Python for Data Analysis and Visualization

Fang (Cherry) Liu, Ph.D
fang.liu@oit.gatech.edu
PACE Gatech
July 2013

Outline

- System requirements and IPython
- Why use python for data analysis and visulation
- Data set – US baby names 1880-2012
  - Data Loading
  - Data Processing using Lists
  - Data Aggregtion and Group
- Plotting and visualization

System Setup

- Option 1 : **(Preferred)** Download and Install the Enthought Canopy product: https://www.enthought.com/products/canopy/academic/ Enthought Canopy is free for Academic Users. This will install a full Python distribution onto your computer.

- Option 2: Download and Install Python(x,y) (This is for Windows only) https://code.google.com/p/pythonxy/wiki/Downloads
  This will install a full Python distribution onto your computer.

  Note1: Options 1 and 2 are mutually exclusive. Please do not install both Canopy and Python(x,y) on your computer.
  Note2: Downloading and installing either Canopy or Python(x,y) will take a long time.
  Note3: During this course, Canopy will be used to type and execute all commands (option 1).

- Option 3: Use the Python installed on PACE clusters. (You need a PACE account for this to work)
  If you choose this option, let me know and I'll send instructions that will help ensure that your environment is setup properly for the tutorials.

- Option 4: Use the Python already installed on your laptop. As long as Numpy, SciPy, Matplotlib, IPython, and Pandas are installed on your laptop, you will be able follow both courses (Scientific Computing and Data Analysis and Visualization).

.

IPython – An Interactive Computing and Development Environment

- It provides an execute-explore workflow instead of typical edit-compile-run workflow of many other programming languages
- It provides very tight integration with the operating system's shell and file system
- It also includes:
  - A rich GUI console with inline plotting
  - A web-based interactive notebook format
  - A lightweight, fast parallel computing engine

Why use Python for Data Analysis

- The Python language is easy to fall in love with
- Python is distinguished by its large and active scientific computing community
- Adoption of Python for scientific computing in both industry applications and academic research has increased significantly since the early 2000s
- Python's improved library support (pandas) made it a strong tool for data manipulation tasks

Example: US Baby Names 1880-2012

- The United States Social Security Administration (SSA) has mad available data on the frequency of baby names from 1880 through 2012, this data set is often used in illustrating data manipulation in R, Python, etc. The data can be obtained at: http://www.ssa.gov/oact/babynames/limits.html

- Things can be done with this data set
  - Visualize the proportion of babies given a particular name
  - Determine the naming trend
  - Determine the most popular names in each year

Check the Data

- In IPython,
  - MacOS or Linux: use the UNIX **head** to look at the first 10 lines of the one of the files.
  - Windows: download the files, and click to open the files
  - This is nicely comma-separated form.

```
In [162]: !head -n 10 yob1910.txt
Mary,F,22840
Helen,F,10477
Margaret,F,8226
Dorothy,F,7315
Ruth,F,7210
Anna,F,6434
Elizabeth,F,5799
Mildred,F,5692
Marie,F,4790
Alice,F,4670
```

Load Data

- Using csv module from the standard library, CSV means Comma Separated Values, and any delimiter can be chosen.

```
In [5]: import csv
   ...: file=open('yob1910.txt','rb')
   ...: data=csv.reader(file)
   ...: table=[row for row in data]
   ...: table[:3]
   ...:
Out[5]: [['Mary', 'F', '22847'], ['Helen', 'F', '10479'], ['Margaret', 'F', '8226']]
```

- The variable ***table*** contains records list in which each record has three fields : *name*, *sex*, *count*

Grouping the data based on sex

- To find the total births by sex, the **groupby** function is used:
  - It returns an iterator for each group based on the *key* value which is extracted from x[1] (*sex*)
  - Then traverses the group and get the total counts
  - Be sure to do "from itertools import groupby" first

```
In [283]: for key,group in groupby(table, lambda x: x[1]):
     ...:     total=0
     ...:     for item in group:
     ...:         total+=int(item[2])
     ...:     print item[1], total
     ...:
F 396416
M 194198
```

