Final project

Group 6: Sangmin Choi, Elif Kara, Luke Krawec, Aishwarya Srivastava

Project Overview and Objective

In our project, we investigated ways to predict movie interests of consumers, primarily in the NYC area (though findings could extend beyond NYC). We combined demographic and favorite movie data from Facebook and joined it with movie names at IMDB.

The ability to predict latent movie interests—by combining tangential, more easily accessible data on background, interests and preferences—would be very valuable for movie studios as well as outlets like Netflix and Amazon Prime, whose ability to appeal to viewers depends on their ability to understand user preferences.*

**Information sourced primarily from project proposal.*

Getting the data

We used IMDB and NYU students' Facebook databases that exist in the current database. We installed sql_magic to run SQL queries within a Jupyter notebook. We also installed matplotlib, numpy, pandas to help with our analysis.

In [0]:

```
!sudo pip3 install -U sql_magic
```

```
The directory '/home/hek303/.cache/pip/http' or its parent directory i
s not owned by the current user and the cache has been disabled. Pleas
e check the permissions and owner of that directory. If executing pip
 with sudo, you may want sudo's -H flag.
The directory '/home/hek303/.cache/pip' or its parent directory is not
owned by the current user and caching wheels has been disabled. check
 the permissions and owner of that directory. If executing pip with su
do, you may want sudo's -H flag.
Collecting sql_magic
  Downloading https://files.pythonhosted.org/packages/a6/a7/ccdc67278d
e3f34db5d484f9b6c59ad9a536beda4a5acad5ecb3b0932246/sql_magic-0.0.4-py3
-none-any.whl (https://files.pythonhosted.org/packages/a6/a7/ccdc67278
de3f34db5d484f9b6c59ad9a536beda4a5acad5ecb3b0932246/sql_magic-0.0.4-py
3-none-any.whl)
Requirement not upgraded as not directly required: findspark in /usr/l
ocal/lib/python3.6/dist-packages (from sql_magic) (1.2.0)
Requirement not upgraded as not directly required: jupyter in /usr/loc
al/lib/python3.6/dist-packages (from sql_magic) (1.0.0)
Requirement not upgraded as not directly required: sqlparse in /usr/lo
cal/lib/python3.6/dist-packages (from sql_magic) (0.2.4)
Requirement not upgraded as not directly required: traitlets in /usr/l
ocal/lib/python3.6/dist-packages (from sql_magic) (4.3.2)
Requirement not upgraded as not directly required: pandas in /usr/loca
l/lib/python3.6/dist-packages (from sql_magic) (0.23.0)
Requirement not upgraded as not directly required: ipython in /usr/loc
al/lib/python3.6/dist-packages (from sql_magic) (6.2.1)
Requirement not upgraded as not directly required: qtconsole in /usr/l
ocal/lib/python3.6/dist-packages (from jupyter->sql_magic) (4.3.1)
Requirement not upgraded as not directly required: nbconvert in /usr/l
ocal/lib/python3.6/dist-packages (from jupyter->sql_magic) (5.3.1)
Requirement not upgraded as not directly required: ipykernel in /usr/l
ocal/lib/python3.6/dist-packages (from jupyter->sql_magic) (4.8.2)
Requirement not upgraded as not directly required: notebook in /usr/lo
cal/lib/python3.6/dist-packages (from jupyter->sql_magic) (5.2.2)
Requirement not upgraded as not directly required: ipywidgets in /usr/
local/lib/python3.6/dist-packages (from jupyter->sql_magic) (7.2.1)
Requirement not upgraded as not directly required: jupyter-console in
 /usr/local/lib/python3.6/dist-packages (from jupyter->sql_magic) (5.
2.0)
Requirement not upgraded as not directly required: decorator in /usr/l
ocal/lib/python3.6/dist-packages (from traitlets->sql_magic) (4.3.0)
Requirement not upgraded as not directly required: ipython-genutils in
/usr/local/lib/python3.6/dist-packages (from traitlets->sql_magic) (0.
2.0)
Requirement not upgraded as not directly required: six in /usr/lib/pyt
hon3/dist-packages (from traitlets->sql_magic) (1.11.0)
Requirement not upgraded as not directly required: python-dateutil>=2.
5.0 in /usr/local/lib/python3.6/dist-packages (from pandas->sql_magic)
(2.7.3)
Requirement not upgraded as not directly required: pytz>=2011k in /us
r/local/lib/python3.6/dist-packages (from pandas->sql_magic) (2018.4)
Requirement not upgraded as not directly required: numpy>=1.9.0 in /us
r/local/lib/python3.6/dist-packages (from pandas->sql_magic) (1.14.4)
```

```
Requirement not upgraded as not directly required: pygments in /usr/lo
cal/lib/python3.6/dist-packages (from ipython->sql_magic) (2.2.0)
Requirement not upgraded as not directly required: simplegeneric>0.8 i
n /usr/local/lib/python3.6/dist-packages (from ipython->sql_magic) (0.
8.1)
Requirement not upgraded as not directly required: jedi>=0.10 in /usr/
local/lib/python3.6/dist-packages (from ipython->sql_magic) (0.12.0)
Requirement not upgraded as not directly required: pickleshare in /us
r/local/lib/python3.6/dist-packages (from ipython->sql_magic) (0.7.4)
Requirement not upgraded as not directly required: pexpect; sys_platfo
rm != "win32" in /usr/local/lib/python3.6/dist-packages (from ipython-
>sql_magic) (4.6.0)
Requirement not upgraded as not directly required: prompt-toolkit<2.0.
0,>=1.0.4 in /usr/local/lib/python3.6/dist-packages (from ipython->sql
_magic) (1.0.15)
Requirement not upgraded as not directly required: setuptools>=18.5 in
/usr/lib/python3/dist-packages (from ipython->sql_magic) (39.0.1)
Requirement not upgraded as not directly required: jupyter-core in /us
r/local/lib/python3.6/dist-packages (from qtconsole->jupyter->sql_magi
c) (4.4.0)
Requirement not upgraded as not directly required: jupyter-client>=4.1
in /usr/local/lib/python3.6/dist-packages (from qtconsole->jupyter->sq
l_magic) (5.2.3)
Requirement not upgraded as not directly required: testpath in /usr/lo
cal/lib/python3.6/dist-packages (from nbconvert->jupyter->sql_magic)
 (0.3.1)
Requirement not upgraded as not directly required: nbformat>=4.4 in /u
sr/local/lib/python3.6/dist-packages (from nbconvert->jupyter->sql_mag
ic) (4.4.0)
Requirement not upgraded as not directly required: jinja2 in /usr/loca
l/lib/python3.6/dist-packages (from nbconvert->jupyter->sql_magic) (2.
10)
Requirement not upgraded as not directly required: bleach in /usr/loca
l/lib/python3.6/dist-packages (from nbconvert->jupyter->sql_magic) (2.
1.3)
Requirement not upgraded as not directly required: entrypoints>=0.2.2
 in /usr/local/lib/python3.6/dist-packages (from nbconvert->jupyter->s
ql_magic) (0.2.3)
Requirement not upgraded as not directly required: pandocfilters>=1.4.
1 in /usr/local/lib/python3.6/dist-packages (from nbconvert->jupyter->
sql_magic) (1.4.2)
Requirement not upgraded as not directly required: mistune>=0.7.4 in /
usr/local/lib/python3.6/dist-packages (from nbconvert->jupyter->sql_ma
gic) (0.8.3)
Requirement not upgraded as not directly required: tornado>=4.0 in /us
r/local/lib/python3.6/dist-packages (from ipykernel->jupyter->sql_magi
c) (4.5.3)
Requirement not upgraded as not directly required: terminado>=0.3.3; s
ys_platform != "win32" in /usr/local/lib/python3.6/dist-packages (from
notebook->jupyter->sql_magic) (0.8.1)
Requirement not upgraded as not directly required: widgetsnbextension~
=3.2.0 in /usr/local/lib/python3.6/dist-packages (from ipywidgets->jup
yter->sql_magic) (3.2.1)
Requirement not upgraded as not directly required: parso>=0.2.0 in /us
r/local/lib/python3.6/dist-packages (from jedi>=0.10->ipython->sql_mag
ic) (0.2.1)
Requirement not upgraded as not directly required: ptyprocess>=0.5 in
```

```
 /usr/local/lib/python3.6/dist-packages (from pexpect; sys_platform !=
"win32"->ipython->sql_magic) (0.5.2)
Requirement not upgraded as not directly required: wcwidth in /usr/loc
al/lib/python3.6/dist-packages (from prompt-toolkit<2.0.0,>=1.0.4->ipy
thon->sql_magic) (0.1.7)
Requirement not upgraded as not directly required: pyzmq>=13 in /usr/l
ocal/lib/python3.6/dist-packages (from jupyter-client>=4.1->qtconsole-
>jupyter->sql_magic) (17.0.0)
Requirement not upgraded as not directly required: jsonschema!=2.5.0,>
=2.4 in /usr/local/lib/python3.6/dist-packages (from nbformat>=4.4->nb
convert->jupyter->sql_magic) (2.6.0)
Requirement not upgraded as not directly required: MarkupSafe>=0.23 in
/usr/local/lib/python3.6/dist-packages (from jinja2->nbconvert->jupyte
r->sql_magic) (1.0)
Requirement not upgraded as not directly required: html5lib!=1.0b1,!=
1.0b2,!=1.0b3,!=1.0b4,!=1.0b5,!=1.0b6,!=1.0b7,!=1.0b8,>=0.99999999pre
 in /usr/lib/python3/dist-packages (from bleach->nbconvert->jupyter->s
ql_magic) (0.999999999)
Installing collected packages: sql-magic
  Found existing installation: sql-magic 0.0.3
    Uninstalling sql-magic-0.0.3:
      Successfully uninstalled sql-magic-0.0.3
Successfully installed sql-magic-0.0.4
You are using pip version 10.0.1, however version 18.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' comm
and.
```

