Rajeev_Simpsons_Characters_Analysis

February 10, 2018

1 Printing graphs of names that match a list of names

2 List of Simpsons Character names

3 Note: this is an interactive script with repeated code

```
In [1]: listed_path = "rajeev_data\simpsons_characters\simpsons_character.list"
        totals_title = ""
        top_cutoff = 10

        top_boys_title = ""
        top_girls_title = ""
        last_year = 2017 #change this when Social Security database is updated
        save_path = "rajeev_plots" # files created by this notebook will be saved in this dire

        import time
        import os
        if not os.path.isdir(save_path): # creates path if it does not exist
            os.makedirs(save_path)
        import math

        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
        import seaborn
        print('This is standard output from Rajeev_download_process_files.py')
        %run Rajeev_download_process_files.py

        print('--------------------\nFirst 80 characters of list:')
        listed_file = open(listed_path, "r").read()
        print(listed_file[:80] + ' ...')
        all_listed = eval(listed_file) # make sure you trust this file!
        all_listed_set = set(all_listed) # to remove duplicates
        all_listed = list(all_listed)
        print("all_listed: list of length", len(all_listed))
```

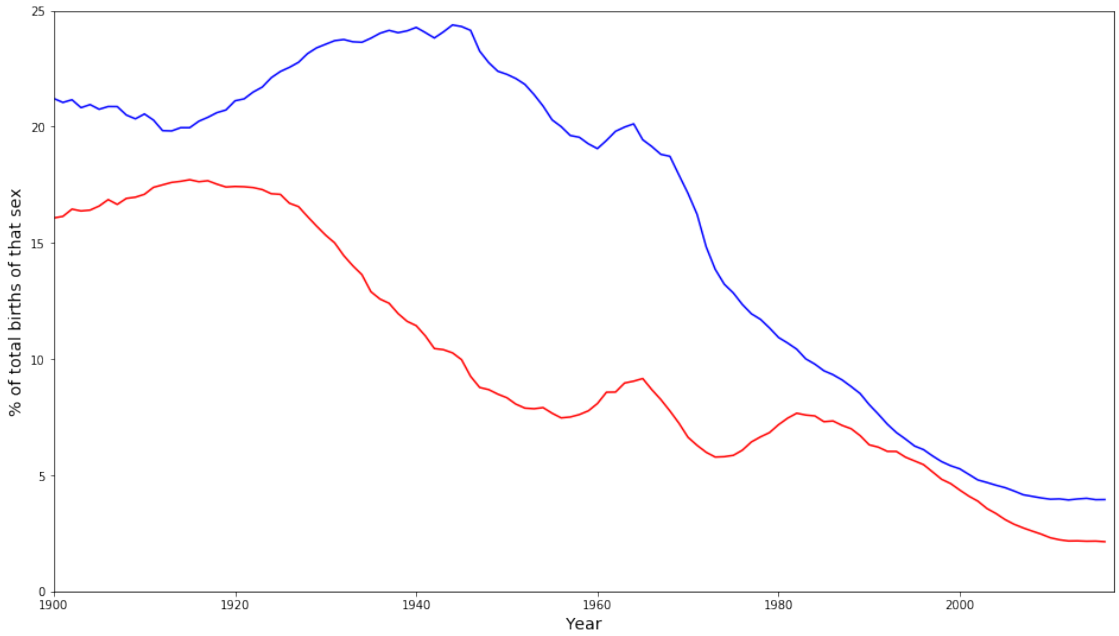
```
# reduce names dataframe to those matching list
print('--------------------\nDataframe names filtered to those that match list')
print("%d records to begin." % (len(names)))
names_listed = names[names.name.isin(all_listed)].copy()
names_listed.sort_values('pct_max', ascending=False, inplace=True)
print("%d records remaining." % (len(names_listed)))
listed_in_df = list(names_listed.name)
print(names_listed.head(10))
listed_m = list(names[(names.sex == 'M') & (names.name.isin(listed_in_df))]['name'])
listed_f = list(names[(names.sex == 'F') & (names.name.isin(listed_in_df))]['name'])

#reduce yob dataframe to those matching list
print('--------------------\nDataframe yob filtered to those that match list (count onl
print("%d records to begin." % (len(yob)))
yob_listed = yob[yob.name.isin(listed_in_df)].copy()
yob_listed.sort_values(['year', 'sex', 'name'], ascending=False, inplace=True)
print("%d records remaining." % (len(yob_listed)))

# m and f totals
yob_listed_f_agg = pd.DataFrame(yob_listed[yob_listed.sex == 'F'].groupby('year').sum(
yob_listed_m_agg = pd.DataFrame(yob_listed[yob_listed.sex == 'M'].groupby('year').sum(
print('--------------------\nHead of total matching list per year, female')
print(yob_listed_f_agg.head())

# print chart of m and f totals
print('\n')

# function to determine a nice y-axis limit a little above the maximum value
# rounds maximum y up to second-most-significant digit
def determine_y_limit(x):
    significance = int(math.floor((math.log10(x))))
    val = math.floor(x / (10 ** (significance - 1))) + 1
    val = val * (10 ** (significance - 1))
    return val

#data
xf = list(yob_listed_f_agg.index)
xm = list(yob_listed_m_agg.index)

plt.figure(figsize=(16,9))
plt.plot(xf, list(yob_listed_f_agg.pct), color="red")
plt.plot(xm, list(yob_listed_m_agg.pct), color="blue")

plt.ylim(0, determine_y_limit(max(list(yob_listed_f_agg.pct)
                                  +list(yob_listed_m_agg.pct))))
plt.xlim(1900, 2017)
plt.title(totals_title, fontsize = 20)
plt.xlabel("Year", fontsize = 14)
plt.ylabel("% of total births of that sex", fontsize = 14)

plt.show()

#function to make dataframe for top names

def top_df(yobdf, names, sexes):
    """ yobdf = dataframe derived from yob; normally it would just be yob itself.
        names = list of names
        sexes = list of length 1 for all the same sex, or same length as names. 'F' an
        """

    df_chart = yobdf.copy()
    assert len(sexes) == 1 or len(names) == len(sexes)
    if len(sexes) == 1:
        sexes = sexes * len(names)

    df_chart = df_chart[df_chart['name'].isin(names)]

    df_chart['temp'] = 0
    for row in range(len(df_chart)):
        for pos in range(len(names)):
            if df_chart.name.iloc[row] == names[pos] and df_chart.sex.iloc[row] == sexe
                df_chart.temp.iloc[row] = 1
    df_chart = df_chart[df_chart.temp == 1]

    print("Tail of dataframe:")
    print(df_chart.tail())

    output_df = pd.DataFrame(pd.pivot_table(df_chart, values='pct', index = 'year', col

    col = output_df.columns[0]

    for yr in range(1900, last_year + 1): #inserts missing years
        if yr not in output_df.index:
            #output_df[col][yr] = 0.0
            output_df = output_df.append(pd.DataFrame(index=[yr], columns=[col], data=

    output_df = output_df.fillna(0)

    return output_df

listed_top_m = top_df(yob, listed_m[:top_cutoff], ['M'])
listed_top_f = top_df(yob, listed_f[:top_cutoff], ['F'])

#a single function to make the four different kinds of charts
```

```
def make_chart(df, form='line', title='', colors= [], smoothing=0, \
               groupedlist = [], baseline='sym', png_name=''):

    dataframe = df.copy()

    startyear = min(list(dataframe.index))
    endyear = max(list(dataframe.index))
    yearstr = '%d-%d' % (startyear, endyear)

    legend_size = 0.01

    has_male = False
    has_female = False
    has_both = False
    max_y = 0
    for name, sex in dataframe.columns:
        max_y = max(max_y, dataframe[(name, sex)].max())
        final_name = name
        if sex == 'M': has_male = True
        if sex == 'F': has_female = True
        if smoothing > 0:
            newvalues = []
            for row in range(len(dataframe)):
                start = max(0, row - smoothing)
                end = min(len(dataframe) - 1, row + smoothing)
                newvalues.append(dataframe[(name, sex)].iloc[start:end].mean())
            for row in range(len(dataframe)):
                dataframe[(name, sex)].iloc[row] = newvalues[row]
    if has_male and has_female:
        y_text = "% of births of indicated sex"
        has_both = True
    elif has_male:
        y_text = "Percent of male births"
    else:
        y_text = "Percent of female births"

    num_series = len(dataframe.columns)

    if colors == []:
        colors = ['#BB2114', '#0C5966', '#BA7814', '#4459AB', '#6B3838',
                  '#B8327B', '#2B947F', '#0D83B5', '#684287', '#8C962C',
                  '#92289E', '#242D7D']
        # my own list of dark contrasting colors
    num_colors = len(colors)

    if num_series > num_colors:
        print("Warning: colors will be repeated.")
