Milestone1_01_explore_API

April 5, 2017

```
In [1]: import tmdbsimple as tmdb
        tmdb.API_KEY = "71e259894a515060876bab2a33d6bdc9"

In [2]: import imdb as ib
        from imdb import IMDb
        import pandas as pd
        from PIL import Image
        from StringIO import StringIO
        import requests
        import os
        import time
        from shutil import copyfile
        import types
        import numpy as np

In [3]: dir_python_notebook = os.getcwd()
        dir_movie_project = os.path.abspath(os.path.join(dir_python_notebook, os.pa
        dir_data = os.path.join(dir_movie_project, 'data')
```

API code to access the genre and movie poster path of your favorite movie

```
In [4]: #use the api to get poster image
        CONFIG_PATTERN = 'http://api.themoviedb.org/3/configuration?api_key={key}'
        url = CONFIG_PATTERN.format(key=tmdb.API_KEY)
        r = requests.get(url)
        config = r.json()
        #find the base url for posters, and available poster sizes
        base_url = config['images']['base_url']
        sizes = config['images']['poster_sizes']

In [5]: #all genre dictionary list
        search = tmdb.Search()
        favorite = search.movie(query='Forrest gump')
        print(tmdb.Genres(search.results[0]['genre_ids']).list().values())

[[{u'id': 28, u'name': u'Action'}, {u'id': 12, u'name': u'Adventure'}, {u'id': 16,
```

In [6]:
```
#create a new dictionary to match genre names
tmdb_genres = {28: 'Action', 12: 'Adventure', 16: 'Animation', 35: 'Comedy'
              10751: 'Family', 14:'Fantasy', 36: 'History', 27:'Horror', 10
              10749:'Romance', 878:'Science Fiction', 10770: 'TV Movie', 53
```

In [7]:
```
#movie_poster_genre: a function to return movie poster url and genres
#Argument:
#    favorite: string of the movie
#Return:
#    url: of the movie poster
#    genres: a list of the movie genres
def movie_poster_genre(favorite):
    #search favorite movie in TMDB
    search = tmdb.Search()
    favorite = search.movie(query = favorite)
    favo_path = search.results[0]['poster_path']
    base_url = 'http://image.tmdb.org/t/p/'
    #find the url of poster
    favo_url = "{0}{1}{2}".format(base_url, 'original', favo_path)
    genres = [tmdb_genres[x] for x in search.results[0]['genre_ids']]
    return favo_url, genres
```

In [8]:
```
favorite_movie = 'Forrest gump'
favo_url, favo_genre = movie_poster_genre(favorite_movie)
print("The genres for "+favorite_movie+ ' are:',favo_genre)
poster = requests.get(favo_url)
Image.open(StringIO(poster.content))
```

```
('The genres for Forrest gump are:', ['Comedy', 'Drama', 'Romance'])
```

Out[8]:


Tom
Hanks is
Forrest
Gump

Genre for this movie listed by TMDb and IMDb

```
In [9]: #genres for this movie are obtained from previous step:
        print("The genres from TMDB for "+favorite_movie+ ' are:',favo_genre)
```

```
('The genres from TMDB for Forrest gump are:', ['Comedy', 'Drama', 'Romance'])
```

```
In [10]: imdb = IMDb()
         favo_matchmost = imdb.search_movie(favorite_movie)[0]
         print favo_matchmost.summary()
```

```
Movie
=====
Title: Forrest Gump (1994)
```

```
In [11]: imdb.update(favo_matchmost)
         print("The genres from IMDB for "+favorite_movie+ ' are:',favo_matchmost['
```

```
('The genres from IMDB for Forrest gump are:', [u'Comedy', u'Drama', u'Romance'])
```

A list of the 10 most popular movies of 2016 from TMDb and their genre obtained via the API

```
In [12]: discover = tmdb.Discover()
         #response = discover.movie(year = 2016, sort_by = 'popularity.desc')
         response = discover.movie(primary_release_year = 2016, sort_by = 'populari
```

```
In [13]: top10_popmovie = pd.DataFrame(columns=['title','id','release_date','popula
```

```
In [14]: for ind, s in enumerate(discover.results[0:10]):
             top10_popmovie.loc[ind] =[s['title'], s['id'], s['release_date'], s['p
```

```
In [151]: top10_popmovie
```

Out[151]:

	title	id	release_date	popula
0	Sing	335797.0	2016-11-23	86.47
1	Finding Dory	127380.0	2016-06-16	46.80
2	Fantastic Beasts and Where to Find Them	259316.0	2016-11-16	45.12
3	Rogue One: A Star Wars Story	330459.0	2016-12-14	34.99
4	Arrival	329865.0	2016-11-10	34.39
5	Doctor Strange	284052.0	2016-10-25	26.27
6	Deadpool	293660.0	2016-02-09	26.06
7	Captain America: Civil War	271110.0	2016-04-27	25.89
8	Underworld: Blood Wars	346672.0	2016-11-28	23.82

```
9                                   Zootopia  269149.0   2016-02-11   21.48

                                                genre
0      [Animation, Comedy, Drama, Family, Music]
1         [Animation, Adventure, Comedy, Family]
2                    [Action, Adventure, Fantasy]
3          [Action, Drama, Science Fiction, War]
4                        [Drama, Science Fiction]
5    [Action, Adventure, Fantasy, Science Fiction]
6           [Action, Adventure, Comedy, Romance]
7                       [Action, Science Fiction]
8                                [Action, Horror]
9         [Animation, Adventure, Family, Comedy]
```

Notes: How to access movie information from IMDB, TMDB

(You can delete all the information below when submit the milestone 1, it is only used to help you get the information easier)

More information from:

TMDBSIMPLE: https://github.com/celiao/tmdbsimple

TMDB API: https://www.themoviedb.org/documentation/api

IMDBPY: http://imdbpy.sourceforge.net/

1. Search by movie names and get desired attributes:

In [257]:
```
#IMDB
imdb = IMDb()
#search by movie_name, [0] is the most closed one
imdb_movie = imdb.search_movie('Forrest Gump')[0]
#access attributes of the movie by dictionary keys
imdb.update(imdb_movie)
#available keys
print imdb_movie.keys()
#for example, get director of the movie
imdb_movie['director']
```

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

Out[257]:
```
[<Person id:0000709[http] name:_Zemeckis, Robert_>]
```

In [260]:
```
#TMDB search by movie name
search = tmdb.Search()
tmdb_movie = search.movie(query = 'Forrest Gump')
#avilable attributes
print search.results[0].keys()
#for example, get vote_average of the movie
search.results[0]['vote_average']
```

```
[u'poster_path', u'title', u'overview', u'release_date', u'popularity', u'original_
```

```
Out[260]: 8.1
```

```
In [270]: #TMDB search by movie id
          tmdb_movie_id = tmdb.Movies(603)
          movie_info = tmdb_movie_id.info()
          #information contained
          print tmdb_movie_id.info()
          #specific attribute, for example, genre
          print tmdb_movie_id.info()['genres']
```

```
{u'poster_path': u'/lZpWprJqbIFpEV5uoHfoK0KCnTW.jpg', u'production_countries': [{u'
[{u'id': 28, u'name': u'Action'}, {u'id': 878, u'name': u'Science Fiction'}]
```

```
In [282]: #TMDB search by specific interests:
          tmdb_movie_discover = tmdb.Discover()
          movie_discover = tmdb_movie_discover.movie(primary_release_year = 2016, l
          #the first three movie matching specific interest
          print tmdb_movie_discover.results[0:3]
          #information contained in the first movie
          print tmdb_movie_discover.results[0].keys()
          #get popularity of a specific movie
          tmdb_movie_discover.results[0]['popularity']
```

```
[{u'poster_path': u'/s9ye87pvq2IaDvjv9x4IOXVjvA7.jpg', u'title': u'Sing', u'overvie
[u'poster_path', u'title', u'overview', u'release_date', u'popularity', u'original_
```

```
Out[282]: 86.471049
```

2. Loop over movie id to get enough movie information

```
In [288]: #tmdb
          #latest_movieid information
          latest_r = requests.get('https://api.themoviedb.org/3/movie/latest?api_ke
          latest_r.json()['id']
```

```
Out[288]: 449744
```

```
In [351]: #just for example purpose, only select certain attributes
          movies = pd.DataFrame(columns=['title', 'release_date','popularity','genr
          for i in range(1,20):
              #skip the non-existing movie ids
              try:
                  tmdb_movies = tmdb.Movies(i)
                  movie_info = tmdb_movies.info()
                  info = tmdb_movies.info()
                  movies.loc[i] =[info['title'], info['release_date'], info['popula
                                  info['revenue'],info['imdb_id'],info['poster_pat
              except Exception:
                  continue
```

```
In [352]: movies

Out[352]:                                   title release_date  popularity  \
          2                                 Ariel   1988-10-21    0.777597
          3                   Shadows in Paradise   1986-10-16    0.372149
          5                            Four Rooms   1995-12-25    1.604237
          6                        Judgment Night   1993-10-15    0.581760
          8    Life in Loops (A Megacities RMX)   2006-01-01    0.081404
          9                      Sunday in August   2004-09-02    0.082581
          11                            Star Wars   1977-05-25    7.949018
          12                         Finding Nemo   2003-05-30    6.110192
          13                         Forrest Gump   1994-07-06    6.915513
          14                      American Beauty   1999-09-15    4.192239
          15                         Citizen Kane   1941-04-30    1.926465
          16                  Dancer in the Dark   2000-05-17    1.075538
          17                             The Dark   2006-01-26    0.343738
          18                    The Fifth Element   1997-05-07    3.341852
          19                           Metropolis   1927-01-10    1.625536

                                                           genre      revenue    imd
          2                                       [Drama, Crime]          0.0  tt009
          3                                      [Drama, Comedy]          0.0  tt009
          5                                             [Comedy]    4300000.0  tt011
          6                          [Action, Thriller, Crime]   12136938.0  tt010
          8                                       [Documentary]          0.0  tt082
          9                                              [Drama]          0.0  tt042
          11             [Adventure, Action, Science Fiction]  775398007.0  tt007
          12                              [Animation, Family]  864625978.0  tt026
          13                         [Comedy, Drama, Romance]  677945399.0  tt010
          14                                             [Drama]  356296601.0  tt016
          15                                             [Drama]   23217674.0  tt003
          16                           [Drama, Crime, Music]   40031879.0  tt016
          17                        [Horror, Thriller, Mystery]          0.0  tt041
          18   [Adventure, Fantasy, Action, Thriller, Science...  263920180.0  tt011
          19                         [Drama, Science Fiction]      650422.0  tt001

                                   poster_path
          2   /gZCJZOn4l0Zj5hAxsMbxoS6CL0u.jpg
          3   /7ad4iku8cYBuB08g9yAU7tHJik5.jpg
          5   /eQs5hh9rxrk1m4xHsIz1w11Ngqb.jpg
          6   /lNXmgUrP6h1nD53gkFh4WDzT6RZ.jpg
          8   /8YyIjOAxwzD3fZMdmJrfiApod4l.jpg
          9                               None
          11  /tvSlBzAdRE29bZe5yYWrJ2ds137.jpg
          12  /zjqInUwldOBa0q07fOyohYCWxWX.jpg
          13  /yE5d3BUhE8hCnkMUJOo1QDoOGNz.jpg
          14  /or1MP8BZIAjqWYxPdPX724ydKar.jpg
          15  /n8wfFsQ5vtm6dM8vdgXb6OLv2GY.jpg
```

```
16  /7xizDTz4Yj4IYm2ud4f6EfEXe5H.jpg
17  /8fzjzQhLXl1afshhsE5Y3MGuco4.jpg
18  /zaFa1NRZEnFgRTv5OVXkNIZO78O.jpg
19  /qriaeUUwdmlgethK3aSAx68mG05.jpg
```

Milestone1_02_store_data_tmdb

April 5, 2017

1 Store data

As calling API everytime may takes a while and we may hit the limit of API, we will store the data from API for now.

```
In [1]: import tmdbsimple as tmdb
        tmdb.API_KEY = "71e259894a515060876bab2a33d6bdc9"
```

```
In [2]: from imdb import IMDb
        import pandas as pd
        from PIL import Image
        from StringIO import StringIO
        import requests
        import os
        import time
        from shutil import copyfile
```

```
In [3]: dir_python_notebook = os.getcwd()
        dir_movie_project = os.path.abspath(os.path.join(dir_python_notebook, os.pa
        dir_data = os.path.join(dir_movie_project, 'data')
```

2 TMDB

```
In [22]: # get column names to retrieve information
         tmdb_movie_id = tmdb.Movies(2)
         movie_info = tmdb_movie_id.info()
         tmdb_info_column  = tmdb_movie_id.info().keys()
         tmdb_info_column  = [str(c) for c in tmdb_info_column]
         tmdb_info_column.sort()
```

```
In [23]: tmdb_info_column
```

```
Out[23]: ['adult',
          'backdrop_path',
          'belongs_to_collection',
          'budget',
          'genres',
```

```
 'homepage',
 'id',
 'imdb_id',
 'original_language',
 'original_title',
 'overview',
 'popularity',
 'poster_path',
 'production_companies',
 'production_countries',
 'release_date',
 'revenue',
 'runtime',
 'spoken_languages',
 'status',
 'tagline',
 'title',
 'video',
 'vote_average',
 'vote_count']
```

```
In [29]: movies = pd.DataFrame(columns=tmdb_info_column)
         tmdb_filename = str(dir_data)+'\\drv_tmdb_movie_details.json'
         tmdb_filename_backup = str(dir_data)+'\\drv_tmdb_movie_details_bkp.json'

         # if we already have a movie data file, we can just continue appending it
         i_start = 1
         i_end = i_start+10000
         i_list = range(i_start,i_end)

         if os.path.isfile(tmdb_filename):
             movies = pd.read_json(tmdb_filename)
             if len(movies['id']) >0:
                 movie_ids = movies['id'].tolist()
                 tmdb_info_column = movies.columns
                 i_list = [x for x in i_list if x not in movie_ids]
             else:
                 movies = pd.DataFrame(columns=tmdb_info_column)


         for i in i_list:
             #skip the non-existing movie ids
             if (i % 40 == 0):
                 # make sure we do not hit API limit
                 time.sleep(12)
             if (i % 100 == 0):
                 movies.to_json(path_or_buf= tmdb_filename)
             try:
```

```
                tmdb_movies = tmdb.Movies(i)
                info = tmdb_movies.info()

                movie_details = []
                for c in tmdb_info_column:
                    if info.has_key(c):
                        if info[c] is not None:
                            if c in ['genres','spoken_languages','production_count
                                movie_details.append([d['name'] for d in info[c]])
                            elif c in ['belongs_to_collection']:
                                movie_details.append(info[c]['name'])
                            else:
                                movie_details.append(info[c])
                        else:
                            movie_details.append(None)
                    else:
                            movie_details.append(None)
                movies.loc[i] = movie_details
            except Exception:
                continue

        movies.to_json(path_or_buf= tmdb_filename)
```

```
In [27]: #make a backup in case of corruption
         copyfile(tmdb_filename, tmdb_filename_backup)
```

```
In [25]: tmdb_filename = str(dir_data)+'\\drv_tmdb_movie_details.json'
         movies = pd.read_json(tmdb_filename)
```

```
In [ ]:
```

Store data

As calling API everytime may takes a while and we may hit the limit of API, we will store the data from API for now.