In [0]:

```
from sqlalchemy import create_engine
```

In [0]:

```
conn_string = 'mysql://{user}:{password}@{host}/?charset=utf8'.format(
    host = 'db.ipeirotis.org',
    user = 'student',
    password = 'dwdstudent2015',
    encoding = 'utf-8')
engine = create_engine(conn_string)
```

In [0]:

```
%reload_ext sql_magic
```

In [0]:

```
%config SQL.conn_name = 'engine'
```

Facebook Database

We started with the facebook database, using FavoriteMovies and Profiles tables from this database.

In [0]:
```
%%read_sql
use facebook
```

Query started at 07:27:32 PM UTC; Query executed in 0.01 m

Out[8]: <sql_magic.exceptions.EmptyResult at 0x7f0f28d00588>

In [0]:
```
%%read_sql
select *
from Profiles
limit 3
```

Query started at 07:35:53 PM UTC; Query executed in 0.00 m

Out[12]:

	ProfileID	Name	MemberSince	LastUpdate	School	Status	Sex	Birthday
0	800001	The Creator	2004-03-07	2005-02-15	NYU '06	Undergrad	Female	NaT
1	800002	Brian Whitton	2004-03-22	2006-01-11	NYU '07	Undergrad	Male	1984-12-16
2	800003	Anita Nagwani	2004-03-22	2006-01-17	NYU '05	Alumnus/Alumna	Female	1984-01-19

c

In [0]:
```
%%read_sql
select *
from FavoriteMovies
limit 3
```

Query started at 07:33:13 PM UTC; Query executed in 0.00 m

Out[11]:

	ProfileID	Movie
0	800002	North By Northwest
1	800002	The Apartment
2	800002	Pickpocket

The two tables will be joined on ProfileID, which is shared by both tables. From the tables we see that a ProfileID (a single FB user) may have more than one favorite movie (one to many connection).

IMDB Database

Let's now move on to the IMDB database. We will be using movies and movies_genres tables from this database.

In [0]:

```
%%read_sql
use imdb
```

```
Query started at 07:45:56 PM UTC; Query executed in 0.00 m
```

Out[13]:

```
<sql_magic.exceptions.EmptyResult at 0x7f0f28738dd8>
```

In [0]:

```
%%read_sql
select *
from movies
limit 3
```

```
Query started at 07:46:18 PM UTC; Query executed in 0.00 m
```

Out[14]:

	id	name	year	rank
0	0	#28	2002	NaN
1	1	#7 Train: An Immigrant Journey, The	2000	NaN
2	2	$	1971	6.4

In [0]:

```
%%read_sql
select *
from movies_genres
limit 3
```

```
Query started at 07:46:37 PM UTC; Query executed in 0.00 m
```

Out[15]:

	movie_id	genre
0	1	Short
1	1	Documentary
2	2	Crime

Based on the table columns shown above, the two IMDB tables can be joined by movie_id. Also, this database can be joined with Facebook database using name of the movie.

ER Diagram

Below is the ER Diagram showing all the relationships between the tables in the two databases.

image.png

Cleaning the data

To understand the discrepancies in the datasets, we examined whether the data from the two datasets matched on a one-on-one basis.

As shown in the code below, we used the favorite movies table and imdb movies table to create two dataframes for analysis.

In [0]:
```
import pandas as pd

query_join1 = '''select * from facebook.FavoriteMovies'''
df_fav = pd.read_sql(query_join1,con=conn_string)
df_fav.head(3)
```

Out[35]:

	ProfileID	Movie
0	800002	North By Northwest
1	800002	The Apartment
2	800002	Pickpocket

In [0]:
```
query_join2 = '''select * from imdb.movies'''
df_mov = pd.read_sql(query_join2,con=conn_string)
df_mov.head(3)
```

Out[62]:

	id	name	year	rank
0	0	#28	2002	NaN
1	1	#7 Train: An Immigrant Journey, The	2000	NaN
2	2	$	1971	6.4

We wanted to check whether each movie in favoriteMovies table also existed in the IMDB movie table. As shown in the code below, only 2,925 of 22,281 movies could be matched. When we looked at some of the movies that did not match, we saw that they were often misspelled in one or both databases (e.g., “**The Godfather I Iii**”), described inconsistently (e.g., **Seven** instead of **Se7en**), or were entries that didn’t correspond to actual movies (e.g., **And Other Old Boring Movies You Haven t Heard Of**).

In [0]:

```
import numpy as np

df_mov=df_mov.rename(columns = {'name':'Movie'})
df = pd.merge(df_fav, df_mov, on=['Movie'], how='left', indicator='Exist')
df['Exist'] = np.where(df.Exist == 'both', True, False)
df.drop('year', axis=1, inplace=True)
df.drop('id', axis=1, inplace=True)
df.drop('rank', axis=1, inplace=True)
df.drop('ProfileID', axis=1, inplace=True)
df.drop_duplicates(subset=None, keep='first', inplace=True)
df['Exist'].value_counts()
```

Out[96]:

```
False    22281
True      2925
Name: Exist, dtype: int64
```

In [0]:

```
df.loc[df['Exist'] == False].head(5)
```

Out[98]:

	Movie	Exist
0	North By Northwest	False
1	The Apartment	False
4	And Other Old Boring Movies You Haven t Heard Of	False
5	Any Movie That Doesn t Put Me To Sleep If You ...	False
6	You Know I Tend To Sleep Through Movies	False

We worked with 2,925 movies, as we were forced to omit certain entries that did not match between FavoriteMovies from Facebook and Movies from IMDB. Under ideal circumstances, we would want to unify the naming conventions between the two databases. One potential method would be using regular expressions to help us with "soft matches" between the two databases—while the variety of inconsistencies would make it hard to resolve them all automatically, it would still make the remaining manual process faster.