```

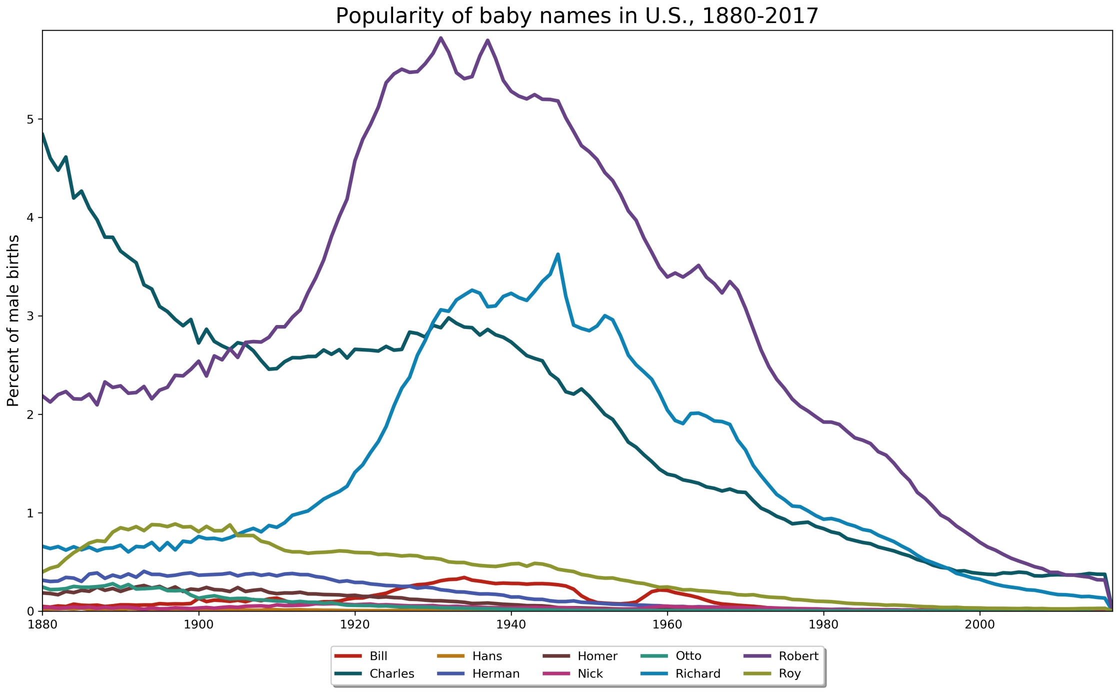
```
if title == '':
    if num_series == 1:
        title = "Popularity of baby name %s in U.S., %s" % (final_name, yearstr)
    else:
        title = "Popularity of baby names in U.S., %s" % (yearstr)

x_values = range(startyear, endyear + 1)
y_zeroes = [0] * (endyear - startyear)

if form == 'line':
    fig, ax = plt.subplots(num=None, figsize=(16, 9), dpi=300, facecolor='w', edgec
    counter = 0
    for name, sex in dataframe.columns:
        color = colors[counter % num_colors]
        counter += 1
        if has_both:
            label = "%s (%s)" % (name, sex)
        else:
            label = name
        ax.plot(x_values, dataframe[(name, sex)], label=label, color=color, linewid
    ax.set_ylim(0,determine_y_limit(max_y))
    ax.set_xlim(startyear, endyear)
    ax.set_ylabel(y_text, size = 13)
    ax.set_title(title, size = 18)
    box = ax.get_position()
    ax.set_position([box.x0, box.y0 + box.height * legend_size,
             box.width, box.height * (1 - legend_size)])
    legend_cols = min(5, num_series)
    ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05), fancybox=True, shad

if form == 'subplots_auto':
    counter = 0
    fig, axes = plt.subplots(num_series, 1, figsize=(12, 3.5*num_series))
    print('Maximum alpha: %d percent' % (determine_y_limit(max_y)))
    for name, sex in dataframe.columns:
        if sex=='M':
            sex_label = 'male'
        else:
            sex_label = 'female'
        label = "Percent of %s births for %s" % (sex_label, name)
        current_ymax = dataframe[(name, sex)].max()
        tint = 1.0 * current_ymax / determine_y_limit(max_y)
        axes[counter].plot(x_values, dataframe[(name, sex)], color='k')
        axes[counter].set_ylim(0,determine_y_limit(current_ymax))
        axes[counter].set_xlim(startyear, endyear)
        axes[counter].fill_between(x_values, dataframe[(name, sex)], color=colors[
            axes[counter].set_ylabel(label, size=11)
            plt.subplots_adjust(hspace=0.1)
            counter += 1

    if form == 'subplots_same':
        counter = 0
        fig, axes = plt.subplots(num_series, 1, figsize=(12, 3.5*num_series))
        print('Maximum y axis: %d percent' % (determine_y_limit(max_y)))
        for name, sex in dataframe.columns:
            if sex=='M':
                sex_label = 'male'
            else:
                sex_label = 'female'
            label = "Percent of %s births for %s" % (sex_label, name)
            axes[counter].plot(x_values, dataframe[(name, sex)], color='k')
            axes[counter].set_ylim(0,determine_y_limit(max_y))
            axes[counter].set_xlim(startyear, endyear)
            axes[counter].fill_between(x_values, dataframe[(name, sex)], color=colors[
            axes[counter].set_ylabel(label, size=11)
            plt.subplots_adjust(hspace=0.1)
            counter += 1

    if form == 'stream':
        plt.figure(num=None, figsize=(20,10), dpi=150, facecolor='w', edgecolor='k')
        plt.title(title, size=17)
        plt.xlim(startyear, endyear)

        if has_both:
            yaxtext = 'Percent of births of indicated sex (scale: '
        elif has_male:
            yaxtext = 'Percent of male births (scale: '
        else:
            yaxtext = 'Percent of female births (scale: '

        scale = str(determine_y_limit(max_y)) + ')'
        yaxtext += scale
        plt.ylabel(yaxtext, size=13)
        polys = plt.stackplot(x_values, *[dataframe[(name, sex)] for name, sex in data
                                 colors=colors, baseline=baseline)
        legendProxies = []
        for poly in polys:
            legendProxies.append(plt.Rectangle((0, 0), 1, 1, fc=poly.get_facecolor()[0
        namelist = []
        for name, sex in dataframe.columns:
            if has_both:
                namelist.append('%s (%s)' % (name, sex))
            else:
                namelist.append(name)
        plt.legend(legendProxies, namelist, loc=3, ncol=2)

        plt.tick_params(\
            axis='y',
            which='both',      #  major and minor ticks
            left='off',
            right='off',
            labelleft='off')

    plt.show()
    if png_name != '':
        filename = save_path + "/" + png_name + ".png"
        plt.savefig(filename)
    plt.close()

# line charts

make_chart(df=listed_top_m,
           form='line', # line , subplots_auto , subplots_same , stream
           title=top_boys_title,
           colors= [],
           smoothing=0,
           baseline='zero',  # zero ,  sym ,  wiggle ,  weighted_wiggle
           )

make_chart(df=listed_top_f,
           form='line', # line , subplots_auto , subplots_same , stream
           title=top_girls_title,
           colors= [],
           smoothing=0,
           baseline='zero',  # zero ,  sym ,  wiggle ,  weighted_wiggle
           )

names_listed.reset_index(drop = True, inplace = True)
names_listed.head()
#names_listed.to_csv("rajeev_data\simpsons_characters\names_matching_list.csv")
```

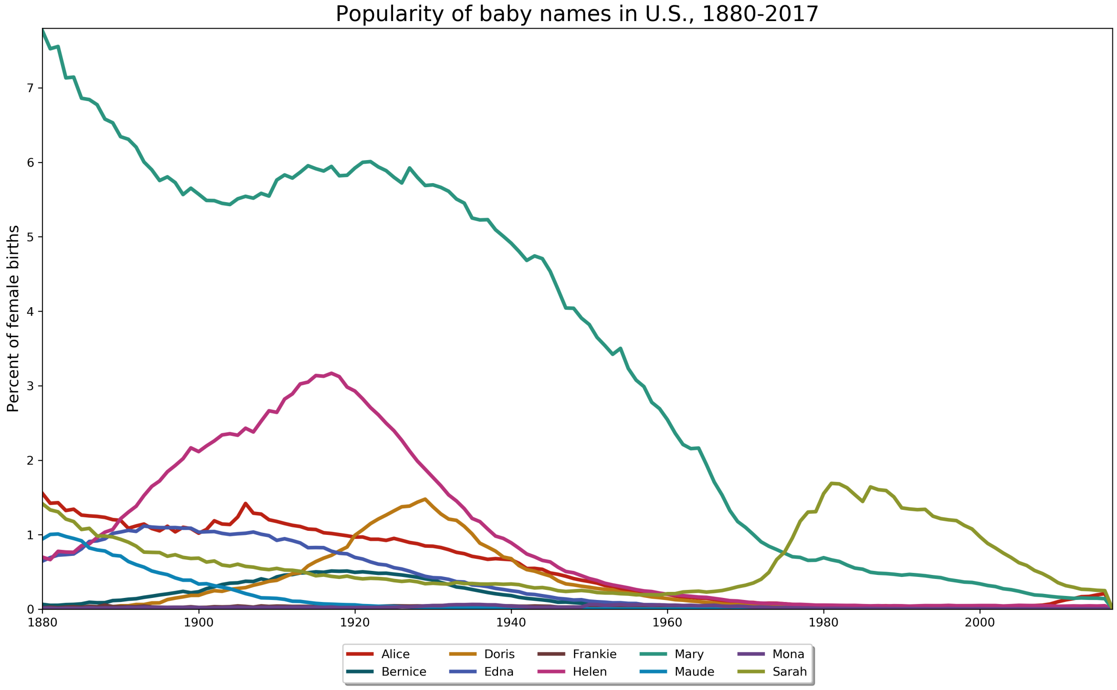
```
