Team Koala Ninja (Team 10)
Ankit Vij
Amanpreet Singh Saini
Joel Haynie

Class: CS 839 - Data Science Spring 2018
Prof: Dr. AnHai Doan
Date: 05/09/2018
RE: Project Stage 4

Entity Type: *Movies*

Table Info & Schema

We collected > 3000 tuples from each site across a variety of movie genres. We collected 3005 movies from rottenTomatoes and 3250 movies from imdb in stage 2. The Tuples are of the form:

```
ID, Title, Score, Rating, Genre, Directed By, Written By, Box Office, Release Date, Runtime, Studio
```

Stage 4 Pipline

ML Matcher from Stage #3 → Merging → Analysis -> Output

The Combination Process and Getting E

We first obtain a Matches table(the PredictedMatchedTuples.csv file). 608 matches were classified / found.
We join the table A with Matches table and then, further with B doing a *Full Outer Join* (Union of A & B). We did not add any data from any other tables or sources. We ran into issues with non-valid values given by NaN. The way we resolved/handled this is shown below.

The rules for merging the two tables are as follows:
For all the places where NaN values are encountered:

If both A and B values are NaN, we put an empty placeholder for the respective data type in E.
If either one value is NaN, put the other in E.
If both are valid values, follow the rule described below:

Column	Merging Rule
ID	Combined ID from A & B with ‘; ’, These will make the ID completely unique.
Title	Pick the non-null title out of the two, if one exists.
Score	0 if both nulls; if one of them null, return the score of another; if both valid, get their average
Directed By	Longer of the strings from A and B
Written By	Longer of the strings from A and B
Runtime	Minimum of the Runtimes from A and B.
Box Office	Maximum of the values from A and B.
Genre	If genres from A and B are not the same, add them using ‘;’ and put in E’s Genre field.
Studio	If Studio from A and B are not the same, add them using ‘;’ and put in E’s Studio field.

Rating	If A has a valid rating, use that, otherwise, use B's Rating
Release Date	Pick the valid date entry out of A and B, giving B the priority if both have valid dates.

Statistics

The size of table E is 5691 total tuples.
The schema for E is the same as that of A and B.

```
ID, Title, Score, Rating, Genre, Directed By, Written By, Box Office, Release Date, Runtime, Studio
```

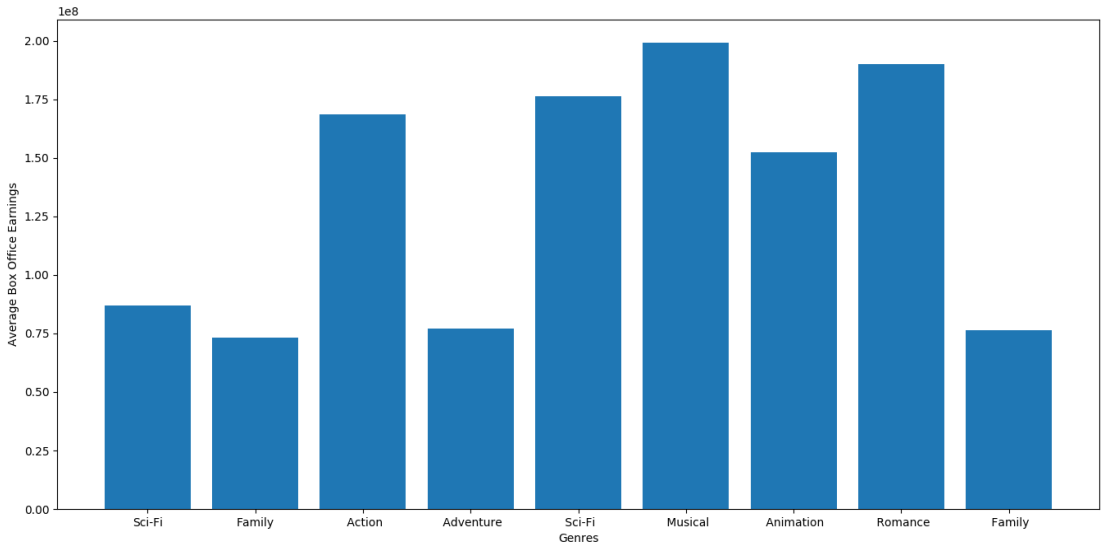
The four sample tuples from Table E are:

