MS1_FINAL

April 5, 2017

1 CS109b Final Project

2 Milestone1

by Danqing Wang, Wenshan Zheng, Zecai Liang ______________

```
In [7]: # import libraries
        import numpy as np
        import pandas as pd
        import matplotlib
        import matplotlib.pyplot as plt
        import random

        %matplotlib inline

        import warnings
        warnings.filterwarnings('ignore')
        import urllib
        from bs4 import BeautifulSoup
        import time
        import re
        import tmdbsimple as tmdb
        tmdb.API_KEY = '4074d0170761c40d9c07d9016ddd4965'
        from collections import Counter

        from imdb import IMDb
        ia = IMDb()
```

2.1 1. Data Extraction

2.1.1 1.1 Accessing TMDB and IMDB Data

1.1.1 API code to access the genre and movie poster your favorite movie from TMDB

```
In [2]: # set up some basic url link strings, to be used later
        APIKeyZ = "api_key=4074d0170761c40d9c07d9016ddd4965"
        base_url_search = "https://api.themoviedb.org/3/discover/movie?"
        popular_desc = "&sort_by=popularity.desc"
```

```
        year =  "&primary_release_year={}"
        page_number = '&page={}'
        query_url = 'https://api.themoviedb.org/3/movie/{}?'
        poster_url = "http://image.tmdb.org/t/p/{size}/{path}"
        poster_size = ["w92", "w154", "w185", "w342", "w500", "w780", "original"]
```

```
In [3]: # use Beauty and Beast (released in 2017) as an example
        movie_name = "beauty and the beast"
        # set up the link to search for the movie
        movie_url = base_url_search + APIKeyZ + "&query=" + movie_name + year.forma

        # take out the TMDB ID and extract details about the movie
        page = urllib.urlopen(movie_url).read()
        soup = BeautifulSoup(page, "lxml")
        prettified = soup.prettify()
        id_index = prettified.find('"id"') #find the index for TMDB ID
        # Extract ID information
        j_beginning = 5
        movie_id = ''
        while (prettified[id_index + j_beginning].isdigit()):
            movie_id += str(prettified[id_index + j_beginning])
            j_beginning += 1
        print "Movie ID is:", movie_id
```

```
Movie ID is: 321612
```

```
In [4]: # search movie by ID and access genres
        movie = tmdb.Movies(movie_id)
        response = movie.info()
        movie.genres # seems that it returns genre id and corresponding genre
```

```
Out[4]: [{u'id': 14, u'name': u'Fantasy'},
         {u'id': 10402, u'name': u'Music'},
         {u'id': 10749, u'name': u'Romance'}]
```

```
In [5]: # download movie poster
        f = open('Beauty_and_Beast.jpg','wb')
        f.write(urllib.urlopen(poster_url.format(size = poster_size[6], path = str
        f.close()
        poster_url.format(size = poster_size[6], path = str(movie.poster_path))
```

```
Out[5]: 'http://image.tmdb.org/t/p/original//tWqifoYuwLETmmasnGHO7xBjEtt.jpg'
```

1.1.2 Extract genre and other information for this movie from IMDb

```
In [6]: # As am example, we search for the movie Beauty and the Beast
        s_result = ia.search_movie('Beauty and the Beast')
```

```
# Select the top result, Beauty and the Beast (2017)
the_unt = s_result[0]
ia.update(the_unt)
print the_unt.keys() # all possible keys
```

```
[u'music department', 'sound crew', 'camera and electrical department', u'distribut
```

```
In [7]: the_unt['title']
```

```
Out[7]: u'Beauty and the Beast'
```

```
In [8]: the_unt['year']
```

```
Out[8]: 2017
```

```
In [9]: the_unt['genres']
```

```
Out[9]: [u'Family', u'Fantasy', u'Musical', u'Romance']
```

The genres labeled by IMDb for `Beauty and the Beast` are `family`, `Fantasy`, `Musical`, and `Romance`.

1.1.3 A list of the 10 most popular movies of 2016 from TMDb, and their genre obtained via the API, and confirm if the genre is consistent with IMDB data

TMDb

```
In [10]: # first extract TMDB ID of each movie, then use package tmdbsimple to obta
         popular_movies_2016 =  base_url_search + APIKeyZ + popular_desc + year.for
         page = urllib.urlopen(popular_movies_2016).read()
         soup = BeautifulSoup(page, "lxml")
         prettified = soup.prettify()
         movie_list = [m.start() for m in re.finditer('"id"', prettified)] # this i
         movie_id_list = []
         for i in range(10):
             i_beginning = 5
             movie_id_temp = ''
             while (prettified[movie_list[i] + i_beginning].isdigit()):
                 movie_id_temp += str(prettified[movie_list[i] + i_beginning])
                 i_beginning += 1
             movie_id_list += [int(movie_id_temp)]
```

```
In [11]: movie_data = [] # to store movie information
         for i in range(len(movie_id_list)):
             movie = tmdb.Movies(movie_id_list[i])
             response = movie.info()
             movie_data += [response]
```

```
In [12]: # Extract IMDB ID
         IMDB_ID = []
         for i in range(len(movie_data)):
             IMDB_ID += [int(str(movie_data[i]['imdb_id'])[2:])]
```

```
In [13]: TMDB_genre_list = [] # to store genre information from TMDB
         for i in range(len(movie_data)):
             genre_temp = []
             for k in range(len(movie_data[i]['genres'])):
                 genre_temp += [str((movie_data[i]['genres'][k]['name']))]
             TMDB_genre_list += [genre_temp]
         TMDB_genre_list
```

```
Out[13]: [['Animation', 'Comedy', 'Drama', 'Family', 'Music'],
          ['Adventure', 'Action', 'Fantasy'],
          ['Adventure', 'Animation', 'Comedy', 'Family'],
          ['Action', 'Adventure', 'Comedy', 'Romance'],
          ['Action', 'Drama', 'Science Fiction', 'War'],
          ['Action', 'Adventure', 'Fantasy', 'Science Fiction'],
          ['Drama', 'Science Fiction'],
          ['Action', 'Science Fiction'],
          ['Action', 'Horror'],
          ['Drama']]
```

IMDb

```
In [14]: ## Store genre information into dataframe
         IMDb_genre_list = []
         title = []
         for i in range(0, 10):
             movie = ia.get_movie(IMDB_ID[i]) # grab movie data by id
             ia.update(movie)
             title += [str(movie['title'])]
             genre_temp = []
             for j in range(len(movie['genres'])):
                 genre_temp += [str(movie['genres'][j])]
             IMDb_genre_list += [sorted(genre_temp)]
```

```
In [15]: IMDb_genre_list
```

```
Out[15]: [['Animation', 'Comedy', 'Family', 'Music'],
          ['Adventure', 'Family', 'Fantasy'],
          ['Adventure', 'Animation', 'Comedy', 'Family'],
          ['Action', 'Adventure', 'Comedy', 'Romance', 'Sci-Fi'],
          ['Action', 'Adventure', 'Sci-Fi'],
          ['Action', 'Adventure', 'Fantasy', 'Sci-Fi'],
          ['Drama', 'Mystery', 'Sci-Fi', 'Thriller'],
          ['Action', 'Adventure', 'Sci-Fi'],
          ['Action', 'Horror'],
          ['Biography', 'Drama']]
```

Comparing TMDb and IMDb genre labels:

```
In [16]: ## top ten  movies titles:
         title

Out[16]: ['Sing',
          'Fantastic Beasts and Where to Find Them',
          'Finding Dory',
          'Deadpool',
          'Rogue One',
          'Doctor Strange',
          'Arrival',
          'Captain America: Civil War',
          'Underworld: Blood Wars',
          'Lion']
```

Comparing the two, we see that TMDb and IMDb labelled genres generally agree with each other. Their diffenrences/similarities fall into one of the three situations below:

1. **The TMDb genres and IMDb genres agree completely with each other**: Movies 2, 3, 9. For example, for the move 'Finding Dory', both databases labelled it as 'Adventure', 'Animation', 'Comedy', and 'Family'.

2. **Genres labelled by one database is a subset of that labeled by the other**: Movies 1, 4, 8, 10. For example, for the movie Sing (movie #1), TMDb labelled it as 'Animation', 'Comedy', 'Drama', 'Family', 'Music', while IMDb did not lable it as 'Drama'.

3. **There is a large intersection between the two sets of genre labels labelled by each database**: Movies 5, 6, 7.
   For example, for the movie Rogue One (movie #5), TMDb labelled it as 'Action', 'Drama', 'Science Fiction', 'War', while IMDb labeled it as 'Action', 'Adventure', 'Sci-Fi'. The common label between the two sets are 'Action' and 'Sci-Fi'.

We also note here that TMDb labels science fiction movies as "Science Fiction', while IMDb labels them as"Sci-Fi".

These differences need to be resolved as we combine the two databases into one. We will decide how to recouncil the differences in genre labeling in the two data bases after our exploratory data analysis.

2.1.2 1.2 Larger Scale: Extract Top 500 Movie

We import the list of top 500 movies from TMDb and extract infomation for these movies from the IMDb database. We use this data in our exploratory data analysis in section 2.

