Homework 1, MATH295

Nick Strayer

Problem 1a

In [1]:
```
import numpy as np
import pandas as pd

states = ["AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DC", "DE", "FL", "GA",
          "HI", "ID", "IL", "IN", "IA", "KS", "KY", "LA", "ME", "MD",
          "MA", "MI", "MN", "MS", "MO", "MT", "NE", "NV", "NH", "NJ",
          "NM", "NY", "NC", "ND", "OH", "OK", "OR", "PA", "RI", "SC",
          "SD", "TN", "TX", "UT", "VT", "VA", "WA", "WV", "WI", "WY"]
```

Get data of states function + merge dictionaries:

What happened:

These files make me sad. I made a genius mistake of opening up the first one, and hard coding all of my fixes (e.g. ignore line 15) This worked beautifully for the first file and so I decided to link it to all of the files and start a loop a goin, oops.

I then decided to open up one of the files that was throwing an error at me, saw it was just shifted down so I made some adjustements and the result was still a really messed up result (pandas dataframe) when I tried to get everything afterwards. **Finally** I had the genius idea to open *.txt, and would you know it, I should have done that from the begining. Completely changed my method of grabbing the data to just simply find a state's row and infer the time from the file name. I hope I am not ruining my reputation as any sort of data scientist to you in this explanation.

In [2]:
```
def getData(filename):
    beginYear = int(filename[2:4])
    endYear   = int(filename[4:6])
    years = range(beginYear, endYear + 1)
    dataDict = {} #initialize with everystate
    for state in states:
        dataDict[state] = []

    textFile = open("data/" + filename)
    for line in textFile:
        if any([state + " " in line for state in states]): #sees if there is a state in the string
            lineClean = line.replace(" ", "", 1).split()
            if not any([state in lineClean[0] for state in states]):
                lineClean = lineClean[1::]
            dicKey = lineClean[0]
            dataDict[dicKey] = zip(years, [eval(value.replace(",","")) for value in lineClean[1::]])

    return dataDict

def mergeDicts(dict1, dict2):
    """
    Merge two dictionaries that have entries that are lists.
    The dictionaries have to have identical keys.
    """
    keys = dict1.keys()
    for key in keys:
        dict1[key] = dict1[key] + dict2[key]

    return dict1
```

In [3]:
```
#do the functions work?

test1 = getData("st0009ts.txt")
test2 = getData("st7080ts.txt")
merged = mergeDicts(test1,test2)
#merged
```

Sweet, they do, now we move on...

Let's grab the names of all the .txt files in the data directory so we can loop over them to get something actuall useful.

In [4]:
```
import os
fileNames = os.listdir("data")
txtFiles = fileNames[1::]

allData = getData("st0009ts.txt") #initialize with first file:
for file in txtFiles[1::]:
    allData = mergeDicts(allData,getData(file))

for key in allData.keys(): #"Hey guys, let's switch the method of reporting"
    allData[key] = [(date, value * 1000) if date < 70 else (date, value)\
                    for (date,value) in allData[key]]

    for i in reversed(range(1,len(allData[key]))):          #Dumb repeates. Also delete in reverse
        if allData[key][i][0] == allData[key][i - 1][0]:    #order because evil
            del allData[key][i - 1]


#allData["AK"] #Let's see if we have something valuable
```

At this point I have all of the data loaded in, into a big dictionary keyed by the state abreviation and containing a list that has tuples of the year and then the population value at that year.

