Chapter 8
Streaming Data in the IPython Notebook

An interest in working with Dynamic data is what brought Dr. Doi and I together on this project. Dynamic data is data that is always changing. Some data sets might change only a few times in a 5 minute interval, others might change 100 times a second. The fascinating thing about dynamic data is any time it is analyzed, the analysis is potentionally behind the data. Consider LADWP's (Los Angeles Department of Water and Power) water data. In 2013, the United States Census Bureau estimated the population of the city of Los Angeles was about 3.9 millon people. With 3.9 million people, it seems impossible that the water consumption in Los Angeles could ever stop. This implys that as we analyze water consumption in LA, we are immediately missing new data. In the future, I intend to use dynamic environmental data streams to provide people with accurate, analyzed in real time, information that enables them to make choices that best support sustainability in their area.

8.1 Python Packages for Streaming and the Import Cell

```
In [4]:  import requests
import json
import pandas as pd
from mpl_toolkits.basemap import Basemap
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

8.2 Streaming Data from USA.gov

USA.gov describes the data as, "We provide a raw pub/sub feed of data created any time anyone clicks on a 1.USA.gov URL. The pub/sub endpoint responds to http requests for any 1.USA.gov URL and returns a stream of JSON entries, one per line, that represent real-time clicks."

A few years ago USA.gov "held a nationwide 1.USA.gov Hack Day... to encourage people to explore the 1.USA.gov data." The projects and code resulting from this are more sophicated than what we're doing here and can be found at: http://www.usa.gov/About/developer-resources/1usagov.shtml

In [5]:
```
url = "http://developer.usa.gov/1usagov"
```

In [87]:
```
#url argument is the live datastream
r = requests.get(url, stream=True)

#after grabbing n data values, the datastream stops
n = 500
data = []

#looks at each line of the request individually and adds it to the list "d
ata"
for i, line in enumerate(r.iter_lines()):
    data.append(line)

