Milestone 1: Getting to know your data

Group 14: David Modjeska, Dominic Murphy, Uday Tadiparty, Maja Campara

Introduction:

In Milestone 1, we investigated IMDb, TMDb, and MovieLens data. Our general approach was to explore the data, structure, and access methods for the three datasets. In addition, we formulated ideas on how to integrate our data, perform classic machine learning, and perform deep learning. Moreover, we did exploratory data analysis to gain insight on upcoming challenges and tradeoffs expected with more final data acquisition.

In our submission, the files in the code.zip folder reflect the code created to access the datasource API's (online or downloaded) and acquire data for us to use throughout the milestone. A particular convenience was a lookup table provided by MovieLens, which matched unique identifiers for our three data sources.

The remainder of this report will cover API code and simple explorations; challenges and questions; and exploratory data analysis (EDA).

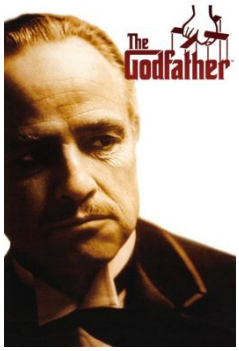
TMDb API Exploration

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

For complete code, please refer to document *2*: *Explore_tmdb_dkm.pdf*

```
#---------- A Favorite Movie: The Godfather ------------
#
# # find by external ID - "The Godfather"
# path <- paste0("3/find/tt0068646?api_key=", api_key,
#                "&language=en-US&external_source=imdb_id")
# this_content <- make_get_request(url, path)
# original_title <- this_content$movie_results$original_title
# genre_ids <- this_content$movie_results$genre_ids
# poster_path <- this_content$movie_results$poster_path
#
# # list this movie's genres on tmdb
# # note: this movie's genres on IMDB are: Crime, Drama
# this_movie_genres <- genres %>%
#     filter(id %in% genre_ids[[1]]) %>%
#     select(name)
# print(original_title)
# print(this_movie_genres)
#
# # plot poster
# poster_url <-paste0(base_url, poster_size, poster_path)
# poster_filename <- "poster.jpg"
# download.file(poster_url, poster_filename)
# poster <- load.image(poster_filename)
# plot(poster)
```

#



Poster of a favorite movie

Genre for this movie listed by TMDb and IMDb:

For full code please reference document *explore_tmdb_dkm.pdf*

IMDb: http://www.imdb.com/title/tt0068646/ Genre: Crime, Drama

TMDB: https://www.themoviedb.org/movie/238-the-godfather Genre: Drama, Crime

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

For full code please reference document *explore_tmdb_dkm.pdf*

Top 10 movies of 2016

MOVIE	GENRE
Sing	Animation, Comedy, Drama, Family, Music
Fantastic Beasts and Where to Find Them	Action, Adventure, Fantasy
Split	Horror, Thriller
Finding Dory	Adventure, Animation, Comedy, Family
Deadpool	Action, Adventure, Comedy, Romance
Rogue One: A Star Wars Story	Action, Drama, Science Fiction, War
Doctor Strange	Action, Adventure, Fantasy, Science Fiction
Arrival	Drama, Science Fiction
Captain America: Civil War	Action, Science Fiction
John Wick	Action, Thriller

Challenges expected for predicting movie genre

The challenges below have been identified regarding various aspects of the final project.

Handling movies with more than one genre

Each film will need a workable set of 1-3 genre identifiers, chosen from a canonical set across data sources. Perhaps these identifiers will form an unordered set per movie. Correlation matrices may be useful for identifying co-occurring genres, which we could use to reduce the genre multiplicity per movie. The heatmap below provide some evidence that this reduction should be possible.

Integrating 2-3 datasets, including TMDb, IMDb, and MovieLens

Each of the three datasets has a different set of unique identifiers for movies. We will need to map these identifiers across the three datasets to integrate them. We will need to rely on a combination of industry-standard IDs and more nuanced multi-variable matching. MovieLens and TMDb both rely on IMDb IDs. However, while IMDB permits bulk file download it excludes this key field. Milestone 2 will provide the team with an opportunity to solve this problem.

Leveraging high-dimensional features while remaining computationally practical

Some available movie data naturally features high dimensionality, for example, movie posters and descriptions. Full-text data will require some encoding, classically in a document-term matrix. We might then use PCA or TF-IDF to identify the most valuable features of this encoding. For graphical data, PCA might also be useful for dimensionality reduction, in order to make computation more tractable.

Selecting features from among rich datasets

Given the richness of the multiple datasets, we will need to keep an eye on potential multicollinearity (to reduce the risk of prediction bias). In addition, while integrating, the team will need to seek 'tidy' or other solutions for representing many-to-one relationships (such as actors per movie).

Choosing the right method for classifying genres

Based on preliminary online research, some potentially useful methods are identified below. (Sources are referenced in the Questions section below.)

- One vs All approach with SVM
- Multi-label KNN
- Parametric Mixture Model
- Neural Network, aka Deep Learning

Combining prediction results from text and image data

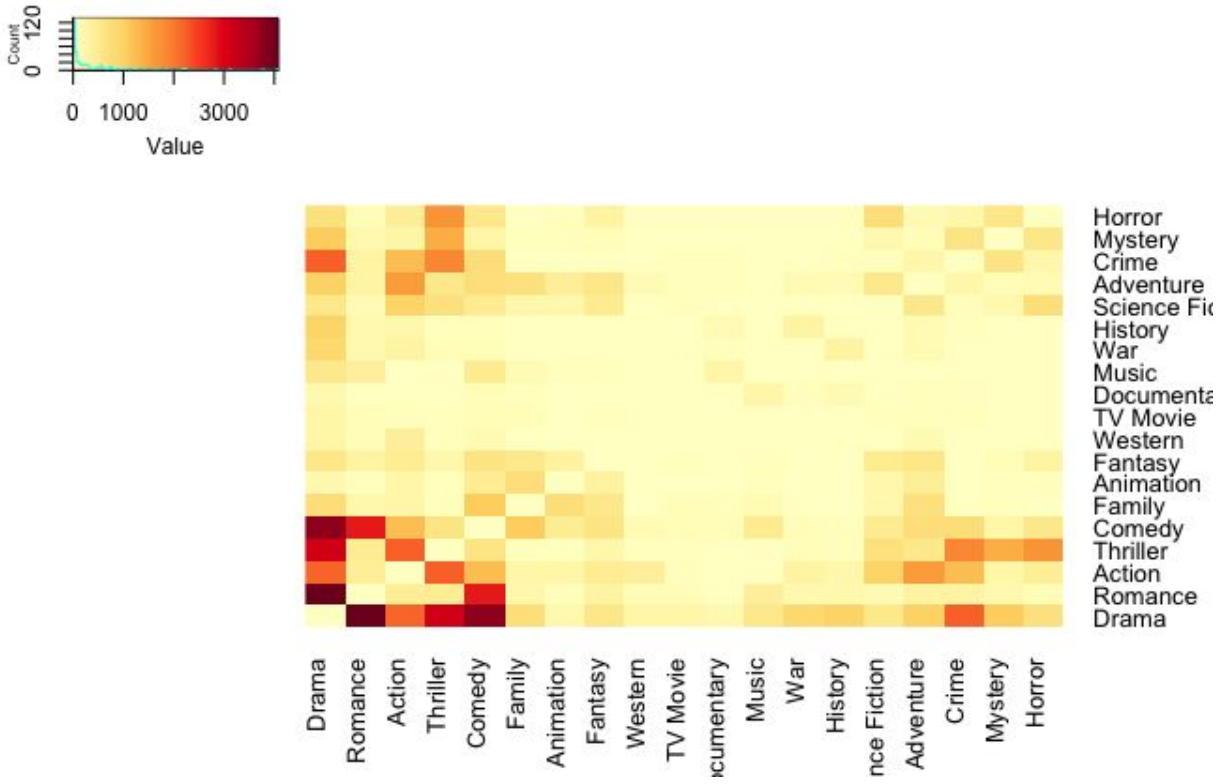
Different data features may require different classification techniques. If so, the challenging of merging prediction results emerges. One classic solution to this is ensemble methods, such as stacking. These methods have proven highly successful under the right conditions.

Determining the optimal mix of hardware and software for each project stage

Following course guidelines should help the team to scale up from the smallest early stages to the more demanding later stages, with some trial and error involved.