1.2.1 Extract Top 500 Movie from TMDB

In [17]:
```
# get top 500 popular movies ID, to be used for further search of IMDB ID
number_page = 0
movie_id_list = []
for i in range(1, 26):
    popular_movies =  base_url_search + APIKeyZ + popular_desc + page_numb
    page = urllib.urlopen(popular_movies).read()
    soup = BeautifulSoup(page, "lxml")
    prettified = soup.prettify()
    movie_list = [m.start() for m in re.finditer('"id"', prettified)] # th

    for j in range(len(movie_list)):
        j_beginning = 5
        movie_id_temp = ''
        while (prettified[movie_list[j] + j_beginning].isdigit()):
            movie_id_temp += str(prettified[movie_list[j] + j_beginning])
            j_beginning += 1
        movie_id_list += [int(movie_id_temp)]
    if i % 40 == 0:
        time.sleep(10)
```

In [18]:
```
# query each movie by TMDB ID and get IMDB ID
movie_id_list_IMDB = []
for i in range(len(movie_id_list)):
    page_url = query_url.format(movie_id_list[i]) + APIKeyZ
    page = urllib.urlopen(page_url).read()
    soup = BeautifulSoup(page, "lxml")
    prettified = soup.prettify()
    imdb_id_index = prettified.find('"imdb_id"')
    j_beginning = 13
    movie_id_temp = ''
    while (prettified[prettified.find('"imdb_id"') + j_beginning].isdigit
        movie_id_temp += str(prettified[prettified.find('"imdb_id"') + j_b
        j_beginning += 1
    movie_id_list_IMDB += [movie_id_temp]
    if i % 40 == 39:
        time.sleep(10.1)
```

In [19]:
```
# Write out movie TMDB IDs
thefile = open('popular_tmdb_id.txt', 'w')
for item in movie_id_list:
    thefile.write("%s\n" % item)
thefile.close()
```

In [20]:
```
# Write out movie IMDB IDs
thefile = open('popular_imdb_id.txt', 'w')
for item in movie_id_list_IMDB:
```

```
        thefile.write("%s\n" % item)
    thefile.close()
```

In [21]:
```
# search one movie by ID and create list of column names
movie = tmdb.Movies(movie_id_list[0])
response = movie.info()
columns = []
for i in range(len(response)):
    column_temp = [str(response.items()[i][0])]
    columns += column_temp
```

In [22]:
```
# create a dataframe to store information of top 500 popular movies
index = range(0, len(movie_id_list))
df = pd.DataFrame(index = index, columns=columns)
df = df.fillna(0)
```

In [23]:
```
# store information into dataframe, some information received is stroed as
time.sleep(11)
for i in range(len(index)):
    movie = tmdb.Movies(movie_id_list[i])
    response = movie.info()
    if i % 40 == 39:
        time.sleep(11)
    for j in range(0, len(columns)):
        if j == 1:
            country_temp = []
            for k in range(len(response[columns[j]])):
                country_temp += [(response[columns[j]][k]['name'])]
            df.iloc[i,j] = str(country_temp)
        elif j == 6:
            genre_temp = []
            for k in range(len(response[columns[j]])):
                genre_temp += [(response[columns[j]][k]['name'])]
            df.iloc[i,j] = str(genre_temp)
        elif j == 11:
            if response[columns[j]] != None:
                df.iloc[i,j] = str(response[columns[j]])
        elif j == 14:
            language_temp = []
            for k in range(len(response[columns[j]])):
                language_temp += [(response[columns[j]][k]['iso_639_1'])]
            df.iloc[i,j] = str(language_temp)
        elif j == 18:
            company_temp = []
            for k in range(len(response[columns[j]])):
                company_temp += [(response[columns[j]][k]['name'])]
            df.iloc[i,j] = str(company_temp)
        else:
```

```
            df.iloc[i,j] = response[columns[j]]
df.head()
```

Out[23]:

```
                        poster_path           production_countries    reve
0  /tWqifoYuwLETmmasnGHO7xBjEtt.jpg  [u'United States of America']  89997
1  /45Y1G5FEgttPAwjTYic6czC9xCn.jpg  [u'United States of America']  58606
2  /s9ye87pvq2IaDvjv9x4IOXVjvA7.jpg  [u'United States of America']  60130
3  /5wBbdNb0NdGiZQJYoKHRv6VbiOr.jpg  [u'United States of America']  47962
4  /myRzRzCxdfUWjkJWgpHHZ1oGkJd.jpg  [u'United States of America']   6010

                                            overview  video      id  \
0  A live-action adaptation of Disney's version o...  False  321612
1  In the near future, a weary Logan cares for an...  False  263115
2  A koala named Buster recruits his best friend ...  False  335797
3  Explore the mysterious and dangerous home of t...  False  293167
4  In the near future, Major is the first of her ...  False  315837

                                              genres                titl
0                [u'Fantasy', u'Music', u'Romance']  Beauty and the Beas
1         [u'Action', u'Drama', u'Science Fiction']                Loga
2  [u'Animation', u'Comedy', u'Drama', u'Family',...                 Sin
3  [u'Science Fiction', u'Action', u'Adventure', ...    Kong: Skull Islan
4         [u'Action', u'Drama', u'Science Fiction']    Ghost in the Shel

                tagline  vote_count   ...      imdb_id  adult  \
0         Be our guest.        1304   ...    tt2771200  False
1     His Time Has Come        2124   ...    tt3315342  False
2  Auditions begin 2016.       1022   ...    tt3470600  False
3     All hail the king         889   ...    tt3731562  False
4                               274   ...    tt1219827  False

                      backdrop_path  \
0  /6aUWe0GSl69wMTSWWexsorMIvwU.jpg
1  /5pAGnkFYSsFJ99ZxDIYnhQbQFXs.jpg
2   /fxDXp8un4qNY9b1dLd7SH6CKzC.jpg
3  /pGwChWiAY1bdoxL79sXmaFBlYJH.jpg
4  /lsRhmB7m36pEX0UHpkpJSE48BW5.jpg

                                production_companies release_date  popula
0     [u'Walt Disney Pictures', u'Mandeville Films']   2017-03-17  180.79
1  [u'Twentieth Century Fox Film Corporation', u"...   2017-02-28  111.85
2  [u'Universal Pictures', u'Fuji Television Netw...   2016-11-23   75.00
3      [u'Warner Bros.', u'Legendary Entertainment']   2017-03-08   56.57
4  [u'Paramount Pictures', u'DreamWorks SKG', u'G...   2017-03-29   52.86

         original_title     budget vote_average runtime
0  Beauty and the Beast  160000000          7.1     129
1                 Logan   97000000          7.6     141
```

```
2                    Sing   75000000           6.7     108
3    Kong: Skull Island  190000000           6.1     118
4    Ghost in the Shell  110000000           6.4     106

[5 rows x 25 columns]
```

```
In [24]: # write the dataframe out
         df.to_csv('TMDb_data.txt', encoding='utf-8')
```

1.2.2 Extract Data of the Same Top 500 Movie from IMDB

```
In [25]: # Import TMDb top 500 movies using imdb_ids
         id_list = pd.read_csv('popular_imdb_id.txt', header=None)
         ID = np.array(id_list)
```

```
In [26]: # Prepare an empty dataframe to record data
         # Among the different variables, we select the following variables of inte
         columns = ['title','genres', 'director', 'distributors', 'year', 'rating',
                    'language codes', 'languages', 'producer', 'mpaa', 'writer', 'to
                     'country codes', 'countries', 'cover url', 'aspect_ratio', 'pro
                     'cinematographer', 'plot outline', 'plot', 'cast', 'animation d
                     'canonical title', 'editorial department', 'canonical title', '
                     'long imdb canonical title', 'smart canonical title', 'smart lo
                     'full-size cover url']
         index = range(1, len(ID)+1)

         df = pd.DataFrame(index = index, columns=columns)
         df = df.fillna(0)
```

```
In [27]: ######################################
         ######################################
         ### DO NOT RUN THIS BLOCK OF CODES ###
         ######################################
         # Run this block of codes takes a long
         # time, only do so when IMDb_data.txt
         # is not availabe locally.
         ######################################
         ######################################


         # Fill in dataframe df
         for i in range(0, len(index)):
             movie = ia.get_movie(ID[i]) # grab movie data by id
             ia.update(movie)
             keys = movie.keys() # generate the available keys of this particular m

             for j in range(0, len(columns)):
                 if columns[j] in keys:
```

```
                if type(movie[columns[j]]) == list:
                    result = str(movie[columns[j]])
                else:
                    result = movie[columns[j]]
                df.iloc[i,j] = result
            else:
                df.iloc[i,j] = 'nan'

    # add column if IDs to the dataframe
    df['imdb_ids'] = pd.Series(ID.reshape(500,), index = df.index)

    # export dataframe to txt file
    df.to_csv('IMDb_data.txt', encoding='utf-8')
```

2.2 2. Exploratory Analysis

Next, we want to take a look the two database about the most popular 500 movies, and perform some exploratory analysis.

The data from TMDB is saved locally as "TMDB_data.txt", and the data from IMDB is saved locally as "IMDB.txt".

Here shows our visualization sketches.

2.2.1 2.1 Genre

2.1.1 Genre Information from the IMDB Website We know from the IMDB website (http://www.imdb.com) that the website categorize movies into 27 genres, as seen in the following picture:

We saved the information to a local file "IMDB_web_genre.txt", and would compare the information on the website to the two dataset we extracted from TMDB and IMDB.

```
In [28]: top500_IMDB_web = pd.read_csv('IMDB_web_genre.csv')

In [29]: top500_IMDB_web.head()

Out[29]:      Genre    Count
         0    Drama  1027694
         1  Romance   279500
         2    Short   171957
         3   Comedy   149370
         4    Crime   113705
```

2.1.2 Genre Information from the IMDB Database Next we calculate the genre information from the [top 500 movie data] (saved as "IMDB_data.txt") sampled from IMDB database.