A_0344,The Foreigner,71.0,R,Drama,Martin Campbell,David Marconi,34393507.0,2017-10-13,113.0,STX Entertainment
A_0217; B_1398,Inside Out,90.0,PG,Comedy; Animation,Pete Docter,Meg LeFauve,356461711.0,2015-06-19,94.0,Pixar Animation Studios; Disney/Pixar
A_0377,Rogue One,78.0,PG-13,Sci-Fi,Gareth Edwards,Chris Weitz,532177324.0,2016-12-16,133.0,Lucasfilm
B_2117,The Hundred-Foot Journey,69.0,PG,Drama,Lasse Hallstrom,,46214579.0,2014-08-08,122.0,Walt Disney Pictures

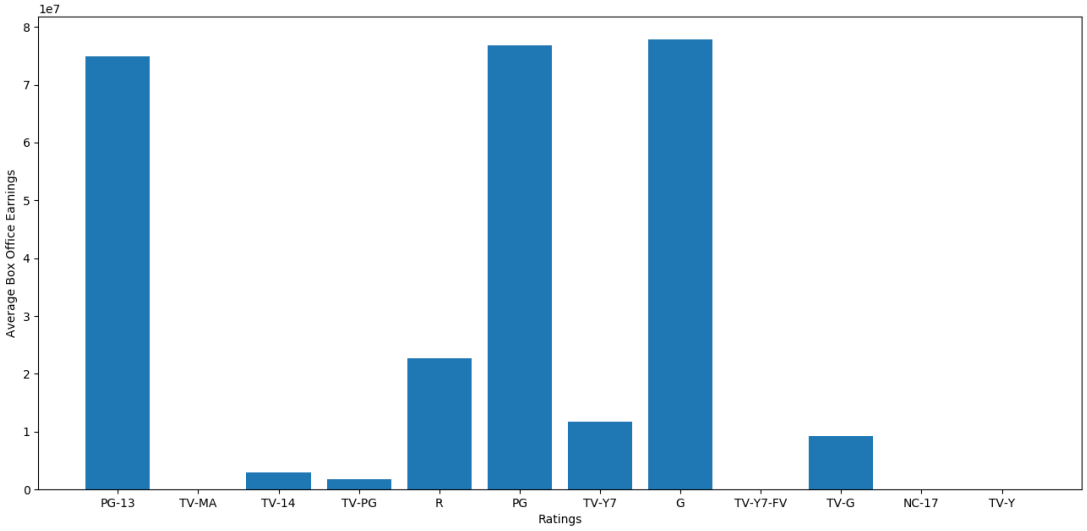
Analysis (OLAP)

We did an OLAP style data exploration where we used operations like roll up, drill down, and slice. We analyzed four scenarios using this style of data exploration.