Movie genre pairs (TMDb) - code and visualization:

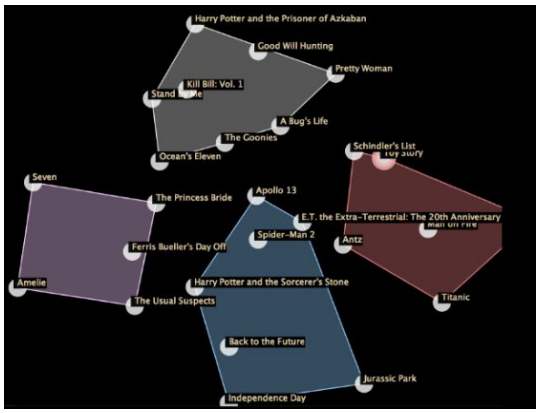
For full code please reference document *explore_tmdb_dkm.pdf*

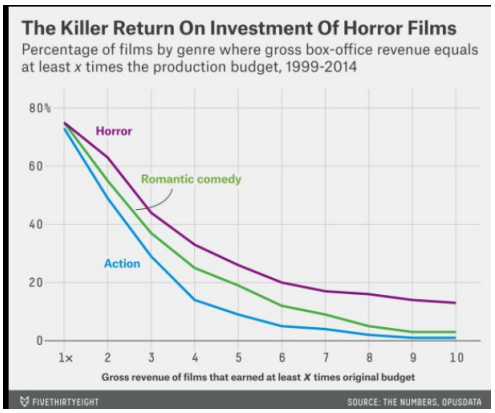


The heatmap above indicates certain popular genre combinations, such as Drama/Romance and Drama/Comedy. There are surprisingly few popular combinations, which should make the team's job of streamlining genre labeling somewhat easier. This heatmap suggests that we may be able to develop a canonical set of mixed single and compound genres to adequately describe all movies in the data sets. TMDb and MovieLens each feature 18 (matching) individual genres, so we could speculate the final number of mixed single+double genres might number approximately 36.

Concept-level EDA

To explore conceptual possibilities for visualization, the team reviewed online galleries and research. The findings have given us some insight into movie and classification generally speaking. Popular approaches to visualization in this area include clustering, series, and small multiples.





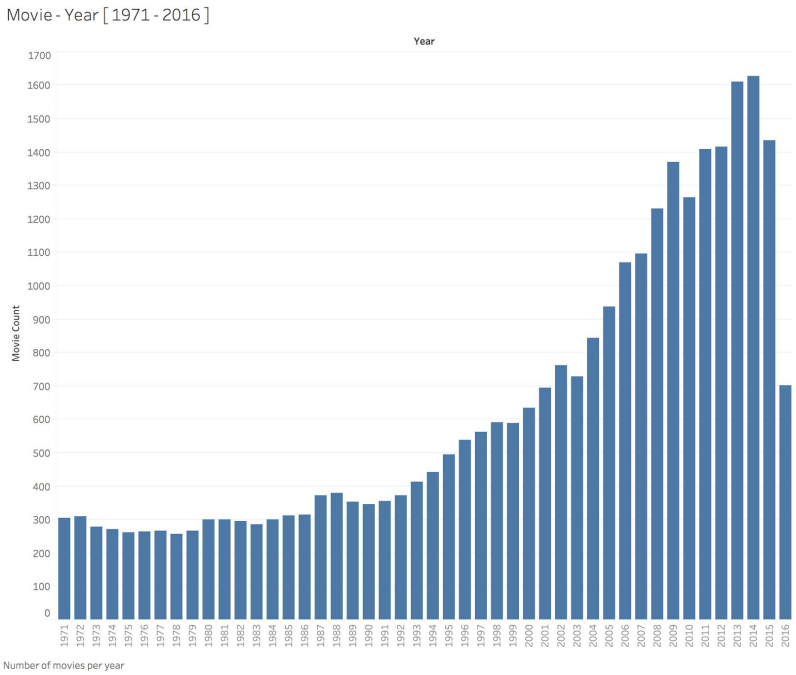
http://vis.ninja/vis/graphlix/ https://fivethirtyeight.com/datalab/scary-movies-are-the-best-investment-in-hollywood/

Implemented EDA

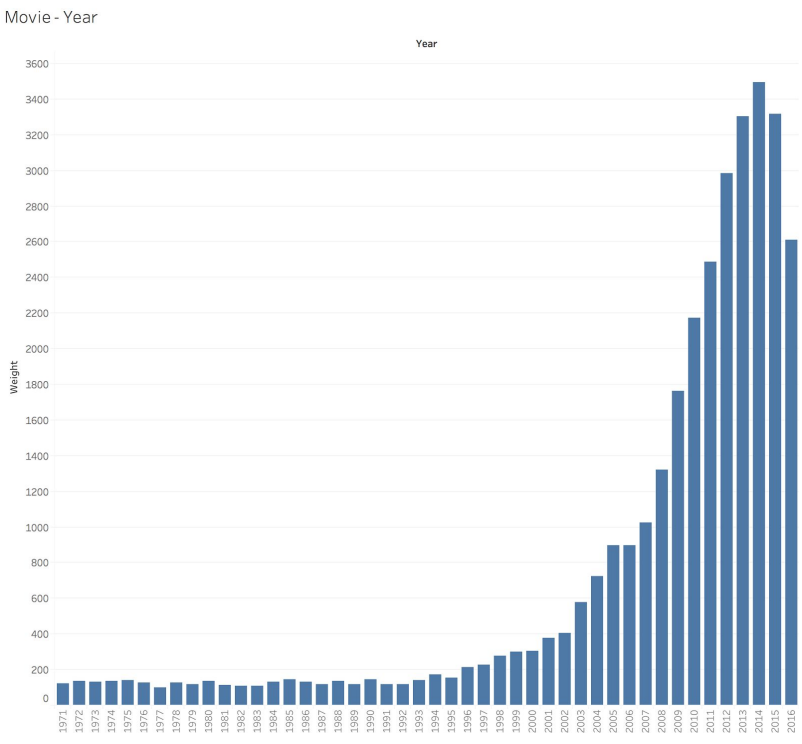
Note: most of these EDA graphs were created in Tableau, so no source code is available.

For the sake of exploratory data analysis, we compared IMDb and TMDb where initially available data was common across both platforms.

Movie Count – Year (TMDb)

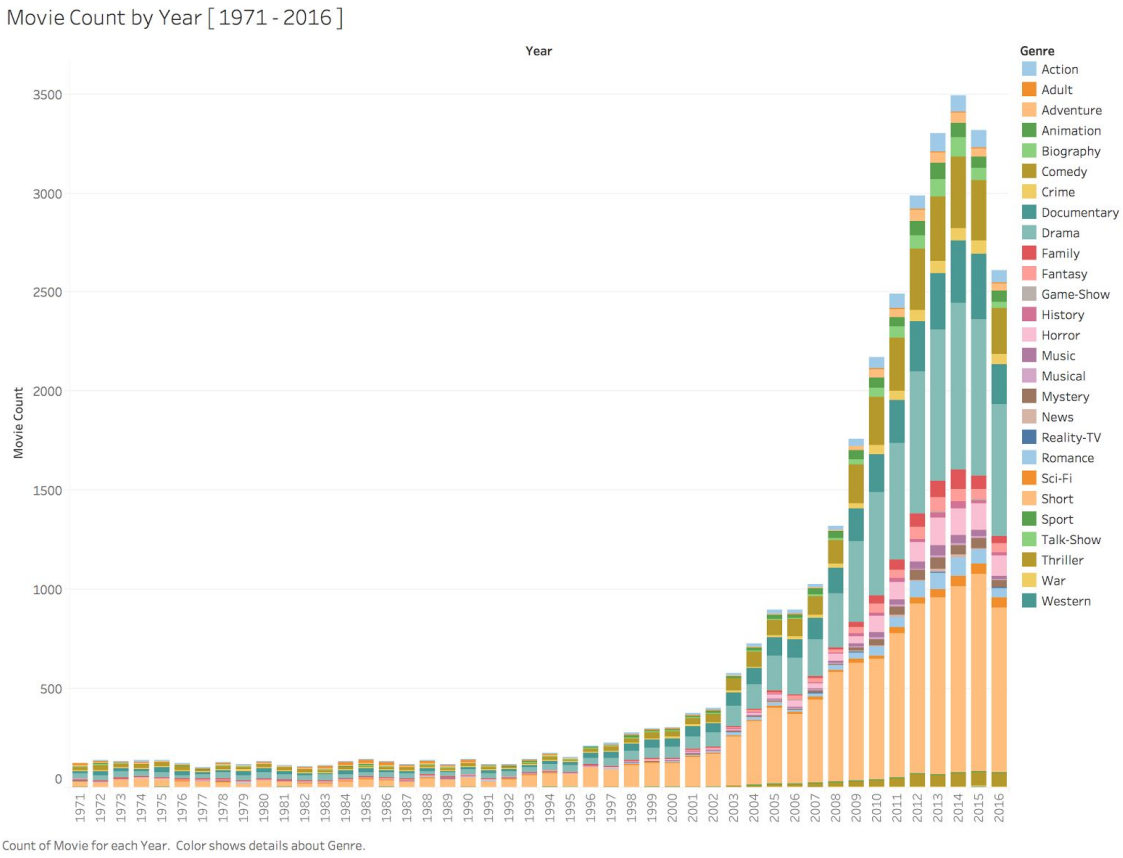


Movie Count - Year (IMDb)



