NYC Taxi Project

Harrison Schwarzer
Clarkson University
schwarhd@clarkson.edu

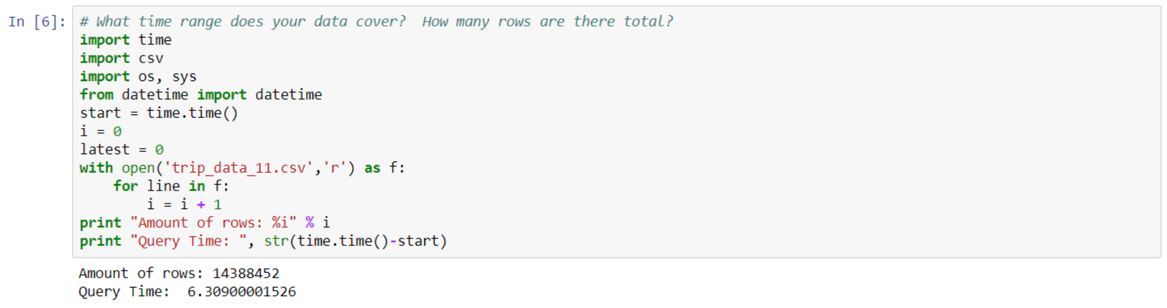
Exploration of a 'large' (15 million rows) dataset using Python. The following ten questions are answered:

- What time range does your data cover? How many rows are there total?
- What are the field names?
- Give some sample data for each field.
- What MySQL data types would you need to store each of the fields?
- What is the geographic range of your data (min/max - X/Y)? Plot this (approximately on a map)
- What are the distinct values for each field? (If applicable)
- For other numeric types besides lat and lon, what are the min and max values?
- Create a chart which shows the average number of passengers each hour of the day.
- Create a new CSV file which has only one out of every thousand rows.
- Repeat step 8 with the reduced dataset and compare the two charts.

I've included a number of code snapshots in this report. This code, however, is not production-ready. Much of it is uncommented and includes a number of print statements for debugging. Full code descriptions are included in each section - that's what's worth reading.

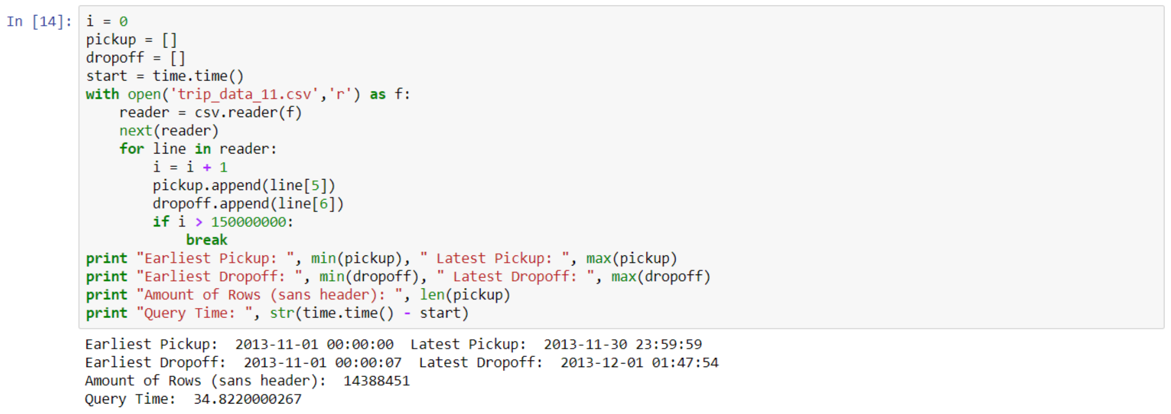
1. What time range does your data cover? How many rows are there total?

The best way to count the number of rows in a file would be to simply run a counter that increments over each line of the file. The code below does that, and tells how much time (about 6 seconds) it takes to read through the entire file.



```
In [6]: # What time range does your data cover?  How many rows are there total?
        import time
        import csv
        import os, sys
        from datetime import datetime
        start = time.time()
        i = 0
        latest = 0
        with open('trip_data_11.csv','r') as f:
            for line in f:
                i = i + 1
        print "Amount of rows: %i" % i
        print "Query Time: ", str(time.time()-start)

        Amount of rows: 14388452
        Query Time:  6.30900001526
```

According to StackExchange, this is the cheapest way to count a CSV. But, since we want to find the time range, we can hack together something that's far less scalable, but will answer our question really quickly.



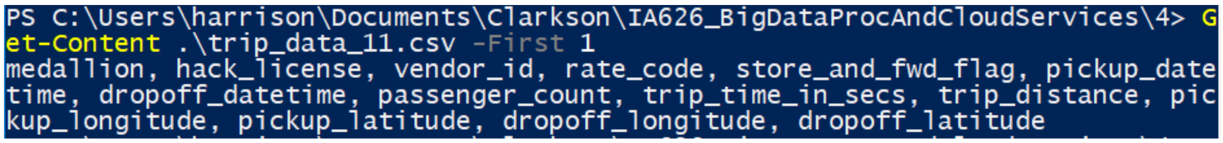
```
In [14]: i = 0
         pickup = []
         dropoff = []
         start = time.time()
         with open('trip_data_11.csv','r') as f:
             reader = csv.reader(f)
             next(reader)
             for line in reader:
                 i = i + 1
                 pickup.append(line[5])
                 dropoff.append(line[6])
                 if i > 150000000:
                     break
         print "Earliest Pickup: ", min(pickup), " Latest Pickup: ", max(pickup)
         print "Earliest Dropoff: ", min(dropoff), " Latest Dropoff: ", max(dropoff)
         print "Amount of Rows (sans header): ", len(pickup)
         print "Query Time: ", str(time.time() - start)

         Earliest Pickup:  2013-11-01 00:00:00  Latest Pickup:  2013-11-30 23:59:59
         Earliest Dropoff:  2013-11-01 00:00:07  Latest Dropoff:  2013-12-01 01:47:54
         Amount of Rows (sans header):  14388451
         Query Time:  34.8220000267
```

The code above writes the pickup and dropoff times into separate lists, then uses the built in $min()$ and $max()$ functions to find the earliest and latest dates. To count the total number of rows, you can just take the $len()$ of one of the lists. This takes about six times as long to run as the code above. Writing a date-parser which only saves one object in memory would be the ideal way to do things.

The earliest date in this data set is *00:00:00 November 1st, 2013*, while the latest is *01:47:54 December 1st, 2013*.

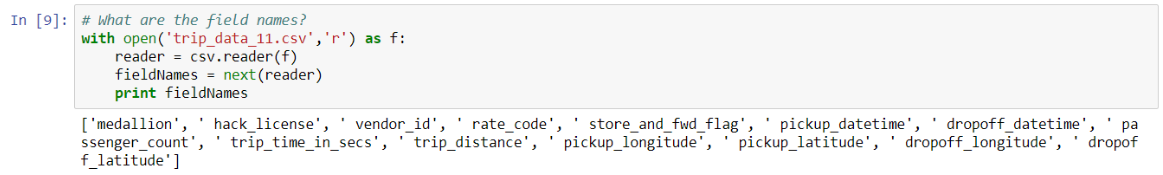
2. What are the field names?

