CAP5768_Assignment2_cgarbin

September 24, 2019

1 CAP 5768 - Data Science - Dr. Marques - Fall 2019

Christian Garbin

1.1 Assignment 2: Exploratory data analysis

1.1.1 Goals

- To increase familiarity with the Python "data science stack" (NumPy, Pandas, Matplotlib).
- To explore (manipulate, summarize, and visualize) datasets.
- To improve the ability to write Python code to answer questions and test hypotheses based on the contents of those datasets.

1.1.2 Instructions

- This assignment is structured in three parts, using a different dataset for each part.
- For each part, there will be some Python code to be written and questions to be answered.
- At the end, you should export your notebook to PDF format; it will "automagically" become your report.
- Submit the report (PDF), notebook (.ipynb file), and (optionally) link to the "live" version of your solution on Google Colaboratory via Canvas.
- The total number of points is 154 (plus up to 85 bonus points), distributed as follows: Part 1 (58+ pts), Part 2 (28+ pts), Part 3 (43+ pts), and Conclusions (25 pts).

1.1.3 Important

- It is OK to attempt the bonus points, but please **do not overdo it!**
- Remember: this is an exercise in performing exploratory data analysis; expanding (and practicing) your knowledge of Python, Jupyter notebooks, Numpy, Pandas, and Matplotlib; and writing code to test hypotheses and answer questions based on the available data (and associated summary statistics).
- This is not (yet) the time to do sophisticated statistical analysis, train ML models, etc.
- You must **organize your data files in the proper folders** for the code to work.

1.2 Part 1: The MovieLens 1M dataset

This is a dataset of movie ratings data collected from users of MovieLens in the late 1990s and early 2000s. The data provide movie ratings, movie metadata, and demographic data about the users. Such data is often of interest in the development of recommendation systems based on machine learning algorithms.

The MovieLens 1M dataset contains ~1 million ratings collected from ~6,000 users on ~4,000 movies. It's spread across three tables: *ratings*, *user information*, and *movie information*. After extracting the data from the ZIP file (available on Canvas), we can load each table into a pandas DataFrame object using the Python code below.

See: https://grouplens.org/datasets/movielens/ for additional information.

[1]:
```
# Imports
import numpy as np
import pandas as pd
from pandas import DataFrame, Series

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

from scipy.stats import pearsonr
```

Before running the cell below, make sure that you have downloaded the movielens.zip file from Canvas, unzipped it, and placed its contents under the 'data' folder.

[2]:
```
import pandas as pd

# Make display smaller
pd.options.display.max_rows = 10

unames = ['user_id', 'gender', 'age', 'occupation', 'zip']
users = pd.read_table('data/movielens/users.dat', sep='::',
                      header=None, names=unames, engine='python')

rnames = ['user_id', 'movie_id', 'rating', 'timestamp']
ratings = pd.read_table('data/movielens/ratings.dat', sep='::',
                        header=None, names=rnames, engine='python')

mnames = ['movie_id', 'title', 'genres']
movies = pd.read_table('data/movielens/movies.dat', sep='::',
                       header=None, names=mnames, engine='python')
```

2 Your turn! (24 points, i.e., 6 pts each)

Write Python code to answer the following questions (make sure the messages displayed by your code are complete and descriptive enough): 1. How many users are stored in the *users* table and what information is stored for each user? 2. How many movies are stored in the *movies* table and what information is stored for each movie? 3. How many ratings are stored in the *ratings* table and what information is stored for each rating? 4. How are users, the movies each user has rated, and the rating related?

Note: ages and occupations are coded as integers indicating *groups* described in the dataset's README file.

[3]:
```
# Uncomment to see details of the movies datasets
# !cat 'data/movielens/README'
```

2.1 Solution

2.1.1 How many users are stored in the users table and what information is stored for each user?

[4]:
```
len(users)
```

[4]:
```
6040
```

[5]:
```
users.dtypes
```

[5]:
```
user_id        int64
gender        object
age            int64
occupation     int64
zip           object
dtype: object
```

[6]:
```
users.head()
```

[6]:

	user_id	gender	age	occupation	zip
0	1	F	1	10	48067
1	2	M	56	16	70072
2	3	M	25	15	55117
3	4	M	45	7	02460
4	5	M	25	20	55455

[7]:
```
users.gender.unique()
```

[7]:
```
array(['F', 'M'], dtype=object)
```

[8]:
```
users.occupation.unique()
```

[8]:
```
array([10, 16, 15,  7, 20,  9,  1, 12, 17,  0,  3, 14,  4, 11,  8, 19,  2,
       18,  5, 13,  6])
```

There are 6,040 users. For each one of them the dataset has:

- `user_id`: a unique id, stored as an integer.
- `gender`: a character that identifies the user's gender - possible values are `F` and `M`.
- `age`: user's age range, coded as explained in the *README* file, stored as an integer.
- `occupation`: user's occupation, coded as an integer.
- `zip`: user's ZIP code, stored as a string.

2.1.2 How many movies are stored in the movies table and what information is stored for each movie?

[9]:
```
len(movies)
```

[9]:
```
3883
```

[10]:
```
movies.dtypes
```

[10]:
```
movie_id     int64
title       object
genres      object
dtype: object
```

[11]:
```
movies.head()
```

[11]:

	movie_id	title	genres
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	Comedy

There are 3,883 movies. For each movie the dataset has:

- `movie_id`: a unique id, stored as an integer.
- `title`: a string with the movie title and year.
- `genres`: all genres for the movie, separated by `|`.

2.1.3 How many ratings are stored in the ratings table and what information is stored for each rating?

[12]:
```
len(ratings)
```

[12]:
```
1000209
```

[13]:
```
ratings.dtypes
```

[13]:
```
user_id      int64
movie_id     int64
rating       int64
timestamp    int64
dtype: object
```

[14]:
```
ratings.head()
```

[14]:
```
   user_id  movie_id  rating  timestamp
0        1      1193       5  978300760
1        1       661       3  978302109
2        1       914       3  978301968
3        1      3408       4  978300275
4        1      2355       5  978824291
```

[15]:
```
ratings.rating.unique()
```

[15]:
```
array([5, 3, 4, 2, 1])
```

There are 1,000,209 ratings. For each rating the dataset has:

- `user_id`: the id of the user who rated the movie.
- `movied_id`: the id of the rated movie.
- `rating`: the user rating, in a range from 1 to 5, as an integer.
- `timestamp`: seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970 (source).

2.1.4 How are users, the movies each user has rated, and the rating related?

They are related by their id fields, `user_id` and `movie_id`.

3 Your turn! (24 points, i.e., 6 pts each)

Write Python code to answer the following questions (make sure the messages displayed by your code are complete and descriptive enough): 5. What is the occupation that maps to most of the users? Plot a bar chart of the occupation counts and report the size of the largest bin. 6. What percentage of users are 50 years old or older? Plot a pie chart showing all percentages (per age group) and report the requested value. 7. Which movie received the highest number of ratings (and how were such ratings distributed)? 8. What is the average rating for all movies/users?

3.1 Solution

(ENTER YOUR ANSWERS HERE)

3.1.1 5. What is the occupation that maps to most of the users? Plot a bar chart of the occupation counts and report the size of the largest bin.

Most of the time we are interested in asnwering questions, e.g. "what is the most frequent occupation". Therefore we will graph them in sorted order.

We also translate the coded occupation into the occupation name, as defined in the README file for the dataset.

[16]:
```
# Table comes from information in the README file
occupation_names = { 'occupation' : {
    0: 'other', 1: 'academic/educator', 2: 'artist',
    3: 'clerical/admin', 4: 'college/grad student',
    5: 'customer service', 6: 'doctor/health care',
    7: 'executive/managerial', 8:  'farmer', 9: 'homemaker',
    10: 'K-12 student', 11: 'lawyer', 12: 'programmer',
    13: 'retired', 14: 'sales/marketing', 15: 'scientist',
    16: 'self-employed', 17: 'technician/engineer',
    18: 'tradesman/craftsman', 19: 'unemployed', 20: 'writer'}}

users.replace(occupation_names, inplace=True)
```

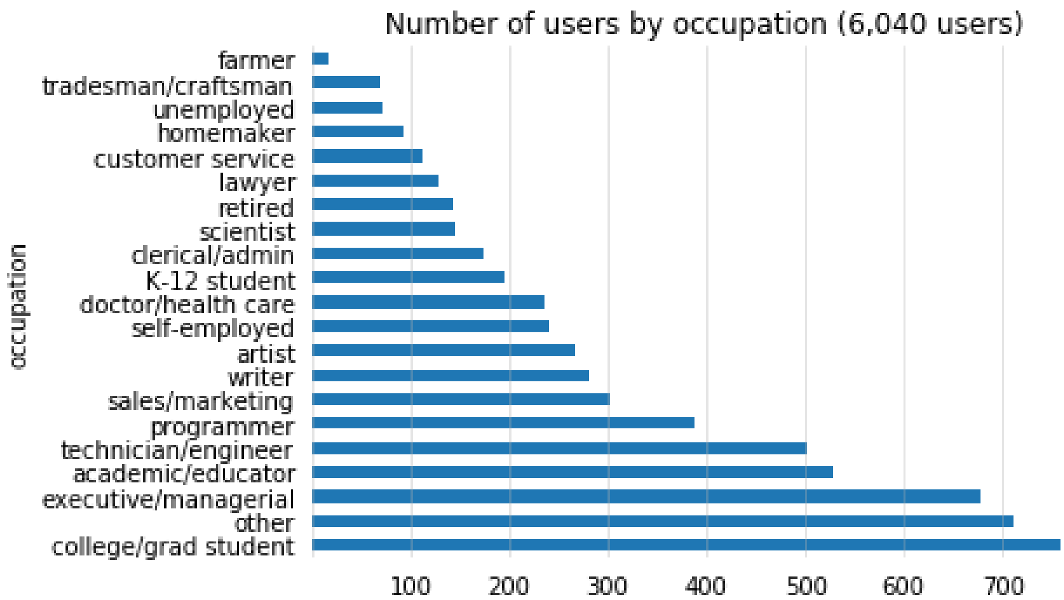
[17]:
```
occupation_by_users = users.groupby('occupation')['user_id'] \
    .count().sort_values(ascending=False)
```

[18]:
```
print('Occupation with most users: {}, with {} users'.format(
    occupation_by_users.index[0], occupation_by_users.iloc[0]))
```

```
Occupation with most users: college/grad student, with 759 users
```

[19]:
```
def format_graph(ax):
    # Remove box around the graph
    for s in ('right', 'left', 'top', 'bottom'):
        ax.spines[s].set_visible(False)

    # Remove all tick marks
    plt.tick_params(bottom=False, top=False, left=False, right=False)


def formatted_barh_graph(df, title):
    ax = df.plot.barh(title=title)
    format_graph(ax)

    # Show a vertical grid to help size the bars
    ax.grid(axis='x', alpha=0.4)

    # And now, nitpicking (zero can be inferred)
    ax.xaxis.get_major_ticks()[0].label1.set_visible(False)
```

```
title = 'Number of users by occupation ({:,} users)'.format(len(users))
formatted_barh_graph(occupation_by_users, title)
```



3.1.2 6. What percentage of users are 50 years old or older? Plot a pie chart showing all percentages (per age group) and report the requested value.

According to the README file:

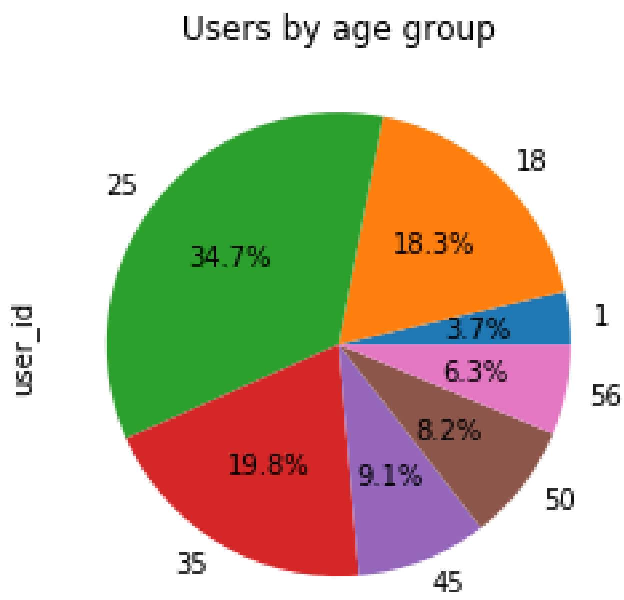
> Age is chosen from the following ranges:
>
> - 1: "Under 18"
> - 18: "18-24"
> - 25: "25-34"
> - 35: "35-44"
> - 45: "45-49"
> - 50: "50-55"
> - 56: "56+"

Thus "50 years old or older" encompasses two groups, "50" and "56".

