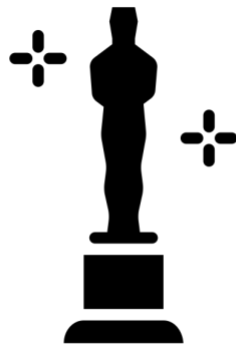


University of Texas at Austin
School of Information
INF 385T: Data Wrangling, SQL & Beyond
Supervisor: Dr. James Howison
December 11 2017



Project workflow and report

Project: Academy awards demographics and popularity

Lalwani, Gaurav
Seguel, Pedro

Project Folder:/home/gaurav_lalwani/project/
Holden Credentials
Username:gaurav_lalwani
password: 9987090417

1) Introduction	3
Goals	3
Datasets	3
Screenshots	4
2) ER Diagram and Relational Vocabulary	7
3) Relational Sketch Tables	7
Screenshots of tables on server	9
4) Workflow	13
Data cleansing, import and merge with Python	13
SQL queries and export	14
Visualization	15
5) Analysis	16
Social media popularity of oscar winning actors and their social demographics	16
Social media popularity, critic ratings of award winning movies and their characteristics.	21
Social media popularity, critic ratings of award winning movies and their characteristics.	24
6) Challenges, Outcomes and Learnings	26
7) Annexed	26
Python code	27
People table	27
B) Movies table	28
C) Awards table	29
D) Popularity_ratings table	31
E) movies_genres and genres tables	34
F) SQL queries and exporting them as CSV files	36

1) Introduction

Goals

This project explores the relationship between data from academy awards winners for best actor/actress and movies. More specifically our goals are to:

1. Analyze the relationship between social media popularity of oscar winning actors/actresses (Facebook likes for actors/actresses) and their social demographics (race/ethnicity, sexual orientation and religion).
2. Analyze the relationship between award winning movies in terms of IMDB recommendation (number of user reviews and IMDB rating scores) and social media popularity (facebook likes for the movie).
3. Analyze relations between actors' characteristics (race/ethnicity, sexual orientation, religion and social media popularity) and movie's popularity scores (metacritic ratings, imdb score)

Datasets

For this project we are using 5 dataset from 3 sources:

- **Crowdfunder “Academy awards demographics made by crowd work” (people.csv):** The data is based primarily in the input of crowdworkers in Crowdfunder crowdsourcing platform. Link: <https://www.crowdfunder.com/data-for-everyone/>
- **Kaggle, “5000+ movie data scraped from IMDB website” (movies_imdb.csv):** The data is based from the data extracted from Internet Movie Database (IMDb) and social media information about the movies (e.g. facebook likes). Link: <https://www.kaggle.com/deepmatrix/imdb-5000-movie-dataset>
- **Waterloo, “Dataset - Winners of the Oscars Award”, where we are using the “Directors”, “Actors” and Actresses” datasets (actor_waterloo.csv, actress_waterloo.csv, director_waterloo.csv):** The data is based in a scraper that was written in R and collected information about movies since 1928 from imdb.com and filmaffinity.com.
Link: <https://cs.uwaterloo.ca/~s255khan/oscars.html>

Screenshots

Movies CSV

```
gaurav_lalwani@holden:~/project$ head movies_imdb.csv
color,director_name,num_critic_for_reviews,duration,director_facebook_likes,actor_3_facebook_likes,actor_2_name,actor_1_facebook_likes,gross,genres,actor_1_name,movie_title,num_voted_users,cast_total_facebook_likes,actor_3_name,facnumber_in_poster,plot_keywords,movie_imdb_link,num_user_for_reviews,language,country,content_rating,budget,title_year,actor_2_facebook_likes,imdb_score,aspect_ratio,movie_facebook_likes,movie_names
Color,James Cameron,723,178,0,855,Joel David Moore,1000,760505847,Action|Adventure|Fantasy|Sci-Fi,CCH Pounder,Avatar ,886204,4834,Wes Studi,0,avatar|future|marine|native|paraplegic,http://www.imdb.com/title/tt0499549/?ref_=fn_tt_tt_1,3054,English,USA,PG-13,237000000,2009,936,7.9,1.78,33000,Avatar
Color,Gore Verbinski,302,169,563,1000,Orlando Bloom,40000,309404152,Action|Adventure|Fantasy,Johnny Depp,Pirates of the Caribbean: At World's End ,471220,48350,Jack Davenport,0,goddess|marriage ceremony|marriage proposal|pirate|singapore,http://www.imdb.com/title/tt0449088/?ref_=fn_tt_tt_1,1238,English,USA,PG-13,300000000,2007,5000,7.1,2.35,0,Pirates of the Caribbean: At World's End
```

People CSV

```
gaurav_lalwani@holden:~/project$ head people.csv
_unit_id,_golden,_unit_state,_trusted_judgments,_last_judgment_at,birthplace,birthplace:confidence,date_of_birth,date_of_birth:confidence,race_ethnicity,race_ethnicity:confidence,religion,religion:confidence,sexual_orientation,sexual_orientation:confidence,year_of_award,year_of_award:confidence,award,biourl,birthplace_gold,date_of_birth_gold,movie,person,race_ethnicity_gold,religion_gold,sexual_orientation_gold,year_of_award_gold
670454353,FALSE,finalized,3,2/10/15 3:45,"Chisinau, Moldova",1,30-Sep-1895,1,White,1,Na,1,Straight,1,1927,1,Best Director,http://www.nndb.com/people/320/000043191/,,,Two Arabian Knights,Lewis Milestone,,,,
670454354,FALSE,finalized,3,2/10/15 2:03,"Glasgow, Scotland",1,2-Feb-1886,1,White,1,Na,1,Straight,0.6842,1930,1,Best Director,http://www.nndb.com/people/626/000042500/,,,The Divine Lady,Frank Lloyd,,,,
670454355,FALSE,finalized,3,2/10/15 2:05,"Chisinau, Moldova",1,30-Sep-1895,1,White,1,Na,1,Straight,1,1931,0.6667,Best Director,http://www.nndb.com/people/320/000043191/,,,All Quiet on the Western Front,Lewis Milestone,,,,
670454356,FALSE,finalized,3,2/10/15 2:04,"Chicago, Il",1,23-Feb-1899,1,White,1,Na,1,Straight,1,1932,1,Best Director,http://www.nndb.com/people/544/000041421/,,,Skippy,Norman Taurog,,,,
```

Actor_waterloo CSV

```
gaurav_lalwani@holden:~/project$ head actor_waterloo.csv
name,year,nominations,rating,duration,genre1,genre2,release,metacritic,synopsis,
movie_names
The Theory of Everything,2014,5,7.7,123,Biography,Drama,January,72,A look at th
e relationship between the famous physicist Stephen Hawking and his wife.,The Th
eory of Everything
Dallas Buyers Club,2013,6,8,117,Biography,Drama,November,84,In 1985 Dallas elec
trician and hustler Ron Woodroof works around the system to help AIDS patients g
et the medication they need after he is diagnosed with the disease.,Dallas Buyer
s Club
Lincoln,2012,12,7.4,150,Biography,Drama,November,86,As the Civil War continues
to rage Americas president struggles with continuing carnage on the battlefield
as he fights with many inside his own cabinet on the decision to emancipate the
slaves.,Lincoln
The Artist,2011,10,8,100,Comedy,Drama,October,89,A silent movie star meets a yo
ung dancer but the arrival of talking pictures sends their careers in opposite d
irections.,The Artist
```