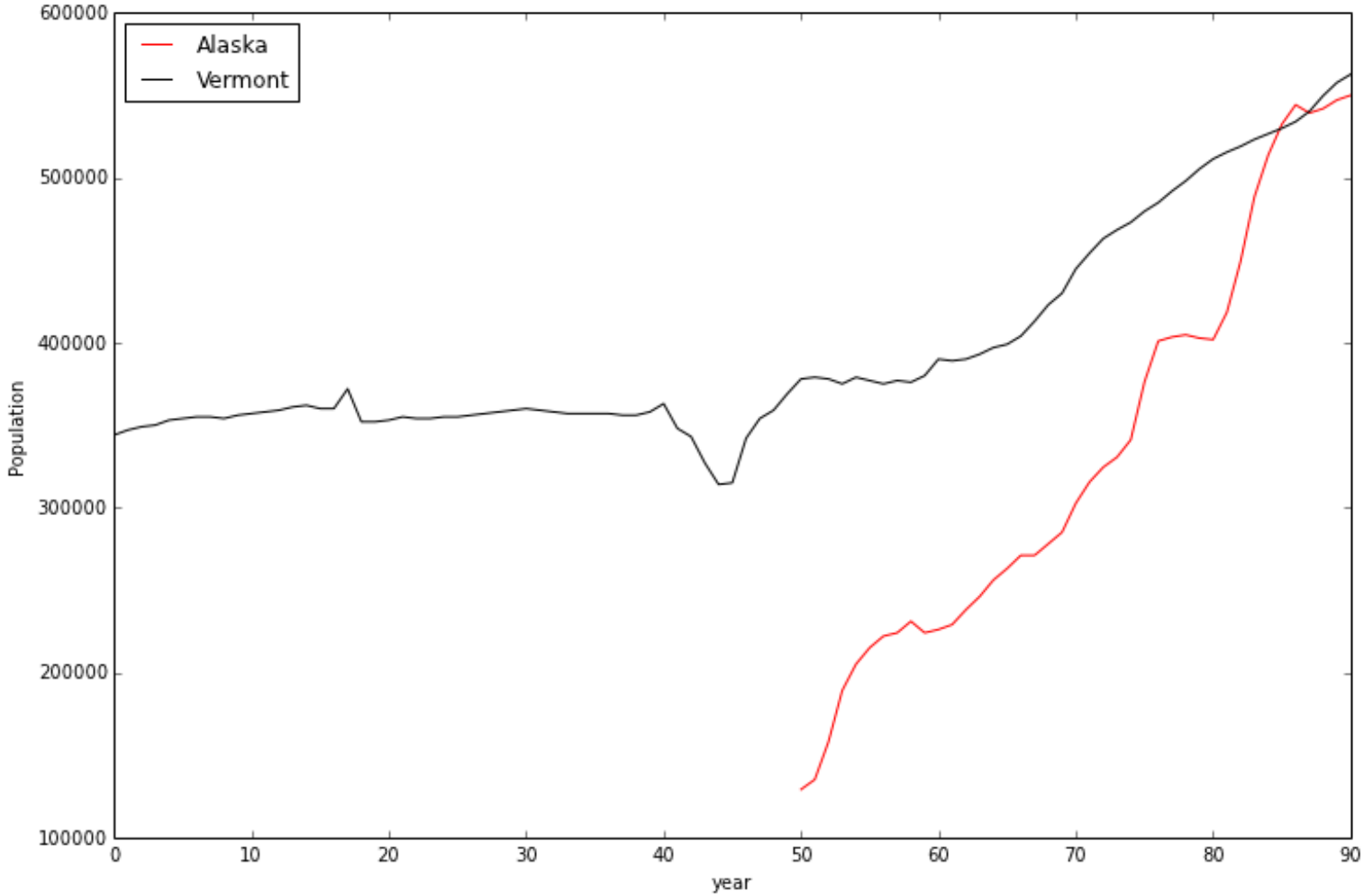
1b: Now we plot

This all went pretty smoothly believe it or not.

Alaska vs. Vermont:

In [5]:
```
import matplotlib.pyplot as plt

x1 = [date for (date, value) in allData["AK"]]
y1 = [value for (date, value) in allData["AK"]]
x2 = [date for (date, value) in allData["VT"]]
y2 = [value for (date, value) in allData["VT"]]

fig = plt.figure(figsize=(12,8))
plt.plot(x1,y1,"r",x2,y2,"k")
plt.xlabel("year")
plt.ylabel("Population")
plt.legend(["Alaska", "Vermont"], loc = 'upper left')
```