[20]:
```
fifty_or_older = users.eval('(age == 50) | (age == 56)')
c = len(users[fifty_or_older])
print('There are {} ({:.2f}%) users who are 50 years old or older'
      .format(c, c/len(users)*100))
```

```
There are 876 (14.50%) users who are 50 years old or older
```

[21]:
```
users.groupby('age').count()['user_id'].plot(
    kind='pie', autopct='%1.1f%%', title='Users by age group');
```



3.1.3 7. Which movie received the highest number of ratings (and how were such ratings distributed)?

[22]:
```
highest_number_ratings = ratings.groupby('movie_id')['rating'] \
    .count().sort_values(ascending=False).index[0]
movies[movies['movie_id'] == highest_number_ratings]
```

[22]:

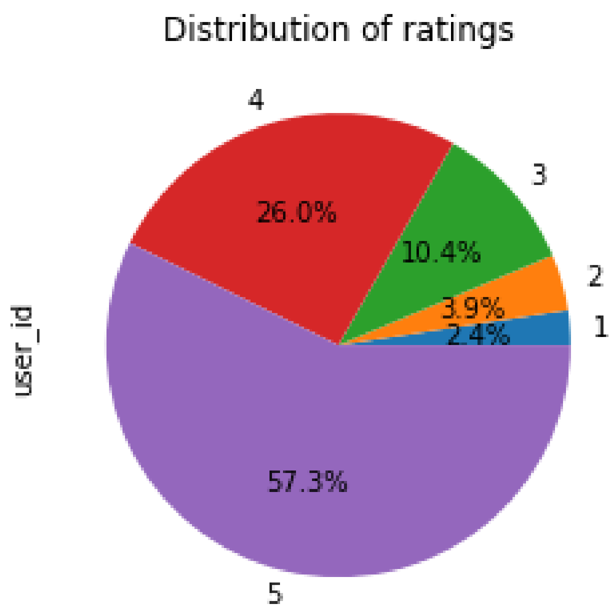
	movie_id	title	genres
2789	2858	American Beauty (1999)	Comedy|Drama

[23]:
```
# calculate absolute number of ratings for that movie, by rating
num_ratings = pd.DataFrame(
    ratings[ratings['movie_id'] == highest_number_ratings] \
    .groupby('rating')['user_id'].count())

# calculate percentage (distribution) for each rating
num_ratings['percentage'] = num_ratings['user_id'] / num_ratings['user_id'].
 ↪sum() * 100

# Print and graph
display(num_ratings)
```

```
num_ratings['user_id'].plot(kind='pie', autopct='%1.1f%%',
                            title='Distribution of ratings')
```

```
        user_id  percentage
rating
1            83    2.421237
2           134    3.908985
3           358   10.443407
4           890   25.962660
5          1963   57.263711
```

[23]: <matplotlib.axes._subplots.AxesSubplot at 0x120eff2b0>



3.1.4 8. What is the average rating for all movies/users?

[24]:
```
print('The average rating for all movies/users: {:.1f}'
     .format(ratings['rating'].sum() / len(ratings)))
```

```
The average rating for all movies/users: 3.6
```

We will use the Python code below to merge all three tables into a unified data frame.

[25]:
```
data = pd.merge(pd.merge(ratings, users), movies)
data.head()
```

[25]:
```
   user_id  movie_id  rating  timestamp gender  age            occupation  \
0        1      1193       5  978300760      F    1          K-12 student
1        2      1193       5  978298413      M   56         self-employed
2       12      1193       4  978220179      M   25            programmer
3       15      1193       4  978199279      M   25  executive/managerial
4       17      1193       5  978158471      M   50      academic/educator

     zip                                 title genres
0  48067  One Flew Over the Cuckoo's Nest (1975)  Drama
1  70072  One Flew Over the Cuckoo's Nest (1975)  Drama
2  32793  One Flew Over the Cuckoo's Nest (1975)  Drama
3  22903  One Flew Over the Cuckoo's Nest (1975)  Drama
4  95350  One Flew Over the Cuckoo's Nest (1975)  Drama
```

The Python code below will show the top 10 films among female viewers (and, for comparison's sake, the ratings for those movies by male viewers) in decreasing order (highest rated movie on top).

[26]:
```
# Build pivot table
mean_ratings = data.pivot_table('rating', index='title',
                                columns='gender', aggfunc='mean')
display(mean_ratings[:3])
```

```
gender                             F         M
title
$1,000,000 Duck (1971)      3.375000  2.761905
'Night Mother (1986)        3.388889  3.352941
'Til There Was You (1997)   2.675676  2.733333
```

[27]:
```
# Group ratings by title
ratings_by_title = data.groupby('title').size()
#display(ratings_by_title.index)
display(ratings_by_title[:3])
```

```
title
$1,000,000 Duck (1971)       37
'Night Mother (1986)         70
'Til There Was You (1997)    52
dtype: int64
```

[28]:
```
# Select only movies with 250 ratings or more
active_titles = ratings_by_title.index[ratings_by_title >= 250]
display(active_titles[:3])
```

```
Index([''burbs, The (1989)', '10 Things I Hate About You (1999)',
       '101 Dalmatians (1961)'],
      dtype='object', name='title')
```

[29]:
```python
# Select rows on the index
mean_ratings = mean_ratings.loc[active_titles]
display(mean_ratings[:3])
```

```
gender                                    F         M
title
'burbs, The (1989)                 2.793478  2.962085
10 Things I Hate About You (1999)  3.646552  3.311966
101 Dalmatians (1961)              3.791444  3.500000
```

[30]:
```python
# Fix naming inconsistency
mean_ratings = mean_ratings.rename(index={'Seven Samurai (The Magnificent␣
 ↪Seven) (Shichinin no samurai) (1954)':
                           'Seven Samurai (Shichinin no samurai) (1954)'})
```

[31]:
```python
top_female_ratings = mean_ratings.sort_values(by='F', ascending=False)
top_female_ratings.head(10)
```

[31]:
```
gender                                                     F         M
title
Close Shave, A (1995)                               4.644444  4.473795
Wrong Trousers, The (1993)                          4.588235  4.478261
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)       4.572650  4.464589
Wallace & Gromit: The Best of Aardman Animation… 4.563107  4.385075
Schindler's List (1993)                             4.562602  4.491415
Shawshank Redemption, The (1994)                    4.539075  4.560625
Grand Day Out, A (1992)                             4.537879  4.293255
To Kill a Mockingbird (1962)                        4.536667  4.372611
Creature Comforts (1990)                            4.513889  4.272277
Usual Suspects, The (1995)                          4.513317  4.518248
```

4 Your turn! (10 points, i.e., 5 pts each)

Modify the Python code to: 9. Display the top 10 favorite movies among male viewers, selecting only movies with 250 ratings or more. 10. Display the top 10 favorite movies among young viewers (17 years old or younger), selecting only movies with 300 ratings or more.

4.1 Solution

4.1.1 9. Display the top 10 favorite movies among male viewers, selecting only movies with 250 ratings or more.

[32]:
```
mean_ratings.sort_values(by='M', ascending=False).head(10)
```

[32]:
```
gender                                           F         M
title
Godfather, The (1972)                     4.314700  4.583333
Seven Samurai (Shichinin no samurai) (1954)  4.481132  4.576628
Shawshank Redemption, The (1994)          4.539075  4.560625
Raiders of the Lost Ark (1981)            4.332168  4.520597
Usual Suspects, The (1995)                4.513317  4.518248
Star Wars: Episode IV - A New Hope (1977)  4.302937  4.495307
Schindler's List (1993)                   4.562602  4.491415
Wrong Trousers, The (1993)                4.588235  4.478261
Close Shave, A (1995)                     4.644444  4.473795
Rear Window (1954)                        4.484536  4.472991
```

4.1.2 10. Display the top 10 favorite movies among young viewers (17 years old or younger), selecting only movies with 300 ratings or more.

According to the README file:

> Age is chosen from the following ranges:
>
> ```
> *  1:  "Under 18"
> ```

Note that the “top 10 favorite” movies may end up being more than ten movies, once we account for rating ties. That is what happened in this case. There are 17 movies in the “top 10 favorite” list because of ties in ratings.

[33]:
```
age_mean_ratings = data.pivot_table('rating', index='title',
                                    columns='age', aggfunc='mean')

# Select movies with 300 ratings or more
ratings_by_title = data.groupby('title').size()
active_titles = ratings_by_title.index[ratings_by_title >= 300]
age_mean_ratings = age_mean_ratings.loc[active_titles]

# Select ratings for young viewers
young_mean_ratings = age_mean_ratings[1]

# Account for possible ties: get the top ten rating values
top_ten_ratings = young_mean_ratings.sort_values(
    ascending=False).unique()[:10]
```

```
# Show all movies that fall into the "top 10 ratings" range
with pd.option_context('display.max_rows', None):
    print(young_mean_ratings[young_mean_ratings >= top_ten_ratings[-1]] \
          .sort_values(ascending=False))
```

```
title
Metropolis (1926)                                  4.888889
GoodFellas (1990)                                  4.840000
Third Man, The (1949)                              4.818182
Double Indemnity (1944)                            4.777778
Fried Green Tomatoes (1991)                        4.750000
Piano, The (1993)                                  4.750000
Raging Bull (1980)                                 4.714286
Roman Holiday (1953)                               4.687500
Citizen Kane (1941)                                4.680000
Charade (1963)                                     4.666667
Manchurian Candidate, The (1962)                   4.666667
From Here to Eternity (1953)                       4.666667
Notorious (1946)                                   4.666667
Real Genius (1985)                                 4.666667
Apostle, The (1997)                                4.666667
Princess Mononoke, The (Mononoke Hime) (1997)      4.636364
Bridge on the River Kwai, The (1957)               4.636364
Name: 1, dtype: float64
```

Precocious these youngsters seem to be… Or perhaps the lesson here is "don't trust in self-identified data" (who knows what the actual age is of those users).

5 BONUS! (up to 20 points)

Write Python code to display the most divisive movies (selecting only movies with 250 ratings or more), i.e.: - The top 10 movies with the greatest rating difference so that we can see which ones were preferred by women. - The top 10 movies with the greatest rating difference in the opposite direction (sign) so that we can see which ones were preferred by men.

Hint/Convention: `mean_ratings['diff'] = mean_ratings['M'] - mean_ratings['F']`

5.1 Solution

5.1.1 The top 10 movies with the greatest rating difference so that we can see which ones were preferred by women.

[34]:
```
# mean_ratings was created above, with movies that have 250 ratings or more
mean_ratings['Preferred by F'] = mean_ratings['F'] - mean_ratings['M']
pref_by_women = mean_ratings.sort_values(by='Preferred by F', ascending=False)[:
 ↪10]
```

```
pref_by_women
```

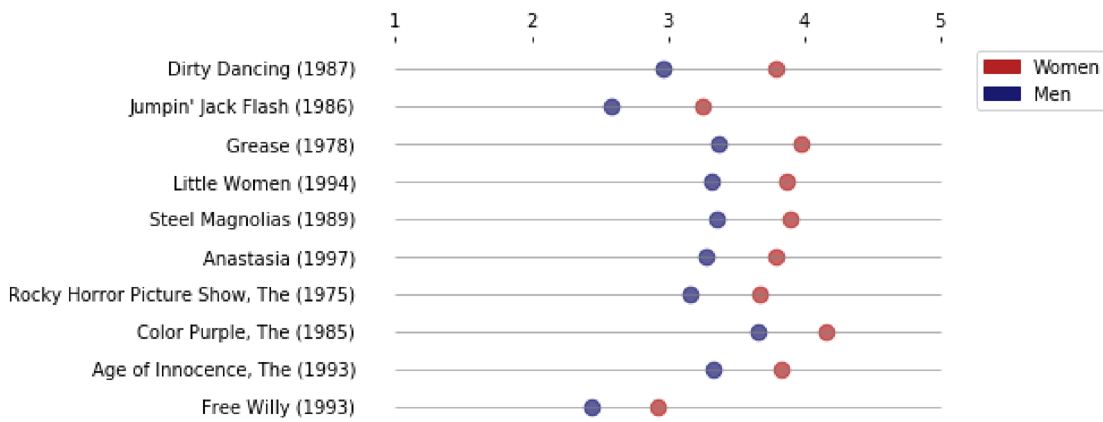
[34]:

```
gender                                     F         M  Preferred by F
title
Dirty Dancing (1987)                3.790378  2.959596        0.830782
Jumpin' Jack Flash (1986)           3.254717  2.578358        0.676359
Grease (1978)                       3.975265  3.367041        0.608224
Little Women (1994)                 3.870588  3.321739        0.548849
Steel Magnolias (1989)              3.901734  3.365957        0.535777
Anastasia (1997)                    3.800000  3.281609        0.518391
Rocky Horror Picture Show, The (1975) 3.673016  3.160131        0.512885
Color Purple, The (1985)            4.158192  3.659341        0.498851
Age of Innocence, The (1993)        3.827068  3.339506        0.487561
Free Willy (1993)                   2.921348  2.438776        0.482573
```

Visualize it: show the rating differences in a full rating scale, to visualize how far apart they actually are (based on this post).

