lat for (lat,lng), _ in selected ] # y-axis
lngs = [lng for (lat,lng), _ in selected ] # x-axis
mags = [v['Magnitude'] for _, v in selected ] # colors
# *** plot with color mapping
plt.scatter(lngs, lats, c=mags, cmap='viridis')
plt.xlabel('longitude in degrees')
plt.ylabel('latitude in degrees')
plt.title('Earthquake Locations')
plt.show()
```

Hmm. The color map is not what I expected. Let's add a color bar to help us interpret the result.

```
plt.colorbar(label='magnitude')
```

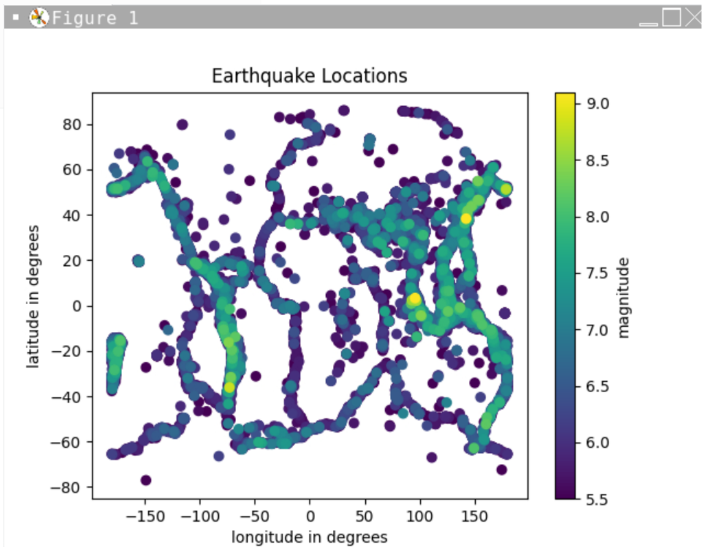


Here's the result of adding the colorbar …



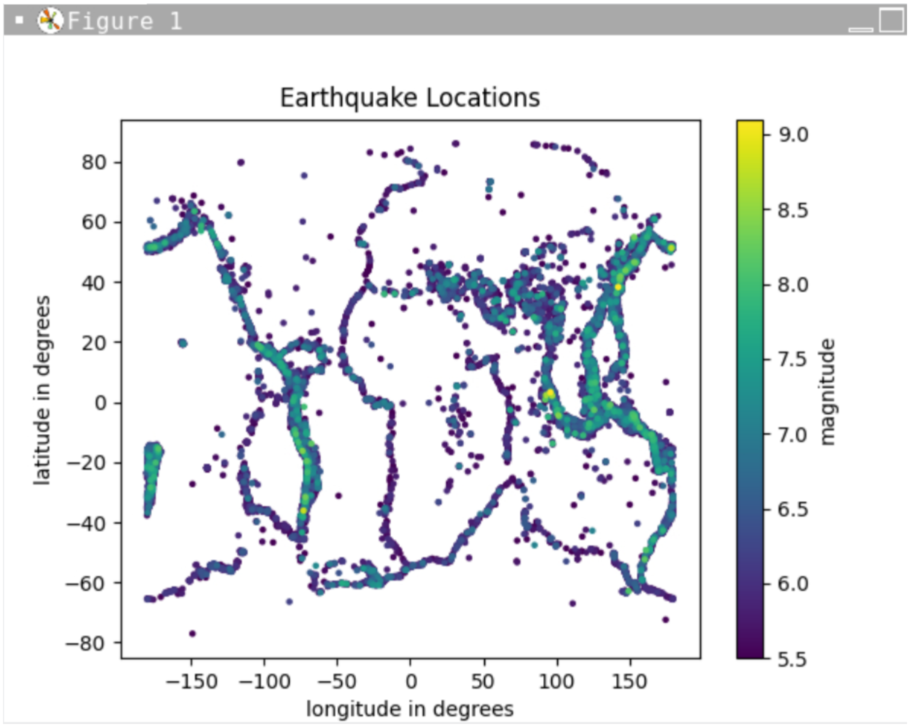
… some thought is required. I know that there are data records that should produce green and yellow dots but they are not shown in the scatter plot. Why not? This graphic has one dot for each of the 23,406 events in the data list. Some dots must be covering up the more severe earthquakes. If we assume that the dots are drawn in order, we may be able to see the more severe (and less frequent) events if we sort the events by magnitude. Let's try that.

```python
# sort by magnitude
selected.sort(key=lambda x:x[1]['Magnitude'])
# gather plot data
lats = [lat for (lat,lng), _ in selected ] # y-axis
lngs = [lng for (lat,lng), _ in selected ] # x-axis
mags = [v['Magnitude'] for _, v in selected ] # colors
plt.scatter(lngs, lats, c=mags, cmap='viridis')
plt.colorbar(label='magnitude')
plt.xlabel('longitude in degrees')
plt.ylabel('latitude in degrees')
plt.title('Earthquake Locations')
plt.show()
```

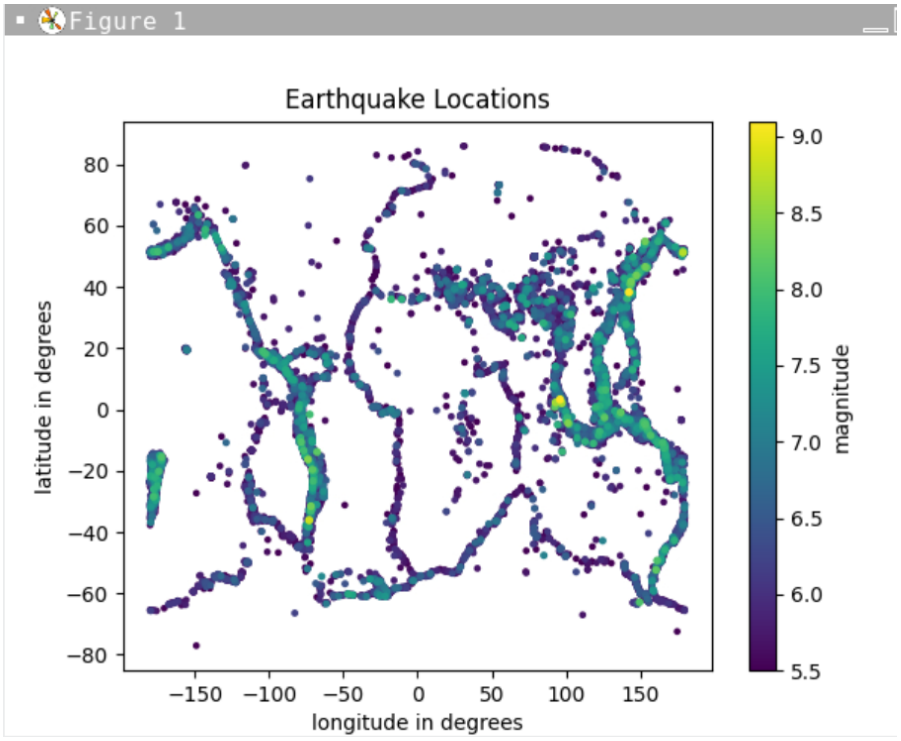


So, this is much better!
The more severe events really stand out now. Maybe just one more adjustment.

Let's make the dots smaller so we can better identify some of the individual events. Simply add an 's' parameter to specify the size: `plt.scatter(lngs, lats, s=5, c=mags, cmap='viridis')`



A list of values could also be assigned to the s parameter to make the dots different sizes based on magnitude. In the plot below, s=mags …



OK? So that's the scatter plot. Pretty easy. I suggest you try it out and modify it. The scatter plot is easy because the coordinates are already in the data and no aggregation is necessary.

What is aggregation? Aggregation is when you group data and compute some statistics for each group. We can call this "aggregate and compile". For example, we could collect the events for each time zone (aggregation). Then, we could count the number of events, and compute the average magnitude or other statistics (compilation) for each time zone. When we aggregate data, we can view each group as a "classification". The term we use to describe data organized into classifications is "ordinal" data. As we move forward, keep in mind that each classification should have a descriptive label for visualization purposes ok? In this exercise, we will compute some numerical value(s) for each classification. The customary way to produce such a figure is a bar chart where each bar represents a classification and the height of each bar represents a numerical value.

A time zone is just a range of latitudes, 15 degrees wide. There are 24 time zones around the globe. 24 bars would make for a very crowded chart. Let's split our longitude data into 6 zones and let's try to generalize the procedure. In the example below, I assume we are are working with all the data and it is in a "selected" list where each item is a tuple consisting of (1) a tuple indicating the coordinates (latitude,longitude) and (2) a dictionary containing the earthquake data (Magnitude, Type, etc.).

The first thing we need to do to aggregate our data is establish the "edges" of our classification regions. Conceptually, this is pretty simple: (1) determine max and min values to establish the entire range, (2) divide the range by the number of classifications (or "bins") to establish the width of each bin, (3) create a list of edges (the number of edges is one more than the number of bins), (4) create a label for each bin. This isn't always easy for numerical data because bar labels need to be short enough to fit under the bars.

To program this, I start by creating a list of longitudes (lngVals) so I can easily find the minimum and maximum values:

