In this project, I am using dataset of US Social Security Baby Names catalogue which reports on the number of males and female newbornes that were given a certain name each year since 1880 ¶

In [5]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as pp
import seaborn
```

In [6]:
```
%matplotlib inline
```

Now we will uncompress the zip file which has the baby names file in it.

We can do that using python zipfile module

In [7]:
```
import zipfile
```

We create zipfile object from names.zip file and we extract its contents into current directory

In [8]:
```
zipfile.ZipFile('names.zip').extractall('.')
```

Using os module, we can look at some contents of the directory

In [9]:
```
import os
os.listdir('names')[:10]
```

Out[9]:
```
['NationalReadMe.pdf',
 'yob1880.txt',
 'yob1881.txt',
 'yob1882.txt',
 'yob1883.txt',
 'yob1884.txt',
 'yob1885.txt',
 'yob1886.txt',
 'yob1887.txt',
 'yob1888.txt']
```

These files are in the names folder. And now we'll read one of the files

In [10]:
```
open('names/yob2011.txt','r').readlines()[:10]
```

Out[10]:
```
['Sophia,F,21816\n',
 'Isabella,F,19870\n',
 'Emma,F,18777\n',
 'Olivia,F,17294\n',
 'Ava,F,15480\n',
 'Emily,F,14236\n',
 'Abigail,F,13229\n',
 'Madison,F,12360\n',
 'Mia,F,11512\n',
 'Chloe,F,10970\n']
```

This is comma seperated file with name, sex and the number of babes born on 2011 with the name

In [11]:
```
names_2011 = pd.read_csv('names/yob2011.txt')
```

In [12]:
```
names_2011.head()
```

Out[12]:

	Sophia	F	21816
0	Isabella	F	19870
1	Emma	F	18777
2	Olivia	F	17294
3	Ava	F	15480
4	Emily	F	14236

You can see that the first row is not column names and it is actually a record. We can avoid this by explicitly giving column names argument when we read the data

In [13]:
```
names_2011 = pd.read_csv('names/yob2011.txt', names = ['name','sex','number'])
```

In [14]:
```
names_2011.head()
```

Out[14]:

	name	sex	number
0	Sophia	F	21816
1	Isabella	F	19670
2	Emma	F	18777
3	Olivia	F	17294
4	Ava	F	15480

Now we will load all the files between year 1880 and 2014. While doing so, we'll add a column year to each table so that the records with the same name but diffrent year will not be confused

We'll collect all these tables in names_all list and then loop over the years from 1880 to 2014,append to the list the result of reading the csv file

I'll also add the column for the year. I do that by operating on the last element of the list after each iteration

Finally I feed this pandas list of dataframes to the function pandas concat which concatenates all

In [15]:
```
names_all = []

for year in range(1880,2014+1):
    names_all.append(pd.read_csv('names/yob{}.txt'.format(year), names = ['name','sex','number']))
    names_all[-1]['year'] = year # new column

all_years = pd.concat(names_all)
```

In [16]:
```
all_years.head()
```

Out[16]:

	name	sex	number	year
0	Mary	F	7065	1880
1	Anna	F	2604	1880
2	Emma	F	2003	1880
3	Elizabeth	F	1939	1880
4	Minnie	F	1746	1880

In [17]:
```
all_years.tail()
```

Out[17]:

	name	sex	number	year
33039	Zykeem	M	5	2014
33040	Zymeer	M	5	2014
33041	Zymiere	M	5	2014
33042	Zyran	M	5	2014
33043	Zyrin	M	5	2014

Awesome!! Loaded all the names and years form the dataset.

Now we will try to analyze the popularity of the baby names across all the years

We will have to loop over this dataset according to the changing popularity of the name.

Lets try that using multiindexing. We'll index first over gender, then name and then year

In [18]:
```
all_years_indexes = all_years.set_index(['sex','name','year']).sort_index()
```

```
In [19]: all_years_indexes.head(20)
```

Out[19]:

			number
sex	name	year	
F	Aabha	2011	7
		2012	5
		2014	9
	Aabriella	2008	5
		2014	5
	Aaden	2009	5
	Aadhira	2012	6
		2013	10
		2014	13
	Aadhya	2007	10
		2008	9
		2009	18
		2010	19
		2011	52
		2012	110
		2013	172
		2014	249
	Aadi	2006	5
		2012	5
		2013	6

We use attribute of indexing loc from iloc to select rows respectively by the value of the index or by the number of the row.

loc also lets us select any combinations of the fields for the multiindex.

E.g. If we want table of Mary across all years