[35]:

```
import matplotlib.patches as mpatches


def plot_ratings_difference(df):
    F_COLOR, M_COLOR = 'firebrick', 'midnightblue'

    fig, ax = plt.subplots()
    ax.hlines(y=df.index, xmin=1, xmax=5, color='gray', alpha=0.7,
              linewidth=0.8)
    ax.scatter(y=df.index, x=df.F, s=75, color=F_COLOR, alpha=0.7)
    ax.scatter(y=df.index, x=df.M, s=75, color=M_COLOR, alpha=0.7)

    format_graph(ax)
    plt.gca().invert_yaxis()
    plt.xticks([1, 2, 3, 4, 5]);
    ax.xaxis.set_ticks_position('top')

    # Manually add the legend
    red_patch = mpatches.Patch(color='red', label='The red data')
    blue_patch = mpatches.Patch(color='blue', label='The blue data')
    plt.legend(loc='upper right', bbox_to_anchor=(1.25, 1),
        handles=[mpatches.Patch(color=F_COLOR, label='Women'),
        mpatches.Patch(color=M_COLOR, label='Men')])

plot_ratings_difference(pref_by_women)
```

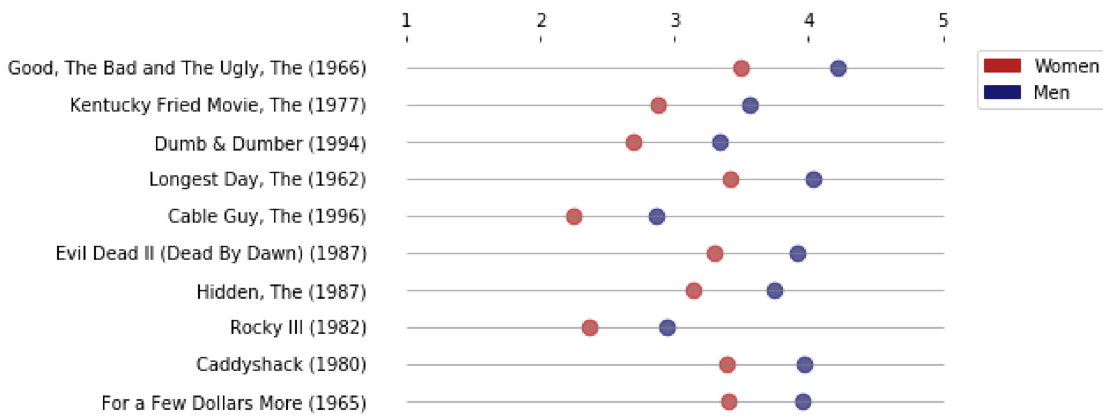


5.1.2 The top 10 movies with the greatest rating difference in the opposite direction (sign) so that we can see which ones were preferred by men.

[36]:
```
pref_by_men = mean_ratings.sort_values(by='Preferred by F', ascending=True)[:10]
pref_by_men
```

[36]:
```
gender                                         F         M  Preferred by F
title
Good, The Bad and The Ugly, The (1966)  3.494949  4.221300       -0.726351
Kentucky Fried Movie, The (1977)        2.878788  3.555147       -0.676359
Dumb & Dumber (1994)                    2.697987  3.336595       -0.638608
Longest Day, The (1962)                 3.411765  4.031447       -0.619682
Cable Guy, The (1996)                   2.250000  2.863787       -0.613787
Evil Dead II (Dead By Dawn) (1987)      3.297297  3.909283       -0.611985
Hidden, The (1987)                      3.137931  3.745098       -0.607167
Rocky III (1982)                        2.361702  2.943503       -0.581801
Caddyshack (1980)                       3.396135  3.969737       -0.573602
For a Few Dollars More (1965)           3.409091  3.953795       -0.544704
```

[37]:
```
plot_ratings_difference(pref_by_men)
```



6 BONUS! (up to 10 points)

Write Python code to display the top 10 movies (with 250 ratings or more) that elicited the most disagreement among viewers, independent of gender identification.

Hint: Disagreement can be measured by the *variance* or *standard deviation* of the ratings.

6.1 Solution

6.1.1 Write Python code to display the top 10 movies (with 250 ratings or more) that elicited the most disagreement among viewers, independent of gender identification.

Step 1: count how many votes each movie received in the 1-5 rating scale. This gives the raw disagreement count.

[38]:
```
total_ratings = data.pivot_table('user_id', index='title',
                                 columns='rating', aggfunc='count')
total_ratings.head(3)
```

[38]:
```
rating                    1     2     3     4     5
title
$1,000,000 Duck (1971)   3.0   8.0  15.0   7.0   4.0
'Night Mother (1986)     4.0  10.0  25.0  18.0  13.0
'Til There Was You (1997) 5.0  20.0  15.0  10.0   2.0
```

Step 2: Change the raw counts into ratios, to normalize by number of reviewers. Otherwise movies with more reviewers would naturally have higher disagreement, just by having larger numbers in the calculations we will do later.

[39]:
```python
sum_ratings = total_ratings.sum(axis=1)
for c in total_ratings.columns:
    total_ratings[c] /= sum_ratings

# Check that we normalized correctly
assert(np.allclose(total_ratings.sum(axis=1), 1))

total_ratings.head(3)
```

[39]:
```
rating                            1         2         3         4         5
title
$1,000,000 Duck (1971)     0.081081  0.216216  0.405405  0.189189  0.108108
'Night Mother (1986)       0.057143  0.142857  0.357143  0.257143  0.185714
'Til There Was You (1997)  0.096154  0.384615  0.288462  0.192308  0.038462
```

Step 3: Calculate a *disagreement measure*. We will use `std()` for that.

[40]:
```python
total_ratings['disagreement'] = total_ratings.std(axis=1)
total_ratings.head(3)
```

[40]:
```
rating                            1         2         3         4         5  \
title
$1,000,000 Duck (1971)     0.081081  0.216216  0.405405  0.189189  0.108108
'Night Mother (1986)       0.057143  0.142857  0.357143  0.257143  0.185714
'Til There Was You (1997)  0.096154  0.384615  0.288462  0.192308  0.038462

rating                     disagreement
title
$1,000,000 Duck (1971)         0.127629
'Night Mother (1986)           0.113838
'Til There Was You (1997)      0.140398
```

Step 4: Filter by number of reviewers, sort and display results

Note that we want the movies with the lowest standard deviation. That means the ratings are more evenly spread in the rating scale, indicating reviewers do not agree on a rating. High standard deviation happens when one of the ratings receives most of the votes, indicating consensus.

The ratings are shown in a heatmap, using Pandas styling. The heatmap was chosen to visualize how close the ratings are (resulting in a low standard deviation). The closeness of ratings shows up in the heatmap as cells (in the same row) having similar colors.

To accomplish that:

1. `low` and `high` were set to match the 0-100% scale of the overal distribution of ratings. If they are not set, the heatmap would color based on values on the table, breaking the visualization.
2. The heatmap uses a sequential colormap, to further highlight how close they are (as opposed to a diverging colormap - see more in this Matplotlib tutorial).

[41]:
```python
total_ratings = total_ratings.loc[active_titles]
top_ten_disagreements = total_ratings.sort_values(
    by='disagreement', ascending=True).head(10)

top_ten_disagreements.style.format('{:.3f}') \
    .background_gradient(subset=[1,2,3,4,5], cmap='binary',
                         axis='columns', low=0, high=1)
```

[41]: <pandas.io.formats.style.Styler at 0x120ec4358>

The next cell is used to export to PDF. Styled Pandas DataFrames are not export. The cell shows a .png saved from the cell above.

rating	1	2	3	4	5	disagreement
title						
Blair Witch Project, The (1999)	0.177	0.174	0.235	0.268	0.146	0.050
Natural Born Killers (1994)	0.157	0.150	0.254	0.269	0.170	0.057
Dumb & Dumber (1994)	0.159	0.126	0.270	0.255	0.191	0.061
Billy Madison (1995)	0.113	0.189	0.279	0.242	0.177	0.064
Eyes Wide Shut (1999)	0.116	0.194	0.232	0.289	0.169	0.065
Bicentennial Man (1999)	0.178	0.188	0.290	0.240	0.104	0.070
Rocky Horror Picture Show, The (1975)	0.118	0.144	0.263	0.282	0.194	0.072
Scary Movie (2000)	0.147	0.186	0.292	0.259	0.116	0.074
Babe: Pig in the City (1998)	0.107	0.157	0.260	0.291	0.185	0.075
Serial Mom (1994)	0.139	0.190	0.270	0.285	0.116	0.076

Contrast with the "agreement" heatmap below, showing the top 10 movies for which users gave similar ratings. Cells in a row bounce between light and dark colors, without other shades in between.

[42]:
```python
total_ratings.sort_values(
    by='disagreement', ascending=True).tail(10) \
    .style.format('{:.3f}') \
    .background_gradient(subset=[1,2,3,4,5], cmap='binary',
                         axis='columns', low=0, high=1)
```

[42]: <pandas.io.formats.style.Styler at 0x120e33da0>

The next cell is used to export to PDF. Styled Pandas DataFrames are not export. The cell shows a .png saved from the cell above.

rating	1	2	3	4	5	disagreement
title						
Raiders of the Lost Ark (1981)	0.002	0.015	0.085	0.302	0.597	0.252
Battlefield Earth (2000)	0.646	0.167	0.126	0.053	0.009	0.257
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	0.004	0.011	0.091	0.277	0.617	0.258
Wrong Trousers, The (1993)	0.007	0.010	0.054	0.325	0.603	0.261
Close Shave, A (1995)	0.005	0.003	0.061	0.330	0.601	0.262
Schindler's List (1993)	0.008	0.012	0.081	0.259	0.640	0.266
Usual Suspects, The (1995)	0.004	0.017	0.076	0.260	0.642	0.267
Godfather, The (1972)	0.008	0.017	0.080	0.231	0.664	0.274
Shawshank Redemption, The (1994)	0.004	0.011	0.066	0.264	0.654	0.275
Seven Samurai (The Magnificent Seven) (Shichinin no samurai) (1954)	0.006	0.018	0.062	0.237	0.677	0.282

7 BONUS! (up to 10 points)

Write Python code to answer the question: What is the most popular movie genre? Plot a bar chart of the genre counts and report the size of the largest bin.

Hint: use the original **movies** data frame, *before* the merge!

7.1 Solution

With thanks to this Stackoverflow answer for pointing to the Pandas `get_dummies` function.

This is done in two steps:

1. `get_dummies()` splits the genres into columns (hot-encodes them).
2. `sum()` adds all the 1s that `get_dummies()` created.

Although we can do it all in one line, we will do in steps to understand it better.