```
min(lngVals)=-179.997  max(lngVals)=179.998
```

Such values are a bit problematic because they won't produce the even 60 degree intervals I'm looking for. So, for my edges, I first compute a start value using floor(min(lngVals)) and an end value using ceil(max(lngVals)). This method (ceil(), floor()) works well here, but it does not work well for all data. For example, if the range is small (less than 1 in an extreme case) the method is useless and you'll need to adapt.

After establishing the total range, divide it by the number of bins to establish the bin width:
width =(end-start)/nBins
Now you can establish a list of edges using a list comprehension …

```
edges = [(int(start+i*width),[]) for i in range(1,bins+1)]
```

Note that my list of edges is actually a list of tuples where the first item in the tuple is the upper bound of a bin and the second item is an empty list. I do this because I'm thinking ahead to the next step which is the actual aggregation. The lists will be used to hold the data belonging to each bin. Here are the edges:

```
[(-120, []),(-60, []),(0, []),(60, []),(120, []),(180, [])]
```

I will convert this list to a dictionary where the keys are the upper bound of each bin:

```
lngRecs = dict(edges)
```

Now, I'm ready to aggregate data in the dictionary based on the keys..

To aggregate the data into the bins I use nested for loops. The outer for loop traverses my "selected" list and the inner for loop traverses the dictionary keys. I compare the longitude of the selected item with the key, if it's less than the key, I append the data record to the list associated with the key and break out of the inner loop. Otherwise, I allow the inner loop to move on to the next key (bin edge).

In the end, I'm left with a dictionary with keys that associate with lists of data records. Each data record is a dictionary that I can use to access any earthquake-related data I wish to compile into a bar chart. But before I do that, I still need a list of labels … one for each bar. The easy way out is to just convert the dictionary keys to strings … but that would only show the value at the end of each bin. I would like the labels to express the entire range. For example, the first label ought to be "`-180 to\n-120`". There are lots of ways to do this, I'll leave it to you to figure it out.

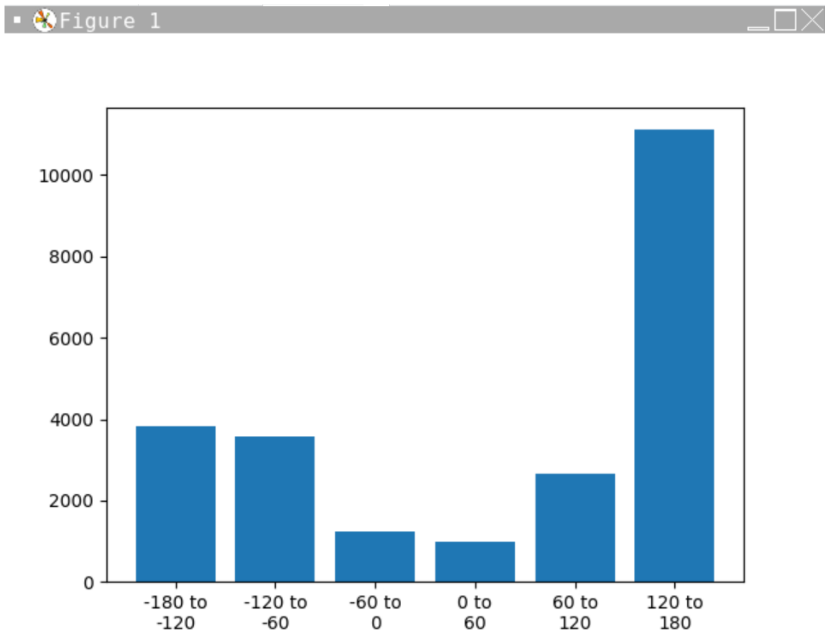
To create a simple bar chart, we minimally need two lists of exactly the same length: a list of labels (one string for each bar) and a list of numbers (each number specifies a bar height). For the purposes of this exercise, I'll just provide a list with the number of events in each bin. This is easily accomplished by taking the length of the list associated with each key. I'll leave this up to you.

Once you have the data, plotting the bar graph is easy:

```
plt.bar(labels,data)
```

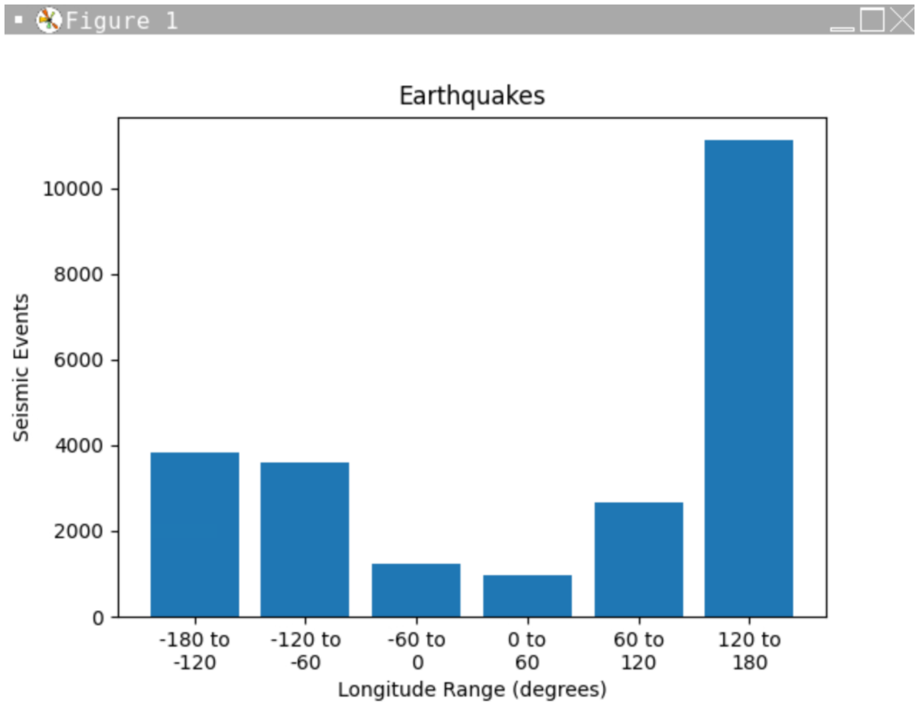
Some elements are missing. Most notably, axis labels and a title. You know how to do those from the scatterplot example. So go ahead and add those titles. "Earthquakes", "Seismic Events", "Longitude Range (degrees)".

What do you see? It's not quite right. It looks like there's not enough room for the x-axis



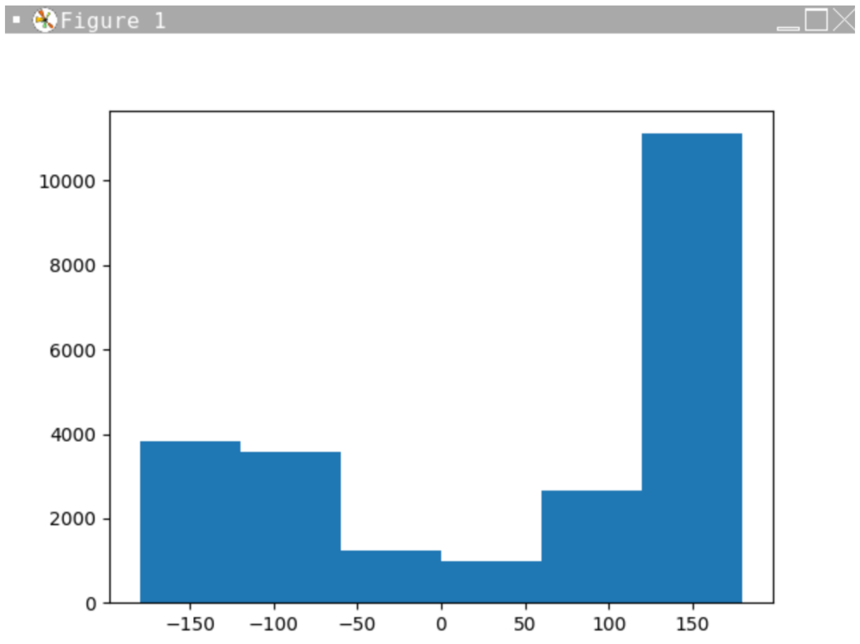
label. That's because the category labels take up two lines and pyplot didn't plan for that. We need to adjust the figure like this: `plt.subplots_adjust(bottom=0.15)`

Now, you ought to see this …



You may have noticed that this is a histogram. It shows the number of events in each category. pyplot can generate a histogram directly. Let's look at that.

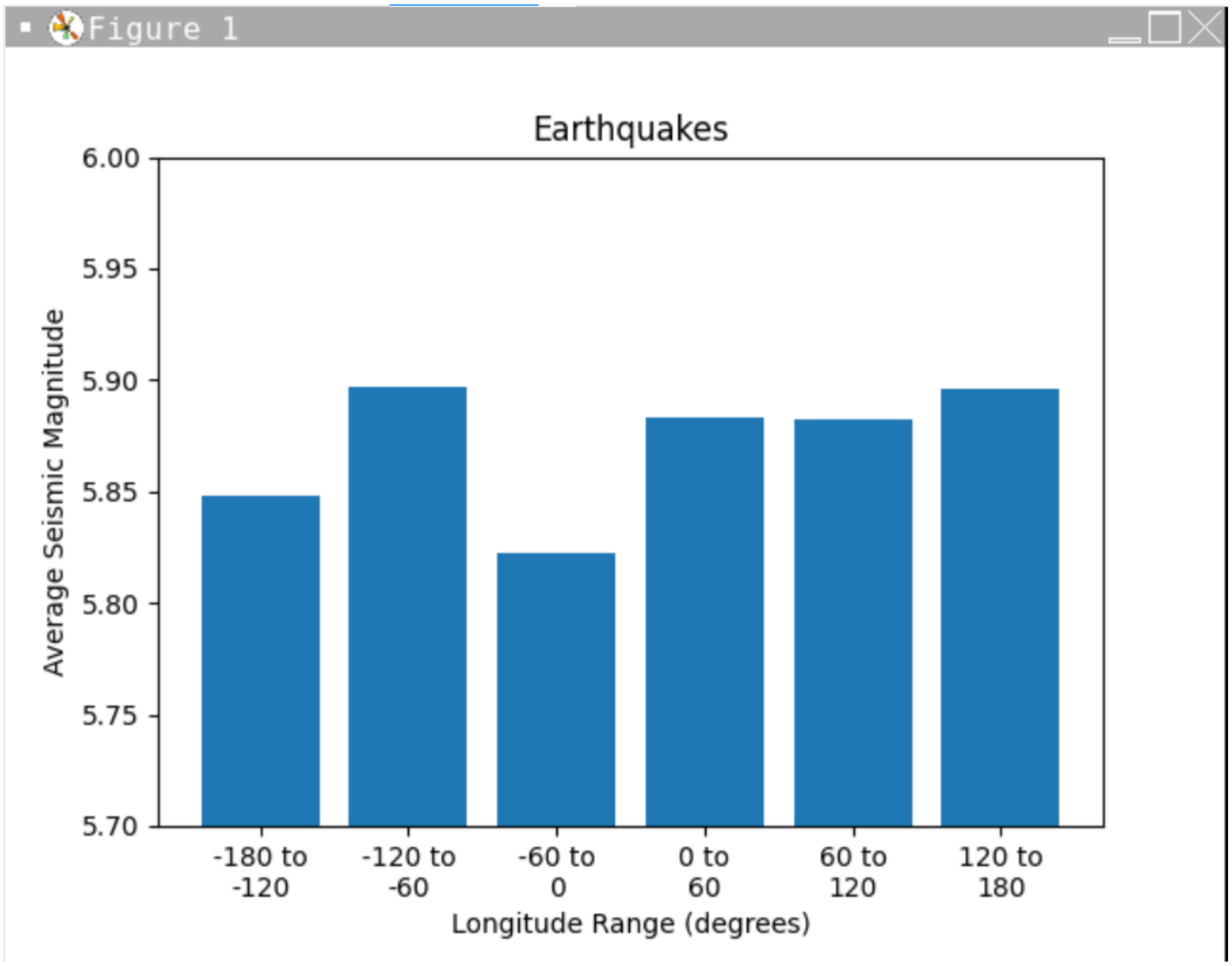
```
plt.hist(lngVals, bins=6, range=(-180, 180))
```



Clearly, the results look similar which provides some validation of our approach. The x-axis labels provide values in the middle of each range.

plt.hist() is a quick and dirty way to generate a histogram for exploratory analysis. When spaces between the bars are absent, the visual presentation is helpful for communicating statistical distributions.

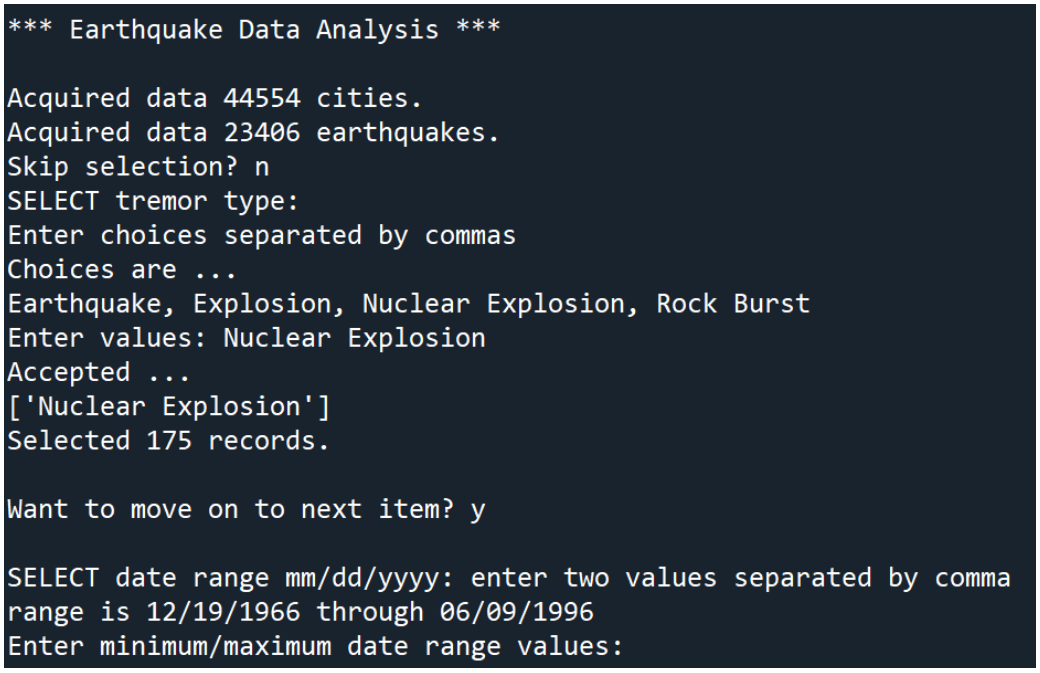
So, what about plotting other things? Once you have aggregated the data, you can choose any data in the dictionary to represent as a bar plot … average magnitude for example.



FINISHING THE PROJECT
This document combined with the in-class presentation you ought to have given you a pretty good idea about how to create graphics using matplotlib / pyplot. You also should also have some idea about how to use the extensive online documentation to figure out how to embellish your charts.

From the previous data selection document and in-class presentation, you ought to have a good idea of how to establish a user interface for data selection. At this point, your program ought to provide a way to select data records based on user criteria for location, date, and magnitude. Through your experiments, you may have noticed that there are additional interesting aspects to this data set. In particular, the presumed causes of the seismic activity are provided as "Type" which includes these categories: "Earthquake", "Explosion", "Nuclear Explosion", "Rock Burst". In your final submission, give the user the option to choose any of these.

In my implementation, the UI looks like this ..

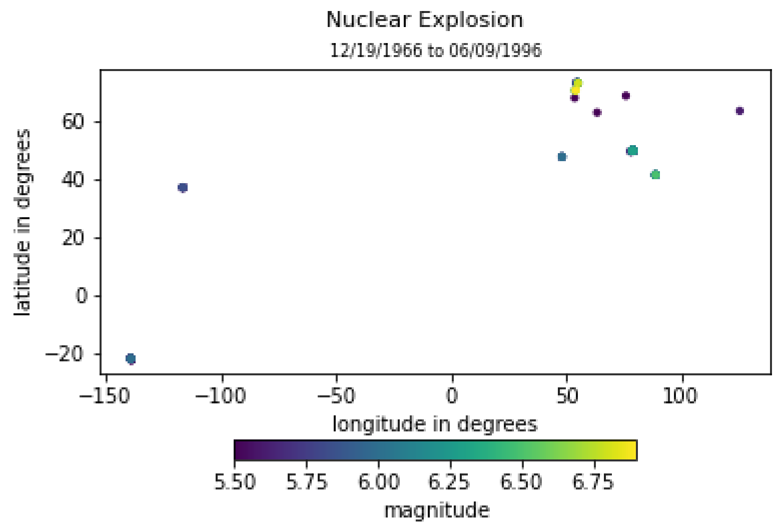


```
*** Earthquake Data Analysis ***

Acquired data 44554 cities.
Acquired data 23406 earthquakes.
Skip selection? n
SELECT tremor type:
Enter choices separated by commas
Choices are ...
Earthquake, Explosion, Nuclear Explosion, Rock Burst
Enter values: Nuclear Explosion
Accepted ...
['Nuclear Explosion']
Selected 175 records.