In [3]:
```
import tmdbsimple as tmdb
tmdb.API_KEY = "71e259894a515060876bab2a33d6bdc9"
```

In [4]:
```
import imdb as ib
from imdb import IMDb
import pandas as pd
from PIL import Image
from StringIO import StringIO
import requests
import os
import time
from shutil import copyfile
import types
import numpy as np
```

In [5]:
```
dir_python_notebook = os.getcwd()
dir_movie_project = os.path.abspath(os.path.join(dir_python_notebook, os.
dir_data = os.path.join(dir_movie_project, 'data')
```

IMDB

In [6]:
```
imdb = IMDb()
#get a movie by id
imdb_movie = imdb.get_movie('0325980')
#access attributes of the movie by dictionary keys
imdb.update(imdb_movie)
#available keys
imdb_info_column  =imdb_movie.keys()
imdb_info_column  = [str(c) for c in imdb_info_column]
imdb_info_column.sort()
```

In [7]:
```
print(len(imdb_info_column))
print(imdb_info_column)
```

```
61
['akas', 'animation department', 'art department', 'art direction', 'a
spect ratio', 'assistant director', 'camera and electrical department'
, 'canonical title', 'cast', 'casting department', 'casting director',
'certificates', 'cinematographer', 'color info', 'costume department',
'costume designer', 'countries', 'country codes', 'cover url', 'direct
or', 'distributors', 'editor', 'editorial department', 'full-size cove
r url', 'genres', 'kind', 'language codes', 'languages', 'location man
agement', 'long imdb canonical title', 'long imdb title', 'make up', '
miscellaneous companies', 'miscellaneous crew', 'mpaa', 'music departm
ent', 'original music', 'plot', 'plot outline', 'producer', 'productio
n companies', 'production design', 'production manager', 'rating', 'ru
ntimes', 'set decoration', 'smart canonical title', 'smart long imdb c
anonical title', 'sound crew', 'sound mix', 'special effects companies
', 'special effects department', 'stunt performer', 'thanks', 'title',
'top 250 rank', 'transportation department', 'visual effects', 'votes'
, 'writer', 'year']
```

In [8]:
```
# IMDB API shows that we can retrieve information from proprty of movies
imdb_info_column = imdb_movie.keys_alias.keys()
imdb_info_column.sort()
print(len(imdb_info_column))
print(imdb_info_column)
```

```
85
['actors', 'actresses', 'aka', 'also known as', 'amazon review', 'art
direction by', 'casting', 'casting by', 'certificate', 'certification'
, 'certifications', 'cinematography', 'cinematography by', 'color', 'c
ostume and wardrobe department', 'costume design', 'costume design by'
, 'country', 'cover', 'created by', 'crew members', 'crewmembers', 'di
rected by', 'distribution', 'distribution companies', 'distribution co
mpany', 'distributor', 'editing', 'episodes cast', 'episodes number',
'faq', 'film editing', 'film editing by', 'frequently asked questions'
, 'full-size cover', 'genre', 'guest', 'guest appearances', 'lang', 'l
anguage', 'make-up', 'makeup', 'makeup department', 'merchandise', 'me
rchandising', 'misc companies', 'misc company', 'misc crew', 'miscella
neous', 'miscellaneous company', 'miscellaneous links', 'miscellaneous
crew', 'music', 'non-original music by', 'notable tv guest appearances
', 'original music by', 'other companies', 'other company', 'other cre
w', 'parental guide', 'photographs', 'plot summaries', 'plot summary',
'produced by', 'production company', 'production countries', 'producti
on country', 'production management', 'runtime', 'sales', 'seasons', '
second unit director', 'second unit director or assistant director', '
set decoration by', 'sound department', 'soundclips', 'special effects
by', 'special effects company', 'stunts', 'tv guests', 'tv schedule',
'user rating', 'videoclips', 'visual effects by', 'writing credits']
```

In [9]:
```
for c in imdb_info_column:
    if imdb_movie.get(c) is None:
        print(c)
    #else:
    #    print (imdb_movie.get(c))
```

```
amazon review
created by
episodes cast
episodes number
faq
frequently asked questions
full-size cover
guest
guest appearances
merchandise
merchandising
miscellaneous
miscellaneous links
non-original music by
notable tv guest appearances
parental guide
photographs
sales
seasons
soundclips
special effects by
tv guests
tv schedule
videoclips
```

In [10]:
```python
for c in imdb_info_column:
    print(c)
    print(imdb_movie.get(c))
```

```
actors
[<Person id:0000136[http] name:_Depp, Johnny_>, <Person id:0001691[htt
p] name:_Rush, Geoffrey_>, <Person id:0089217[http] name:_Bloom, Orlan
do_>, <Person id:0461136[http] name:_Knightley, Keira_>, <Person id:02
02603[http] name:_Davenport, Jack_>, <Person id:0000596[http] name:_Pr
yce, Jonathan_>, <Person id:0034305[http] name:_Arenberg, Lee_>, <Pers
on id:0188871[http] name:_Crook, Mackenzie_>, <Person id:1400933[http]
name:_O'Hare, Damian_>, <Person id:1099016[http] name:_New, Giles_>, <
Person id:0055845[http] name:_Barnett, Angus_>, <Person id:0047549[htt
p] name:_Bailie, David_>, <Person id:1400913[http] name:_Jr., Michael
Berry_>, <Person id:0802280[http] name:_Jr., Isaac C. Singleton_>, <Pe
rson id:0573618[http] name:_McNally, Kevin_>, <Person id:0262125[http]
name:_Etienne, Treva_>, <Person id:0757855[http] name:_Saldana, Zoe_>,
<Person id:0801838[http] name:_Siner, Guy_>, <Person id:0552924[http]
name:_Martin, Ralph P._>, <Person id:0628225[http] name:_Newman, Paula
J._>, <Person id:0445284[http] name:_Keith, Paul_>, <Person id:0808057
[http] name:_Smith, Dylan_>, <Person id:1096355[http] name:_Dryzek, Lu
cinda_>, <Person id:0212472[http] name:_de Woolfson, Luke_>, <Person i
d:0992126[http] name:_Tighe, Michael Sean_>, <Person id:0254862[http]
```

In [11]:
```python
# IMDB API shows that we can retrieve information from proprty of movies
imdb_info_column2 = imdb_movie.keys_alias.values()
imdb_info_column2 = list(set(imdb_info_column2))
imdb_info_column2.sort()
print(len(imdb_info_column2))
print(imdb_info_column2)
```

```
49
['airing', 'akas', 'amazon reviews', 'art direction', 'assistant direc
tor', 'cast', 'casting director', 'certificates', 'cinematographer', '
color info', 'costume department', 'costume designer', 'countries', 'c
over url', 'creator', 'director', 'distributors', 'editor', 'faqs', 'f
ull-size cover url', 'genres', 'guests', 'languages', 'make up', 'merc
handising links', 'misc links', 'miscellaneous companies', 'miscellane
ous crew', 'non-original music', 'number of episodes', 'number of seas
ons', 'original music', 'parents guide', 'photo sites', 'plot', 'produ
cer', 'production companies', 'production manager', 'rating', 'runtime
s', 'set decoration', 'sound clips', 'sound crew', 'special effects',
'special effects companies', 'stunt performer', 'video clips', 'visual
effects', 'writer']
```

In [12]:
```
for c in imdb_info_column2:
    print(c)
    if (imdb_movie.has_key(c)):
        print(imdb_movie[c])
    else:
        print(None)
```

```
xe9rola Negra::Brazil (imdb display title)', u'Pirates des Cara\xefbes
- La mal\xe9diction de la Perle Noire::Canada (French title)', u'Pirat
es des Cara\xefbes - La mal\xe9diction du Black Pearl::France (imdb di
splay title)', u'Pirates of the Caribbean: Den sorte forbandelse::Denm
ark (imdb display title)', u'Pirates of the Caribbean: Mustan helmen k
irous::Finland', u'Pirates of the Caribbean: Svarta P\xe4rlans f\xf6rb
annelse::Sweden (imdb display title)', u'Pirati dei Caraibi: La maledi
zione della prima luna::Italy (alternative title)', u'Pirati s Kariba:
Prokletstvo Crnog bisera::Croatia (imdb display title)', u'Pirati s Ka
ribov: prekletstvo \u010drnega bisera::Slovenia', u'Pirati sa Kariba -
Prokletstvo Crnog bisera::Serbia (imdb display title)', u'Piratii din
Caraibe: Blestemul Perlei Negre::Romania (imdb display title)', u"Shod
eday Ha-Caribim: Klalat Ha-Pnina Ha-Sh'hora::Israel (Hebrew title)", u
'Sj\xf3r\xe6ningjar \xe1 Kar\xedbahafi: B\xf6lvun sv\xf6rtu perlunnar:
:Iceland (imdb display title)']
amazon reviews
None
art direction
[<Person id:0384192[http] name:_Hill, Derek R._>, <Person id:0694358[h
ttp] name:_Powels, Michael_>, <Person id:0865042[http] name:_Tocci, Ja
```

In [14]:
```
tmdb_filename = str(dir_data)+'\\drv_tmdb_movie_details.json'
tmdb_movies = pd.read_json(tmdb_filename)
tmdb_movies.head(1)
```

Out[14]:

	adult	backdrop_path	belongs_to_collection	budget	genres
100	False	/kzeR7BA0htJ7BeI6QEUX3PVp39s.jpg	None	1350000	[Comedy Crime]

1 rows × 25 columns

In [15]:
```
# get IMDB movies that we already have data for TMDB
imdb_ids = tmdb_movies['imdb_id'].tolist()
tmdb_ids = tmdb_movies['id'].tolist()

imdb_ids = [ str(imdb_id.replace('tt','')) for imdb_id in imdb_ids]
imdb_ids = [ imdb_id for imdb_id in imdb_ids if imdb_id !='' and imdb_id

imdb_ids = list(set(imdb_ids))
```

In [16]:
```
tmdb_movies.head(1)
```

Out[16]:

	adult	backdrop_path	belongs_to_collection	budget	genres
100	False	/kzeR7BA0htJ7BeI6QEUX3PVp39s.jpg	None	1350000	[Comedy Crime]

1 rows × 25 columns

In [17]:
```
# verify that we get the same movie as TMDB

imdb = IMDb()
imdb_movie = imdb.get_movie(imdb_id)
#access attributes of the movie by dictionary keys
imdb.update(imdb_movie)
imdb_movie['title']
```

Out[17]: u'Der freie Wille'

In [18]:
```
print(imdb_movie)
```

Der freie Wille

In [115]:
```
imdb_info_column = imdb_movie.keys_alias.values()
imdb_info_column = list(set(imdb_info_column))
if 'imdb_id' not in imdb_info_column:
    imdb_info_column.append('imdb_id')
imdb_info_column.sort()

movies = pd.DataFrame(columns=imdb_info_column)
invalid_imdb_ids = []

imdb_filename = str(dir_data)+'\\drv_imdb_movie_info.json'
imdb_filename_backup = str(dir_data)+'\\drv_imdb_movie_info_bkp.json'
imdb_invalid_id_filename = str(dir_data)+'\\drv_imdb_movie_invalid_id.jso
count = 0
```

```
count = 0
# if we already have a movie data file, we can just continue appending it
if os.path.isfile(imdb_filename):
    movies = pd.read_json(imdb_filename)
    if len(movies['imdb_id'].tolist()) >0:
        movie_ids = movies['imdb_id'].tolist()
        imdb_ids = [x for x in imdb_ids if x not in movie_ids]
        imdb_info_column = movies.columns
    else:
        movies = pd.DataFrame(columns=imdb_info_column)

# we don't want to waste our effort to load invalid imdb_id
if os.path.isfile(imdb_invalid_id_filename):
    invalid_imdb_id_df = pd.read_json(imdb_invalid_id_filename)
    if len(invalid_imdb_id_df['invalid_imdb_id'].tolist()) >0:
        invalid_imdb_ids = invalid_imdb_id_df['invalid_imdb_id'].tolist()
        imdb_ids = [x for x in imdb_ids if x not in invalid_imdb_ids]

for i in imdb_ids:
    count += 1
    if (count % 500 == 0):
        movies.to_json(path_or_buf= imdb_filename)
        if (len(invalid_imdb_ids)>0):
            invalid_imdb_id_df = pd.DataFrame({'invalid_imdb_id': invalid
            invalid_imdb_id_df.to_json(path_or_buf= imdb_invalid_id_filen
    #skip the non-existing movie ids
    #print(i)
    try:
        imdb_movie = imdb.get_movie(i)
        imdb.update(imdb_movie)
        if (len(imdb_movie.keys()) > 0) :
            movie_details = []
            for c in imdb_info_column:
                if imdb_movie.has_key(c):
                    info_field = imdb_movie[c]
                    if info_field is not None:
                        if type(info_field) is list:
                            if isinstance(info_field[0], ib.Person.Person
                                info_list = []
                                for item in info_field:
                                    info_list.append(item.getID())
                                movie_details.append(info_list)
                            elif isinstance(info_field[0], ib.Company.Com
                                info_list = []
                                for item in info_field:
                                    info_list.append(item.getID())
                                movie_details.append(info_list)
                            else:
                                movie_details.append(info_field)
                        else:
                            movie_details.append(info_field)
```

```
                    else:
                        if c == "imdb_id":
                            movie_details.append(i)
                        else:
                            movie_details.append(None)

                else:
                        movie_details.append(None)

            movies.loc[len(movies.index)] = movie_details

            movies.head(5)
    except Exception:
        invalid_imdb_ids.append(i)
        continue

movies.to_json(path_or_buf= imdb_filename)
if (len(invalid_imdb_ids)>0):
    invalid_imdb_id_df = pd.DataFrame({'invalid_imdb_id': invalid_imdb_id
    invalid_imdb_id_df.to_json(path_or_buf= imdb_invalid_id_filename)
```

```
2017-04-05 12:31:48,582 CRITICAL [imdbpy] C:\Users\Chrystal\Anaconda2\
envs\py27\lib\site-packages\imdb\_exceptions.py:35: IMDbDataAccessErro
r exception raised; args: ({'exception type': 'IOError', 'url': 'http:
//akas.imdb.com/title/tt0072996/combined', 'errcode': 'socket error',
'proxy': '', 'original exception': IOError('socket error', error(10060
, 'A connection attempt failed because the connected party did not pro
perly respond after a period of time, or established connection failed
because connected host has failed to respond')), 'errmsg': '[Errno 100
60] A connection attempt failed because the connected party did not pr
operly respond after a period of time, or established connection faile
d because connected host has failed to respond'},); kwds: {}
Traceback (most recent call last):
  File "C:\Users\Chrystal\Anaconda2\envs\py27\lib\site-packages\imdb\p
arser\http\__init__.py", line 202, in retrieve_unicode
    uopener = self.open(url)
  File "C:\Users\Chrystal\Anaconda2\envs\py27\lib\urllib.py", line 213
, in open
    return getattr(self, name)(url)
  File "C:\Users\Chrystal\Anaconda2\envs\py27\lib\urllib.py", line 350
```

In [116]:

```
#make a backup in case of corruption
copyfile(imdb_filename, imdb_filename_backup)
```

In [117]:
```
imdb_filename = str(dir_data)+'\\drv_imdb_movie_info.json'
imdb_movies = pd.read_json(imdb_filename)
imdb_movies.head(5)
```

Out[117]:

	airing	akas	amazon reviews	art direction	assistant director	cast	casting director	certificat
0	NaN	[Ultimate Avengers 2: Rise of the Panther::USA...	NaN	None	None	[1225431, 0217221, 0557219, 0941404, 0001882, ...	[0800493]	[Argentina (DVD ratir Russia:12 Russ...
1	NaN	[Bajo la piel::Argentina (imdb display title),...	NaN	None	[0116502, 0348808, 0601635, 0931271]	[0164809, 0891895, 0252961, 0862328, 0001026, ...	[0058089, 0338418]	[Argentina Australia: Netherlar
10	NaN	[Ultimate Avengers 2: Rise of the Panther::USA...	NaN	None	None	[1225431, 0217221, 0557219, 0941404, 0001882, ...	[0800493]	[Argentina (DVD ratir Russia:12 Russ...
100	NaN	[One Wild Night::, USA (working title), Возмож...	NaN	[0055618]	[0742835, 0815340]	[0001844, 0000124, 0000551, 0612487, 0413698, ...	[0228938]	[Australia Germany Iceland:L Kor...
1000	NaN	[Wicked City::Australia (imdb display title), ...	NaN	[0644470]	None	[0946344, 0810987, 0589645, 0297847, 0448950, ...	[0532146, 0810987, 3108562]	[Australia Canada:1 (Alberta/E ...

5 rows × 50 columns

```
In [ ]:
```

Milestone1_04_comment_genre_classification

April 5, 2017

1 Movie genre classification

1.1 Questions

We need to address the following problems before we do movie genre classification: 1. TMDB and IMDB have different movie genre list and this can create issues for prediction. Which list of genre we use? What do we do if they disagree? 2. A movie can have more than 1 genre. The data from TMDB and IMDB will not indicate which one is the main genre if there is more than 1 genre. However, when we do movie genre prediction, we may only want our reponse to be 1 genre. What should we predict? A genre or a list of genre? 3. If we aim to tag a movie with multiple genres, what metrics do we use to evaluate our methods? The accuracy is no longer a simple 0-1 evaluation.