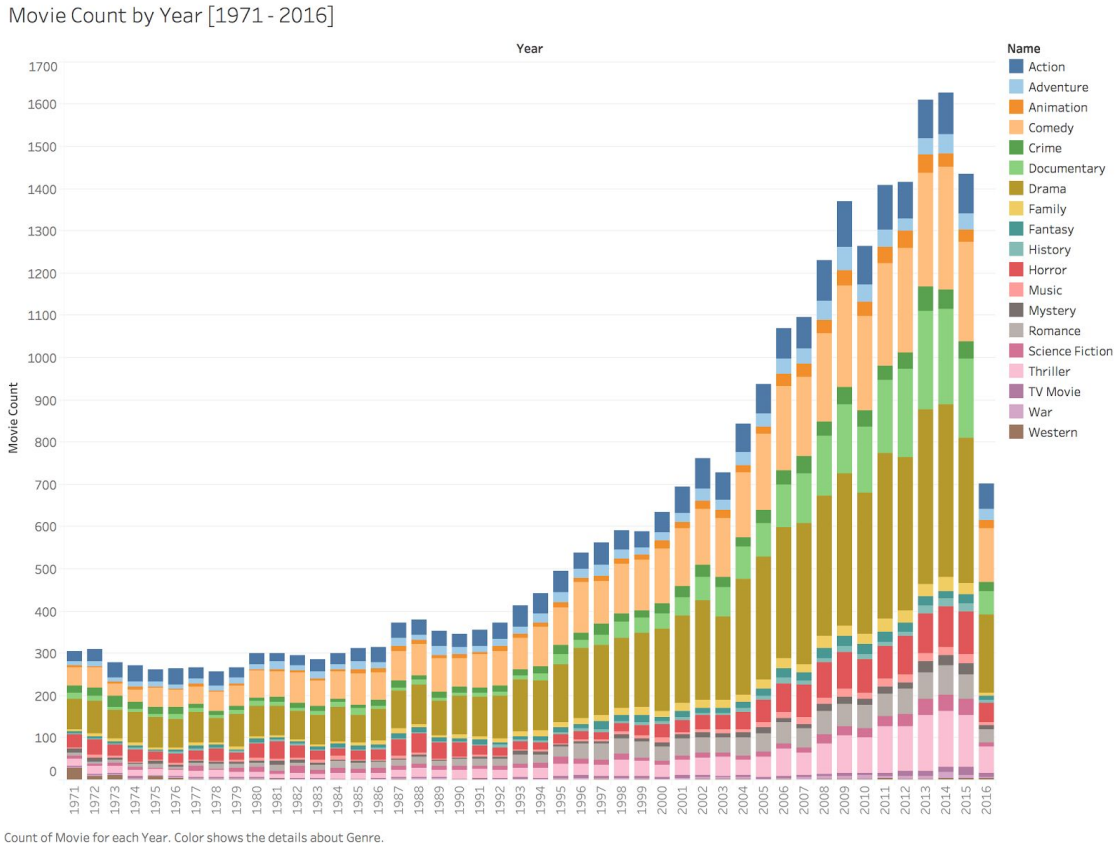
From 40,000 movie samples in TMdb and IMdb both plots show a general increasing trend of movie counts by year. The growth appears exponential from about 1990 through 2014, after which it drops a bit. We can expect the databases to be richly populated from these years. The decline in movie volume from 2013 to 2016 is interesting. However it seems unrelated to genres. The graph below shows the shrinkage to be spread reasonably evenly across genres.

Genre by Year (IMDb)



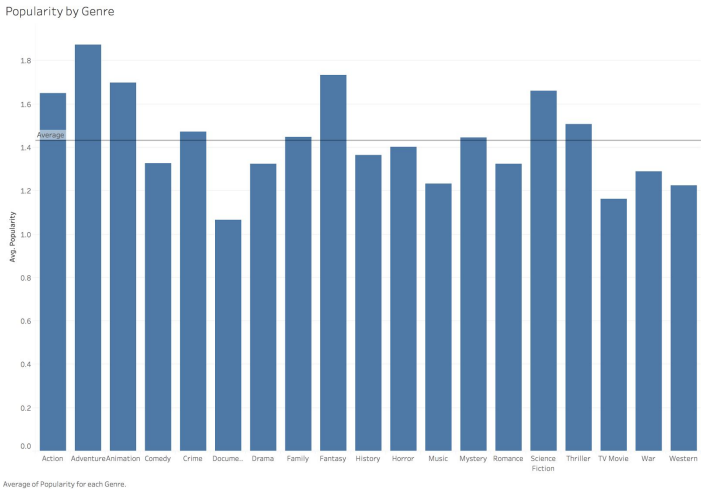
The stacked bar graphs above and below show the evolution of genres over the history of the movies. It's a fun visualization to review, in much the same way as the well-known Baby Names Explorer. Certain contemporary genres have grown tremendously in recent years - the Action genre is a good example. Datasource integration should provide a richer perspective on these trends.

Genre by Year (TMDb)



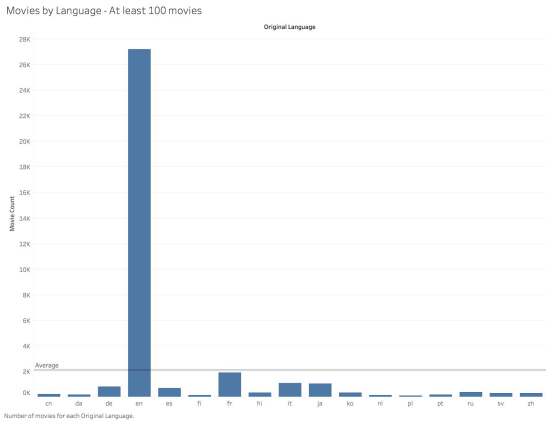
For the year 2016 the TMdb data appears to be missing data. The decline from 2015 is far steeper than for the IMdb dataset. Perhaps there is a lag in populating of the TMdb. This will have an obvious impact on matching the data. This limitation in the data will be taken into consideration when interpreting analysis of 2016 data.

Popularity by Genre (TMDb)



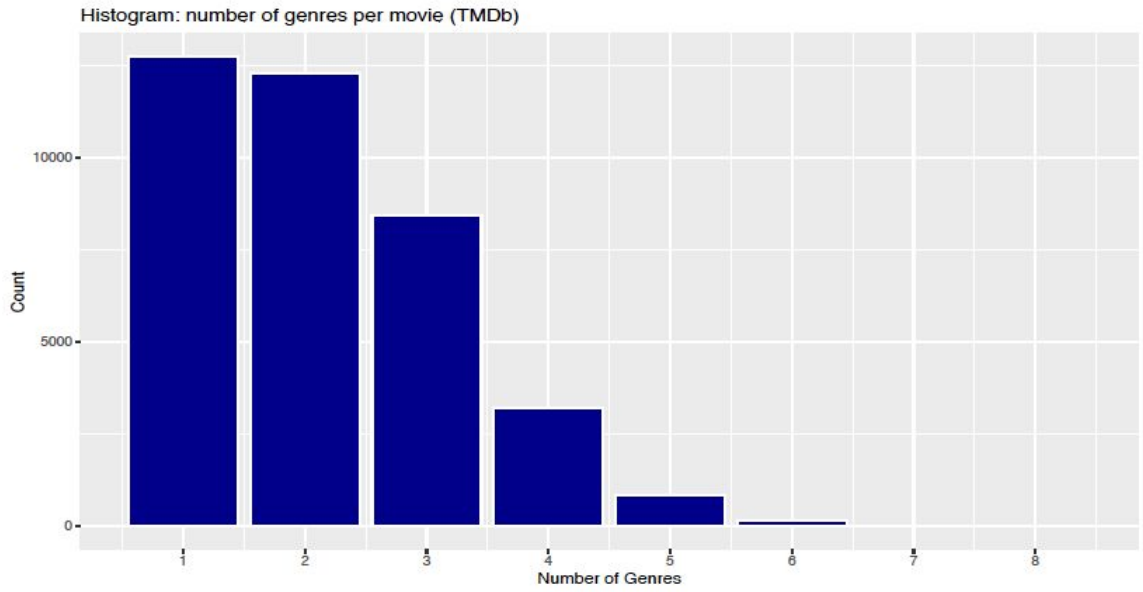
The Popularity variable from TMDb is intended to be a trending score, largely based on the date of the movie release. In the plot above we see genres like action that have an above average popularity score than a genre such as documentary.