Anonymous (lamda) Functions

- Anonymous or lambda functions are simple functions consisting of a single statement, the result is the return value.
- Lamda functions are convenient in data analysis since there are many cases where data transformation functions will take functions as arguments.

```
In [269]: def f(x):
     ...:     return x*2

In [270]: f_lambda=lambda x: x*2

In [271]: f(3)
Out[271]: 6

In [272]: f_lambda(3)
Out[272]: 6
```

```
In [277]: apply_g(input_list,lambda x: x*2)
Out[277]: [10, 12, 20, 14, 2, 6]

In [278]: def apply_g(alist, g):
     ...:     return[g(x) for x in alist]
     ...:

In [279]: input_list=[5,6,10,7,1,3]

In [280]: apply_g(input_list,f)
Out[280]: [10, 12, 20, 14, 2, 6]

In [281]: apply_g(input_list,lambda x: x*2)
Out[281]: [10, 12, 20, 14, 2, 6]
```

Aggregate the data at the year and sex level

- Since the data set is split into files by year, one need to traverse all the files to get the total number of births per year per sex

```
In [381]: years = range(1880,2012)
     ...: pieces=[]
     ...:
     ...: for year in years:
     ...:     count=[year,0,0]
     ...:     path='yob%d.txt'%year
     ...:     file=open(path,'rb')
     ...:     data=csv.reader(file,delimiter=',')
     ...:     table=[row for row in data]
     ...:
     ...:     for key,group in groupby(table, lambda x: x[1]):
     ...:         total=0
     ...:         for item in group:
     ...:             total+=int(item[2])
     ...:         if item[1]=='F':
     ...:             count[1]=total
     ...:         elif item[1]=='M':
     ...:             count[2]=total
     ...:         print count
     ...:     pieces.append(count)
```

The result list

- (Left) first 10 records in **pieces** list
- (Right) last 10 records in **pieces** list

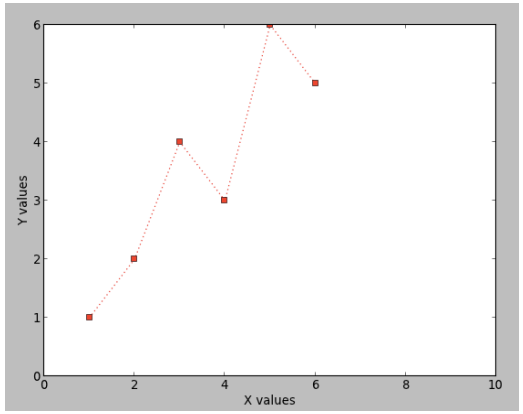
```
In [390]: pieces[:10]
Out[390]:
[[1880, 90993, 110491],
 [1881, 91955, 100746],
 [1882, 107850, 113687],
 [1883, 112322, 104630],
 [1884, 129022, 114445],
 [1885, 133055, 107801],
 [1886, 144534, 110786],
 [1887, 145982, 101414],
 [1888, 178628, 120854],
 [1889, 178365, 110587]]
```

```
In [391]: pieces[-10:]
Out[391]:
[[2002, 1794898, 1939507],
 [2003, 1825012, 1973072],
 [2004, 1833743, 1982335],
 [2005, 1844852, 1994344],
 [2006, 1897825, 2051668],
 [2007, 1918618, 2071178],
 [2008, 1886109, 2035075],
 [2009, 1831382, 1977632],
 [2010, 1770632, 1911572],
 [2011, 1750078, 1889557]]
```

Matplotlib review

- Before we start plotting the result, let's review the plot first

```
In [420]: import matplotlib.pyplot as ptl

In [421]: x=[1,2,3,4,5,6]

In [422]: y=[1,2,4,3,6,5]

In [423]: ptl.plot(x,y,':rs')
Out[423]: [<matplotlib.lines.Line2D at 0x17af1a70>]

In [424]: ptl.axis([0,10,0,6])
Out[424]: [0, 10, 0, 6]

In [425]: ptl.xlabel("X values")
Out[425]: <matplotlib.text.Text at 0x179b7610>

In [426]: ptl.ylabel("Y values")
Out[426]: <matplotlib.text.Text at 0xf9b2730>

In [427]: ptl.show()
```