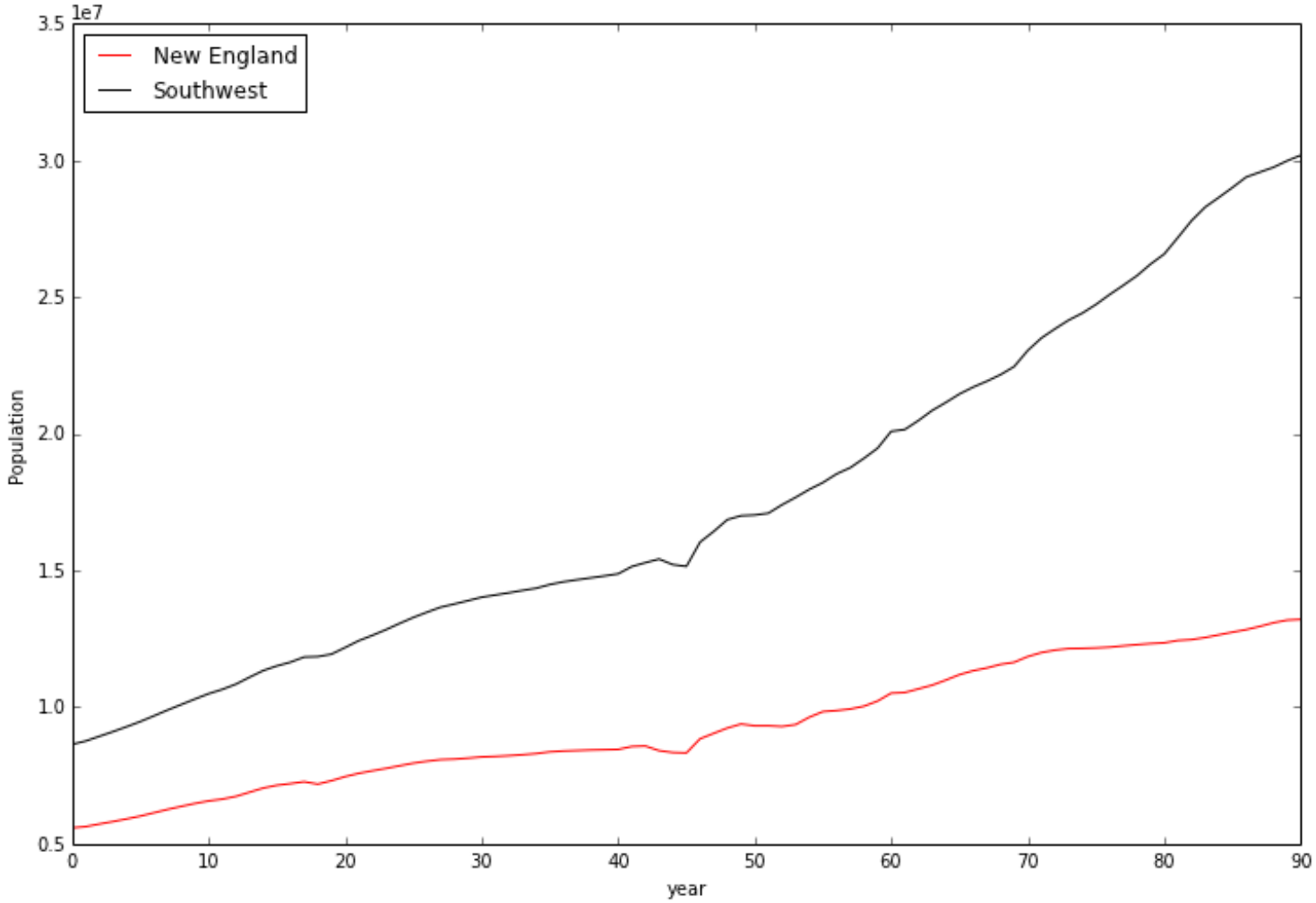
Out[5]:
```
<matplotlib.legend.Legend at 0x10db491d0>
```



New England vs. the American Southwest:

In [6]:
```
from operator import add
newEngland = ["VT", "MA", "ME", "NH", "RI", "CT"] #Actual definition
southWest  = ["NM", "AZ", "CO", "UT", "AZ", "TX"] #My definition

NEx = []
NEy = []

for state in newEngland:
    NEx = [date for (date, value) in allData[state]]
    try:
        NEy = map(add, NEy, [value for (date, value) in allData[state]])
    except: #We cant add a list to a non-existant list.
        NEy = [value for (date, value) in allData[state]]


SWx = []
SWy = []

for state in southWest:
    SWx = [date for (date, value) in allData[state]]
    try:
        SWy = map(add, NEy, [value for (date, value) in allData[state]])
    except:
        SWy = [value for (date, value) in allData[state]]


fig = plt.figure(figsize=(12,8))
plt.plot(NEx,NEy,"r",SWx,SWy,"k")
plt.xlabel("year")
plt.ylabel("Population")
plt.legend(["New England", "Southwest"], loc = 'upper left')
```