This is standard output from Rajeev_download_process_files.py
Data already downloaded.
Data already extracted.
Processing.
Tail of dataframe 'yob':

Tail of dataframe 'names':

Tail of dataframe 'years':
Tail of dataframe 'yob1900':
Tail of dataframe 'names1900':
```

```
Tail of dataframe 'years1900':
--------------------
First 80 characters of list:
["Homer", "Marge", "Bart", "Lisa", "Maggie", "Akira", "Albright", "Aristotle", " ...
all_listed: list of length 143
--------------------
Dataframe names filtered to those that match list
106695 records to begin.
214 records remaining.
            name sex  year_count  year_min  year_max     pct_sum   pct_max
66822       John   M         137      1880      2016  560.205488  8.738268
215         Mary   F         137      1880      2016  474.088924  7.764419
66473     Robert   M         137      1880      2016  399.245833  5.821043
66488    Charles   M         137      1880      2016  266.892668  4.840213
66371    Richard   M         137      1880      2016  190.069663  3.624920
1080        Lisa   F         113      1886      2016   55.329716  3.414340
76         Helen   F         137      1880      2016  129.709519  3.167608
366         Ruth   F         137      1880      2016   99.147235  2.223862
423    Elizabeth   F         137      1880      2016  151.498017  2.130957
66911       Gary   M         135      1880      2016   51.795390  2.026481
--------------------
Dataframe yob filtered to those that match list (count only)
1891894 records to begin.
19056 records remaining.
--------------------
Head of total matching list per year, female
      births        pct
year
1880   15856  17.425708
1881   15497  16.853175
1882   18473  17.128896
1883   18548  16.513827
1884   21207  16.436987
```



```
C:\Users\Shreya\Anaconda3\lib\site-packages\pandas\core\indexing.py:179: SettingWithCopyWarnin
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.htm
  self._setitem_with_indexer(indexer, value)


Tail of dataframe:
           name sex  births  year       pct  ranked  temp
1878895    Nick   M     175  2016  0.009305  1110.0     1
1879130    Hans   M     131  2016  0.006966  1349.0     1
1879636  Herman   M      80  2016  0.004254  1853.5     1
1880830    Bill   M      38  2016  0.002021  3065.0     1
1881989   Homer   M      24  2016  0.001276  4233.5     1
Tail of dataframe:
            name sex  births  year       pct  ranked  temp
1860767     Mona   F     119  2016  0.006774  1739.5     1
1860784    Doris   F     117  2016  0.006660  1761.5     1
1861359     Edna   F      80  2016  0.004554  2335.0     1
1862114  Bernice   F      53  2016  0.003017  3101.0     1
1866090    Maude   F      17  2016  0.000968  7017.5     1
```





```
Out[1]:     name sex  year_count  year_min  year_max     pct_sum   pct_max
        0   John   M         137      1880      2016  560.205488  8.738268
```

```
1      Mary   F         137       1880       2016  474.088924  7.764419
2    Robert   M         137       1880       2016  399.245833  5.821043
3   Charles   M         137       1880       2016  266.892668  4.840213
4   Richard   M         137       1880       2016  190.069663  3.624920
```

4 Refine list

```
In [4]: print(names_listed.name.unique())
```

```
['John' 'Mary' 'Robert' 'Charles' 'Richard' 'Lisa' 'Helen' 'Ruth'
 'Elizabeth' 'Gary' 'Scott' 'Sarah' 'Timothy' 'Alice' 'Willie' 'Doris'
 'Edna' 'Maude' 'Amber' 'Rachel' 'Dewey' 'Gerald' 'Roy' 'Gloria' 'Todd'
 'Ralph' 'Benjamin' 'Roger' 'Carl' 'Charlie' 'Herbert' 'Agnes' 'Maggie'
 'Jacqueline' 'Lindsey' 'Bernice' 'Lewis' 'Allison' 'Troy' 'Terri' 'Johnny'
 'Marvin' 'Herman' 'Tony' 'Chase' 'Martin' 'Bill' 'Eddie' 'Cecil' 'Sherri'
 'Otto' 'Homer' 'Jake' 'Lance' 'Declan' 'Julius' 'Dave' 'Patty' 'Lou'
 'Wendell' 'Kent' 'Kirk' 'Selma' 'Waylon' 'Nelson' 'Ginger' 'Jasper' 'Doug'
 'Luann' 'Marty' 'Nick' 'Seymour' 'Mona' 'Barney' 'Loren' 'Frankie' 'Louie'
 'Desmond' 'Bart' 'Francesca' 'Ned' 'Artie' 'Lionel' 'Hans' 'Raphael' 'Rod'
 'Akira' 'Cletus' 'Murphy' 'Horatio' 'Gino' 'Leopold' 'Moe' 'Jacques'
 'Marge' 'Lenny' 'Dolph' 'Harm' 'Janey' 'Arnie' 'Gil' 'Luigi' 'Sanjay'
 'Kashmir' 'Rainier' 'Aristotle' 'Clancy' 'Jebediah' 'Kwan' 'Kearney'
 'Lurleen' 'Kang' 'Elves' 'Jimbo' 'Mayor']
```

```
In [4]: cutoffn = 0
        # how many names will remain to evaluate after duplicates removed

        from collections import OrderedDict
        evallistm = OrderedDict()
        evallistf = OrderedDict()

        # remove names with more common duplicates in other sex
        # this happens frequently in ssa db

        for name in listed_m:
            try:
                pctf = names_listed[(names_listed.sex == 'F') &
                                    (names_listed.name == name)].pct_max.iloc[0]
                pctm = names_listed[(names_listed.sex == 'M') &
                                    (names_listed.name == name)].pct_max.iloc[0]
            except:
                pctf = 98
                pctm = 99
            if (name not in names_listed[names_listed.sex == 'F'].name.unique() or
                pctf < pctm):