1.2 Additional data set

Amazon API provides genre tag and frequency weighting. We can use it to find movie primary genre.

1.3 Approach

Below are some of our thoughts on the questions in the first section.

1.3.1 TMDB genre vs IMDB genre

Issues: 1. TMDB has a shorter genre list than IMDB list. Although we aim to pull information of the same set of movies, the genre lists extracted from IMDB and TMDB data are still different. 2. Some genre distinction are not based on plots. For example, "Foreign" genre can refer to film out of the country. 3. Some genre have too few movies. Possible solutions: 1. We can check if the genre classification are similiar for IMDB and TMDB by the following: look at percentage breakdown of each genre. If the results are significantly different amonng the two databases, we will have to explore what causes the difference. We can then merge the list or just to use IMDB genre list if the genre are not that different from TMDB. 2. We should perhaps remove some genres based on movie release date or country. 3. For genre with little movies, we can merge them into another genre. Ex. If we do cluster analysis and find Noir is indeed close to Crime, we can group movies from Noir to Crime.

1.3.2 What are our reponse variable?

Issues: 1. Movies have 3 genres on average from IMDB and TMDB. Unforunately, neither database tags primary genre of a movie. Possible Solutions: 1. We create clusteres of genres. Ex. Cluser 1 consists of Action,Drama,Romance; cluster 2 consists of Horror. A movie can be assigned to cluster 1. Some researches used this method. 2. We will have a outpuf vecot of Y. Y = [Y_horror, Y_romance, Y_drama,. . .] 3. We can do output only 1 Y of primary genre. But to do this, we will need to get data from Amazon. Currently, we prefer option 2.

1.3.3 Multi-label classification

We will use multi-label classification. Before we discuss which methods to use (ex. KNN, SVM), we should consdier how we will treat the response variable first.

1. Binary Relevence(BR) we seperate each genre into seperate problems (one for each genre). However, this ignore label dependence. Ex. if a movie is tagged as Drama, it is likely that it is also tagged as Action. if a movie is tagged as Horror, it is likely that it is also tagged as Romance. If two classes of a genre (Yes/No) have very uneven sizes in the training set, the classifier will lean toward the class with higher movie number. There is a method called a label correction strategy that can help to improve accuracy For example, if our prediction is [Y_horror, Y_romance, Y_drama]= [1,1,0], which does not really happen in training set. We find another likely matching vector. We may change our prediction to be [1,0,1].

2. Classifier Chains (CC) We seperate each genre into seperate problems, but include previous predictions as predictors. For example, X is our predictor for Y_horror. Next, X, Y_horror are our predictor for Y_romance. However, error may be propagated down the chain.

3. Label Powerset (LP) Instead of having seperate Y_i for each genre i, we will predict only Y. Y has 2ˆI possible values where I is the number of genre. For example, if Y_horror = 1, Y_romance = 0, Y_drama = 1, Y = [101] However, imbalance of the data can be an issue.

4. Neural network The output node will be each genre label.

We aim to try out meothds 1 to 4 with different algorithms. We also need to examine label dependence for each genre by examining co-occurance frequency of genre. It will give us a better picture of which method to use. We can also transform Y to [Y_OR,Y_AND, Y_XOR].

1.3.4 Classification Algorithms

Below is the list of algorithms. We will likely add more algorithms later. 1. KNN 2. Kmeans 3. Naive Bayes 4. SVM 5. LDA

1.3.5 Classification Evaluation

Imagine if the reponse variable Y are Y_horror, Y_romance, Y_drama, . . . for each genre i, we will need to find appropriate methods to evaluate classification methods.

We can calculate different measurements and evalute by multiple measurements. 1. Compare bit-wise This can be too lenient 2. Compare vector-ise This can be too strict 3. Hamming loss (for BR) 4. 0/1 loss (for CC) 5. precision 6. recall 7. F measures

For some of the measurements above, we can use macro-average(which give equal qeight to every class) or micro-averaging(which weights class relatively to its example frequency).

1.4 useful resources

- IMDB genre guide

```
In [2]: import tmdbsimple as tmdb
        tmdb.API_KEY = "71e259894a515060876bab2a33d6bdc9"
```

```
In [3]: import imdb as ib
        from imdb import IMDb
        import pandas as pd
        from PIL import Image
        from StringIO import StringIO
        import requests
        import os
        import time
        from shutil import copyfile
        import types
        import numpy as np
```

```
In [4]: dir_python_notebook = os.getcwd()
        dir_movie_project = os.path.abspath(os.path.join(dir_python_notebook, os.pa
        dir_data = os.path.join(dir_movie_project, 'data')
```

2 Load data

Use the data from the files for now instead of calling API.

```
In [5]: tmdb_filename = str(dir_data)+'\\drv_tmdb_movie_details.json'
        imdb_filename = str(dir_data)+'\\drv_imdb_movie_info.json'
        tmdb_movies = pd.read_json(tmdb_filename)
        imdb_movies = pd.read_json(imdb_filename)
```

```
In [40]: def get_genre(tmdb_movies ,key):
             tmdb_genre = tmdb_movies[tmdb_movies[key].notnull()][key].tolist()
             tmdb_genre_set = set()
             for g in tmdb_genre:
                 if g is not None:
                     tmdb_genre_set = tmdb_genre_set.union(set(g))
             tmdb_genre = list(tmdb_genre_set)
             tmdb_genre.sort()
             return(tmdb_genre)
```

```
In [41]: def get_genere_num (row, column_name):
             if row[column_name] is None :
                 return 0
             else:
                 return len(row[column_name])
```

In [42]:
```
def is_genre (row, column_name, genre):
    """check if that movie is in this genre as a movie can have more than
    if row[column_name] is None :
        return 0
    else:
        if genre in row[column_name] :
            return 1
        else:
            return 0
```

In [43]:
```
tmdb_genre = get_genre(tmdb_movies,u'genres')
tmdb_movies[u'genre_num'] = tmdb_movies.apply(lambda row: get_genere_num(r

for g in tmdb_genre:
    tmdb_movies[g] = tmdb_movies.apply(lambda row: is_genre(row,u'genres',
```

In [44]:
```
np.mean(tmdb_movies[u'genre_num'])
```

Out[44]:
```
2.255087358684481
```

In [45]:
```
tmdb_genre_df = tmdb_movies[tmdb_genre].mean(axis=0)
tmdb_genre_df
```

Out[45]:
```
Action             0.180473
Adventure          0.123535
Animation          0.025488
Comedy             0.324974
Crime              0.156835
Documentary        0.037410
Drama              0.498869
Family             0.049538
Fantasy            0.072765
Foreign            0.021583
History            0.045427
Horror             0.099692
Music              0.024666
Mystery            0.063926
Romance            0.142652
Science Fiction    0.100925
TV Movie           0.005550
Thriller           0.233299
War                0.027338
Western            0.020144
dtype: float64
```

In [58]:
```
tmdb_genre_df[tmdb_genre_df < 0.05]
```

Out[58]:
```
Animation      0.025488
Documentary    0.037410
```

```
Family          0.049538
Foreign         0.021583
History         0.045427
Music           0.024666
TV Movie        0.005550
War             0.027338
Western         0.020144
dtype: float64
```

In [46]:
```
imdb_genre = get_genre(imdb_movies,u'genres')
imdb_movies[u'genre_num'] = imdb_movies.apply(lambda row: get_genere_num(r

for g in imdb_genre:
    imdb_movies[g] = imdb_movies.apply(lambda row: is_genre(row,u'genres',
```

In [51]:
```
# number of movies with no genre
sum(imdb_movies[u'genres'].isnull())
```

Out[51]: 209

In [47]:
```
imdb_genre_df = imdb_movies[imdb_genre].mean(axis=0)
imdb_genre_df
```

Out[47]:
```
Action          0.154828
Adult           0.005870
Adventure       0.114323
Animation       0.054104
Biography       0.037081
Comedy          0.355396
Crime           0.153067
Documentary     0.065600
Drama           0.459642
Family          0.084532
Fantasy         0.081646
Film-Noir       0.010664
History         0.032042
Horror          0.108600
Music           0.043587
Musical         0.029009
Mystery         0.076460
News            0.000147
Reality-TV      0.000098
Romance         0.188876
Sci-Fi          0.082233
Short           0.032531
Sport           0.033216
Talk-Show       0.000098
Thriller        0.201106
War             0.040603
```

```
Western        0.023188
dtype: float64
```

```
In [57]: imdb_genre_df[imdb_genre_df < 0.05]
```

```
Out[57]: Adult          0.005870
         Biography      0.037081
         Film-Noir      0.010664
         History        0.032042
         Music          0.043587
         Musical        0.029009
         News           0.000147
         Reality-TV     0.000098
         Short          0.032531
         Sport          0.033216
         Talk-Show      0.000098
         War            0.040603
         Western        0.023188
         dtype: float64
```

```
In [52]: np.mean(imdb_movies[u'genre_num'])
```

```
Out[52]: 2.4685451521377555
```

2.1 Observations on genre list

1. The genre lists between IMDB and TMDB are very similiar so far. Perhaps it is because we have extracted the same movie.
2. A movie has 2 genres on average.
3. Some of the genres(ex. Biography and War) only account for less than 5%. One-vs-all approach may not be good.

2.2 Compare Genre from IMDB, TMDB

```
In [21]: tmdb_genre = tmdb_genres.values()
         tmdb_genre.sort()
         tmdb_genre
```

```
Out[21]: ['Action',
          'Adventure',
          'Animation',
          'Comedy',
          'Crime',
          'Documentary',
          'Drama',
          'Family',
          'Fantasy',
          'History',
          'Horror',
```

```
         'Music',
         'Mystery',
         'Romance',
         'Science Fiction',
         'TV Movie',
         'Thriller',
         'War',
         'Western']
```

IMDB genre list: http://www.imdb.com/help/search?domain=helpdesk_faq&index=2&file=genres

```
In [87]: imdb_genre = ['Action','Adult','Adventure','Animation','Biography','Comedy
                      'Fantasy','Film Noir','Game-Show','History','Horror','Musica
                      'Romance','Sci-Fi','Short','Sport','Talk-Show','Thriller','W
```

IMBD has more genres than TMDB, most of them have the same corresponds, except 'Science Fiction' (TMDB) = 'Sci-Fi' (IMDB).

```
In [88]: genre = list(set(imdb_genre + tmdb_genre))

In [89]: #remove duplicated
         genre.remove('Sci-Fi')

In [90]: #complete genres
         genre.sort()
         genre

Out[90]: ['Action',
          'Adult',
          'Adventure',
          'Animation',
          'Biography',
          'Comedy',
          'Crime',
          'Documentary',
          'Drama',
          'Family',
          'Fantasy',
          'Film Noir',
          'Game-Show',
          'History',
          'Horror',
          'Music',
          'Musical',
          'Mystery',
          'News',
          'Reality-TV',
          'Romance',
```

```
          'Science Fiction',
          'Short',
          'Sport',
          'TV Movie',
          'Talk-Show',
          'Thriller',
          'War',
          'Western']
```

In [91]:
```
tmdb_genres
```

Out[91]:
```
{12: 'Adventure',
 14: 'Fantasy',
 16: 'Animation',
 18: 'Drama',
 27: 'Horror',
 28: 'Action',
 35: 'Comedy',
 36: 'History',
 37: 'Western',
 53: 'Thriller',
 80: 'Crime',
 99: 'Documentary',
 878: 'Science Fiction',
 9648: 'Mystery',
 10402: 'Music',
 10749: 'Romance',
 10751: 'Family',
 10752: 'War',
 10770: 'TV Movie'}
```

In [92]:
```
imdb_genres = {12: 'Adventure',
 14: 'Fantasy',
 16: 'Animation',
 18: 'Drama',
 27: 'Horror',
 28: 'Action',
 35: 'Comedy',
 36: 'History',
 37: 'Western',
 53: 'Thriller',
 80: 'Crime',
 99: 'Documentary',
 878: 'Sci-Fi',
 9648: 'Mystery',
 10402: 'Music',
 10749: 'Romance',
 10751: 'Family',
```

```
 10752: 'War',
 10770: 'TV Movie',
 1: 'Adult',
 2: 'Biography',
 3: 'Film Noir',
 4: 'Game-Show',
 5: 'Musical',
 6: 'News',
 7: 'Reality-TV',
 8: 'Short',
 9: 'Sport',
10: 'Talk-Show'}
```

In [93]:
```
genre_dic = tmdb_genres.copy()
genre_dic.update(imdb_genres )
```

In [94]:
```
#as a dictironary format, most id is same as id in tmdb (updated 878 will
genre_dic
```

Out[94]:
```
{1: 'Adult',
 2: 'Biography',
 3: 'Film Noir',
 4: 'Game-Show',
 5: 'Musical',
 6: 'News',
 7: 'Reality-TV',
 8: 'Short',
 9: 'Sport',
 10: 'Talk-Show',
 12: 'Adventure',
 14: 'Fantasy',
 16: 'Animation',
 18: 'Drama',
 27: 'Horror',
 28: 'Action',
 35: 'Comedy',
 36: 'History',
 37: 'Western',
 53: 'Thriller',
 80: 'Crime',
 99: 'Documentary',
 878: 'Sci-Fi',
 9648: 'Mystery',
 10402: 'Music',
 10749: 'Romance',
 10751: 'Family',
 10752: 'War',
 10770: 'TV Movie'}
```

```
In [95]: items = imdb_genres.items()
         df_imdb = pd.DataFrame({'keys': [i[0] for i in items], 'values': [i[1] for
```

```
In [96]: df_imdb
```

```
Out[96]:      keys        values
         0   10752           War
         1       1         Adult
         2       2     Biography
         3       3     Film Noir
         4       4     Game-Show
         5       5       Musical
         6       6          News
         7       7    Reality-TV
         8       8         Short
         9       9         Sport
         10     10     Talk-Show
         11     12     Adventure
         12     14       Fantasy
         13     16     Animation
         14  10770      TV Movie
         15     27        Horror
         16     28        Action
         17  10402         Music
         18     35        Comedy
         19     36       History
         20     37       Western
         21   9648       Mystery
         22     53      Thriller
         23     80         Crime
         24     99   Documentary
         25     18         Drama
         26    878        Sci-Fi
         27  10749       Romance
         28  10751        Family
```

```
In [97]: items = tmdb_genres.items()
         df_tmdb = pd.DataFrame({'keys': [i[0] for i in items], 'values': [i[1] for
```

```
In [98]: items = genre_dic.items()
         df_genre = pd.DataFrame({'keys': [i[0] for i in items], 'values': [i[1] fo
```

```
In [99]: df1 = pd.merge(df_genre, df_tmdb, how='left', on=['keys'])
         df1 = df1.sort_values(['values_x'])
```

```
In [100]: df2 = pd.merge(df1, df_imdb,how='left', on=['keys'])
          df2.columns = ['id','genres','tmdb_genre','imdb_genre']
```

```
In [101]: df2
```

Out[101]:

	id	genres	tmdb_genre	imdb_genre
0	28	Action	Action	Action
1	1	Adult	NaN	Adult
2	12	Adventure	Adventure	Adventure
3	16	Animation	Animation	Animation
4	2	Biography	NaN	Biography
5	35	Comedy	Comedy	Comedy
6	80	Crime	Crime	Crime
7	99	Documentary	Documentary	Documentary
8	18	Drama	Drama	Drama
9	10751	Family	Family	Family
10	14	Fantasy	Fantasy	Fantasy
11	3	Film Noir	NaN	Film Noir
12	4	Game-Show	NaN	Game-Show
13	36	History	History	History
14	27	Horror	Horror	Horror
15	10402	Music	Music	Music
16	5	Musical	NaN	Musical
17	9648	Mystery	Mystery	Mystery
18	6	News	NaN	News
19	7	Reality-TV	NaN	Reality-TV
20	10749	Romance	Romance	Romance
21	878	Sci-Fi	Science Fiction	Sci-Fi
22	8	Short	NaN	Short
23	9	Sport	NaN	Sport
24	10770	TV Movie	TV Movie	TV Movie
25	10	Talk-Show	NaN	Talk-Show
26	53	Thriller	Thriller	Thriller
27	10752	War	War	War
28	37	Western	Western	Western

Summary:

1. IMDB covers all TMDB genres.
2. The only genre that has different names in IMDB and TMDB is “Science Fiction”
3. Combining all genres from IMDB, TMDB, we have total 28 genres
4. For TMDB, since genres are stored in {“id”:“genre”} pairs, we can use “genre_dic” to convert it to same genres as IMDB (i.e, ‘Science Fiction’ from TMDB will be ‘Sci-Fi’ after conversion).
5. For IMDB, we can just use its genres, since it will have same genres in our defined genres.