You can find this out just from the shell:



```
PS C:\Users\harrison\Documents\Clarkson\IA626_BigDataProcAndCloudServices\4> Get-Content .\trip_data_11.csv -First 1
medallion, hack_license, vendor_id, rate_code, store_and_fwd_flag, pickup_datetime, dropoff_datetime, passenger_count, trip_time_in_secs, trip_distance, pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude
```

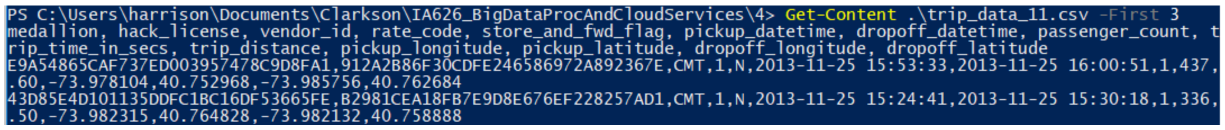
Or you can do it in Python using the $next()$ command.



```
In [9]: # What are the field names?
        with open('trip_data_11.csv','r') as f:
            reader = csv.reader(f)
            fieldNames = next(reader)
            print fieldNames

['medallion', ' hack_license', ' vendor_id', ' rate_code', ' store_and_fwd_flag', ' pickup_datetime', ' dropoff_datetime', ' pa
ssenger_count', ' trip_time_in_secs', ' trip_distance', ' pickup_longitude', ' pickup_latitude', ' dropoff_longitude', ' dropof
f_latitude']
```

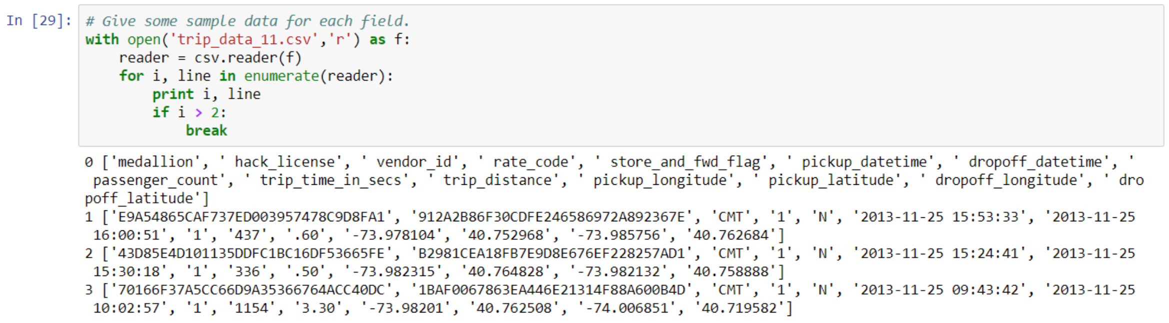
3. Give some sample data for each field.

Again, you can do this in shell:



```
PS C:\Users\harrison\Documents\Clarkson\IA626_BigDataProcAndCloudServices\4> Get-Content .\trip_data_11.csv -First 3
medallion, hack_license, vendor_id, rate_code, store_and_fwd_flag, pickup_datetime, dropoff_datetime, passenger_count, t
rip_time_in_secs, trip_distance, pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude
E9A54865CAF737ED003957478C9D8FA1,912A2B86F30CDFE246586972A892367E,CMT,1,N,2013-11-25 15:53:33,2013-11-25 16:00:51,1,437,
.60,-73.978104,40.752968,-73.985756,40.762684
43D85E4D101135DDFC1BC16DF53665FE,B2981CEA18FB7E9D8E676EF228257AD1,CMT,1,N,2013-11-25 15:24:41,2013-11-25 15:30:18,1,336,
.50,-73.982315,40.764828,-73.982132,40.758888
```

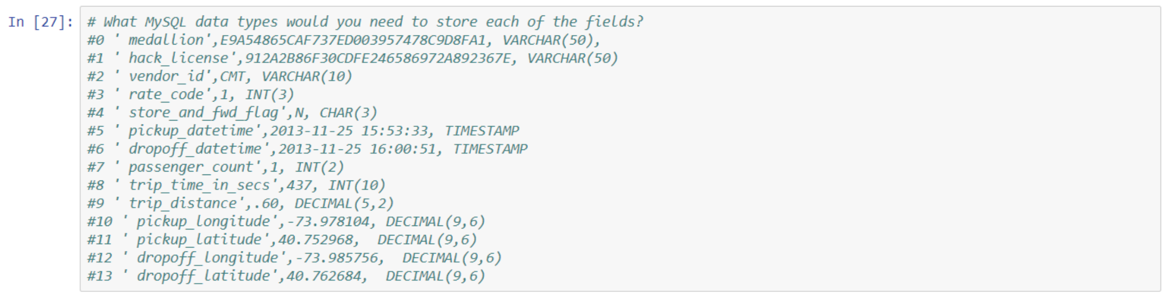
Or you can do it in Python



```
In [29]: # Give some sample data for each field.
         with open('trip_data_11.csv','r') as f:
             reader = csv.reader(f)
             for i, line in enumerate(reader):
                 print i, line
                 if i > 2:
                     break

0 ['medallion', ' hack_license', ' vendor_id', ' rate_code', ' store_and_fwd_flag', ' pickup_datetime', ' dropoff_datetime', '
 passenger_count', ' trip_time_in_secs', ' trip_distance', ' pickup_longitude', ' pickup_latitude', ' dropoff_longitude', ' dro
poff_latitude']
1 ['E9A54865CAF737ED003957478C9D8FA1', '912A2B86F30CDFE246586972A892367E', 'CMT', '1', 'N', '2013-11-25 15:53:33', '2013-11-25
 16:00:51', '1', '437', '.60', '-73.978104', '40.752968', '-73.985756', '40.762684']
2 ['43D85E4D101135DDFC1BC16DF53665FE', 'B2981CEA18FB7E9D8E676EF228257AD1', 'CMT', '1', 'N', '2013-11-25 15:24:41', '2013-11-25
 15:30:18', '1', '336', '.50', '-73.982315', '40.764828', '-73.982132', '40.758888']
3 ['70166F37A5CC66D9A35366764ACC40DC', '1BAF0067863EA446E21314F88A600B4D', 'CMT', '1', 'N', '2013-11-25 09:43:42', '2013-11-25
 10:02:57', '1', '1154', '3.30', '-73.98201', '40.762508', '-74.006851', '40.719582']
```

4. What MySQL data types would you need to store each of the fields?

I wrote this into a big commented section so I could keep referring back to it. A couple of fields here are a tad off. Here's the translation:



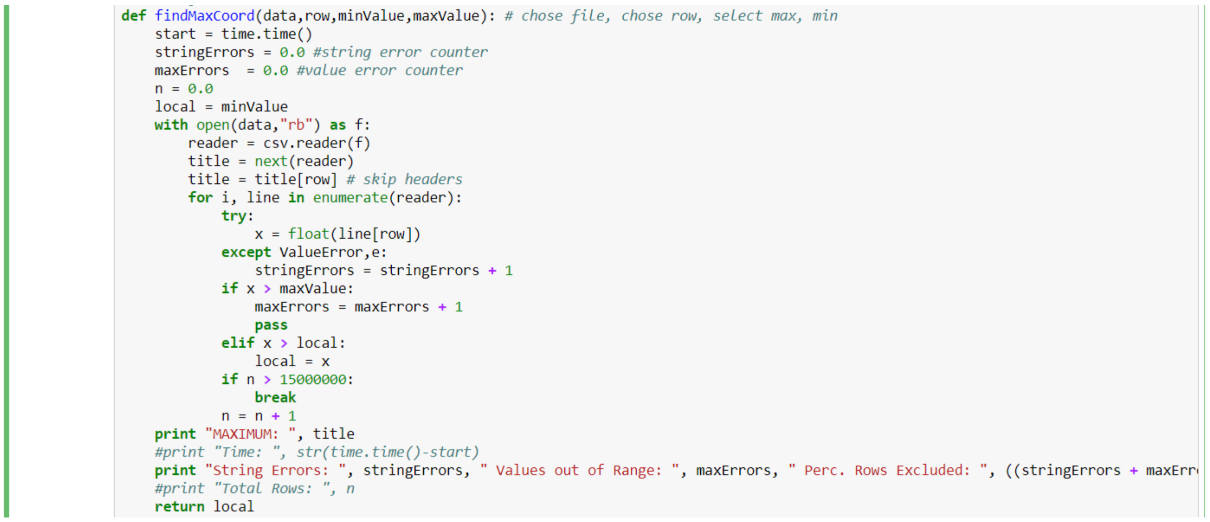
```
In [27]: # What MySQL data types would you need to store each of the fields?
         #0 ' medallion',E9A54865CAF737ED003957478C9D8FA1, VARCHAR(50),
         #1 ' hack_license',912A2B86F30CDFE246586972A892367E, VARCHAR(50)
         #2 ' vendor_id',CMT, VARCHAR(10)
         #3 ' rate_code',1, INT(3)
         #4 ' store_and_fwd_flag',N, CHAR(3)
         #5 ' pickup_datetime',2013-11-25 15:53:33, TIMESTAMP
         #6 ' dropoff_datetime',2013-11-25 16:00:51, TIMESTAMP
         #7 ' passenger_count',1, INT(2)
         #8 ' trip_time_in_secs',437, INT(10)
         #9 ' trip_distance',.60, DECIMAL(5,2)
         #10 ' pickup_longitude',-73.978104, DECIMAL(9,6)
         #11 ' pickup_latitude',40.752968,  DECIMAL(9,6)
         #12 ' dropoff_longitude',-73.985756,  DECIMAL(9,6)
         #13 ' dropoff_latitude',40.762684,  DECIMAL(9,6)
```

Here's the translation:

- 0 medallion, VARCHAR(50)
- 1 hack_license, VARCHAR(50)
- 2 vendor_id, VARCHAR(10)
- 3 rate_code, INT(3)
- 4 store_and_fwd_flag, CHAR(3)
- 5 pickup_datetime, TIMESTAMP
- 6 dropoff_datetime, TIMESTAMP
- 7 passenger_count, INT(3)
- 8 trip_time_in_secs, INT(6)
- 9 trip_distance, DECIMAL(5,2)
- 10 pickup_longitude, DECIMAL(9,6)
- 11 pickup_latitude, DECIMAL(9,6)
- 12 dropoff_longitude, DECIMAL(9,6)
- 13 dropoff_latitude, DECIMAL(9,6)

5. What is the geographic range of your data (min/max - X/Y)? Plot this (approximately on a map)

This problem ended up yielding some pretty boring results. As it turns out, the Geographic ranges in use are basically all of North America. I'll walk you through my process of disappointment.



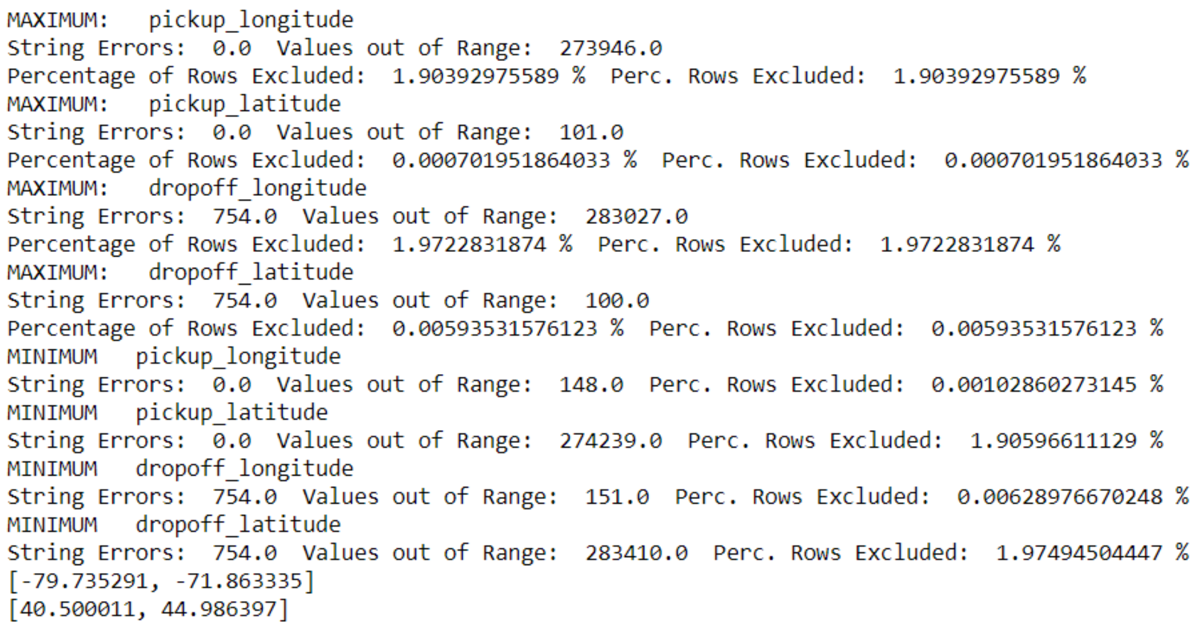
```
def findMaxCoord(data,row,minValue,maxValue): # chose file, chose row, select max, min
    start = time.time()
    stringErrors = 0.0 #string error counter
    maxErrors  = 0.0 #value error counter
    n = 0.0
    local = minValue
    with open(data,"rb") as f:
        reader = csv.reader(f)
        title = next(reader)
        title = title[row] # skip headers
        for i, line in enumerate(reader):
            try:
                x = float(line[row])
            except ValueError,e:
                stringErrors = stringErrors + 1
            if x > maxValue:
                maxErrors = maxErrors + 1
                pass
            elif x > local:
                local = x
            if n > 15000000:
                break
            n = n + 1
    print "MAXIMUM: ", title
    #print "Time: ", str(time.time()-start)
    print "String Errors: ", stringErrors, " Values out of Range: ", maxErrors, " Perc. Rows Excluded: ", ((stringErrors + maxErr
    #print "Total Rows: ", n
    return local
```

Figure 1: Find Maximum in a Specified Range Function

This function iterates through a specified row of data and finds the maximum value within a specified range. For example, if you were to feed the Natural Numbers through this function and specify min 0, max 100, if would return the value 99. I wrote a similar function, taking the same arguments, that finds the minimum of a specified range. This function also prints the number of string errors (values that can't be converted to floats) and value errors (values that are outside the specified range).