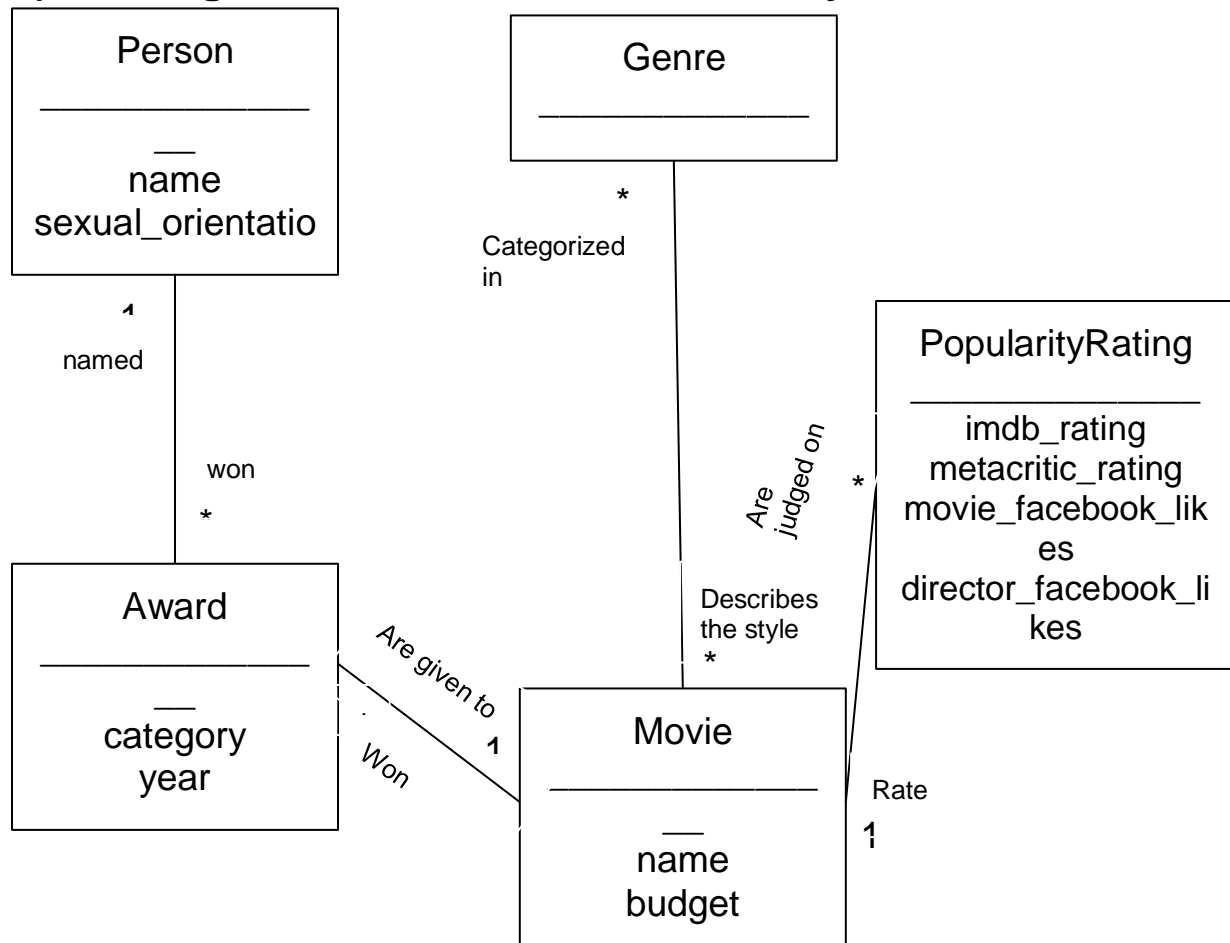
Actress_waterloo CSV

```
gaurav_lalwani@holden:~/project$ head actress_waterloo.csv
name,year,nominations,rating,duration,genre1,genre2,release,metacritic,synopsis,
movie_names
Still Alice,2014,,7.5,101,Drama,,February,72,A linguistics professor and her fa
mily find their bonds tested when she is diagnosed with Alzheimers Disease.,Stil
l Alice
Blue Jasmine,2013,3,7.3,98,Drama,,August,78,A New York socialite deeply trouble
d and in denial arrives in San Francisco to impose upon her sister. She looks a
million but isnt bringing money peace or love...,Blue Jasmine
Silver Linings Playbook,2012,8,7.8,122,Comedy,Drama,November,81,After a stint i
n a mental institution former teacher Pat Solitano moves back in with his parent
s and tries to reconcile with his ex-wife. Things get more challenging when Pat
meets Tiffany a mysterious girl with problems of her own.,Silver Linings Playboo
k
The Iron Lady,2011,2,6.4,105,Biography,Drama,January,54,An elderly Margaret Tha
tcher talks to the imagined presence of her recently deceased husband as she str
uggles to come to terms with his death while scenes from her past life from girl
hood to British prime minister intervene.,The Iron Lady
Black Swan,2010,5,8,108,Drama,Thriller,December,79,"A committed dancer wins the
lead role in a production of Tchaikovskys ""Swan Lake"" only to find herself st
ruggling to maintain her sanity.",Black Swan
The Blind Side,2009,2,7.7,129,Biography,Drama,November,53,The story of Michael
Oher a homeless and traumatized boy who became an All American football player a
nd first round NFL draft pick with the help of a caring woman and her family.,Th
```

Director Waterloo CSV

```
gaurav_lalwani@holden:~/project$ head director_waterloo.csv
name,year,nominations,rating,duration,genre1,genre2,release,metacritic,synopsis,
movie_names
Birdman,2014,9,7.8,119,Comedy,Drama,November,88,Illustrated upon the progress o
f his latest Broadway play a former popular actors struggle to cope with his cur
rent life as a wasted actor is shown.,Birdman
Gravity,2013,10,7.9,91,Sci-Fi,Thriller,October,96,A medical engineer and an ast
ronaut work together to survive after an accident leaves them adrift in space.,G
ravity
Life of Pi,2012,11,8,127,Adventure,Drama,November,79,A young man who survives a
disaster at sea is hurtled into an epic journey of adventure and discovery. Whi
le cast away he forms an unexpected connection with another survivor: a fearsome
Bengal tiger.,Life of Pi
The Artist,2011,10,8,100,Comedy,Drama,October,89,A silent movie star meets a yo
ung dancer but the arrival of talking pictures sends their careers in opposite d
irections.,The Artist
The King's Speech,2010,12,8,118,Biography,Drama,December,88,The story of King G
eorge VI of the United Kingdom of Great Britain and Northern Ireland his impromp
tu ascension to the throne and the speech therapist who helped the unsure monarc
h become worthy of it.,The King's Speech
The Hurt Locker,2009,9,7.6,131,Drama,History,July,94,During the Iraq War a Serg
eant recently assigned to an army bomb squad is put at odds with his squad mates
due to his maverick way of handling his work.,The Hurt Locker
Slumdog Millionaire,2008,10,8,120,Drama,Romance,January,86,"A Mumbai teen refle
```

2) ER Diagram and Relational Vocabulary



Person **has_many** Movie **through** Award
 Movie **has_many** Person **through** Award
 Award **belongs_to** Person
 Award **belongs_to** Movie

Movie **has_many** PopularityRating
 PopularityRating **belongs_to** Movie

Movie **has_and_belongs_to_many** Genre
 Genre **has_and_belongs_to_many** Movie

3) Relational Sketch Tables

people				
id	name	sexual_orientation	race	religion
1	Tom Hanks	Straight	White	Born-Again Christian

awards				
id	category	year	person_id	movie_id
1	Best Director	1994	1	1
2	Best Actor	1995	1	2

movies			
id	name	budget	country
1	Philadelphia	26000000	USA
2	Forrest Gump	55000000	USA

popularity_ratings							
id	imdb_rating	metacritic_rating	movie_facebook_likes	director_facebook_likes	actor_facebook_likes	no_of_nominations	movie_id
1	7.7	66	0	438	15000	5	1
2	8.8	82	59000	0	15000	13	2

genres

id	name
1	Comedy
2	Drama



























movies_genres		
id	genre_id	movie_id
1	1	1
2	5	2

Screenshots of tables on server

Movies Table

				id	name	budget	country
<input type="checkbox"/>	 Edit	 Copy	 Delete	1	Avatar	237000000	USA
<input type="checkbox"/>	 Edit	 Copy	 Delete	2	Pirates of the Caribbean: At World's End	300000000	USA
<input type="checkbox"/>	 Edit	 Copy	 Delete	3	Spectre	245000000	UK
<input type="checkbox"/>	 Edit	 Copy	 Delete	4	The Dark Knight Rises	250000000	USA

People Table

		id	name	sexual_orientation	race	religion
<input type="checkbox"/>	 Edit  Copy  Delete	1	Lewis Milestone	Straight	White	Na
<input type="checkbox"/>	 Edit  Copy  Delete	2	Frank Lloyd	Straight	White	Na
<input type="checkbox"/>	 Edit  Copy  Delete	3	Lewis Milestone	Straight	White	Na
<input type="checkbox"/>	 Edit  Copy  Delete	4	Norman Taurog	Straight	White	Na
<input type="checkbox"/>	 Edit  Copy  Delete	5	Frank Borzage	Straight	White	Roman Catholic
<input type="checkbox"/>	 Edit  Copy  Delete	6	Frank Lloyd	Straight	White	Na
<input type="checkbox"/>	 Edit  Copy  Delete	7	Frank Capra	Straight	White	Roman Catholic
<input type="checkbox"/>	 Edit  Copy  Delete	8	John Ford	Bisexual	White	Roman Catholic





