                evallistm[name] = ''
        for name in listed_f:
            try:
                pctf = names_listed[(names_listed.sex == 'F') &
                                    (names_listed.name == name)].pct_max.iloc[0]
                pctm = names_listed[(names_listed.sex == 'M') &
                                    (names_listed.name == name)].pct_max.iloc[0]
            except:
                pctf = 99
                pctm = 98
            if (name not in names_listed[names_listed.sex == 'M'].name.unique() or
                pctm < pctf):
                evallistf[name] = ''

        if cutoffn > 0:
            assert len(evallistm) > cutoffn
            assert len(evallistf) > cutoffn
            print(evallistm[:cutoffn])
            print(evallistf[:cutoffn])
        else:
            print('Length of lists: %d male, %d female\n' % (len(evallistm), len(evallistf)))
            print(evallistm)
            print(' ')
            print(evallistf)
```

```
Length of lists: 80 male, 35 female

OrderedDict([('Richard', ''), ('Roy', ''), ('Otto', ''), ('Hans', ''), ('Nick', ''), ('Bill',

OrderedDict([('Alice', ''), ('Bernice', ''), ('Maude', ''), ('Helen', ''), ('Doris', ''), ('Sa
```

```
In [5]: #manually copy and paste the above lists and assign
        #'acc' or 'rej' individually to accept or reject

        evallistm = OrderedDict([('Cecil', 'acc'), ('Jake', 'acc'), ('John', 'acc'), ('Dave',

        evallistf = OrderedDict([('Elizabeth', 'acc'), ('Rachel', 'acc'), ('Maggie', 'acc'), (

        # Test that all names have 'acc' or 'rej' values

        final_m = []
        final_f = []

        names_not_validated = []
        for item in evallistm:
            if evallistm[item] not in ['acc', 'rej']:
                names_not_validated.append(item)
            elif evallistm[item] == 'acc':
```

```
        final_m.append(item)
for item in evallistf:
    if evallistf[item] not in ['acc', 'rej']:
        names_not_validated.append(item)
    elif evallistf[item] == 'acc':
        final_f.append(item)

final_all = final_m + final_f

if len(names_not_validated) > 0:
    print("The following names do not have 'acc' or 'rej' values: ", names_not_validate
    raise exception("Names not validated")

print('Accepted male names:', final_m)
print('Accepted female names:', final_f)

print('Length: %d male, %d female\n' % (len(final_m), len(final_f)))

cutmin = min(len(final_m), len(final_f))

final_m = final_m[:cutmin]
final_f = final_f[:cutmin]

print('After resizing to %d names each:' % (cutmin))
print('Accepted male names:', final_m)
print('Accepted female names:', final_f)
```

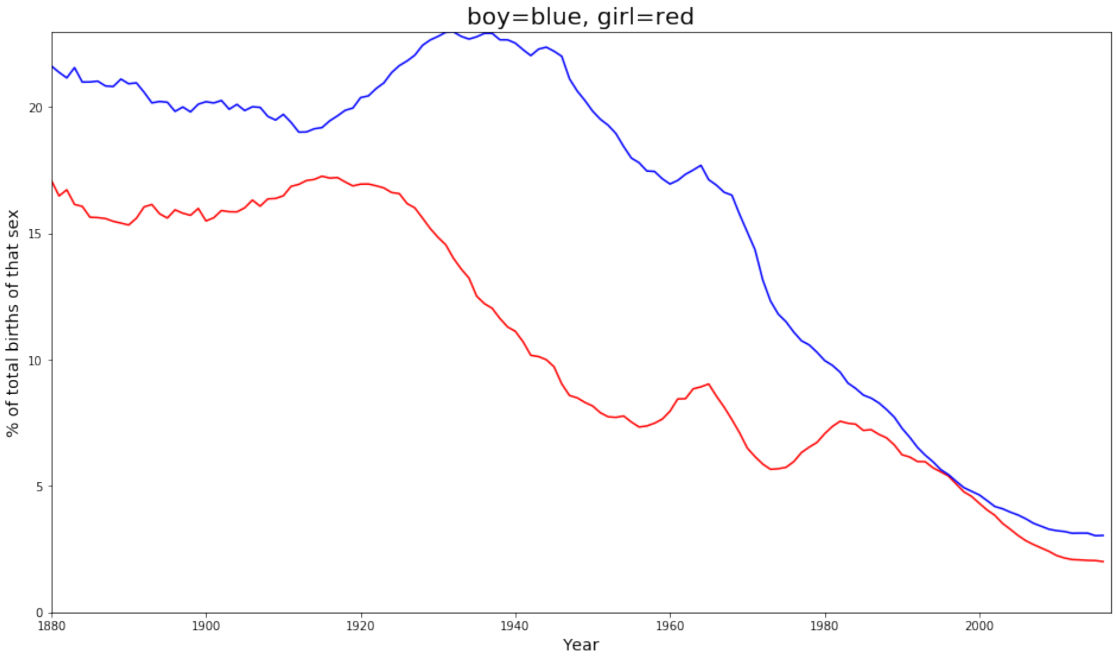
```
Accepted male names: ['Cecil', 'Jake', 'John', 'Dave', 'Herbert', 'Nelson', 'Tony', 'Ralph', '
Accepted female names: ['Elizabeth', 'Rachel', 'Maggie', 'Ruth', 'Selma', 'Maude', 'Agnes', 'F
Length: 80 male, 35 female

After resizing to 35 names each:
Accepted male names: ['Cecil', 'Jake', 'John', 'Dave', 'Herbert', 'Nelson', 'Tony', 'Ralph', '
Accepted female names: ['Elizabeth', 'Rachel', 'Maggie', 'Ruth', 'Selma', 'Maude', 'Agnes', 'F
```

```
In [6]: %run Rajeev_download_process_files.py

        # reduce names dataframe to those matching list
        # print '--------------------\nDataframe names filtered to those that match list'
        # print "%d records to begin." % (len(names))
        names_listed = names[((names.name.isin(final_m) & (names.sex == 'M')) |
                              (names.name.isin(final_f) & (names.sex == 'F')) )].copy()
        names_listed.sort_values('pct_max', ascending=False, inplace=True)
        # print "%d records remaining." % (len(names_listed))
        # listed_in_df = list(names_listed.name)
        # print names_listed.head(10)
        # listed_m = list(names[(names.sex == 'M') & (names.name.isin(final_m))]['name'])
        # listed_f = list(names[(names.sex == 'F') & (names.name.isin(final_f))]['name'])

        #reduce yob dataframe to those matching list
        print('--------------------\nDataframe yob filtered to those that match list (count onl
        print("%d records to begin." % (len(yob)))
        yob_listed_m = yob[(yob.name.isin(final_m)) & (yob.sex == 'M')].copy()
        yob_listed_m.sort_values(['year', 'sex', 'name'], ascending=False, inplace=True)
        yob_listed_f = yob[(yob.name.isin(final_f)) & (yob.sex == 'F')].copy()
        yob_listed_f.sort_values(['year', 'sex', 'name'], ascending=False, inplace=True)
        print("%d records remaining." % (len(yob_listed)))

        # m and f totals
        yob_listed_f_agg = pd.DataFrame(yob_listed_f.groupby('year').sum())[['births', 'pct']]
        yob_listed_m_agg = pd.DataFrame(yob_listed_m.groupby('year').sum())[['births', 'pct']]
        print('--------------------\nHead of total matching list per year, female')
        print(yob_listed_f_agg.head())

        # print chart of m and f totals
        print('\n')

        # function to determine a nice y-axis limit a little above the maximum value