In [8]:
```
## data frame extracted from IMDB by movieID (top-500 in TMDB)
top500_IMDB = pd.read_csv("IMDB_data.txt")

top500_IMDB.columns.values
```

Out[8]:
```
array(['Unnamed: 0', 'title', 'genres', 'director', 'distributors', 'year',
       'rating', 'votes', 'runtimes', 'language codes', 'languages',
       'producer', 'mpaa', 'writer', 'top 250 rank', 'kind',
       'country codes', 'countries', 'cover url', 'aspect_ratio',
       'production companies', 'cinematographer', 'plot outline', 'plot',
       'cast', 'animation department', 'original music', 'canonical title',
       'editorial department', 'canonical title.1', 'long imdb title',
       'long imdb canonical title', 'smart canonical title',
       'smart long imdb canonical title', 'full-size cover url', 'imdb_ids'
```

Parse genres into dummy coding

In [9]:
```
## ideally coule package this code into a function of spliting all variable
# 1. head and tail characters to delet
# 2. split by

gr = top500_IMDB.ix[:, 'genres']

## Split variable `Genres` into list of strings

for i in range(len(gr)):
        # each row in 'genres' column
    st = gr[i]
#### --------------------------- for spliting other variables, change here -
        # delete the first three character " [u' ", and the last two charac
        # split by " ' u "
    ls = st[3:-2].split("', u'")
        # return a list to the 'genres' column
    gr[i] = ls

top500_IMDB_genre = pd.DataFrame(gr)["genres"].str.join(sep='*').str.get_du
top500_IMDB_genre['imdb_id'] = top500_IMDB['imdb_ids']

top500_IMDB_genre.head()
```

Out[9]:
```
   Action  Adult  Adventure  Animation  Biography  Comedy  Crime  Documenta
0       0      0          0          0          0       0      0
1       1      0          0          0          0       0      0
2       0      0          0          1          0       1      0
3       1      0          1          0          0       0      0
4       1      0          1          0          0       0      0

   Drama  Family   ...     Musical  Mystery  Romance  Sci-Fi  Short  Sport
```

```
0       0       1    ...             1         0         1        0        0        0
1       1       0    ...             0         0         0        1        0        0
2       0       1    ...             0         0         0        0        0        0
3       0       0    ...             0         0         0        1        0        0
4       0       0    ...             0         0         0        1        0        0

   Thriller  War  Western  imdb_id
0         0    0        0  2771200
1         1    0        0  3315342
2         0    0        0  3470600
3         0    0        0  3731562
4         0    0        0   369610

[5 rows x 24 columns]
```

```
In [10]: ## count number of labels for each genre
         top500_IMDB_genre_count = top500_IMDB_genre.sum(axis = 0)[:-1]
         top500_IMDB_genre_count.head()
```

```
Out[10]: Action       210
         Adult          1
         Adventure    220
         Animation     55
         Biography     22
         dtype: int64
```

```
In [33]: ## save as local files
         top500_IMDB_genre.to_csv('IMDB_split_genre.txt') # include 'imdb_id'
         top500_IMDB_genre_count.to_csv('IMDB_split_genre_count.txt')
```

2.1.3 Genre Information from the TMDB Database Similarly we organize the genre information from the [top 500 movie data] (saved as "TMDB_data.txt") sampled from TMDB database.

```
In [34]: ## data frame extracted from TMDB by movieID
         top500_TMDB = pd.read_csv("TMDB_data.txt")

         top500_TMDB.columns.values
```

```
Out[34]: array(['Unnamed: 0', 'poster_path', 'production_countries', 'revenue',
                'overview', 'video', 'id', 'genres', 'title', 'tagline',
                'vote_count', 'homepage', 'belongs_to_collection',
                'original_language', 'status', 'spoken_languages', 'imdb_id',
                'adult', 'backdrop_path', 'production_companies', 'release_date',
                'popularity', 'original_title', 'budget', 'vote_average', 'runtime'
```

```
In [35]: ## ideally coule package this code into a function of spliting all variabl
         # 1. head and tail characters to delet
```

```
# 2. split by

gr = top500_TMDB.ix[:, 'genres']

## Split variable `Genres` into list of strings

for i in range(len(gr)):
        # each row in 'genres' column
    st = gr[i]
#### -------------------------- for spliting other variables, change here
        # delete the first three character " [u' ", and the last two chara
        # split by " ' u "
    ls = st[3:-2].split("', u'")
        # return a list to the 'genres' column
    gr[i] = ls

top500_TMDB_genre = pd.DataFrame(gr)["genres"].str.join(sep='*').str.get_d
top500_TMDB_genre['imdb_id'] = top500_IMDB['imdb_ids']

top500_TMDB_genre.head()
```

Out[35]:

```
   Action  Adventure  Animation  Comedy  Crime  Documentary  Drama  Family
0       0          0          0       0      0            0      0       0
1       1          0          0       0      0            0      1       0
2       0          0          1       1      0            0      1       1
3       1          1          0       0      0            0      0       0
4       1          0          0       0      0            0      1       0

   Fantasy  History  Horror  Music  Mystery  Romance  Science Fiction  \
0        1        0       0      1        0        1                0
1        0        0       0      0        0        0                1
2        0        0       0      1        0        0                0
3        1        0       0      0        0        0                1
4        0        0       0      0        0        0                1

   TV Movie  Thriller  War  Western  imdb_id
0         0         0    0        0  2771200
1         0         0    0        0  3315342
2         0         0    0        0  3470600
3         0         0    0        0  3731562
4         0         0    0        0   369610
```

In [36]:

```
## count number of labels for each genre
top500_TMDB_genre_count = top500_TMDB_genre.sum(axis = 0)[:-1]
top500_TMDB_genre_count.head()
```

Out[36]:

```
Action       214
Adventure    193
```

```
Animation     62
Comedy       112
Crime         66
dtype: int64
```

```
In [37]: ## save as local files
         top500_TMDB_genre.to_csv('TMDB_split_genre.txt') # include 'imdb_id'
         top500_TMDB_genre_count.to_csv('TMDB_split_genre_count.txt')
```

2.1.4 Basic statistics about genre

Q. How many genres in total? Check the differences between the three information sources: IMDB website, IMDB database, TMDB database.

```
In [38]: ## total number of genres
         print "Total Number of Genres from the IMDB website:", top500_IMDB_web.sha
         print "Total Number of Genres from the IMDB database:" , (top500_IMDB_genr
         print "Total Number of Genres from the RMDB database:" , (top500_TMDB_genr
```

```
Total Number of Genres from the IMDB website: 27
Total Number of Genres from the IMDB database: 23
Total Number of Genres from the RMDB database: 19
```

```
In [39]: set1 = top500_IMDB_web.loc[:,"Genre"].values
         set2 = top500_IMDB_genre.columns.values[:-1]
         set3 = top500_TMDB_genre.columns.values[:-1]

         print "Baseline Genres from TMDB database:", "\n", set3, "\n"

         print "Extra Gerens from IMDB database:", "\n", set(set2) - set(set3), "\n

         print "Extra Gerens from IMDB website:", "\n", set(set1) - set(set2), "\n'
```

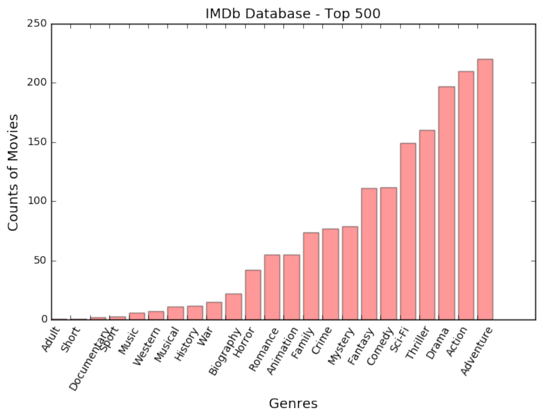
```
Baseline Genres from TMDB database:
['Action' 'Adventure' 'Animation' 'Comedy' 'Crime' 'Documentary' 'Drama'
 'Family' 'Fantasy' 'History' 'Horror' 'Music' 'Mystery' 'Romance'
 'Science Fiction' 'TV Movie' 'Thriller' 'War' 'Western']

Extra Gerens from IMDB database:
set(['Short', 'Sci-Fi', 'Adult', 'Sport', 'Musical', 'Biography'])

Extra Gerens from IMDB website:
set(['News', 'Game-Show', 'Reality-TV', 'Film-Noir', 'Talk-Show'])
```

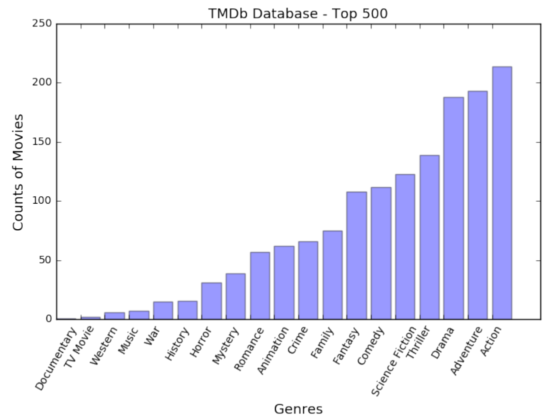
From previous analysis, we already know that the IMDB genre label is not always different from the TMDB label.

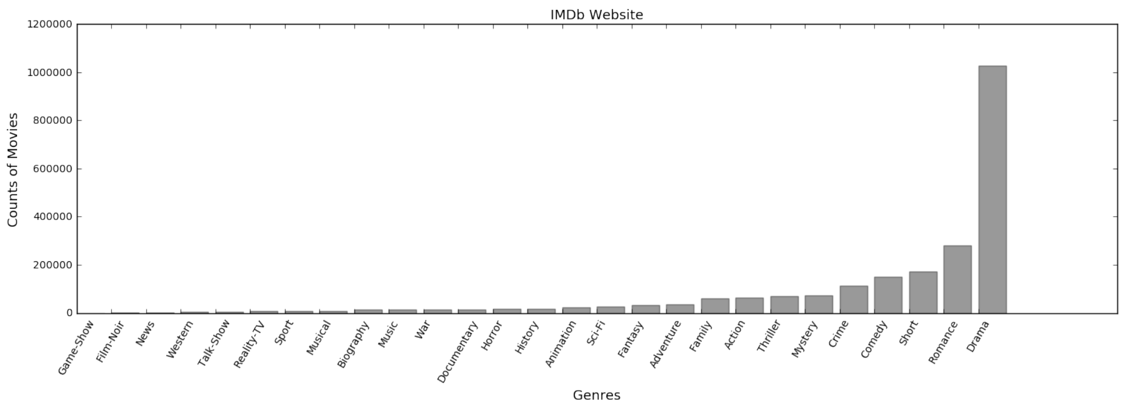
Here we see that the IMDB databse have six more genre types than TMDB. It remains to be decided how we would combine the genre labels from the two database.