At first, I specified the min and max values using the maximum geographic coordinates of New York State[1].

The function returned the following:

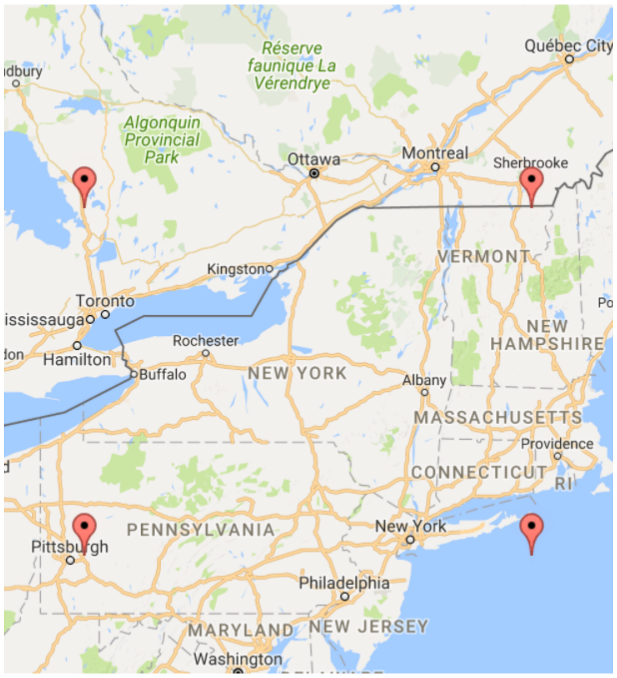


```
MAXIMUM:   pickup_longitude
String Errors:  0.0  Values out of Range:  273946.0
Percentage of Rows Excluded:  1.90392975589 %  Perc. Rows Excluded:  1.90392975589 %
MAXIMUM:   pickup_latitude
String Errors:  0.0  Values out of Range:  101.0
Percentage of Rows Excluded:  0.000701951864033 %  Perc. Rows Excluded:  0.000701951864033 %
MAXIMUM:   dropoff_longitude
String Errors:  754.0  Values out of Range:  283027.0
Percentage of Rows Excluded:  1.9722831874 %  Perc. Rows Excluded:  1.9722831874 %
MAXIMUM:   dropoff_latitude
String Errors:  754.0  Values out of Range:  100.0
Percentage of Rows Excluded:  0.00593531576123 %  Perc. Rows Excluded:  0.00593531576123 %
MINIMUM   pickup_longitude
String Errors:  0.0  Values out of Range:  148.0  Perc. Rows Excluded:  0.00102860273145 %
MINIMUM   pickup_latitude
String Errors:  0.0  Values out of Range:  274239.0  Perc. Rows Excluded:  1.90596611129 %
MINIMUM   dropoff_longitude
String Errors:  754.0  Values out of Range:  151.0  Perc. Rows Excluded:  0.00628976670248 %
MINIMUM   dropoff_latitude
String Errors:  754.0  Values out of Range:  283410.0  Perc. Rows Excluded:  1.97494504447 %
[-79.735291, -71.863335]
[40.500011, 44.986397]
```

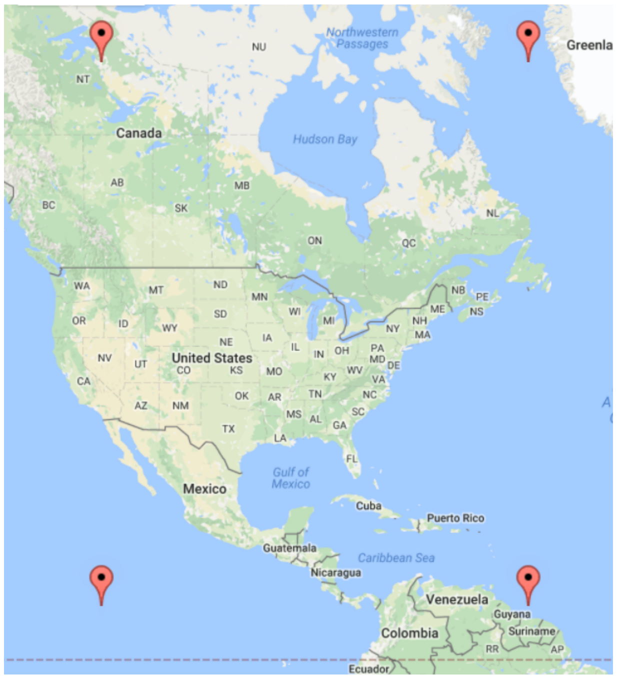
Min, Max Long: (-79.73,-71.86)
Min, Max Lat: (40.50,44.99)

[1]`http://en.wikipedia.org/wiki/List_of_extreme_points_of_U.S._States`

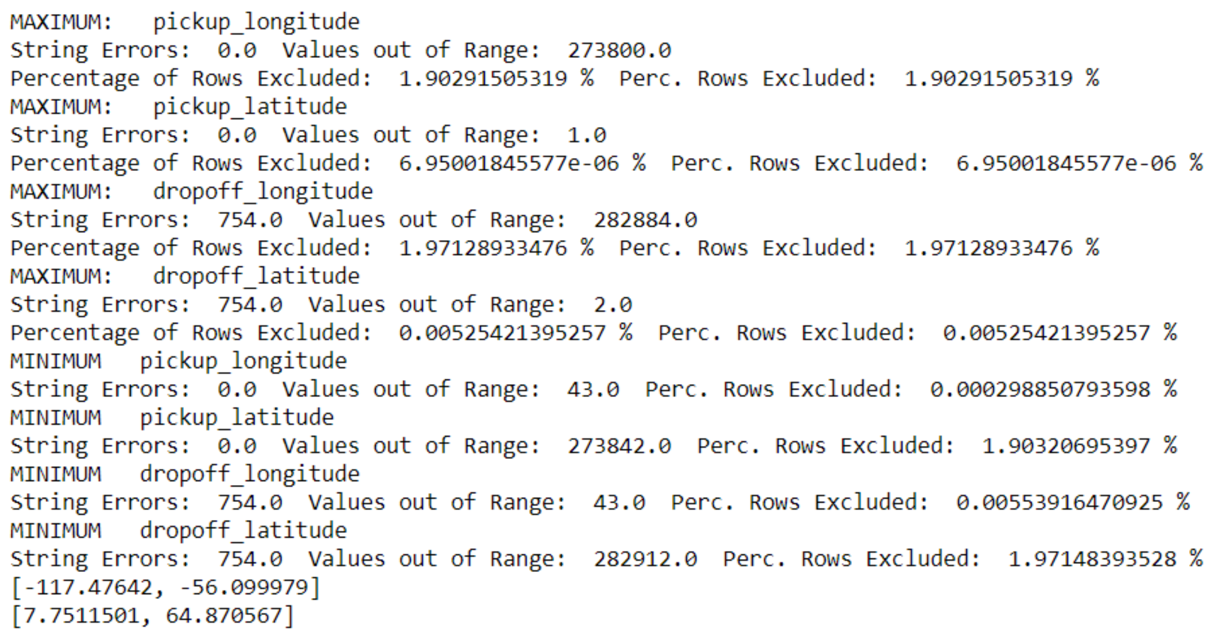
The plot of these extremes are basically the extreme points of New York state:



This function was excluding about 2% of data in some cases. That made me suspect that I needed to widen the range. So, I ran this again, using the geographic extremes of North America. It yielded the following plot:



Well, that's just as crappy. Turns out now the extremes are just the extremes of North America. So, I looked at the output and I saw that in some cases, about 2% of the data was still being omitted.


```
MAXIMUM:   pickup_longitude
String Errors:  0.0  Values out of Range:  273800.0
Percentage of Rows Excluded:  1.90291505319 %  Perc. Rows Excluded:  1.90291505319 %
MAXIMUM:   pickup_latitude
String Errors:  0.0  Values out of Range:  1.0
Percentage of Rows Excluded:  6.95001845577e-06 %  Perc. Rows Excluded:  6.95001845577e-06 %
MAXIMUM:   dropoff_longitude
String Errors:  754.0  Values out of Range:  282884.0
Percentage of Rows Excluded:  1.97128933476 %  Perc. Rows Excluded:  1.97128933476 %
MAXIMUM:   dropoff_latitude
String Errors:  754.0  Values out of Range:  2.0
Percentage of Rows Excluded:  0.00525421395257 %  Perc. Rows Excluded:  0.00525421395257 %
MINIMUM   pickup_longitude
String Errors:  0.0  Values out of Range:  43.0  Perc. Rows Excluded:  0.000298850793598 %
MINIMUM   pickup_latitude
String Errors:  0.0  Values out of Range:  273842.0  Perc. Rows Excluded:  1.90320695397 %
MINIMUM   dropoff_longitude
String Errors:  754.0  Values out of Range:  43.0  Perc. Rows Excluded:  0.00553916470925 %
MINIMUM   dropoff_latitude
String Errors:  754.0  Values out of Range:  282912.0  Perc. Rows Excluded:  1.97148393528 %
[-117.47642, -56.099979]
[7.7511501, 64.870567]
```

Min, Max Long: (-117.48,-56.10)
Min, Max Lat: (7.75,64.87)

What's likely happening is the the GPS in some of these taxis are just off. Maybe they're set to 0 degrees, or some other initial value that's outside our range. In order to really find out what the extremes are, you'd have to write something a little smarter, which dips into the other columns to see if the numbers are legitimate[2]. For example, you could look at the trip lengths[3], to see if people are actually driving from, say, Venezuela, or you could look at the complementary coordinate and see if trips are originating from the middle of the Atlantic.

6. What are the distinct values for each field? (If applicable)

This function runs through a column and writes the unique values into a dictionary. The user specifies the csv, the column, and the cardinality tolerance. It checks the cardinality of the dictionary once every 10000 rows. If the column has, say, 1 million unique values, it will print ”Over Threshold” and exit.

[2]See end of project

[3]The longest trip length, according to my function with a maximum of 1 trillion seconds (3̃0000 years) was only 3 hours. I suspect a lot of the data is straight bunk.

```
def cardinality(data,row,threshold):
    dict = {}
    with open(data,"rb") as f:
        reader = csv.reader(f)
        title = next(reader)
        title = title[row] # skip headers
        for i, line in enumerate(reader):
            if line[row] in dict.keys():
                dict[line[row]] = dict[line[row]] + 1
            else:
                dict[line[row]] = 1
            if i % 10000 == 0:
                if len(dict) > threshold:
                    print "OVER THRESHOLD"
                    break
            if i > 15000000:
                break
    print "Column: ", title
    return dict
```

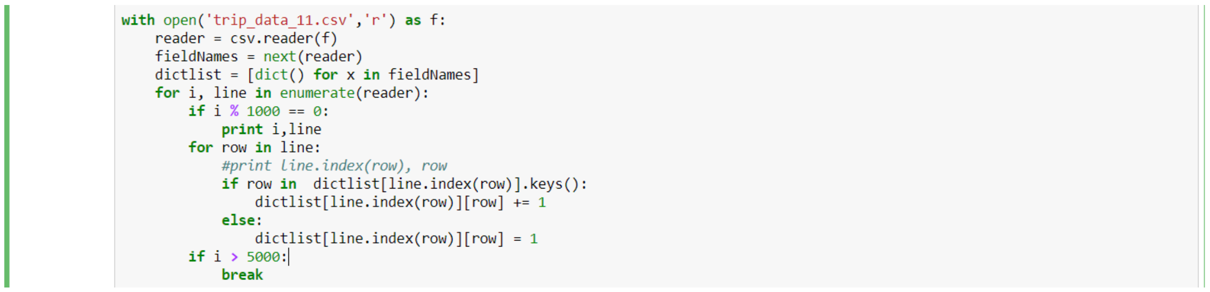
I then ran it over each column in the dataset, and returned the dictionary for ever field with fewer than 100 unique values.

```
for i in range(14):
    x = cardinality('trip_data_11.csv',i,1000)
    if len(x) < 100:
        print "Cardinality:", len(x), x.keys()
    else:
        print "Cardinality over 100: ", len(x)
```

Here's the output it gave:

```
OVER THRESHOLD
Column:  medallion
Cardinality over 100:  5275
OVER THRESHOLD
Column:   hack_license
Cardinality over 100:  5601
Column:   vendor_id
Cardinality: 2 ['VTS', 'CMT']
Column:   rate_code
Cardinality: 12 ['10', '210', '1', '0', '3', '2', '5', '4', '7', '6', '9', '8']
Column:   store_and_fwd_flag
Cardinality: 3 ['Y', '', 'N']
OVER THRESHOLD
Column:   pickup_datetime
Cardinality over 100:  1069
OVER THRESHOLD
Column:   dropoff_datetime
Cardinality over 100:  1025
Column:   passenger_count
Cardinality: 11 ['208', '1', '0', '3', '2', '5', '4', '7', '6', '9', '8']
OVER THRESHOLD
Column:   trip_time_in_secs
Cardinality over 100:  1037
OVER THRESHOLD
Column:   trip_distance
Cardinality over 100:  1276
OVER THRESHOLD
Column:   pickup_longitude
Cardinality over 100:  5746
OVER THRESHOLD
Column:   pickup_latitude
Cardinality over 100:  7615
OVER THRESHOLD
Column:   dropoff_longitude
Cardinality over 100:  6112
OVER THRESHOLD
Column:   dropoff_latitude
Cardinality over 100:  8117
```