        # rounds maximum y up to second-most-significant digit
        def determine_y_limit(x):
            significance = int(math.floor((math.log10(x))))
            val = math.floor(x / (10 ** (significance - 1))) + 1
            val = val * (10 ** (significance - 1))
            return val

        #data
        xf = list(yob_listed_f_agg.index)
        xm = list(yob_listed_m_agg.index)

        plt.figure(figsize=(16,9))
        plt.plot(xf, list(yob_listed_f_agg.pct), color="red")
        plt.plot(xm, list(yob_listed_m_agg.pct), color="blue")

        plt.ylim(0, determine_y_limit(max(list(yob_listed_f_agg.pct)
                                          +list(yob_listed_m_agg.pct))))
        plt.xlim(1880, 2017)

        plt.title('boy=blue, girl=red', fontsize = 20)
        plt.xlabel("Year", fontsize = 14)
        plt.ylabel("% of total births of that sex", fontsize = 14)

        plt.show()
```

```
Data already downloaded.
Data already extracted.
```

```
Reading from pickle.
Tail of dataframe 'yob':

Tail of dataframe 'names':

Tail of dataframe 'years':
Tail of dataframe 'yob1900':
Tail of dataframe 'names1900':
Tail of dataframe 'years1900':
--------------------
Dataframe yob filtered to those that match list (count only)
1891894 records to begin.
19056 records remaining.
--------------------
Head of total matching list per year, female
      births        pct
year
1880   15556  17.096008
1881   15156  16.482333
1882   18037  16.724619
1883   18134  16.145231
1884   20731  16.068051
```

```
<matplotlib.figure.Figure at 0x1d38040d668>
```



```
In [7]: # all names_listed, so we can see which ones to aggregate
        # cutoff of 10 already done

        print(names_listed[names_listed.sex == 'M'].head(50))
        print('')
        print(names_listed[names_listed.sex == 'F'].head(50))
```

```
           name sex  year_count  year_min  year_max     pct_sum   pct_max
66822      John   M         137      1880      2016  560.205488  8.738268
66473    Robert   M         137      1880      2016  399.245833  5.821043
66488   Charles   M         137      1880      2016  266.892668  4.840213
66371   Richard   M         137      1880      2016  190.069663  3.624920
66815     Scott   M         137      1880      2016   42.881843  1.747625
66759   Timothy   M         137      1880      2016   59.157969  1.639091
66620    Willie   M         137      1880      2016   60.765320  1.491646
66565    Gerald   M         137      1880      2016   33.913051  0.908144
66405       Roy   M         137      1880      2016   51.758053  0.885976
66720     Ralph   M         137      1880      2016   49.215852  0.805214
66611  Benjamin   M         137      1880      2016   52.787020  0.801269
66768     Roger   M         137      1880      2016   29.562580  0.785572
66527      Carl   M         137      1880      2016   55.560453  0.779285
66571   Charlie   M         137      1880      2016   30.600348  0.746913
66689   Herbert   M         137      1880      2016   32.539698  0.731688
66694     Lewis   M         137      1880      2016   21.367361  0.467911
66781      Troy   M         137      1880      2016   12.991100  0.459021
66733    Johnny   M         137      1880      2016   19.957072  0.423664
66675    Marvin   M         137      1880      2016   21.636650  0.419294
66597      Tony   M         137      1880      2016   16.204490  0.395962
66596    Martin   M         137      1880      2016   27.916805  0.360328
66460      Bill   M         137      1880      2016   13.969894  0.342322
66584     Eddie   M         137      1880      2016   19.947310  0.306362
66602     Cecil   M         137      1880      2016   13.506134  0.288605
66414      Otto   M         137      1880      2016    8.215615  0.279433
66499     Homer   M         137      1880      2016   11.511944  0.260869
66763      Jake   M         137      1880      2016    8.998176  0.225385
66588    Julius   M         137      1880      2016    9.994816  0.180213
66664      Dave   M         137      1880      2016    7.085158  0.155381
66562    Nelson   M         137      1880      2016    7.383460  0.100743
66456      Nick   M         137      1880      2016    4.581789  0.076175
66793     Loren   M         137      1880      2016    3.993729  0.057171
66679     Louie   M         137      1880      2016    3.227575  0.054475
66710       Ned   M         137      1880      2016    2.126858  0.044134
66427      Hans   M         137      1880      2016    1.422626  0.030613

            name sex  year_count  year_min  year_max     pct_sum   pct_max
```

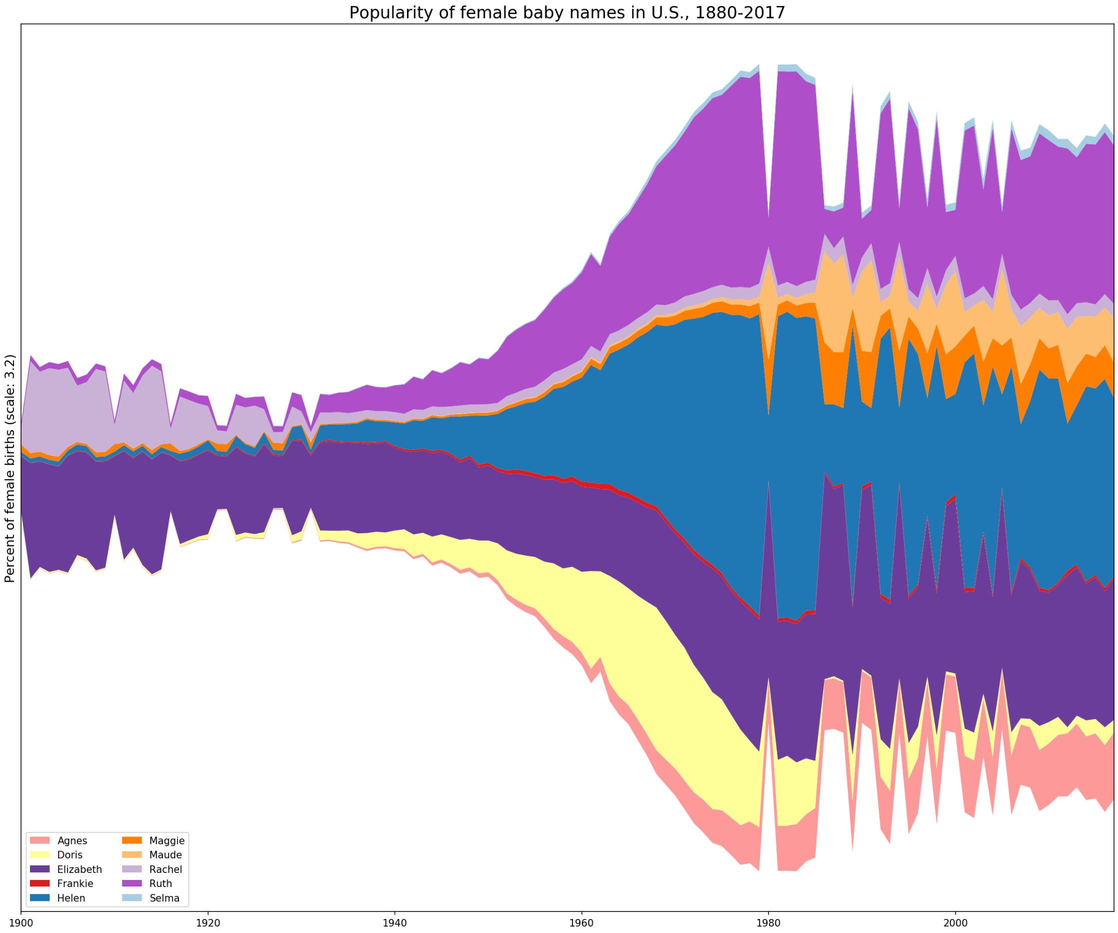
```
215          Mary   F        137      1880      2016  474.088924  7.764419
1080         Lisa   F        113      1886      2016   55.329716  3.414340
76          Helen   F        137      1880      2016  129.709519  3.167608
366          Ruth   F        137      1880      2016   99.147235  2.223862
423     Elizabeth   F        137      1880      2016  151.498017  2.130957
108         Sarah   F        137      1880      2016   89.578481  1.689169