Movies by Language – At least 100 Movies (TMDb)



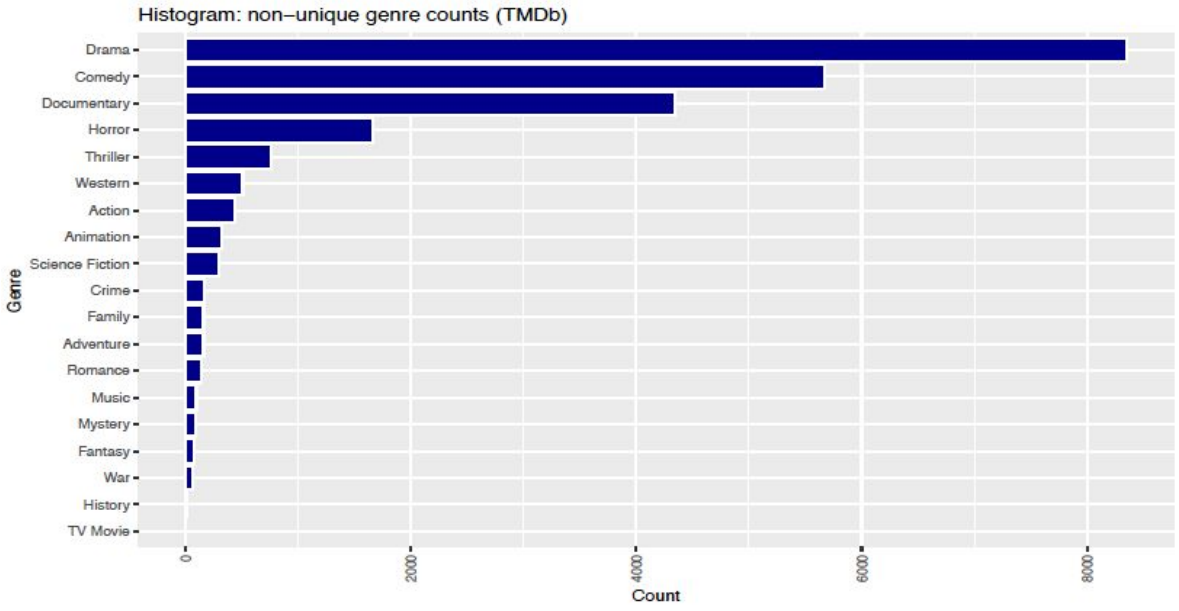
The movies by language graph shows an obvious trend that our data contains predominantly English language movies.

Number of genres per movie (TMDb)



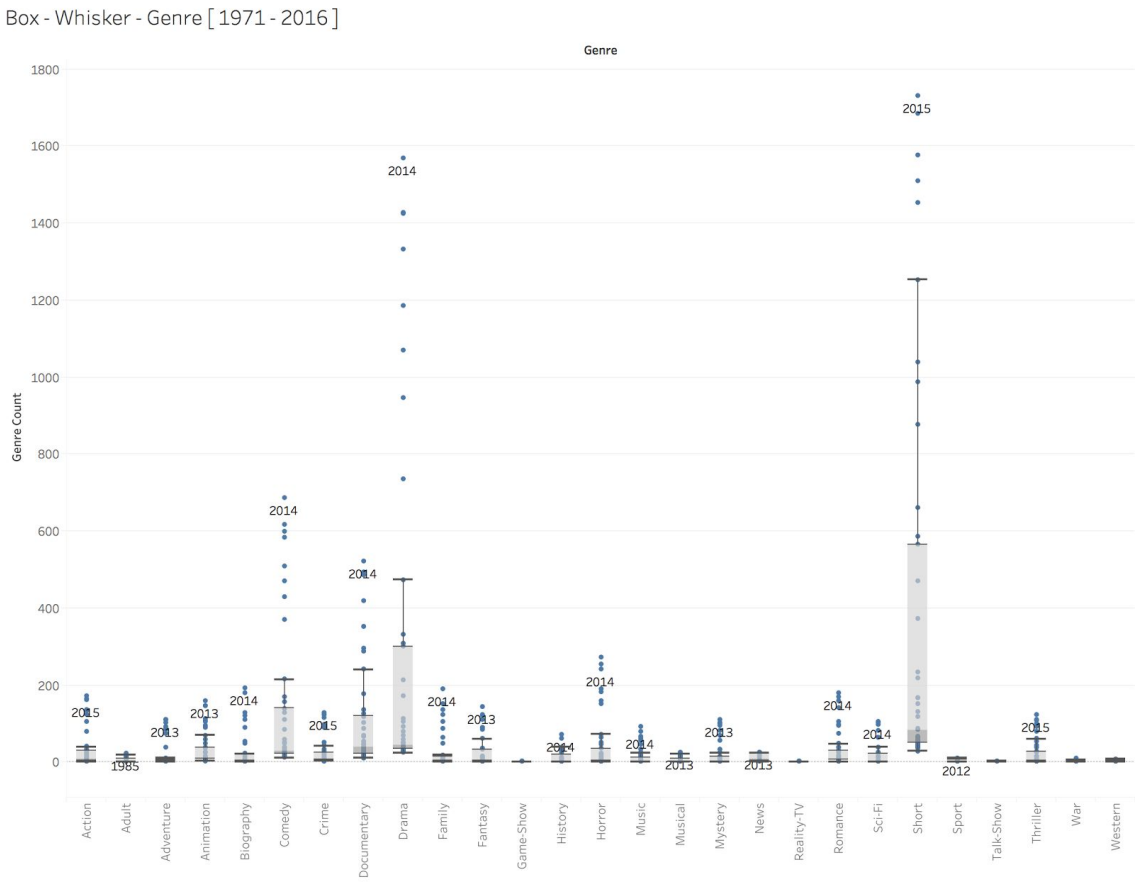
The histogram above shows the relative frequency of genres among the TMDb movies. The right skew of the plot supports the notion above that a restricted set of 1-2 genres may be sufficient for labeling the movies in our dataset.

Non-unique genre counts (TMDb)



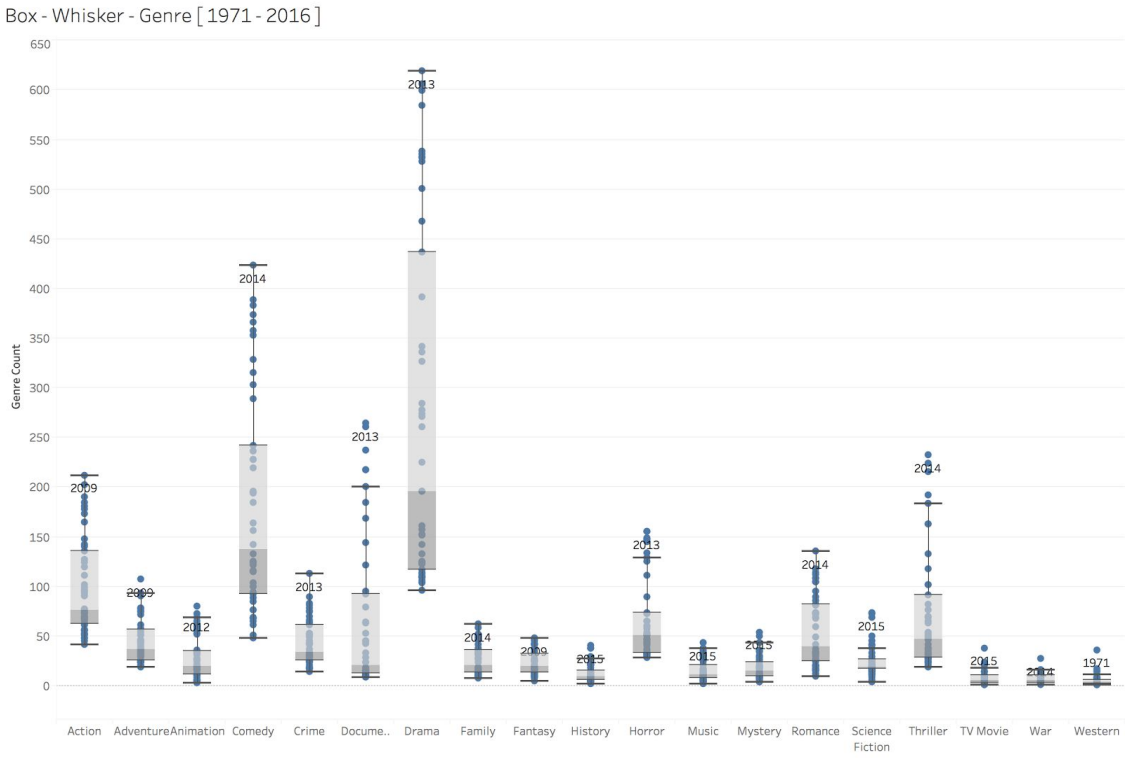
The genre counts above range from more than 8000 for Drama to less than 500 for War. When creating learning models, it may be useful to balance class weights for effective classification.

Variation in Genre Popularity over Time (IMDb)



Count of Weight for each Genre. Details are shown for Year. The view is filtered on Genre and Year. The Genre filter excludes Null. The Year filter keeps 46 of 128 members.

Variation in Genre Popularity over Time (TMDb)



These more conceptual box-and-whisker plots explore the variation in genre popularity over time. With IMDB, the most popular genres - Drama and Short(s) - rise above the rest. With TMDb, the most popular genres - Action, Comedy, and Drama - hover above the rest.

Questions to be Answered:

The questions below have been identified as interesting and necessary for Milestone 1 and the duration of the project.

How well can movie genre be predicted from posters alone? Are some genres (e.g., sci-fi, western) more typical in their style than other genres?

The look and feel of movie posters may be similar on the basis of genre. Anecdotally speaking, horror movie posters tend to be dark, while romance posters are often lighter, colorful, and focused on human characters. If these stylistic differences are prominent enough, classic and deep learning classification methods should be able to detect them.

What tradeoff between grayscale and color processing will be effective for prediction with posters?

Dimensionality reduction will likely be key in effective classification on the basis of posters. One option is to convert color images to grayscale, but this conversion risks losing chromatic style differences. Another option is PCA, as discussed above, which may be more effective for trading off between richness and computability.

To what extent do ratings/reviews and sales correlate with genre? Are user ratings and critic reviews different in this regard?