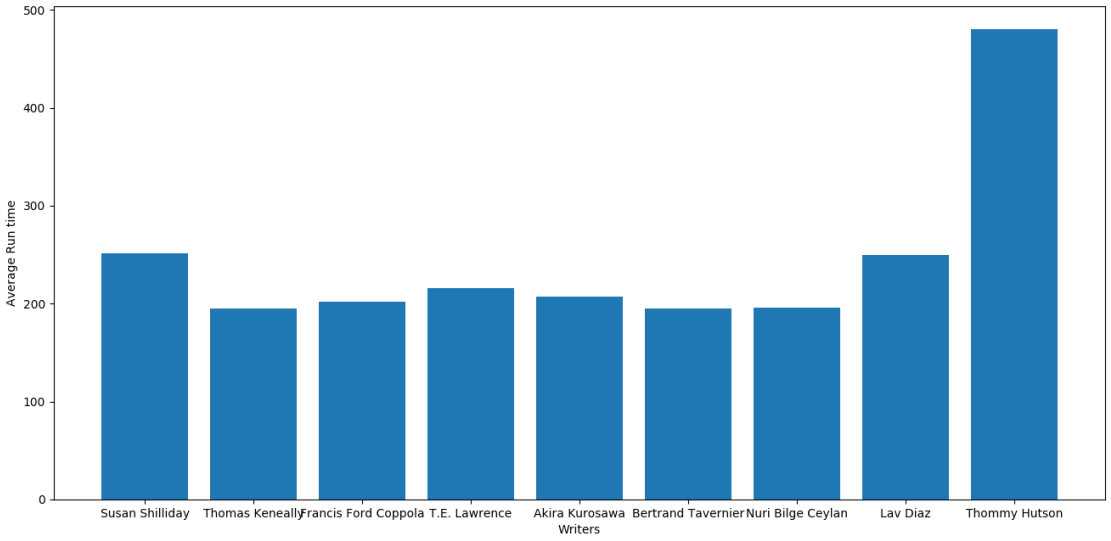
1. **Which genres are popular in terms of average box office earnings?**
   For this analysis we drilled down on individual genres that movies are associated with and found out the average box office earnings for each genre. Then in order to get a better picture of the popular genres we removed all those genres that had an average box office earnings less than 70 million. We found that movies that had a musical genre had the highest average earnings, followed by romance and sci-fi.



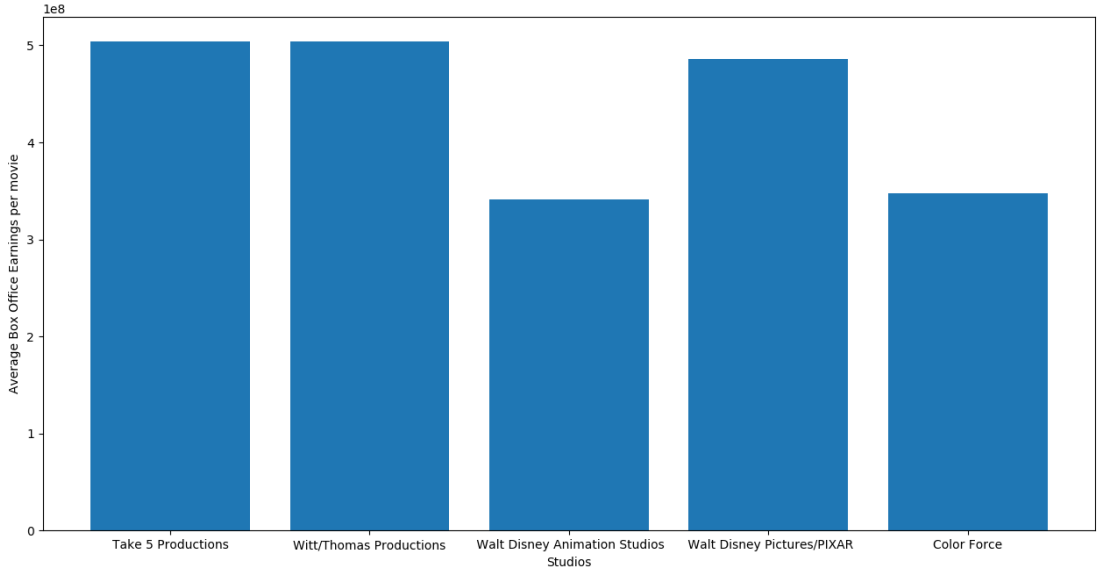
2. **Which movie ratings are viewed the most?**
   For this analysis question, we chose to find the average box office earnings per movie rating. We assumed that if a movie has a high average earning then it is because that more people watched that movie, and thus is viewed more. We averaged out the box office earning per movie rating. We found out that G rated movies are viewed the most followed by PG and PG-13 rated movies.