Want to move on to next item? y

SELECT date range mm/dd/yyyy: enter two values separated by comma
range is 12/19/1966 through 06/09/1996
Enter minimum/maximum date range values:
```

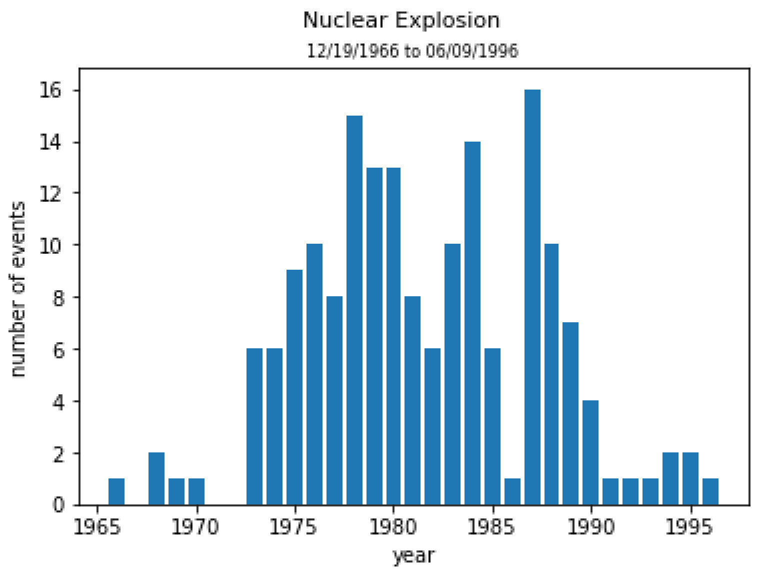
The choices shown are derived from the "Type" values in the data file (not hardcoded). In the example above, the first question "Skip selection?" is provided so that all data may be included in the selection without having to go through each specific category. Note that when "Nuclear Explosion" is selected, there are 175 events found in the data file that meet the criteria.

The next choices in my UI are date range, magnitude range, latitude range, and longitude range. If the user does not enter any criteria, all items are selected. The user interface was described in the previous (data selection) document.

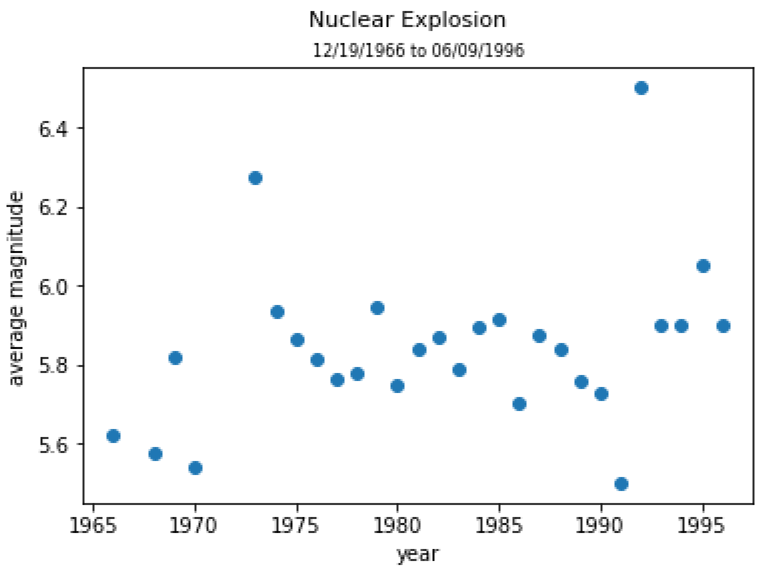
To complete the project, you will add three visualizations …
Scatter plot of events by location. The code for this is explained in the first part of this document and was demonstrated in class. The dot colors should represent tremor magnitude. In the example below, I show what this looks like when the 175 "Nuclear Explosions" are the only selected events.



A bar plot showing the number of events per year. Data records must be aggregated for each year. To accomplish this, create a dictionary where year is the key that refers to a list of selected events that occurred in that year. Then, in the final plot, the years form the x (categorical) axis and the length of the list associated with each year becomes the y-axis value for each bar (the number of events). In the example below, I show what this looks like when the 175 "Nuclear Explosions" are the only selected events.



A scatter plot showing the average magnitude for the selected earthquakes for each year. You have already aggregated the data to create the second graph and determined the number of events. So, creating the list of magnitudes for each year and then computing average magnitude should be relatively straightforward. In the example below, I show what this looks like when the 175 "Nuclear Explosions" are the only selected events.



THINGS TO NOTICE
There are some pyplot attributes used here that were not described in class to encourage you to do some exploration on your own. Here are some tips.

Titles. The titles are two lines: a title (smaller, providing the date range) and a "super-title" (larger, describing the tremor type). plt.title() is used to define the small title and plt.suptitle() is used to define the larger title. Use the "fontsize" keyword argument to establish the size (8 for the smaller title and 11 for the larger)

Colorbar. The default location for the colorbar is to the right of the data plot. Butthis causes some problems with centering of the titles so I moved it to the bottom and shrunk it. Use "location" and "shrink" keyword arguments for colorbar(). Mine is on the bottom and shrunk to 60%.

Divide by zero / missing values. Notice that there were no "Nuclear Explosions" for some years ('65, '67, '71, '72, and others). If you are computing the average and have zero events you will get a divide by zero error. We should include a "missing value" in the average magnitude data list for those years because the lists for x and y axis in pyplot must be the same length. Internally, matplotlib uses "numpy" representations for numbers. "numpy" is a module that can perform data manipulations very quickly, but to achieve that, it uses its own data structures for collections that differ from pure Python. So, to define a missing value (i.e. "nan" for not-a-number) we should use the numpy nan value. To do this I import numpy as np and then use np.nan anywhere I need to add a missing value.

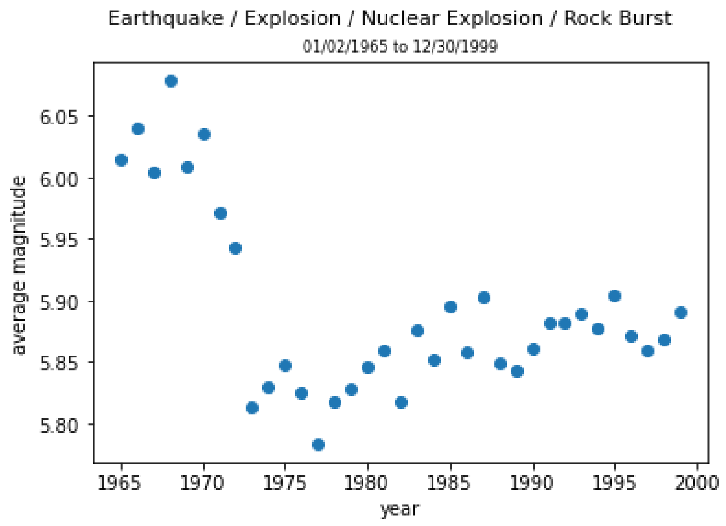
Title composition. In the graphs shown above, I've selected "Nuclear Explosion" and so that becomes the super-title and the title contains the date range. If I change these things, the titles change. In the example below, I choose *all events* across dates from 1/2/1965 through 12/31/2000.

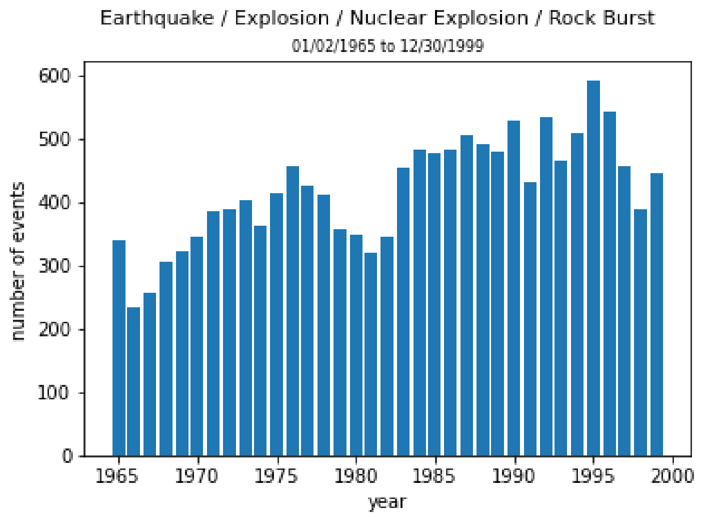
Here are the results. Note how the plot titles change with the selections made. This is easily done using the join() method and/or f-strings.

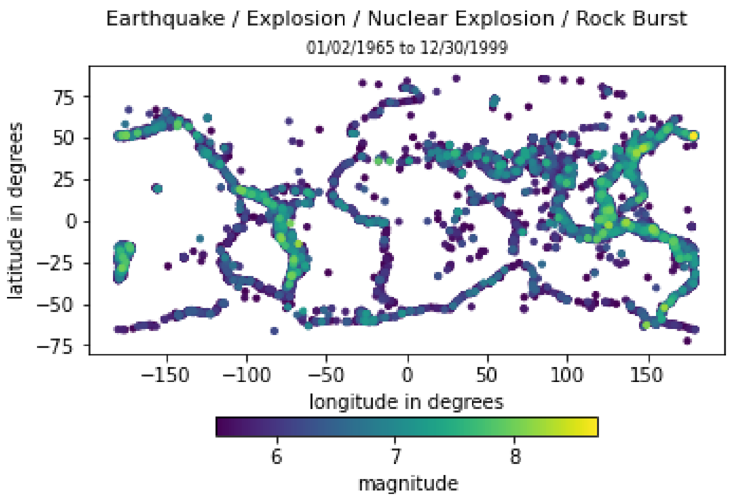
These simple pyplot visualizations are all that is required. pyplot does a good job of taking care of details like scaling the axes, generating axis ticks and generating properly spaced labels. Be careful diving into this topic … stick with what I have shown you … every one of these graphs is produced using a few select function calls. Avoid all the internet advice that has you working with figures and axes … they are not needed to produce these graphs. The only functions I use are:
plt.scatter()
plt.bar()
plt.colorbar()
plt.xlabel()
plt.ylabel()
plt.title()
plt.suptitle()
plt.show()

The most important thing is to select the data carefully. The data that represents the values for each axis is a simple list. The only place where you might be concerned about using numpy is where you want to represent a missing value.

This program will challenge your debugging skills and a good IDE can save you a lot of time. I use spyder because I like to use the "variable explorer" and breakpoints instead of using print() and input() statements to trace execution.







TESTING

This document (and the previous one on data selection) provide some information that you can use for testing. See details of my user interface and results below. Here are some things to pay close attention to …

I give the option to skip selection. Responding "y" moves forward to analysis using all available data. This is useful for debugging the graphics because you don't spend a lot of time going through the selection process.
Data selection involves 5 parameters: tremor types, date range, magnitude range, and location (latitude and longitude).
For each item, I report the number of records available and the range of values in the current selection. You should do the same as this information is useful in debugging your application.
Note that when providing a range, the endpoints are inclusive.
If the user enters a value outside of the range, report an error and repeat the selection sequence for the parameter. datetime objects in the data records include the time-of-day but the user is only asked for the date. This can lead to some complications in verifying that the inputs are within the data range. One way to accommodate this is to make sure the datetime object that represents the low end of the acceptable range has its time set to 00:00:00. I do this using the "replace" method like this:

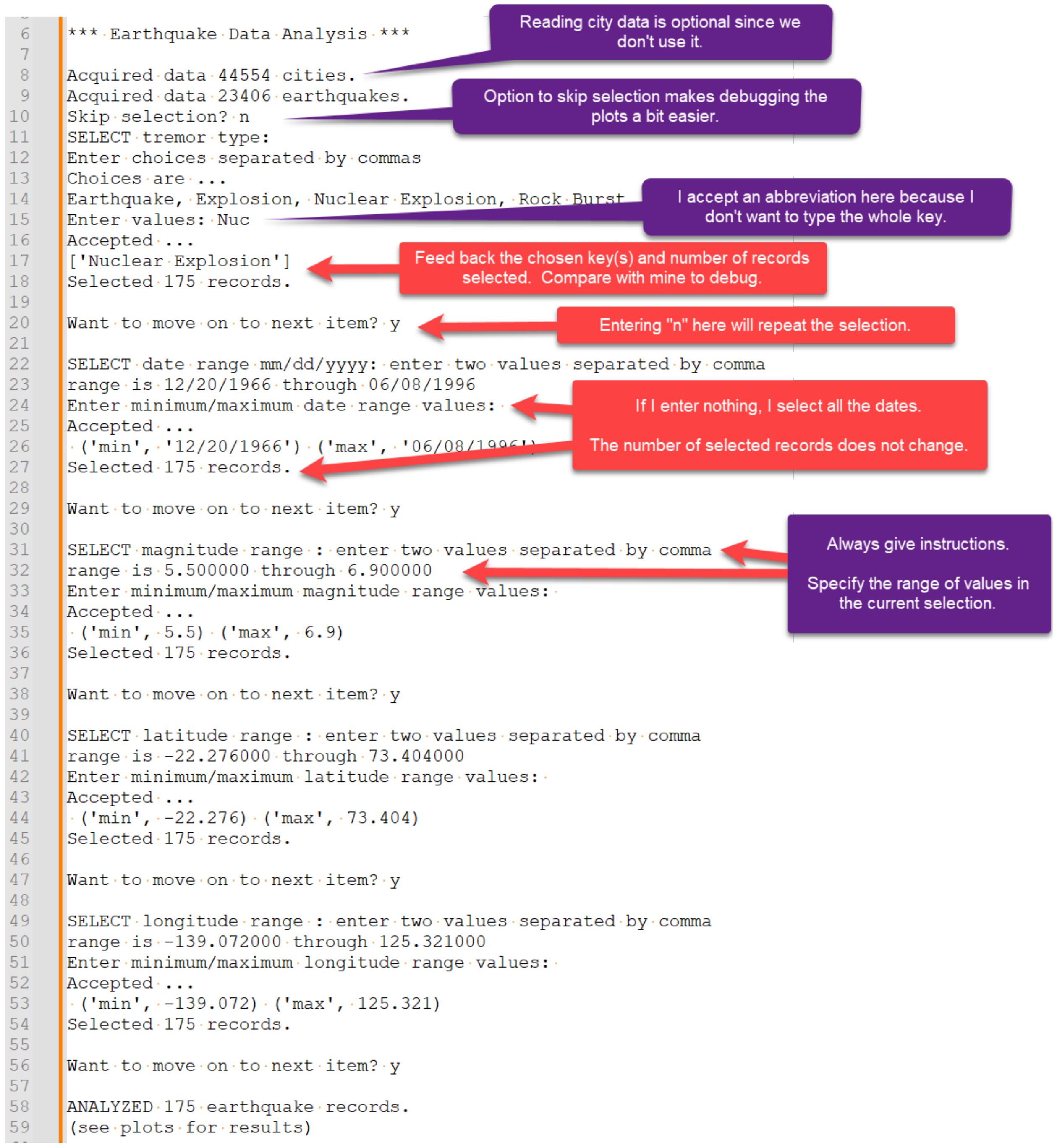
```
rng['min'] = rng['min'].replace(hour=0,minute=0,second=0)
```

Similarly, it's important to also adjust the time associated with high end like this:

```
rng['max'] = rng['max'].replace(hour=23,minute=59,second=59)
```

This adjustment should also apply to the high end of the date provided by the user. Otherwise, one would not be able to select all events for a single day.
When I ask "Move on to the next item?", I'm giving the user the option to change the currently entered and accepted parameter … "n" will repeat selection of the current parameter. There is no way to jump back to a previously defined parameter.

EXAMPLE SELECTING ALL NUCLEAR EXPLOSIONS



```
*** Earthquake Data Analysis ***

Acquired data 44554 cities.
Acquired data 23406 earthquakes.
Skip selection? n
SELECT tremor type:
Enter choices separated by commas
Choices are ...
Earthquake, Explosion, Nuclear Explosion, Rock Burst
Enter values: Nuc
Accepted ...
['Nuclear Explosion']
Selected 175 records.

Want to move on to next item? y

SELECT date range mm/dd/yyyy: enter two values separated by comma
range is 12/20/1966 through 06/08/1996
Enter minimum/maximum date range values: 
Accepted ...
 ('min', '12/20/1966') ('max', '06/08/1996')
Selected 175 records.

Want to move on to next item? y

SELECT magnitude range : enter two values separated by comma
range is 5.500000 through 6.900000
Enter minimum/maximum magnitude range values: 
Accepted ...
 ('min', 5.5) ('max', 6.9)
Selected 175 records.