Although ratings/reviews and sales are in principle neutral across genres, certain genres tend to do better at the box office and/or with audiences, anecdotally speaking. Classic machine learning techniques should help the team to get a handle on the relationships here.

What number of genres is most typical for movie identification?

The EDA above shows that the most frequent number of genres to label a movie is 1-2, followed by 3 at a lower level. As discussed in the challenges section, reducing the genre labeling to two per movie should strike an effective balance between computability and expressiveness.

Has the genre-prediction problem been tackled before?

Preliminary research by our team has uncovered a few methods that are relevant to this problem. These methods are itemized above in the Challenges section. Here are some relevant articles referenced to date. We expect to identify other research papers throughout the project to gather ideas.

> *"Judging a Book by Its Cover"*
> https://arxiv.org/pdf/1610.09204.pdf
>
> *"Movie Genre Classification by Synopsis"*
> http://cs229.stanford.edu/proj2011/Ho-MoviesGenresClassificationBySynopsis.pdf
>
> *"Fast Film Genre Classification Combining Poster and Synopsis"*
> http://link.springer.com/chapter/10.1007/978-3-319-23989-7_8

Are certain actors and directors linked with certain genres? (Yes.) If so, how might we want to tabulate and visualize these results?

Anecdotally, this is a rich area to explore. Certain people are presumably more closely associated with given genres, while other people may have broader coverage. We might expect that such close relationships will be clear signals for a minority of movies, while other movies will need to rely on other signals for classification. Once the data sources are merged for Milestone 2, we should be able to pursue EDA and initial modeling.

How should the data acquisition be organized for domain coverage - all movies? Recent movies? Statistical sample? English only?

For Milestone 1, the full research set of data for ~40,000 movies downloaded from MovieLens. This set included films in multiple languages. For later milestones, we will need to ensure good text mining for the English-language titles of movies, whatever their original language. (This may require filtering to English-language originals only.) Final data size may depend on the dataset-matching challenge described above.

Caveat

The team chose to explore a range of three data sources for Milestone 1: MovieLens, IMDb, and TMDb. Because of this breadth, we have deferred certain integration challenges until Milestone 2, as discussed above.

Conclusion:

Our initial exploration for Milestone 1 has raised the team's awareness of the complexity and volume of data sources in the movie domain. We will need to assure high data quality, especially during Milestone 2, as we finalize genre categorization, training/testing set sizes, candidate features, data acquisition plans, and so on. The baseline sources seem to be IMDb and TMDb, which we might augment with MovieLens if its data volume is acceptably large. As we prepare for Milestone 3, we will need to keep an eye on the risk of multicollinearity among richly overlapping features.

At the same time, we are excited to be working in this data domain. The quantity and quality of publicly available data is immense, as well as the innate interest to laypeople and professionals alike. We look forward to exploring the baseline question - how does one predict movie genre? - with a variety of methods from classic and deep machine learning.

Appendix

Note: some graphs were produced in Tableau.

1) R Markdown

2) Ipython Notebook

Project Milestone 1 - TMDB download and EDA

Team 14, Harvard CS109B, Spring 2017

April 2017

```
#-------- General Setup ---------
#
# options(stringsAsFactors = FALSE)
#
# url <- "https://api.themoviedb.org"
# api_key <- "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" # for user David Modjeska
#
# # function to make a GET request and convert the raw content to JSON
# make_get_request <- function(url, path) {
#     raw_result <- GET(url = url, path = path)
#     if (raw_result$headers[["x-ratelimit-remaining"]] == '0')
#         Sys.sleep(10)
#
#     if (raw_result$status_code != 200)
#         stop(paste0("Creating request token failed with error code ",
#                     raw_result$status_code))
#
#     this_raw_content <- rawToChar(raw_result$content)
#     this_content <- fromJSON(this_raw_content)
#
#     return(this_content)
# }
#
# #----------- API Session -------------
#
# # create request token
# path <- paste0("3/authentication/token/new?api_key=", api_key)
# this_content <- make_get_request(url, path)
# request_token <- this_content$request_token
#
# # validate request token interactively
# validate_url <- paste0("https://www.themoviedb.org/authenticate/",
#                        request_token)
# browseURL(validate_url)
#
# # create session
# path <- paste0("3/authentication/session/new?api_key=", api_key,
#                "&request_token=", request_token)
# this_content <- make_get_request(url, path)
# session_id <- this_content$session_id
#
# # get genre list
# path <- paste0("3/genre/movie/list?api_key=", api_key, "&language=en-US")
# this_content <- make_get_request(url, path)
# genres <- this_content$genres
#
# genres_filename <- "tmdb_genre_list.csv"
```

```
# out_file <- str_c(project_dir, genres_filename)
# write_csv(genres, out_file)
#
# # get configuration
# path <- paste0("3/configuration?api_key=", api_key)
# this_content <- make_get_request(url, path)
# base_url <- this_content$images$base_url
# poster_size <- 'w500'
#
# #---------- A Favorite Movie: The Godfather ------------
#
# # find by external ID - "The Godfather"
# path <- paste0("3/find/tt0068646?api_key=", api_key,
#                "&language=en-US&external_source=imdb_id")
# this_content <- make_get_request(url, path)
# original_title <- this_content$movie_results$original_title
# genre_ids <- this_content$movie_results$genre_ids
# poster_path <- this_content$movie_results$poster_path
#
# # list this movie's genres on tmdb
# # note: this movie's genres on IMDB are: Crime, Drama
# this_movie_genres <- genres %>%
#     filter(id %in% genre_ids[[1]]) %>%
#     select(name)
# print(original_title)
# print(this_movie_genres)
#
# # plot poster
# poster_url <-paste0(base_url, poster_size, poster_path)
# poster_filename <- "poster.jpg"
# download.file(poster_url, poster_filename)
# poster <- load.image(poster_filename)
# plot(poster)
#
# #---------- Top movies in 2016 ------------
#
# # get data
# num_movies <- 10
# path <- paste0("3/discover/movie?api_key=", api_key,
#                "&language=en-US",
#                "&sort_by=popularity.desc",
#                "&include_adult=false",
#                "&include_video=false",
#                "&page=1",
#                "&year=2016")
# this_content <- make_get_request(url, path)
# original_title_10 <- this_content$results$original_title[1:num_movies]
# genre_ids_10 <- this_content$results$genre_ids[1:num_movies]
#
# # print data
# print_data <- data.frame(MOVIE <- character(), GENRE <- character())
# for (i in 1:num_movies) {
#     this_title <- original_title_10[i]
```

```
#     these_ids <- genre_ids_10[i]
#     these_genres <- genres %>%
#         filter(id %in% these_ids[[1]]) %>%
#         select(name) %>%
#         summarize(GENRE = paste(name, collapse = ", "))
#     this_data <- data.frame(MOVIE = this_title, GENRE = these_genres)
#     print_data <- rbind(print_data, this_data)
# }
#
# out_file <- str_c(project_dir, "top_10_movies_2016.csv")
# write_csv(print_data, out_file)
#
# #------- DOWNLOAD -----------------
#
# # set directories
# project_dir <- "/Users/davidmodjeska/CS109b/Project/"
# ml_data_dir <- "MovieLens/ml-latest/"
# tmdb_data_dir <- "TMDb/"
#
# # read ML data to get IMDb IDs
# in_file <- str_c(project_dir, "ml_data.csv")
# ml_data <- read_csv(in_file, col_names = TRUE, col_types = 'iccinii') %>%
#     mutate(num_genres = as.integer(num_genres))
#
# # extract movie results from one downloaded data chunk
# make_movie_rec <- function(this_content) {
#     this_movie_record <- this_content$movie_results
#     this_movie_record$genre_ids = sub("^c[(](.*)[)]$", "\\1",
#                                       this_movie_record$genre_ids)
#
#     return(this_movie_record)
# }
#
# # download a sample record
# imdb_id <- ml_data$imdbId[1]
# path <- paste0("3/find/tt0", imdb_id, "?api_key=", api_key,
#                "&language=en-US&external_source=imdb_id")
# this_content <- make_get_request(url, path)
# first_movie_record <- make_movie_rec(this_content)
#
# # get multiple records (note that make_get_request() will sleep as needed)
# all_movie_records <- first_movie_record
# for (i in 2:nrow(ml_data)) {
#     imdb_id <- str_pad(ml_data$imdbId[i], 7, pad = '0')
#     path <- paste0("3/find/tt", imdb_id, "?api_key=", api_key,
#                    "&language=en-US&external_source=imdb_id")
#     this_content <- make_get_request(url, path)
#     if (is.data.frame(this_content$movie_results)) {
#         this_movie_record <- make_movie_rec(this_content)
#         all_movie_records <- rbind(all_movie_records, this_movie_record)
#     }
# }
#
```

```
# # count number of genres per movie
# all_movie_records <- mutate(all_movie_records,
#                             num_genres = str_count(genre_ids, '[,]') + 1)
#
# # save file
# out_file <- str_c(project_dir, data_filename)
# write_csv(all_movie_records, out_file)
```