Prepare the data for plot

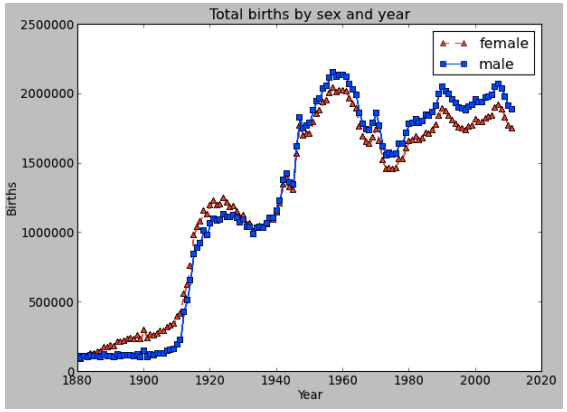
- Currently, the result is a list of list, each internal list include three values, [year, female births, male births], to plot the births according to year and sex, the plot needs to have year as x-axis, and births as y-axis, while two lines will be showing to represent female and male birth.

```
In [437]: pieces[:4]
Out[437]:
[[1880, 90993, 110491],
 [1881, 91955, 100746],
 [1882, 107850, 113687],
 [1883, 112322, 104630]]

In [438]: X=[year for [year,female,male] in pieces]

In [439]: Y1=[female for [year,female,male] in pieces]

In [440]: Y2=[male for [year,female,male] in pieces]
```

Plot the total births by sex and year

- Plot

```
In [486]: import matplotlib.pyplot as plt

In [487]: p1=plt.plot(X,Y1,'r^--')

In [488]: p2=plt.plot(X,Y2,'bs-')

In [489]: plt.legend((p1[0],p2[0]),('female','male'))
Out[489]: <matplotlib.legend.Legend at 0xff85d70>

In [490]: plt.title("Total births by sex and year")
Out[490]: <matplotlib.text.Text at 0x1757ed70>

In [491]: plt.xlabel("Year")
Out[491]: <matplotlib.text.Text at 0x179cdef0>

In [492]: plt.ylabel("Births")
Out[492]: <matplotlib.text.Text at 0xffc92b0>

In [493]: plt.show()
```



Reorganize the data

- Concatenate the all files together to prepare the further analysis.

```
In [23]: import csv
    ...: years =range(1880,2013)
    ...: names=[]
    ...: for year in years:
    ...:         path='yob%d.txt'%year
    ...:         file=open(path,'rb')
    ...:         data=csv.reader(file,delimiter=',')
    ...:         table=[[year] + row for row in data]
    ...:         names+=table
    ...:

In [24]: names[:5]
Out[24]:
[[1880, 'Mary', 'F', '7065'],
 [1880, 'Anna', 'F', '2604'],
 [1880, 'Emma', 'F', '2003'],
 [1880, 'Elizabeth', 'F', '1939'],
 [1880, 'Minnie', 'F', '1746']]

In [25]: names[-5:]
Out[25]:
[[2012, 'Zylin', 'M', '5'],
 [2012, 'Zymari', 'M', '5'],
 [2012, 'Zyrin', 'M', '5'],
 [2012, 'Zyrus', 'M', '5'],
 [2012, 'Zytaevius', 'M', '5']]
```

Extract a subset of the data

- Find the top 1000 names for each sex/year combination, further narrow down the data set to facilitate further analysis, the sorting is ignored here since the input files are already in descending order

```
In [9]: top1000=[]
   ...: for year in years:
   ...:  boys=[y for y in names if (int(y[0])==year and y[2]=='M')]
   ...:  girls=[x for x in names if (int(x[0])==year and x[2]=='F')]
   ...:  top1000+=boys[:1000]
   ...:  top1000+=girls[:1000]
   ...:
```