Joining the tables for cleaning

The second part of cleaning is to be done in the joined sql database. Therefore, we will first join the 4 tables in the 2 databases as shown in ER Diagram above and write it into a dataframe.

In [0]:

```
query_join = '''select *
from imdb.movies m
inner join imdb.movies_genres im on im.movie_id = m.id
inner join facebook.FavoriteMovies f ON m.name = f.movie
inner join facebook.Profiles p on f.ProfileID = p.ProfileID'''
df_raw = pd.read_sql(query_join,con=conn_string)
```

We will first get rid of duplicated columns, then will get rid of the columns that we will not use in our analysis.

In [0]:
```
df_raw = df_raw.loc[:,~df_raw.columns.duplicated()] #get rid of the two dup
df_raw.drop('id', axis=1, inplace=True)
df_raw.drop('name', axis=1, inplace=True)
df_raw.drop('Website', axis=1, inplace=True)
df_raw.drop('Geography', axis=1, inplace=True)
df_raw.drop('HighSchool', axis=1, inplace=True)
df_raw.drop('HomeTown', axis=1, inplace=True)
df_raw.drop('Residence', axis=1, inplace=True)
df_raw.drop('CurrentAddress', axis=1, inplace=True)
df_raw.drop('CurrentTown', axis=1, inplace=True)
df_raw.drop('CurrentState', axis=1, inplace=True)
df_raw.drop('MemberSince', axis=1, inplace=True)
df_raw.drop('LastUpdate', axis=1, inplace=True)
df_raw.drop('School', axis=1, inplace=True)
df_raw.drop('Status', axis=1, inplace=True)
df_raw.drop('AIM', axis=1, inplace=True)
```

As can be seen below, Sunny Kim has both Good Will Hunting and Amadeus as her favorite movies and both of the movies are classified as Drama in the genre column. As our analysis will be based on genres, we need to get rid of repating genres for one FB profile to prevent inaccurate results.

In [0]:

```
df_raw.head(10)
```

Out[134]:

	year	rank	movie_id	genre	ProfileID	Movie	Name	Sex	Birthday	Political\
0	1959	8.6	235062	Adventure	800002	North By Northwest	Brian Whitton	Male	1984-12-16	Liber
1	1959	8.6	235062	Thriller	800002	North By Northwest	Brian Whitton	Male	1984-12-16	Liber
2	1959	8.6	235062	Mystery	800002	North By Northwest	Brian Whitton	Male	1984-12-16	Liber
3	1959	7.8	255250	Crime	800002	Pickpocket	Brian Whitton	Male	1984-12-16	Liber
4	1959	7.8	255250	Drama	800002	Pickpocket	Brian Whitton	Male	1984-12-16	Liber
5	1997	7.8	131665	Drama	800004	Good Will Hunting	Sunny Kim	Female	1985-08-08	Conser
7	1984	8.3	12937	Music	800004	Amadeus	Sunny Kim	Female	1985-08-08	Conser
9	2004	8.6	104338	Romance	800004	Eternal Sunshine Of The Spotless Mind	Sunny Kim	Female	1985-08-08	Conser
10	2004	8.6	104338	Sci-Fi	800004	Eternal Sunshine Of The Spotless Mind	Sunny Kim	Female	1985-08-08	Conser
11	2004	8.6	104338	Comedy	800004	Eternal Sunshine Of The Spotless Mind	Sunny Kim	Female	1985-08-08	Conser

In [0]:
```
df_raw.drop_duplicates(subset=['ProfileID','genre'], keep='first', inplace=
df_raw.head(10)
```

Out[131]:

	year	rank	movie_id	genre	ProfileID	Movie	Name	Sex	Birthday	Political\
0	1959	8.6	235062	Adventure	800002	North By Northwest	Brian Whitton	Male	1984-12-16	Liber
1	1959	8.6	235062	Thriller	800002	North By Northwest	Brian Whitton	Male	1984-12-16	Liber
2	1959	8.6	235062	Mystery	800002	North By Northwest	Brian Whitton	Male	1984-12-16	Liber
3	1959	7.8	255250	Crime	800002	Pickpocket	Brian Whitton	Male	1984-12-16	Liber
4	1959	7.8	255250	Drama	800002	Pickpocket	Brian Whitton	Male	1984-12-16	Liber
5	1997	7.8	131665	Drama	800004	Good Will Hunting	Sunny Kim	Female	1985-08-08	Conser
7	1984	8.3	12937	Music	800004	Amadeus	Sunny Kim	Female	1985-08-08	Conser
9	2004	8.6	104338	Romance	800004	Eternal Sunshine Of The Spotless Mind	Sunny Kim	Female	1985-08-08	Conser
10	2004	8.6	104338	Sci-Fi	800004	Eternal Sunshine Of The Spotless Mind	Sunny Kim	Female	1985-08-08	Conser
11	2004	8.6	104338	Comedy	800004	Eternal Sunshine Of The Spotless Mind	Sunny Kim	Female	1985-08-08	Conser

Now that the duplicates are dropped, we can start our analysis.

Analysis & Results

Relationship between genre and sex

We will first begin with data on Facebook users' sex. To do that, we first need to create a pivot table showing the total number of each sex group that like certain genre. Then, we need to identify how many unique Facebook profiles are there in our raw data dataframe to normalize the data.

In [0]:

```
myList = ['genre','Name','Sex']
df_sex = df_raw[myList]
df_sex_pivot = df_sex.pivot_table(index='genre', columns = 'Sex', values =
df_sex_pivot
```

Out[203]:

Sex	Female	Male
genre		
Action	3194	3226
Adult	854	593
Adventure	3216	2349
Animation	3430	2370
Comedy	7005	4252
Crime	3766	3405
Documentary	1410	1089
Drama	7221	4831
Family	2672	1136
Fantasy	3403	2199
Film-Noir	154	161
Horror	1236	1265
Music	1861	669
Musical	2713	1132
Mystery	2572	2223
Romance	5838	2829
Sci-Fi	2192	1987
Short	3892	2671
Thriller	3441	3159
War	1026	1214
Western	558	809

In [0]:

```
#find out how many unique males and females are there in the list
df_sex_unique= df_sex
df_sex_unique.drop('genre', axis=1, inplace=True)
df_sex_unique.drop_duplicates(subset=None, keep='first', inplace=True)
print("Number of Profiles based on sex")
df_sex_unique['Sex'].value_counts()
```

```
Number of Profiles based on sex

/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py:3694: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/
stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pa
ndas-docs/stable/indexing.html#indexing-view-versus-copy)
  errors=errors)
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/
stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pa
ndas-docs/stable/indexing.html#indexing-view-versus-copy)
  This is separate from the ipykernel package so we can avoid doing imports
until
```

Out[204]:

```
Female    7761
Male      5274
Name: Sex, dtype: int64
```

In order to normalize the data, we will divide each row by the total number of each sex. This will give us what percent of each sex group likes a certain genre.