3           Alice   F        137      1880      2016   76.002337  1.553983
77          Doris   F        137      1880      2016   41.531301  1.477660
282          Edna   F        137      1880      2016   50.300873  1.116930
19          Maude   F        137      1880      2016   18.234735  1.009764
571         Amber   F        133      1880      2016   21.480472  0.988335
386        Rachel   F        137      1880      2016   37.953275  0.950824
753        Gloria   F        126      1881      2016   28.828671  0.844510
369         Agnes   F        137      1880      2016   29.208190  0.695198
347        Maggie   F        137      1880      2016   18.807296  0.639617
793    Jacqueline   F        124      1891      2016   26.018376  0.631908
2692      Lindsey   F         78      1924      2016    8.864843  0.535160
14        Bernice   F        137      1880      2016   20.778847  0.512814
1229      Allison   F        108      1908      2016   16.630456  0.462202
1917        Terri   F         92      1922      2016    7.993503  0.434133
2292       Sherri   F         84      1924      2016    4.719302  0.283142
757         Patty   F        126      1880      2016    4.077445  0.154119
533           Lou   F        135      1880      2016    4.271691  0.143969
309         Selma   F        137      1880      2016    4.231291  0.111062
1406       Ginger   F        104      1913      2016    2.721773  0.100383
1922        Luann   F         92      1921      2016    0.843730  0.084346
109          Mona   F        137      1880      2016    3.554248  0.063806
125       Frankie   F        137      1880      2016    3.307343  0.056509
1057    Francesca   F        114      1895      2016    1.629999  0.044794
1521        Artie   F        101      1880      1989    1.314750  0.037578
7581        Akira   F         42      1973      2016    0.319541  0.022403
2484        Marge   F         81      1889      1976    0.372193  0.009437
986         Janey   F        116      1884      2016    0.322025  0.006886
19709     Lurleen   F         17      1913      1969    0.010244  0.001219
25520        Kang   F         12      1980      1995    0.006164  0.001078
```

4.0.1 Popularity of Simpsons Character female names

```
In [8]: names = final_f[:10]
        sexes = ['F'] # can be length 1 or same length as names

        yearstart=1880 # for data, not graph
        yearend=2017

        xmin = 1900

        start = time.time()
```

```
df_chart = yob.copy()
if len(sexes) == 1:
    sexes = sexes * len(names)

df_chart = df_chart[df_chart['name'].isin(names)]

df_chart['temp'] = 0
for row in range(len(df_chart)):
    for pos in range(len(names)):
        if df_chart.name.iloc[row] == names[pos] and df_chart.sex.iloc[row] == sexes[po
            df_chart.temp.iloc[row] = 1
df_chart = df_chart[df_chart.temp == 1]


#To keep more than one data set for charts in memory, change name of chart_1

chart_1 = pd.DataFrame(pd.pivot_table(df_chart, values='pct', index = 'year', columns=

col = chart_1.columns[0]

for yr in range(yearstart, yearend+1): #inserts missing years
    if yr not in chart_1.index:
        #chart_1[col][yr] = 0.0
        chart_1 = chart_1.append(pd.DataFrame(index=[yr], columns=[col], data=[0.0]))

chart_1 = chart_1.fillna(0)

chart_1.sort_values(by=[col], inplace=True, ascending=True)

#a single function to make the four different kinds of charts

def make_chart(df=chart_1, form='line', title='', colors= [], smoothing=0, \
               groupedlist = [], baseline='sym', png_name=''):

    dataframe = df.copy()

    startyear = min(list(dataframe.index))
    endyear = max(list(dataframe.index))
    yearstr = '%d-%d' % (startyear, endyear)

    legend_size = 0.01

    has_male = False
    has_female = False
    has_both = False
    max_y = 0
    for name, sex in dataframe.columns:
        max_y = max(max_y, dataframe[(name, sex)].max())
        final_name = name
        if sex == 'M': has_male = True
        if sex == 'F': has_female = True
        if smoothing > 0:
            newvalues = []
            for row in range(len(dataframe)):
                start = max(0, row - smoothing)
                end = min(len(dataframe) - 1, row + smoothing)
                newvalues.append(dataframe[(name, sex)].iloc[start:end].mean())
            for row in range(len(dataframe)):
                dataframe[(name, sex)].iloc[row] = newvalues[row]
    if has_male and has_female:
        y_text = "% of births of indicated sex"
        has_both = True
    elif has_male:
        y_text = "Percent of male births"
    else:
        y_text = "Percent of female births"

    num_series = len(dataframe.columns)

    if colors == []:
       colors = ["#1f78b4","#ae4ec9","#33a02c","#fb9a99","#e31a1c","#a6cee3",
                 "#fdbf6f","#ff7f00","#cab2d6","#6a3d9a","#ffff99","#b15928"]
        #colors = ['#ff0000', '#b00000', '#870000', '#550000', '#e4e400', '#baba00', '
    from random import shuffle
    shuffle(colors)
    num_colors = len(colors)

    if num_series > num_colors:
        print("Warning: colors will be repeated.")