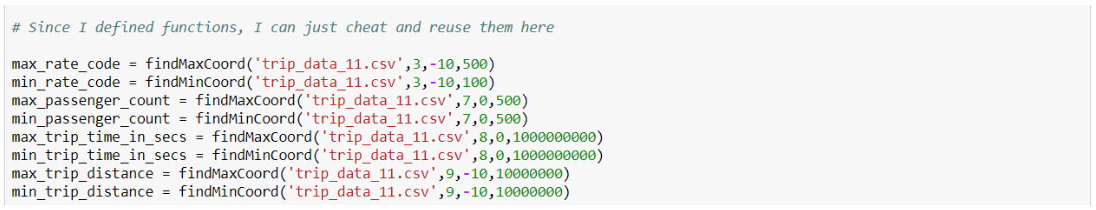
I also wrote a function which runs row-by-row through our csv and returns the distinct values. The problem with using is that you very quickly end up writing dictionaries nearly the length of their respective columns. However, this code is pretty clever about assigning different columns to dictionaries by index, and it would work well for smaller sets. I included the code for fun. Here it is:



```python
with open('trip_data_11.csv','r') as f:
    reader = csv.reader(f)
    fieldNames = next(reader)
    dictlist = [dict() for x in fieldNames]
    for i, line in enumerate(reader):
        if i % 1000 == 0:
            print i,line
        for row in line:
            #print line.index(row), row
            if row in  dictlist[line.index(row)].keys():
                dictlist[line.index(row)][row] += 1
            else:
                dictlist[line.index(row)][row] = 1
        if i > 5000:
            break
```

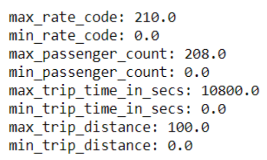
7. For other numeric types besides lat and lon, what are the min and max values?

Remember that maximum function from Figure 1? I just ran the max and min functions again on the other numeric rows, rate_code, passenger_count, trip_time, and trip_distance. It set the ranges very wide, but there are potential nonsensical outliers. Here's are the function calls with the min and max ranges:



```python
# Since I defined functions, I can just cheat and reuse them here

max_rate_code = findMaxCoord('trip_data_11.csv',3,-10,500)
min_rate_code = findMinCoord('trip_data_11.csv',3,-10,100)
max_passenger_count = findMaxCoord('trip_data_11.csv',7,0,500)
min_passenger_count = findMinCoord('trip_data_11.csv',7,0,500)
max_trip_time_in_secs = findMaxCoord('trip_data_11.csv',8,0,1000000000)
min_trip_time_in_secs = findMinCoord('trip_data_11.csv',8,0,1000000000)
max_trip_distance = findMaxCoord('trip_data_11.csv',9,-10,10000000)
min_trip_distance = findMinCoord('trip_data_11.csv',9,-10,10000000)
```

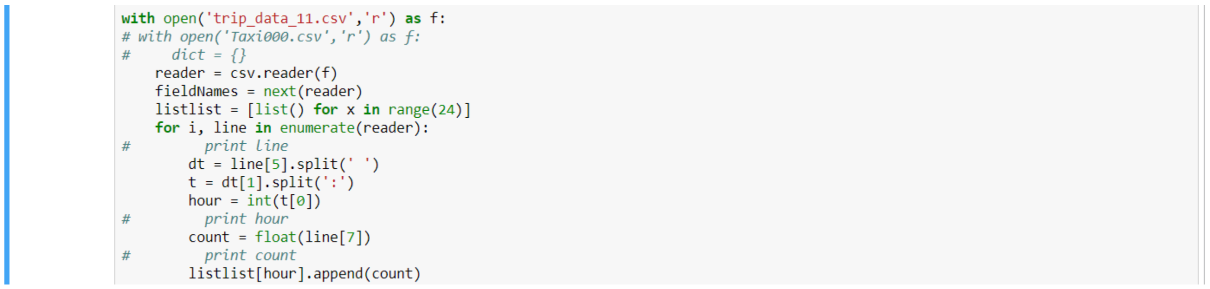
And here's the output:



```
max_rate_code: 210.0
min_rate_code: 0.0
max_passenger_count: 208.0
min_passenger_count: 0.0
max_trip_time_in_secs: 10800.0
min_trip_time_in_secs: 0.0
max_trip_distance: 100.0
min_trip_distance: 0.0
```

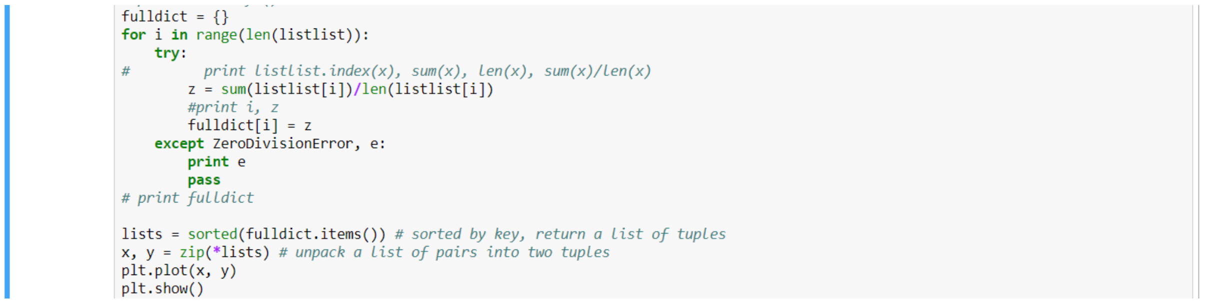
8. Create a chart which shows the average number of passengers each hour of the day.

Initialize a list of lists with 24 entries. Go to the 'pickup_time' column, split on whitespace, then split on colons. Select the first entry to identify hour, then convert it to an int. Use that integer to specify the index of your list. Append the passenger count of that row to that list.



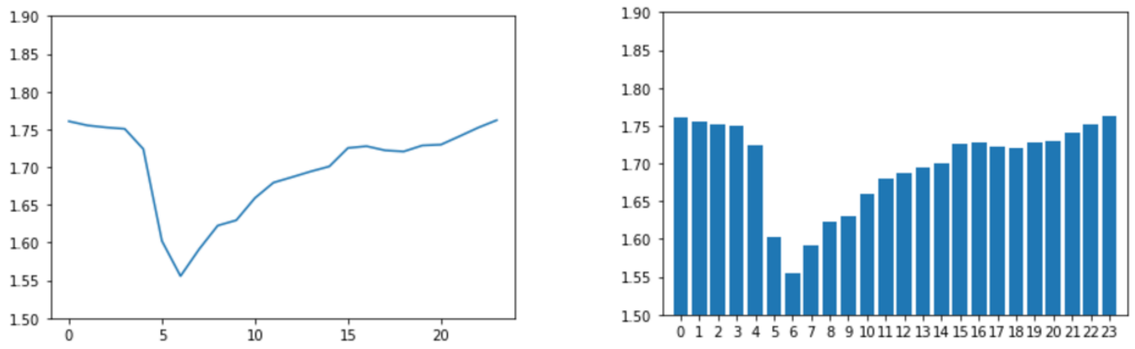
```
with open('trip_data_11.csv','r') as f:
# with open('Taxi000.csv','r') as f:
#     dict = {}
    reader = csv.reader(f)
    fieldNames = next(reader)
    listlist = [list() for x in range(24)]
    for i, line in enumerate(reader):
#         print line
        dt = line[5].split(' ')
        t = dt[1].split(':')
        hour = int(t[0])
#         print hour
        count = float(line[7])
#         print count
        listlist[hour].append(count)
```