Want to move on to next item? y

SELECT latitude range : enter two values separated by comma
range is -22.276000 through 73.404000
Enter minimum/maximum latitude range values: 
Accepted ...
 ('min', -22.276) ('max', 73.404)
Selected 175 records.

Want to move on to next item? y

SELECT longitude range : enter two values separated by comma
range is -139.072000 through 125.321000
Enter minimum/maximum longitude range values: 
Accepted ...
 ('min', -139.072) ('max', 125.321)
Selected 175 records.

Want to move on to next item? y

ANALYZED 175 earthquake records.
(see plots for results)
```

Reading city data is optional since we don't use it.

Option to skip selection makes debugging the plots a bit easier.

I accept an abbreviation here because I don't want to type the whole key.

Feed back the chosen key(s) and number of records selected. Compare with mine to debug.

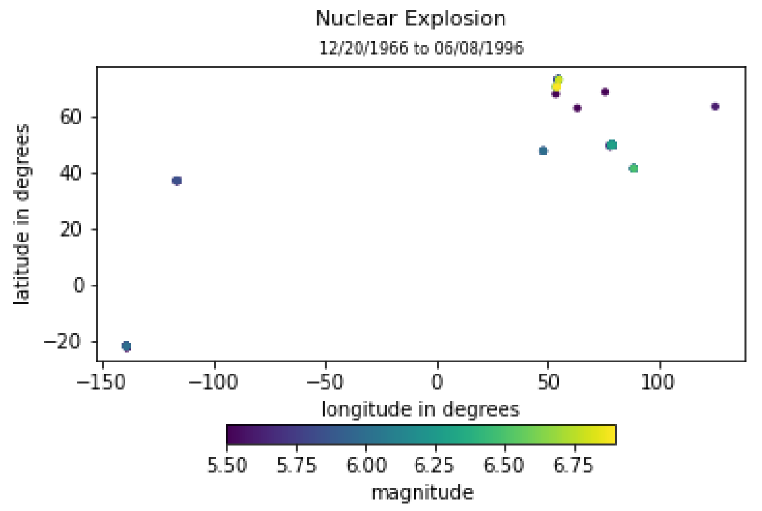
Entering "n" here will repeat the selection.

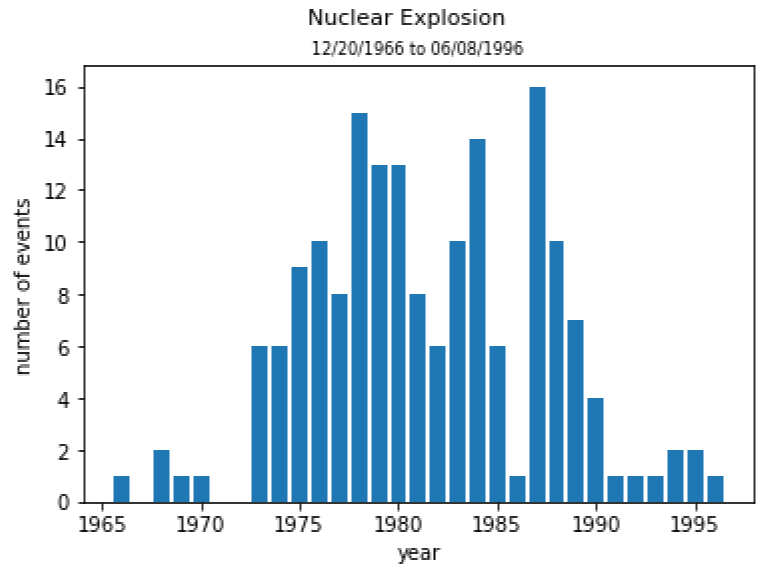
If I enter nothing, I select all the dates.

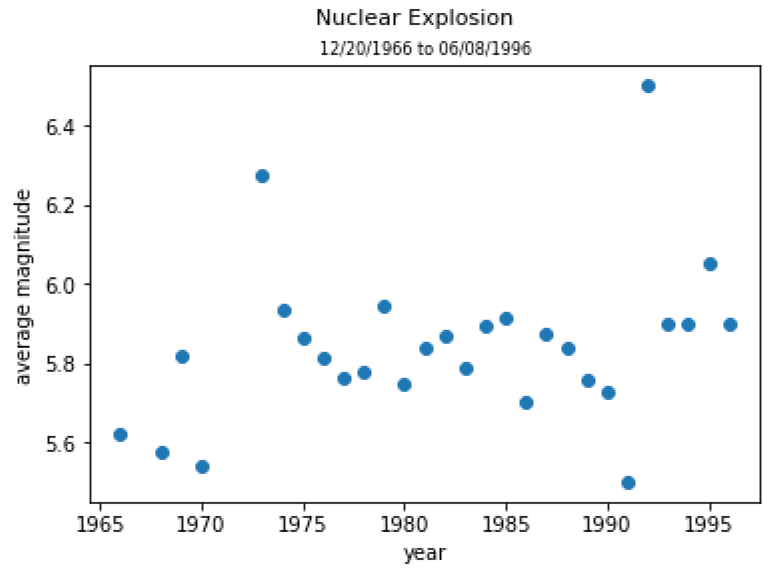
The number of selected records does not change.

Always give instructions.

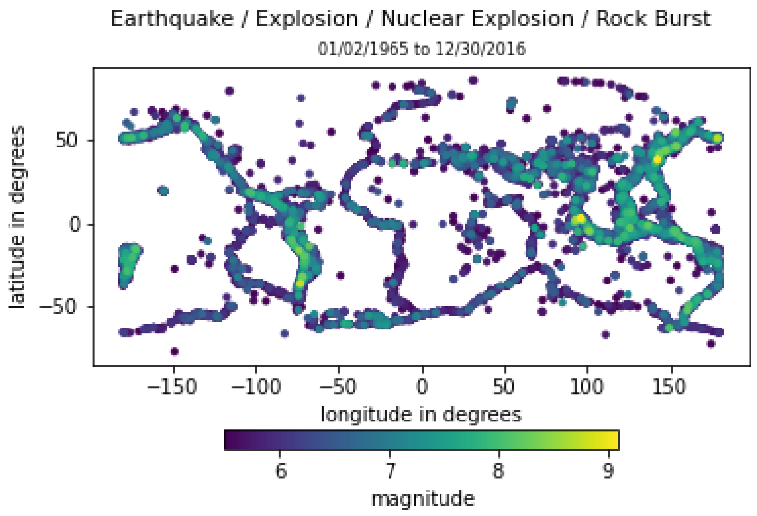
Specify the range of values in the current selection.

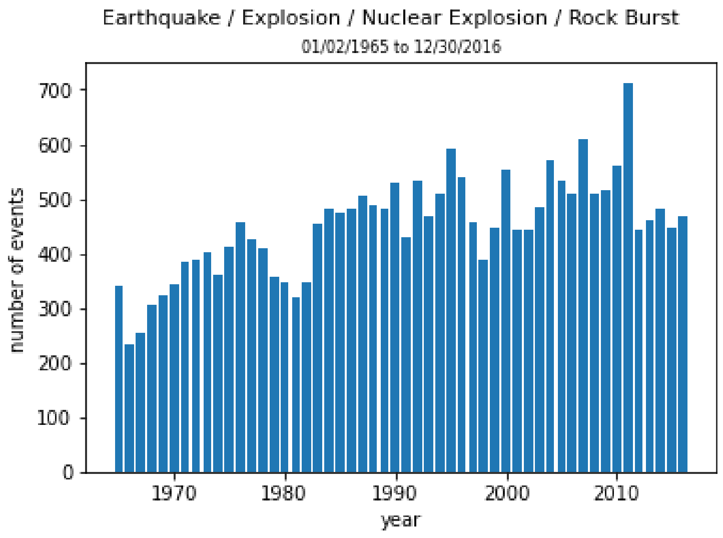

Nuclear Explosion
12/20/1966 to 06/08/1996
latitude in degrees
60
40
20
0
−20
−150
−100
−50
0
50
100
longitude in degrees
5.50
5.75
6.00
6.25
6.50
6.75
magnitude

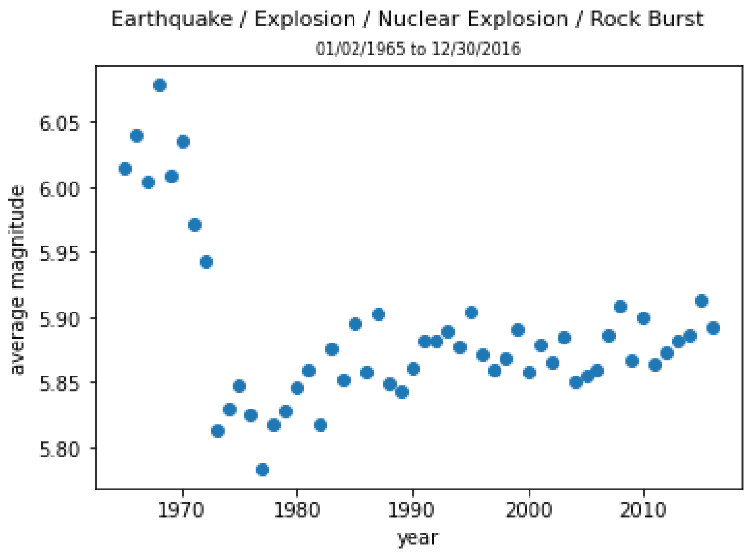

Nuclear Explosion