3 Steps to obtain genre list

1. filter data set to only use movies that have data in TMDB and IMDB, IMDB and TMDB genres must not be null
2. Use the genre list from IMDB.
3. Out of 28 genres, we should merge the genre with really small number(ex. less than 5% of movies) to another genre. The final result is unique genre list.

4. convert each genre from unique genre list to a seperate data field, and we will use 1/0 to indicate if a movie has this genre

3.1 Future Work:

1. Up to now, we maintain the genres from IMDB and TMDB. Some genres has very limited number of movies, so we are planning to merge minor genres to other major genres. One possible solution for merging genres is clustering genres, and we will do this part in milestone2.

Milestone1_05_EDA_TMDB

April 5, 2017

```
In [1]: %matplotlib inline
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn.apionly as sns
        import datetime as dt
```

1 Load data

Use the data from the files for now instead of calling API.

```
In [2]: tmdb_movies = pd.read_json("drv_tmdb_movie_details.json")
        imdb_movies = pd.read_json("drv_imdb_movie_details.json")
```

```
In [3]: # tmdb
        # max id
        print(max(tmdb_movies['id']))

        # rows
        print(len(tmdb_movies.index))
```

```
10010
4865
```

2 Format data

```
In [4]: def get_genre(tmdb_movies ,key):
            tmdb_genre = tmdb_movies[key].tolist()
            tmdb_genre_set = set()
            for g in tmdb_genre:
                tmdb_genre_set = tmdb_genre_set.union(set(g))
            tmdb_genre = list(tmdb_genre_set)
            tmdb_genre.sort()
            return(tmdb_genre)
```

```
In [5]: def is_genre (row, column_name, genre):
            """check if that movie is in this genre as a movie can have more than 1
            if genre in row[column_name] :
                return True
            else:
                return False
```

```
In [6]: # as the genre is in list format in the data field, we cannot utilize any c
        # thus, we have to transform data set

        def add_genre_columns(tmdb_movies, genre_column):
            tmdb_genre = get_genre(tmdb_movies,genre_column)
            tmdb_movies[u'genre_num'] = tmdb_movies.apply(lambda row: len(row[genre

            for g in tmdb_genre:
                tmdb_movies[g] = tmdb_movies.apply(lambda row: is_genre(row,genre_c
```

```
In [7]: tmdb_genre = get_genre(tmdb_movies,u'genres')
        tmdb_movies[u'genre_num'] = tmdb_movies.apply(lambda row: len(row[u'genres'

        for g in tmdb_genre:
            tmdb_movies[g] = tmdb_movies.apply(lambda row: is_genre(row,u'genres',g
```

```
In [ ]: tmdb_genre_df = tmdb_movies[tmdb_genre].apply(pd.value_counts).transpose()
```

3 Exploratory Analysis for TMDB

```
In [13]: print "Dimension of TMDB dataset:" ,tmdb_movies.shape
```

```
Dimension of TMDB dataset: (4865, 46)
```

```
In [14]: tmdb_movies.head(5)
```

```
Out[14]:        adult                    backdrop_path           belongs_to_collect
         10000  False  /Ar87X5wNldG33l3Ka1iMK26I6xZ.jpg                            N
         10003  False  /lJnl8xIhplfzUBtlJLsWBwvXVBj.jpg             The Saint Collect
         10005  False  /u7IzK6tISpsSuNkWMowl17gSA4e.jpg  Behind Enemy Lines Collect
         10007  False  /l9fdcaqaQQLE2K8cYwfoIPXi0i7.jpg           See No Evil Collect
         10009  False   /fAzT4AboZXP2Sj3zE2HcQ7qjMi.jpg        Brother Bear Collect

                   budget                                              genres  \
         10000          0                                    [Comedy, Drama]
         10003   68000000  [Thriller, Action, Romance, Science Fiction, A...
         10005          0                      [Action, Adventure, Thriller]
         10007    8000000                                [Horror, Thriller]
         10009  100000000                    [Adventure, Animation, Family]
```

```
                                                    homepage     id    imdb_i
10000                                                            10000  tt010974
10003                                                            10003  tt012005
10005  http://www.foxhome.com/behindenemylinesiiaxiso...  10005  tt049732
10007                                                            10007  tt043717
10009                http://movies.disney.com/brother-bear  10009  tt032888

      original_language                       original_title   ...   Hist
10000                es           La estrategia del caracol   ...     Fa
10003                en                           The Saint   ...     Fa
10005                en  Behind Enemy Lines II: Axis of Evil   ...     Fa
10007                en                         See No Evil   ...     Fa
10009                en                        Brother Bear   ...     Fa

       Horror  Music Mystery Romance Science Fiction  TV Movie  Thriller
10000   False  False   False   False           False     False     False
10003   False  False   False    True            True     False      True
10005   False  False   False   False           False     False      True
10007    True  False   False   False           False     False      True
10009   False  False   False   False           False     False     False

         War Western
10000  False   False
10003  False   False
10005  False   False
10007  False   False
10009  False   False

[5 rows x 46 columns]
```

Replace None with NaN:

```
In [15]: tmdb_movies.fillna(value=np.nan, inplace=True)
```

Check features

```
In [16]: tmdb_movies.columns.values
```

```
Out[16]: array([u'adult', u'backdrop_path', u'belongs_to_collection', u'budget',
                u'genres', u'homepage', u'id', u'imdb_id', u'original_language',
                u'original_title', u'overview', u'popularity', u'poster_path',
                u'production_companies', u'production_countries', u'release_date',
                u'revenue', u'runtime', u'spoken_languages', u'status', u'tagline',
                u'title', u'video', u'vote_average', u'vote_count', u'genre_num',
                u'Action', u'Adventure', u'Animation', u'Comedy', u'Crime',
                u'Documentary', u'Drama', u'Family', u'Fantasy', u'Foreign',
                u'History', u'Horror', u'Music', u'Mystery', u'Romance',
                u'Science Fiction', u'TV Movie', u'Thriller', u'War', u'Western'],
```

3.1 1. Check Missing Value and Duplicate

In [17]:
```
n_missing=[]
p=[]
for col in tmdb_movies:
    n_missing.append(tmdb_movies[col].isnull().values.sum())
    p.append((tmdb_movies[col].isnull().values.sum())/float(tmdb_movies.sh
missing=pd.DataFrame({'Features': tmdb_movies.columns.values, 'Missing':n_
```

In [18]:
```
df_cat=tmdb_movies[['genres', 'original_language',
         'original_title', 'homepage','overview',
         'production_companies', 'production_countries','spoken_languages','
```

In [19]:
```
n_missing=[]
p=[]
for col in df_cat:
    count=0
    for i in range(tmdb_movies.shape[0]):
        if len(tmdb_movies.iloc[i][col])==0:
            count+=1
    n_missing.append(count)
    p.append(count/float(tmdb_movies.shape[0]))
missing1=pd.DataFrame({'Features': df_cat.columns.values, 'Missing':n_miss
frames=[missing, missing1]
result=pd.concat(frames)
print "Number of missing values"
result[result['Missing']!=0]
```

```
Number of missing values
```

Out[19]:

	Features	Missing	Missing Precentage
1	backdrop_path	890	0.182939
2	belongs_to_collection	3940	0.809866
12	poster_path	366	0.075231
0	genres	50	0.010277
3	homepage	4030	0.828366
4	overview	116	0.023844
5	production_companies	871	0.179034
6	production_countries	51	0.010483
7	spoken_languages	26	0.005344
9	tagline	2300	0.472765

As shown above, 11 features have missing values. “belongs_to_collection”, “homepage” and “tagline” has comparatively high missing rate. Other features all have missing rates below 20%. For furture modeling, we may need to consider impute some missing values, or delete some features (with high missing rate) that is not very infomative.

In [20]:
```
tmdb_movies['title'].describe()
```

```
Out[20]: count                    4865
         unique                   4754
         top          Finders Keepers
         freq                        7
         Name: title, dtype: object
```

```
In [21]: tmdb_movies['id'].describe()
```

```
Out[21]: count     4865.000000
         mean      4712.598150
         std       3108.842204
         min          2.000000
         25%       1955.000000
         50%       4484.000000
         75%       7351.000000
         max      10010.000000
         Name: id, dtype: float64
```

```
In [22]: tmdb_movies[tmdb_movies['title']=='Finders Keepers'].iloc[:,2:24]
```

```
Out[22]:      belongs_to_collection  budget                      genres homepage    i
         5507                   NaN       0  [Comedy, Music, Foreign]          550
         5508                   NaN       0                  [Comedy]          550
         5510                   NaN       0                  [Comedy]          551
         5513                   NaN       0          [Action, Comedy]          551
         5514                   NaN       0                  [Comedy]          551
         5515                   NaN       0           [Comedy, Drama]          551
         5517                   NaN       0            [Crime, Drama]          551

                 imdb_id original_language   original_title  \
         5507  tt0060413                en  Finders Keepers
         5508  tt0087260                en  Finders Keepers
         5510  tt0043534                en  Finders Keepers
         5513  tt0846779                en  Finders Keepers
         5514  tt0231588                en  Finders Keepers
         5515  tt0018889                en  Finders Keepers
         5517  tt0311132                en  Finders Keepers

                                                   overview  popularity  \
         5507                                No overview found.    0.000518
         5508  On the run from the police and a female roller...    0.281762
         5510                                No overview found.    0.000230
         5513                                No overview found.    0.000393
         5514                                No overview found.    0.000144
         5515                                No overview found.    0.000143
         5517  This Vitaphone one-reel short, written by the ...    0.000178

                ...           production_countries release_date revenue runtime
         5507   ...               [United Kingdom]   1966-12-08       0      94
```

```
5508     ...       [United States of America]   1984-05-18        0       96
5510     ...                 [United Kingdom]   1952-01-01        0       74
5513     ...       [United States of America]   2005-01-01        0       90
5514     ...       [United States of America]   1921-01-01        0       35
5515     ...       [United States of America]   1928-02-18        0       60
5517     ...       [United States of America]   1929-12-27        0       20

      spoken_languages    status tagline            title  video vote_aver
5507          [English]  Released         Finders Keepers  False
5508          [English]  Released         Finders Keepers  False
5510          [English]  Released         Finders Keepers  False
5513          [English]  Released         Finders Keepers  False
5514          [English]  Released         Finders Keepers  False
5515          [English]  Released         Finders Keepers  False
5517          [English]  Released         Finders Keepers  False

[7 rows x 22 columns]
```

Although 7 movies has title "Finders Keepers", their are completely different movies, and their imdb_id and id are different. So we'll keep movies that have same titles.

There's no duplicate movie in the dataset in terms of TMDB ID.

4 2. Feature Exploration

In [23]:
```
numeric=tmdb_movies.select_dtypes(include = ['float64', 'int64'])
numeric.columns.values
numeric.describe()
```

Out[23]:

	budget	id	popularity	revenue	runtime
count	4.865000e+03	4865.000000	4865.000000	4.865000e+03	4865.000000
mean	1.352701e+07	4712.598150	0.728872	4.310044e+07	103.146146
std	2.945895e+07	3108.842204	1.080560	1.146627e+08	31.940519
min	0.000000e+00	2.000000	0.000000	0.000000e+00	0.000000
25%	0.000000e+00	1955.000000	0.024937	0.000000e+00	90.000000
50%	0.000000e+00	4484.000000	0.296467	0.000000e+00	100.000000
75%	1.300000e+07	7351.000000	1.023493	2.682836e+07	115.000000
max	3.800000e+08	10010.000000	13.130939	1.845034e+09	480.000000

	vote_average	vote_count	genre_num
count	4865.000000	4865.000000	4865.000000
mean	5.305694	266.885509	2.255087
std	2.481026	649.184148	1.167723
min	0.000000	0.000000	0.000000
25%	5.000000	3.000000	1.000000
50%	6.100000	37.000000	2.000000
75%	6.900000	232.000000	3.000000
max	10.000000	9653.000000	7.000000

```
In [24]: #Delete id and genre_num, which are not original features
         numeric=numeric.drop(numeric.columns[[1,7]], axis=1)
         print "Number of numerical features:", numeric.shape[1]
         numeric.describe()
```

```
Number of numerical features: 6
```

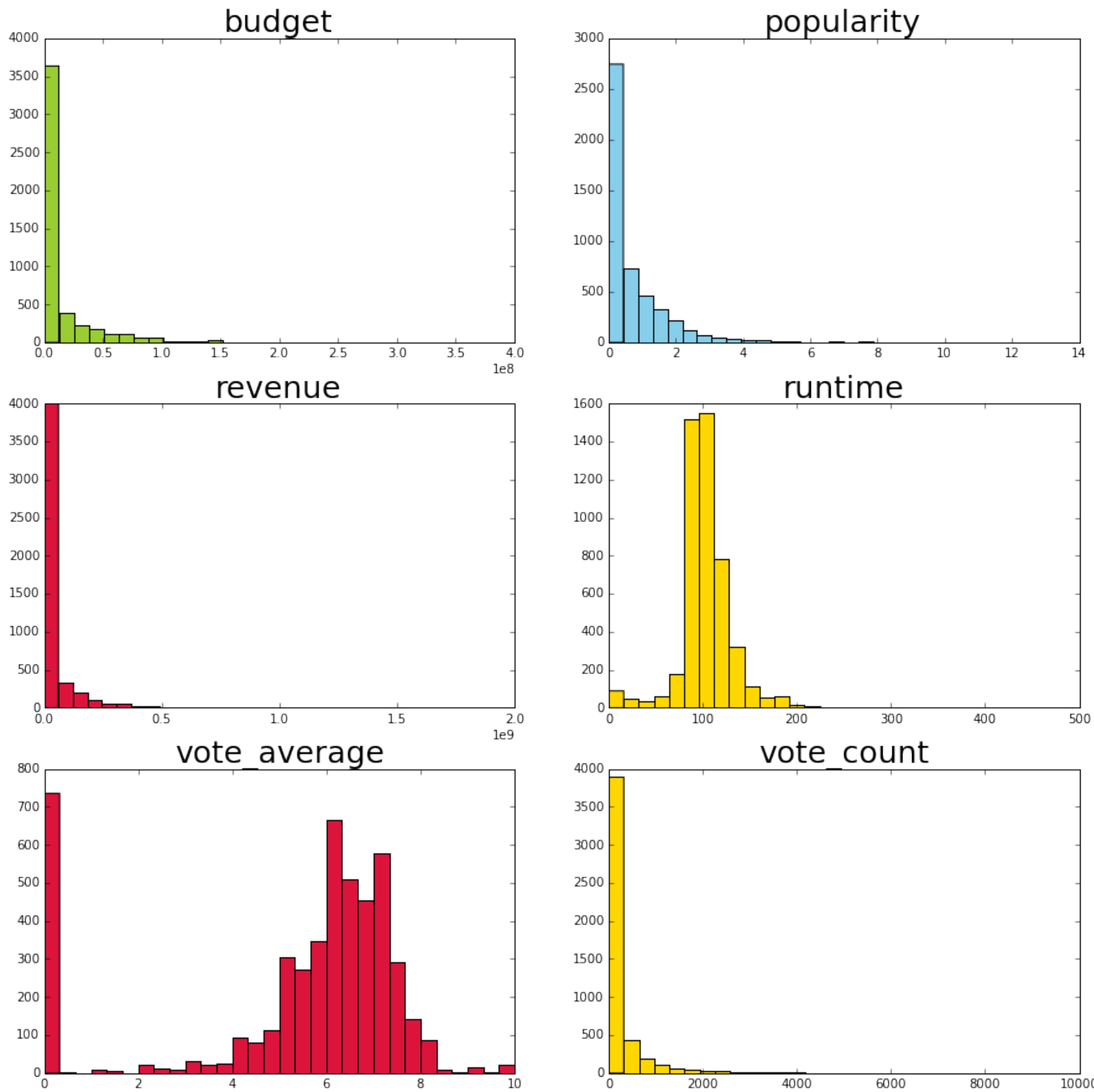
Out[24]:

	budget	popularity	revenue	runtime	vote_average
count	4.865000e+03	4865.000000	4.865000e+03	4865.000000	4865.000000
mean	1.352701e+07	0.728872	4.310044e+07	103.146146	5.305694
std	2.945895e+07	1.080560	1.146627e+08	31.940519	2.481026
min	0.000000e+00	0.000000	0.000000e+00	0.000000	0.000000
25%	0.000000e+00	0.024937	0.000000e+00	90.000000	5.000000
50%	0.000000e+00	0.296467	0.000000e+00	100.000000	6.100000
75%	1.300000e+07	1.023493	2.682836e+07	115.000000	6.900000
max	3.800000e+08	13.130939	1.845034e+09	480.000000	10.000000

	vote_count
count	4865.000000
mean	266.885509
std	649.184148
min	0.000000
25%	3.000000
50%	37.000000
75%	232.000000
max	9653.000000

a. Distribution of numeric features

```
In [25]: fig, ((ax1, ax2), (ax3, ax4), (ax5, ax6)) = plt.subplots(3, 2, figsize=(15
         ax1.set_title('budget',fontsize=25)
         ax1=ax1.hist(numeric['budget'].values,color='yellowgreen',bins=30)
         ax2.set_title('popularity',fontsize=25)
         ax2=ax2.hist(numeric['popularity'].values,color='skyblue', bins=30)
         ax3.set_title('revenue',fontsize=25)
         ax3=ax3.hist(numeric['revenue'].values,color='crimson', bins=30)
         ax4.set_title('runtime',fontsize=25)
         ax4=ax4.hist(numeric['runtime'].values, color='gold',bins=30)
         ax5.set_title('vote_average',fontsize=25)
         ax5=ax5.hist(numeric['vote_average'].values,color='crimson', bins=30)
         ax6.set_title('vote_count',fontsize=25)
         ax6=ax6.hist(numeric['vote_count'].values, color='gold',bins=30)
         plt.show()
```



The distribution of budget, revenue, popularity and vote_count are highly right skewed.
The distribution of vote_average and runtime are approximately normal.
Log transformation or other kinds of transformation may need to be considered in future modeling.