Out[6]:
```
<matplotlib.legend.Legend at 0x10dbc7c90>
```



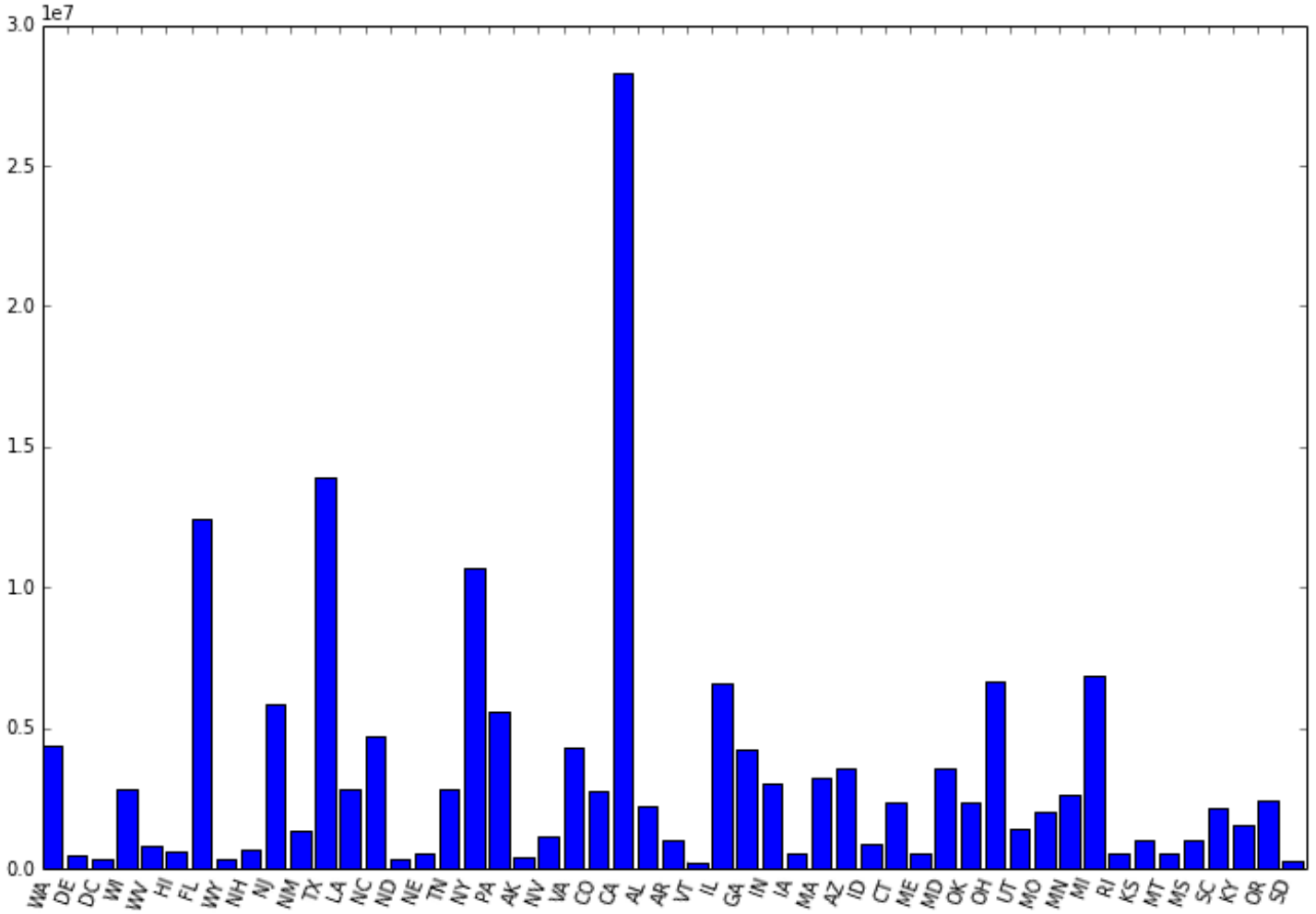
1c: Population change:

In [7]:
```
changeDict = {}
for state in allData.keys():
    startPop = allData[state][0][1]
    endPop   = allData[state][-1][1]
    changeDict[state] = [startPop, endPop]
```

Raw change in numbers:

In [8]:
```
fig = plt.figure(figsize=(12,8))
plt.bar(range(len(changeDict)), [(b - a) for (a,b) in changeDict.values()])
plt.xticks(range(len(changeDict)), changeDict.keys())
plt.xticks(rotation=70)
plt.xlim([0,51])
```