Step 1: split the genres into hot-encoded columns

[43]:
```
genres = movies.genres.str.get_dummies()
genres.head(3)
```

[43]:
```
   Action  Adventure  Animation  Children's  Comedy  Crime  Documentary  \
0       0          0          1           1       1      0            0
1       0          1          0           1       0      0            0
2       0          0          0           0       1      0            0

   Drama  Fantasy  Film-Noir  Horror  Musical  Mystery  Romance  Sci-Fi  \
0      0        0          0       0        0        0        0       0
1      0        1          0       0        0        0        0       0
2      0        0          0       0        0        0        1       0
```

```
   Thriller  War  Western
0         0    0        0
1         0    0        0
2         0    0        0
```

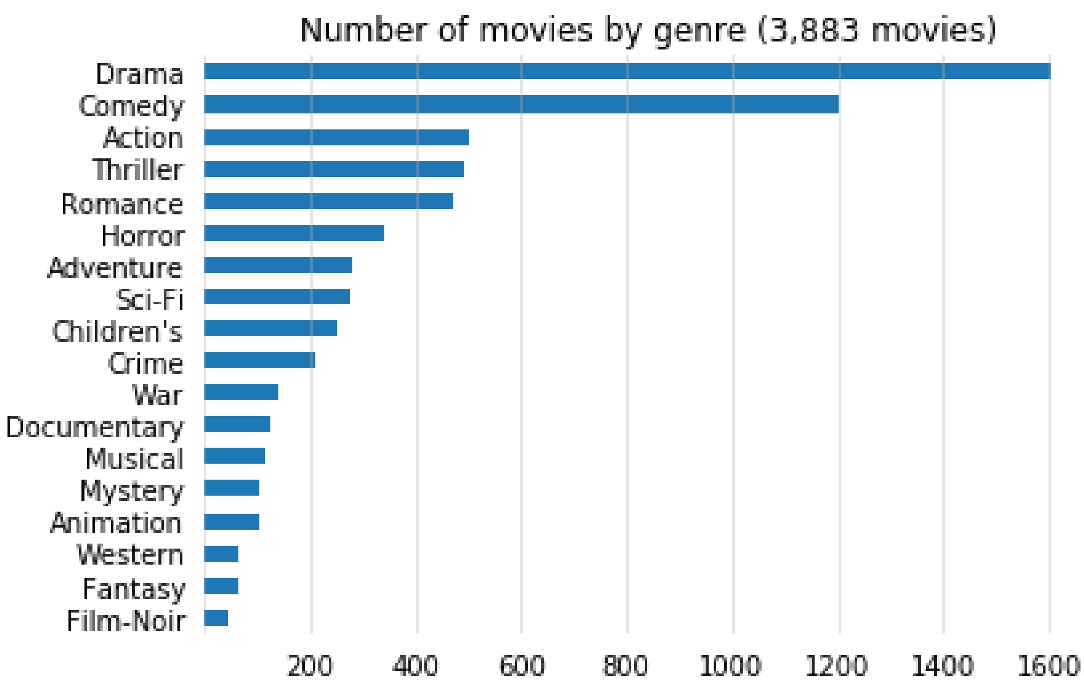
Step 2: count each genre and sort them so the chart looks better.

[44]:
```
genres = genres.sum().sort_values()
genres.head(3)
```

[44]:
```
Film-Noir    44
Fantasy      68
Western      68
dtype: int64
```

Step 3: plot the genres and customize the graph to increase information/pixels ratio.

[45]:
```
title = 'Number of movies by genre ({:,} movies)'.format(len(movies))
formatted_barh_graph(genres, title)
```



The largest category, as requested in the question.

[46]:
```
print('The largest movie category is {}, with {:,} movies'
      .format(genres.tail(1).index[0],genres[-1]))
```

```
The largest movie category is Drama, with 1,603 movies
```

7.2 Part 2: Titanic

In this part we'll use the dataset of passengers on the *Titanic*, available through the Seaborn library.

See https://www.kaggle.com/c/titanic/data for codebook and additional information.

[47]:
```
titanic = sns.load_dataset('titanic')
```

[48]:
```
titanic.head()
```

[48]:
```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
```

7.3 Questions 11-14 (16 points total, i.e. 4 pts each)

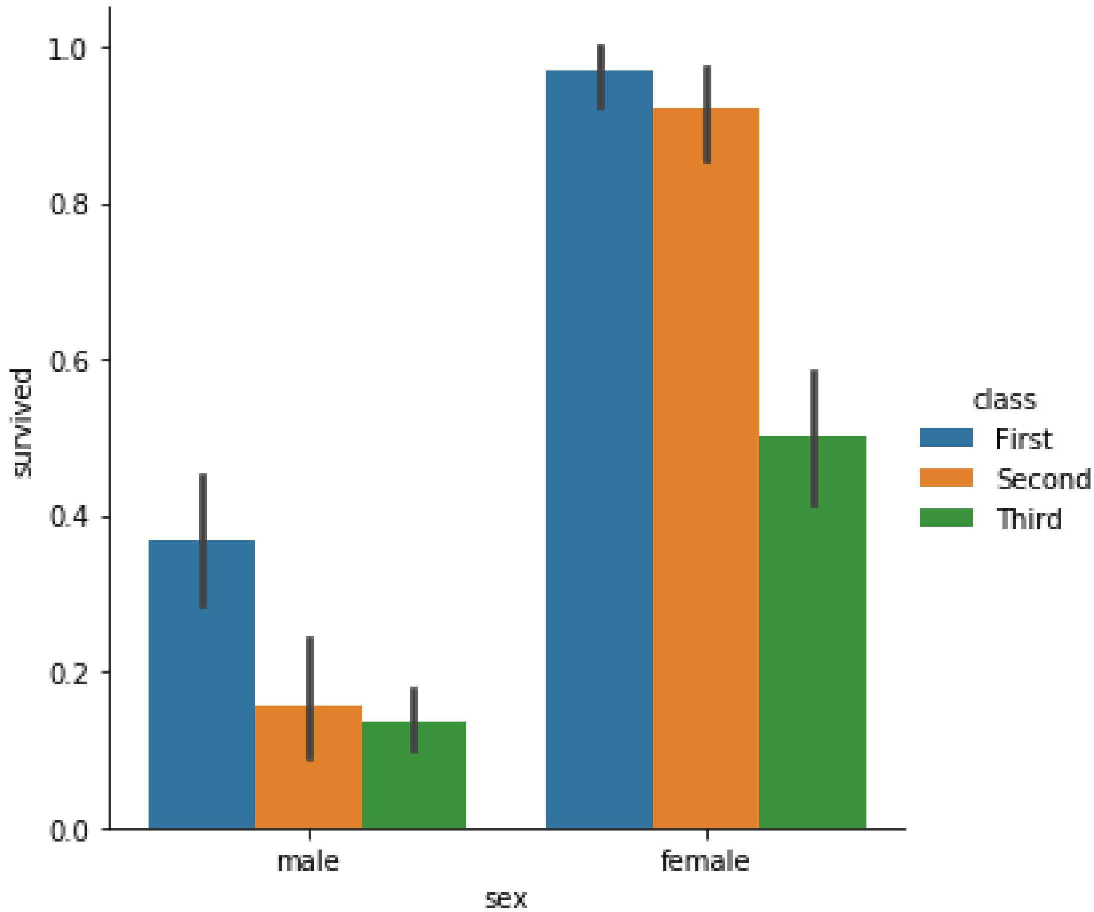
Look at the Python code below and answer the following questions (expressing the amounts in absolute terms):

11. How many female passengers did not survive (regardless of their class)?
12. How many first class female passengers did not survive?
13. How many male passengers did not survive (regardless of their class)?
14. How many third class male passengers did not survive?

[49]:
```
titanic.pivot_table('survived', index='sex', columns='class', margins=True)
```

[49]:
```
class      First    Second     Third       All
sex
female  0.968085  0.921053  0.500000  0.742038
male    0.368852  0.157407  0.135447  0.188908
All     0.629630  0.472826  0.242363  0.383838
```

[50]:
```
sns.catplot(x="sex", y="survived", hue="class", kind="bar", data=titanic);
```



7.4 Solution

7.4.1 11. How many female passengers did not survive (regardless of their class)?

```
[51]: def genre_died(genre):
          return (titanic['sex'] == genre) & (titanic['survived'] == 0)

      print('{} female passangers did not survive'.format(
          len(titanic[genre_died('female')])))
```

```
81 female passangers did not survive
```

7.4.2 12. How many first class female passengers did not survive?

[52]:
```
print('{} first class female passengers did not survive'.format(
    len(titanic[genre_died('female') & (titanic['class'] == 'First')])))
```

```
3 first class female passengers did not survive
```

7.4.3 13. How many male passengers did not survive (regardless of their class)?

[53]:
```
print('{} male passangers did not survive'.format(
    len(titanic[genre_died('male')])))
```

```
468 male passangers did not survive
```

7.4.4 14. How many third class male passengers did not survive?

[54]:
```
print('{} third class male passengers did not survive'.format(
    len(titanic[genre_died('male') & (titanic['class'] == 'Third')])))
```

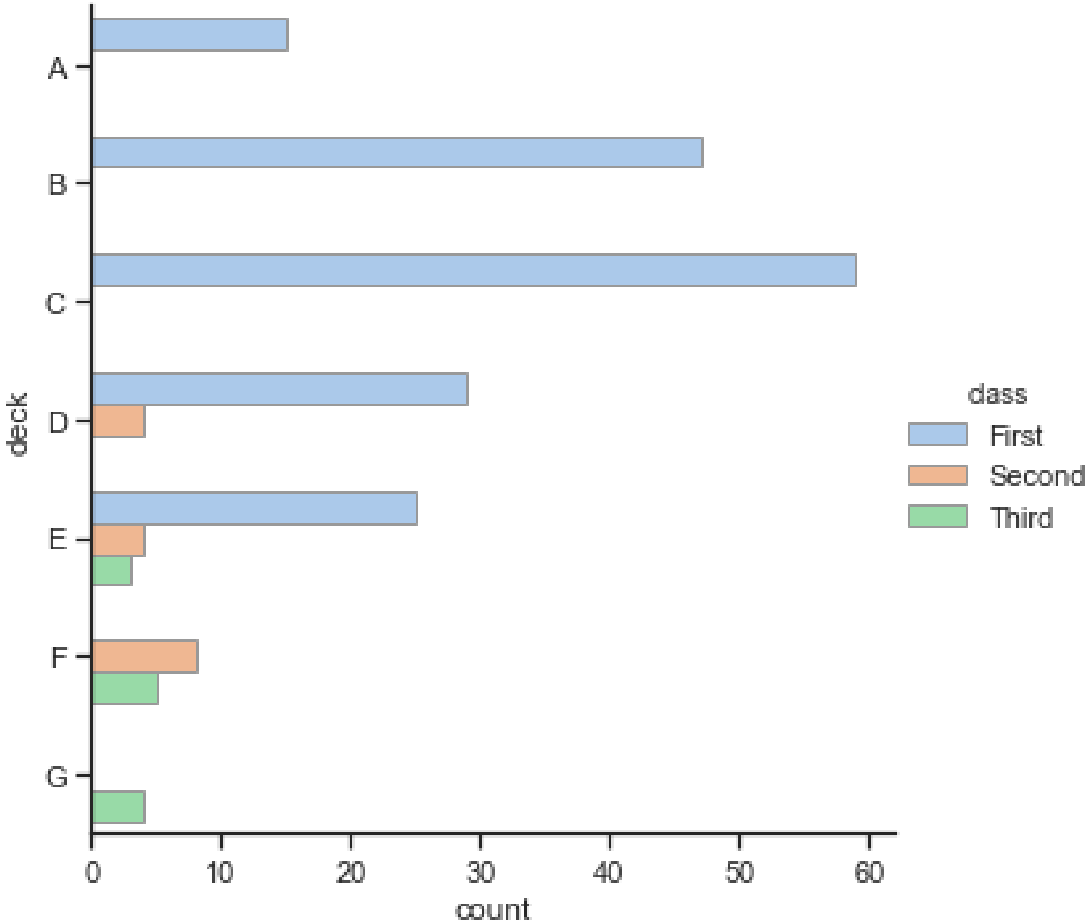
```
300 third class male passengers did not survive
```

7.5 Your turn! (12 points, i.e., 4 pts each)

Write Python code to answer the following questions (make sure the messages displayed by your code are complete and descriptive enough):

15. How many passengers (absolute number) were there per deck/class?

(**Hint**: The plot below shows how decks and classes were related and provides a visual estimate.) 16. How many passengers (absolute number) in Deck A survived? 17. How many passengers (absolute number) in Deck E survived and what was the breakdown (in Deck E) per class?



7.6 Solution

7.6.1 15. How many passengers (absolute number) were there per deck/class?

First we need to fix the missing deck entries. Because it is a category, we need to expand the category with a value that represents "missing".

[55]:
```
# https://stackoverflow.com/a/36193135
titanic.deck = titanic.deck.cat.add_categories(['Unknown'])
titanic.deck.fillna('Unknown', inplace=True)
```

With that in place, we can find the counts with the pivot table. We could `count` on different columns to get the number of passagers, so we picked a column that does not have `NaN` (or we would have to deal with that first).

[56]:
```
# Aggregate by `count` to consider all passengers (survivors or not)
# To show only survivors (survived=1), aggregate by `sum`
titanic.pivot_table('survived', index='deck', columns='class',
                    aggfunc='count', margins=True, fill_value='')
```

[56]:

class	First	Second	Third	All
deck				
A	15			15
B	47			47
C	59			59
D	29	4		33
E	25	4	3	32
F		8	5	13
G			4	4
Unknown	41	168	479	688
All	216	184	491	891

7.6.2 16. How many passengers (absolute number) in Deck A survived?

[57]:
```
print('{} passengers in deck A survived'
      .format(len(titanic.query('(deck == "A") & (survived == 1)'))))
```

```
7 passengers in deck A survived
```

7.6.3 17. How many passengers (absolute number) in Deck E survived and what was the breakdown (in Deck E) per class?