The index of each list in your list of lists refers to the hour of the day, and its contents are the number of passengers at that hour.



```
fulldict = {}
for i in range(len(listlist)):
    try:
#         print listlist.index(x), sum(x), len(x), sum(x)/len(x)
        z = sum(listlist[i])/len(listlist[i])
        #print i, z
        fulldict[i] = z
    except ZeroDivisionError, e:
        print e
        pass
# print fulldict

lists = sorted(fulldict.items()) # sorted by key, return a list of tuples
x, y = zip(*lists) # unpack a list of pairs into two tuples
plt.plot(x, y)
plt.show()
```

For each list, sum the items in that list, and divide them by the length of the list to find the average number of passengers. Assign the index of that list to a key in a dictionary, and let that key's value be the average number of passengers. This makes the data easy to pass to matplotlib. I plotted it a couple of different ways, shown here:



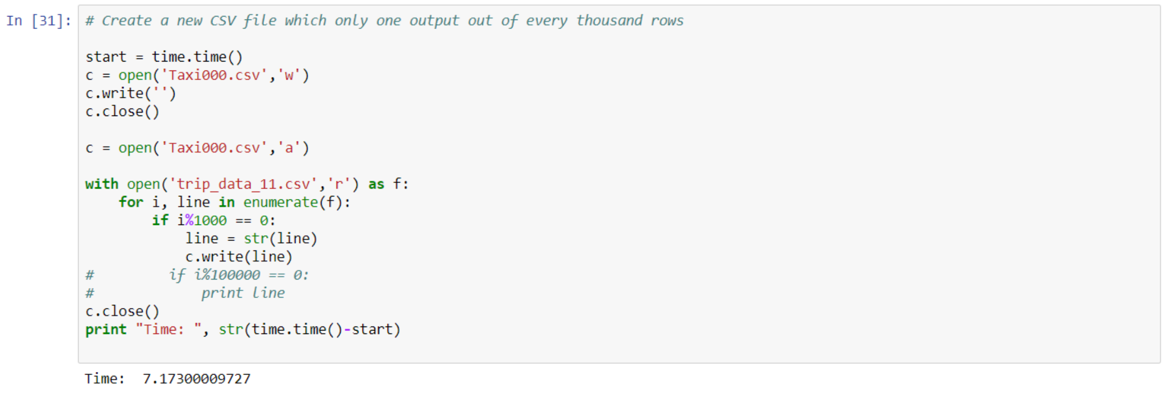
(a) Line Graph

(b) Histogram

Figure 2: Passenger Counts Seem to Drop around 6 AM

9. Create a new CSV file which has only one out of every thousand rows.

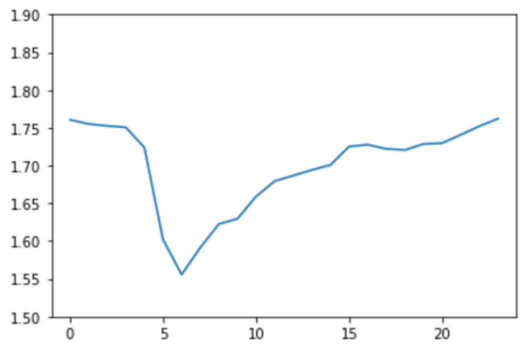
Just write it into a file. Use the modulus operator to select lines.



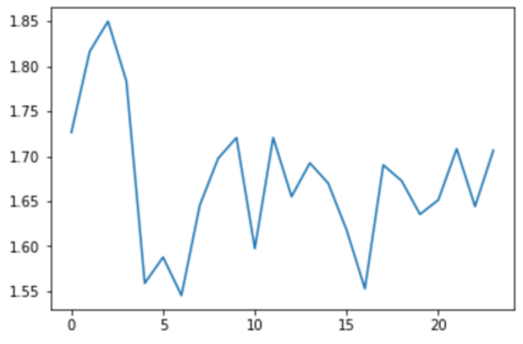
```
In [31]: # Create a new CSV file which only one output out of every thousand rows

         start = time.time()
         c = open('Taxi000.csv','w')
         c.write('')
         c.close()

         c = open('Taxi000.csv','a')

         with open('trip_data_11.csv','r') as f:
             for i, line in enumerate(f):
                 if i%1000 == 0:
                     line = str(line)
                     c.write(line)
         #        if i%100000 == 0:
         #            print line
         c.close()
         print "Time: ", str(time.time()-start)

         Time:  7.17300009727
```

10. Repeat step 8 with the reduced dataset and compare the two charts.

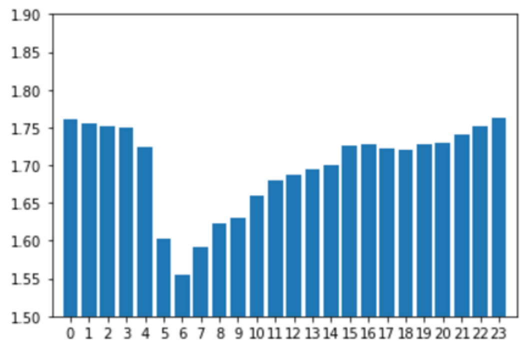
I can run identical code on the reduced data set. Here are the charts, side-by-side:



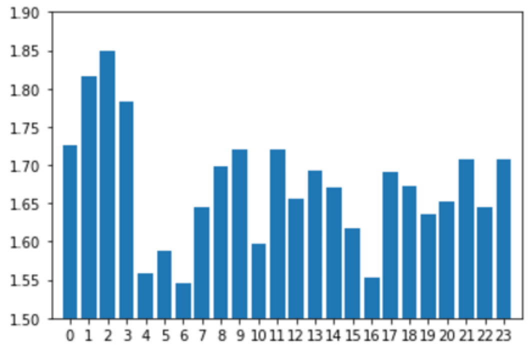
(a) Full Set



(b) Reduced Set



(a) Full Set

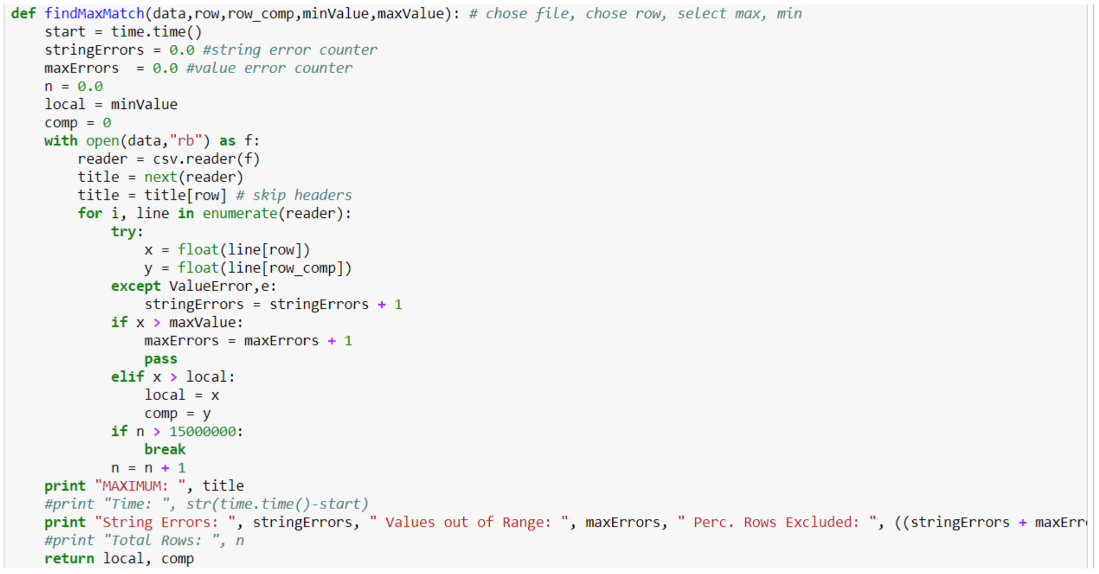


(b) Reduced Set

This is an acceptable result. The means are approximately the same, but the data is not as smooth in the reduced set. This is what we would expect to see, statistically, from a sample of a dataset: that smaller sample will have a greater variance but similar mean. The additional data seems to smooth out the curve.