Q. How many movies for each genre? Check the differences between the three information sources: IMDB website, IMDB database, TMDB database.

```
In [40]: plt.figure(figsize = (18,5))

         # IMDB
         plt.subplot(1,2,1)
         top500_IMDB_genre_count.sort()
         plt.bar(np.arange(0,top500_IMDB_genre_count.shape[0]),
                 top500_IMDB_genre_count.values, color = "red", alpha = 0.4)
         plt.xticks(np.arange(0,top500_IMDB_genre_count.shape[0]),
                    top500_IMDB_genre_count.index, rotation = 60)
         plt.xlabel("Genres", fontsize = 13)
         plt.ylabel("Counts of Movies", fontsize = 13)
         plt.title("IMDb Database - Top 500", fontsize = 13)

         # TMDB
         plt.subplot(1,2,2)
         top500_TMDB_genre_count.sort()
         plt.bar(np.arange(0,top500_TMDB_genre_count.shape[0]),
                 top500_TMDB_genre_count.values, color = "blue", alpha = 0.4)
         plt.xticks(np.arange(0,top500_TMDB_genre_count.shape[0]),
                    top500_TMDB_genre_count.index, rotation = 60)
         plt.xlabel("Genres", fontsize = 13)
         plt.ylabel("Counts of Movies", fontsize = 13)
         plt.title("TMDb Database - Top 500", fontsize = 13)

         plt.show()

         # IMDB website
         plt.figure(figsize = (18,5))
         top500_IMDB_web = top500_IMDB_web.sort("Count")
         plt.bar(np.arange(0,top500_IMDB_web.shape[0]),
                 top500_IMDB_web['Count'], color = "black", alpha = 0.4)
         plt.xticks(np.arange(0,top500_IMDB_web.shape[0]),
                    top500_IMDB_web['Genre'], rotation = 60)
         plt.xlabel("Genres", fontsize = 13)
         plt.ylabel("Counts of Movies", fontsize = 13)
         plt.title("IMDb Website", fontsize = 13)
         plt.show()
```







We can see that the ranks and distributions of genres between IMDB and TMDB are similar, with slight differences.

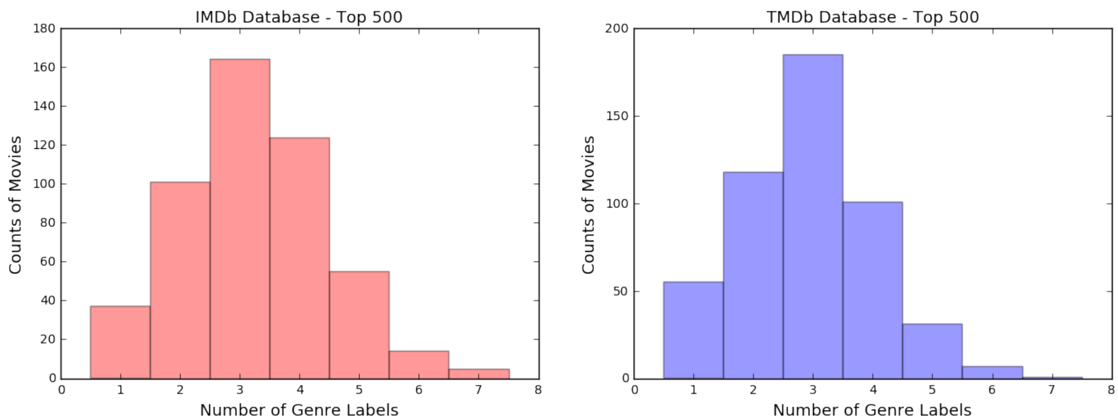
While when we look at the distribution of the IMDB website (which samples a much larger population rather than the top 500 movies only), and ranks are quite different (with `Drama` and `Romance` ranking first and second).

This illustrates how we should sample more randomely for the training data to be representative. It also points out that the data is quite unbasied between different genres, and needs to be taken into consideration when building our models.

Q. How many genre labels for each movie? Check the differences between the three information sources: IMDB website, IMDB database, TMDB database.

```
In [41]: ## number of genres for each movie, saved in variable 'n_genre'
         plt.figure(figsize = (15,5))

             # IMDB
         plt.subplot(1, 2, 1)
         plt.hist(top500_IMDB_genre.iloc[:,:-1].sum(axis = 1),
```

```
            bins = np.arange(0.5,8.5), color = "red", alpha = 0.4)
plt.xlabel("Number of Genre Labels", fontsize = 13)
plt.ylabel("Counts of Movies", fontsize = 13)
plt.title("IMDb Database - Top 500", fontsize = 13)
    # TMDB
plt.subplot(1, 2, 2)
plt.hist(top500_TMDB_genre.iloc[:,:-1].sum(axis = 1),
            bins = np.arange(0.5,8.5), color = "blue", alpha = 0.4)
plt.xlabel("Number of Genre Labels", fontsize = 13)
plt.ylabel("Counts of Movies", fontsize = 13)
plt.title("TMDb Database - Top 500", fontsize = 13)

#plt.suptitle("Distribution of Number of Genres Per Movie", fontsize = 15)

plt.show()
```

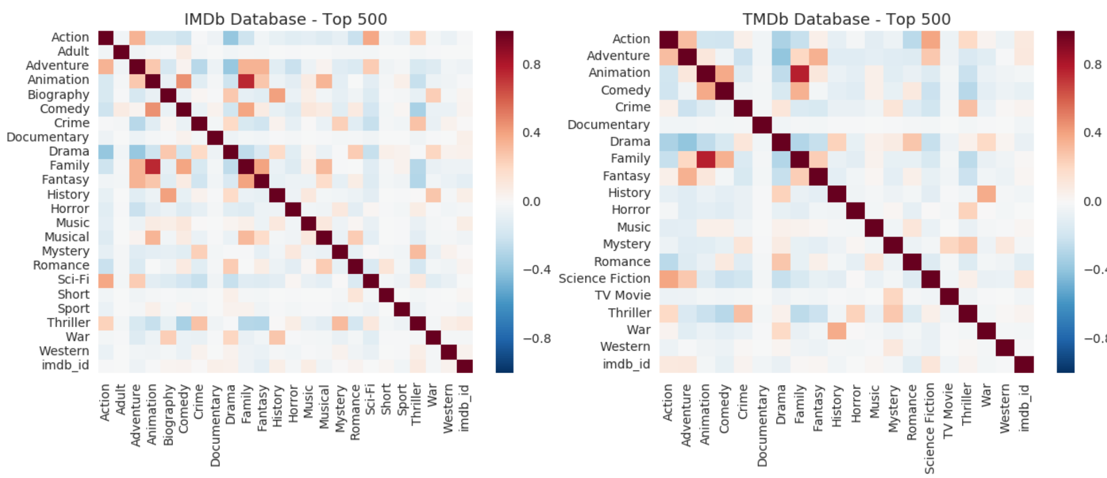


We can see that most movies have around 2 to 4 labels, and the two distributions are quite similar between IMDB and TMDB database.

2.1.5 Heatmap between genres

```
In [42]: import seaborn as sns

         ## correlation between different genres

         plt.figure(figsize = (15,5))

         # IMDB
         plt.subplot(1, 2, 1)
         corr = top500_IMDB_genre.corr(method = "pearson")
         sns.heatmap(corr,
                     xticklabels=corr.columns.values,
                     yticklabels=corr.columns.values)
```

```
plt.title("IMDb Database - Top 500", fontsize = 13)

# TMDB
plt.subplot(1, 2, 2)
corr = top500_TMDB_genre.corr(method = "pearson")
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values)
plt.title("TMDb Database - Top 500", fontsize = 13)

plt.show()
```



We can see that there are correlations between certain genres (meaning that they are likely to be assigned to the same movie). `Animation` and `Family` is stringly correlated, and then `Action` + `Sci-Fi`, `Action` + `Adventure`, `Animatino` + `Adventure`, `Adventure` + `Family`, `Adventure` + `Fantasy`, `Comedy` + `Family` et al.

For future modeling, it needs to be decided how we can cooperate the correlation information into prediction.

2.2.2 2.2 Genre vs Other Variables

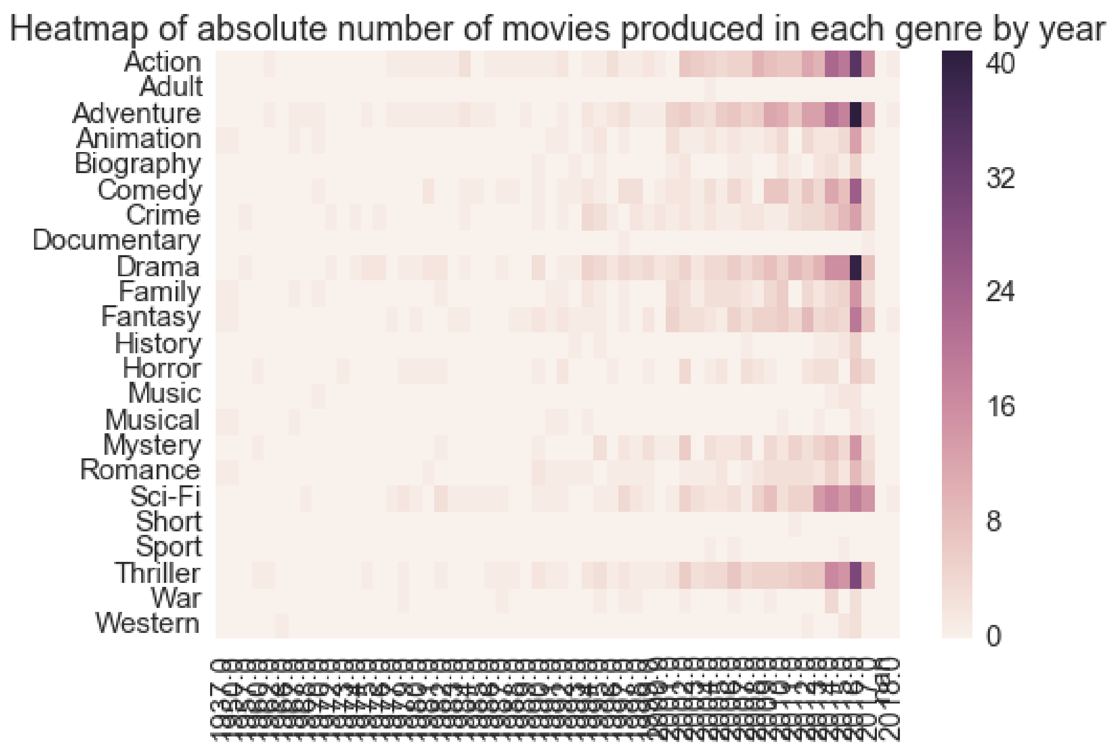
2.2.1 Quantitative:

A. Year of Release

```
In [11]: ## Extracting the year column
         top500_IMDB_year = top500_IMDB.ix[:, 'year']

         ## Create a new dataframe containing information of genre and year
         top500_IMDB_genre_year = top500_IMDB_genre.ix[:, :-1].copy()
         top500_IMDB_genre_year['year'] = pd.Series(top500_IMDB_year, index = top50
```

```
In [12]: year_list = sorted(top500_IMDB_genre_year['year'].unique())
         df_genre_year = pd.DataFrame(columns=year_list)

         # Create a dataframe with number of films for a particular genre and year
         for i in range(0, len(year_list)):
             temp = top500_IMDB_genre.ix[:, :-1][top500_IMDB_genre_year['year'] ==
             df_genre_year.iloc[:, i] = temp.sum(axis = 0)
```

```
In [13]: import seaborn as sns
         sns.set(font_scale=1.5)
         ax = plt.axes()
         sns.heatmap(df_genre_year, ax = ax)
         ax.set_title('Heatmap of absolute number of movies produced in each genre
         plt.show()
```



However, it may not always make sense to compare the absolute number of movies, since the total number of movies produced each year are different. In the following, we normalize our data by the total number of movies produced each year, so we look at percentage of each genre for each year.

```
In [14]: total_movies_by_year = np.array(df_genre_year.sum(axis = 0))
```

```
In [15]: df_genre_year_ptg = pd.DataFrame(columns=year_list)

         # Create a dataframe with number of films for a particular genre and year
```

```
for i in range(0, len(year_list)):
    temp = top500_IMDB_genre.ix[:, :-1][top500_IMDB_genre_year['year'] ==
    temp.sum(axis = 0)
    df_genre_year_ptg.iloc[:, i] = temp.sum(axis = 0) / total_movies_by_ye
```

```
In [16]: sns.set(font_scale=1.5)
         ax = plt.axes()
         sns.heatmap(df_genre_year_ptg, ax = ax)
         ax.set_title('Heatmap of percentage of movies produced in each genre by ye
         plt.show()
```

From the heatmap, we can tell that in the early 1900s, Horror films and Western films are produced in larger number compared to other genres, but are produce less in recent years. Action, Adventure, Fantasy, and Sci-Fi films are produced in larger numbers in recent years compared to other genres.

B. Rating

```
In [17]: ## Extracting the year column
         top500_IMDB_rating = top500_IMDB.ix[:, 'rating']

         ## Create a new dataframe containing information of genre and rating
         top500_IMDB_genre_rating = top500_IMDB_genre.ix[:, :-1].copy()
         top500_IMDB_genre_rating['rating'] = pd.Series(top500_IMDB_rating, index =
```

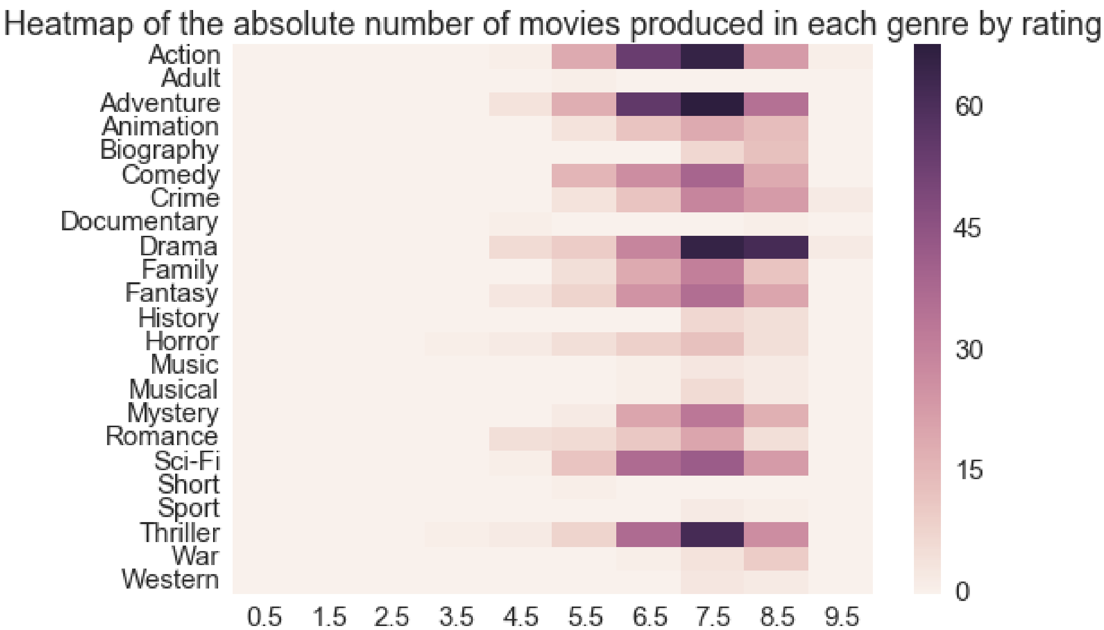
```
In [18]: rating_list = sorted(top500_IMDB_genre_rating['rating'].unique())
         min(rating_list), max(rating_list)
```

```
Out[18]: (3.8999999999999999, 9.3000000000000007)
```

```
In [19]: rating_list = [0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5]
```

```
In [20]: df_genre_rating = pd.DataFrame(columns=rating_list)

         # Create a dataframe with number of films for a particular genre and year
         for i in range(0, len(rating_list)):
             temp = top500_IMDB_genre.ix[:, :-1][top500_IMDB_genre_rating['rating']
             temp.sum(axis = 0)
             df_genre_rating.iloc[:, i] = temp.sum(axis = 0)
```

```
In [22]: ax = plt.axes()
         sns.heatmap(df_genre_rating, ax = ax)
         ax.set_title('Heatmap of the absolute number of movies produced in each ge
         plt.show()
```

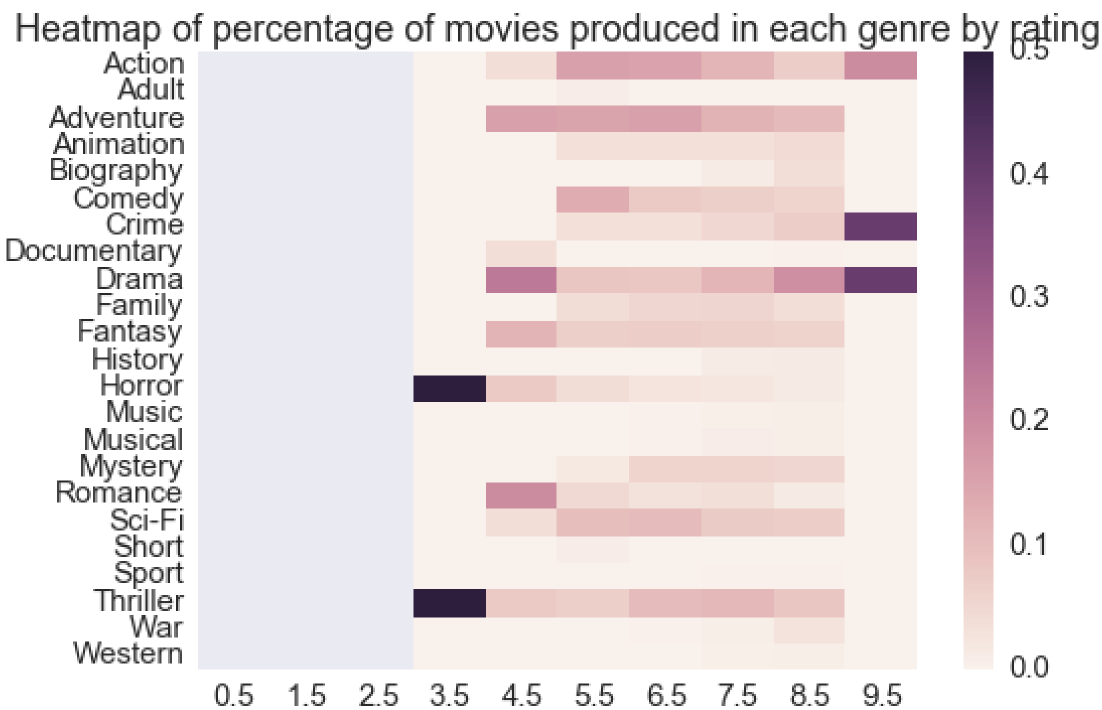


However, it may not always make sense to compare the absolute number of movies, since the total number of movies under each rating range are different. In the following, we normalize our data by the total number of movies under each rating range, so we look at percentage of each genre for rating range.

```
In [23]: total_movies_by_rating = np.array(df_genre_rating.sum(axis = 0))
```

In [24]:
```
df_genre_rating_ptg = pd.DataFrame(columns=rating_list)

# Create a dataframe with number of films for a particular genre and year
for i in range(0, len(rating_list)):
    temp = top500_IMDB_genre.ix[:, :-1][top500_IMDB_genre_rating['rating']
    temp.sum(axis = 0)
    df_genre_rating_ptg.iloc[:, i] = temp.sum(axis = 0) / total_movies_by_
```

In [25]:
```
sns.set(font_scale=1.5)
ax = plt.axes()
sns.heatmap(df_genre_rating_ptg, ax = ax)
ax.set_title('Heatmap of percentage of movies produced in each genre by ra
plt.show()
```



From the heatmap, we see that `Horror` and `Thriller` movies most often have the lowest ratings, while `Crime` and `Drama` movies most often have the highest ratings. Among films that have ratings between 5 and 9, there are more `Action` and `Adventure` movies than other genres.