12/20/1966 to 06/08/1996
number of events
16
14
12
10
8
6
4
2
0
1965
1970
1975
1980
1985
1990
1995
year


Nuclear Explosion
12/20/1966 to 06/08/1996
average magnitude
6.4
6.2
6.0
5.8
5.6
1965
1970
1975
1980
1985
1990
1995
year

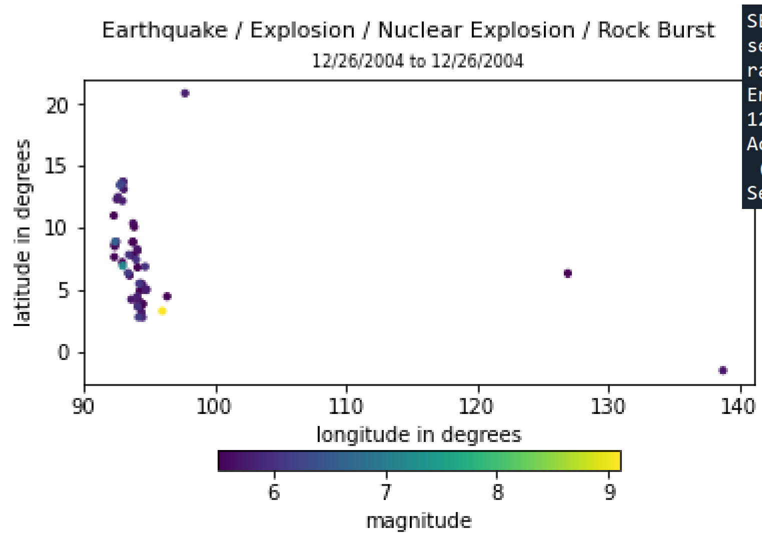
EXAMPLE SELECTING ALL DATA (skip selection, 23,406 records selected)

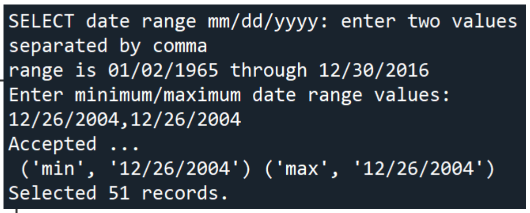






SELECTING A SINGLE DAY (12/26/2004, 51 seismic events)



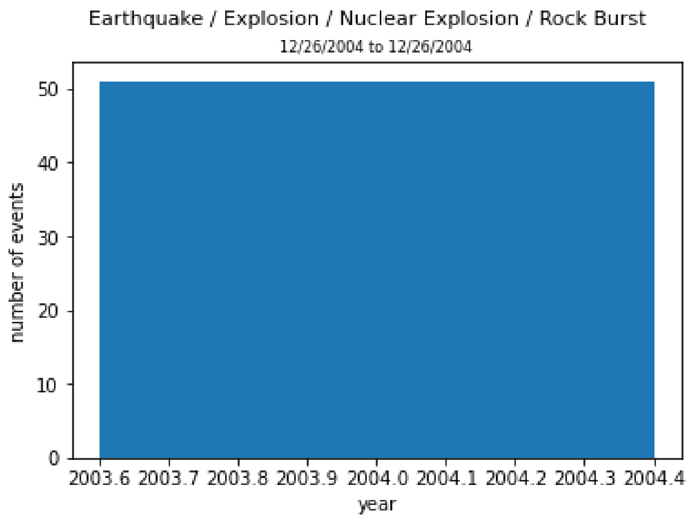


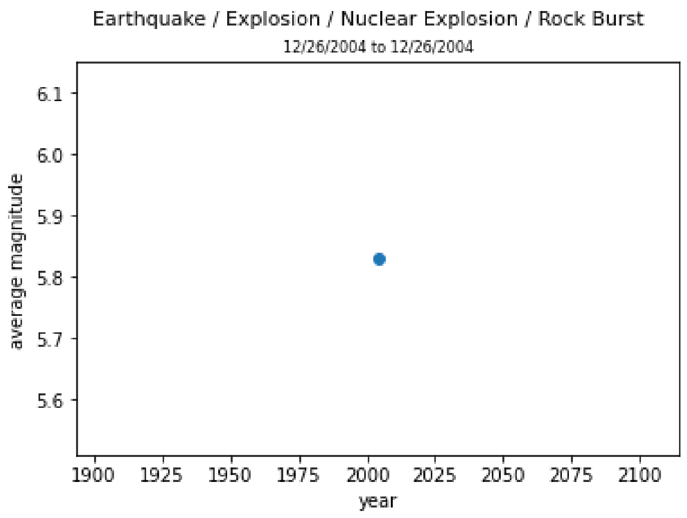
```
SELECT date range mm/dd/yyyy: enter two values
separated by comma
range is 01/02/1965 through 12/30/2016
Enter minimum/maximum date range values:
12/26/2004,12/26/2004
Accepted ...
 ('min', '12/26/2004') ('max', '12/26/2004')
Selected 51 records.
```

Note that for a single day, the graphs to the right have very little information.

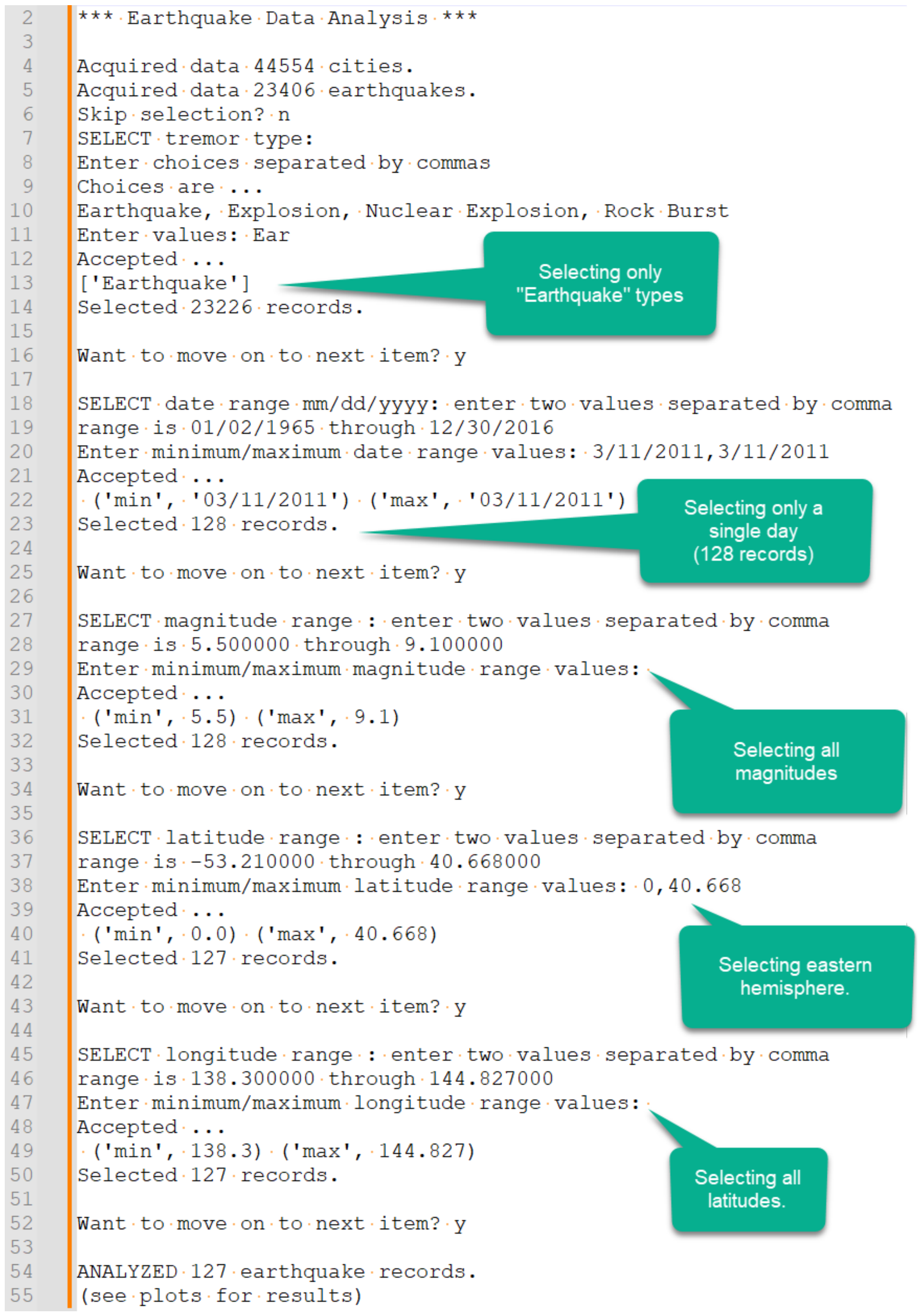
One possible embellishment might be to change the x-axis to data aggregated over hours instead of years when events are restricted to a single day.

Another very bad single day was 3/11/2011. You can see that on the next page with location restricted to north latitudes and eastern longitudes.

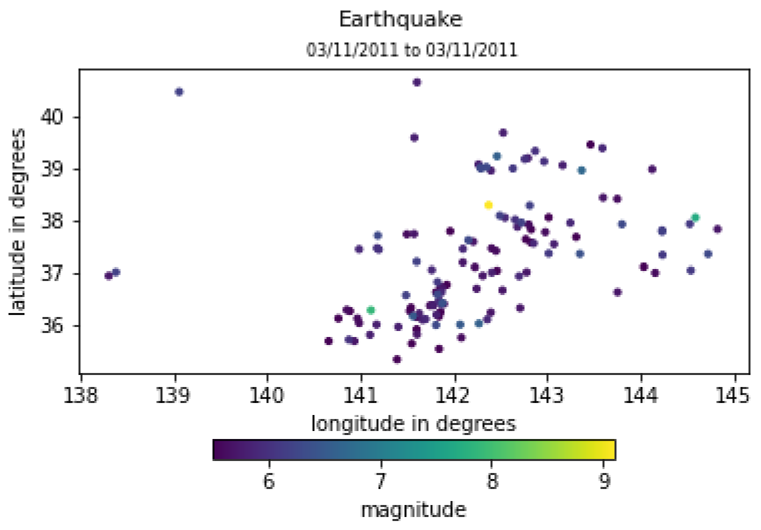


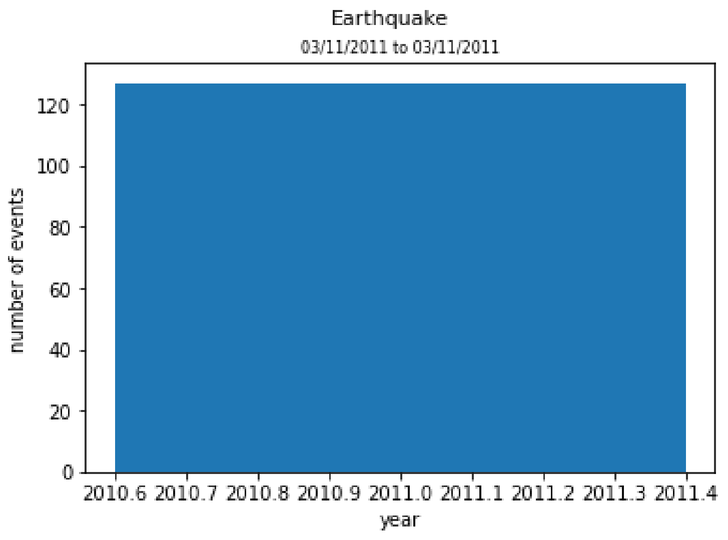


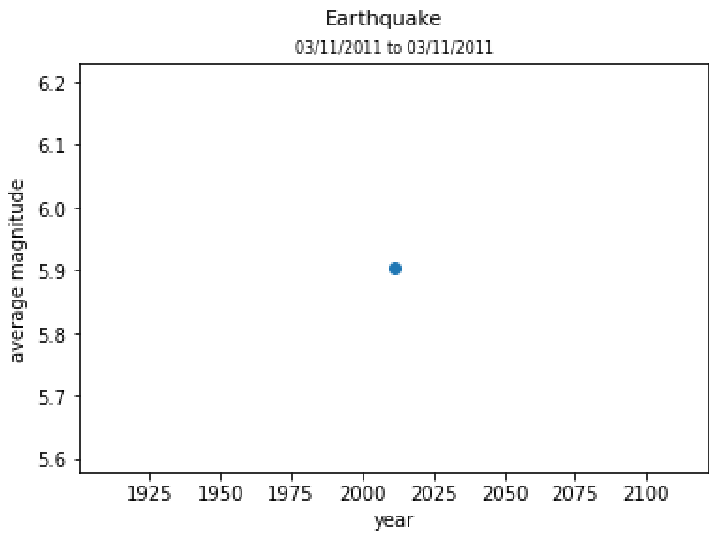
EXAMPLE: SINGLE DAY (3/11/2011, 128 events)


Selecting only "Earthquake" types
Selecting only a single day (128 records)
Selecting all magnitudes
Selecting eastern hemisphere.
Selecting all latitudes.