Awards Table

		id	year	category	person_id	movie_id	 1
<input type="checkbox"/>	 Edit  Copy  Delete	72	1998	Best Director	72		27
<input type="checkbox"/>	 Edit  Copy  Delete	246	2009	Best Supporting Actor	246		67
<input type="checkbox"/>	 Edit  Copy  Delete	87	2013	Best Director	80		221
<input type="checkbox"/>	 Edit  Copy  Delete	407	2005	Best Supporting Actress	338		255
<input type="checkbox"/>	 Edit  Copy  Delete	160	2001	Best Actor	160		281
<input type="checkbox"/>	 Edit  Copy  Delete	250	2013	Best Supporting Actor	247		294
<input type="checkbox"/>	 Edit  Copy  Delete	81	2007	Best Director	81		358
<input type="checkbox"/>	 Edit  Copy  Delete	33	1960	Best Director	15		364

PopularityRatings Table

id	imdb_rating	metacritic_rating ▼ 1	movie_facebook_likes	director_facebook_likes	actor_facebook_likes	no_of_nominations	movie_id
2606	8	100	11000	767	607	10	2606
3404	9	100	43000	0	14000	11	3404
244	8	96	147000	0	39	10	244
2155	8	96	0	607	813	8	2155
2641	8	94	16000	0	10000	9	2641
336	9	94	16000	0	5000	11	336
1858	9	93	41000	14000	14000	12	1858
2379	8	93	16000	869	692	11	2379

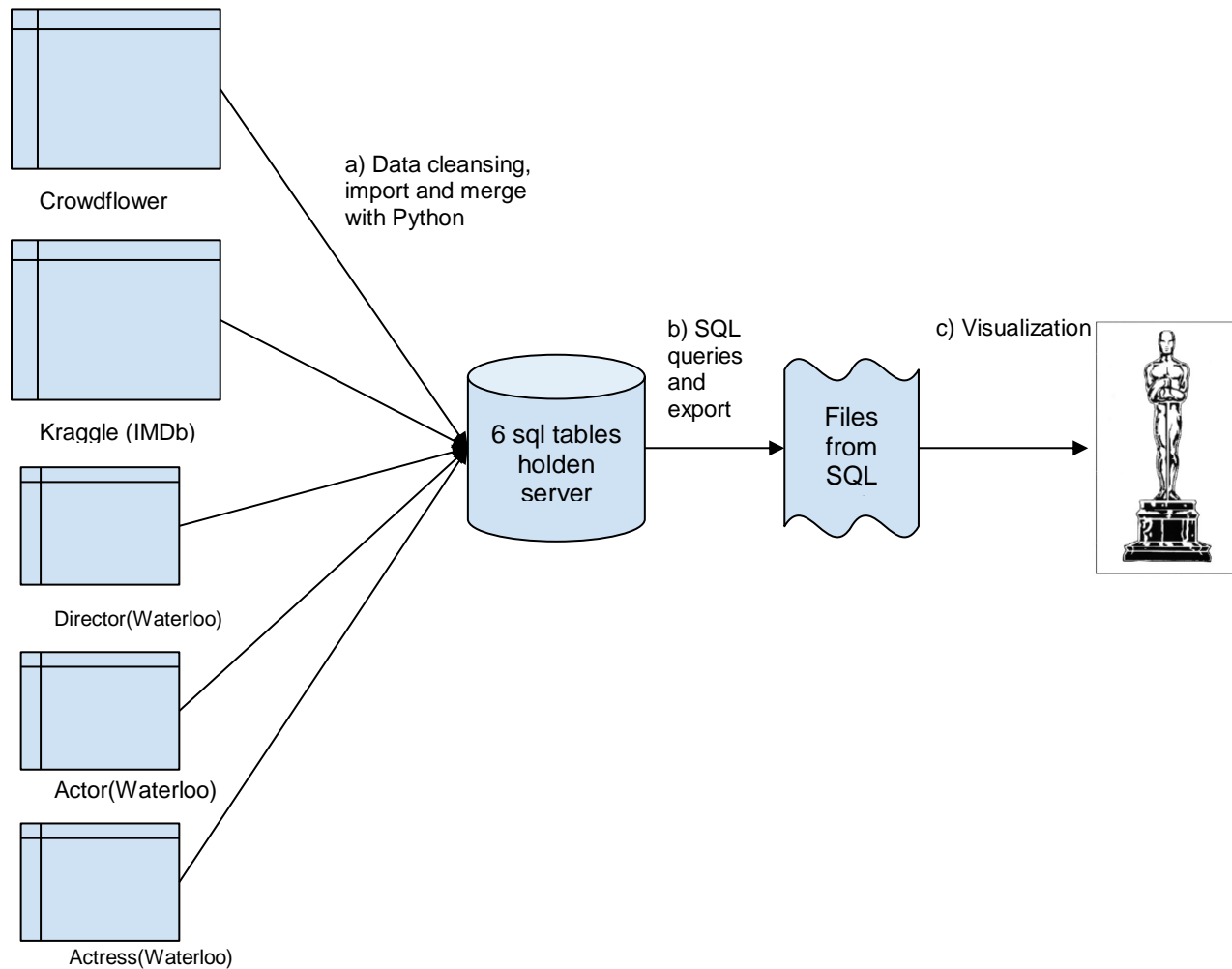
Genres Table

 ▼	id	name
<input type="checkbox"/>  Edit  Copy  Delete	1	Action
<input type="checkbox"/>  Edit  Copy  Delete	2	Adventure
<input type="checkbox"/>  Edit  Copy  Delete	3	Fantasy
<input type="checkbox"/>  Edit  Copy  Delete	4	Sci-Fi
<input type="checkbox"/>  Edit  Copy  Delete	5	Thriller
<input type="checkbox"/>  Edit  Copy  Delete	6	Documentary
<input type="checkbox"/>  Edit  Copy  Delete	7	Romance
<input type="checkbox"/>  Edit  Copy  Delete	8	Animation
<input type="checkbox"/>  Edit  Copy  Delete	9	Comedy

Movies_Genres Table

<div><div><div><div><div></div><div></div></div><div><div></div><div></div></div><div><div></div><div></div></div></div><div></div></div></div>						id	movie_id	genre_id
<div><div><div></div></div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Edit</div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Copy</div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Delete</div></div>	1	1	1					
<div><div><div></div></div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Edit</div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Copy</div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Delete</div></div>	2	1	2					
<div><div><div></div></div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Edit</div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Copy</div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Delete</div></div>	3	1	3					
<div><div><div></div></div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Edit</div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Copy</div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Delete</div></div>	4	1	4					
<div><div><div></div></div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Edit</div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Copy</div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Delete</div></div>	5	2	1					
<div><div><div></div></div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Edit</div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Copy</div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Delete</div></div>	6	2	2					
<div><div><div></div></div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Edit</div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Copy</div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Delete</div></div>	7	2	3					
<div><div><div></div></div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Edit</div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Copy</div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Delete</div></div>	8	3	1					
<div><div><div></div></div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Edit</div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Copy</div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>Delete</div></div>	9	3	2					

4) Workflow



a) Data cleansing, import and merge with Python

We merged all databases using python in order to create the 6 tables, with respect to the ER Diagram presented above, in the Holden Server.

The details of the Python code that was used is presented in the Annexed section. The work process that was used is described below:

- *People table:* We use the people.csv from Crowdflower to add the name, sexual orientation, race and religion of each person that won an award. These people could be actors, actresses or directors.
- *Movies table:* We use the movies_imdb.csv from IMDB to add the name, budget and country of all the movies that were in that dataset (more than 5000). The command `row["movie_title"]=row["movie_title"].strip()` was used to remove white spaces from the start and end of each movie name and the command

`row["movie_title"]=row["movie_title"][:-1]` was used to remove the special character at the end of each movie name. The movies csv also had some duplicate movie_names which were removed using the logic of inserting a new movie only when matching the same movie name in the table yields no results.