Top 10 movies of 2016

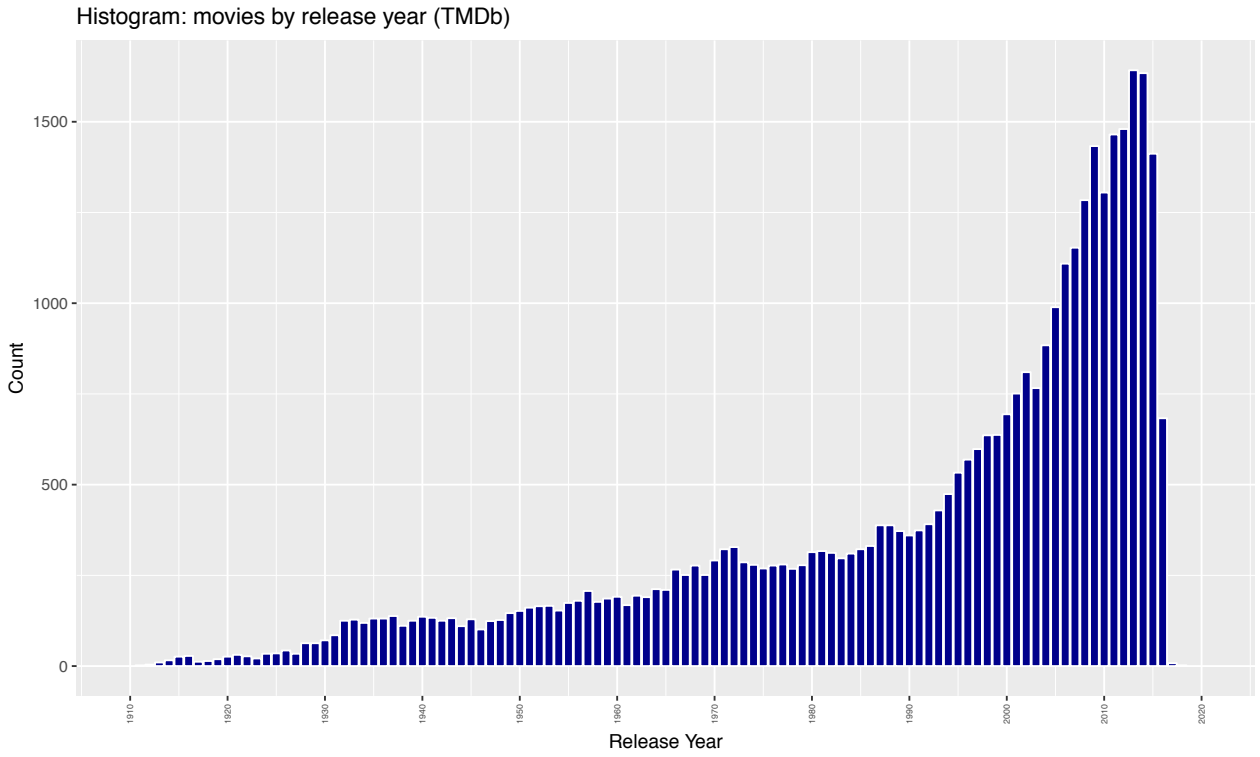
```
in_file <- str_c(project_dir, top_movies_filename)
print_data <- read_csv(in_file, col_types = 'cc')
knitr::kable(print_data)
```

MOVIE	GENRE
Sing	Animation, Comedy, Drama, Family, Music
Fantastic Beasts and Where to Find Them	Action, Adventure, Fantasy
Split	Horror, Thriller
Finding Dory	Adventure, Animation, Comedy, Family
Deadpool	Action, Adventure, Comedy, Romance
Rogue One: A Star Wars Story	Action, Drama, Science Fiction, War
Doctor Strange	Action, Adventure, Fantasy, Science Fiction
Arrival	Drama, Science Fiction
Captain America: Civil War	Action, Science Fiction
John Wick	Action, Thriller

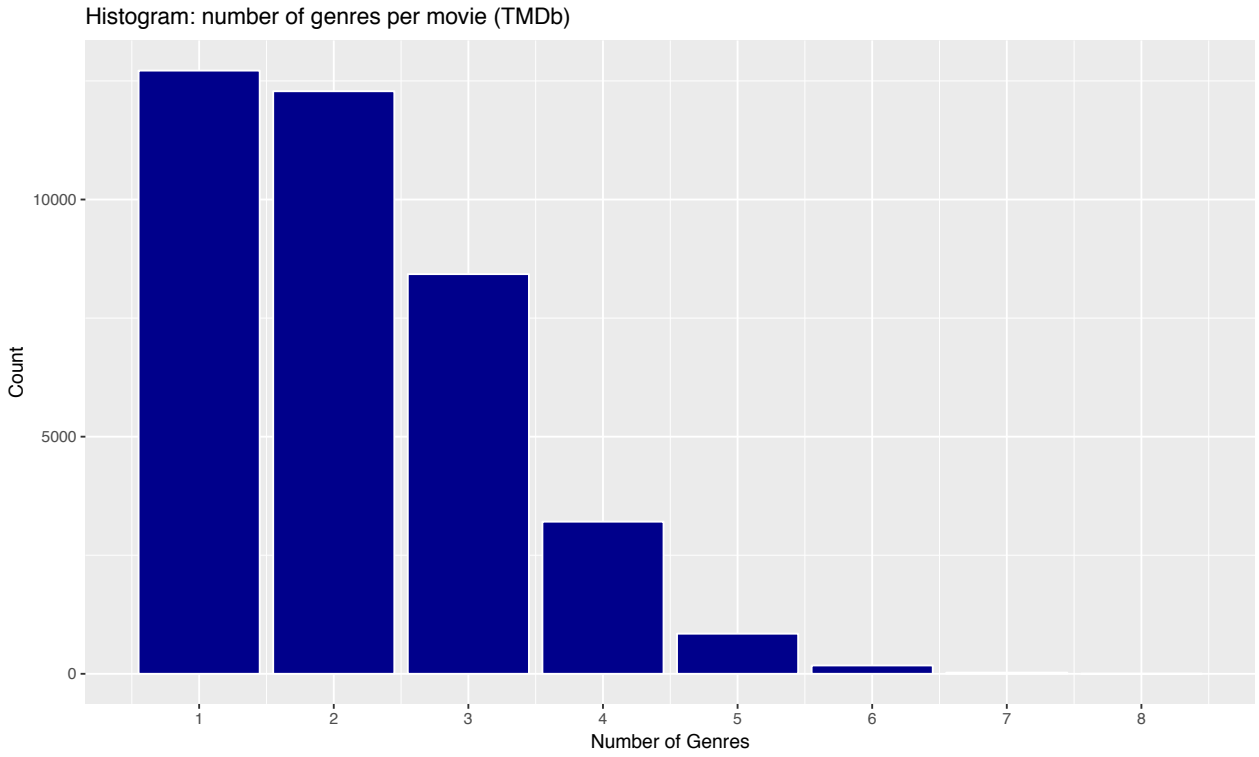
EDA

```
in_file <- str_c(project_dir, genres_filename)
genres <- read_csv(in_file)

in_file <- str_c(project_dir, data_filename)
all_movie_records <- read_csv(in_file, col_types = 'lcciccccccnclnin')

plot_data <- all_movie_records %>%
    mutate(release_date = ymd(release_date))
```

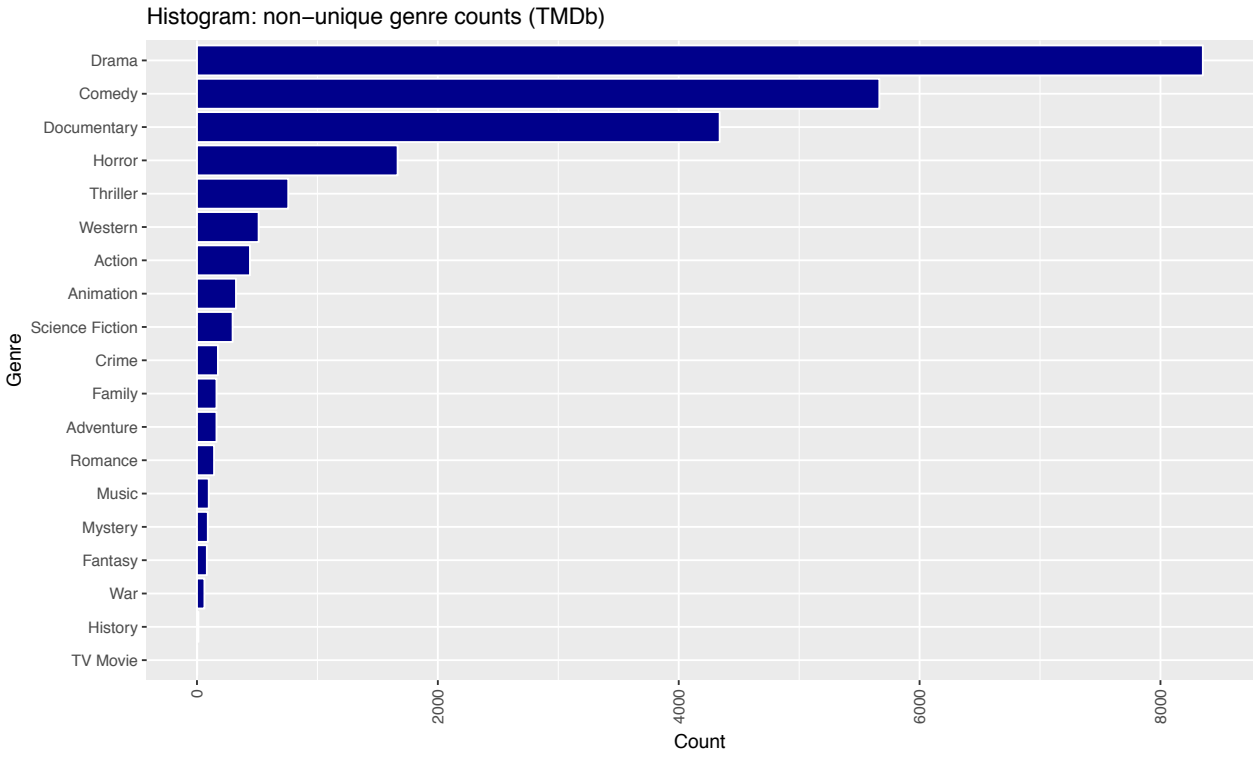
```
# release year
ggplot(all_movie_records, aes(x = year(release_date))) +
    geom_histogram(stat = 'count', color = 'white', fill = 'darkblue') +
    scale_x_continuous(name = "Release Year", breaks = seq(1910, 2020, by = 10),
                       limits = c(1910, 2020)) +
    ylab("Count") +
    ggtitle("Histogram: movies by release year (TMDb)") +
    theme(axis.text.x=element_text(angle = 90, hjust = 1, vjust = 0.5, size = 5))
```




```
# number of genres per movie
max_num_genres <- max(plot_data$num_genres)
ggplot(plot_data, aes(x = num_genres)) +
    geom_histogram(stat = 'count', color = 'white', fill = 'darkblue') +
    scale_x_continuous(name = "Number of Genres", breaks = 1:max_num_genres) +
    ylab("Count") +
    ggtitle("Histogram: number of genres per movie (TMDb)")
```