C. Votes

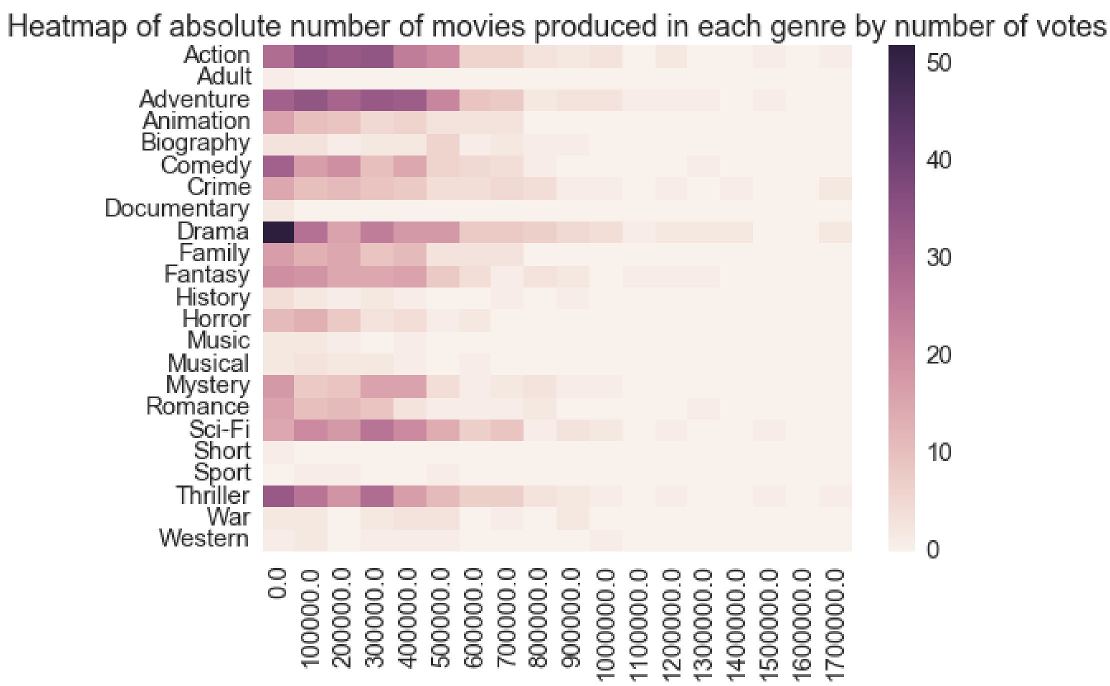
In [26]:
```
## Extracting the year column
top500_IMDB_votes = top500_IMDB.ix[:, 'votes']

## Create a new dataframe containing information of genre and rating
top500_IMDB_genre_votes = top500_IMDB_genre.ix[:, :-1].copy()
top500_IMDB_genre_votes['votes'] = pd.Series(top500_IMDB_votes, index = to
```

In [27]:
```
top500_IMDB_genre_votes = top500_IMDB_genre_votes[pd.notnull(top500_IMDB_v
```

In [28]:
```
votes_list = np.linspace(0, 1800000, 19)
```

In [29]:
```
df_genre_votes = pd.DataFrame(columns=votes_list[:-1])

# Create a dataframe with number of films for a particular genre and year
for i in range(0, len(votes_list)-1):
    temp = top500_IMDB_genre_votes.ix[:, :-1][top500_IMDB_genre_votes['vot
    temp.sum(axis = 0)
    df_genre_votes.iloc[:, i] = temp.sum(axis = 0)
```

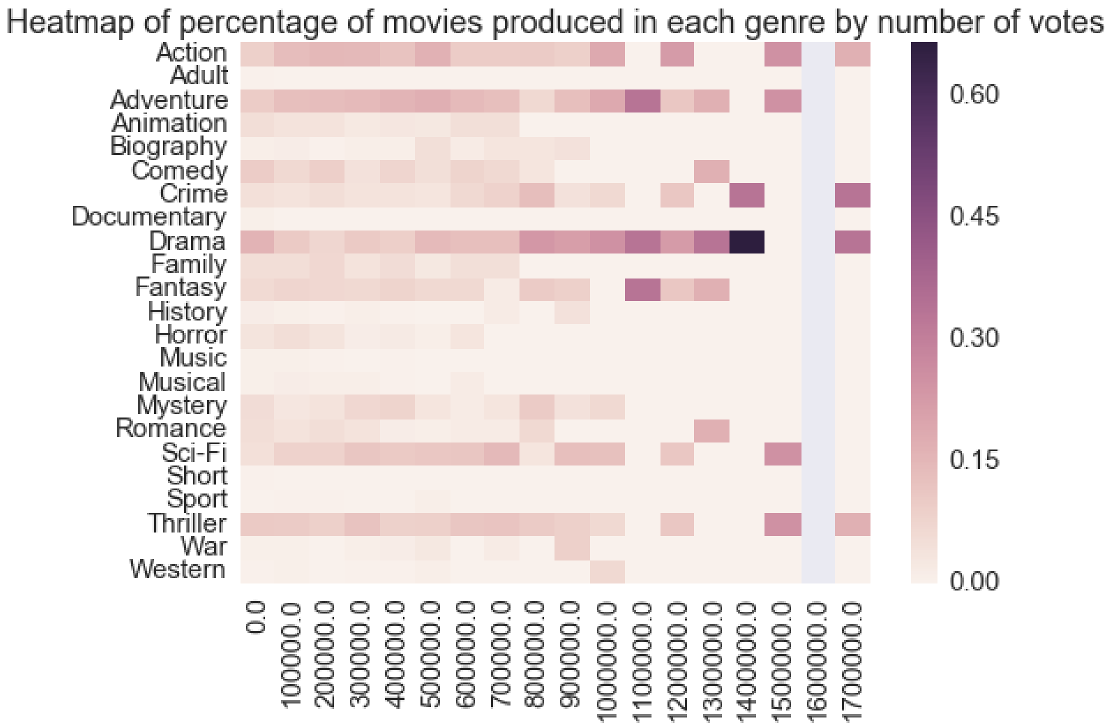
In [30]:
```
ax = plt.axes()
sns.heatmap(df_genre_votes, ax = ax)
ax.set_title('Heatmap of absolute number of movies produced in each genre
plt.show()
```



In [31]:
```
total_movies_by_votes = np.array(df_genre_votes.sum(axis = 0))
```

In [32]:
```
df_genre_votes_ptg = pd.DataFrame(columns=votes_list[:-1])

# Create a dataframe with number of films for a particular genre and year
for i in range(0, len(votes_list)-1):
    temp = top500_IMDB_genre_votes.ix[:, :-1][top500_IMDB_genre_votes['vot
    temp.sum(axis = 0)
    df_genre_votes_ptg.iloc[:, i] = temp.sum(axis = 0) / total_movies_by_v
```

```
In [33]: ax = plt.axes()
         sns.heatmap(df_genre_votes_ptg, ax = ax)
         ax.set_title('Heatmap of percentage of movies produced in each genre by nu
         plt.show()
```



It looks like the genre `Drama` has the most number of viewers giving their votes, while `Musical`, `Music`, and `Adult` movies have very few viewers giving their votes.

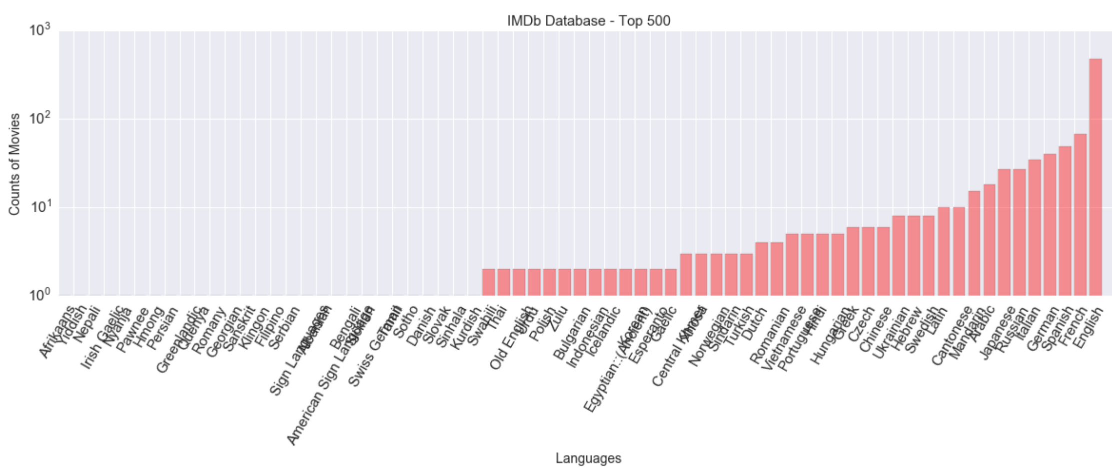
2.2.2 Qualitative:

- language
- country of release
- MAPP

```
In [34]: top500_IMDB = pd.read_csv('IMDB_data.txt')

         # combine column `language` with columns of genres, delete NA values
         df_lg = pd.concat([top500_IMDB['languages'], top500_IMDB_genre], axis = 1)
         top500_IMDB_lg = df_lg.reset_index()['languages']

         ## Split variable `Genres` into list of strings

         for i in range(len(top500_IMDB_lg)):
                 # each row in 'genres' column
             st = top500_IMDB_lg[i]
```

```
#### --------------------------- for spliting other variables, change here
        # delete the first three character " [u' ", and the last two chara
        # split by " ' u "
    ls = st[3:-2].split("', u'")
        # return a list to the 'genres' column
    top500_IMDB_lg[i] = ls

df_lg_sp = pd.DataFrame(top500_IMDB_lg)["languages"].str.join(sep='*').str
```

```
In [35]: # count the frequency of different languages
         df_lg_count = df_lg_sp.sum(axis=0)

         df_lg_count.sort()

         plt.figure(figsize = (20, 5))
         plt.bar(np.arange(0,df_lg_count.shape[0]),
                 df_lg_count.values, color = "red", alpha = 0.4, log = True)
         plt.xticks(np.arange(0,df_lg_count.shape[0]),
                    df_lg_count.index, rotation = 60)
         plt.xlabel("Languages", fontsize = 14)
         plt.ylabel("Counts of Movies", fontsize = 14)
         plt.title("IMDb Database - Top 500", fontsize = 15)

         plt.show()
```



Throught log-scale plot, we can see that majority of the movies use English, French, Spanish et al. These languages are less likely to provide information about genre type. On the other hand, the less common languages may be indicative of certain genre (although the data points are limited.)

2.2.3 Production related:

- director

- writer
- distributor (company)
- major cast

We suspect that some distributors are more likely to produce movies with specific styles thus can be tightly correlated with some kinds of genres. The same goes with directors, writers, actors/actresses and so on. Here, with production company "Walt Disney Pictures", director Martin Scorsese and actor Arnold Schwarzenegger as examples, we check if there is this kind of correlations.