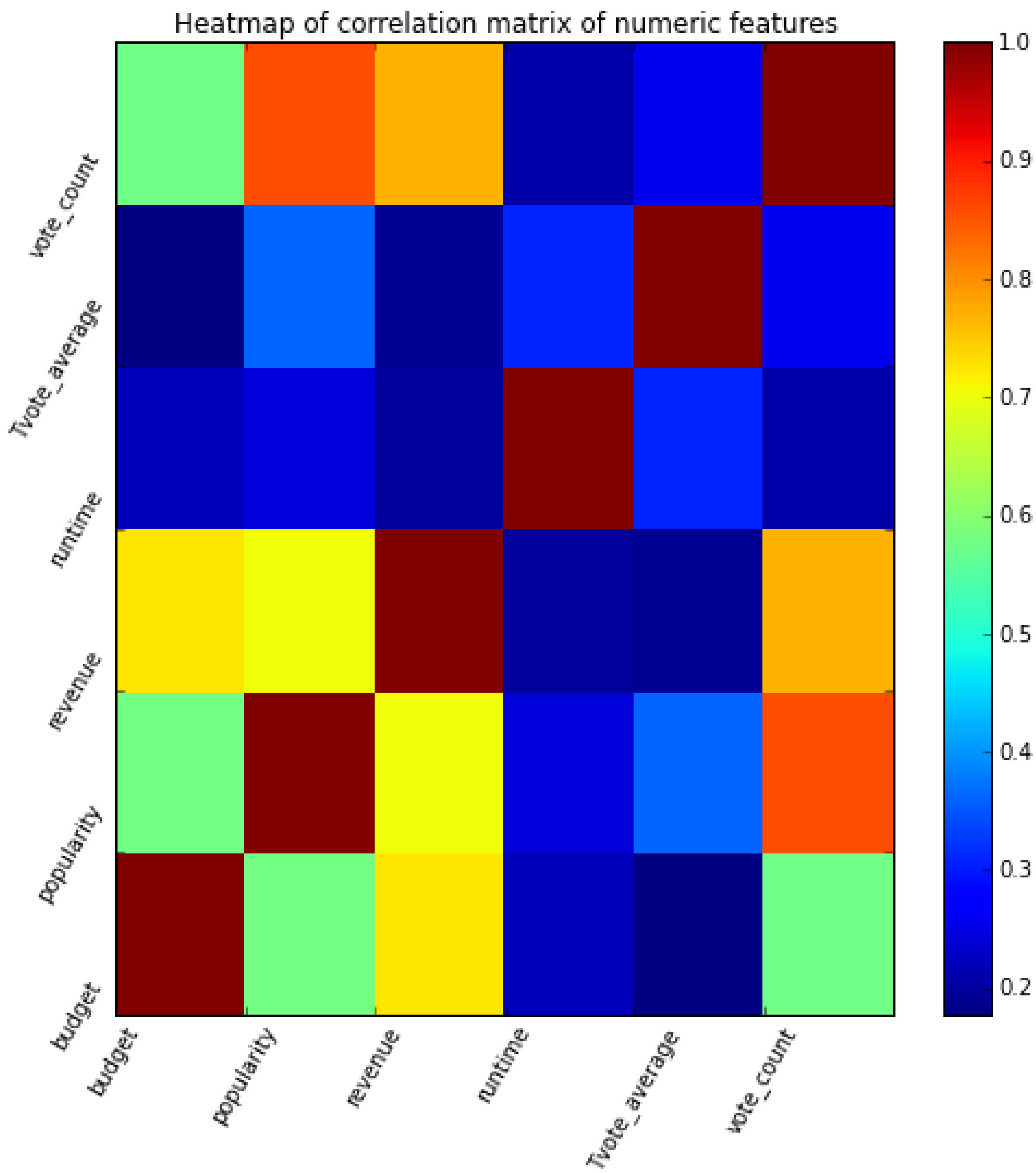
b. Correlation of numeric features

```
In [26]: corr = numeric.corr()

In [27]: corr

Out[27]:               budget  popularity   revenue   runtime  vote_average  \
         budget      1.000000    0.580444  0.724113  0.221215      0.177355
         popularity  0.580444    1.000000  0.701881  0.241776      0.362759
```

```
revenue       0.724113    0.701881  1.000000  0.199201      0.192514
runtime       0.221215    0.241776  0.199201  1.000000      0.310753
vote_average  0.177355    0.362759  0.192514  0.310753      1.000000
vote_count    0.575838    0.855649  0.770402  0.208134      0.259351

              vote_count
budget          0.575838
popularity      0.855649
revenue         0.770402
runtime         0.208134
vote_average    0.259351
vote_count      1.000000
```

In [28]:
```
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
ax.pcolor(corr)
ax.set_title('Heatmap of correlation matrix of numeric features')
plt.colorbar(ax.pcolor(corr))
ax.set_xticklabels(( 'budget', 'popularity', 'revenue', 'runtime', 'Tvote_
ax.set_yticklabels(( 'budget', 'popularity', 'revenue', 'runtime', 'Tvote_
plt.show()
```

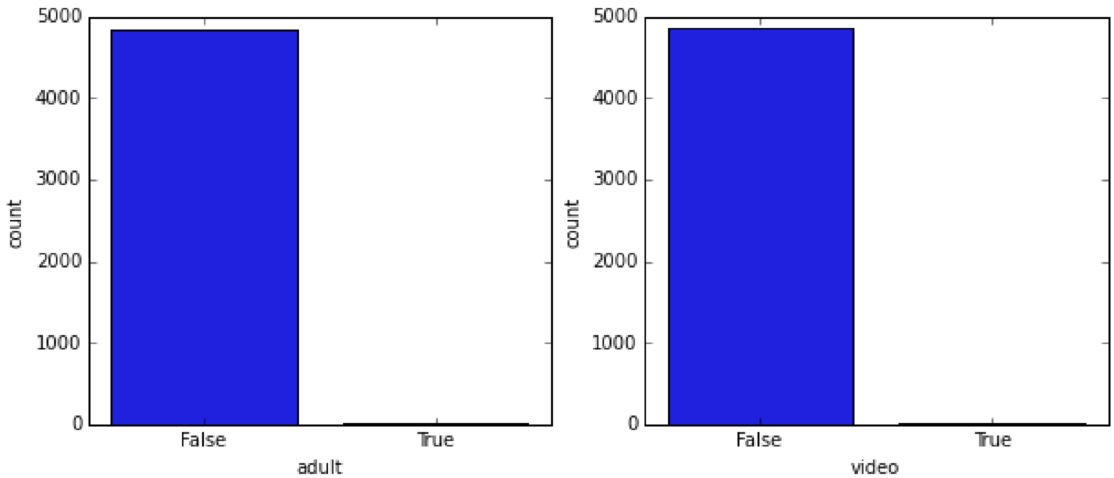
```
/Users/aixu/anaconda/lib/python2.7/site-packages/matplotlib/collections.py:590: Fut
  if self._edgecolors == str('face'):
```



Popularity and vote count are highly correlated. Revenue is also correlated with vote count. We need to pay more attention to these two pairs to avoid collinearity.

c. Dummy features

```
In [29]: fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,4))
         sns.countplot(x = 'adult', data=tmdb_movies, ax=ax1)
         sns.countplot(x = 'video', data=tmdb_movies, ax=ax2)
         plt.show()
```



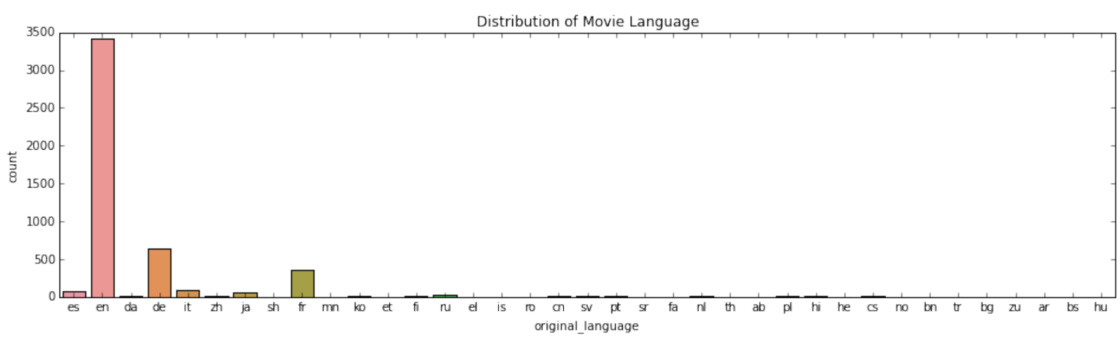
Most of the movie are not adult type, and no video provided. These two dummy features are not bery informative.

d. Categorical Features

```
In [30]: categorical = tmdb_movies.select_dtypes(include = ['object'])
         categorical.columns.values
```

```
Out[30]: array([u'backdrop_path', u'belongs_to_collection', u'genres', u'homepage',
                u'imdb_id', u'original_language', u'original_title', u'overview',
                u'poster_path', u'production_companies', u'production_countries',
                u'release_date', u'spoken_languages', u'status', u'tagline',
                u'title'], dtype=object)
```

```
In [31]: f, ax1 = plt.subplots(1,1, figsize = (16, 4))
         sns.countplot(x = 'original_language', data = categorical, ax=ax1)
         ax1.set_title("Distribution of Movie Language")
         plt.show()
```



Most movies have **English**(more than 3500) as their original language, while some other movies set **French**, **German**, **Italian** and **Spanish** as their original language.

5 3. EDA on genres

5.1 Questions that could be answered using TMDB:

1. What is the most common movie genre?
2. How many movie has more than one genre? What is the distribution?
3. What is the relationship between movie genre and popularity?
4. Which year had the largest number of movies released?
5. What are the highest rated science fiction movies?

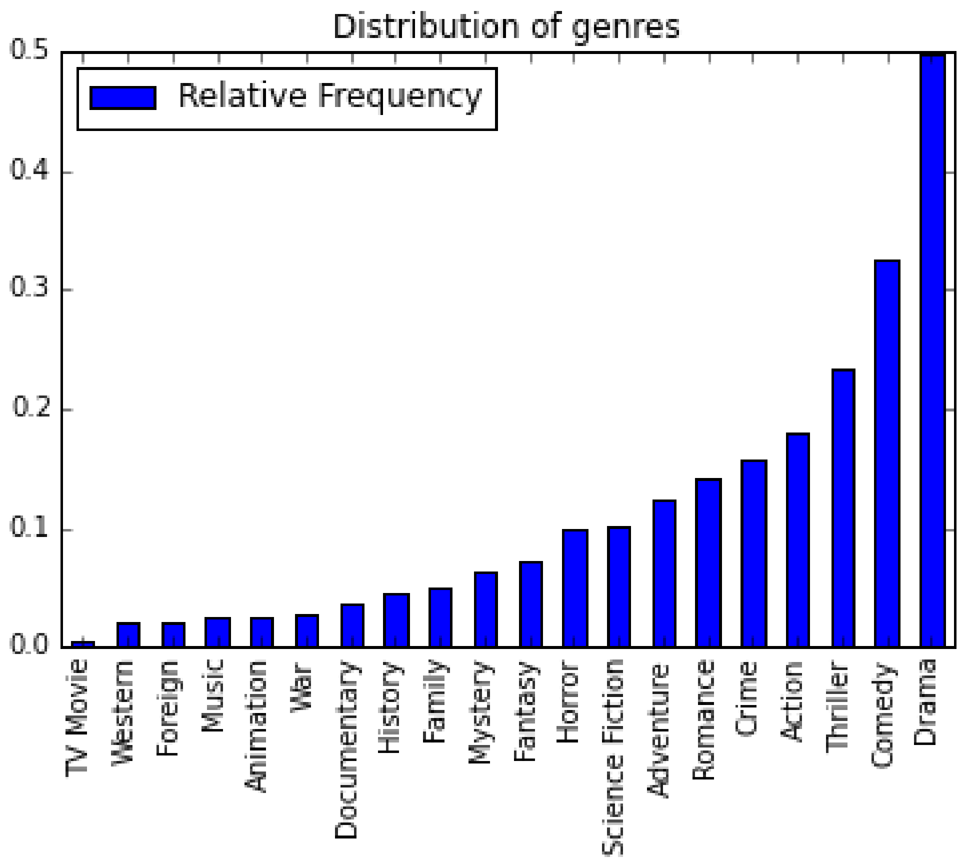
5.2 1) What is the most common genre?

In [32]:
```
tmdb_genre_df.columns.values[1]='Count'
```

In [33]:
```
tmdb_genre_df["Relative Frequency"]=tmdb_genre_df["Count"]/float(tmdb_movi
```

In [34]:
```
sort=tmdb_genre_df.sort_values(by="Relative Frequency")
```

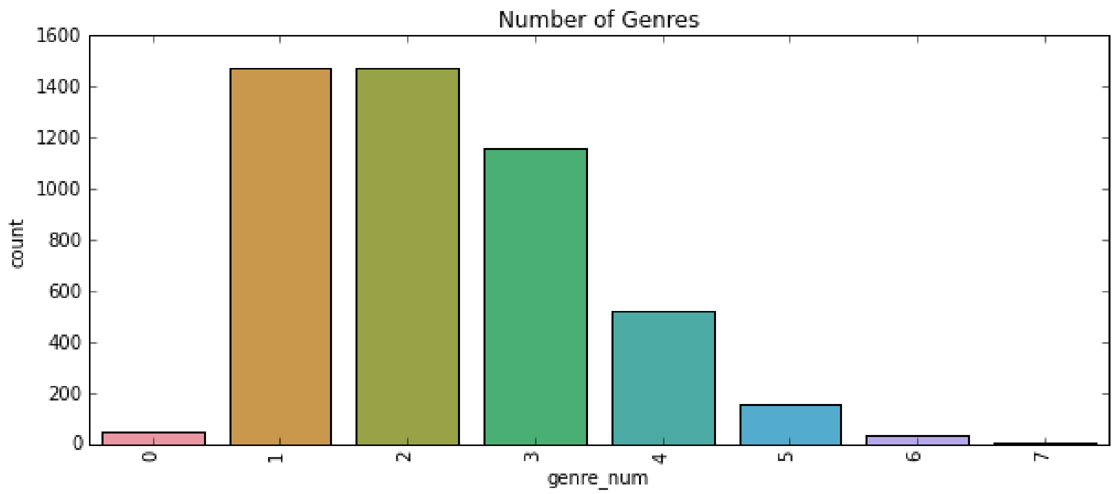
In [35]:
```
sort[["Relative Frequency"]].plot(kind='bar')
plt.title("Distribution of genres")
plt.show()
```



The most common movie genre is **Drama**
There are also a quite amont of **Comedy**, **Thriller** and **Action** movies in TMDB.