```
In [20]: all_years_indexes.loc['F','Mary']
```

Out[20]:

	number
year	
1880	7065
1881	6919
1882	8148
1883	8012
1884	9217
1885	9128
1886	9890
1887	9888
1888	11754
1889	11648
1890	12078
1891	11703
1892	13173
1893	12784
1894	13151
1895	13446
1896	13811
1897	13413
1898	14406
1899	13172
1900	16707
1901	13136
1902	14486
1903	14275
1904	14962
1905	16067
1906	16370
1907	17580

2005	4447
2006	4081
2007	3673
2008	3490
2009	3154
2010	2862
2011	2701
2012	2565
2013	2632
2014	2611

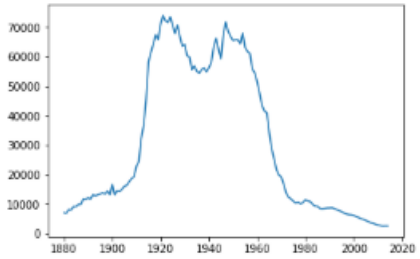
135 rows × 1 columns

This is what we need to plot popularity of a name

In [22]:
```
def plot_names(sex, name):
    data = all_years_indexes.loc[sex, name]

    pp.plot(data.index,data.values)
```

In [23]:
```
plot_names('F','Mary')
```



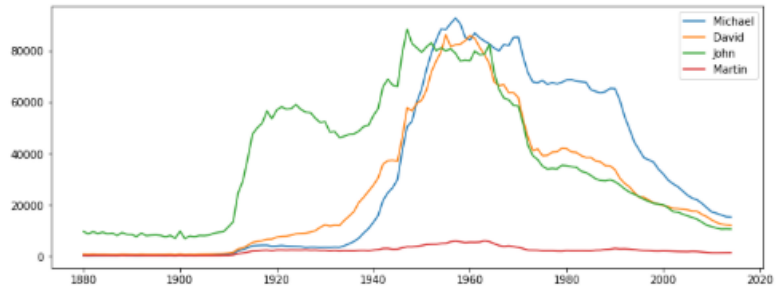
Looks like the popularity of the Mary was highest in 1920 and 1950

In [25]:
```
pp.figure(figsize=(12,4.5))
names = ['Michael', 'David','John','Martin']

for name in names:
    plot_names('M',name)

pp.legend(names)
```

Out[25]:
```
<matplotlib.legend.Legend at 0x1bec1411e10>
```

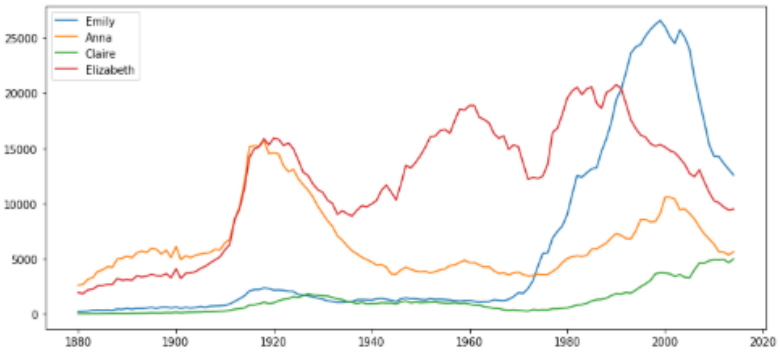


In [26]:
```
pp.figure(figsize=(12,5.5))
names = ['Emily','Anna','Claire','Elizabeth']

for name in names:
    plot_names('F',name)