[58]:
```
titanic[titanic['deck'] == 'E'].pivot_table(
    'survived', index='class', aggfunc='sum', margins=True)
```

[58]:

	survived
class	
First	18
Second	3
Third	3
All	24

Why `aggfunc('sum')` works here: `survived` is an integer with 0 or 1 as value. Summing up that column is the same as counting survivors (the 1s).

8 BONUS! (up to 20 points)

Write Python code to answer the following questions (using percentage values): - How many women traveling alone did not survive? - How many men 35 years old or younger did not survive? - What was the average fare per class?

8.1 Solution

8.1.1 How many women traveling alone did not survive?

```
[59]: print('{} women travelling alone did not survive'
            .format(len(titanic.query('(sex == "female") & alone & (survived ==␣
       ↪0)'))))
```

```
27 women travelling alone did not survive
```

8.1.2 How many men 35 years old or younger did not survive?

```
[60]: print('{} men 35 years old or younger did not survive'
            .format(len(titanic.query('(sex == "male") & (age <= 35) & (survived ==␣
       ↪0)'))))
```

```
242 men 35 years old or younger did not survive
```

8.1.3 What was the average fare per class?

Two solutions, for comparison.

```
[61]: titanic.groupby('class')['fare'].mean()
```

```
[61]: class
      First     84.154687
      Second    20.662183
      Third     13.675550
      Name: fare, dtype: float64
```

```
[62]: # aggregration by `mean` is the default
      titanic.pivot_table('fare', index='class')
```

```
[62]:              fare
      class
      First   84.154687
      Second  20.662183
      Third   13.675550
```

8.2 Part 3: US Baby Names 1880–2018

The United States Social Security Administration (SSA) has made available data on the frequency of baby names from 1880 through the present. These plain text data files, one per year, contain the total number of births for each sex/name combination. The raw archive of these files can be obtained from http://www.ssa.gov/oact/babynames/limits.html.

After downloading the 'National data' file *names.zip* and unzipping it, you will have a directory containing a series of files like *yob1880.txt* through *yob2018.txt*. We need to do some data wrangling to load this dataset (see code below).

For your convenience, I have made the *names.zip* file available on Canvas. Before running the cell below, make sure that you have downloaded it, unzipped it, and placed its contents under the 'data' folder.

[63]:
```
years = range(1880, 2019)

pieces = []
columns = ['name', 'sex', 'births']

for year in years:
    path = 'data/names/yob%d.txt' % year
    frame = pd.read_csv(path, names=columns)

    frame['year'] = year
    pieces.append(frame)

# Concatenate everything into a single DataFrame
names = pd.concat(pieces, ignore_index=True)
```

[64]:
```
names
```

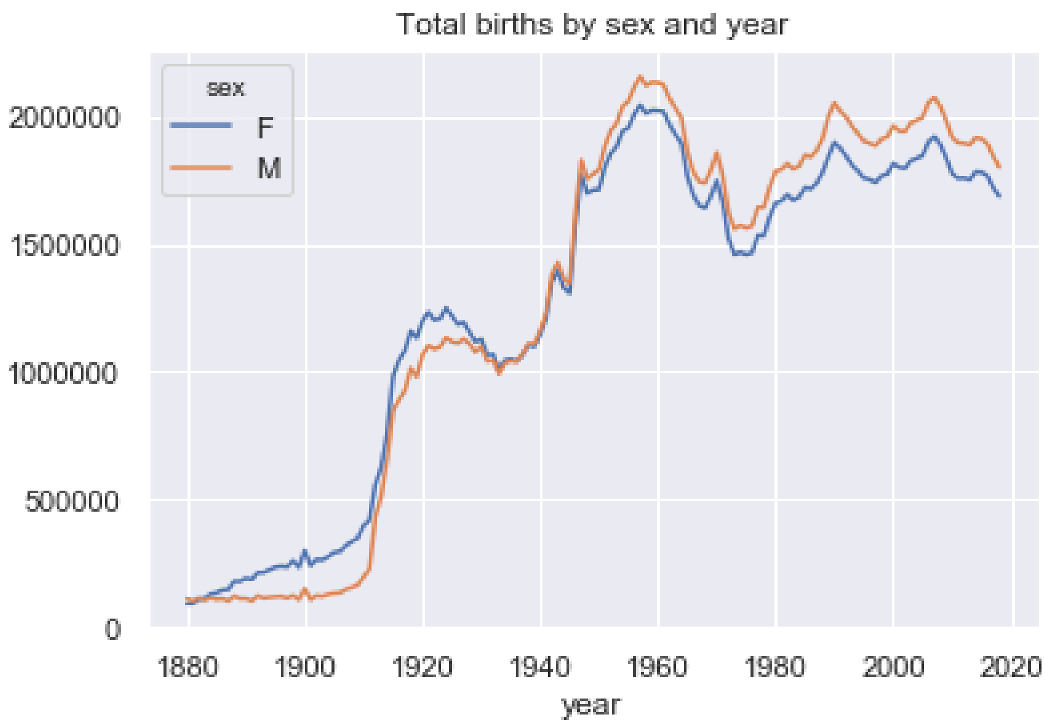
[64]:
```
              name sex  births  year
0             Mary   F    7065  1880
1             Anna   F    2604  1880
2             Emma   F    2003  1880
3        Elizabeth   F    1939  1880
4           Minnie   F    1746  1880
…              …  ..     …   …
1957041      Zylas   M       5  2018
1957042      Zyran   M       5  2018
1957043      Zyrie   M       5  2018
1957044      Zyron   M       5  2018
1957045      Zzyzx   M       5  2018

[1957046 rows x 4 columns]
```

8.3 Your turn! (25 points)

Write Python code to compute the number of baby boys and baby girls born each year and display the two line plots over time.

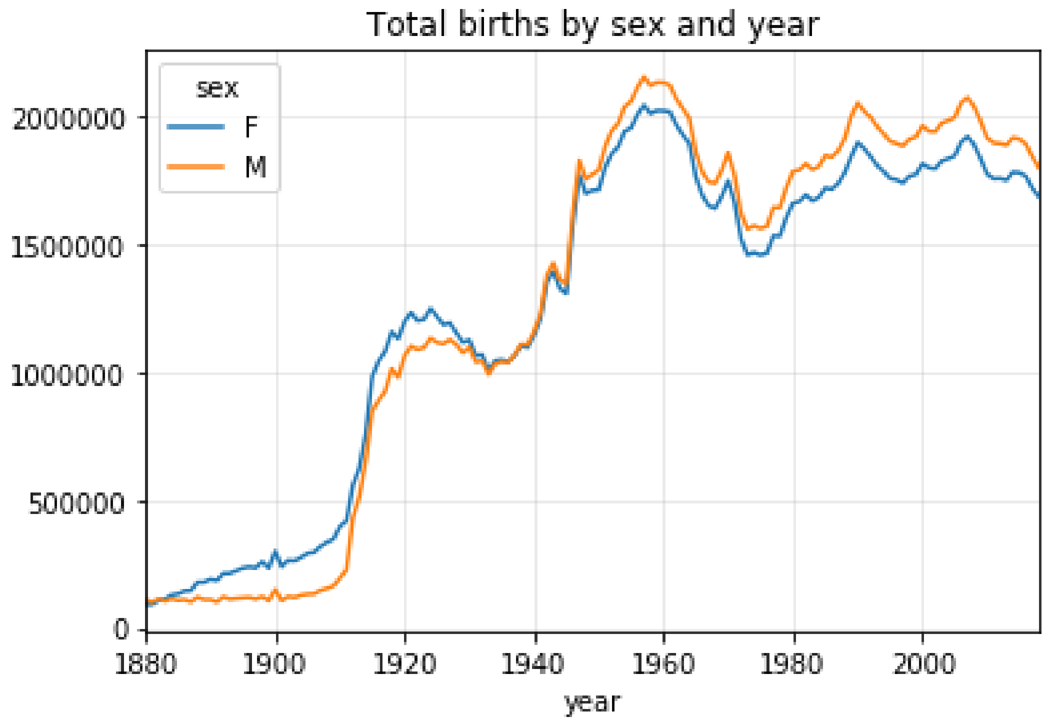
Hint: Start by aggregating the data at the year and sex level using `groupby` or `pivot_table`.

Your plot should look like this:



8.4 Solution

```
[65]: ax = names.pivot_table('births', index='year', columns='sex', aggfunc='sum') \
            .plot(title='Total births by sex and year')
      ax.grid(alpha=0.3)
```



8.5 Analyzing Naming Trends

Suppose we're interested in analyzing the Top 1000 most popular baby names per year.

We will do so by following these steps: 1. Insert a column *prop* with the fraction of babies given each name relative to the total number of births. A prop value of 0.02 would indicate that 2 out of every 100 babies were given a particular name in a given year. 2. Group the data by year and sex, then add the new column to each group. 3. Extract a subset of the data (the top 1,000 names for each sex/year combination). This is yet another group operation. 4. Split the Top 1,000 names into the boy and girl portions. 5. Build a pivot table of the total number of births by year and name.

Finally, we will plot the absolute number of babies named 'John', 'Noah', 'Madison', or 'Lorraine' over time.

[66]:
```
def add_prop(group):
    group['prop'] = group.births / group.births.sum()
    return group
names = names.groupby(['year', 'sex']).apply(add_prop)
```

[67]:
```
names
```

[67]:
```
                 name sex  births  year      prop
0                Mary   F    7065  1880  0.077642
1                Anna   F    2604  1880  0.028617
2                Emma   F    2003  1880  0.022012
3           Elizabeth   F    1939  1880  0.021309
4              Minnie   F    1746  1880  0.019188
…                 …  ..      …    …         …
1957041         Zylas   M       5  2018  0.000003
1957042         Zyran   M       5  2018  0.000003
1957043         Zyrie   M       5  2018  0.000003
1957044         Zyron   M       5  2018  0.000003
1957045         Zzyzx   M       5  2018  0.000003

[1957046 rows x 5 columns]
```

[68]:
```python
# Sanity check (all percentages should add up to 1, i.e., 100%)
names.groupby(['year', 'sex']).prop.sum()
```

[68]:
```
year  sex
1880  F      1.0
      M      1.0
1881  F      1.0
      M      1.0
1882  F      1.0
            …
2016  M      1.0
2017  F      1.0
      M      1.0
2018  F      1.0
      M      1.0
Name: prop, Length: 278, dtype: float64
```

[69]:
```python
def get_top1000(group):
    return group.sort_values(by='births', ascending=False)[:1000]
grouped = names.groupby(['year', 'sex'])
top1000 = grouped.apply(get_top1000)
# Drop the group index, not needed
top1000.reset_index(inplace=True, drop=True)
```

[70]:
```python
top1000
```

[70]:
```
                 name sex  births  year      prop
0                Mary   F    7065  1880  0.077642
1                Anna   F    2604  1880  0.028617
2                Emma   F    2003  1880  0.022012
3           Elizabeth   F    1939  1880  0.021309
4              Minnie   F    1746  1880  0.019188
```

```
...         ...  ..     ...   ...      ...
277872      Korbyn   M     207  2018  0.000115
277873     Randall   M     207  2018  0.000115
277874      Benton   M     206  2018  0.000114
277875     Coleman   M     206  2018  0.000114
277876      Markus   M     206  2018  0.000114

[277877 rows x 5 columns]
```

[71]:
```python
boys = top1000[top1000.sex == 'M']
girls = top1000[top1000.sex == 'F']
```

[72]:
```python
total_births = top1000.pivot_table('births', index='year',
                                   columns='name',
                                   aggfunc=sum)
```

[73]:
```python
total_births.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 139 entries, 1880 to 2018
Columns: 7174 entries, Aaden to Zyaire
dtypes: float64(7174)
memory usage: 7.6 MB
```