In [0]:

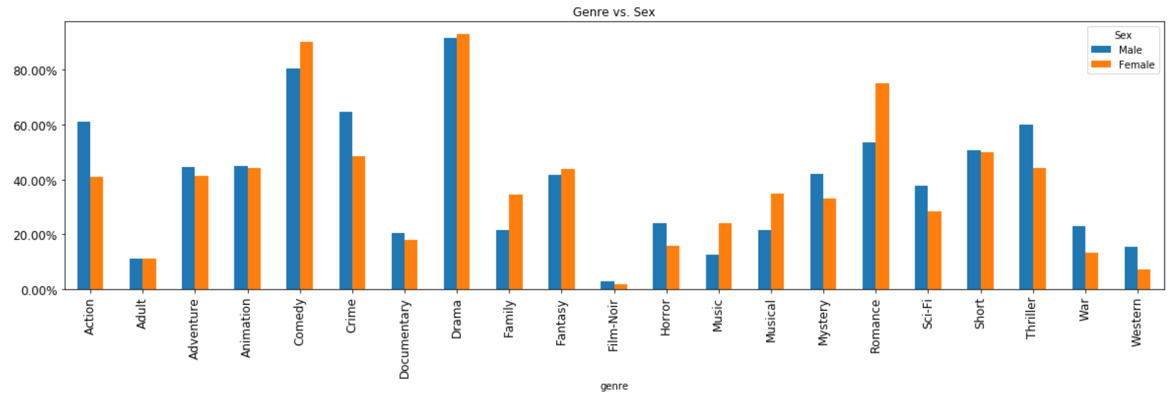
```
female=7761
male = 5274
df_sex_pivot['Female'] = df_sex_pivot['Female']/female
df_sex_pivot['Male'] = df_sex_pivot['Male']/male
df_sex_pivot
```

Out[205]:

Sex genre	Female	Male
Action	0.411545	0.611680
Adult	0.110037	0.112438
Adventure	0.414380	0.445392
Animation	0.441953	0.449374
Comedy	0.902590	0.806219
Crime	0.485247	0.645620
Documentary	0.181678	0.206485
Drama	0.930421	0.916003
Family	0.344286	0.215396
Fantasy	0.438474	0.416951
Film-Noir	0.019843	0.030527
Horror	0.159258	0.239856
Music	0.239789	0.126849
Musical	0.349568	0.214638
Mystery	0.331401	0.421502
Romance	0.752223	0.536405
Sci-Fi	0.282438	0.376754
Short	0.501482	0.506447
Thriller	0.443371	0.598976
War	0.132199	0.230186
Western	0.071898	0.153394

In [0]:
```
import matplotlib.pyplot as plt
ax = df_sex_pivot[['Male','Female']].plot(kind='bar', stacked=False, figsiz
vals = ax.get_yticks()
ax.set_yticklabels(['{:,.2%}'.format(x) for x in vals])
```

Out[229]:
```
[Text(0,0,'0.00%'),
 Text(0,0,'20.00%'),
 Text(0,0,'40.00%'),
 Text(0,0,'60.00%'),
 Text(0,0,'80.00%'),
 Text(0,0,'100.00%')]
```



Results

Actions: While 60% of males like action movies, 40% of females like them.
Comedy: While 80% of males like action movies, 90% of females like them.
Family: While 21% of males like action movies, 34% of females like them.
Romance: While 53% of males like action movies, 75% of females like them.
Thriller: While 60% of males like action movies, 43% of females like them.

Relationship between genre and political view

To conduct this analysis, we first need to create a pivot table showing the total number of each political view that like certain genres. Then, we need to identify how many unique Facebook profiles there are in our raw data dataframe to normalize the data.

In [0]:

```
myList = ['genre','Name','PoliticalViews']
df_pol = df_raw[myList]
df_pol_pivot = df_pol.pivot_table(index='genre', columns = 'PoliticalViews'
df_pol_pivot
```

Out[232]:

PoliticalViews	Apathetic	Conservative	Liberal	Libertarian	Moderate	Other	Very Conservative	Li
genre								
Action	268	377	2220	116	1131	251	50	
Adult	60	67	544	24	212	54	5	
Adventure	204	292	2027	95	955	210	30	
Animation	238	298	2166	97	940	198	32	
Comedy	409	551	4235	163	1791	390	63	
Crime	288	361	2612	129	1179	271	51	
Documentary	95	77	963	45	377	111	22	
Drama	450	600	4436	201	1950	441	73	
Family	138	196	1407	42	632	115	18	
Fantasy	219	246	2132	102	886	208	22	
Film-Noir	16	10	118	5	33	18	3	
Horror	125	98	923	58	369	114	14	
Music	87	94	1039	36	339	67	14	
Musical	120	143	1571	43	566	119	13	
Mystery	209	198	1840	99	752	178	26	
Romance	299	391	3348	127	1361	265	45	
Sci-Fi	181	169	1578	90	640	176	18	
Short	249	307	2473	110	1083	223	38	
Thriller	288	325	2410	127	1074	244	43	
War	77	149	736	37	472	90	21	
Western	47	103	418	27	296	52	14	

In [0]:

```
#find out how many unique people with specific political views are there in
df_pol_unique= df_pol
df_pol_unique.drop('genre', axis=1, inplace=True)
df_pol_unique.drop_duplicates(subset=None, keep='first', inplace=True)
print("Number of Profiles based on Political Views")
df_pol_unique['PoliticalViews'].value_counts()
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py:3694: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/
stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pa
ndas-docs/stable/indexing.html#indexing-view-versus-copy)
  errors=errors)

Number of Profiles based on Political Views

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/
stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pa
ndas-docs/stable/indexing.html#indexing-view-versus-copy)
  after removing the cwd from sys.path.
```

Out[233]:

```
Liberal              4767
Moderate             2075
Very Liberal         1572
Conservative          648
Apathetic             509
Other                 503
Libertarian           217
Very Conservative      90
Name: PoliticalViews, dtype: int64
```

In order to normalize the data, we will divide each row by the total number of each political view. This will give us what percent of users with each political view likes a certain genre.

In [0]:

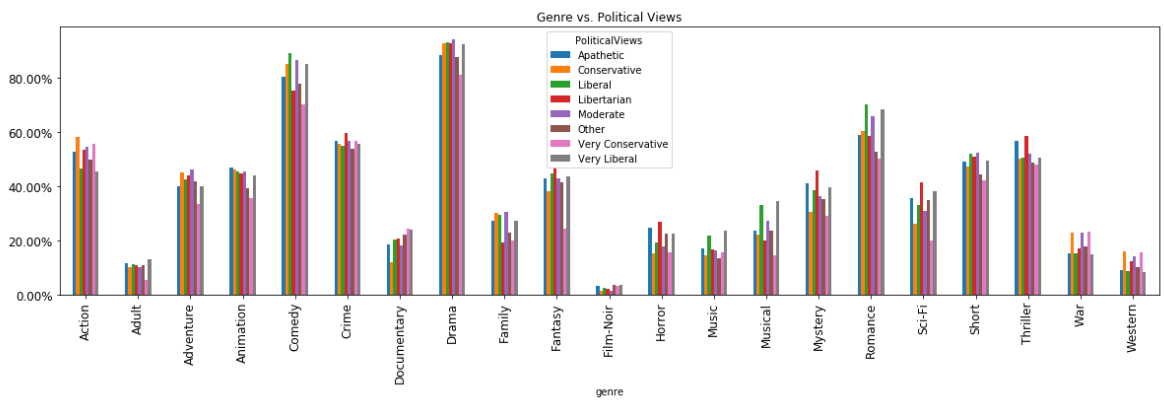
```
Liberal=4767
Moderate=2075
VeryLiberal=1572
Conservative=648
Apathetic=509
Other     = 503
Libertarian=  217
VeryConservative= 90
df_pol_pivot['Liberal'] = df_pol_pivot['Liberal']/Liberal
df_pol_pivot['Moderate'] = df_pol_pivot['Moderate']/Moderate
df_pol_pivot['Very Liberal'] = df_pol_pivot['Very Liberal']/VeryLiberal
df_pol_pivot['Conservative'] = df_pol_pivot['Conservative']/Conservative
df_pol_pivot['Apathetic'] = df_pol_pivot['Apathetic']/Apathetic
df_pol_pivot['Other'] = df_pol_pivot['Other']/Other
df_pol_pivot['Libertarian'] = df_pol_pivot['Libertarian']/Libertarian
df_pol_pivot['Very Conservative'] = df_pol_pivot['Very Conservative']/VeryC

df_pol_pivot
```