5.3 2) Distribution of Multiple Genres

In [36]:
```
fig, ax = plt.subplots(1, 1, figsize=(10,4))
ax = sns.countplot(x="genre_num", data=tmdb_movies)
xt = plt.xticks(rotation=90)
ax.set_title("Number of Genres")
plt.show()
```



Most of the movies has 1,2 or 3 genres. Also quite an amount of movies have 4 or more genres. Let's further investigate movies that have too many genres here:

In [52]:
```
print "Movie with 7 Genres"
tmdb_movies[tmdb_movies['genre_num']==7][['title', 'overview', 'genres', '
```

```
Movie with 7 Genres
```

Out[52]:
```
                                                   title  \
2322                                            Sneakers
2362                                           Westworld
3098   The Tulse Luper Suitcases, Part 2: Vaux to the...
8076                                              Tuvalu

                                                overview  \
2322   When shadowy U.S. intelligence agents blackmai...
2362   In a futuristic resort, wealthy patrons can vi...
3098   The Tulse Luper Suitcases reconstructs the lif...
8076   Set in a dilapidated indoor swimming pool (the...

                                                  genres  genre_num  \
2322   [Adventure, Drama, Action, Comedy, Thriller, C...          7
2362   [Action, Adventure, Drama, Horror, Science Fic...          7
```

```
3098  [War, Drama, History, Adventure, Comedy, Roman...          7
8076  [Fantasy, Drama, Comedy, Science Fiction, Roma...          7

      popularity release_date
2322    0.566098   1992-09-09
2362    1.516945   1973-10-22
3098    0.008423   2004-02-09
8076    0.032928   1999-11-19
```

According to the definition in Wikipedia: "Sneakers" is a comedy caper movie. "Westworld" is a science fiction Western thriller movie. "The Tulse Luper Suitcases" is a multimedia project , initially intended to comprise four films. "Tuvalu" an experimental movie.

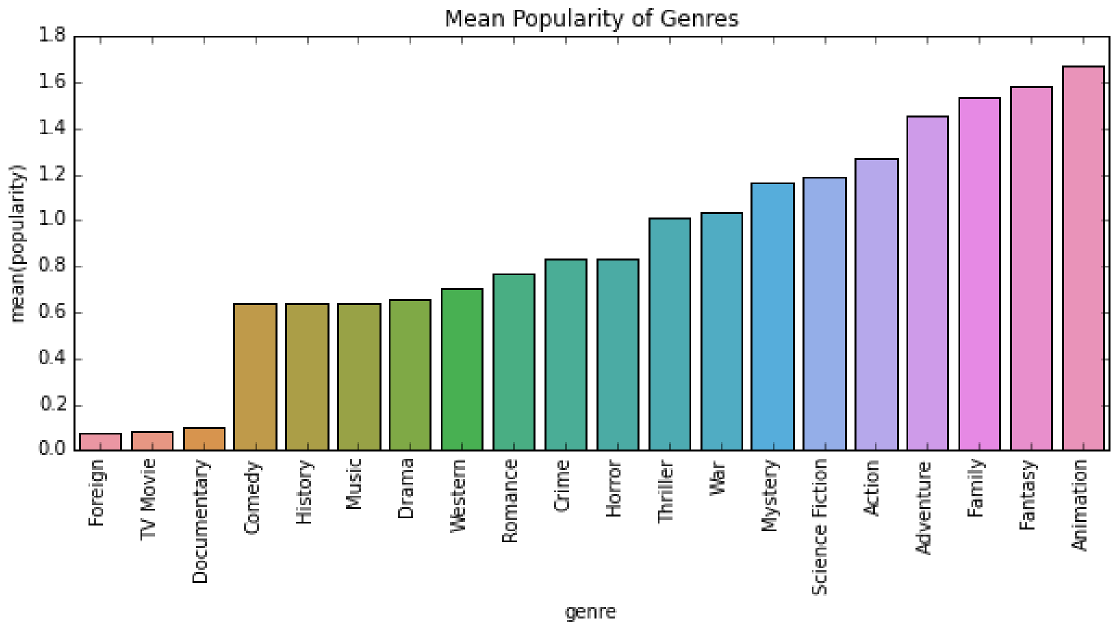
We can find that it is hard to determine the main genre of a movie when it contains a bunch of characterstics, like "Sneakers" and "Westworld", or it is an experimental movie that doesn't belong to any specific genre, or it is contains a series of movie of different genres.

5.4 3) What is the relationship between movie genre and popularity?

```
In [39]: meanpop=[]
         for genre in tmdb_genre:
             meanpop.append(tmdb_movies.groupby(genre)["popularity"].mean()[1])
```

```
In [40]: pop=pd.DataFrame(tmdb_genre)
         pop["popularity"]=meanpop
         pop=pop.sort_values(by='popularity')
         pop.columns.values[0]='genre'
```

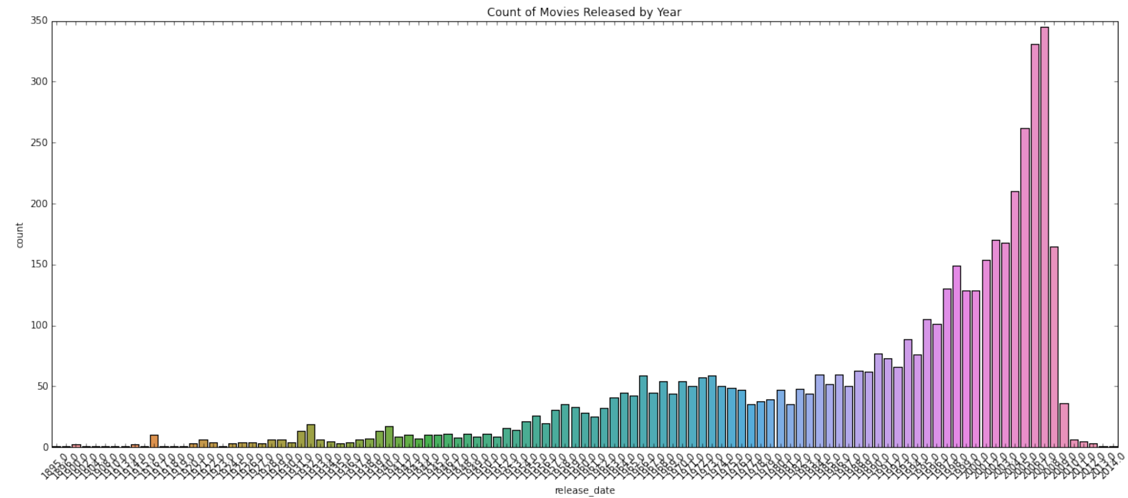
```
In [41]: fig, ax = plt.subplots(1, 1, figsize=(10,4))
         ax = sns.barplot(x='genre',y="popularity", data=pop)
         xt = plt.xticks(rotation=90)
         ax.set_title("Mean Popularity of Genres")
         plt.show()
```



Animation is the most popular movie genre. **Fantasy, Family, Adventure and Action** movie also receive high popularity. **Foreign, TV Movie and Documentary** has much lower popularity comparing to other genres.

5.5 4) Which year had the most number of movies released?

```
In [42]: tmdb_movies['release_date'].describe()

Out[42]: count              4865
         unique             3660
         top          2007-01-01
         freq                 19
         Name: release_date, dtype: object

In [43]: tmdb_movies['release_date'] = pd.to_datetime(tmdb_movies['release_date'],

In [44]: year = tmdb_movies['release_date'].dt.year

In [45]: fig, ax = plt.subplots(1, 1, figsize=(20,8))
         sns.countplot(x = year, ax=ax)
         ax.set_title("Count of Movies Released by Year")
         xt = plt.xticks(rotation=45)
```



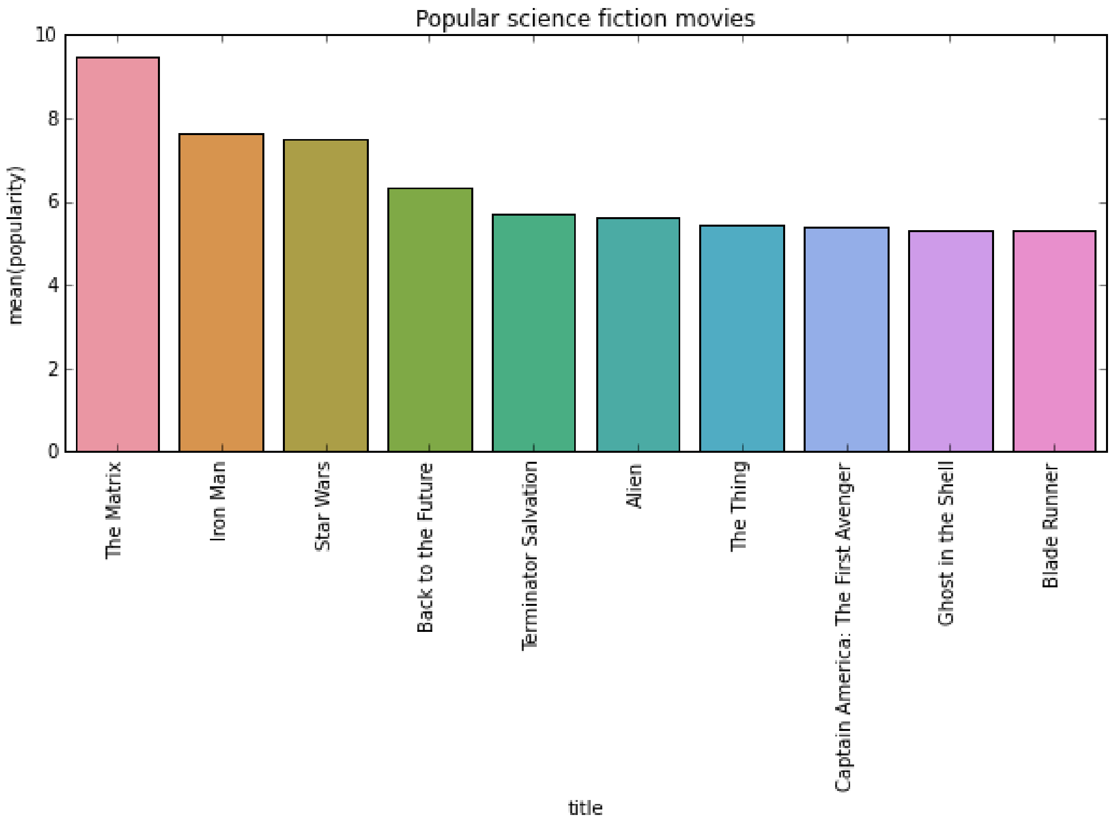
More than 300 movies were released in year 2005 and 2006. The number of movies are rockting in 21th century. (As we haven't load complete dataset in TMDB, the data for 2010-2016 are not complete)

5.6 5) What are the most popular science fiction movies?

```
In [53]: sf=tmdb_movies[tmdb_movies['Science Fiction']==True]
```

```
In [54]: top10=sf.sort_values(by=['popularity'], ascending=False).head(10)
```

```
In [55]: fig, ax = plt.subplots(1, 1, figsize=(10,4))
         ax = sns.barplot(x="title", y="popularity", data=top10)
         xt = plt.xticks(rotation=90)
         ax.set_title("Popular science fiction movies")
         plt.show()
```




```
In [58]: top10[['title','overview', 'popularity','production_countries', 'release_c
                'revenue', 'runtime', 'vote_average']].head(5)
```

```
Out[58]:                          title                                              overv
         603              The Matrix  Set in the 22nd century, The Matrix tells the
         1726               Iron Man  After being held captive in an Afghan cave, bi
         11                Star Wars  Princess Leia is captured and held hostage by
         105      Back to the Future  Eighties teenager Marty McFly is accidentally
         534    Terminator Salvation  All grown up in post-apocalyptic 2018, John Co

              popularity                           production_countries  \
         603    9.466876               [Australia, United States of America]
         1726   7.615126                          [United States of America]
         11     7.484705                          [United States of America]
         105    6.336256                          [United States of America]
         534    5.728618  [Germany, Italy, United Kingdom, United States...

             release_date    revenue  runtime  vote_average
         603   1999-03-30  463517383      136           7.8
         1726  2008-04-30  585174222      126           7.3
         11    1977-05-25  775398007      121           8.0
```

105	1985-07-03	381109762	116	7.9
534	2009-05-20	371353001	115	5.8

Not surprisingly, **The Matrix**, **Iron Man** and **Star Wars** are top 3 popular science fiction movies. All top 10 are very well-known movies released in last 30 years.

5.6.1 Problems for future investigation

1. **Correlated Covariates**: Since several pair of features are correlated, we need to deal with colinearity in the modeling part.
2. **Covariates with a set of values**: Some movie has multiple genres, production companies, production countries, etc. We need to come up with a way to transform these features in to useful predictors, at the same time not losing information. One possible method is to create dummy indicators, for example an indicator for whether the movies is produced by U.S or not.
3. **Skewed Distribution**: Several numeric features have highly skewed distribution. We need to determine whether to transform them or not in modeling.
4. **Missing Value Imputation**: Need to determine whether to impute the missing value, and what method to use.
5. **Genre Dependency**: Since some movie have more than one genre, we are curious about the dependecy between genres. For example, some genres are likely to appear together with another for a movie. We need to consider the depedency in modeling part.

Milestone1_06_EDA_IMDB

April 5, 2017

```
In [147]: %matplotlib inline
          import pandas as pd
          import numpy as np
          import json
          import matplotlib.pyplot as plt
          import seaborn.apionly as sns
          import re
```

1 Load JSON File

```
In [34]: with open('/Users/Victoria_G/Desktop/CS109B/proj/movie_project/data/drv_im
             data = json.load(f)
         imdb_movies=pd.DataFrame(data)
```

2 Format the data

```
In [35]: def get_genre(tmdb_movies ,key):
             tmdb_genre = tmdb_movies[key].tolist()
             tmdb_genre_set = set()
             for g in tmdb_genre:
                 tmdb_genre_set = tmdb_genre_set.union(set(g))
             tmdb_genre = list(tmdb_genre_set)
             tmdb_genre.sort()
             return(tmdb_genre)

In [36]: def is_genre (row, column_name, genre):
             """check if that movie is in this genre as a movie can have more than
             if genre in row[column_name] :
                 return True
             else:
                 return False

In [37]: # as the genre is in list format in the data field, we cannot utilize any
         # thus, we have to transform data set

         def add_genre_columns(tmdb_movies, genre_column):
             tmdb_genre = get_genre(tmdb_movies,genre_column)
```

```
tmdb_movies[u'genre_num'] = tmdb_movies.apply(lambda row: len(row[genr

for g in tmdb_genre:
    tmdb_movies[g] = tmdb_movies.apply(lambda row: is_genre(row,genre_
```

In [38]:
```
imdb_genre = get_genre(imdb_movies,u'genre')
imdb_movies[u'genre_num'] = imdb_movies.apply(lambda row: len(row[u'genre'

for g in imdb_genre:
    imdb_movies[g] = imdb_movies.apply(lambda row: is_genre(row,u'genre',g
```