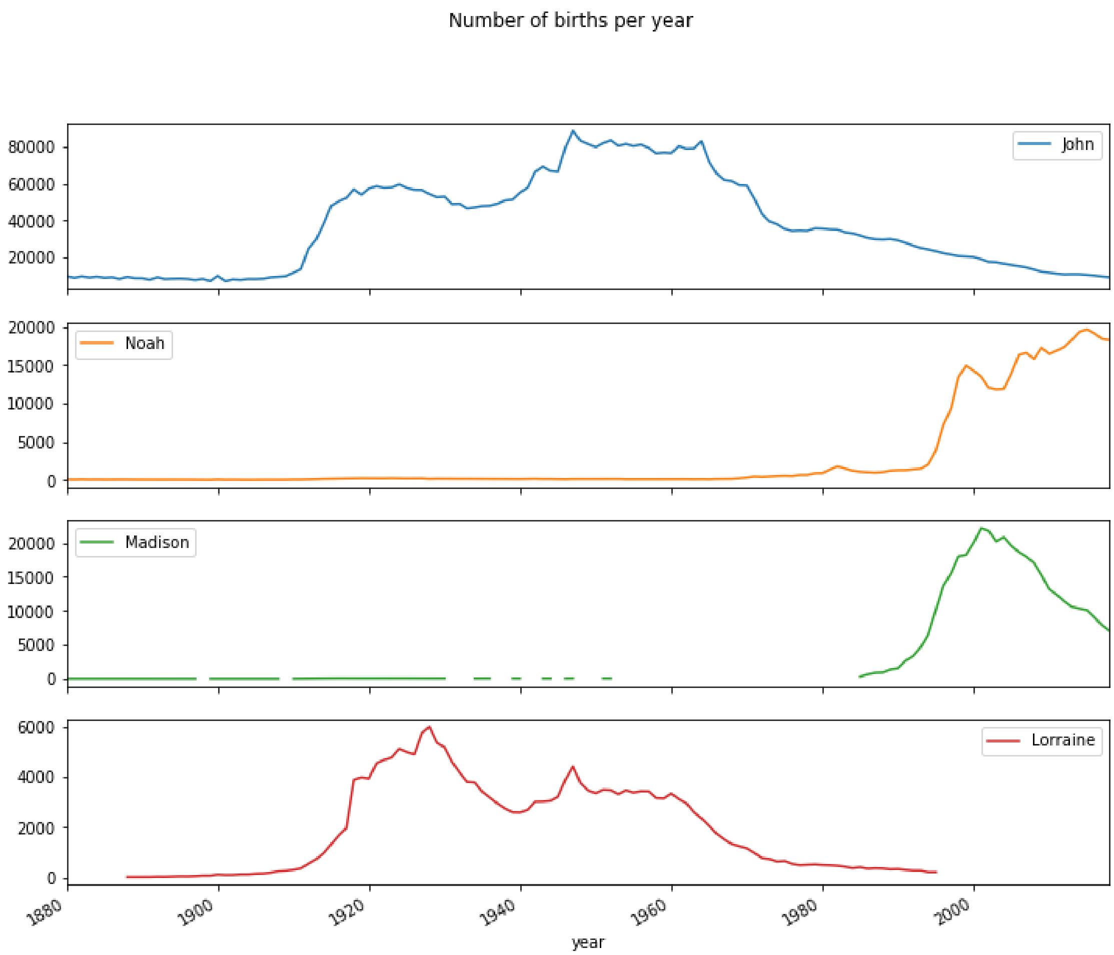
[74]:
```python
total_births
```

[74]:
```
name  Aaden  Aadhya  Aaliyah  Aanya  Aarav   Aaron  Aarush  Ab  Abagail  Abb  \
year
1880    NaN     NaN      NaN    NaN    NaN   102.0     NaN NaN      NaN  NaN
1881    NaN     NaN      NaN    NaN    NaN    94.0     NaN NaN      NaN  NaN
1882    NaN     NaN      NaN    NaN    NaN    85.0     NaN NaN      NaN  NaN
1883    NaN     NaN      NaN    NaN    NaN   105.0     NaN NaN      NaN  NaN
1884    NaN     NaN      NaN    NaN    NaN    97.0     NaN NaN      NaN  NaN
...     ...     ...      ...    ...    ...     ...     ...  ..      ...  ...
2014  239.0     NaN   4883.0  266.0  531.0  7392.0     NaN NaN      NaN  NaN
2015  297.0     NaN   4863.0    NaN  540.0  7159.0   211.0 NaN      NaN  NaN
2016    NaN   284.0   4641.0    NaN  519.0  7157.0     NaN NaN      NaN  NaN
2017  241.0   291.0   4174.0    NaN  526.0  7196.0     NaN NaN      NaN  NaN
2018    NaN     NaN   3811.0    NaN  488.0  5953.0     NaN NaN      NaN  NaN

name  ...    Zoe   Zoey   Zoie   Zola  Zollie  Zona   Zora  Zula    Zuri  \
year  ...
1880  ...   23.0    NaN    NaN    7.0     NaN   8.0   28.0  27.0     NaN
1881  ...   22.0    NaN    NaN   10.0     NaN   9.0   21.0  27.0     NaN
1882  ...   25.0    NaN    NaN    9.0     NaN  17.0   32.0  21.0     NaN
1883  ...   23.0    NaN    NaN   10.0     NaN  11.0   35.0  25.0     NaN
1884  ...   31.0    NaN    NaN   14.0     6.0   8.0   58.0  27.0     NaN
```

```
…    …      …      …     …     …      …    …      …    …       …
2014  …  5877.0  7411.0  366.0    NaN     NaN   NaN    NaN   NaN   666.0
2015  …  6041.0  6944.0  371.0    NaN     NaN   NaN    NaN   NaN   714.0
2016  …  5743.0  6444.0  312.0    NaN     NaN   NaN    NaN   NaN   889.0
2017  …  5158.0  6045.0  323.0    NaN     NaN   NaN    NaN   NaN   849.0
2018  …  5062.0  5899.0  320.0  275.0     NaN   NaN  268.0   NaN  1122.0

name  Zyaire
year
1880     NaN
1881     NaN
1882     NaN
1883     NaN
1884     NaN
…        …
2014     NaN
2015     NaN
2016   248.0
2017   301.0
2018   322.0

[139 rows x 7174 columns]
```

[75]:

```
subset = total_births[['John', 'Noah', 'Madison', 'Lorraine']]
ax = subset.plot(subplots=True, figsize=(12, 10), grid=False,
            title="Number of births per year")
# ax = subset.plot(subplots=True, figsize=(12, 10), grid=False,
#              title="Number of births per year", ylim=(0,0.1))
```



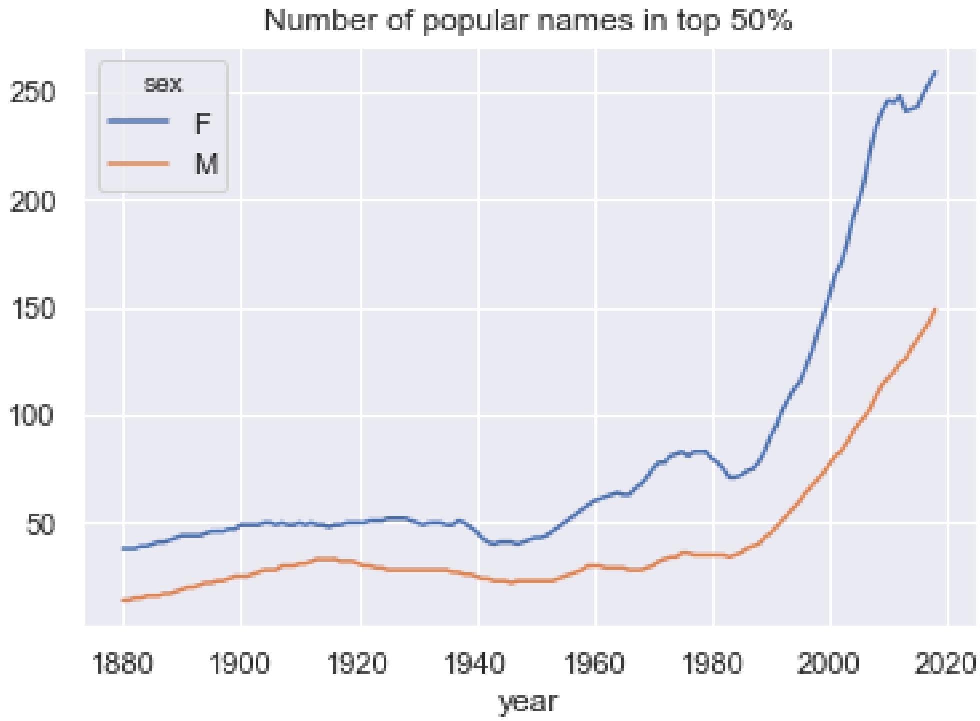
9 BONUS! (up to 25 points)

Write Python code to test the hypothesis:

H1: There has been an increase in naming diversity over time.

Hint: Compute a metric that consists of the number of distinct names, taken in order of popularity from highest to lowest, in the top 50% of births, and plot that metric over time.

Your plot should look like this:



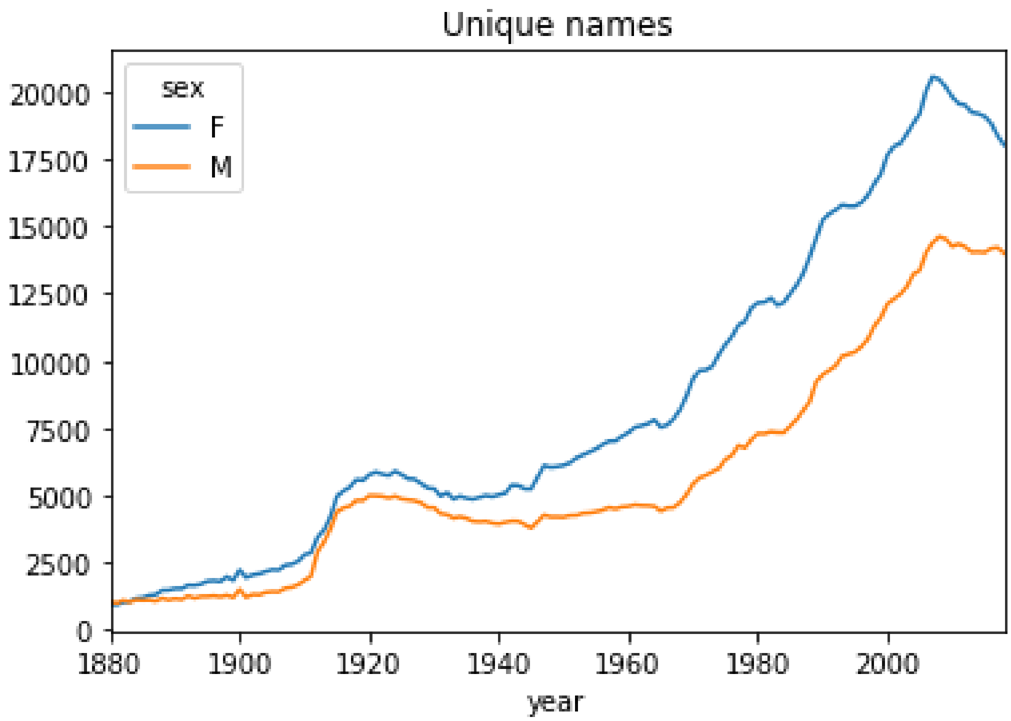
9.1 Solution

9.1.1 *Diversity* as "more names are being used"

This is the simplest possible measure of name diversity: more names are being used over time.

The graph shows that the number of unique names increased rapidly until the early 2000s. After that it started to decrease (more pronouncedly for girl names). By this metric, name diversity greatly increased during the 20th century, but in the 21st century it is decreasing.

[76]:
```
names_by_year = names.pivot_table('name', index='year', columns='sex',
                                  aggfunc='count', fill_value=0)
names_by_year.plot(title='Unique names');
```



9.1.2 *Diversity* as "more names in the top 50% births"

Another way to look at diversity is to inspect the names responsible for 50% of total number of births.

We will inspect them in two ways:

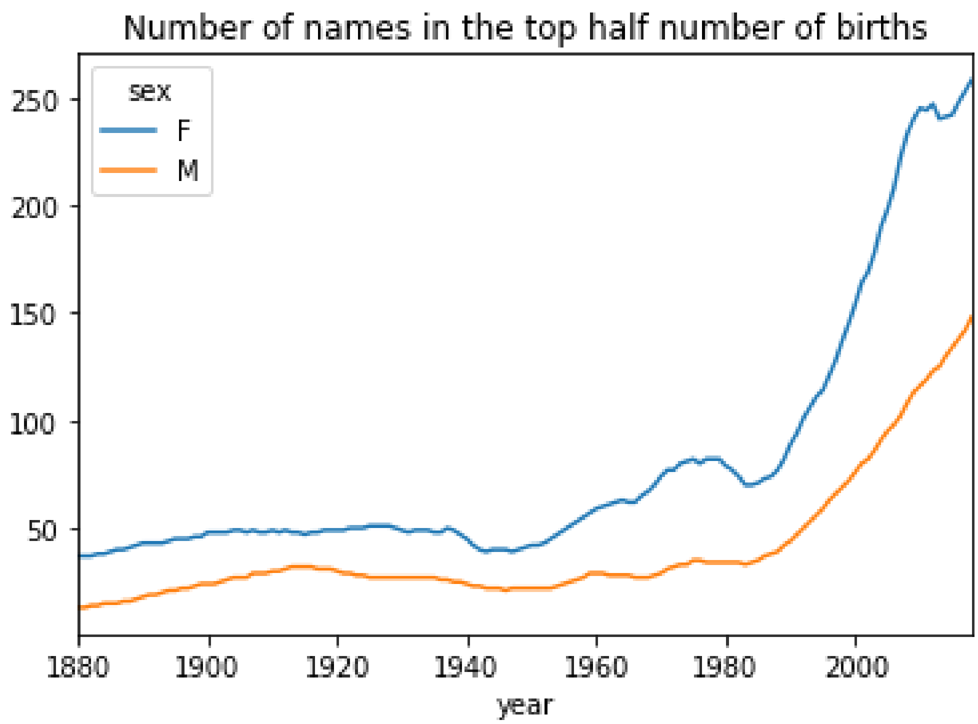
1. The absolute number of names
2. The percentage of names

The graph below shows the total number of names accounting for 50% of the number of births.