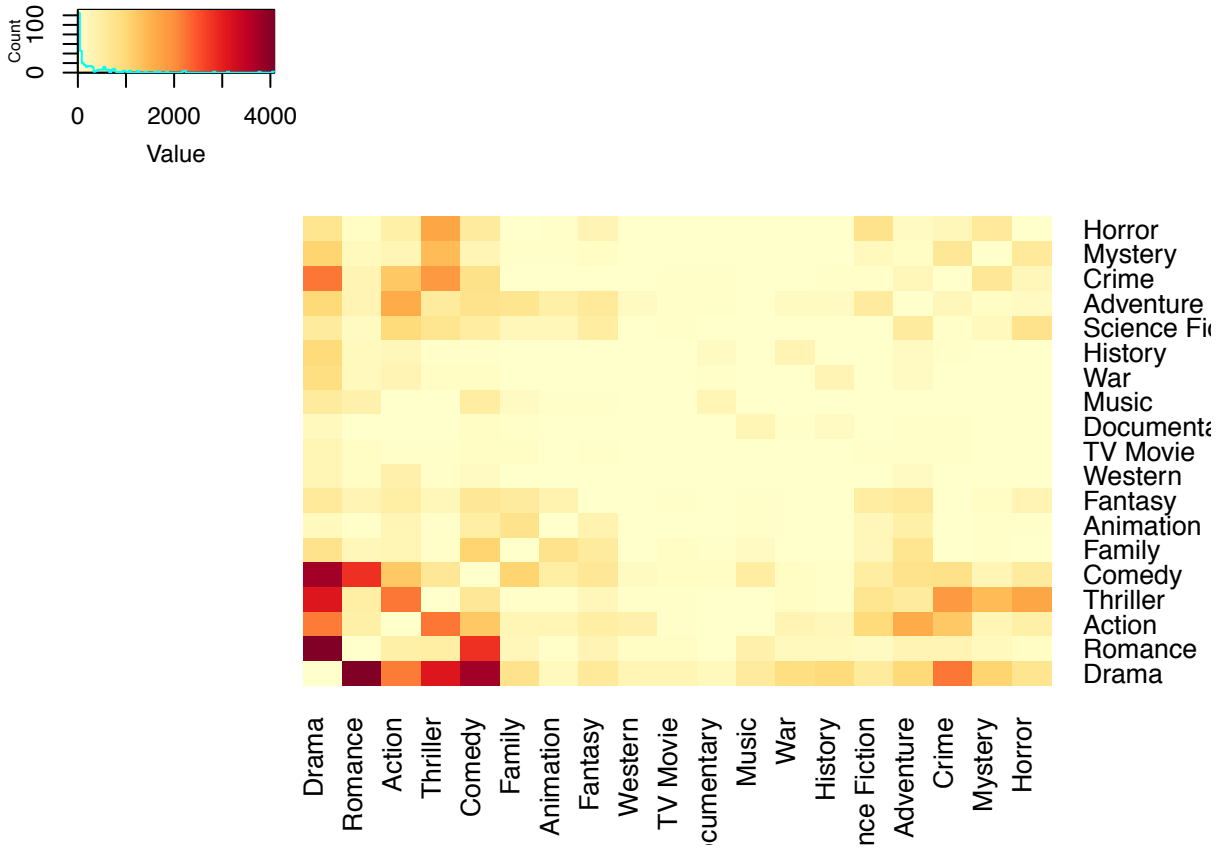



```
# prepare non-unique genre counts # FIX - make tidy
unique_genre_ids <- c(
    str_c('^', genres$id, '$'),
    str_c(', ', genres$id)
)
genre_counts <- sapply(unique_genre_ids,
                       function(x) { sum(str_count(x, plot_data$genre_ids)) })
genre_counts_df <- data.frame(id = names(genre_counts),
                              count = genre_counts) %>%
    mutate(id = gsub("\\^| |,|\\$", "", id)) %>%
    mutate(id = as.integer(id)) %>%
    group_by(id) %>%
    dplyr::summarize(count = sum(count)) %>%
    inner_join(genres, by = "id")

# genre counts
ggplot(genre_counts_df, aes(x = reorder(name, count), y = count)) +
    geom_histogram(stat = 'identity', color = 'white', fill = 'darkblue') +
    xlab("Genre") +
    ylab("Count") +
    ggtitle("Histogram: non-unique genre counts (TMDb)") +
    theme(axis.text.x=element_text(angle = 90, hjust = 1, vjust = 0.5)) +
    coord_flip()
```


Histogram: non−unique genre counts (TMDb)
Drama
Comedy
Documentary
Horror
Thriller
Western
Action
Animation
Science Fiction
Crime
Family
Adventure
Romance
Music
Mystery
Fantasy
War
History
TV Movie
Genre
0
2000
4000
6000
8000
Count

Heatmap of Genre Pairs

```
# create dictionary of genres
num_movies <- nrow(all_movie_records)
num_genres <- nrow(genres)
genre_dict <- 1:num_genres
names(genre_dict) <- genres$id
foreign_genre = '10769'

# create zero-count matrix with labeled rows and columns
genre_matrix <- matrix(0, nrow = num_genres, ncol = num_genres)
rownames(genre_matrix) <- genres$id
colnames(genre_matrix) <- genres$id

# count genre pairs (skipping singletons and foreign)
for (movie in 1:nrow(all_movie_records)) {
    these_genres <- str_split(all_movie_records$genre_ids[movie],
                              pattern = ', ')[[1]]
    num_these_genres <- length(these_genres)
    if (num_these_genres > 1) {
        for (j in 1:(num_these_genres - 1)) {
            for (k in (j + 1):num_these_genres) {
                genre_1 <- these_genres[j]
                genre_2 <- these_genres[k]
                if (genre_1 != foreign_genre & genre_2 != foreign_genre)
                    genre_matrix[genre_1, genre_2] = genre_matrix[genre_1, genre_2] + 1
            }
        }
    }
}

# fill symmetrical matrix
genre_matrix_copy <- matrix(genre_matrix, nrow = num_genres, ncol = num_genres)
genre_matrix_copy <- genre_matrix_copy + t(genre_matrix) # fill matrix
rownames(genre_matrix_copy) <- genres$name
colnames(genre_matrix_copy) <- genres$name

# plot heatmap
oranges = brewer.pal(9, "YlOrRd")
palette = colorRampPalette(oranges)(100)
heatmap.2(genre_matrix_copy, dendrogram='none', Rowv = TRUE, Colv = TRUE,
          trace = 'none', col = palette, key.title = NA)
```


Count
0
100
0
2000
4000
Value
Horror
Mystery
Crime
Adventure
Science Fi
History
War
Music
Documenta
TV Movie
Western
Fantasy
Animation
Family
Comedy
Thriller
Action
Romance
Drama
Drama
Romance
Action
Thriller
Comedy
Family
Animation
Fantasy
Western
TV Movie
cumentary
Music
War
History
nce Fiction
Adventure
Crime
Mystery
Horror

Query_IMDB_SQlite

April 4, 2017

0.1 Data Extraction and Manipulation from IMDB text files

Links

https://docs.python.org/2/library/sqlite3.html
http://www.sqlite.org/quickstart.html
http://imdbpy.sourceforge.net

```
In [1]: import pandas as pd
        import sqlite3 #pip install sqlite3
        import subprocess
        #import urllib
        import omdb
        from xml.etree import ElementTree as ET
        import datetime
```