In [148]:
```
imdb_movies.head()
```

Out[148]:
```
                                              actors  \
0   [{u'personID': u'0000552', u'name': u'Eddie Mu...
1   [{u'personID': u'0000160', u'name': u'Ethan Ha...
10  [{u'personID': u'0000136', u'name': u'Johnny D...
11  [{u'personID': u'0000235', u'name': u'Uma Thur...
12  [{u'personID': u'0671231', u'name': u'Matti Pe...

                                           actresses  \
0   [{u'personID': u'0000552', u'name': u'Eddie Mu...
1   [{u'personID': u'0000160', u'name': u'Ethan Ha...
10  [{u'personID': u'0000136', u'name': u'Johnny D...
11  [{u'personID': u'0000235', u'name': u'Uma Thur...
12  [{u'personID': u'0671231', u'name': u'Matti Pe...

                                                 aka  \
0   [Un detective suelto en Hollywood::Argentina, ...
1   [Antes del atardecer::Argentina, International...
10  [P.O.T.C. 2::USA (promotional abbreviation), P...
11  [Kill Bill::USA (informal short title), Kill B...
12  [Shadows in Paradise::International (English t...

                                    also known as amazon review  \
0   [Un detective suelto en Hollywood::Argentina, ...          None
1   [Antes del atardecer::Argentina, International...          None
10  [P.O.T.C. 2::USA (promotional abbreviation), P...          None
11  [Kill Bill::USA (informal short title), Kill B...          None
12  [Shadows in Paradise::International (English t...          None

                                     art direction by  \
0   [{u'personID': u'0613468', u'name': u'James J...
1                                                None
10  [{u'personID': u'0188692', u'name': u'Bruce Cr...
11  [{u'personID': u'0103011', u'name': u'Daniel B...
12  [{u'personID': u'0383991', u'name': u'Pertti H...
```

```
                                              casting  \
0   [{u'personID': u'0799557', u'name': u'Margery ...
1                                                None
10  [{u'personID': u'0150522', u'name': u'Denise C...
11  [{u'personID': u'0535338', u'name': u'Koko Mae...
12                                               None

                                           casting by  \
0   [{u'personID': u'0799557', u'name': u'Margery ...
1                                                None
10  [{u'personID': u'0150522', u'name': u'Denise C...
11  [{u'personID': u'0535338', u'name': u'Koko Mae...
12                                               None

                                          certificate  \
0   [Argentina:16, Australia:M, Brazil:12, Canada:...
1   [Argentina:Atp, Australia:M, Austria:0, Brazil...
10  [Argentina:13, Australia:M, Brazil:12, Canada:...
11  [Argentina:16, Australia:R18+, Brazil:18, Cana...
12  [Finland:S, Iceland:L, Singapore:PG, UK:12::(v...

                                        certification     ...     Musical
0   [Argentina:16, Australia:M, Brazil:12, Canada:...     ...       False
1   [Argentina:Atp, Australia:M, Austria:0, Brazil...     ...       False
10  [Argentina:13, Australia:M, Brazil:12, Canada:...     ...       False
11  [Argentina:16, Australia:R18+, Brazil:18, Cana...     ...       False
12  [Finland:S, Iceland:L, Singapore:PG, UK:12::(v...     ...       False

   Romance Sci-Fi  Short  Sport Thriller    War runtime_avg major_country
0    False  False  False  False    False  False       105.0           USA
1     True  False  False  False    False  False        80.0           USA
10   False  False  False  False    False  False       151.0           USA
11   False  False  False  False     True  False       112.0           USA
12    True  False  False  False    False  False        76.0       Finland

   casting_num
0          2.0
1          NaN
10         1.0
11         2.0
12         NaN

[5 rows x 109 columns]
```

3 Data Pre-Processing

3.1 1. Average runtime

Since each movie may have different runtime in different countries. The basic step here is first use regular expression to extract the number and then calculate the average runtime for each movie.

```
In [64]: # average run time
         L=[]
         for row in imdb_movies['runtime']:
             for e in row:
                 number=[]
                 number.extend(map(int,re.findall('\d+',str(e))))
             L.append(1.0* sum(number)/len(number))
         imdb_movies['runtime_avg']=L
```

3.2 2. Major Country

The basic idea here is that each movie may be produced by multi-countries. We assume the major porduce country is the first country in the list.

```
In [72]: L=[]
         for e in imdb_movies['country']:
             L.append(str(e[0]))
         imdb_movies['major_country']=L
```

3.3 3. Casting number

```
In [79]: genre=list(imdb_movies.ix[:,'Action':'War'].columns.values)
         genre

Out[79]: [u'Action',
          u'Adventure',
          u'Animation',
          u'Biography',
          u'Comedy',
          u'Crime',
          u'Documentary',
          u'Drama',
          u'Family',
          u'Fantasy',
          u'Horror',
          u'Music',
          u'Musical',
          u'Romance',
          u'Sci-Fi',
          u'Short',
          u'Sport',
          u'Thriller',
          u'War']
```

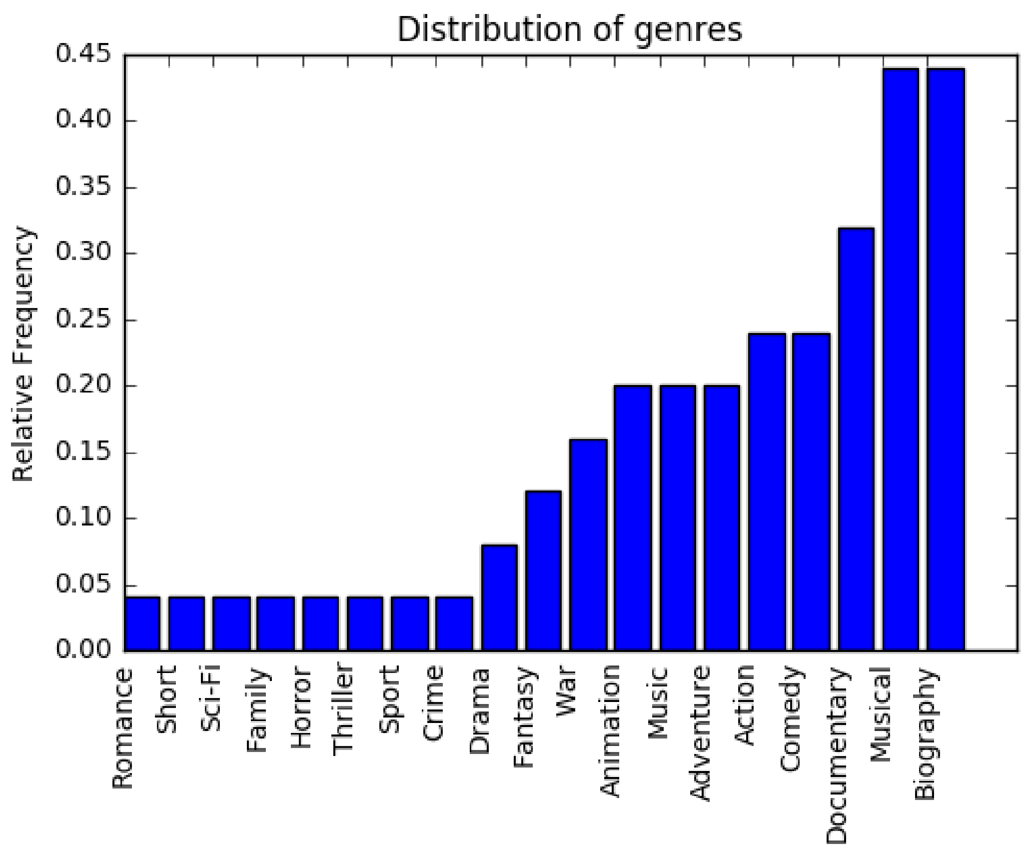
```
In [92]: L=[]
         for e in imdb_movies['casting']:
             if e != None:
                 L.append(len(e))
             else:
                 L.append(None)
         imdb_movies['casting_num']=L
```

4 Insight 1: Distribution of the genres

As we can see, there are in total 19 genres in the data set. The most popular genre types are musical and biography type. And documentary and comedy are also very popolar types.

```
In [119]: dic={}
          for g in genre:
              dic[str(g)]= sum(imdb_movies[str(g)])
          dic

Out[119]: {'Action': 44,
           'Adventure': 44,
           'Animation': 12,
           'Biography': 4,
           'Comedy': 20,
           'Crime': 20,
           'Documentary': 4,
           'Drama': 32,
           'Family': 4,
           'Fantasy': 16,
           'Horror': 4,
           'Music': 4,
           'Musical': 4,
           'Romance': 24,
           'Sci-Fi': 24,
           'Short': 8,
           'Sport': 4,
           'Thriller': 20,
           'War': 4}
```

```
In [143]: #fig=figsize(5,5)
          ind=range(len(genre))
          plt.bar(ind, np.array(sorted(dic.values()))/100.)
          plt.xticks(ind, dic.keys(),rotation='vertical')
          plt.ylabel("Relative Frequency")
          plt.title("Distribution of genres")

Out[143]: <matplotlib.text.Text at 0x126bd4950>
```



5 Insight 2: Genre vs. Average casting number

In this part, I calculated the average of the casting number for each genre first and then draw the distribution of of it. We can see that from the distribution, biography tend to have the largest aberage casting number. Horrer tend to have the smallest average casting number

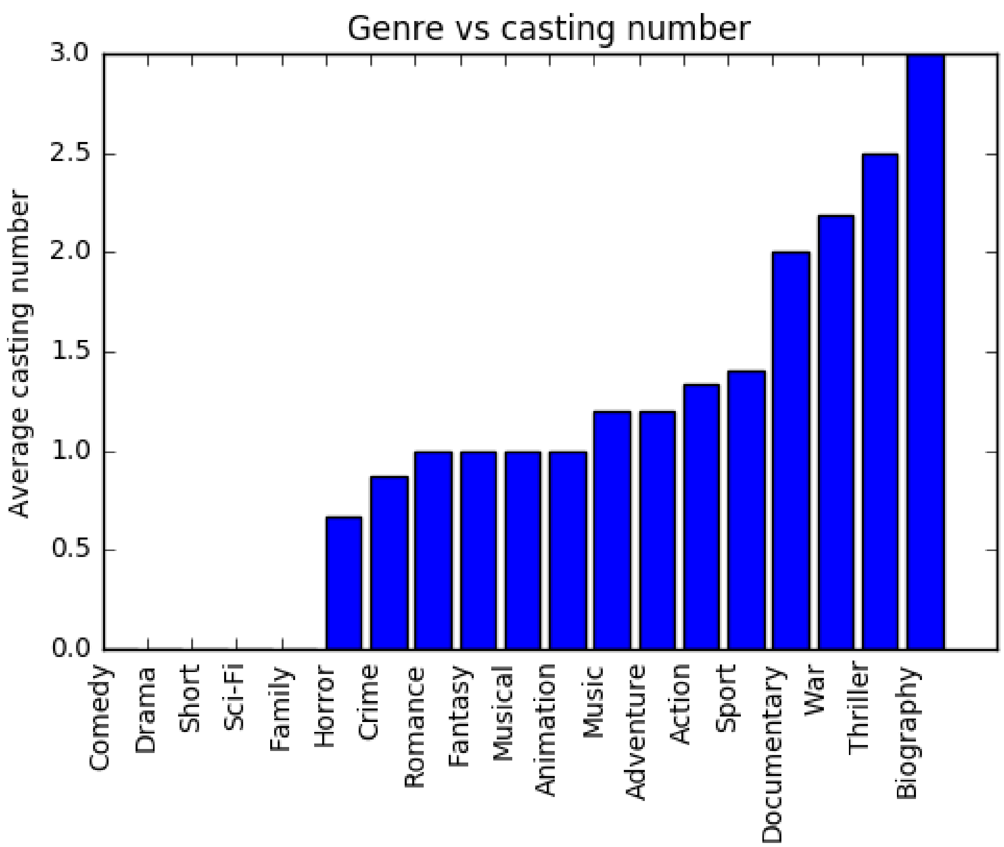
In [115]:
```
dic_casting={}
for g in dic.keys():
    df_temp=imdb_movies[imdb_movies[g]==True]
    dic_casting[g]= df_temp['casting_num'].sum()*1./len(df_temp['casting_
dic_casting
```

Out[115]:
```
{'Action': 2.1818181818181817,
 'Adventure': 2.0,
 'Animation': 1.0,
 'Biography': 0.0,
 'Comedy': 1.2,
 'Crime': 1.2,
 'Documentary': 0.0,
 'Drama': 0.875,
```

```
 'Family': 3.0,
 'Fantasy': 2.5,
 'Horror': 0.0,
 'Music': 1.0,
 'Musical': 1.0,
 'Romance': 0.6666666666666666,
 'Sci-Fi': 1.3333333333333333,
 'Short': 0.0,
 'Sport': 0.0,
 'Thriller': 1.4,
 'War': 1.0}
```

In [117]:
```
#fig=figsize(5,5)
ind=range(len(genre))
plt.bar(ind, sorted(dic_casting.values()))
plt.xticks(ind, dic_casting.keys(),rotation='vertical')
plt.ylabel("Average casting number")
plt.title("Genre vs casting number")
```

Out[117]: <matplotlib.text.Text at 0x11fd72150>



6 Insight 3: Genre vs Average number of runtime

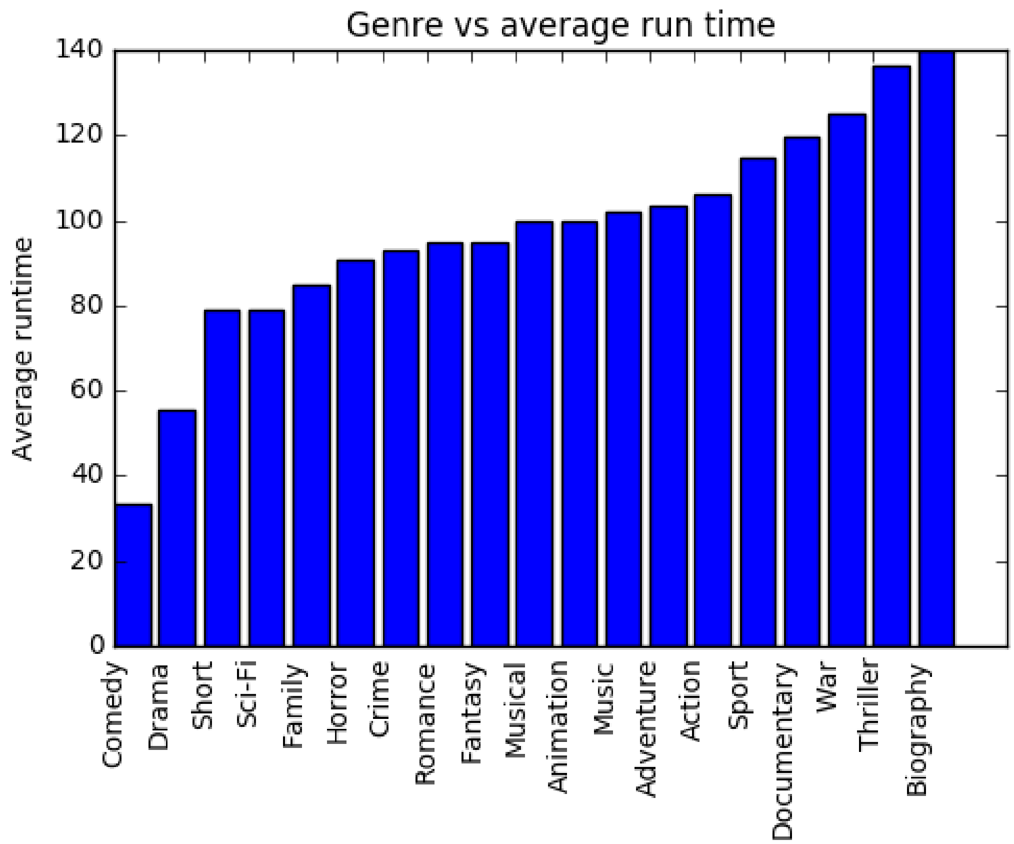
As we can see from the plot below, Thriller and biography seems to have the longest average runtime.

```
In [123]: dic_runtime={}
          for g in dic.keys():
              df_temp=imdb_movies[imdb_movies[g]==True]
              dic_runtime[g]=df_temp['runtime_avg'].sum()*1./len(df_temp['runtime_a
          dic_runtime
```

```
Out[123]: {'Action': 119.72727272727273,
           'Adventure': 114.54545454545455,
           'Animation': 55.666666666666664,
           'Biography': 79.0,
           'Comedy': 90.8,
           'Crime': 106.0,
           'Documentary': 95.0,
           'Drama': 102.0,
           'Family': 100.0,
           'Fantasy': 136.5,
           'Horror': 79.0,
           'Music': 100.0,
           'Musical': 140.0,
           'Romance': 84.83333333333333,
           'Sci-Fi': 93.16666666666667,
           'Short': 33.5,
           'Sport': 95.0,
           'Thriller': 103.2,
           'War': 125.0}
```

```
In [130]: #fig=figsize(5,5)
          ind=range(len(genre))
          plt.bar(ind, sorted(dic_runtime.values()))
          plt.xticks(ind, dic_casting.keys(),rotation='vertical')
          plt.ylabel("Average runtime")
          plt.title("Genre vs average run time")
```