Out[234]:

PoliticalViews	Apathetic	Conservative	Liberal	Libertarian	Moderate	Other	Very Conservativ
genre							
Action	0.526523	0.581790	0.465702	0.534562	0.545060	0.499006	0.55555
Adult	0.117878	0.103395	0.114118	0.110599	0.102169	0.107356	0.05555
Adventure	0.400786	0.450617	0.425215	0.437788	0.460241	0.417495	0.33333
Animation	0.467583	0.459877	0.454374	0.447005	0.453012	0.393638	0.35555
Comedy	0.803536	0.850309	0.888399	0.751152	0.863133	0.775348	0.70000
Crime	0.565815	0.557099	0.547934	0.594470	0.568193	0.538767	0.56666
Documentary	0.186640	0.118827	0.202014	0.207373	0.181687	0.220676	0.24444
Drama	0.884086	0.925926	0.930564	0.926267	0.939759	0.876740	0.81111
Family	0.271120	0.302469	0.295154	0.193548	0.304578	0.228628	0.20000
Fantasy	0.430255	0.379630	0.447241	0.470046	0.426988	0.413519	0.24444
Film-Noir	0.031434	0.015432	0.024754	0.023041	0.015904	0.035785	0.03333
Horror	0.245580	0.151235	0.193623	0.267281	0.177831	0.226640	0.15555
Music	0.170923	0.145062	0.217957	0.165899	0.163373	0.133201	0.15555
Musical	0.235756	0.220679	0.329557	0.198157	0.272771	0.236581	0.14444
Mystery	0.410609	0.305556	0.385987	0.456221	0.362410	0.353877	0.28888
Romance	0.587426	0.603395	0.702329	0.585253	0.655904	0.526839	0.50000
Sci-Fi	0.355599	0.260802	0.331026	0.414747	0.308434	0.349901	0.20000
Short	0.489194	0.473765	0.518775	0.506912	0.521928	0.443340	0.42222
Thriller	0.565815	0.501543	0.505559	0.585253	0.517590	0.485089	0.47777
War	0.151277	0.229938	0.154395	0.170507	0.227470	0.178926	0.23333
Western	0.092338	0.158951	0.087686	0.124424	0.142651	0.103380	0.15555

In [0]:
```
ax1 = df_pol_pivot[['Apathetic','Conservative','Liberal','Libertarian','Mod
vals = ax1.get_yticks()
ax1.set_yticklabels(['{:,.2%}'.format(x) for x in vals])
```

Out[236]:
```
[Text(0,0,'0.00%'),
 Text(0,0,'20.00%'),
 Text(0,0,'40.00%'),
 Text(0,0,'60.00%'),
 Text(0,0,'80.00%'),
 Text(0,0,'100.00%')]
```



Results

Actions: While 58% of conservatives like action movies, 46% of liberals like it.
Musical: While 22% of conservatives like musicals, 33% of liberals like it.
Romance: While 60% of conservatives like action movies, 70% of liberals like it.
War: While 23% of conservatives like action movies, 15% of liberals like it.
Western: While 16% of conservatives like action movies, 9% of liberals like it.

Relationship between genre and birthday

Analyzing birthday information was a little different from the analyses above. The birthday data had three basic areas that we can explore: year, month, and day.
Also, in conducting our analysis there were null items that we needed to get rid of.

In [0]:
```
myList = ['genre','Name','Birthday']
df_bd = df_raw[myList]
df_bd[:20]
```

Out[305]:

	genre	Name	Birthday
0	Adventure	Brian Whitton	1984-12-16
1	Thriller	Brian Whitton	1984-12-16
2	Mystery	Brian Whitton	1984-12-16
3	Crime	Brian Whitton	1984-12-16
4	Drama	Brian Whitton	1984-12-16
5	Drama	Sunny Kim	1985-08-08
7	Music	Sunny Kim	1985-08-08
9	Romance	Sunny Kim	1985-08-08
10	Sci-Fi	Sunny Kim	1985-08-08
11	Comedy	Sunny Kim	1985-08-08
12	Western	Sunny Kim	1985-08-08
14	War	Sunny Kim	1985-08-08
15	Action	Sunny Kim	1985-08-08
16	Animation	Erica Bern	NaT
17	Fantasy	Erica Bern	NaT
18	Drama	Erica Bern	NaT
19	Comedy	Erica Bern	NaT
20	Family	Erica Bern	NaT
21	Musical	Erica Bern	NaT
26	Romance	Erica Bern	NaT

In [0]:
```
df_bd['Birthday'].describe()
```

Out[306]:
```
count                   98132
unique                   2668
top       1987-08-17 00:00:00
freq                      145
first     1900-08-31 00:00:00
last      1989-02-27 00:00:00
Name: Birthday, dtype: object
```

As the FB data is from NYU, most of the students and faculty are at similar age range. Therefore, we will look into the months that people are born in.

In [0]:
```
df_bd['month'] = df_bd['Birthday'].dt.month
df_bd[95:100]
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/
stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pa
ndas-docs/stable/indexing.html#indexing-view-versus-copy)
  """Entry point for launching an IPython kernel.
```

Out[307]:

	genre	Name	Birthday	month
234	Short	Cathy Xu	1985-04-03	4.0
235	Crime	Cathy Xu	1985-04-03	4.0
243	Comedy	Dave Birinyi	1985-07-12	7.0
244	Romance	Dave Birinyi	1985-07-12	7.0
245	Drama	Dave Birinyi	1985-07-12	7.0

In [0]:
```
#cleaning the null items
df_bd[pd.notnull(df_bd['Birthday'])]
df_bd.head()
```

Out[308]:

	genre	Name	Birthday	month
0	Adventure	Brian Whitton	1984-12-16	12.0
1	Thriller	Brian Whitton	1984-12-16	12.0
2	Mystery	Brian Whitton	1984-12-16	12.0
3	Crime	Brian Whitton	1984-12-16	12.0
4	Drama	Brian Whitton	1984-12-16	12.0

In [0]:
```
df_bd_pivot = df_bd.pivot_table(index='genre', columns = 'month', values =
df_bd_pivot
```