- *Awards table:* We use the people.csv from Crowdfunder to add the information related to category of the award (best director, actor or actress) and the year of the award. We related this information to our people and movie tables using foreign keys: people_id and movie_id. For matching the movie_id we traversed through the movies table and if we didn't find a match for the movie_name (that means the movie name was not present in the movies table) we inserted the name in the movies table and stored the newly created id as our foreign key.
- *Popularity ratings:* In order to create this table we had to add information from multiple CSVs. The information related to imdb_rating, movie_facebook_likes, director_facebook_likes and actor_facebook_likes was extracted from movies_imdb.csv dataset. The information related to metacritic_rating and no_of_nominations was extracted from the three Waterloo datasets: actor_waterloo.csv, actress_waterloo.csv, director_waterloo.csv. These three datasets had the same structure, where each row represents a movie that won a specific award (e.g. in the case of actor_waterloo.csv the movies that won the best actor award). In order to create the movie_id, all the datasets were matched using the names of the movies. If the movie wasn't present in the movies table the same movie was created using an insert statement and the id of the newly created movie was extracted to be used in this table. After the first step of storing the movie_id for each movie name we used the same id to update all other measures/values based on our movie_id.
- *Genres table and Movies_genres table:* These tables were built using the genres from the data movies_imdb.csv. Some of the movies had multiple genres separated by "|" (e.g. "Action|Adventure|Fantasy|Sci-Fi"). We used the split function to separate the genres. So at first we had a dictionary of genres which was later converted to a list of genres using a for loop. Using another for loop we can traverse through each list and get individual genres and compare them with the genres in the genre table.
- *Note:* The genre table was empty and we haven't manually inserted the genres. The insertion happens when individual genres (derived from split function) are compared to the genres table and they don't find a match. So initially all the distinct genres get inserted on comparison which later give ids when duplicate genres are compared.

b) SQL queries and export

For the analysis we decided to use specific SQL queries from the tables that were created in the server in the previous step and we exported them in csv files. We run the queries and export the tables from the terminal using Python code (more details in the Annexed section part F).

For the analysis of actor and actresses characteristics with their popularity ratings we used the following query:

```

SELECT people.name, people.race, people.sexual_orientation, people.religion,
popularity_ratings.actor_facebook_likes
FROM people, movies, awards, popularity_ratings
WHERE awards.category IN ("Best Actor", "Best Actress")
AND people.id = awards.person_id
AND awards.movie_id = movies.id
AND movies.id = popularity_ratings.movie_id

```

For the analysis of award winning movies characteristics and their popularity ratings and recommendation scores we used the following query:

```

SELECT people.name, people.race, people.sexual_orientation, people.religion, movies.name, movies.budget,
movies.country, popularity_ratings.actor_facebook_likes, popularity_ratings.director_facebook_likes,
popularity_ratings.movie_facebook_likes, popularity_ratings.imdb_rating, popularity_ratings.no_of_nominations,
popularity_ratings.metacritic_rating, awards.category, awards.year, genres.name
FROM people, movies, awards, popularity_ratings, genres, movies_genres
WHERE people.id = awards.person_id
AND awards.movie_id = movies.id
AND movies.id = popularity_ratings.movie_id
AND movies.id = movies_genres.movie_id
AND genres.id = movies_genres.genre_id

```

c) Visualization

We used Tableau as our analysis tool, though neither of the group members had worked with it before. However, we are aware of the relevance that visualizations tools are havin in data driven marketplaces and research, so we decided to learn how to use it. We learned the basic procedures from the screencasts tutorials provided by the platform that were focus in how to read the data, as well as general functions to visualize and update the data.

We learned about the tool using youtube tutorials. Like any tool learning from a new API to start from scratch we took a lot of help from the tableau documentation <http://onlinehelp.tableau.com/current/pro/desktop/en-us/help.htm> to guide us through the basics of each step. Also it was fun to explore a new visualization tool to play around with random analysis of our data and come up with new methods to reach our desired goal.

Exporting the csv files from the previous step we read them as text files in Tableau and started to run different type of visualization graphs. We decided to display the information in terms of bar charts, highlight tables, text tables and maps.

5) Analysis

The analysis is divided in three sections to respond to our three main goals. First, the analysis of the relationship between social media popularity of oscar winning actors (facebook likes for actors/actresses) and their social demographics (race/ethnicity, sexual orientation and religion). Second, the analyze of award winning movies in terms of rating scores and social media popularity. Finally, the analyze relations between actors' characteristics (race/ethnicity, sexual orientation, religion and social media popularity) and movie's popularity scores (metacritic ratings, imdb score).

a) Social media popularity of oscar winning actors and their social demographics

Tables 1,2 and 3 present the distribution of actors and actresses that won an award by Sexual Orientation, Religion and Race. Table 1 and 3 present a concentration of awards in actors and actresses identified as Straight and White from a total of 74 people.

It should be noticed that in Table and Graph 1 no results are present for "Gay" actors. The reason of this is that, the data from the dataset (Crowdfunder dataset) had one male winning award actor which was recognized with this sexual orientation for his role in the movie "The Private Life of Henry VIII", but this movie was not present in our final merged table for data analysis, which in other words mean that no actor was present with the sexual orientation "Gay" in our data analysis.

Table 1. Number
of actors by
Sexual
Orientation

Sexual Orientation	
Bisexual	5.41%
Lesbian	1.35%
Matter of Dispute	1.35%
Na	1.35%
Straight	90.54%

Table 2. Number of actors by Religion

Religion	
Atheist	5.41%
Baptist	1.35%
Born-Again Christian	2.70%
Deist	1.35%
Disciples of Christ	1.35%
Hindu	1.35%
Jewish	6.76%
Lutheran	2.70%
Na	54.05%
Protestant	4.05%
Quaker	1.35%
Roman Catholic	17.57%

Table 3.
Number of actors by Race

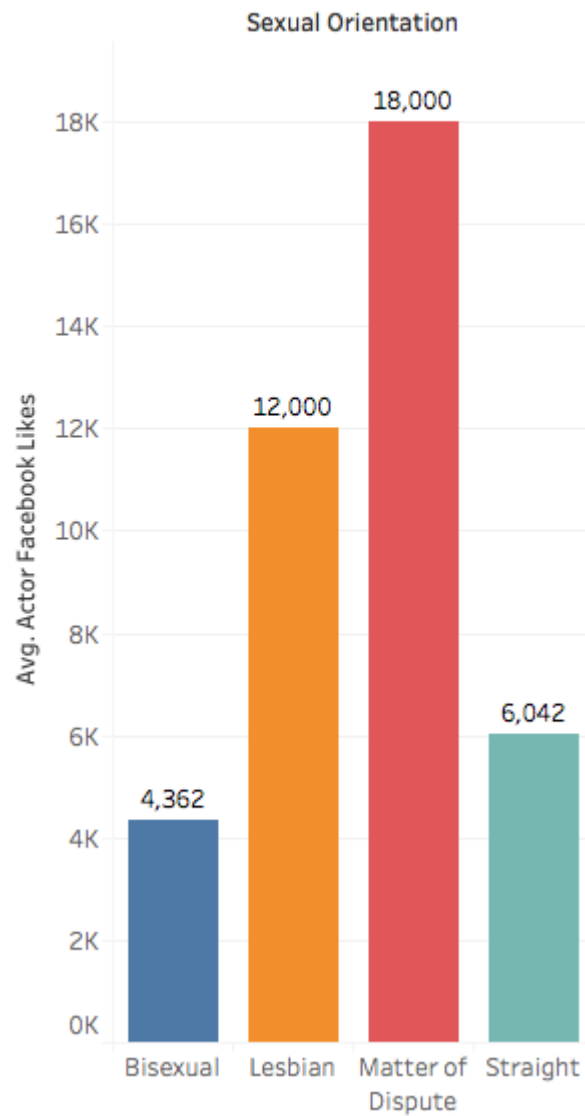
Race	
Black	4.05%
Middle Eastern	1.35%
Multiracial	2.70%
White	91.89%

Graphs 1, 2 and 3 present the information of actor's social demographics Vs their social media popularity. In Graph 1, it is seen for some actors and actresses whose sexual orientation is a Matter of Dispute¹ are the most popular with 18,000 Facebook likes on average. Actresses identified as Lesbian have more Facebook likes on average (12,000), than actor and actresses identified as straight (6,042) or bisexual (4,362). In Graph 2 and in terms of religion preferences, it's shown that actors and actresses that are recognized as Born-again Christians, Deist and Protestant are the most popular in social media with 12,970, 12,000 and 10,311 facebook likes on average respectively. Baptist and Quaker actors and actresses are the least popular with 904 and 545 Facebook likes on average respectively. Finally, in Graph 3 and in relation to race, it's shown that actors and actresses that are identified as multiracial, black and white are the most popular in social media with 6,773, 6,635 and 6,488, Facebook likes on average. However, Middle

