Lesson 1

Create Data - We begin by creating our own data set for analysis. This prevents the end user reading this tutorial from having to download any files to replicate the results below. We will export this data set to a text file so that you can get some experience pulling data from a text file.
Get Data - We will learn how to read in the text file. The data consist of baby names and the number of baby names born in the year 1880.
Prepare Data - Here we will simply take a look at the data and make sure it is clean. By clean I mean we will take a look inside the contents of the text file and look for any anomalities. These can include missing data, inconsistencies in the data, or any other data that seems out of place. If any are found we will then have to make decisions on what to do with these records.
Analyze Data - We will simply find the most popular name in a specific year.
Present Data - Through tabular data and a graph, clearly show the end user what is the most popular name in a specific year.

The ***pandas*** library is used for all the data analysis excluding a small piece of the data presentation section. The ***matplotlib*** library will only be needed for the data presentation section. Importing the libraries is the first step we will take in the lesson.

In [1]:
```
# Import all libraries needed for the tutorial

# General syntax to import specific functions in a library:
##from (library) import (specific library function)
from pandas import DataFrame, read_csv

# General syntax to import a library but no functions:
##import (library) as (give the library a nickname/alias)
import matplotlib.pyplot as plt
import pandas as pd #this is how I usually import pandas
import sys #only needed to determine Python version number

# Enable inline plotting
%matplotlib inline
```

In [2]:
```
print 'Python version ' + sys.version
print 'Pandas version ' + pd.__version__
```

```
Python version 2.7.5 |Anaconda 2.1.0 (64-bit)| (default, Jul  1 2013, 1
Pandas version 0.15.2
```

Create Data

The data set will consist of 5 baby names and the number of births recorded for that year (1880).

In [3]:
```
# The inital set of baby names and bith rates
names = ['Bob','Jessica','Mary','John','Mel']
births = [968, 155, 77, 578, 973]
```

To merge these two lists together we will use the ***zip*** function.

In [4]:
```
zip?
```

In [5]:
```
BabyDataSet = zip(names,births)
BabyDataSet
```

Out[5]:
```
[('Bob', 968), ('Jessica', 155), ('Mary', 77), ('John', 578), ('Mel', 9
```

We are basically done creating the data set. We now will use the ***pandas*** library to export this data set into a csv file.

df will be a ***DataFrame*** object. You can think of this object holding the contents of the BabyDataSet in a format similar to a sql table or an excel spreadsheet. Lets take a look below at the contents inside ***df***.

In [6]:
```
df = pd.DataFrame(data = BabyDataSet, columns=['Names', 'Births'])
df
```

Out[6]:

	Names	Births
0	Bob	968
1	Jessica	155
2	Mary	77
3	John	578
4	Mel	973

Export the dataframe to a ***csv*** file. We can name the file ***births1880.csv***. The function ***to_csv*** will be used to export the file. The file will be saved in the same location of the notebook unless specified otherwise.

In [7]:
```
df.to_csv?
```

The only parameters we will use is ***index*** and ***header***. Setting these parameters to True will prevent the index and header names from being exported. Change the values of these parameters to get a better understanding of their use.

In [8]:
```
df.to_csv('births1880.csv',index=False,header=False)
```

Get Data

To pull in the csv file, we will use the pandas function *read_csv*. Let us take a look at this function and what inputs it takes.

In [9]:
```
read_csv?
```

Even though this functions has many parameters, we will simply pass it the location of the text file.

Location = C:\Users\ENTER_USER_NAME.xy\startups\births1880.csv

Note: Depending on where you save your notebooks, you may need to modify the location above.

In [10]:
```
Location = r'C:\Users\david\notebooks\pandas\births1880.csv'
df = pd.read_csv(Location)
```

Notice the ***r*** before the string. Since the slashes are special characters, prefixing the string with a ***r*** will escape the whole string.

In [11]:
```
df
```

Out[11]:

	Bob	968
0	Jessica	155
1	Mary	77
2	John	578
3	Mel	973

This brings us the our first problem of the exercise. The ***read_csv*** function treated the first record in the csv file as the header names. This is obviously not correct since the text file did not provide us with header names.

To correct this we will pass the ***header*** parameter to the *read_csv* function and set it to ***None*** (means null in python).

In [12]:
```
df = pd.read_csv(Location, header=None)
df
```

Out[12]:

	0	1
0	Bob	968
1	Jessica	155
2	Mary	77
3	John	578
4	Mel	973

If we wanted to give the columns specific names, we would have to pass another paramter called ***names***. We can also omit the *header* parameter.

In [13]:
```
df = pd.read_csv(Location, names=['Names','Births'])
df
```

Out[13]:

	Names	Births
0	Bob	968
1	Jessica	155
2	Mary	77
3	John	578
4	Mel	973

You can think of the numbers [0,1,2,3,4] as the row numbers in an Excel file. In pandas these are part of the ***index*** of the dataframe. You can think of the index as the primary key of a sql table with the exception that an index is allowed to have duplicates.