Out[309]:

month genre	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0
Action	488	455	536	443	531	514	490	474	494	487	498	523
Adult	113	117	120	93	114	112	124	134	125	96	110	127
Adventure	416	424	477	406	441	434	437	431	448	444	415	435
Animation	433	422	493	421	488	465	466	446	458	452	442	432
Comedy	834	787	902	825	902	918	900	848	892	906	857	889
Crime	518	512	564	492	606	599	543	544	571	566	571	579
Documentary	202	173	197	149	214	210	208	183	222	184	187	208
Drama	904	875	966	860	988	985	935	918	950	933	904	933
Family	275	271	333	294	294	306	315	298	292	311	280	315
Fantasy	425	424	461	410	468	437	440	436	436	459	416	451
Film-Noir	27	28	37	17	30	27	20	23	23	17	25	27
Horror	174	164	198	171	220	204	201	196	211	215	185	195
Music	199	188	236	180	195	209	210	189	220	198	195	190
Musical	296	278	321	293	316	305	310	301	327	299	283	298
Mystery	359	338	403	321	390	401	362	371	392	357	386	384
Romance	655	621	724	628	686	723	689	655	705	683	648	686
Sci-Fi	304	320	345	287	335	319	303	315	359	328	324	343
Short	490	458	550	468	554	539	537	482	541	513	497	517
Thriller	492	462	535	458	540	552	521	498	508	512	504	522
War	187	165	185	145	179	168	158	168	173	179	167	219
Western	92	98	121	94	120	108	100	112	101	108	97	114

In [0]:
```
#find out how many unique people with specific birth months are there in th
df_bd_unique= df_bd
df_bd_unique.drop('genre', axis=1, inplace=True)
df_bd_unique.drop_duplicates(subset=None, keep='first', inplace=True)
print("Number of Profiles based on Birth Months")
m = df_bd_unique['month'].value_counts()
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py:3694: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/
stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pa
ndas-docs/stable/indexing.html#indexing-view-versus-copy)
  errors=errors)

Number of Profiles based on Birth Months

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/
stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pa
ndas-docs/stable/indexing.html#indexing-view-versus-copy)
  after removing the cwd from sys.path.
```

In [0]:
```
m
```

Out[311]:
```
5.0     1061
6.0     1055
3.0     1051
9.0     1025
10.0    1021
12.0    1012
7.0     1008
8.0      992
11.0     984
1.0      967
4.0      945
2.0      929
Name: month, dtype: int64
```

In [0]:
```
for i in range(1,13):
    df_bd_pivot[i] = df_bd_pivot[i]/m[i]
```

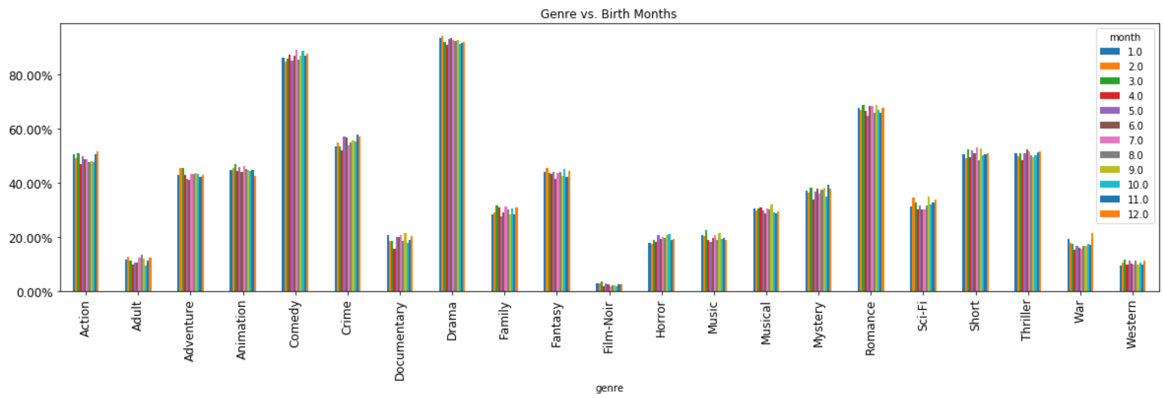
In [0]:
```
df_bd_pivot
```

Out[313]:

month genre	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0
Action	0.504654	0.489774	0.509990	0.468783	0.500471	0.487204	0.486111	0.477823
Adult	0.116856	0.125942	0.114177	0.098413	0.107446	0.106161	0.123016	0.135081
Adventure	0.430196	0.456405	0.453853	0.429630	0.415646	0.411374	0.433532	0.434476
Animation	0.447777	0.454252	0.469077	0.445503	0.459943	0.440758	0.462302	0.449597
Comedy	0.862461	0.847147	0.858230	0.873016	0.850141	0.870142	0.892857	0.854839
Crime	0.535677	0.551130	0.536632	0.520635	0.571159	0.567773	0.538690	0.548387
Documentary	0.208893	0.186222	0.187441	0.157672	0.201697	0.199052	0.206349	0.184476
Drama	0.934850	0.941873	0.919125	0.910053	0.931197	0.933649	0.927579	0.925403
Family	0.284385	0.291712	0.316841	0.311111	0.277097	0.290047	0.312500	0.300403
Fantasy	0.439504	0.456405	0.438630	0.433862	0.441093	0.414218	0.436508	0.439516
Film-Noir	0.027921	0.030140	0.035205	0.017989	0.028275	0.025592	0.019841	0.023185
Horror	0.179938	0.176534	0.188392	0.180952	0.207352	0.193365	0.199405	0.197581
Music	0.205791	0.202368	0.224548	0.190476	0.183789	0.198104	0.208333	0.190524
Musical	0.306101	0.299247	0.305423	0.310053	0.297832	0.289100	0.307540	0.303427
Mystery	0.371251	0.363832	0.383444	0.339683	0.367578	0.380095	0.359127	0.373992
Romance	0.677353	0.668461	0.688868	0.664550	0.646560	0.685308	0.683532	0.660282
Sci-Fi	0.314374	0.344456	0.328259	0.303704	0.315740	0.302370	0.300595	0.317540
Short	0.506722	0.493003	0.523311	0.495238	0.522149	0.510900	0.532738	0.485887
Thriller	0.508790	0.497309	0.509039	0.484656	0.508954	0.523223	0.516865	0.502016
War	0.193382	0.177610	0.176023	0.153439	0.168709	0.159242	0.156746	0.169355
Western	0.095140	0.105490	0.115128	0.099471	0.113101	0.102370	0.099206	0.112903

In [0]:
```
ax2 = df_bd_pivot[[1,2,3,4,5,6,7,8,9,10,11,12]].plot(kind='bar', stacked=Fa
vals = ax2.get_yticks()
ax2.set_yticklabels(['{:,.2%}'.format(x) for x in vals])
```

Out[314]:
```
[Text(0,0,'0.00%'),
 Text(0,0,'20.00%'),
 Text(0,0,'40.00%'),
 Text(0,0,'60.00%'),
 Text(0,0,'80.00%'),
 Text(0,0,'100.00%')]
```



Results

Based on the chart above, it appears that people's tendency to like certain genres is not related to their birth month. The only outcome that really stands out is for the genre "War,"" where 22% of people born in December like War films vs. 16% in general.

Relationship between genre and home state

We will start this by cleaning the data, to deal with the many "None's" entries for home state as shown below.

In [0]:

```
myList = ['genre','Name','HomeState']
df_state = df_raw[myList]
df_state[:20]
```

Out[437]:

	genre	Name	HomeState
0	Adventure	Brian Whitton	None
1	Thriller	Brian Whitton	None
2	Mystery	Brian Whitton	None
3	Crime	Brian Whitton	None
4	Drama	Brian Whitton	None
5	Drama	Sunny Kim	None
7	Music	Sunny Kim	None
9	Romance	Sunny Kim	None
10	Sci-Fi	Sunny Kim	None
11	Comedy	Sunny Kim	None
12	Western	Sunny Kim	None
14	War	Sunny Kim	None
15	Action	Sunny Kim	None
16	Animation	Erica Bern	NY
17	Fantasy	Erica Bern	NY
18	Drama	Erica Bern	NY
19	Comedy	Erica Bern	NY
20	Family	Erica Bern	NY
21	Musical	Erica Bern	NY
26	Romance	Erica Bern	NY

In [0]:

```
#cleaning the null items
df_state = df_state.replace(to_replace='None', value=np.nan).dropna()
df_state.head()
```

Out[438]:

	genre	Name	HomeState
16	Animation	Erica Bern	NY
17	Fantasy	Erica Bern	NY
18	Drama	Erica Bern	NY
19	Comedy	Erica Bern	NY
20	Family	Erica Bern	NY

We also need to clean the rest of the data. As can be seen from the table below, there are entries for states that are wrong or meaningless for the purposes of our analysis. Therefore, the data should be replaced accordingly.