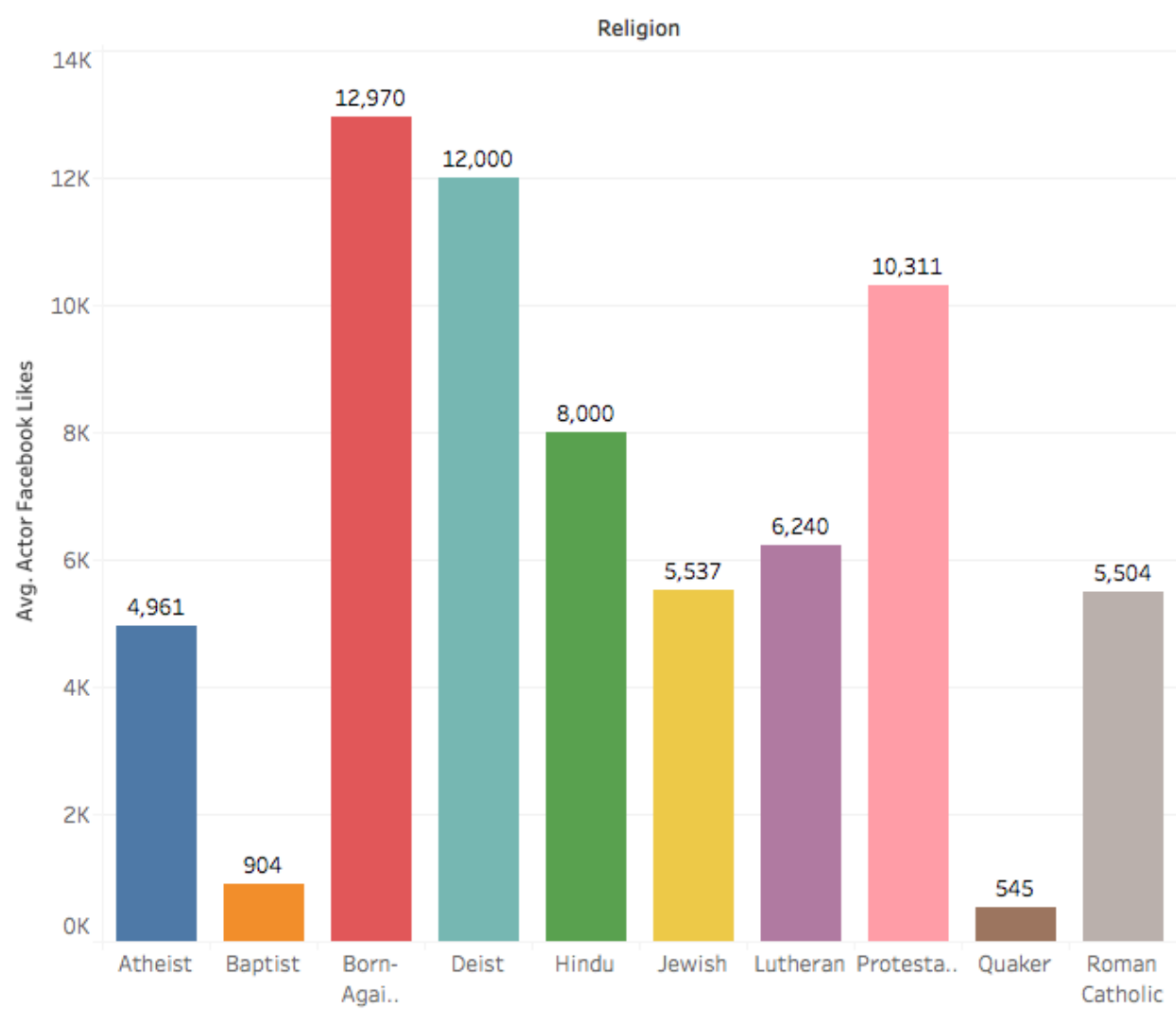
¹ Is a Matter of Dispute -This data was given by crowdworkers and hence some of the data can be well-argued or is not justifiable.

Eastern award winners register a low popularity in social media with 692 Facebook likes on average.

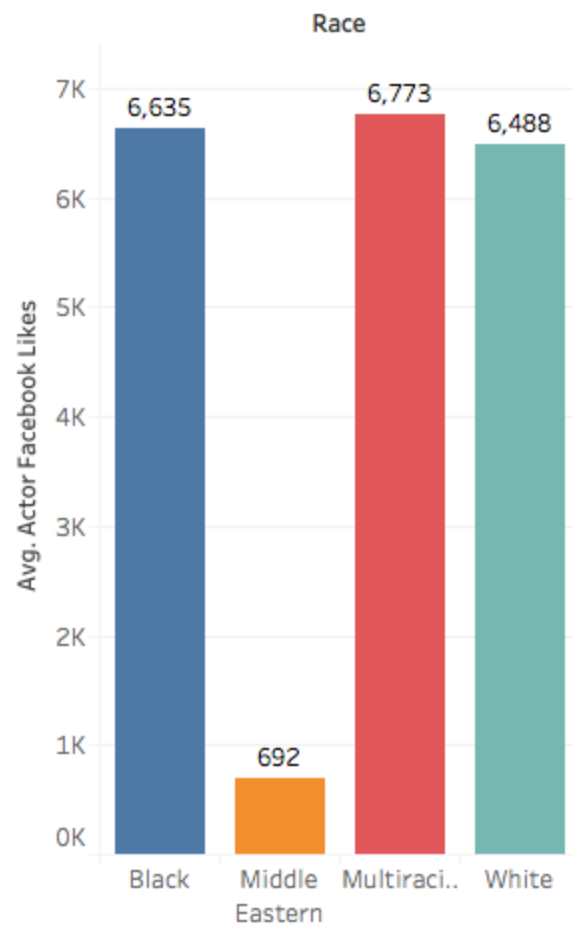
Graph 1. Actor's facebook likes
Vs Sexual orientation



Graph 2. Actor's facebook likes Vs Religion



Graph 3. Actor's facebook likes Vs Race



- b) Social media popularity, critic ratings of award winning movies and their characteristics.

Table 4 presents the distribution of data from movies that won an award by the award Category. Movies that have won at least the Best Supporting Actor and Actress awards have the biggest budget on average with 24,747,459 and 23,860,964 respectively. This is different in terms of rating scores and popularity. IMDb scores are very similar across award categories, with an average score of 8, being the movies that won the Best Supporting Actor category the ones with the lowest average score (7). In relation to metacritic, movies that won Best Actress and Actor got, on average, the best ratings: 52 and 50 respectively. Movies that won Best Actress category are the most popular in social media reaching an average of 14,699 of Facebook likes, follow up the ones that won the categories of Best Director and Supporting Actor with more than 10,000 facebook likes on average.

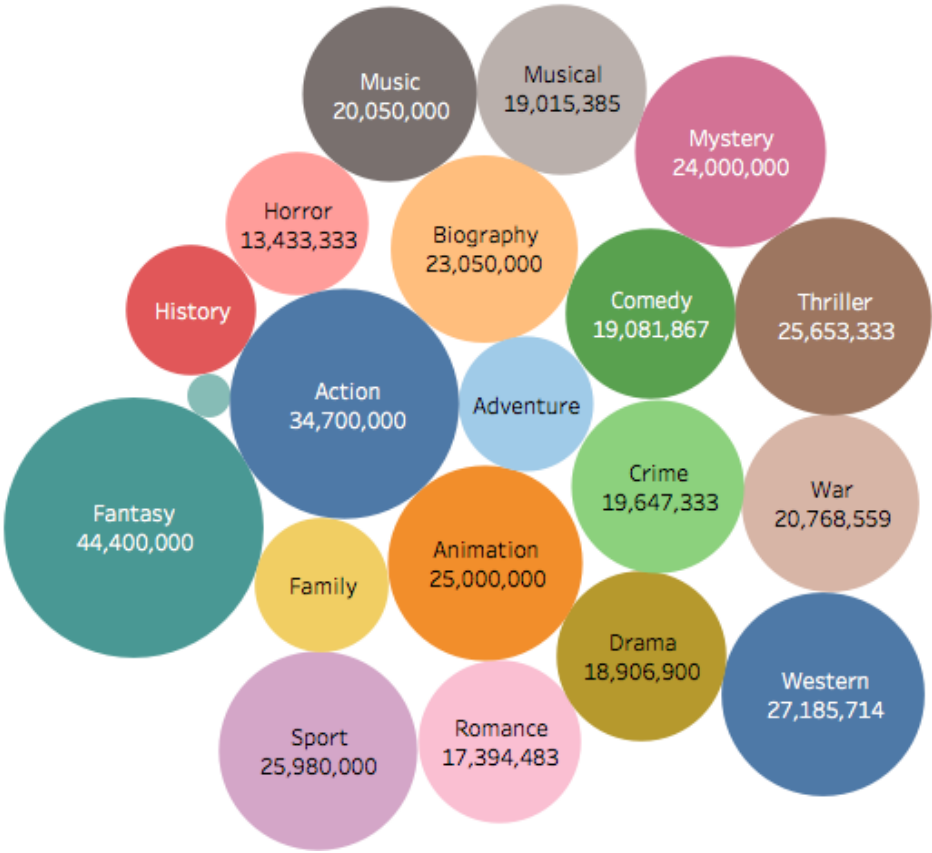
Table 4. Average budget, IMDb score, metacritic and facebook likes of movies by category of the award

Category	Avg. Budget	Avg. Imdb Rating	Avg. Metacritic Rating	Avg. Movie Facebook Likes
Best Actor	14,489,576	8	50	9,494
Best Actress	18,124,337	8	52	14,669
Best Director	17,616,249	8	44	10,702
Best Supporting Actor	24,747,459	7	25	10,423
Best Supporting Actress	23,860,964	8	32	7,527

Graph 4 and Map 1 shows the average of the budget of the awards winning movies by genre and country of production. In terms of genres, fantasy, action, western and animation movies have the biggest budgets on average, with more than 25,000 USD on average. In terms of countries, awards movies that are from Germany have the biggest budget, follow up by movies from France and the United States.