Compare the subset data with original data

- The subset data has much less records than the original data set, but represents the majority information

```
In [14]: names[:4]
Out[14]:
[[1880, 'Mary', 'F', '7065'],
 [1880, 'Anna', 'F', '2604'],
 [1880, 'Emma', 'F', '2003'],
 [1880, 'Elizabeth', 'F', '1939']]

In [15]: names[-4:]
Out[15]:
[[2012, 'Zymari', 'M', '5'],
 [2012, 'Zyrin', 'M', '5'],
 [2012, 'Zyrus', 'M', '5'],
 [2012, 'Zytaevius', 'M', '5']]
```

```
In [10]: size(top1000)
Out[10]: 1063508

In [11]: size(names)
Out[11]: 7034920

In [12]: top1000[:4]
Out[12]:
[[1880, 'John', 'M', '9655'],
 [1880, 'William', 'M', '9532'],
 [1880, 'James', 'M', '5927'],
 [1880, 'Charles', 'M', '5348']]

In [13]: top1000[-4:]
Out[13]:
[[2012, 'Tess', 'F', '252'],
 [2012, 'Ashtyn', 'F', '251'],
 [2012, 'Jessa', 'F', '251'],
 [2012, 'Katalina', 'F', '251']]
```

Analyzing Naming Trends

- With the full data set and Top 1,000 data set in hand, we can start analyzing various naming trends of interest. Splitting the Top 1,000 names into the boy and girl portions:

```
In [17]: boys=[y for y in top1000 if y[2]=='M']

In [18]: size(boys)
Out[18]: 531988

In [19]: boys[:4]
Out[19]:
[[1880, 'John', 'M', '9655'],
 [1880, 'William', 'M', '9532'],
 [1880, 'James', 'M', '5927'],
 [1880, 'Charles', 'M', '5348']]
```

```
In [21]: girls=[x for x in top1000 if x[2]=='F']

In [22]: size(girls)
Out[22]: 531520

In [23]: girls[:4]
Out[23]:
[[1880, 'Mary', 'F', '7065'],
 [1880, 'Anna', 'F', '2604'],
 [1880, 'Emma', 'F', '2003'],
 [1880, 'Elizabeth', 'F', '1939']]
```

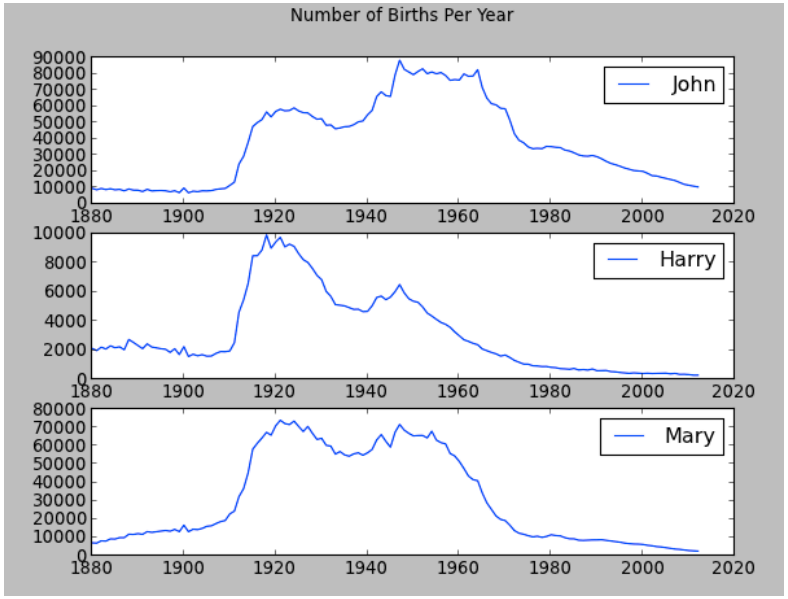
Analyzing Naming Trends (Cont.)

- Plot for a handful of names in a subplot, John, Harry, Marry, to compare their trends over the years, first prepare data set for each chosen name.

```
In [208]: John=[y[3] for y in boys if y[1]=='John']

In [209]: John[:3]
Out[209]: ['9655', '8769', '9557']

In [210]: size(John)
Out[210]: 133

In [211]: Harry=[y[3] for y in boys if y[1]=='Harry']

In [212]: Harry[:3]
Out[212]: ['2152', '2002', '2232']

In [213]: size(Harry)
Out[213]: 133

In [214]: Mary=[x[3] for x in girls if x[1]=='Mary']

In [214]:

In [215]: Mary[:3]
Out[215]: ['7065', '6919', '8148']

In [216]: size(Mary)
Out[216]: 133

In [217]: years=range(1880,2013)

In [218]: years[:3]
Out[218]: [1880, 1881, 1882]

In [219]: size(years)
Out[219]: 133
```

Analyzing Naming Trends (Cont.)