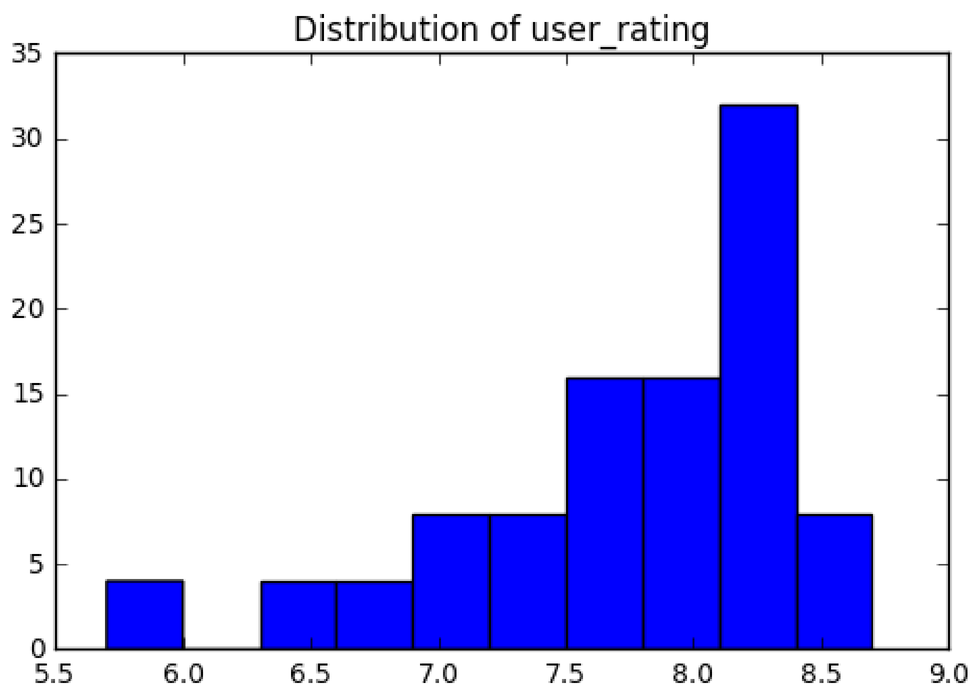
```
Out[130]: <matplotlib.text.Text at 0x121e97f50>
```



7 Insight 4: User rating

1. ** Distribution of user rating**: As we can see from the distribution plot, we found that the distribution is a little bit skewed. Most of movies has rating around 8 to 8.5.
2. ** Average User rating vs. genre **: As we can see from the distribution, it is obvious that comedy and drama seems to have the lowest user rating, whereas most of other genres have the similar ratings but not the same. They are all above 7 and less than 9.

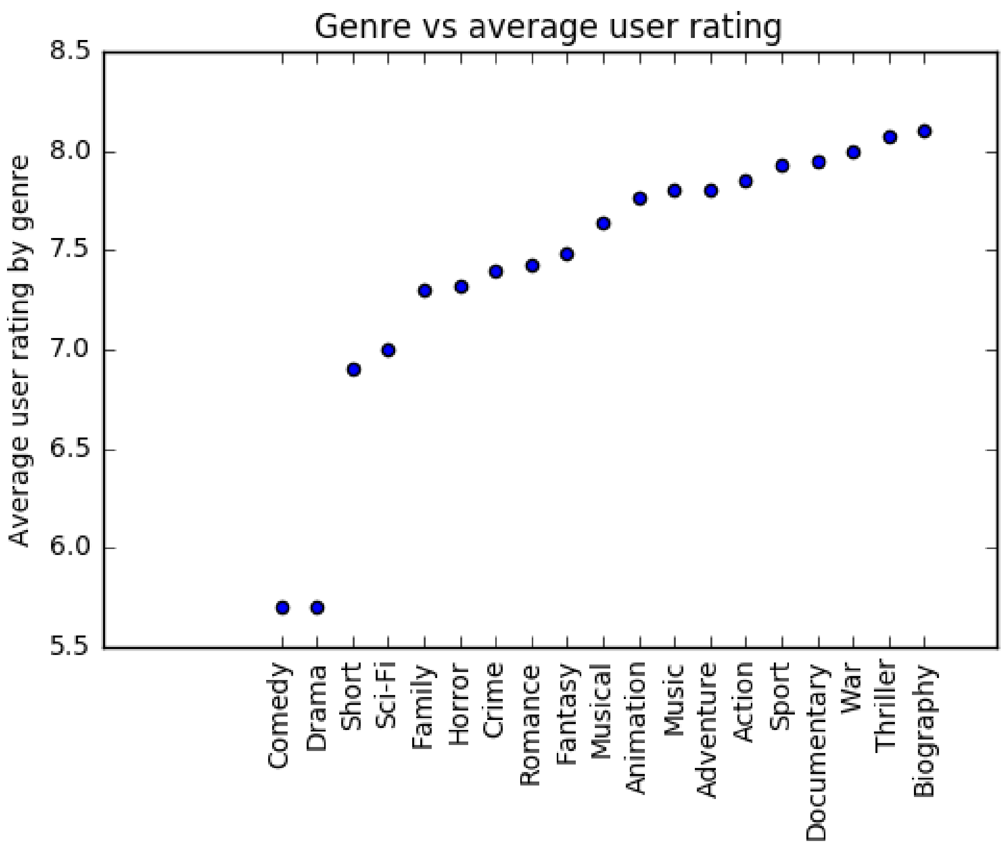
```
In [142]: plt.hist(imdb_movies['user rating'].values, bins=10)
          plt.title('Distribution of user_rating')

Out[142]: <matplotlib.text.Text at 0x126a26b10>
```




```
In [144]: dic_rating={}
          for g in dic.keys():
              df_temp=imdb_movies[imdb_movies[g]==True]
              dic_rating[g]=df_temp['user rating'].sum()*1./len(df_temp['user ratin
          dic_rating

Out[144]: {'Action': 7.636363636363639,
           'Adventure': 7.7636363636363654,
           'Animation': 7.933333333333333,
           'Biography': 5.7,
           'Comedy': 7.3999999999999999,
           'Crime': 7.4799999999999999,
           'Documentary': 7.8,
           'Drama': 7.4249999999999998,
           'Family': 8.1,
           'Fantasy': 8.075,
           'Horror': 5.7,
           'Music': 6.9,
           'Musical': 8.0,
           'Romance': 7.3166666666666655,
           'Sci-Fi': 7.9499999999999998,
           'Short': 7.8500000000000005,
           'Sport': 7.8,
           'Thriller': 7.2999999999999999,
           'War': 7.0}
```

```
In [149]: ind=range(len(genre))
          plt.scatter(ind, sorted(dic_rating.values()))
          plt.xticks(ind, dic_casting.keys(),rotation='vertical')
          plt.ylabel("Average user rating by genre")
          plt.title("Genre vs average user rating")
```

```
Out[149]: <matplotlib.text.Text at 0x127ea8510>
```



8 Major problems

1. The major problem here is that the value of each covariate here is not a single value. Instead, there are lists of values for each cell of the data frame. This could be very difficult for us to do exploratory data analysis. Also, it would very hard to fit machine learning models on the existing model. So we should do some pre-data processing to deal with this issue.
2. There are several ways that we came up with to deal with this multi-value cell issue.

- The first one is that we create new indicator variables. For example, the value of the variable “country” may have more than one values. According to the data, we found that most of the movie comes from the US. So we set a new indicator variable indicating whether this movies is from US. In addition, we could set a indicator variable called “global”, indicate whether this movie is produced by multi-countries.

- The second one is to choose some of the variables in multi-value value covariates. For example, we have the casting column having lots of actors/actress. Since from the documentation, every list here is ordered by the importance of the casting. So we may set the threshold of the number of the casting we want later.

3. **Correlated covariates**: There are lots of covariates in the dataset, that are highly correlated. For example, the "casting" and "casting by" are actually the same. So we may next examine the highly correlated covariates.

9 Future work

1. We should come up a way to quantify the label dependency. That is, we should find a way to measure the dependence between each labels. We may want to generate a occurrence matrix to try to see the related labels.
2. This IMDB data is not our final dataset. We are working on getting more clear-formatted data later and we will do the new EDA in the milestone 2.

```
In [ ]:
```

Missing Value Analysis

Yufei Gui

4/4/2017

```r
#load data
dat<- read.csv('imdb_raw.csv')

# change all the empty cell to NA form
dat[dat==""]<-NA

dat<- data.frame(dat)

# number of the total observations and columns
nrow(dat)
```

```
## [1] 100
```

```r
ncol(dat)
```

```
## [1] 87
```

```r
# explore the type of the input variables
variables<-split(names(dat),sapply(dat, function(x) paste(class(x), collapse=" ")))
variables
```

```
## $factor
##  [1] "actors"
##  [2] "actresses"
##  [3] "aka"
##  [4] "also.known.as"
##  [5] "art.direction.by"
##  [6] "casting"
##  [7] "casting.by"
##  [8] "certificate"
##  [9] "certification"
## [10] "certifications"
## [11] "cinematography"
## [12] "cinematography.by"
## [13] "color"
## [14] "costume.and.wardrobe.department"
## [15] "costume.design"
## [16] "costume.design.by"
## [17] "country"
## [18] "cover"
## [19] "crew.members"
## [20] "crewmembers"
## [21] "directed.by"
## [22] "distribution"
## [23] "distribution.companies"
## [24] "distribution.company"
## [25] "distributor"
## [26] "editing"
## [27] "film.editing"
```

```
## [28] "film.editing.by"
## [29] "genre"
## [30] "lang"
## [31] "language"
## [32] "make.up"
## [33] "makeup"
## [34] "makeup.department"
## [35] "misc.companies"
## [36] "misc.company"
## [37] "misc.crew"
## [38] "miscellaneous.company"
## [39] "miscellaneouscrew"
## [40] "music"
## [41] "original.music.by"
## [42] "other.companies"
## [43] "other.company"
## [44] "other.crew"
## [45] "plot.summaries"
## [46] "plot.summary"
## [47] "produced.by"
## [48] "production.company"
## [49] "production.countries"
## [50] "production.country"
## [51] "production.management"
## [52] "runtime"
## [53] "second.unit.director"
## [54] "second.unit.director.or.assistant.director"
## [55] "set.decoration.by"
## [56] "sound.department"
## [57] "special.effects.company"
## [58] "stunts"
## [59] "visual.effects.by"
## [60] "writing.credits"
##
## $integer
## [1] "X"       "imdb_id"
##
## $logical
##  [1] "amazon.review"                "created.by"
##  [3] "episodes.cast"                "episodes.number"
##  [5] "faq"                          "frequently.asked.questions"
##  [7] "full.size.cover"              "guest"
##  [9] "guest.appearances"            "merchandise"
## [11] "merchandising"                "miscellaneous"
## [13] "miscellaneous.links"          "non.original.music.by"
## [15] "notable.tv.guest.appearances" "parental.guide"
## [17] "photographs"                  "sales"
## [19] "seasons"                      "soundclips"
## [21] "special.effects.by"           "tv.guests"
## [23] "tv.schedule"                  "videoclips"
##
## $numeric
## [1] "user.rating"
```

```r
summary(variables)
```

```
##         Length Class  Mode
## factor  60     -none- character
## integer  2     -none- character
## logical 24     -none- character
## numeric  1     -none- character
```

```r
# explore missing rate of each variables

# define a function of explore the missing rate
propmiss <- function(dataframe) {
    m <- sapply(dataframe, function(x) {
        data.frame(
            nmiss=sum(is.na(x)),
            n=length(x),
            propmiss=sum(is.na(x))/length(x)
        )
    })
    d <- data.frame(t(m))
    d <- sapply(d, unlist)
    d <- as.data.frame(d)
    d$variable <- row.names(d)
    row.names(d) <- NULL
    d <- cbind(d[ncol(d)],d[-ncol(d)])
    return(d[order(d$propmiss), ])
}

# missing rate for the train set
propmiss(dat)
```

```
##                                       variable nmiss   n propmiss
## 1                                            X     0 100     0.00
## 2                                       actors     0 100     0.00
## 3                                    actresses     0 100     0.00
## 4                                          aka     0 100     0.00
## 5                                also.known.as     0 100     0.00
## 10                                 certificate     0 100     0.00
## 11                               certification     0 100     0.00
## 12                              certifications     0 100     0.00
## 15                                       color     0 100     0.00
## 19                                     country     0 100     0.00
## 20                                       cover     0 100     0.00
## 24                                 directed.by     0 100     0.00
## 37                                       genre     0 100     0.00
## 40                                     imdb_id     0 100     0.00
## 41                                        lang     0 100     0.00
## 42                                    language     0 100     0.00
## 64                              plot.summaries     0 100     0.00
## 65                                plot.summary     0 100     0.00
## 67                          production.company     0 100     0.00
## 68                        production.countries     0 100     0.00
## 69                          production.country     0 100     0.00
## 71                                     runtime     0 100     0.00
```

```
## 84                                     user.rating    0 100      0.00
## 87                                 writing.credits    0 100      0.00
## 66                                     produced.by    4 100      0.04
## 13                                  cinematography    8 100      0.08
## 14                               cinematography.by    8 100      0.08
## 25                                    distribution    8 100      0.08
## 26                          distribution.companies    8 100      0.08
## 27                            distribution.company    8 100      0.08
## 28                                     distributor    8 100      0.08
## 29                                         editing    8 100      0.08
## 33                                    film.editing    8 100      0.08
## 34                                 film.editing.by    8 100      0.08
## 22                                    crew.members   12 100      0.12
## 23                                     crewmembers   12 100      0.12
## 48                                  misc.companies   12 100      0.12
## 49                                    misc.company   12 100      0.12
## 50                                       misc.crew   12 100      0.12
## 52                           miscellaneous.company   12 100      0.12
## 54                               miscellaneouscrew   12 100      0.12
## 59                                 other.companies   12 100      0.12
## 60                                   other.company   12 100      0.12
## 61                                      other.crew   12 100      0.12
## 77                                sound.department   12 100      0.12
## 55                                           music   16 100      0.16
## 58                               original.music.by   16 100      0.16
## 70                           production.management   16 100      0.16
## 74                            second.unit.director   20 100      0.20
## 75 second.unit.director.or.assistant.director   20 100      0.20
## 43                                         make.up   24 100      0.24
## 44                                          makeup   24 100      0.24
## 45                               makeup.department   24 100      0.24
## 8                                          casting   28 100      0.28
## 9                                       casting.by   28 100      0.28
## 16                 costume.and.wardrobe.department   28 100      0.28
## 17                                  costume.design   28 100      0.28
## 18                               costume.design.by   28 100      0.28
## 7                                 art.direction.by   32 100      0.32
## 86                               visual.effects.by   32 100      0.32
## 81                                          stunts   36 100      0.36
## 76                               set.decoration.by   40 100      0.40
## 80                         special.effects.company   48 100      0.48
## 6                                    amazon.review  100 100      1.00
## 21                                      created.by  100 100      1.00
## 30                                   episodes.cast  100 100      1.00
## 31                                 episodes.number  100 100      1.00
## 32                                             faq  100 100      1.00
## 35                      frequently.asked.questions  100 100      1.00
## 36                                 full.size.cover  100 100      1.00
## 38                                           guest  100 100      1.00
## 39                               guest.appearances  100 100      1.00
## 46                                     merchandise  100 100      1.00
## 47                                   merchandising  100 100      1.00
## 51                                   miscellaneous  100 100      1.00
## 53                             miscellaneous.links  100 100      1.00
```

```
## 56                non.original.music.by   100 100      1.00
## 57          notable.tv.guest.appearances   100 100      1.00
## 62                        parental.guide   100 100      1.00
## 63                           photographs   100 100      1.00
## 72                                 sales   100 100      1.00
## 73                               seasons   100 100      1.00
## 78                            soundclips   100 100      1.00
## 79                    special.effects.by   100 100      1.00
## 82                             tv.guests   100 100      1.00
## 83                           tv.schedule   100 100      1.00
## 85                            videoclips   100 100      1.00
```