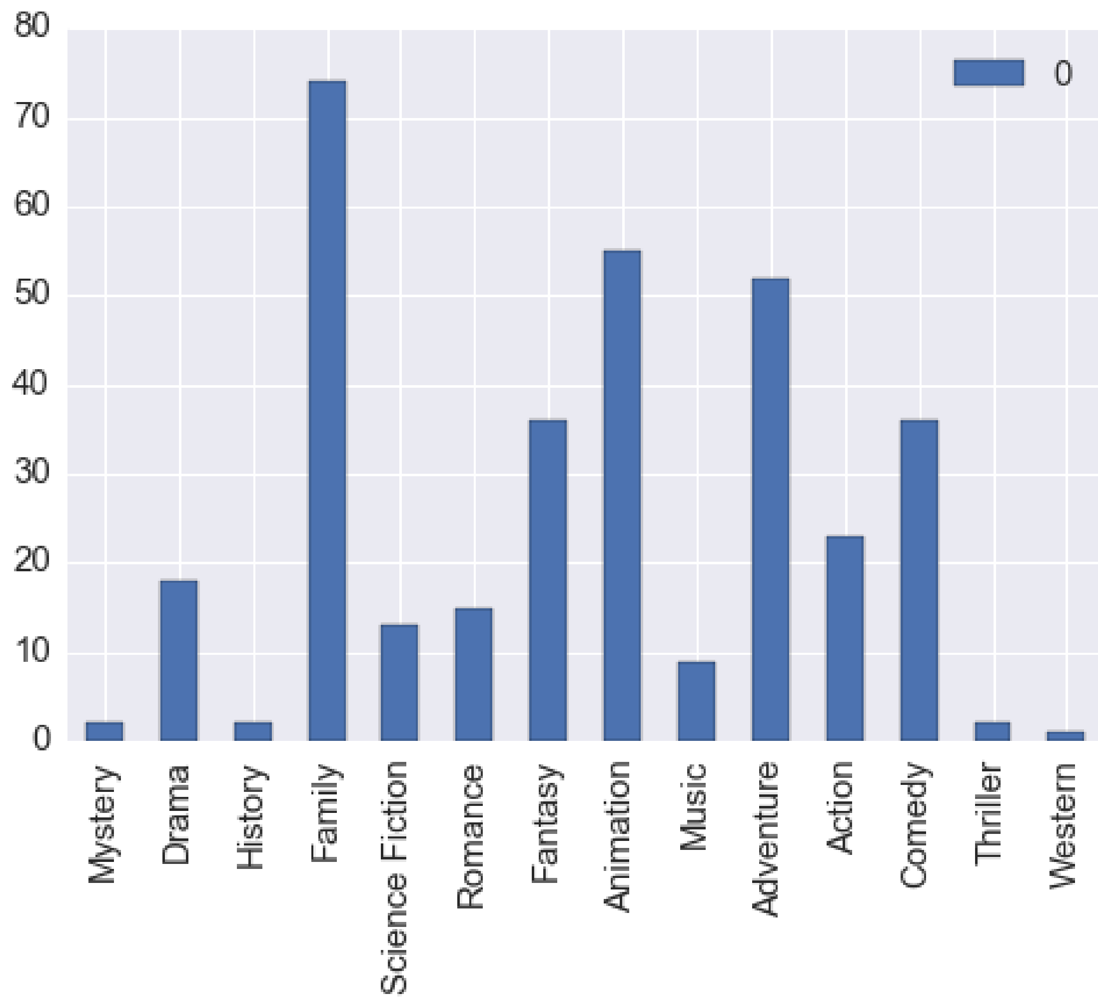
2.2.3 Genres of Walt Disney Pictures

In [36]:
```
# API request url for Walt Disney Pictures movies
company_url = base_url_search + APIKeyZ + popular_desc + "&with_companies=
```

In [37]:
```
# get top 100 popular movies IDs by Walt Disney Pictures, to be used for i
number_page = 0
WDP_movie_id_list = []
for i in range(1, 6):
    WDP_movies =  company_url + page_number.format(i)
    page = urllib.urlopen(WDP_movies).read()
    soup = BeautifulSoup(page, "lxml")
    prettified = soup.prettify()
    movie_list = [m.start() for m in re.finditer('"id"', prettified)] # th

    for j in range(len(movie_list)):
        j_beginning = 5
        movie_id_temp = ''
        while (prettified[movie_list[j] + j_beginning].isdigit()):
            movie_id_temp += str(prettified[movie_list[j] + j_beginning])
            j_beginning += 1
        WDP_movie_id_list += [int(movie_id_temp)]
    if i % 40 == 39:
        time.sleep(10)
time.sleep(10)
# store movie information
WDP_movie_data = [] # to store movie information
for i in range(len(WDP_movie_id_list)):
    movie = tmdb.Movies(WDP_movie_id_list[i])
    response = movie.info()
    WDP_movie_data += [response]
    if i % 40 == 39:
        time.sleep(11)

# store genre information of all movies
WDP_genre_list = [] # to store genre information from TMDB
for i in range(len(WDP_movie_data)):
    genre_temp = []
    for k in range(len(WDP_movie_data[i]['genres'])):
```

```
            genre_temp += [str((WDP_movie_data[i]['genres'][k]['name']))]
        WDP_genre_list += genre_temp
    WDP_genre_list[:5]

    # make histogram of genres
    genre_counts = Counter(WDP_genre_list)
    df = pd.DataFrame.from_dict(genre_counts, orient='index')
    df.plot(kind='bar')
```

```
Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0x11b4fbc90>
```

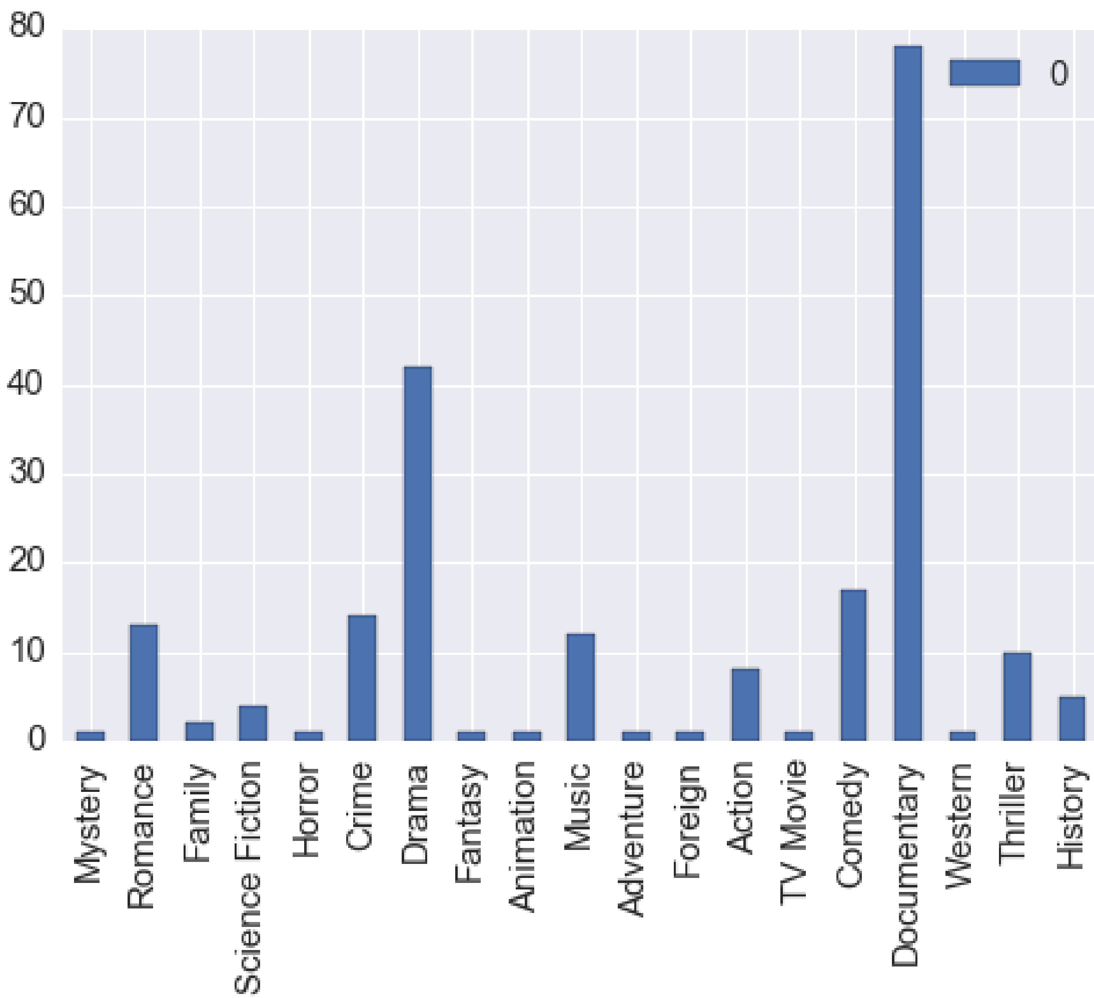


Clearly seen from above histogram, movies from Walt Disney Pictures tend to be in genres Family, Animation, Adventure and Comedy.

2.2.4 Genres of Martin Scorsese movies

```
In [38]: # API request url for Martin Scorsese movies
         director_url = base_url_search + APIKeyZ + popular_desc + "&with_people=1(
```

```
In [39]: # get all movie IDs by Martin Scorsese, to be used for further search of g
         number_page = 0
         MS_movie_id_list = []
         for i in range(1, 9):
             MS_movies =  director_url + page_number.format(i)
             page = urllib.urlopen(MS_movies).read()
             soup = BeautifulSoup(page, "lxml")
             prettified = soup.prettify()
             movie_list = [m.start() for m in re.finditer('"id"', prettified)] # th

             for j in range(len(movie_list)):
                 j_beginning = 5
                 movie_id_temp = ''
                 while (prettified[movie_list[j] + j_beginning].isdigit()):
                     movie_id_temp += str(prettified[movie_list[j] + j_beginning])
                     j_beginning += 1
                 MS_movie_id_list += [int(movie_id_temp)]
             if i % 40 == 39:
                 time.sleep(10)
         time.sleep(10)
         # store movie information
         MS_movie_data = [] # to store movie information
         MS_movie_id_list.remove(MS_movie_id_list[107]) # The 107 value returns an
         for i in range(len(MS_movie_id_list)):
             movie = tmdb.Movies(MS_movie_id_list[i])
             response = movie.info()
             MS_movie_data += [response]
             if i % 40 == 39:
                 time.sleep(11)

         # store genre information of all movies
         MS_genre_list = [] # to store genre information from TMDB
         for i in range(len(MS_movie_data)):
             genre_temp = []
             for k in range(len(MS_movie_data[i]['genres'])):
                 genre_temp += [str((MS_movie_data[i]['genres'][k]['name']))]
             MS_genre_list += genre_temp
         MS_genre_list[:5]

         # make histogram of genres
         genre_counts = Counter(MS_genre_list)
         df = pd.DataFrame.from_dict(genre_counts, orient='index')
         df.plot(kind='bar')

Out[39]: <matplotlib.axes._subplots.AxesSubplot at 0x11c1ba450>
```