It shows that in general name diversity is growing over time, with a few declines, but generally trending up.

[77]:
```
def get_count_top_half(group):
    # Our dataset is already sorted by number of births, but we should
    # be defensive and not assume that, or the cumsum code will break
    group = group.sort_values(by='prop', ascending=False)
    return len(group[group['prop'].cumsum() <= 0.5])

# Count of births in the top half of total births
count_top_50_births = names.groupby(['year', 'sex']).apply(get_count_top_half)
# Move genre to a column, in preparation to plot it
count_top_50_births = count_top_50_births.unstack()
```

```
count_top_50_births.plot(
    title='Number of names in the top half number of births');
```

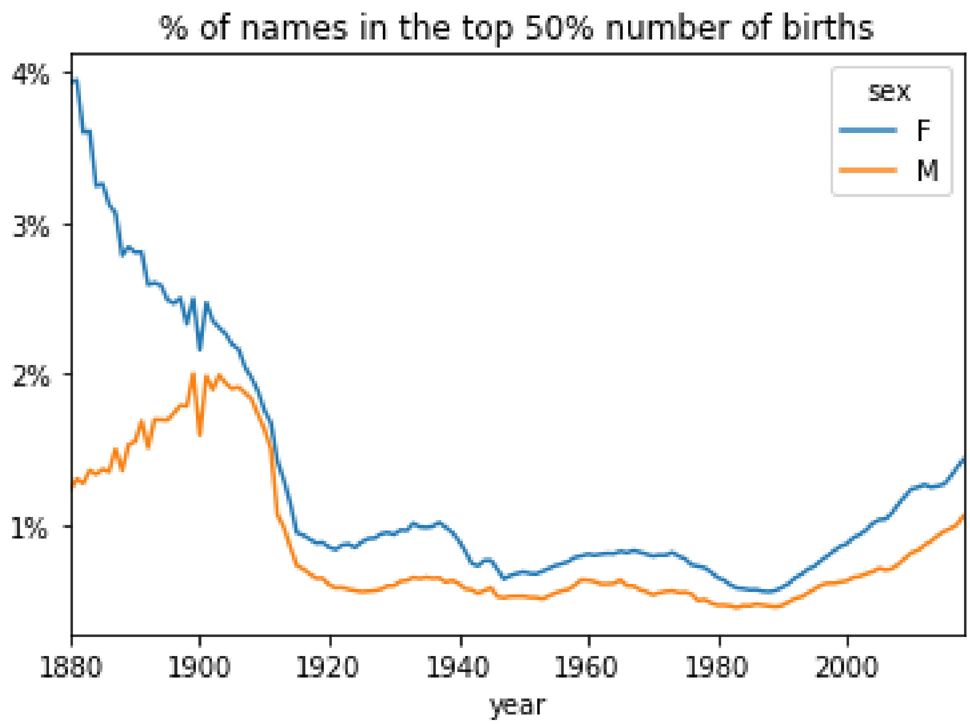


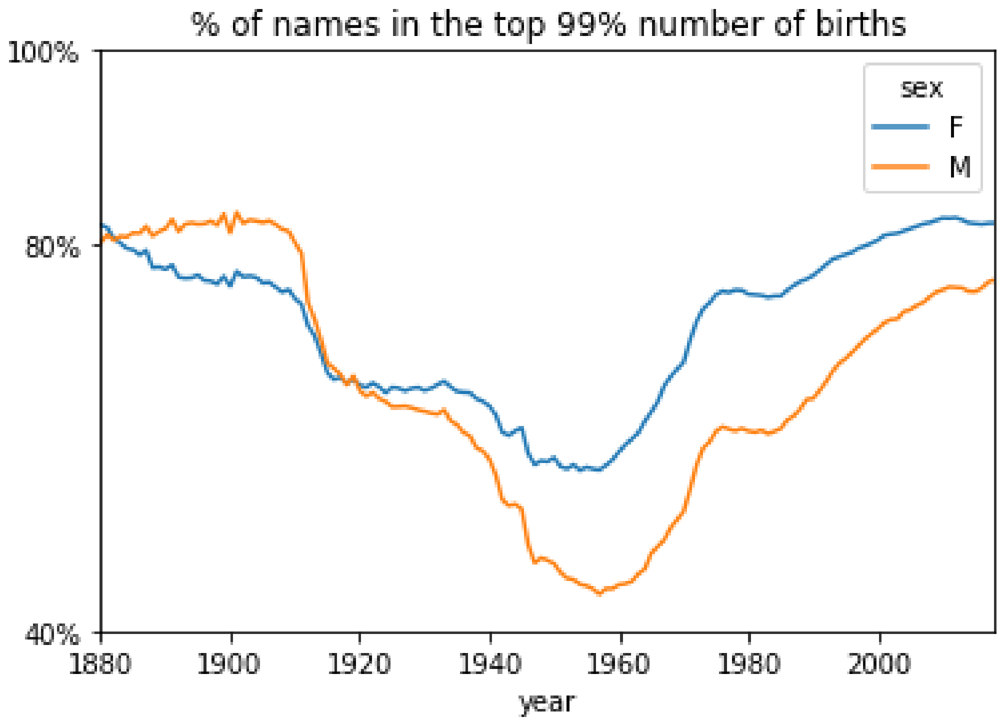
The next graphs looks at the same metric, but now in relative terms. They graph the number of names accounting for 50% and 99% of the births.

They show that the proportion of names accouting for 50% and 99% of the births declined until the 1980s (50%) and 1960s (99%), increasing diversity (less concentration of names). After that the proportion started to rise again, decreasing diversity. In other words, although we are using more names in absolute numbers (previous graph), we are picking from a smaller subset of all names used in a given year (picking from a large subset, but a smaller percentage than previous years - therefore, in that sense, decreasing diversity).

[78]:
```
q = 0.5
def get_prop_top_pct(group):
    # Our dataset is already sorted by number of births, but we should
    # be defensive and not assume that, or the cumsum code will break
    group = group.sort_values(by='prop', ascending=False)
    return len(group[group['prop'].cumsum() <= q]) / len(group)

def graph_pct_names(pct, ticks_value):
    global q
    q = pct
```

```
    # Proportion of births in the top % of total births
    top_pct_births = names.groupby(['year', 'sex']).apply(get_prop_top_pct)
    # Move genre to a column, in preparation to plot it
    top_pct_births = top_pct_births.unstack()

    top_pct_births.plot(
        title='% of names in the top {:.0f}% number of births'.format(pct*100))
    # Make the ticks more readable (match the graph title)
    ticks_text = ['{:.0f}%'.format(x*100) for x in ticks_value]
    plt.yticks(ticks_value, ticks_text);

graph_pct_names(0.5, [0.01, 0.02, 0.03, 0.04])
graph_pct_names(0.99, [0.4, 0.8, 1])
```





9.2 Boy names that became girl names (and vice versa)

Next, let's look at baby names that were more popular with one sex earlier in the sample but have switched to the opposite sex over the years. One example is the name Lesley or Leslie (or other possible, less common, spelling variations).

We will do so by following these steps: 1. Go back to the top1000 DataFrame and compute a list of names occurring in the dataset starting with "lesl". 2. Filter down to just those names and sum births grouped by name to see the relative frequencies. 3. Aggregate by sex and year and normalize within year. 4. Plot the breakdown by sex over time.

[79]:
```
all_names = pd.Series(top1000.name.unique())
lesley_like = all_names[all_names.str.lower().str.contains('lesl')]
lesley_like
```

[79]:
```
632     Leslie
2294    Lesley
4264    Leslee
4732     Lesli
6108     Lesly
dtype: object
```

[80]:
```
filtered = top1000[top1000.name.isin(lesley_like)]
filtered.groupby('name').births.sum()
```

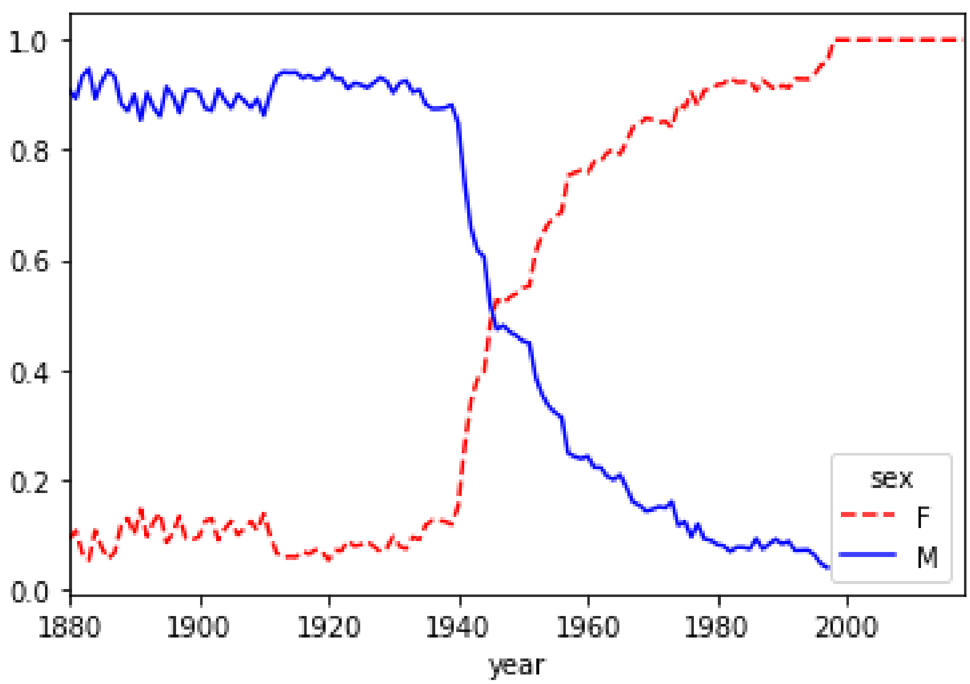
[80]:
```
name
Leslee       993
Lesley     35033
Lesli        929
Leslie    378168
Lesly      11433
Name: births, dtype: int64
```

[81]:
```
table = filtered.pivot_table('births', index='year',
                             columns='sex', aggfunc='sum')
table = table.div(table.sum(1), axis=0)
```