Graph 5 and Map 2 show the average of Facebook likes of the awards winning movies by genre and country of production. In terms of genres, western, horror, fantasy and biography movies have the biggest number of Facebook likes on average, with more than 16,000 on average. In terms of countries, awards winning movies from France and Spain have the biggest number of Facebook likes on average.

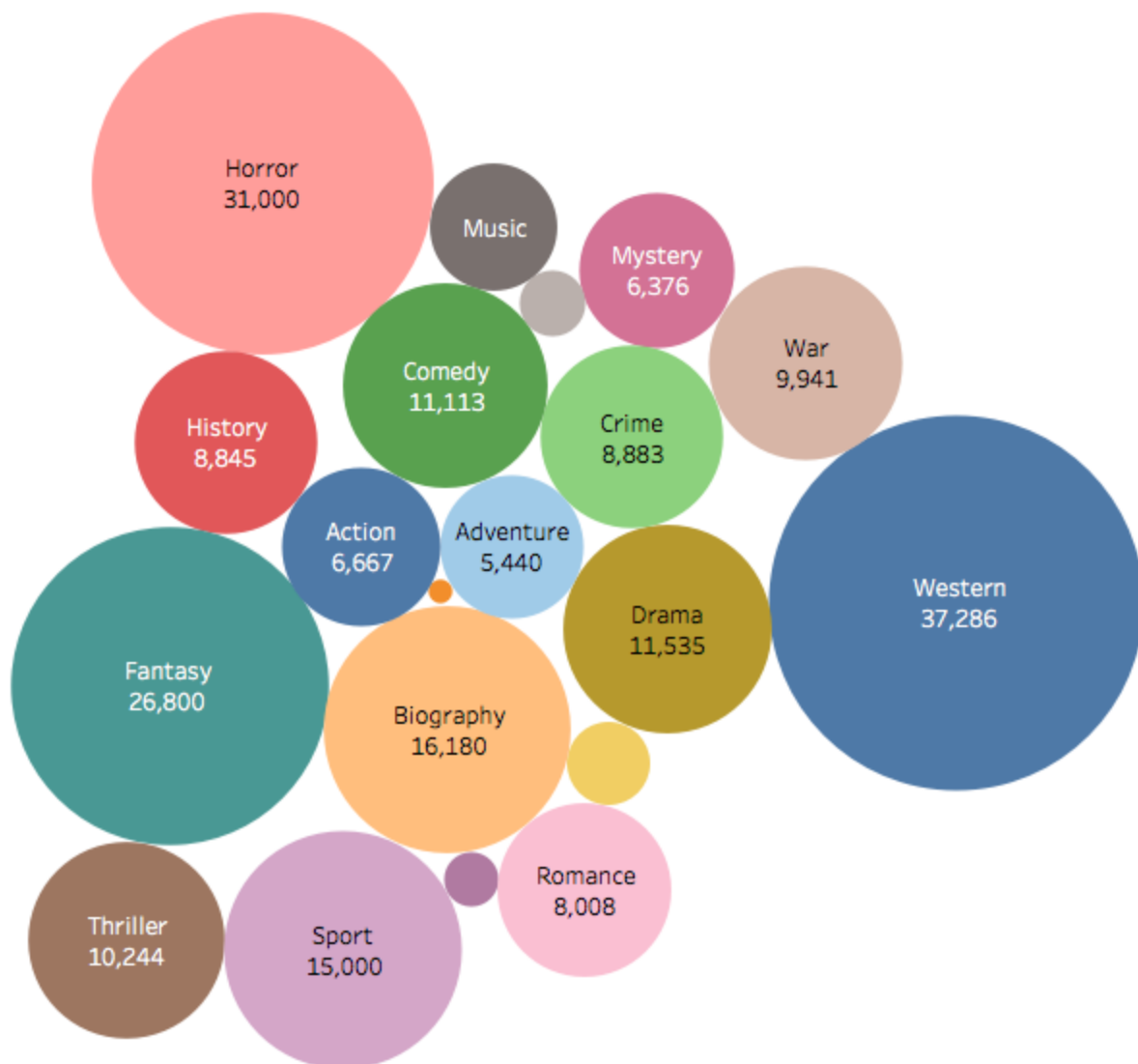
Graph 4. Average budget by Genre



Map 1. Average budget by Country



Graph 5. Average Facebook likes of the movie by Genre



Map 2. Average Facebook likes of the movie by Country



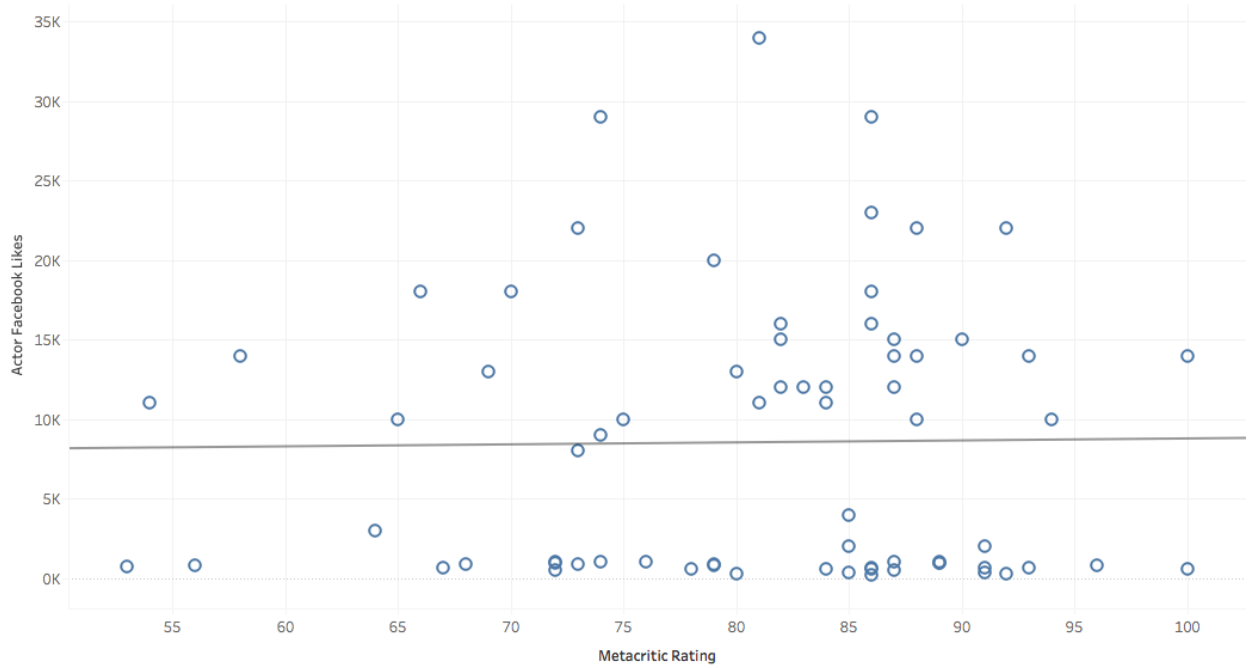
c) Social media popularity, critic ratings of award winning movies and their characteristics.

As a stretch goal, we wanted to explore and test if there were observable relationships between data from people involved in winning award movies and data from the movies itself. We tested this with simple correlations and observe their trends. We are aware that a more robust statistical should consider the number of our sample, the variance, as well as control variables.

Graph 6 presents a scatter plot that shows the relationship between Facebook Likes of Actors, that is assigned by fans in social media, and Metacritic rating, that is assigned by experts. However, the p-value of the observed trend is 0.90, therefore, there's no statistical evidence, following the conventional standards², that there is a relation between two variables.