- Plot three curves vertically, with x-axis as years, y-axis as births, the result shows that those names have grown out of favor with American population




```
In [206]: plt.close('all')
     ...: fig, ((ax1, ax2, ax3)) = plt.subplots(nrows=3, ncols=1)
     ...: ax1.plot(years,John,label="John")
     ...: ax2.plot(years,Harry,label="Harry")
     ...: ax3.plot(years,Mary,label="Mary")
     ...: ax1.legend()
     ...: ax2.legend()
     ...: ax3.legend()
     ...: plt.suptitle("Number of Births Per Year")
     ...:
```

Measuring the increase in naming diversity

- To explain why there is a decrease in the previous plots, we can measure the proportion of births represented by the top 1000 most popular names by year and sex
  - Step 1: find total of birth per year for each sex

```
In [448]: allboys=[y for y in names if (y[2]=='M')]

In [449]: boyyearsum=[]
     ...: for key, group in groupby(allboys, lambda x: x[0]):
     ...:     sum=0;
     ...:     for item in group:
     ...:         sum+=int(item[3])
     ...:     boyyearsum.append([item[0],sum])
     ...:

In [450]: boyyearsum[:3]
Out[450]: [[1880, 110491], [1881, 100746], [1882, 113687]]

In [451]: size(boyyearsum)/2
Out[451]: 133
```

```
In [452]: allgirls=[x for x in names if (x[2]=='F')]

In [453]: girlyearsum=[]
     ...: for key, group in groupby(allgirls, lambda x: x[0]):
     ...:     sum=0;
     ...:     for item in group:
     ...:         sum+=int(item[3])
     ...:     girlyearsum.append([item[0],sum])
     ...:

In [454]: girlyearsum[:3]
Out[454]: [[1880, 90993], [1881, 91955], [1882, 107850]]

In [455]: size(girlyearsum)
Out[455]: 266
```

Measuring the increase in naming diversity (Cont.)

– Step 2: compute the proportion of top 1000 births to the total births per year per sex

For boys:

```
In [460]: boys1000=[y for y in top1000 if y[2]=='M']

In [461]: boys1000prop=[]
     ...: for key, group in groupby(boys1000, lambda x: x[0]):
     ...:         partialsum=0
     ...:         yearsum=[x[1] for x in boyyearsum if int(x[0])==key]
     ...:         for item in group:
     ...:             partialsum+=int(item[3])
     ...:         boys1000prop+=[[key,float(partialsum)/float(yearsum[0])]]
     ...:

In [462]: boys1000prop[:3]
Out[462]: [[1880, 0.9973753518386113], [1881, 1.0], [1882, 0.9956459401690607]]

In [463]: size(boys1000prop)/2
Out[463]: 133
```

Measuring the increase in naming diversity (Cont.)

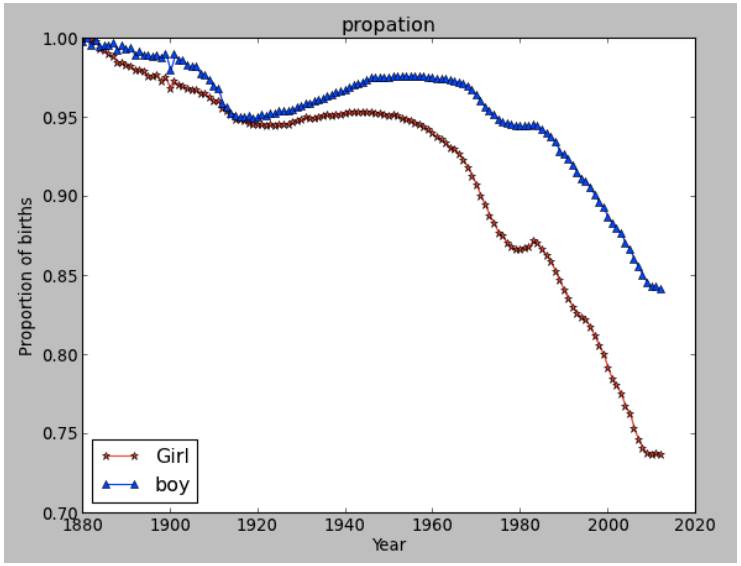
For girls:

```
In [464]: girls1000=[x for x in top1000 if x[2]=='F']

In [465]: girls1000prop=[]
     ...: for key, group in groupby(girls1000, lambda x: x[0]):
     ...:         partialsum=0
     ...:         yearsum=[x[1] for x in girlyearsum if int(x[0])==key]
     ...:         for item in group:
     ...:             partialsum+=int(item[3])
     ...:         girls1000prop+=[[key,float(partialsum)/float(yearsum[0])]]
     ...:

In [466]: girls1000prop[:3]
Out[466]: [[1880, 1.0], [1881, 1.0], [1882, 0.9987019007881317]]

In [467]: size(girls1000prop)/2
Out[467]: 133
```