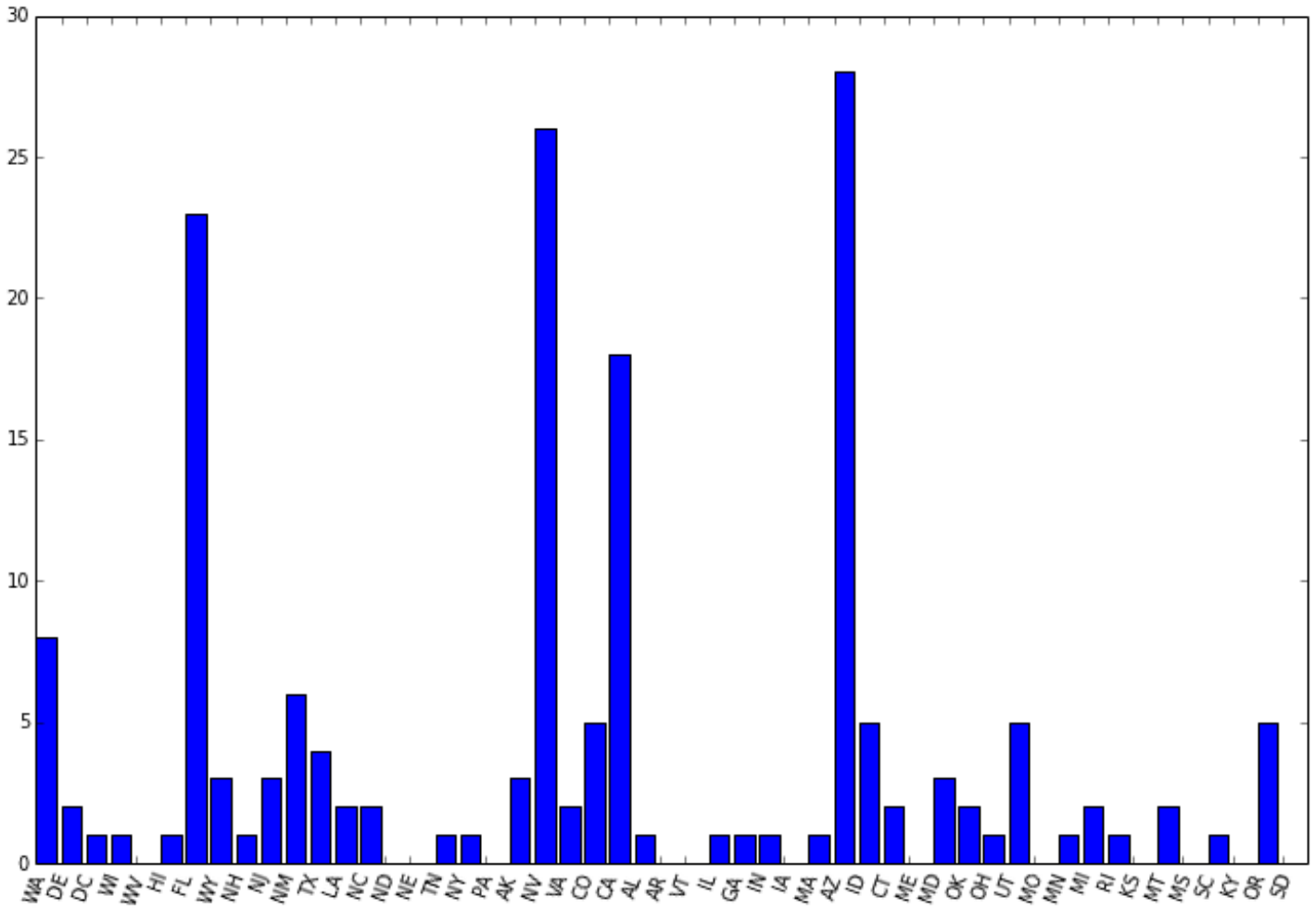
Out[8]: (0, 51)



Change in population by %

In [9]:
```
fig = plt.figure(figsize=(12,8))
plt.bar(range(len(changeDict)), [((b - a)/a) for (a,b) in changeDict.values()]) #change in percent
plt.xticks(range(len(changeDict)), changeDict.keys())
plt.xticks(rotation=70)
plt.xlim([0,51])
```

Out[9]: (0, 51)



Problem 2a

This provided me with way more issues than I expected, I mean it's a csv and I am armed with Pandas, what could

possibly stop me?

I first had to deal with getting meaningful information out of the dates. Since the population data is at the year resolution I decided to downscale the polio data, alterior motivation: interpolation is hard.

I tried for way to long to get the pandas groupby to work with sum and was getting frustrated until I realized that some of the column data was in fact in string format and not integer, maybe due to my replace function.

Note: They should put in a graceful way for list comprehensions to fail, e.g. a try statement; everytime I write a for loop in python I feel dirty.

After the data was summed by year it was all pretty good though. I am a big fan of pandas.

Sources of uncertainty: Given that the data was from a long time ago lots of things could have gone wrong.

- Mis-diagnoses.
- Multiple counting of people close to state lines.
- Counting of the same person for multiple years.

Basically people back then a lot of the time didn't have their stuff together so we should take what they give us with a grain of salt.

In [10]:
```
polioData = pd.read_csv("data/POLIO_Cases_1928-1969.csv", skiprows = 2)
polioData = polioData.replace("-",0)

for key in polioData.keys(): #some of the values are coming in as strings and not ints.
    fixedList = []
    for x in polioData[key]:
        try:
            fixedList.append(eval(x))
        except:
            fixedList.append(x)
    polioData[key] = fixedList

grouped = polioData.groupby("YEAR").sum()
polioByYear = grouped
```

Now I need to convert the written out names to the state abrevs for future combining.