In [0]:

```
df_state['HomeState'].value_counts()
```

Out[439]:

```
NY             917
NJ             676
CA             283
CT             155
PA             133
TX             113
MA              77
VA              68
JERSEY          60
RI              60
MD              53
TEXAS           44
GA              40
DE              38
FL              37
NY 11375        32
CO              32
MD 20878        26
NEW JERSEY      26
WI              24
ME              20
MN              18
CALIFORNIA      16
OR              16
CA 90210        15
NM              15
OH              15
IL              14
NJ!!            14
NJ 08540        14
              ...
NJ 08028         5
NY 11355         5
NY 11421         5
NJ 07660         5
JERSEY.          5
NY 10029         5
IRONY            5
NY 10305         5
NY 10009         4
BELGIUM          4
NY 11230         4
MA 01524         4
NY 10019         4
VA 20132         4
NJ 07932         3
OHIO             3
NY 10003         2
NY 11553         2
FLORIDA          2
NY 10990         2
MARYLAND         2
CA 90275         2
```

```
OK                     2
NY 11568               1
N.Y                    1
VA 20169               1
NJ/NY                  1
MASSA2SHITS 01581      1
FL 34683               1
NJ/NYC                 1
Name: HomeState, Length: 123, dtype: int64
```

In [0]:

```
df_state['HomeState']=df_state['HomeState'].str.replace('\d+', '')
df_state['HomeState']=df_state['HomeState'].str.replace('.', '')
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].apply(lambda x:str(x).replace('
df_state['HomeState']=df_state['HomeState'].str.replace(' ', '')
```

In [0]:
```
df_state.head()
```

Out[427]:

	genre	Name	HomeState
16	Animation	Erica Bern	NY
17	Fantasy	Erica Bern	NY
18	Drama	Erica Bern	NY
19	Comedy	Erica Bern	NY
20	Family	Erica Bern	NY

In [0]:
```
df_state['HomeState'].value_counts()
```

Out[428]:
```
NY    1147
NJ     868
CA     367
CT     185
PA     160
TX     157
Name: HomeState, dtype: int64
```

In [0]:

```
#We will keep our analysis limited to the states that have the highest numb
a = ['NY','NJ','CA','CT','PA','TX']
df_state = df_state[df_state['HomeState'].isin(a)]
df_state
```

Out[441]:

	genre	Name	HomeState
16	Animation	Erica Bern	NY
17	Fantasy	Erica Bern	NY
18	Drama	Erica Bern	NY
19	Comedy	Erica Bern	NY
20	Family	Erica Bern	NY
21	Musical	Erica Bern	NY
26	Romance	Erica Bern	NY
557	Drama	Kathleen Kane	PA
558	Fantasy	Kathleen Kane	PA
559	Comedy	Kathleen Kane	PA
560	Adventure	Kathleen Kane	PA
562	Thriller	Kathleen Kane	PA
564	Crime	Kathleen Kane	PA
569	Documentary	Kathleen Kane	PA
574	Mystery	Kathleen Kane	PA
1193	Animation	Larry Lin	NJ
1194	Drama	Larry Lin	NJ
1196	War	Larry Lin	NJ
1198	Short	Larry Lin	NJ
3402	Drama	Brijesh Malkani	NJ
3403	Crime	Brijesh Malkani	NJ
3404	Thriller	Brijesh Malkani	NJ
3405	Adult	Brijesh Malkani	NJ
3407	Comedy	Brijesh Malkani	NJ
3636	Fantasy	Lisa Maniglia	CT
3637	Mystery	Lisa Maniglia	CT
3638	Romance	Lisa Maniglia	CT
3639	Drama	Lisa Maniglia	CT
3726	Action	BJ Kraska	NJ
3727	Crime	BJ Kraska	NJ
...	...	...	...
289411	Drama	Lola The Dog	NJ

	genre	Name	HomeState
289412	Documentary	Lola The Dog	NJ
289413	Family	Lola The Dog	NJ
289414	Comedy	Lola The Dog	NJ
289417	Animation	Lola The Dog	NJ
289419	Musical	Lola The Dog	NJ
289420	Romance	Lola The Dog	NJ
289512	Drama	Maryam Syed	NY
289513	Romance	Maryam Syed	NY
289514	Thriller	Maryam Syed	NY
289515	Horror	Maryam Syed	NY
289806	Crime	Mark Stoholski	PA
289807	Film-Noir	Mark Stoholski	PA
289809	Thriller	Mark Stoholski	PA
289810	Drama	Mark Stoholski	PA
291867	Family	Kate Prilik	NY
291868	Adventure	Kate Prilik	NY
291869	Action	Kate Prilik	NY
291870	Fantasy	Kate Prilik	NY
291871	Comedy	Kate Prilik	NY
292182	Comedy	Erica Del Greco	NY
292183	Drama	Erica Del Greco	NY
292185	Adventure	Erica Del Greco	NY
292186	Animation	Erica Del Greco	NY
292188	Family	Erica Del Greco	NY
292193	Crime	Erica Del Greco	NY
292195	Romance	Erica Del Greco	NY
292196	Fantasy	Erica Del Greco	NY
292198	Sci-Fi	Erica Del Greco	NY
292199	Mystery	Erica Del Greco	NY

2884 rows × 3 columns

In [0]:
```
df_state_pivot = df_state.pivot_table(index='genre', columns = 'HomeState',
df_state_pivot.fillna(0)
```

Out[442]:

HomeState	CA	CT	NJ	NY	PA	TX
genre						
Action	21.0	12.0	53.0	72.0	10.0	8.0
Adult	6.0	0.0	12.0	18.0	1.0	4.0
Adventure	19.0	10.0	33.0	61.0	7.0	8.0
Animation	23.0	10.0	56.0	75.0	9.0	8.0
Comedy	41.0	20.0	90.0	130.0	18.0	16.0
Crime	27.0	10.0	56.0	81.0	15.0	10.0
Documentary	7.0	6.0	26.0	27.0	4.0	6.0
Drama	42.0	23.0	102.0	127.0	18.0	15.0
Family	14.0	6.0	31.0	49.0	0.0	6.0
Fantasy	19.0	12.0	50.0	48.0	8.0	6.0
Film-Noir	3.0	0.0	3.0	4.0	2.0	1.0
Horror	11.0	1.0	17.0	25.0	6.0	5.0
Music	11.0	3.0	25.0	26.0	4.0	3.0
Musical	11.0	6.0	31.0	41.0	3.0	7.0
Mystery	16.0	9.0	32.0	52.0	11.0	8.0
Romance	31.0	17.0	70.0	92.0	14.0	14.0
Sci-Fi	13.0	9.0	32.0	41.0	8.0	8.0
Short	23.0	16.0	65.0	65.0	7.0	8.0
Thriller	23.0	12.0	49.0	76.0	12.0	9.0
War	3.0	2.0	21.0	21.0	1.0	5.0
Western	3.0	1.0	14.0	16.0	2.0	2.0