    #this is a dirty little trick that should be avoided in larger functio
ns and
    #programs, but works great for this quick line fectching function
    if i > n:
        break
```

In [88]:
```
#load the json lines from the list
jdata = [json.loads(item) for item in data[1:]]
```

In [89]:
```
#create a DataFrame
USAGovFrame = pd.DataFrame(jdata)
```

We built a DataFrame from USAGov's live stream. Let's take a peek at it and see what we ended up with.

In [90]:
```
USAGovFrame.head()
```

Out[90]:

	heartbeat	_id	a	al	c	ckw	cy	dp	g	gr
		543ae232-	Mozilla/5.0							

0	NaN	002b9-0416e-cf1cf10a	(Windows NT 6.1; WOW64) AppleWebKi...	es-419,es;q=0.8	MX	NaN	NaN	NaN	15r91	N
1	NaN	543ae232-003ac-038db-301cf10a	Mozilla/5.0 (iPhone; CPU iPhone OS 7_1_2 like ...	en-us	US	NaN	Ponder	NaN	1v02m1T	T
2	NaN	543ae233-00395-07c0e-361cf10a	Mozilla/5.0 (iPhone; CPU iPhone OS 8_0_2 like ...	fr-fr	FR	NaN	NaN	NaN	ZzdKp8	N
3	NaN	543ae234-00165-07334-261cf10a	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKi...	en-US,en;q=0.8	US	NaN	Spirit Lake	NaN	1qfk3VH	I/
4	NaN	543ae234-00364-06587-2a1cf10a	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5)...	es-es	ES	NaN	Vigo	NaN	K6Cor	5

5 rows × 21 columns

After looking at the descriptions of the variables provided from 1USA.gov, I don't know what I would do with several of them. Let's reduce the DataFrame to variables we are interested in playing with.

In [91]:
```
#drop all the columns that I don't know anything about
#ckw and dp aren't consistently collected, especially with smaller sample
sizes
#this will check for them and drop them if they are present
if 'ckw' and 'dp' in USAGovFrame.columns:
    USAGovFrame.drop(['_heartbeat_','_id','al','ckw','nk','g','h','kw','hc
','dp'],inplace=True, axis=1)
else:
    USAGovFrame.drop(['_heartbeat_','_id','al','nk','g','h','kw','hc'],inp
lace=True, axis=1)
```

In [92]:
```
#add user friendly names for the remaining variables
USAGovFrame.columns=['User_Agent','Country_Code','Geo_city_name',
                     'Geo_Region','Short_url_Cname','Encoding_user_login',
                     '[Latitude,Longitude]','Referring_URL','Timestamp',
                     'Timezone','Long_URL']
```

In [93]:
```
#look at the new DataFrame
USAGovFrame.head()
```

Out[93]:

	User_Agent	Country_Code	Geo_city_name	Geo_Region	Short_url_Cname	Encoding_
	Mozilla/5.0					

0	(Windows NT 6.1; WOW64) AppleWebKi...	MX	NaN	NaN	j.mp	pontifier
1	Mozilla/5.0 (iPhone; CPU iPhone OS 7_1_2 like ...	US	Ponder	TX	ift.tt	ifttt
2	Mozilla/5.0 (iPhone; CPU iPhone OS 8_0_2 like ...	FR	NaN	NaN	1.usa.gov	tweetdecl
3	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKi...	US	Spirit Lake	IA	1.usa.gov	theusnav
4	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5)...	ES	Vigo	58	1.usa.gov	anonymo

With the DataFrame we have, let's plot all the coordinates on a world map to get an idea of who just visited .gov websites when we excuted the code above.

First, we need to learn about the coordinates in the Latitude/Longitude column.

In [101]:
```
#check the type of data at the first index
type(USAGovFrame['[Latitude,Longitude]'][0])
```

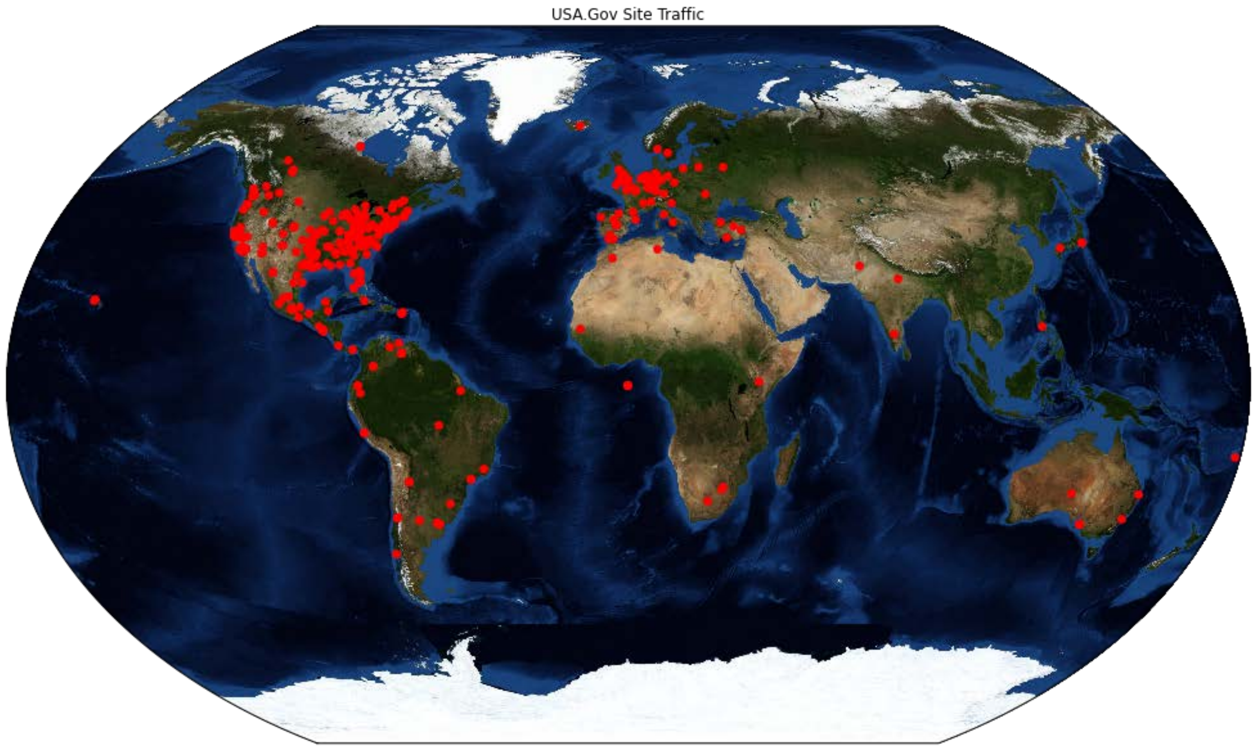
Out[101]:
```
list
```

I tried to zip and unpack this data the short way, but I received a "too many values" error. As you might have figured out by now, it's time for another function to help us with this process.

The goal is to pass the column of the DataFrame to a function that will separate the latitude and the longitude into their own variables in order to plot them as x and y variables on a world map.

In [94]:
```
#function for unpacking a list in each row of a column into two separate l
ists
def lat_lon(column):

    #create two empty lists
    lat=[]
    lon=[]

    #at each row in the column, add first index to lat, second to lon
    for i in column:
        lat.append(i[0])
        lon.append(i[1])
    #return the new lists
    return lat,lon
```

In [97]:
```
#open a larger figure so we have a better since of the global web traffic
plt.figure(figsize=(20,10))

# lon_0 is central longitude of projection.
# resolution = 'c' means use crude resolution coastlines.
map = Basemap(projection='kav7',lon_0=0,resolution='c')

#bluemarble is a built in function to basemap
map.bluemarble()

#drop the NaN values from the data frame
coords = USAGovFrame['[Latitude,Longitude]'].dropna()

#run the function from the cell above
lat, lon = lat_lon(coords)

#plot the points on the bluemarble basemap
# '.' is the marker type, c is the color
x,y = map(lon, lat)
map.plot(x,y,'.', c='red',markersize=12)
plt.title("USA.Gov Site Traffic");
```



Chapter 9
Time Series, More with Data Frames, and Advanced Plotting in the IPython Notebook