Measuring the increase in naming diversity (Cont.)

- Plot the result shows that fewer parents are choosing the popular names for their children over the years

```
In [471]: import matplotlib.pyplot as plt
     ...: gx=[x[0] for x in girls1000prop]
     ...: gy=[x[1] for x in girls1000prop]
     ...: bx=[y[0] for y in boys1000prop]
     ...: by=[y[1] for y in boys1000prop]
     ...: plt.close('all')
     ...: plt.plot(gx,gy,'r*-',label='Girl')
     ...: plt.plot(bx,by,'b^-',label='boy')
     ...: plt.xlabel('Year')
     ...: plt.ylabel('Proportion of births')
     ...: plt.title('propation')
     ...: plt.legend(loc=3)
     ...:
Out[471]: <matplotlib.legend.Legend at 0x28f80830>
```



Measuring the increase in naming diversity (Cont.)

- Another interest metric is the number of distinct popular names, taken in order of popularity from highest to lowest in the top 50% of births.
  - Step 1: Add the fourth column to girls1000 and boys1000 list, to represent the birth proportion to the total birth of the given year, then sort the list in descending order on proportion, sort the list again in ascending order on years. The result list will have each years records in a chunk with proportion number in decreasing order.

Measuring the increase in naming diversity (Cont.)

- For girls:

```
In [474]: girlportion=[]
     ...: for key, group in groupby(girls1000, lambda x: x[0]):
     ...:         yearsum=[x[1] for x in girlyearsum if int(x[0])==key]
     ...:         for item in group:
     ...:             girlportion+=[item+[float(item[3])/float(yearsum[0])]]
     ...:

In [475]: girlportion.sort(reverse=True,key=lambda x:x[4])

In [476]: girlportion.sort(key=lambda x:x[0])

In [477]: girlportion[:3]
Out[477]:
[[1880, 'Mary', 'F', '7065', 0.07764333520160892],
 [1880, 'Anna', 'F', '2604', 0.028617585968151397],
 [1880, 'Emma', 'F', '2003', 0.0220126822942424 14]]

In [478]: girlportion[-3:]
Out[478]:
[[2012, 'Ashtyn', 'F', '251', 0.00014395288889796321],
 [2012, 'Jessa', 'F', '251', 0.00014395288889796321],
 [2012, 'Katalina', 'F', '251', 0.00014395288889796321]]
```

Measuring the increase in naming diversity (Cont.)

- For boys:

```
In [479]: boyportion=[]
     ...: for key, group in groupby(boys1000, lambda x: x[0]):
     ...:         yearsum=[x[1] for x in boyyearsum if int(x[0])==key]
     ...:         for item in group:
     ...:             boyportion+=[item+[float(item[3])/float(yearsum[0])]]
     ...:

In [480]: boyportion.sort(reverse=True,key=lambda x:x[4])

In [481]: boyportion.sort(key=lambda x:x[0])

In [482]: boyportion[:3]
Out[482]:
[[1880, 'John', 'M', '9655', 0.08738268275244138],
 [1880, 'William', 'M', '9532', 0.08626946991157651],
 [1880, 'James', 'M', '5927', 0.05364237811224443]]

In [483]: boyportion[-3:]
Out[483]:
[[2012, 'Kylen', 'M', '198', 0.00010544787386730077],
 [2012, 'Augustine', 'M', '197', 0.00010491530884776895],
 [2012, 'Dangelo', 'M', '197', 0.00010491530884776895]]
```

Measuring the increase in naming diversity (Cont.)