² $P < 0.05$, $P < 0.01$, $P < 0.001$

Graph 6. Actor Facebook likes Vs Metracritic rating



Metacritic Rating vs. Actor Facebook Likes. The view is filtered on Exclusions (Actor Facebook Likes, Metacritic Rating), which keeps 96 members.

6) Challenges, Outcomes and Learnings

This project presented a number of challenges and learnings:

Challenge: Dealing with 5 different dataset of 3 different sources, as well as modelling data from this data sources to respond general research questions. This included finding links between different datasets and model our database which when traversed through would answer all possible answers necessary for the overall analysis of this project.

Outcome: We came up with a robust model which helped us gain insight in the detailed analysis of our research questions. We linked different datasets on movie names and used foreign keys to link multiple tables in our ER diagram

Learning: Data modelling techniques, ER modelling.

Challenge: With a number of repeated trial error methods it was difficult to predict the primary cause of our failing joins. It was later discovered that we need to pre-process our data before pipelining it to the source. The main faults include white-spaces at the end of some columns and presence of special characters at the end of each movie name.

Outcome: Our initial approach was to use our excel skills to handle these issues. But later it was evident that this approach could only provide temporary results and would fail if new data of the same kind is to be introduced in our dataset. So we changed our code in python to take care of the data discrepancies at the code level rather than performing them manually.

Learning: Excel, Python, Database management skills.

Challenge: One of the major challenges and also a stretch goal for us was the genre column since some movies had multiple genres separated by '|'. We were needed to prepare a separate genre table and also a junction table due to the many-to-many relationship between movies and genres.

Outcome: We used the split function in python to separate the genres and by using loops we separated each individual genre and matched them with the corresponding movie name. While doing so we consecutively inserted data in the genres and the junction table.

Learning: Python and SQL skills.

Challenge: Our final challenge in this project was the analysis part which involved learning of a new API from scratch and also export results from SQL queries in python.

Outcome: After learning from documentation, videos and trying different analysis with Tableau we came up with our final analysis required for the project.

Learning: Python, SQL and Tableau skills.

7) Annexed

Our code to read and write the csvs is written in five parts for the six different tables, as well as the code that run the sql queries for analysis and export the tables as CSVs is presented in the last part.

Python code

A) People table

```
#####  
# This file shows how to execute queries to create people table  
#####  
  
import pymysql  
import csv  
  
# Open the connection to the database  
connection = pymysql.connect(host="localhost",  
                             user="gaurav_lalwani",  
                             passwd="9987090417",  
                             db="gaurav_lalwani_oscar_movies",  
                             autocommit=True,  
                             cursorclass=pymysql.cursors.DictCursor)  
  
# # We get a cursor from the connection to talk to the server  
cursor = connection.cursor() # now we can use cursor.execute() to talk to the server  
  
#Open the people.csv to read the data  
with open('people.csv') as csvfile:  
    myCSVReader = csv.DictReader(csvfile)  
  
    # Placeholders are same as in our csv file  
    sql = """INSERT INTO people(name,sexual_orientation,race,religion)  
            VALUE (%(person)s,%(sexual_orientation)s,%(race_ethnicity)s,%(religion)s)"""  
  
    for row in myCSVReader:  
        cursor.execute(sql,row)
```

B) Movies table

```
#####
```

```
# This file shows how to execute queries to create movies table
```

```
#####
```

```
import pymysql
```

```
import csv
```

```
# Open the connection to the database
```

```
connection = pymysql.connect(host="localhost",  
                             user="gaurav_lalwani",  
                             passwd="9987090417",  
                             db="gaurav_lalwani_oscar_movies",  
                             autocommit=True,  
                             cursorclass=pymysql.cursors.DictCursor)
```

```
# # We get a cursor from the connection to talk to the server
```

```
cursor = connection.cursor() # now we can use cursor.execute() to talk to the server
```

```
#Open the movies_imdb.csv to read the data
```

```
with open('movies_imdb.csv') as csvfile:
```

```
    myCSVReader = csv.DictReader(csvfile)
```

```
#SQL query to insert a new movie into our database
```

```
sql = """INSERT INTO movies(name,budget,country)  
        VALUE %(movie_title)s,%(budget)s,%(country)s  
        """
```

```
#SQL query to find the id of the movie that matches the name from the csv
```

```
sql_select_movie = "SELECT id from movies WHERE name = %(movie_title)s"
```

```
#SQL query to check the name of the movie in the table to avoid duplicates
```

```
sql_select_movie_name = "SELECT name from movies WHERE name = %(movie_title)s"
```

```
#SQL query to insert the data into popularity_ratings table
```

```
sql_insert = """INSERT INTO  
popularity_ratings(imdb_rating,movie_facebook_likes,director_facebook_likes,actor_facebook_likes,movie_id)
```

```

VALUE
(%(imdb_score)s,%(movie_facebook_likes)s,%(director_facebook_likes)s,%(actor_1_facebook
_likes)s,%(movie_id)s)
"""

```

for row in myCSVReader:

```

#Removes extra white spaces from both sides
row["movie_title"]=row["movie_title"].strip()
#row["movie_title"]=row["movie_title"][:-2]
cursor.execute(sql_select_movie, row)
results = cursor.fetchall()

#Checking for duplicates
if (len(results) == 0):
    cursor.execute(sql,row)

cursor.execute(sql_select_movie, row)
results = cursor.fetchone()
movie_id = results['id']

param_dict = {
    'movie_id': movie_id,
    'imdb_score': row['imdb_score'],
    'movie_facebook_likes' : row['movie_facebook_likes'],
    'director_facebook_likes' : row['director_facebook_likes'],
    'actor_1_facebook_likes' : row['actor_1_facebook_likes']
}

cursor.execute(sql_insert, param_dict)

```

C) Awards table

```

#####
# This file shows how to execute queries to create table awards
#####

import pymysql
import csv

```

```

import pprint

# Open the connection to the database
connection = pymysql.connect(host="localhost",
                             user="gaurav_lalwani",
                             passwd="9987090417",
                             db="gaurav_lalwani_oscar_movies",
                             autocommit=True,
                             cursorclass=pymysql.cursors.DictCursor)

# # We get a cursor from the connection to talk to the server
cursor = connection.cursor() # now we can use cursor.execute() to talk to the server

#SQL query to get the id of the person that matches the name from the csv
sql_select_person = "SELECT id from people WHERE name = %(person)s"

#SQL query to get the id of the movie that matches the movie name from the csv
sql_select_movie = "SELECT id from movies WHERE name = %(movie)s"

#SQL query to insert a new movie into the database
sql_insert_new_movie="""INSERT INTO movies(name)
                        VALUE %(movie)s)
"""

#SQL query to insert data in the awards table
sql = """INSERT INTO awards(year,category,person_id,movie_id)
        VALUE %(year_of_award)s,%(award)s,%(person_id)s,%(movie_id)s)
"""

#Open the people.csv to read the data
with open('people.csv') as csvfile:

    myCSVReader = csv.DictReader(csvfile)

    for row in myCSVReader:
        cursor.execute(sql_select_person, row)
        results = cursor.fetchone()
        person_id = results['id']

        #Algorithm to store the id of the movie if it exists or insert the new movie and get the newly
        created id
        cursor.execute(sql_select_movie, row)
        results = cursor.fetchall()
        if (len(results) == 0):

```

```

        cursor.execute(sql_insert_new_movie,row)
        movie_id = cursor.lastrowid
    else:
        movie_id = results[0]["id"]

    param_dict = {'person_id': person_id,
                  'movie_id': movie_id,
                  'year_of_award': row['year_of_award'],
                  'award' : row['award'],
                  }

    cursor.execute(sql, param_dict)

```

D) Popularity_ratings rable

#####

This file shows how to execute queries to create popularity_ratings table

#####

```

import pymysql
import csv

```

Open the connection to the database (must be one you can write to!)

```

connection = pymysql.connect(host="localhost",
                             user="gaurav_lalwani",
                             passwd="9987090417",
                             db="gaurav_lalwani_oscar_movies",
                             autocommit=True,
                             cursorclass=pymysql.cursors.DictCursor)

```

We get a cursor from the connection to talk to the server

cursor = connection.cursor() # now we can use cursor.execute() to talk to the server

#SQL query to get the id of the person that matches the person name from the csv

sql_select_person = "SELECT id from people WHERE name = %(person)s"

#SQL query to find the id of the movie that matches the name from the csv

sql_select_movie = "SELECT id from movies WHERE name = %(name)s"

#SQL query to insert a new movie into our database

```

sql_insert_new_movie="""INSERT INTO movies(name)
                        VALUE (%(name)s)