```
*** Earthquake Data Analysis ***

Acquired data 44554 cities.
Acquired data 23406 earthquakes.
Skip selection? n
SELECT tremor type:
Enter choices separated by commas
Choices are ...
Earthquake, Explosion, Nuclear Explosion, Rock Burst
Enter values: Ear
Accepted ...
['Earthquake']
Selected 23226 records.

Want to move on to next item? y

SELECT date range mm/dd/yyyy: enter two values separated by comma
range is 01/02/1965 through 12/30/2016
Enter minimum/maximum date range values: 3/11/2011,3/11/2011
Accepted ...
 ('min', '03/11/2011') ('max', '03/11/2011')
Selected 128 records.

Want to move on to next item? y

SELECT magnitude range : enter two values separated by comma
range is 5.500000 through 9.100000
Enter minimum/maximum magnitude range values: 
Accepted ...
 ('min', 5.5) ('max', 9.1)
Selected 128 records.

Want to move on to next item? y

SELECT latitude range : enter two values separated by comma
range is -53.210000 through 40.668000
Enter minimum/maximum latitude range values: 0,40.668
Accepted ...
 ('min', 0.0) ('max', 40.668)
Selected 127 records.

Want to move on to next item? y

SELECT longitude range : enter two values separated by comma
range is 138.300000 through 144.827000
Enter minimum/maximum longitude range values: 
Accepted ...
 ('min', 138.3) ('max', 144.827)
Selected 127 records.

Want to move on to next item? y

ANALYZED 127 earthquake records.
(see plots for results)
```


Earthquake
03/11/2011 to 03/11/2011
latitude in degrees
36
37
38
39
40
138
139
140
141
142
143
144
145
longitude in degrees
6
7
8
9
magnitude


Earthquake
03/11/2011 to 03/11/2011
number of events
0
20
40
60
80
100
120
2010.6
2010.7
2010.8
2010.9
2011.0
2011.1
2011.2
2011.3
2011.4
year


Earthquake
03/11/2011 to 03/11/2011
average magnitude
5.6
5.7
5.8
5.9
6.0
6.1
6.2
1925
1950
1975
2000
2025
2050
2075
2100
year

FINAL THOUGHTS

Stumbling blocks. This project was developed to help you prepare for future data analysis experiences using specialized tools. In developing this application, you are likely to encounter many small (but significant) issues along the way. The idea of selecting and aggregating data is relatively easy to understand but the implementation has some stumbling blocks due to the variety of data types. There are classifications (key="Type" … Earthquake, Explosion, Nuclear Explosion, Rock Burst), datetime objects, and floating point numbers (Magnitude, latitude, longitude). I expect that you will contact me if you come across something that seems incorrect as it is possible that there is something that I have not accounted for in my implementation. I welcome these interactions.

Evaluation. All submissions will be graded manually. There is no autograder to identify where your errors are. The data selection process can be programmed in many ways. Your specific coding approach will most likely be different than mine. When I look at your running submission I will be looking for the interface to be relatively flexible and for your results to look "close to" mine. What does "close to" mean? Selected records ought to be +/- 1. Calculated values ought to be within 1%. When I look at your code, I expect to see helpful docstrings for functions (as I have modeled them for you) and comments that describe what each block of code intends to do. In other words, your code should convince me that you understand what you are doing.

Future directions. In this course, I stress Python fundamentals and problem-solving. With a firm grasp of the fundamentals you have what you need to learn more advanced topics on your own. There's a lot to know, but everything begins by having a fundamental vocabulary to work with.