    if title == '':
        if num_series == 1:
            title = "Popularity of female baby name %s in U.S., %s" % (final_name, yea
        else:
            title = "Popularity of female baby names in U.S., %s" % (yearstr)

    x_values = range(startyear, endyear + 1)
    y_zeroes = [0] * (endyear - startyear)

    if form == 'line':
        fig, ax = plt.subplots(num=None, figsize=(16, 9), dpi=300, facecolor='w', edge
        counter = 0
        for name, sex in dataframe.columns:
            color = colors[counter % num_colors]
            counter += 1
            if has_both:
                label = "%s (%s)" % (name, sex)
            else:
                label = name
            ax.plot(x_values, dataframe[(name, sex)], label=label, color=color, linewi
        ax.set_ylim(0,determine_y_limit(max_y))
        ax.set_xlim(xmin, endyear)
        ax.set_ylabel(y_text, size = 13)
        box = ax.get_position()
        ax.set_position([box.x0, box.y0 + box.height * legend_size,
                         box.width, box.height * (1 - legend_size)])
        legend_cols = min(5, num_series)
        ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05), fancybox=True, shad

    if form == 'subplots_auto':
        counter = 0
        fig, axes = plt.subplots(num_series, 1, figsize=(12, 3.5*num_series))
        print('Maximum alpha: %d percent' % (determine_y_limit(max_y)))
        for name, sex in dataframe.columns:
            if sex=='M':
                sex_label = 'male'
            else:
                sex_label = 'female'
            label = "Percent of %s births for %s" % (sex_label, name)
            current_ymax = dataframe[(name, sex)].max()
            tint = 1.0 * current_ymax / determine_y_limit(max_y)
            axes[counter].plot(x_values, dataframe[(name, sex)], color='k')
            axes[counter].set_ylim(0,determine_y_limit(current_ymax))
            axes[counter].set_xlim(xmin, endyear)
            axes[counter].fill_between(x_values, dataframe[(name, sex)], color=colors[

            axes[counter].set_ylabel(label, size=11)
            plt.subplots_adjust(hspace=0.1)
            counter += 1

    if form == 'subplots_same':
        counter = 0
        fig, axes = plt.subplots(num_series, 1, figsize=(12, 3.5*num_series))
        print('Maximum y axis: %d percent' % (determine_y_limit(max_y)))
        for name, sex in dataframe.columns:
            if sex=='M':
                sex_label = 'male'
            else:
                sex_label = 'female'
            label = "Percent of %s births for %s" % (sex_label, name)
            axes[counter].plot(x_values, dataframe[(name, sex)], color='k')
            axes[counter].set_ylim(0,determine_y_limit(max_y))
            axes[counter].set_xlim(xmin, endyear)
            axes[counter].fill_between(x_values, dataframe[(name, sex)], color=colors[
            axes[counter].set_ylabel(label, size=11)
            plt.subplots_adjust(hspace=0.1)
            counter += 1

    if form == 'stream':
        plt.figure(num=None, figsize=(20,16.67), dpi=150, facecolor='w', edgecolor='k'
        plt.title(title, size=17)
        plt.xlim(xmin, endyear)

        if has_both:
            yaxtext = 'Percent of births of indicated sex (scale: '
        elif has_male:
            yaxtext = 'Percent of male births (scale: '
        else:
            yaxtext = 'Percent of female births (scale: '

        scale = str(determine_y_limit(max_y)) + ')'
        yaxtext += scale
        plt.ylabel(yaxtext, size=13)
        polys = plt.stackplot(x_values, *[dataframe[(name, sex)] for name, sex in data
                                 colors=colors, baseline=baseline)
        legendProxies = []
        for poly in polys:
            legendProxies.append(plt.Rectangle((0, 0), 1, 1, fc=poly.get_facecolor()[0
        namelist = []
        for name, sex in dataframe.columns:
            if has_both:
                namelist.append('%s (%s)' % (name, sex))
            else:
                namelist.append(name)
        plt.legend(legendProxies, namelist, loc=3, ncol=2)

        plt.tick_params(\
            axis='y',
            which='both',      #  major and minor ticks
            left='off',
            right='off',
            labelleft='off')

    plt.show()
    if png_name != '':
        filename = save_path + "/" + png_name + ".png"
        plt.savefig(filename)
    plt.close()

#stream graph

make_chart(df=chart_1,
                form='stream', # line , subplots_auto , subplots_same , stream
                title='',
                colors= [],
                smoothing=0,
                baseline='sym',  # zero ,  sym ,  wiggle ,  weighted_wiggle
                png_name = 'simpsons_female_names',  # if '', will not be saved
                )
```

```
C:\Users\Shreya\Anaconda3\lib\site-packages\pandas\core\indexing.py:179: SettingWithCopyWarnin
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.htm
  self._setitem_with_indexer(indexer, value)
```



4.0.2 Popularity of Simpsons Character male names

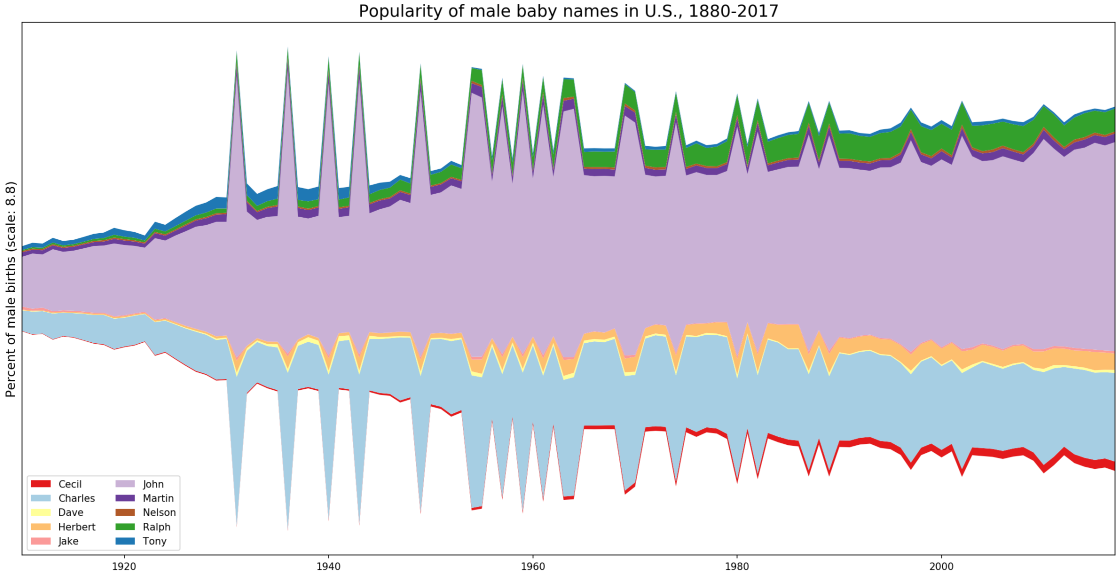
```
In [9]: names = final_m[:10]
        sexes = ['M'] # can be length 1 or same length as names
```

```
yearstart=1880 # for data, not graph
yearend=2017

xmin = 1910

start = time.time()
df_chart = yob.copy()
if len(sexes) == 1:
    sexes = sexes * len(names)

df_chart = df_chart[df_chart['name'].isin(names)]

df_chart['temp'] = 0
for row in range(len(df_chart)):
    for pos in range(len(names)):
        if df_chart.name.iloc[row] == names[pos] and df_chart.sex.iloc[row] == sexes[po
            df_chart.temp.iloc[row] = 1
df_chart = df_chart[df_chart.temp == 1]


#To keep more than one data set for charts in memory, change name of chart_1

chart_1 = pd.DataFrame(pd.pivot_table(df_chart, values='pct', index = 'year', columns=

col = chart_1.columns[0]

for yr in range(yearstart, yearend+1): #inserts missing years
    if yr not in chart_1.index:
        #chart_1[col][yr] = 0.0
        chart_1 = chart_1.append(pd.DataFrame(index=[yr], columns=[col], data=[0.0]))

chart_1 = chart_1.fillna(0)

chart_1.sort_values(by=[col], inplace=True, ascending=True)

#a single function to make the four different kinds of charts