```

```
"""
```

```
#SQL query to update the data in the popularity ratings table
sql_update = """UPDATE popularity_ratings SET metacritic_rating = %(metacritic)s,
no_of_nominations= %(nominations)s,
movie_id = %(movie_id)s
WHERE movie_id = %(movie_id)s
"""
```

```
#Open the actor_waterloo.csv to read the data
with open('actor_waterloo.csv') as csvfile:
```

```
    myCSVReader = csv.DictReader(csvfile)
```

```
    for row in myCSVReader:
        #Removes extra white spaces from both sides
        row["name"]=row["name"].strip()
```

```
        #Algorithm to store the id of the movie if it exists or insert the new movie and get the newly
        created id
```

```
        cursor.execute(sql_select_movie, row)
        results = cursor.fetchall()
        if (len(results) == 0):
            cursor.execute(sql_insert_new_movie,row)
            movie_id = cursor.lastrowid
        else:
            movie_id = results[0]["id"]
```

```
        param_dict = {
            'movie_id': movie_id,
            'nominations': row['nominations'],
            'metacritic' : row['metacritic']}
```

```
        cursor.execute(sql_update, param_dict)
```

```
#Open the actress_waterloo.csv to read the data
with open('actress_waterloo.csv') as csvfile:
```

```
    myCSVReader = csv.DictReader(csvfile)
```

```
    for row in myCSVReader:
        #Removes extra white spaces from both sides
        row["name"]=row["name"].strip()
```



```
#Algorithm to store the id of the movie if it exists or insert the new movie and get the newly created id
```

```
cursor.execute(sql_select_movie, row)
results = cursor.fetchall()
if (len(results) == 0):
    cursor.execute(sql_insert_new_movie,row)
    movie_id = cursor.lastrowid
else:
    movie_id = results[0]["id"]
```

```
param_dict = {
    'movie_id': movie_id,
    'nominations': row['nominations'],
    'metacritic' : row['metacritic']}
```

```
cursor.execute(sql_update, param_dict)
```

```
#Open the director_waterloo.csv to read the data
with open('director_waterloo.csv') as csvfile:
```

```
myCSVReader = csv.DictReader(csvfile)
```

```
for row in myCSVReader:
```

```
    #Removes extra white spaces from both sides
    row["name"]=row["name"].strip()
```

```
    #Algorithm to store the id of the movie if it exists or insert the new movie and get the newly created id
```

```
    cursor.execute(sql_select_movie, row)
    results = cursor.fetchall()
    if (len(results) == 0):
        cursor.execute(sql_insert_new_movie,row)
        movie_id = cursor.lastrowid
    else:
        movie_id = results[0]["id"]
```

```
    param_dict = {
        'movie_id': movie_id,
        'nominations': row['nominations'],
        'metacritic' : row['metacritic']}
```

```
    cursor.execute(sql_update, param_dict)
```

E) movies_genres and genres tables

#####

This file shows how to execute queries to create movies_genres table

#####

```
import pymysql
```

```
import csv
```

```
import pprint
```

```
# Open the connection to the database
```

```
connection = pymysql.connect(host="localhost",
                             user="gaurav_lalwani",
                             passwd="9987090417",
                             db="gaurav_lalwani_oscar_movies",
                             autocommit=True,
                             cursorclass=pymysql.cursors.DictCursor)
```

```
# # We get a cursor from the connection to talk to the server
```

```
cursor = connection.cursor()
```

```
#SQL query to get the id of the movie that matches the movie name from the csv
```

```
sql_select_movie = "SELECT id from movies WHERE name = %(movie_title)s"
```

```
#SQL query to get the id of the genre that matches the genre name from the csv
```

```
sql_select_genre = "SELECT id from genres WHERE name = %(genre_name)s"
```

```
#SQL query to insert a new movie into the database
```

```
sql_insert_new_movie="""INSERT INTO movies(name)
                        VALUE (%(movie_names)s)
                        """
```

```
#SQL query to insert a new genre into the database
```

```
sql_insert_new_genre="""INSERT INTO genres(name)
                        VALUE (%(genre_name)s)
                        """
```

```
#SQL query to insert data into the movies_genres table
```

```
sql = """INSERT INTO movies_genres(genre_id,movie_id)
        VALUE (%(genre_id)s,%(movie_id)s)
        """
```

```
#Open the movies_imdb.csv to read the data
```

```
with open('movies_imdb.csv') as csvfile:
```

```

myCSVReader = csv.DictReader(csvfile)

for row in myCSVReader:
    #Removes extra white spaces from both sides
    row["movie_title"]=row["movie_title"].strip()

    #Algorithm to store the id of the movie if it exists or insert the new movie and get the newly
    created id
    cursor.execute(sql_select_movie, row)
    results = cursor.fetchall()
    if (len(results) == 0):
        cursor.execute(sql_insert_new_movie,row)
        movie_id = cursor.lastrowid
    else:
        movie_id = results[0]["id"]

    #Genres in the genre column are multiple names separated by | eg:-
    Comedy|Action|Adventure
    genres = row['genres'].split("|")

    #Once we split the genres we have a list for each movie and we need to traverse through
    each list
    for genre_str in genres:

        param_dict = { 'genre_name': genre_str}

        #Algorithm to store the id of the genre if it exists or insert the new movie and get the
        newly created id
        cursor.execute(sql_select_genre, param_dict)
        results = cursor.fetchall()
        if (len(results) == 0):
            cursor.execute(sql_insert_new_genre,param_dict)
            genre_id = cursor.lastrowid
        else:
            genre_id = results[0]["id"]

        param_dict1 = {'genre_id': genre_id,
                       'movie_id': movie_id
                       }

        cursor.execute(sql,param_dict1)

```

F) SQL queries and exporting them as CSV files

#####

This file shows how to execute sql queries and export CSV files

#####

pymysql.cursors is the library that will talk to mysql

import pymysql

csv helps us write out the csv files.

import csv

convenience methods for debugging

import pprint

First set up the connection to the server

Open the connection to the database (must be one you can write to!)

```
connection = pymysql.connect(host="localhost",          # your host, usually localhost
                             user="gaurav_lalwani",    # your username
                             passwd="9987090417",     # your password
                             db="gaurav_lalwani_oscar_movies", # name of the db
                             autocommit=True,         # removes a step in queries
                             cursorclass=pymysql.cursors.DictCursor)
```

as with opening a file we can use with to open the connection

the cursor is the object through which we talk to the sql server.

with connection.cursor() as cursor:

```
    sql = """
        SELECT people.name, people.race, people.sexual_orientation, people.religion,
        popularity_ratings.actor_facebook_likes
        FROM people, movies, awards, popularity_ratings
        WHERE awards.category IN ("Best Actor", "Best Actress")
        AND people.id = awards.person_id
        AND awards.movie_id = movies.id
        AND movies.id = popularity_ratings.movie_id
    """
```

```
    sql2 = """
        SELECT people.name, people.race, people.sexual_orientation, people.religion,
        movies.name, movies.budget, movies.country, popularity_ratings.actor_facebook_likes,
        popularity_ratings.director_facebook_likes, popularity_ratings.movie_facebook_likes,
        popularity_ratings.imdb_rating, popularity_ratings.no_of_nominations,
        popularity_ratings.metacritic_rating, awards.category, awards.year, genres.name
        FROM people, movies, awards, popularity_ratings, genres, movies_genres
        WHERE people.id = awards.person_id
        AND awards.movie_id = movies.id
        AND movies.id = popularity_ratings.movie_id
        AND movies.id = movies_genres.movie_id
    """
```

```

AND genres.id = movies_genres.genre_id

"""
cursor.execute(sql)
results = cursor.fetchall()

csv_column_order = list(results[0].keys())

with open('table1.csv', 'w', newline='') as csvfile:
# Note that here we ask for a DictWriter, which works with the Dicts
# provided by the DictCursor.
    myCsvWriter = csv.DictWriter(csvfile, delimiter=',',
                                quotechar='"',
                                fieldnames = csv_column_order)

    # write the header row (it gets those from the fieldnames)
    myCsvWriter.writeheader()

    # and then each of the other results, row by row.
    for row in results:

        myCsvWriter.writerow(row)

cursor.execute(sql2)
results = cursor.fetchall()

csv_column_order = list(results[0].keys())

with open('table2.csv', 'w', newline='') as csvfile:
# Note that here we ask for a DictWriter, which works with the Dicts
# provided by the DictCursor.
    myCsvWriter = csv.DictWriter(csvfile, delimiter=',',
                                quotechar='"',
                                fieldnames = csv_column_order)

    # write the header row (it gets those from the fieldnames)
    myCsvWriter.writeheader()

    # and then each of the other results, row by row.
    for row in results:

        myCsvWriter.writerow(row)

```