In [11]:
```
#import us #if my installation of python/ pip/ life wasnt messed up this is how I would do this:
stateAbrevs = {
        'AK': 'Alaska',
        'AL': 'Alabama',
        'AR': 'Arkansas',
        'AS': 'American Samoa',
        'AZ': 'Arizona',
        'CA': 'California',
        'CO': 'Colorado',
        'CT': 'Connecticut',
        'DC': 'District of Columbia',
        'DE': 'Delaware',
        'FL': 'Florida',
        'GA': 'Georgia',
        'GU': 'Guam',
        'HI': 'Hawaii',
        'IA': 'Iowa',
        'ID': 'Idaho',
        'IL': 'Illinois',
        'IN': 'Indiana',
        'KS': 'Kansas',
        'KY': 'Kentucky',
        'LA': 'Louisiana',
        'MA': 'Massachusetts',
        'MD': 'Maryland',
        'ME': 'Maine',
        'MI': 'Michigan',
        'MN': 'Minnesota',
        'MO': 'Missouri',
        'MP': 'Northern Mariana Islands',
        'MS': 'Mississippi',
        'MT': 'Montana',
        'NA': 'National',
        'NC': 'North Carolina',
        'ND': 'North Dakota',
        'NE': 'Nebraska',
        'NH': 'New Hampshire',
        'NJ': 'New Jersey',
        'NM': 'New Mexico',
        'NV': 'Nevada',
        'NY': 'New York',
        'OH': 'Ohio',
        'OK': 'Oklahoma',
```

```
        'OR': 'Oregon',
        'PA': 'Pennsylvania',
        'PR': 'Puerto Rico',
        'RI': 'Rhode Island',
        'SC': 'South Carolina',
        'SD': 'South Dakota',
        'TN': 'Tennessee',
        'TX': 'Texas',
        'UT': 'Utah',
        'VA': 'Virginia',
        'VI': 'Virgin Islands',
        'VT': 'Vermont',
        'WA': 'Washington',
        'WI': 'Wisconsin',
        'WV': 'West Virginia',
        'WY': 'Wyoming'
}
flippedStateAbrevs = {}
for key in stateAbrevs.keys():
    flippedStateAbrevs[stateAbrevs[key].upper()] = key #need to be uppercase because shouting!

polioByYear.rename(columns = flippedStateAbrevs, inplace = True)
polioByYear.drop("WEEK",1, inplace = True) #The week col just makes no sense now
polioByYear["U.S"] = polioByYear.sum(axis = 1) #get the sum of the rows for us total
#polioByYear.head()
```

We will start by plotting the trends in polio country-wide:

Annotation info courtesy of historyofvaccines.org

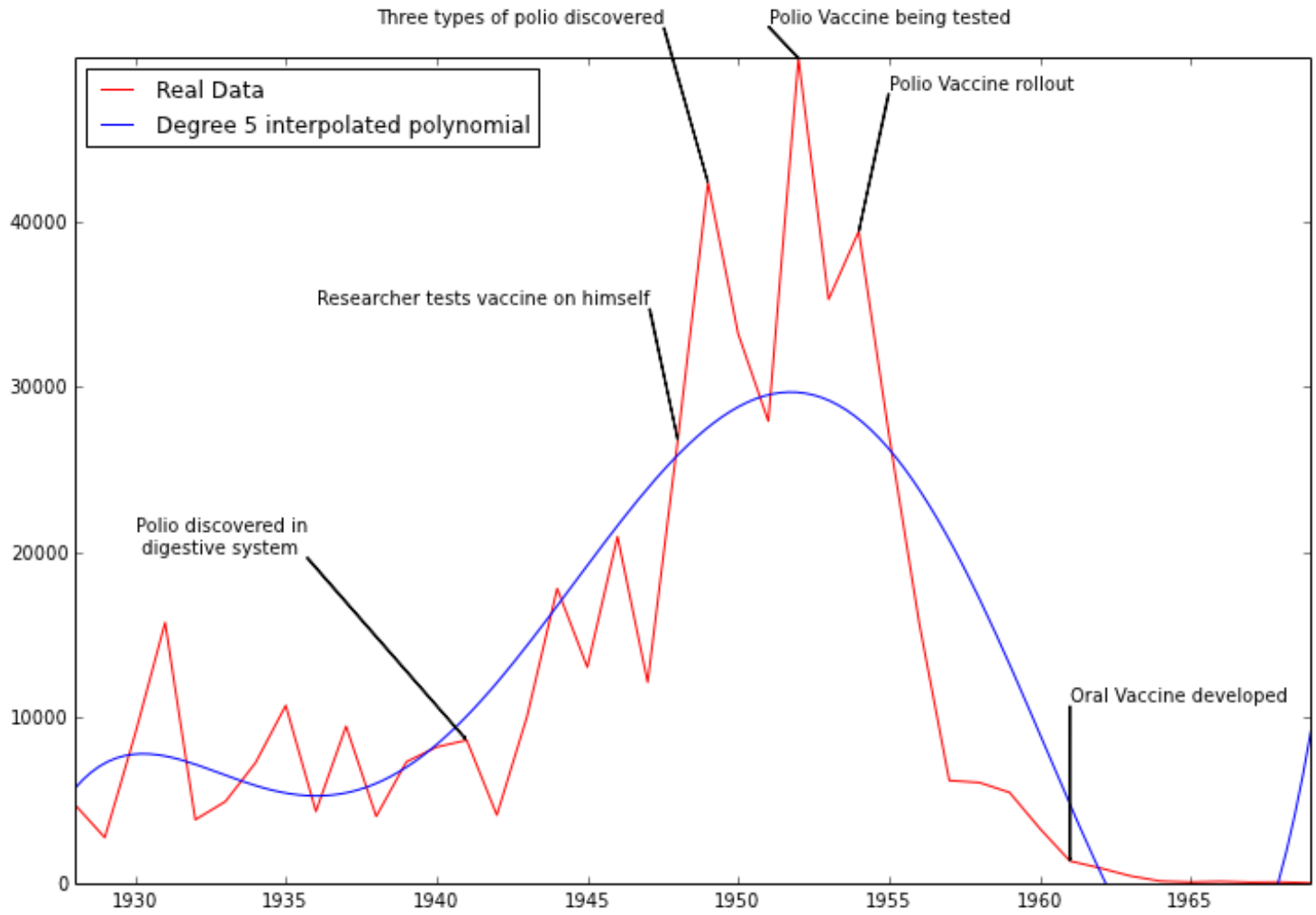
In [12]:

```
x = polioByYear.index.values
y = polioByYear["U.S"]

z = np.polyfit(x, y, 5) #Lets fit an interpolated polynomial to it.
f = np.poly1d(z)

print(z)

x_new = np.linspace(x[0], x[-1], 150)
y_new = f(x_new)

fig = plt.figure(figsize=(12,8))
plt.plot(x,y, 'r', x_new, y_new, 'b')
plt.ylim([0,max(y)])
plt.xlim([min(x), max(x)])

lineSettings = dict(facecolor='black', frac = 0, width = 0.5, headwidth = 1, shrink=0.01)

plt.legend(["Real Data", "Degree 5 interpolated polynomial"], loc = "upper left")


plt.annotate('Polio discovered in\n digestive system', xy=(1941, polioByYear.ix[1941]["U.S"]), xytext=(1930,20000),
            arrowprops=lineSettings,
            )

plt.annotate('Researcher tests vaccine on himself', xy=(1948, polioByYear.ix[1948]["U.S"]), xytext=(1936,35000),
            arrowprops=lineSettings,
            )

plt.annotate('Three types of polio discovered', xy=(1949, polioByYear.ix[1949]["U.S"]), xytext=(1938,52000),
            arrowprops=lineSettings,
            )

plt.annotate('Polio Vaccine being tested', xy=(1952, polioByYear.ix[1952]["U.S"]), xytext=(1951,52000),
            arrowprops=lineSettings,
            )

plt.annotate('Polio Vaccine rollout', xy=(1954, polioByYear.ix[1954]["U.S"]), xytext=(1955,48000),
            arrowprops=lineSettings,
            )
```

```
plt.annotate('Oral Vaccine developed', xy=(1961, polioByYear.ix[1961]["U.S"]), xytext=(1961,11000),
             arrowprops=lineSettings,
             )
```

```
[  2.67716094e-02  -2.60469007e+02   1.01365550e+06  -1.97236592e+09
   1.91887812e+12  -7.46723967e+14]
```

Out[12]: <matplotlib.text.Annotation at 0x10e3c24d0>



Combine all the disparate!

Now I want to combine the two datasets so that I can export it and work some d3.js magic to show areas hit.

I want to get the data into the form

```
{year1: [{state: state1, percentPolio: %polio}, {state: state2, percentPolio: %polio}, ...],
year2: [...],...}
```

Note: Now is the time I curse my decision to get the data working in different forms.

I chose this format due to optimizing my d3.js code.