def make_chart(df=chart_1, form='line', title='', colors= [], smoothing=0, \
               groupedlist = [], baseline='sym', png_name=''):

    dataframe = df.copy()

    startyear = min(list(dataframe.index))
    endyear = max(list(dataframe.index))
    yearstr = '%d-%d' % (startyear, endyear)

    legend_size = 0.01
```

```
has_male = False
has_female = False
has_both = False
max_y = 0
for name, sex in dataframe.columns:
    max_y = max(max_y, dataframe[(name, sex)].max())
    final_name = name
    if sex == 'M': has_male = True
    if sex == 'F': has_female = True
    if smoothing > 0:
        newvalues = []
        for row in range(len(dataframe)):
            start = max(0, row - smoothing)
            end = min(len(dataframe) - 1, row + smoothing)
            newvalues.append(dataframe[(name, sex)].iloc[start:end].mean())
        for row in range(len(dataframe)):
            dataframe[(name, sex)].iloc[row] = newvalues[row]
if has_male and has_female:
    y_text = "% of births of indicated sex"
    has_both = True
elif has_male:
    y_text = "Percent of male births"
else:
    y_text = "Percent of female births"

num_series = len(dataframe.columns)

if colors == []:
   colors = ["#1f78b4","#ae4ec9","#33a02c","#fb9a99","#e31a1c","#a6cee3",
             "#fdbf6f","#ff7f00","#cab2d6","#6a3d9a","#ffff99","#b15928"]
    #colors = ['#ff0000', '#b00000', '#870000', '#550000', '#e4e400', '#baba00', '
from random import shuffle
shuffle(colors)
num_colors = len(colors)

if num_series > num_colors:
    print("Warning: colors will be repeated.")

if title == '':
    if num_series == 1:
        title = "Popularity of male baby name %s in U.S., %s" % (final_name, years
    else:
        title = "Popularity of male baby names in U.S., %s" % (yearstr)

x_values = range(startyear, endyear + 1)
y_zeroes = [0] * (endyear - startyear)
```

```
if form == 'line':
    fig, ax = plt.subplots(num=None, figsize=(16, 9), dpi=300, facecolor='w', edgec
    counter = 0
    for name, sex in dataframe.columns:
        color = colors[counter % num_colors]
        counter += 1
        if has_both:
            label = "%s (%s)" % (name, sex)
        else:
            label = name
        ax.plot(x_values, dataframe[(name, sex)], label=label, color=color, linewic
    ax.set_ylim(0,determine_y_limit(max_y))
    ax.set_xlim(xmin, endyear)
    ax.set_ylabel(y_text, size = 13)
    box = ax.get_position()
    ax.set_position([box.x0, box.y0 + box.height * legend_size,
             box.width, box.height * (1 - legend_size)])
    legend_cols = min(5, num_series)
    ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05), fancybox=True, shado

if form == 'subplots_auto':
    counter = 0
    fig, axes = plt.subplots(num_series, 1, figsize=(12, 3.5*num_series))
    print('Maximum alpha: %d percent' % (determine_y_limit(max_y)))
    for name, sex in dataframe.columns:
        if sex=='M':
            sex_label = 'male'
        else:
            sex_label = 'female'
        label = "Percent of %s births for %s" % (sex_label, name)
        current_ymax = dataframe[(name, sex)].max()
        tint = 1.0 * current_ymax / determine_y_limit(max_y)
        axes[counter].plot(x_values, dataframe[(name, sex)], color='k')
        axes[counter].set_ylim(0,determine_y_limit(current_ymax))
        axes[counter].set_xlim(xmin, endyear)
        axes[counter].fill_between(x_values, dataframe[(name, sex)], color=colors[(

        axes[counter].set_ylabel(label, size=11)
        plt.subplots_adjust(hspace=0.1)
        counter += 1

if form == 'subplots_same':
    counter = 0
    fig, axes = plt.subplots(num_series, 1, figsize=(12, 3.5*num_series))
    print('Maximum y axis: %d percent' % (determine_y_limit(max_y)))
    for name, sex in dataframe.columns:
        if sex=='M':
            sex_label = 'male'
        else:
            sex_label = 'female'
        label = "Percent of %s births for %s" % (sex_label, name)
        axes[counter].plot(x_values, dataframe[(name, sex)], color='k')
        axes[counter].set_ylim(0,determine_y_limit(max_y))
        axes[counter].set_xlim(xmin, endyear)
        axes[counter].fill_between(x_values, dataframe[(name, sex)], color=colors[
        axes[counter].set_ylabel(label, size=11)
        plt.subplots_adjust(hspace=0.1)
        counter += 1

if form == 'stream':
    plt.figure(num=None, figsize=(20,10), dpi=150, facecolor='w', edgecolor='k')
    plt.title(title, size=17)
    plt.xlim(xmin, endyear)

    if has_both:
        yaxtext = 'Percent of births of indicated sex (scale: '
    elif has_male:
        yaxtext = 'Percent of male births (scale: '
    else:
        yaxtext = 'Percent of female births (scale: '

    scale = str(determine_y_limit(max_y)) + ')'
    yaxtext += scale
    plt.ylabel(yaxtext, size=13)
    polys = plt.stackplot(x_values, *[dataframe[(name, sex)] for name, sex in data
                             colors=colors, baseline=baseline)
    legendProxies = []
    for poly in polys:
        legendProxies.append(plt.Rectangle((0, 0), 1, 1, fc=poly.get_facecolor()[0
    namelist = []
    for name, sex in dataframe.columns:
        if has_both:
            namelist.append('%s (%s)' % (name, sex))
        else:
            namelist.append(name)
    plt.legend(legendProxies, namelist, loc=3, ncol=2)

    plt.tick_params(\
        axis='y',
        which='both',      #  major and minor ticks
        left='off',
        right='off',
        labelleft='off')

plt.show()
if png_name != '':
        filename = save_path + "/" + png_name + ".png"
        plt.savefig(filename)
    plt.close()

#stream graph

make_chart(df=chart_1,
           form='stream', # line , subplots_auto , subplots_same , stream
           title='',
           colors= [],
           smoothing=0,
           baseline='sym',  # zero ,  sym ,  wiggle ,  weighted_wiggle
           png_name = 'simpsons_male_names',  # if '', will not be saved
           )
```

```
C:\Users\Shreya\Anaconda3\lib\site-packages\pandas\core\indexing.py:179: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  self._setitem_with_indexer(indexer, value)
```


Popularity of male baby names in U.S., 1880-2017
Percent of male births (scale: 8.8)
Cecil
Charles
Dave
Herbert
Jake
John
Martin
Nelson
Ralph
Tony