Create & connect to local SQlite db

```
In [2]: #connect to db
        #note: creates empty sql db if it does not exist.
        con = sqlite3.connect('imdbpy/imdb.db')
        c = con.cursor()
```

Populate database from imdb text file downloads (shell script)

```
In [ ]: #warning will take a long tome to run (P.S I actualy ran this from the term
        #result = subprocess.run(
        #    ['imdbpy2sql.py -d ~/Downloads -u sqlite:imdb.db --sqlite-transactions
        #    , stdout=subprocess.PIPE)
        #result.stdout
```

DB Admin Tasks

```
In [3]: #create index on movie type (kind_type) ie movie, tv ect
        c.execute("CREATE INDEX IF NOT EXISTS title_idx_kind on title (kind_id)")

        #rebuilds the database file, repacking it into a minimal amount of disk spa
        c.execute("VACUUM")
```

```
Out[3]: <sqlite3.Cursor at 0x113d280a0>
```

```
In [4]: #list the tables
        str_sql = "SELECT name as tables FROM sqlite_master WHERE type='table';"
        tables = pd.read_sql_query(str_sql, con)
        tables
```

Out[4]:

	tables
0	name
1	sqlite_sequence
2	char_name
3	company_name
4	kind_type
5	title
6	company_type
7	aka_name
8	aka_title
9	role_type
10	cast_info
11	comp_cast_type
12	complete_cast
13	info_type
14	link_type
15	keyword
16	movie_keyword
17	movie_link
18	movie_info
19	movie_info_idx
20	movie_companies
21	person_info
22	movie_genre

Roll up multiple genres per movie into a piped string

```
In [29]: #Drop table first incase the script is run a second time
         c.execute('DROP TABLE IF EXISTS movie_genre;')

         #create table
         # movie_id    : Primary Key, unique identifer for movie
         # genre       : Pipe seperated list of genres for the movie ie Romantic |
         # genre_count : The number of genres ie 2 for Romantic | Comedy

         #genres (info_type_id = 3)
         #movie  (kind_id = 1)
         str_sql = 'CREATE TABLE movie_genre as                          ' + \
                   'SELECT                                               ' + \
                   '   i.movie_id                    as movie_id         ' + \
                   ' , group_concat(i.info," | ")  as genre            ' + \
                   ' , count(i.info)                 as genre_count    ' + \
                   'FROM title t                                         ' + \
```

```
                '  INNER JOIN movie_info i on (t.id = i.movie_id) ' + \
                'WHERE t.kind_id = 1                              ' + \
                '  AND i.info_type_id = 3                         ' + \
                'GROUP BY i.movie_id                              '
        c.execute(str_sql)

        #create index on movie type (kind_type) ie movie, tv ect
        c.execute("CREATE UNIQUE INDEX IF NOT EXISTS pk_movie_genre_movie_id on mo

        #show example table data
        query_result = pd.read_sql_query("select * from movie_genre LIMIT 5", con)
        query_result
```

Out[29]:

	movie_id	genre	genre_count
0	2891047	Comedy | Short	2
1	2891048	Comedy | Short	2
2	2891050	Drama | Short	2
3	2891051	Animation | Short	2
4	2891052	Comedy | Short	2

- Extract a list of movie names
- The IMDB text files do not contain the unique imdb_id code for the movies.
- The db comes form the files. So . . . the db does not have the imdb_id

We will need to extract the imdb by querying using omdb API. But first we need a list of movie titles and production years

```
In [43]: #Extract id, title & year from the local db
         #id is the unique id for the movie in the local db. it is not the imdb_id
         #note kind_id = 1 = movies

         str_sql = 'SELECT id                                ' + \
                   ', trim(substr(title,2)) as title         ' + \
                   ', ltrim(production_year,4) as year       ' + \
                   'FROM title                               ' + \
                   'WHERE kind_id = 1                        ' + \
                   ' AND ifnull(production_year,0) <> 0      ' + \
                   ' AND title <> "#"                        ' + \
                   ' AND imdb_id IS NULL;                    '

         movie_list = pd.read_sql_query(str_sql, con)
         movie_list.head(6)
```

Out[43]:

	id	title	year
0	2891058	1 Dad	2016
1	2891102	Alleman	2013
2	2891103	allthatissolidmeltsintoair	2016
3	2891104	am/pm	2018
4	2891111	armoire	2016

Query omdb to gain imdb_id code

```
In [44]: success = 0
         failure = 0
         for index, row in movie_list.iterrows():
             id    = row['id']
             year  = row['year']
             title = row['title'].strip()

             #query omdb
             try:
                 res = omdb.request(t=title, y=year, r='xml', type='movie')
                 xml_content = res.content

                 #Check to see if the movie was matched
                 root = ET.fromstring(xml_content)
                 if(root.get('response') == 'True'):

                     #extract the imdb_id from the top match
                     imdbID = root.findall('movie')[0].get('imdbID')

                     #write imdb_id back to database
                     str_sql = "UPDATE title SET imdb_id = '" + imdbID + "' WHERE i
                     c.execute(str_sql)
                     success = success + 1
                 else:
                     failure = failure + 1

             except:
                 failure = failure + 1

             #every thousand records: print status update
             if index % 1000 == 0:
                 #commit changes to the DB
                 con.commit()

                 #Print status update
                 print index, success, failure, datetime.datetime.now().time()
             con.commit()
```

```
0 0 1 17:20:28.614747
```

```
In [35]: #Outputs SQL query to csv / utf-8 encoding
         def sql_to_file(str_sql, file_name):
             df = pd.read_sql_query(str_sql, con)
             df.to_csv(file_name, encoding='utf-8')
             return df
```

Export map between local db keys and imdb_id (include pipe seperated genre data)

```
In [36]: str_sql = 'SELECT id                                                    ' + \
                   ', t.imdb_id                                                  ' + \
                   ', trim(substr(t.title,2)) as title                           ' + \
                   ', ltrim(t.production_year,4) as year                         ' + \
                   ', g.genre                                                    ' + \
                   ', g.genre_count                                              ' + \
                  'FROM title t                                                  ' + \
                  ' LEFT OUTER JOIN movie_genre g on (t.id = g.movie_id)         ' + \
                  'WHERE t.kind_id = 1                                           ' + \
                  ' AND t.imdb_id IS NOT NULL ;                                  '

         df = sql_to_file(str_sql, 'imdb_id.csv')
         print 'exported ', len(df), ' records'
         df.head(5)

exported  44821  records
```

Out[36]:

	id	imdb_id	title	year	genre	genre_count
0	2891050	tt0491587	1	2005	Drama | Short	2.0
1	2891051	tt0408060	1	2009	Animation | Short	2.0
2	2891052	tt0926084	1	2010	Comedy | Short	2.0
3	2891055	tt4856314	1 at the Apocalypse Box Office	2015	Comedy | Short	2.0
4	2891056	tt3597346	1 Beauty Nail Salon	2014	Drama | History | Short | War	4.0

Export Genre Combination Summary

```
In [37]: str_sql = 'SELECT                                                  ' + \
                   '  ltrim(t.production_year,4) as year                   ' + \
                   ', g.genre                    as genre_pair             ' + \
                   ', avg(g.genre_count)         as pair_len               ' + \
                   ', COUNT(t.id)                as movie_count            ' + \
                  'FROM movie_genre as g                                    ' + \
                  '  INNER JOIN title t on (g.movie_id = t.id)              ' + \
                  'GROUP BY ltrim(t.production_year,4)                      ' + \
                  '        , g.genre                                        '

         df = sql_to_file(str_sql, 'GenreCombinationSummary.csv')
         print 'exported ', len(df), ' records'
         df.head(5)
```

```
exported  58291  records
```

Out[37]:

	year	genre_pair	pair_len	movie_count
0	None	Action	1.0	1523
1	None	Action | Adventure	2.0	107
2	None	Action | Adventure | Animation	3.0	5
3	None	Action | Adventure | Animation | Biography | C...	7.0	1
4	None	Action | Adventure | Animation | Comedy	4.0	3

Export genre count (not unique a movie may be counted in two genres)

```
In [42]: str_sql = 'SELECT                                          ' + \
                   '  LTRIM(t.production_year,4)   as year          ' + \
                   ', i.info                       as genre         ' + \
                   ', count(i.info)                as movie_count   ' + \
                   'FROM title t                                    ' + \
                   '  INNER JOIN movie_info i on (t.id = i.movie_id) ' + \
                   'WHERE t.kind_id = 1                             ' + \
                   '  AND i.info_type_id = 3                        ' + \
                   'GROUP BY LTRIM(t.production_year,4)             ' + \
                   '       , i.info                                 '
         df = sql_to_file(str_sql, 'GenreSummary.csv')
         print 'exported ', len(df), ' records'
         df.head(5)
```

```
exported  3088  records
```

Out[42]:

	year	genre	movie_count
0	None	Action	4103
1	None	Adult	121
2	None	Adventure	1720
3	None	Animation	660
4	None	Biography	805