In [0]:
```
# we need to find out how many unique people with specific birth months the
df_state_unique= df_state
df_state_unique.drop('genre', axis=1, inplace=True)
df_state_unique.drop_duplicates(subset=None, keep='first', inplace=True)
m = df_state_unique['HomeState'].value_counts()
```

In [0]:
```
#we will focus just on the 5 states below as the people were mainly from th
m
```

Out[444]:
```
NY    146
NJ    107
CA     44
CT     23
PA     21
TX     16
Name: HomeState, dtype: int64
```

In [0]:
```
df_state_pivot['CA']=df_state_pivot['CA']/m['CA']
df_state_pivot['CT']=df_state_pivot['CT']/m['CT']
df_state_pivot['PA']=df_state_pivot['PA']/m['PA']
df_state_pivot['TX']=df_state_pivot['TX']/m['TX']
df_state_pivot['NY']=df_state_pivot['NY']/m['NY']
df_state_pivot['NJ']=df_state_pivot['NJ']/m['NJ']
```

In [0]:

```
df_state_pivot.fillna(0)
```

Out[435]:

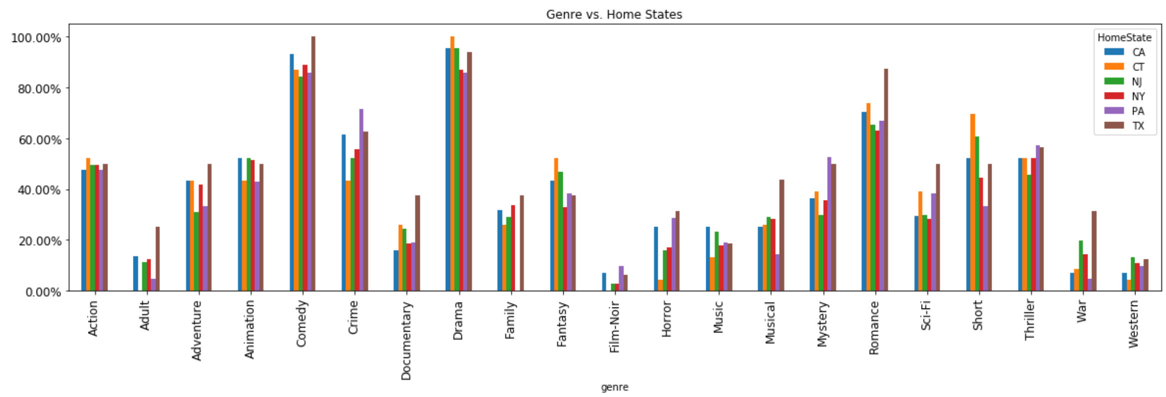
HomeState	CA	CT	NJ	NY	PA	TX
genre						
Action	0.477273	0.521739	0.495327	0.493151	0.476190	0.5000
Adult	0.136364	0.000000	0.112150	0.123288	0.047619	0.2500
Adventure	0.431818	0.434783	0.308411	0.417808	0.333333	0.5000
Animation	0.522727	0.434783	0.523364	0.513699	0.428571	0.5000
Comedy	0.931818	0.869565	0.841121	0.890411	0.857143	1.0000
Crime	0.613636	0.434783	0.523364	0.554795	0.714286	0.6250
Documentary	0.159091	0.260870	0.242991	0.184932	0.190476	0.3750
Drama	0.954545	1.000000	0.953271	0.869863	0.857143	0.9375
Family	0.318182	0.260870	0.289720	0.335616	0.000000	0.3750
Fantasy	0.431818	0.521739	0.467290	0.328767	0.380952	0.3750
Film-Noir	0.068182	0.000000	0.028037	0.027397	0.095238	0.0625
Horror	0.250000	0.043478	0.158879	0.171233	0.285714	0.3125
Music	0.250000	0.130435	0.233645	0.178082	0.190476	0.1875
Musical	0.250000	0.260870	0.289720	0.280822	0.142857	0.4375
Mystery	0.363636	0.391304	0.299065	0.356164	0.523810	0.5000
Romance	0.704545	0.739130	0.654206	0.630137	0.666667	0.8750
Sci-Fi	0.295455	0.391304	0.299065	0.280822	0.380952	0.5000
Short	0.522727	0.695652	0.607477	0.445205	0.333333	0.5000
Thriller	0.522727	0.521739	0.457944	0.520548	0.571429	0.5625
War	0.068182	0.086957	0.196262	0.143836	0.047619	0.3125
Western	0.068182	0.043478	0.130841	0.109589	0.095238	0.1250

In [0]:

```python
ax2 = df_state_pivot[['CA','CT','NJ','NY','PA','TX']].plot(kind='bar', stac
vals = ax2.get_yticks()
ax2.set_yticklabels(['{:,.2%}'.format(x) for x in vals])
```

Out[446]:

```
[Text(0,0,'0.00%'),
 Text(0,0,'20.00%'),
 Text(0,0,'40.00%'),
 Text(0,0,'60.00%'),
 Text(0,0,'80.00%'),
 Text(0,0,'100.00%'),
 Text(0,0,'120.00%')]
```



Results

Because of the limited amount of data, the reliability of our results is limited. However, our analysis suggests that it would be valuable to conduct additional research, incorporating a larger data set. For example, it would be interesting to see if Californians really do like comedy movies a lot more than New Yorkers. (Given recent criticism of data privacy practices of Facebook and similar companies, getting more highly detailed data may be easier said than done!)

In [0]:

Conclusions and Next Steps

More broadly, the good news is that we see meaningful relationships between movie genre preferences and a variety of other demographic/personal dimensions like sex, political views, and so on. From a Netflix/movie marketer’s perspective, even marginal differences between demographic groups that we discussed above (like popularity of action movies among liberals vs. conservatives) could have big business repercussions when the groups comprise millions of people. As a result, targeting the right consumer segments is particularly important for certain types of campaigns on FB or other digital channels (e.g., broader campaigns focusing on reach/impressions).

There are of course limitations in our analysis, some of which we've touched on in our project proposal, in our in-class update presentation, and in the project discussion above. One key limitation is that our analysis was limited to members of the NYU community, who are not necessarily representative of the broader US population. Furthermore, within that university body, certain groups are disproportionately represented (e.g., people from NY, NJ, and CA). Similarly, the data is not very current—it is possible that certain movie-preference trends we observed have grown, decreased, or changed entirely in recent years. It is also possible that we have reporting bias in our data. For example, certain FB users may like certain genres or movies, but avoid listing them due to a fear of embarrassment. Among other users, the opposite may be true: for example, a user might list a “favorite” movie to appear more intellectual or cultured.

At the same time, inconsistent entries between the FB and IMDB databases (and sometimes within one database itself) was an ongoing challenge. While we conducted data cleaning where possible, in an ideal world we'd be able to spend more time poring over the tens of thousands of rows of data to unify entries where possible. Even the types of inconsistencies we saw were inconsistent—sometimes people misspelled movie names, other times they entered information in the wrong place (e.g., inputting a city name where they should have input the state), etc. Having an effective solution to these challenges would become even more important if the database included data on millions of FB users, for instance.

With this in mind, we propose a few next steps to maximize the utility of our analysis. Aside from the data-cleaning mentioned previously, it would be great to incorporate more up-to-date user data, and from a broader swath of the US (or global) population. It would also be interesting to incorporate data from other sources (from Twitter, Instagram, etc.). It would also be very valuable to incorporate more observational data to complement the descriptive data provided by users. This could help in dealing with sources of possible bias. For example, added information on the click-through rate for different movie ads on Facebook would help us understand what users are actually interested in vs. what they say they are interested in. It would also be interesting to see whether the relationships we saw apply elsewhere: e.g., to action vs. horror video games.