- Step 2: Adding the proportion for each year from highest until the total proportion reaches 50%, recording the number of individual names

Measuring the increase in naming diversity (Cont.)

- For girls:

```
In [484]: girltopname=[]
     ...: for key, group in groupby(girlportion, lambda x: x[0]):
     ...:         sum=0.0
     ...:         count=0
     ...:         for item in group:
     ...:                 sum+=float(item[4])
     ...:                 count=count+1
     ...:                 if sum>0.5:
     ...:                         girltopname+=[[key,count]]
     ...:                         break
     ...:

In [485]: girltopname[:3]
Out[485]: [[1880, 38], [1881, 38], [1882, 38]]

In [486]: girltopname[-3:]
Out[486]: [[2010, 246], [2011, 244], [2012, 247]]
```

Measuring the increase in naming diversity (Cont.)

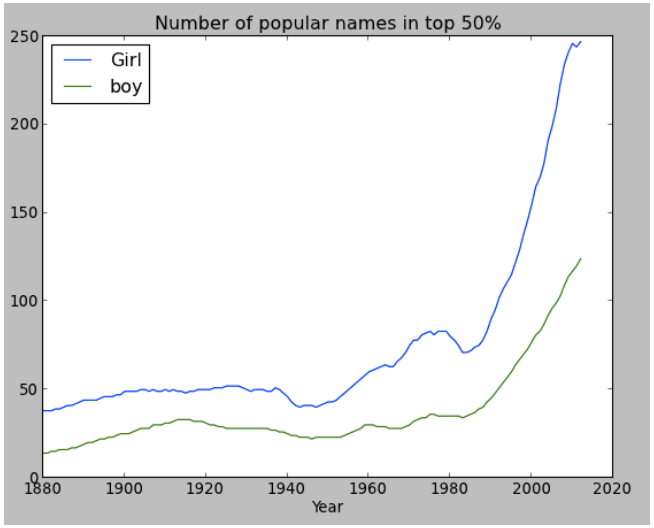
- For boys:

```
In [487]: boytopname=[]
     ...: for key, group in groupby(boyportion, lambda x: x[0]):
     ...:         sum=0.0
     ...:         count=0
     ...:         for item in group:
     ...:                 sum+=float(item[4])
     ...:                 count=count+1
     ...:                 if sum>0.5:
     ...:                         boytopname+=[[key,count]]
     ...:                         break
     ...:

In [488]: boytopname[:3]
Out[488]: [[1880, 14], [1881, 14], [1882, 15]]

In [489]: boytopname[-3:]
Out[489]: [[2010, 117], [2011, 120], [2012, 124]]
```

Measuring the increase in naming diversity (Cont.)

- Step 3: Plot the result, as you can see, girl names has always been more diverse than boy names, and the distinguished names become more over time.

```
In [493]: import matplotlib.pyplot as plt
     ...: gx=[x[0] for x in girltopname]
     ...: gy=[x[1] for x in girltopname]
     ...: bx=[y[0] for y in boytopname]
     ...: by=[y[1] for y in boytopname]
     ...: plt.close('all')
     ...: plt.plot(gx,gy,label='Girl')
     ...: plt.plot(bx,by,label='boy')
     ...: plt.xlabel('Year')
     ...: plt.title('Number of popular names in top 50%')
     ...: plt.legend(loc=2)
     ...:
Out[493]: <matplotlib.legend.Legend at 0x29704e10>
```



Python Library for Data Analysis

- Pandas written by Wes McKinney http://pandas.pydata.org/
  - provides rich data structures and functions working with structured data
  - It is one of the critical ingredients enabling Python to be a powerful and productive data analysis environment.
  - The primary object in pandas is called DataFrame – a two-dimensional tabular, column-oriented data structure with both row and column labels
  - Pandas combines the features of NumPy, spreadsheets and relational databases

Useful Links

- Python Scientific Lecture Notes http://scipy-lectures.github.io/
- Matplotlib http://matplotlib.org/
- Documentation http://docs.python.org