pp.legend(names)
```

Out[26]: <matplotlib.legend.Legend at 0x1bec1589828>

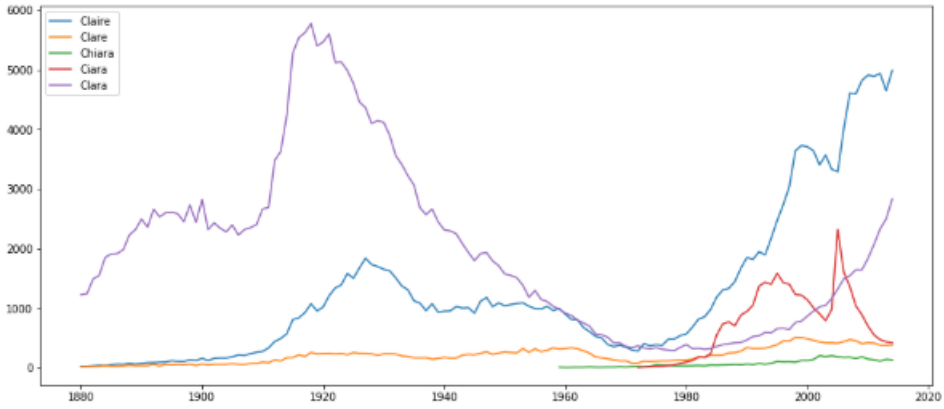


How about the variance in different names that sound similar ?

In [28]:
```
pp.figure(figsize=(15,6.5))
names = ['Claire','Clare','Chiara','Ciara', 'Clara']
for name in names:
    plot_names('F',name)

pp.legend(names)
```

Out[28]: <matplotlib.legend.Legend at 0x1bec14b1a58>



lets look at cumulative plots now using stacked line charts- I am thinking to plot the frequency of each variance on top or other

So total height will tell us total variance. For this in matplotlib, we can use stackplot() command

For this we'll have to change shape of our table a little bit

First we select only variance related names into table

In [30]:
```
all_years_indexes.loc['F'].loc[names].head()
```

Out[30]:

		number
name	year	
Chiara	1959	9
	1960	5
	1962	8
	1963	8
	1964	12

Now I will unstack the index names into coulmn values(level 0 is name, level 1 is year)

In [32]:
```
all_years_indexes.loc['F'].loc[names].unstack(level=0).head()
```

Out[32]:

	number				
name	Chiara	Ciara	Claire	Clara	Clare
year					
1880	NaN	NaN	21.0	1226.0	15.0
1881	NaN	NaN	23.0	1242.0	20.0
1882	NaN	NaN	30.0	1490.0	21.0
1883	NaN	NaN	38.0	1548.0	22.0
1884	NaN	NaN	33.0	1852.0	38.0

To get good charts, we'll replace NaN with 0 by using pandas function fillna()

In [34]:
```
all_years_indexes.loc['F'].loc[names].unstack(level =0).fillna(0).head()
```

Out[34]:

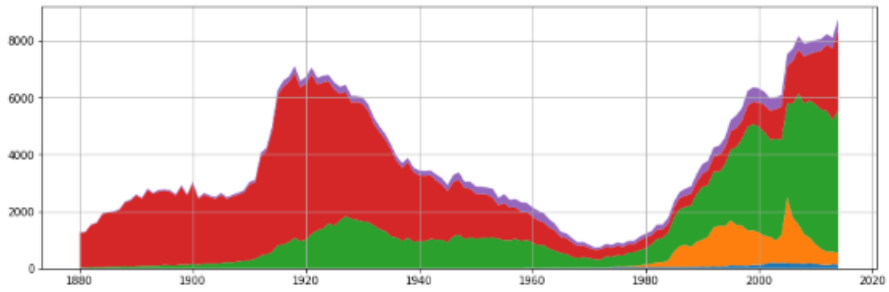
	number				
name	Chiara	Ciara	Claire	Clara	Clare
year					
1880	0.0	0.0	21.0	1226.0	15.0
1881	0.0	0.0	23.0	1242.0	20.0
1882	0.0	0.0	30.0	1490.0	21.0
1883	0.0	0.0	38.0	1548.0	22.0
1884	0.0	0.0	33.0	1852.0	38.0

In [35]:
```
variants = all_years_indexes.loc['F'].loc[names].unstack(level =0).fillna(0)
```

In [36]:
```
pp.figure(figsize=(14,4.5))

pp.stackplot(variants.index,variants.values.T,labels =names)