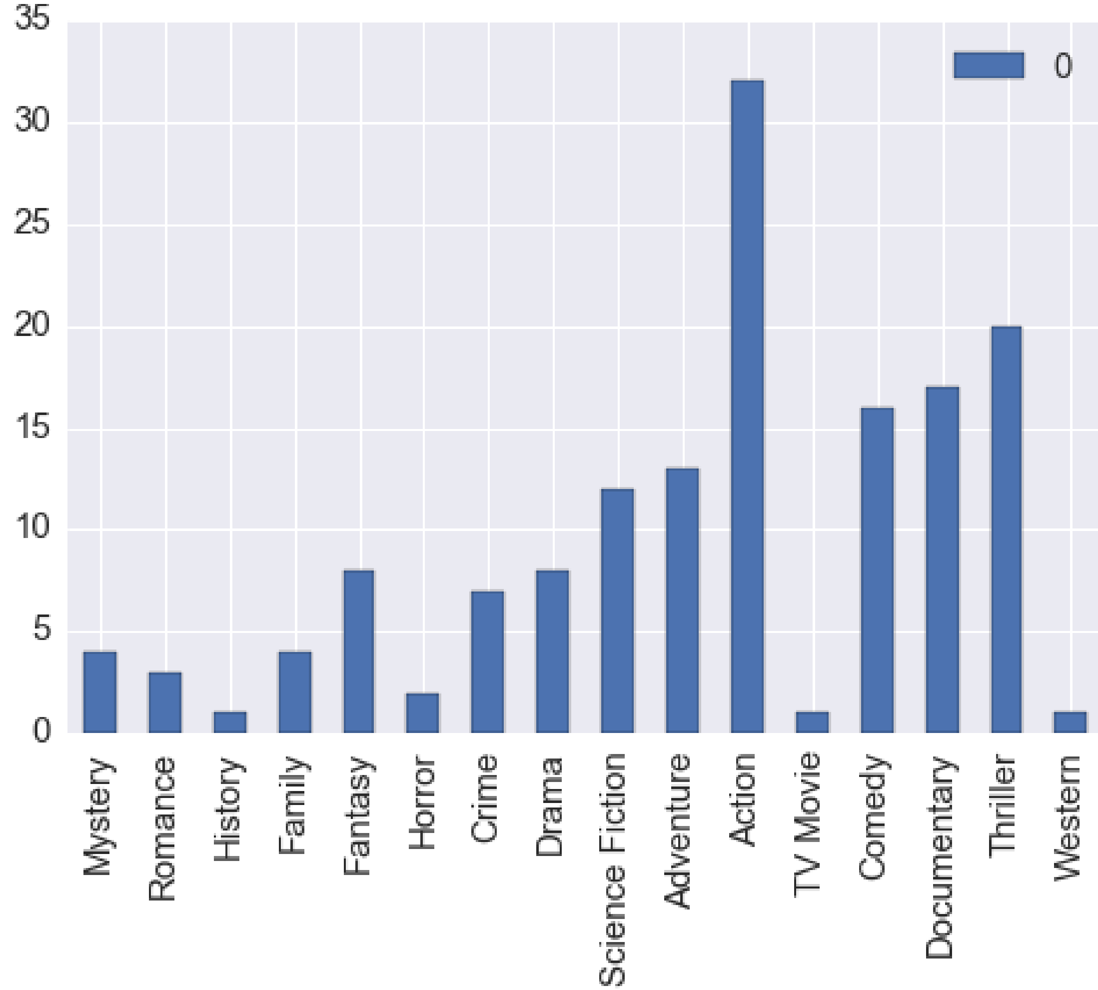
Clearly seen from above histogram, movies related to Martin Scorsese tend to be in genres Documentary and Drama.

2.2.5 Genres of Arnold Schwarzenegger movies

```
In [40]: # API request url for Arnold Schwarzenegger movies
         actor_url = base_url_search + APIKeyZ + popular_desc + "&with_people=1100'

In [41]: # get all movie IDs by Arnold Schwarzenegger, to be used for further searc
         number_page = 0
         AS_movie_id_list = []
         for i in range(1, 5):
             AS_movies =  actor_url + page_number.format(i)
             page = urllib.urlopen(AS_movies).read()
             soup = BeautifulSoup(page, "lxml")
             prettified = soup.prettify()
             movie_list = [m.start() for m in re.finditer('"id"', prettified)] # th
```

```
    for j in range(len(movie_list)):
        j_beginning = 5
        movie_id_temp = ''
        while (prettified[movie_list[j] + j_beginning].isdigit()):
            movie_id_temp += str(prettified[movie_list[j] + j_beginning])
            j_beginning += 1
        AS_movie_id_list += [int(movie_id_temp)]
    if i % 40 == 39:
        time.sleep(10)
time.sleep(10)
# store movie information
AS_movie_data = [] # to store movie information
for i in range(len(AS_movie_id_list)):
    movie = tmdb.Movies(AS_movie_id_list[i])
    response = movie.info()
    AS_movie_data += [response]
    if i % 40 == 39:
        time.sleep(11)

# store genre information of all movies
AS_genre_list = [] # to store genre information from TMDB
for i in range(len(AS_movie_data)):
    genre_temp = []
    for k in range(len(AS_movie_data[i]['genres'])):
        genre_temp += [str((AS_movie_data[i]['genres'][k]['name']))]
    AS_genre_list += genre_temp
AS_genre_list[:5]

# make histogram of genres
genre_counts = Counter(AS_genre_list)
df = pd.DataFrame.from_dict(genre_counts, orient='index')
df.plot(kind='bar')
```

```
Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x11cf0d110>
```



Clearly seen from above histogram, Arnold Schwarzenegger movies are most likely to be in genre Action, a few other genres such as Thriller, Documentary and Comedy are also common.

From above 3 examples, we found that movies from specific companies, directors and actors/actresses tend to be in one or a few genres, there is a strong correlations between the crew/production company of a movie and the genre(s) of a movie. Crew members and production companies can be good variables to predict genres of a movie.

2.2.4 Text Analysis:

- plot outline
- plot
- title
- reason for MPAA rating

We think text analysis can be very useful for predicting movie genres especially if we can choose concise text information. We will explore this part later.

2.3 3. Challenges for Next Step

How to treat the multiple genres of one movie? We clearly see from the above analysis that most movies carry multiple genres (2 to 4 for most movies). Since there are only 23 genres in total, we'd like to try to turn the problem into a multi-class classification (class = 23).

From the heatmap we do see correlations between some of the genres, which should help predicting multi-class. Although we still need to find a way to incooperate the correlatino into the model.

Another idea we could try: instead of multi-class classification of 0 and 1, we could assign probability into each genre, and select at most top 4 or 5 genres as prediction.

How to combine the data from the two database (IMDB and TMDB)? The `imdb_id` is consistent between the two database, and could be used as the key to merge the two database.

Most of the variables we would like to use exists in the IMDB database (except poster and the `production cost`). So we only need to add there two variables from TMDB database to IMDB database.

There are limited cases when the genre labes from the two databases are not exactly the same. We tend to merge the genres when it happens.

Missing information / sparse data for some genres In our analysis of the two databases, we notice that there are some missing information. Not all movies have information on all the predictors. For example, some movies do not have a plot synopsis, some movies do not have the list of distributors, etc. However, among our preditors of intersts, the amount of missing information is very little. For now, we have dropped out entires with missing information in our analysis of the top 500 movies.

However, as we download the entire database for the project, we expect to have more missing information. We also forsee that there might be sparse data for some genres, such as "Music", "Musical", "Documentary" etc., which are produced in less numbers compared to other genres.

We plan to evaluate the necessity of certain predictors in our analysis once we have the entire databse. If certain predictors have many missing values, we may consider dropping the predictor column totally. If there is only a small percentage of missing values for a certain predictor, we may consider fill in the missing values by methods such as knn, regression, mode, etc. depending on whether it is a numerical or categorical variable.

Merge/combine two database (inconsistency) We also noticed a number of inconsistencies between the two database. For instance, the genres labelled by TMDb for a particualr movie may either agree completely, is a subset or superset of, or intersects with genres labelled by IMDb. We also noticed that there are only 19 genres on TMDb but 23 and IMDb. For instance. IMDb labels adult movies as one genre, while TMDb does not label adult movies as a separate genre, but contains information of whether a movie is an adult movie in a separate column. We would need to consolidate these information and resolve any differences. Also, the IMDb database does not comtain information on the production cost of movies while TMDb has that information. We might want to combine the two in our final version of data to use.

Feature Selection Another possible problem is to select features. This can be very necessary because for each movie, APIs only return 25 and 36 variables from TMDB and IMDB, respectively. Furthermore, there are lots of other resources such as Wikipedia and forums which provide tons of information about each movie. If we use all available predictors, not only it will be very time-consuming to train prediction models but also the models will become unstable.

As of now, we tentatively separate predictors as numeric variables, categorical variables, texts and posters. To select numeric and categorical variables, we can train models with different variables and compare models to see if some variables are necessary. For texts, we think that official abstract, Wikipedia abstract, plots and highlighted comments from forums can concise and we will see if this is true. For posters, we don't have much experience yet, but most popular or official posters can be better predictors for models.

2.4 4. Potential applications with data

The first major application with all these data is to predict popularity/revenue of a movie. For example, with a specific movie style, even before the movie is made or released, we can predict whether the movie will be popular/profitable, thus can decide if a company should invest in the movie.

Similar to the first application, the second application is to engineer movie features to improve its popularity/revenue. For example, probably a movie star is strongly correlated with popularity/revenue for on kind of movies, than it might be worthy it to choose this person to perform in the movie.

The third application is to sort movies into different categories thus it can be used for a preliminary recommendation system.

Another application is movie rating. We can train a model with all features and ratings so we can rate a new movie based on its features.