In [14]:
```
popPercentDict = {}
for year in polioByYear.index:
    keyname = eval(str(year)[2::])
    popPercentDict[keyname] = []
    for state in polioByYear.columns.values:
        try:
            pop = float(dict(allData[state])[ eval(str(year)[2::]) ])
            polioPop = float(polioByYear.ix[year][state])
            percentPolio = polioPop/pop * 100
        except:
            percentPolio = 0
            #some states don't have data for all years, give these 0 for plotting purposes

        popPercentDict[keyname].append({"state": state, "polioPercent":percentPolio})
import json

with open('polioPercent.json', 'w') as outfile:
  json.dump(popPercentDict, outfile)
```

Here is the d3 visualization for the impact of polio on the country. http://www.uvm.edu/~nstrayer/histOfPolio/

Note: I did by state and not as a whole country because of the fact that incomplete polio data indicates not all cases were recorded and so simply the sum across recorded countries would not be accurate.

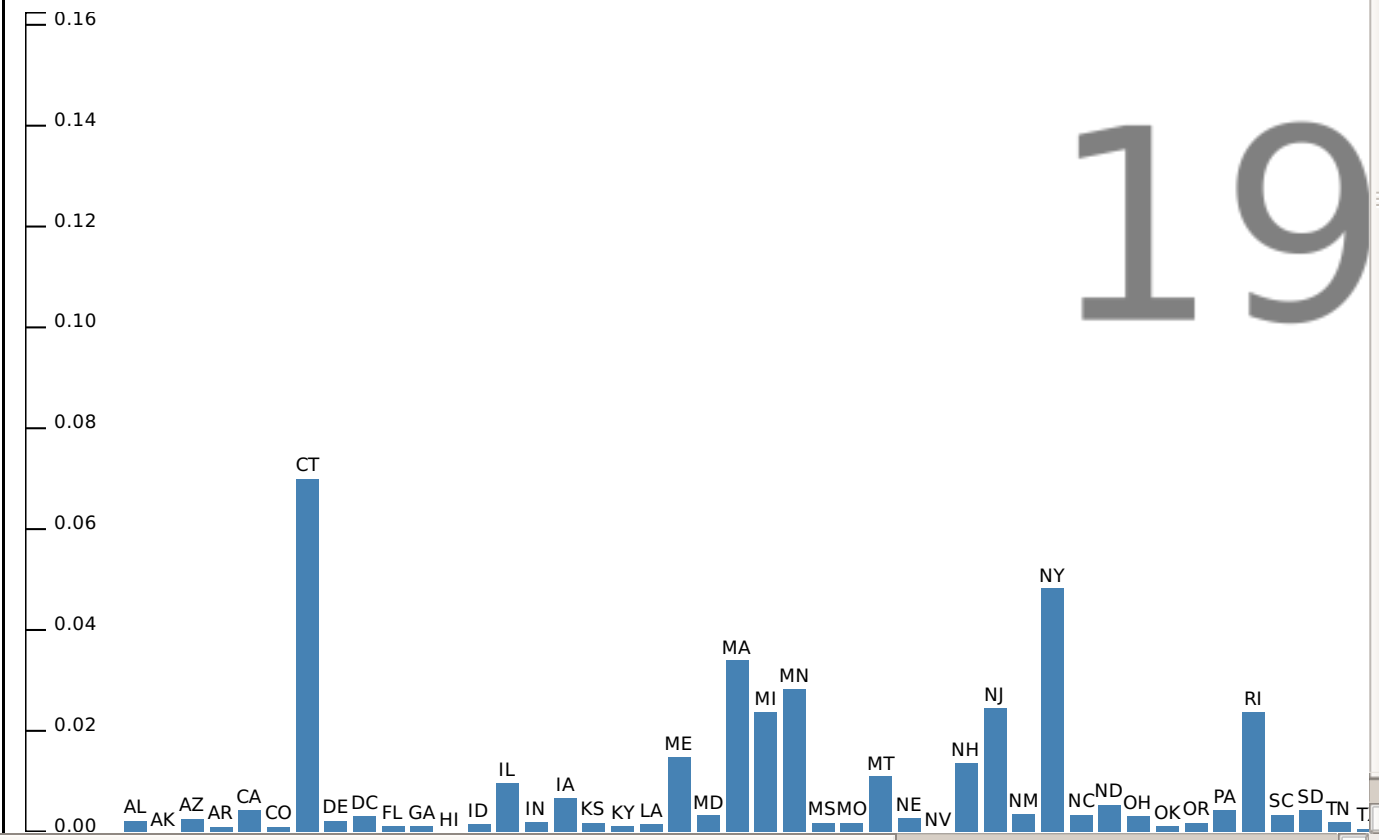
In the iframe output for some reason the css is messed up. For full effect visit the link.

In [29]:
```
from IPython.display import HTML
HTML('<iframe src=http://www.uvm.edu/~nstrayer/histOfPolio/ width=800 height=620></iframe>')
```

Out[29]:

Visualizing the impact of Polio:

Click **here** to watch polio hit the United States or use the slider to manually drag year. population with Polio.



In []:

Back to top