pp.grid(True, which='both')
```



Now stackplot doesn't support legend so we'll have to explicitly use matplotlib.text() to get legends

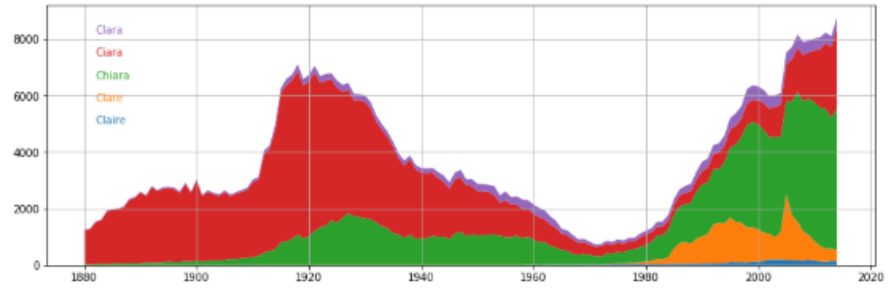
Also, I can use palette to color each name

In [38]:
```
pp.figure(figsize=(14,4.5))

palette = seaborn.color_palette()
pp.stackplot(variants.index,variants.values.T,labels =names,colors = palette)

pp.grid(True, which='both')# gives x and y grid lines

# I am going to loop over the names and over the index that I get by enumerating the names
for i,name in enumerate(names):
    pp.text(1882,5000 + 800*i,name, color = palette[i])
```



Lets see which are the 10 most popular names in a given year

In [40]:
```
all_years_indexes.loc['M',:,2008].sort_values('number',ascending = False).head()
```

Out[40]:

			number
sex	name	year	
M	Jacob	2008	22568
	Michael	2008	20590
	Ethan	2008	20196
	Joshua	2008	19186
	Daniel	2008	18985

In [41]:
```
pop2008 = all_years_indexes.loc['M',:,2008].sort_values('number',ascending = False).head()
```

In [42]:
```
pop2008.reset_index().drop(['sex','year','number'],axis=1).head()
```

Out[42]:

	name
0	Jacob
1	Michael
2	Ethan
3	Joshua
4	Daniel

We'll built a function that will run the above code for any year

What we'll do is take sex and year, compute the dataframe simple-select sex and year-sort it-resets the index-then drop the columns- change the name of column to the year

In [44]:
```
def pop_names(sex, year):
    simple = all_years_indexes.loc[sex,:,year].sort_values('number',ascending=False).reset_index()
    simple = simple.drop(['sex','year','number'], axis = 1).head(10)
    simple.columns = [year]
    simple.index = simple.index +1

    return simple
```

In [45]:
```
pop_names('M',2009)
```

Out[45]:

	2009
1	Jacob
2	Ethan
3	Michael
4	Alexander
5	William
6	Joshua
7	Daniel
8	Jayden
9	Noah
10	Christopher

Now lets try to find the 10 most popular names for range of years

To put all in one dataframe, we'll use pandas function join

Since join is method of dataframe, we need to call on first datframe in the list

In [47]:
```
def top_10_each_year(sex,year1,year2):
    top_tens = [pop_names(sex,year) for year in range(year1,year2+1)]