[82]:
```
fig = plt.figure()
table.plot(style={'M': 'b-', 'F': 'r--'})
```

[82]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x120df0128>
```

```
<Figure size 432x288 with 0 Axes>
```



Now it's time for you to come up with a different hypotheses, which we will call H2. **Be creative!**

Example: The name 'Reese' has been more prevalent among baby girls than baby boys since 2000.

9.3 Your turn! (28 points)

Write Python code to test hypothesis H2 (and some text to explain whether it was confirmed or not).

9.4 Solution

According to Wikipedia's article "Naming in the United States":

> Gender name usage also plays a role in the way parents view names. It is not uncommon for American parents to give girls names that have traditionally been used for boys. Boys, on the other hand, are almost never given feminine names. Names like Ashley, Sidney, Aubrey, and Avery originated as boys' names. Traditionally masculine or androgynous names that are used widely for girls have a tendency to be abandoned by the parents of boys and develop an almost entirely female usage

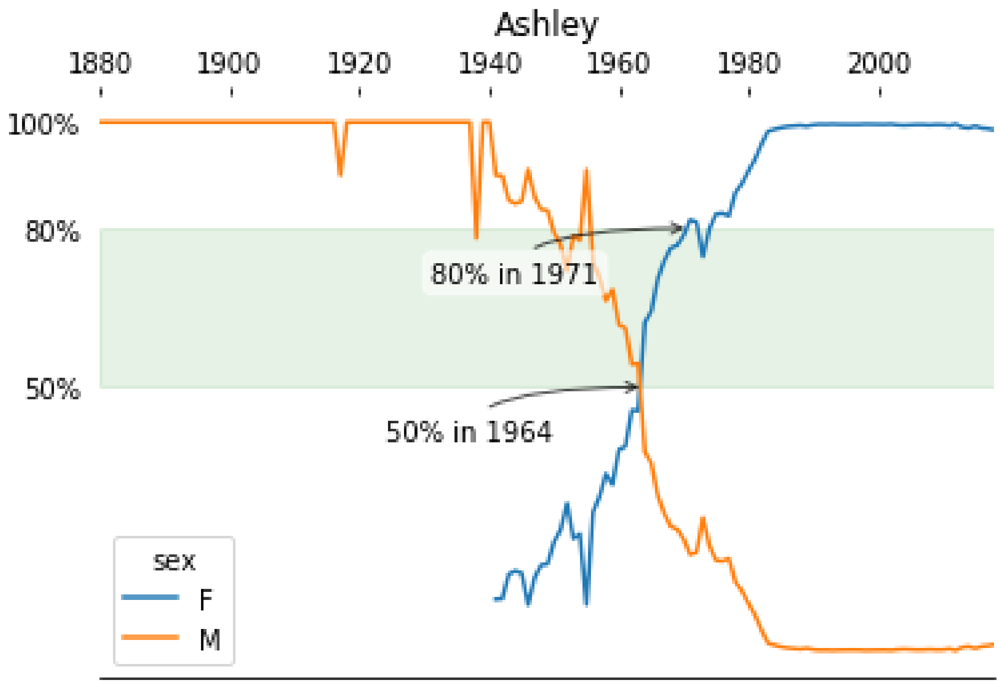
Given that statement, the hypothesis we will test is:

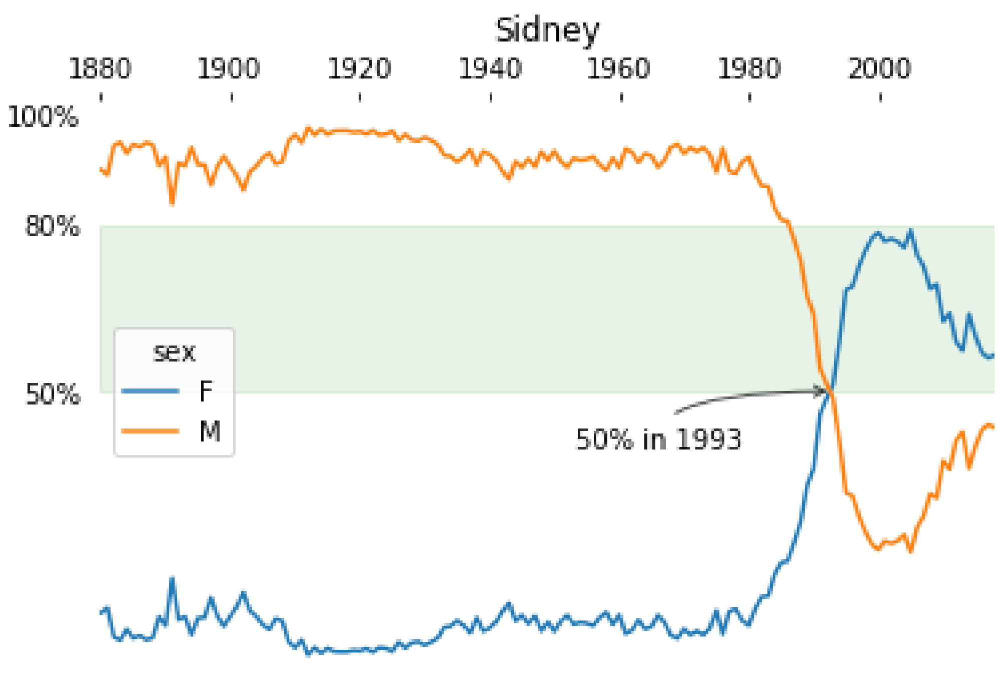
H2: Once a predominantly boy name is adopted by 50% or more of girls, within one generation (about 30 years) it will become almost exclusively (over 80%) a girl name.

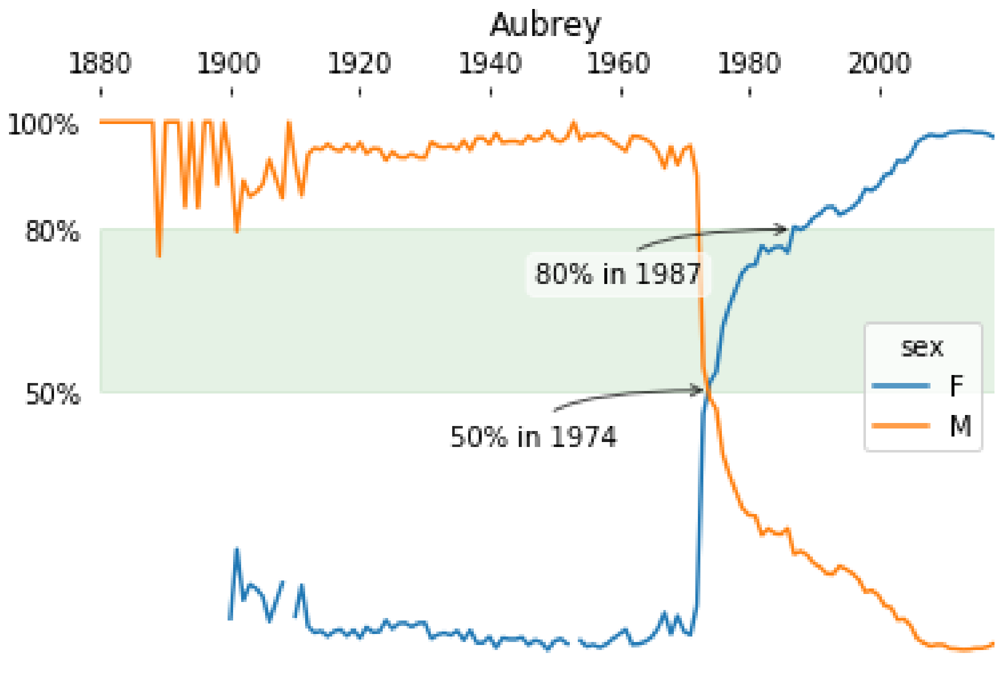
We will use the list of names mentioned in the Wikipedia article (Ashley, Sidney, Aubrey, and Avery) to test the hypothesis.

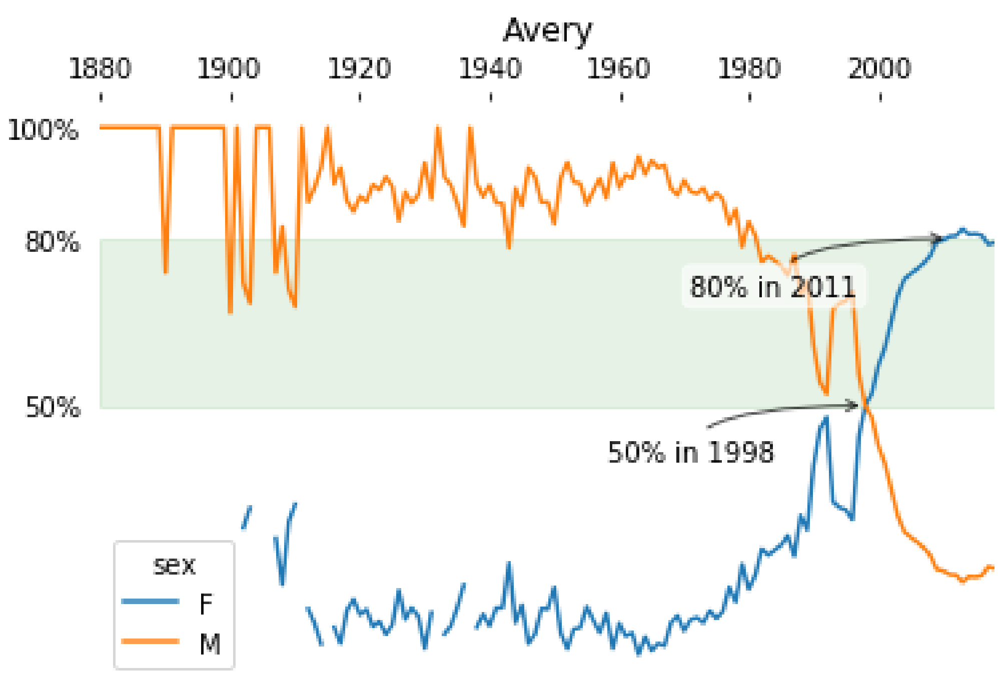
[83]:
```
def plot_name(name):
    this_name = names[names['name'] == name]

    # Count by year/sex
    table = this_name.pivot_table('births', index='year',
                                  columns='sex', aggfunc='sum')

    # Change count to proportion F/M in each year
    table = table.div(table.sum(axis='columns'), axis='rows')

    # Plot the proportions
    ax = table.plot(title=name, label='')

    # Format the graph to help analyze the hypothesis
    # 1. Mark the 50% and 80% levels we are using in the hypothesis
    ax.axhspan(0.5, 0.8, alpha=0.1, color='green')
    # 2. Show only those labels to draw even more attention to them
    #    And remove the tick marks from those label to clean up a bit
    plt.yticks([0.5, 0.8, 1.0], ['50%', '80%', '100%'])
    plt.tick_params(left=False)
    # 3. Remove the boxes (noise, most of the time)
    #    Leave bottom line to "ground" the graph
```

```python
    ax.spines['right'].set_visible(False)
    ax.spines['left'].set_visible(False)
    ax.spines['top'].set_visible(False)
    # 4. Move years to the top, remove obvious "years" label
    #    The eyes now hit the year more quickly, making the
    #    purpose of the x axis clearer from the start
    ax.xaxis.set_ticks_position('top')
    ax.xaxis.set_label_text('')

    # Point to the years when the thresholds were crossed
    # (we may not have the upper threshold in some cases)
    def draw_arrow(pct):
        crossed_pct = table[table['F'] >= pct]
        if (len(crossed_pct) > 0):
            year = crossed_pct.index[0]
            ap = dict(arrowstyle='->', connectionstyle='angle3',
                      alpha=0.7)
            bbox = dict(boxstyle='round', fc='white', ec='white',
                        alpha=0.6)
            ax.annotate('{:.0f}% in {}'.format(pct*100, year),
                        xy=(year, pct), xytext=(year-40, pct-0.1),
                        arrowprops=ap, bbox=bbox)

    draw_arrow(0.5)
    draw_arrow(0.8)

for name in ('Ashley', 'Sidney', 'Aubrey', 'Avery'):
    plot_name(name)
```


Ashley
1880
1900
1920
1940
1960
1980
2000
100%
80%
50%
80% in 1971
50% in 1964
sex
F
M


Sidney
1880
1900
1920
1940
1960
1980
2000
100%
80%
50%
50% in 1993
sex
F
M





Conclusion: H2 is false. We found one example, Sidney, where the name was not adopted by over 80% of girls (although it is still in the thirty-year window, it is unlikely it will revert the trend shown in the graph). However, even with Sidney not quite following the same pattern, we can say that H2 is a good predictor for a boy name becoming a girl name in a relatively short amount of time, once it is used as a girl name by half of the births.

9.5 Conclusions (25 points)

Write your conclusions and make sure to address the issues below: - What have you learned from this assignment? - Which parts were the most fun, time-consuming, enlightening, tedious? - What would you do if you had an additional week to work on this?

9.6 Solution

9.6.1 What have you learned from this assignment?

- `pivot_table` - before this assignment, I used `groupby` for these types of problems. Now I have a better understanding of pivot tables.
- `query` - before this assignment, I used traditional filtering. `query()` is cleaner, thus easier to follow and to maintain.
- Got a bit better in cleaning up graphs (removing boxes, making grids less prominent, etc.). Used in one example so far (the movie genres horizontal bar graph), but getting more confident in the APIs to try in other graphs in the future.

9.6.2 Which parts were the most fun, time-consuming, enlightening, tedious?

Fun:

- Exploring data with graphs continue to be fun :)
- Learning how to customize graphs also continues to be fun and educational

Enlightening:

- The power of `pivot_table`
- The cleaness of `query`
- Defining "diversity" is harder than it looks

Tedious:

- None

9.6.3 What would you do if you had an additional week to work on this?

- Investigate when `query()` is slower than traditional filtering. The textbook has some general statements, but no specific guidelines.
- Try `pivot_table` even more. I struggle to define what should be the main variable, the index and the columns in a few cases. I would like for that to come more naturally to me, i.e. first visualize I want to get done, then effortlessly translate that into the different pieces of the `pivot_table` API.