11. Bonus Round

I was upset that the min / max geo values were garbage, so I wrote a little something to see where those maximum values are coming from. I just wanted to see if they were over landmasses, or in the center of the ocean. Maybe some of these trips are legitimate, but just don't originate and terminate in New York city. I used the North American borders, this time.



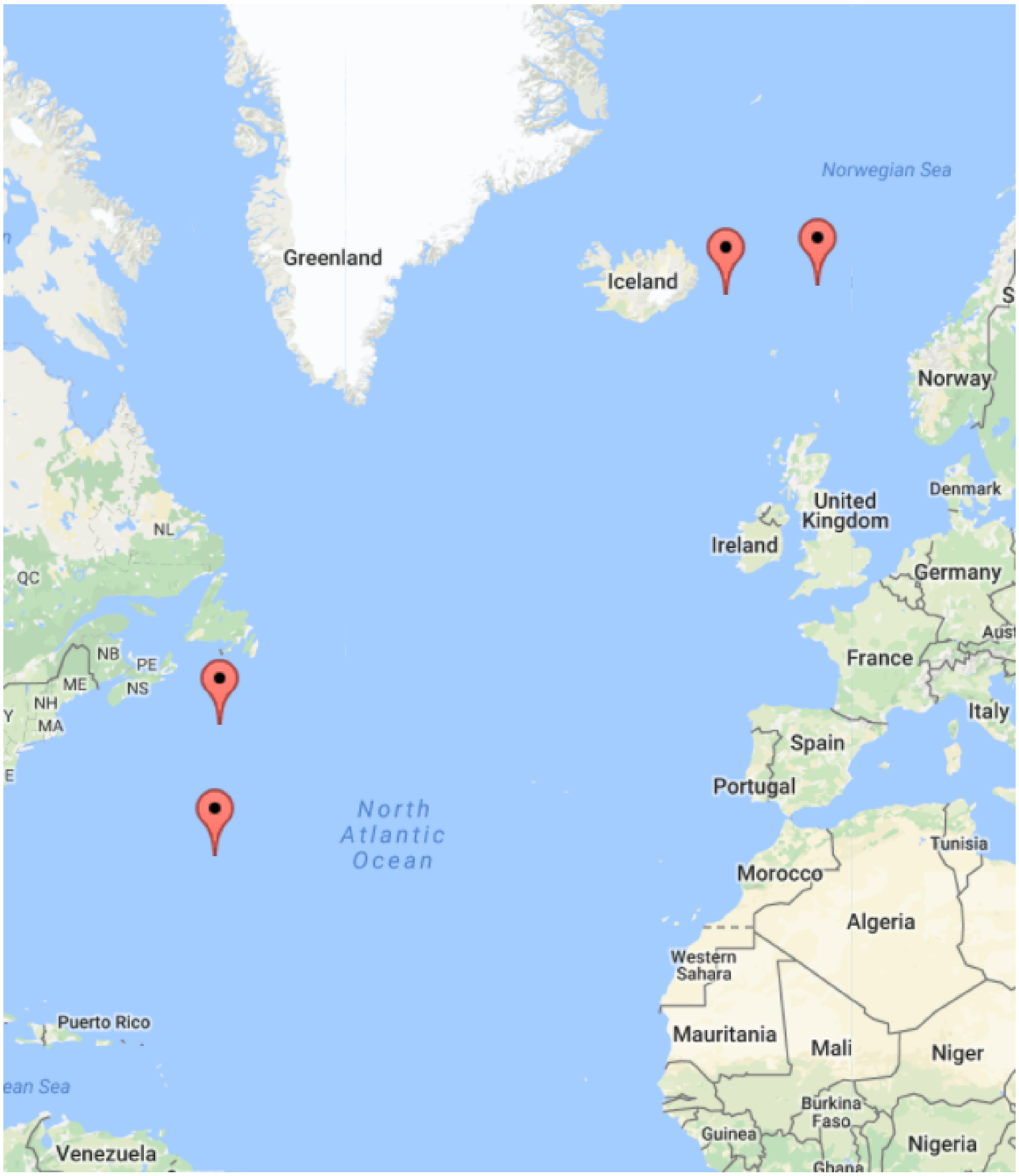
```
def findMaxMatch(data,row,row_comp,minValue,maxValue): # chose file, chose row, select max, min
    start = time.time()
    stringErrors = 0.0 #string error counter
    maxErrors  = 0.0 #value error counter
    n = 0.0
    local = minValue
    comp = 0
    with open(data,"rb") as f:
        reader = csv.reader(f)
        title = next(reader)
        title = title[row] # skip headers
        for i, line in enumerate(reader):
            try:
                x = float(line[row])
                y = float(line[row_comp])
            except ValueError,e:
                stringErrors = stringErrors + 1
            if x > maxValue:
                maxErrors = maxErrors + 1
                pass
            elif x > local:
                local = x
                comp = y
            if n > 15000000:
                break
            n = n + 1
    print "MAXIMUM: ", title
    #print "Time: ", str(time.time()-start)
    print "String Errors: ", stringErrors, " Values out of Range: ", maxErrors, " Perc. Rows Excluded: ", ((stringErrors + maxErr
    #print "Total Rows: ", n
    return local, comp
```

This function just pulls out the corresponding GPS coordinate, longitude or latitude, for each column. I ran this, and an identical min function on all four rows, returning a total of 8 GPS coordinates. Here's the results of those calls:

1. (33.093407,-56.633026)
2. (0,-168.3)
3. (64.870567, -2.9665699)
4. (7.0, 0)
5. (42.283688, -56.099979)
6. (0,-168.3)
7. (64.445435, -11.044415)
8. (7.0, 0)

So, we can see that a truly intelligent way to do this would have to reach into other columns and also see if the values given there are legitimate. As of now, we can probably throw away coordinate #2, #4, #6, and #8[4]. Plotting the four legitimate coordinates on a map, we get the following:

[4]For reference, those are all of the minimum function calls. That means that all of the minimum values are paired incorrectly. That probably gives us some insight into the nature of the GPS problem we're seeing.



This seems totally legit to me. Looks like the GPS data was perfect, after all[5].

In actuality, what's happening here is that the two leftmost points are longitude-limited points with unrealistic latitudes, and the the two topmost points are latitude-limited points with unrealistic longitudes.

[5]This is what's referred to as 'sarcasm' in Meatspace.