    return top_tens[0].join(top_tens[1:])
```

Lets check for Males

In [49]:
```
top_10_each_year('M',2000,2004) # Jacob and Micahel are dominant
```

Out[49]:

	2000	2001	2002	2003	2004
1	Jacob	Jacob	Jacob	Jacob	Jacob
2	Michael	Michael	Michael	Michael	Michael
3	Matthew	Matthew	Joshua	Joshua	Joshua
4	Joshua	Joshua	Matthew	Matthew	Matthew
5	Christopher	Christopher	Ethan	Andrew	Ethan
6	Nicholas	Nicholas	Andrew	Ethan	Andrew
7	Andrew	Andrew	Joseph	Joseph	Daniel
8	Joseph	Joseph	Christopher	Daniel	William
9	Daniel	Daniel	Nicholas	Christopher	Joseph
10	Tyler	William	Daniel	Anthony	Christopher

Lets look for females ¶

In [51]:
```
top_10_each_year('F',1985,1995) # Jessica and Ashley are Dominant
```

Out[51]:

	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995
1	Jessica	Jessica	Jessica	Jessica	Jessica	Jessica	Ashley	Ashley	Jessica	Jessica	Jessica
2	Ashley	Ashley	Ashley	Ashley	Ashley	Ashley	Jessica	Jessica	Ashley	Ashley	Ashley
3	Jennifer	Amanda	Amanda	Amanda	Brittany	Brittany	Brittany	Amanda	Sarah	Emily	Emily
4	Amanda	Jennifer	Jennifer	Sarah	Amanda	Amanda	Amanda	Brittany	Samantha	Samantha	Samantha
5	Sarah	Sarah	Sarah	Jennifer	Sarah	Samantha	Samantha	Sarah	Emily	Sarah	Sarah
6	Stephanie	Stephanie	Stephanie	Brittany	Samantha	Sarah	Sarah	Samantha	Brittany	Taylor	Taylor
7	Nicole	Nicole	Brittany	Stephanie	Jennifer	Stephanie	Stephanie	Emily	Taylor	Brittany	Hannah
8	Heather	Brittany	Nicole	Samantha	Stephanie	Jennifer	Jennifer	Stephanie	Amanda	Amanda	Brittany
9	Elizabeth	Heather	Heather	Nicole	Lauren	Elizabeth	Elizabeth	Elizabeth	Elizabeth	Elizabeth	Amanda
10	Megan	Elizabeth	Elizabeth	Elizabeth	Elizabeth	Lauren	Emily	Megan	Stephanie	Megan	Elizabeth

Lets take 6 most popular names in this case and plot their popularity across all years

Now we need count of each name so we'll convert the dataframe to series using stack() so that we can get counts of each name

In [53]:
```
top_10_each_year('F',1985,1995).stack().head()
```

Out[53]:
```
1  1985    Jessica
   1986    Jessica
   1987    Jessica
   1988    Jessica
   1989    Jessica
dtype: object
```

In [54]:
```
top_10_each_year('F',1985,1995).stack().value_counts()
```

Out[54]:
```
Ashley       11
Amanda       11
Elizabeth    11
Sarah        11
Jessica      11
Brittany     10
Stephanie     9
Samantha      8
Jennifer      7
Emily         5
Nicole        4
Taylor        3
Megan         3
Heather       3
Lauren        2
Hannah        1
dtype: int64
```

Using index, we'll plot top 6 names

In [56]:
```
popular = top_10_each_year('F',1985,1995).stack().value_counts().index[:6]
```

In [57]:
```
popular
```

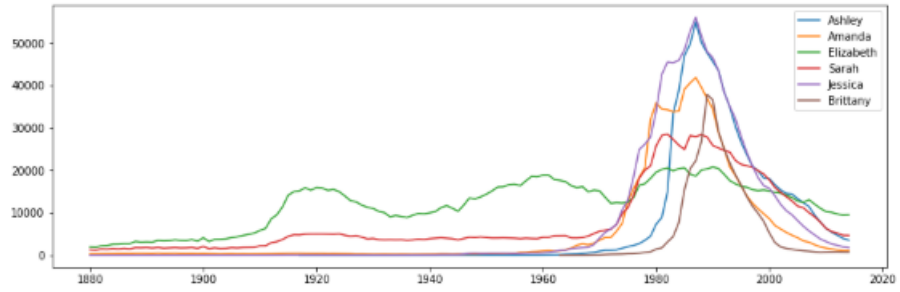
Out[57]:
```
Index(['Ashley', 'Amanda', 'Elizabeth', 'Sarah', 'Jessica', 'Brittany'], dtype='object')
```

In [58]:
```
pp.figure(figsize=(14,4.5))

for pop in popular:
    plot_names('F',pop)