[Names, Births] can be though of as column headers similar to the ones found in an Excel spreadsheet or sql database.

Delete the csv file now that we are done using it.

In [14]:
```
import os
os.remove(Location)
```

Prepare Data

The data we have consists of baby names and the number of births in the year 1880. We already know that we have 5 records and none of the records are missing (non-null values).

The ***Names*** column at this point is of no concern since it most likely is just composed of alpha numeric strings (baby names). There is a chance of bad data in this column but we will not worry about that at this point of the analysis. The ***Births*** column should just contain integers representing the number of babies born in a specific year with a specific name. We can check if the all the data is of the data type integer. It would not make sense to have this column have a data type of float. I would not worry about any possible outliers at this point of the analysis.

Realize that aside from the check we did on the "Names" column, briefly looking at the data inside the dataframe should be as far as we need to go at this stage of the game. As we continue in the data analysis life cycle we will have plenty of opportunities to find any issues with the data set.

In [15]:
```
# Check data type of the columns
df.dtypes
```

Out[15]:
```
Names     object
Births     int64
dtype: object
```

In [16]:
```
# Check data type of Births column
df.Births.dtype
```

Out[16]:
```
dtype('int64')
```

As you can see the *Births* column is of type ***int64***, thus no floats (decimal numbers) or alpha numeric characters will be present in this column.

Analyze Data

To find the most popular name or the baby name with the higest birth rate, we can do one of the following.

- Sort the dataframe and select the top row
- Use the ***max()*** attribute to find the maximum value

In [17]:
```
# Method 1:
Sorted = df.sort(['Births'], ascending=False)
Sorted.head(1)
```

Out[17]:

	Names	Births
4	Mel	973

In [18]:
```
# Method 2:
df['Births'].max()
```

Out[18]: 973

Present Data

Here we can plot the ***Births*** column and label the graph to show the end user the highest point on the graph. In conjunction with the table, the end user has a clear picture that **Mel** is the most popular baby name in the data set.

plot() is a convinient attribute where pandas lets you painlessly plot the data in your dataframe. We learned how to find the maximum value of the Births column in the previous section. Now to find the actual baby name of the 973 value looks a bit tricky, so lets go over it.

Explain the pieces:
df['Names'] - This is the entire list of baby names, the entire Names column
df['Births'] - This is the entire list of Births in the year 1880, the entire Births column
df['Births'].max() - This is the maximum value found in the Births column

[df['Births'] == df['Births'].max()] **IS EQUAL TO** [Find all of the records in the Births column where it is equal to 973]
df['Names'][df['Births'] == df['Births'].max()] **IS EQUAL TO** Select all of the records in the Names column **WHERE** [The Births column is equal to 973]

An alternative way could have been to use the ***Sorted*** dataframe:
Sorted['Names'].head(1).value

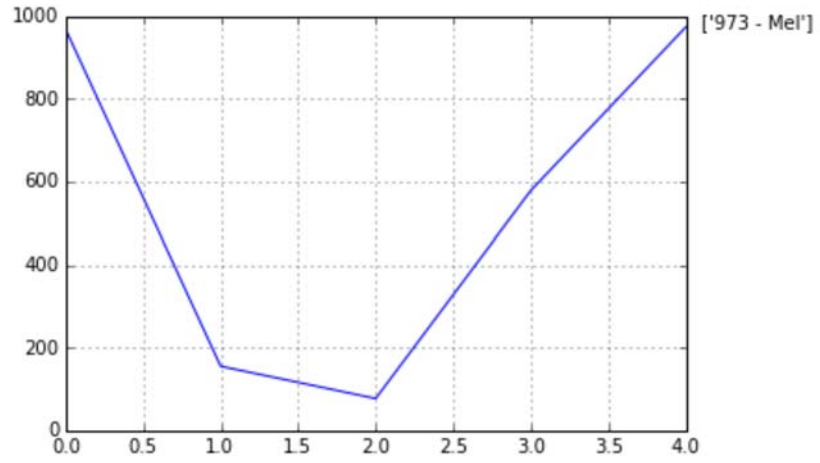
The ***str()*** function simply converts an object into a string.

In [19]:
```
# Create graph
df['Births'].plot()

# Maximum value in the data set
MaxValue = df['Births'].max()

# Name associated with the maximum value
MaxName = df['Names'][df['Births'] == df['Births'].max()].values

# Text to display on graph
Text = str(MaxValue) + " - " + MaxName

# Add text to graph
plt.annotate(Text, xy=(1, MaxValue), xytext=(8, 0),
                 xycoords=('axes fraction', 'data'), textcoords='offset p

print "The most popular name"
df[df['Births'] == df['Births'].max()]
#Sorted.head(1) can also be used
```

```
The most popular name
```

Out[19]:

	Names	Births
4	Mel	973



Author: David Rojas (http://www.hedaro.com/)