pp.legend(popular)
```

Out[58]:
```
<matplotlib.legend.Legend at 0x1bec254a2e8>
```



Now we will find names that become popular suddenly and fade away suddenly i.e. the spikeyness of names

Lets start by finding total number of babies for a given name across the years.. we'll use group by function

In [61]:
```
all_years.groupby(['sex','name']).sum().head()
```

Out[61]:

In [61]:
```
all_years.groupby(['sex','name']).sum().head()
```

Out[61]:

		number	year
sex	name		
F	Aabha	21	6037
	Aabriella	10	4022
	Aaden	5	2009
	Aadhira	29	6039
	Aadhya	639	16084

This is even adding years which doesn't make any sense sowe'll drop the year column

In [63]:
```
all_years.groupby(['sex','name'])['number'].sum().head()
```

Out[63]:
```
sex  name
F    Aabha         21
     Aabriella     10
     Aaden          5
     Aadhira       29
     Aadhya       639
Name: number, dtype: int64
```

Now we get the series. We'll assign it to a variable

In [65]:
```
totals = all_years.groupby(['sex','name'])['number'].sum()
```

In [66]:
```
totals.head()
```

Out[66]:
```
sex  name
F    Aabha         21
     Aabriella     10
     Aaden          5
     Aadhira       29
     Aadhya       639
Name: number, dtype: int64
```

For this spikeyness, we need a function..sum of squares and not just sum()

so that the frequencies of each name will be insensitive to the total appeareance of other numbers

In [68]:
```
def sumsq(x):
    return sum(x**2)
```

We divide by total as doing so will put very popular and least popular names in clear way

In [69]:
```
Spikyness= all_years.groupby(['sex','name'])['number'].agg(sumsq)/totals**2
```

In [70]:
```
Spikyness.head()
```

Out[70]:
```
sex  name
F    Aabha        0.351474
     Aabriella    0.500000
     Aaden        1.000000
     Aadhira      0.362663
     Aadhya       0.262673
Name: number, dtype: float64
```

Spykiness is a number between o and 1 which happens only when a number appears only in a single year

I will only the names that will appear relatively frequently..I;ll consider totals>5000

In [73]:
```
spiky_common = Spikyness[totals>5000].copy()# I am copying this so that I can sort it
```

In [74]:
```
# arranging in descending order
spiky_common.sort_values(ascending = False).head()
```

Out[74]:
```
sex  name
M    Iker         0.199368
     Shaquille    0.195689
     Jase         0.182165
F    Adalynn      0.177107
     Harper       0.164827
Name: number, dtype: float64
```

Lets look at least spikey names

In [76]:
```
spiky_common.sort_values(ascending = False).tail()
```

Out[76]:
```
sex  name
F    Rosa      0.008512
     Mollie    0.008437
M    Ike       0.008310
F    Delia     0.008263
     Louisa    0.007931
Name: number, dtype: float64
```

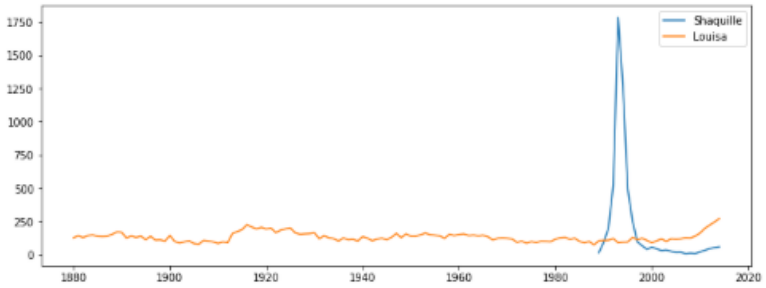
Shaquille has more spikness than Louisa

In [78]:
```
pp.figure(figsize=(12,4.5))

plot_names('M','Shaquille')
plot_names('F','Louisa')

pp.legend(['Shaquille','Louisa'])
```

Out[78]: <matplotlib.legend.Legend at 0x1bec0e88cf8>



In [79]:
```
pp.figure(figsize=(20,6.5))
```

Out[79]: <Figure size 1440x468 with 0 Axes>

<Figure size 1440x468 with 0 Axes>

Lets plot top ten most spiky names

In [81]:
```
most_spiky = spiky_common.sort_values(ascending = False).head(10).index.values
```

In [82]:
```
pp.figure(figsize=(15,8))

for i,spike in most_spiky:
    plot_names(i,spike)

pp.legend([name for sex,name in most_spiky], loc = 'upper_left')
pp.grid()
```

```
C:\Users\admin\Anaconda3\lib\site-packages\matplotlib\legend.py:497: UserWarning: Unrecognized location "upper_left". Falling b
ack on "best"; valid locations are
        best
        upper right
        upper left
        lower left
        lower right
        right
        center left
        center right
        lower center
        upper center
        center

  % (loc, '\n\t'.join(self.codes)))
```