3. **Which writers write long scripts i.e movies that have 3 hours plus runtime?**
   For this analysis, we wanted to narrow down to the writers that write long scripts leading to longer runtime. We drilled down on the writers and per writer we found the average runtime per movie. In order narrow our search for writers with runtimes greater than 3 hours we filtered out the writers who had a runtime of less than 190 minutes. We found out that Thommy Hutson writes the longest scripts with a per movie runtime of greater than 450 minutes. He as followed by Lav Diaz and Susan Shilliday, both of which have movie runtimes close to 250 minutes.



4. **Which studios makes the most profits per movie?**
   This analysis provided us with an interesting insight about studios and which studios made the most profit per movie. We first found all the possible studios for the various movies and found out the earnings per movie for each of the studios. To get a small subset of the most profitable studios we filtered out the studios that had a per movie earning less than 330 million. We found out that Take 5 Productions and Witt/Thomas Productions earned the most per movie (~500 million) followed by Walt Disney Pictures/PIXAR.



Problems Faced

- The Genre and Studio column had multiple values separated by a semi-colon for which we had to do some preprocessing during the analysis phase
- There were NaN entries in the string column values which needed to be handled differently as compared to the normal values while doing analysis on them.

Future Work

- We would include more data into our analysis 5691 movies is a lot, but there are more in the DBs.
- We would include more data from other movie sites.
- We would include more data from the raw tables, We thinned out the Genres, Directors, and Studios to make the previous stages easier.
- We would Compare the difference between the ratings for each site. Answer: How does RottenTomatoes rating compare to IMDBs?
- We would use some clustering techniques during our analysis phase to answer: Is there Clustering in the x / y plot for "Score vs Box Office" or "Rating vs Box Office"?

Conclusions & What we learned

There are many choices on what “merge two table” means. (Union, intersection, left only, right only, relative complement left, relative complement right). There are cases where each one makes sense for your data analysis goals.

Tools Used:

- Development & Execution Environment:
  - PyCharm v2018.1.2 (community edition)
- Libraries / Packages:
  - python v3.6
  - py_entitymatching v0.3.0
  - scipy v1.1.0
  - ipython v6.3.1
  - matplotlib v2.2.2
  - pandas v0.22.0

These libraries have many sub tool dependencies, but these are the major ones we used, these other dependencies were left out for brevity.

Merger Code Appendix:

```
import sys
import py_entitymatching as em
import pandas as pd
import math

# Display the versions
print('python version: ' + sys.version)
print('pandas version: ' + pd.__version__)
print('magellan version: ' + em.__version__)

# Get the paths
path_A = './../data/A.csv'
path_B = './../data/B.csv'
path_Matched = './../data/PredictedMatchedTuples.csv'

# Load csv files as dataframes and set the key attribute in the dataframe
A = em.read_csv_metadata(path_A, key='ID')
B = em.read_csv_metadata(path_B, key='ID')
M = em.read_csv_metadata(path_Matched)

print('Number of tuples in A: ' + str(len(A)))
print('Number of tuples in B: ' + str(len(B)))
print('Number of tuples in Matches table: ' + str(len(M)))

print(list(A))
print(list(B))
print(list(M))

# join A with M
A_M = pd.merge(A, M, how='outer', left_on='ID', right_on='ltable_ID')
print('num A_M: ' + str(len(A_M)))

# join further with B
merged_df = pd.merge(A_M, B, how='outer', left_on='rtable_ID', right_on='ID')
print('num merged_df: ' + str(len(merged_df)))
print(list(merged_df))


# test merging the other direction.
# M_B = pd.merge(M, B, how='outer', left_on='rtable_ID', right_on='ID')
# print('num M_B: ' + str(len(M_B)))
# merged_df2 = pd.merge(A, M_B, how='outer', left_on='ID', right_on='ltable_ID')
# print('num merged_df2: ' + str(len(merged_df2)))

# Numbers come to (Check!)
#   num A_M: 3259
#   num merged_df: 5691
#   num M_B: 3040
#   num merged_df2: 5691


def pick_title(x, y):
    if pd.isnull(x):
```

```
        if pd.isnull(y):
            return ''
        else:
            return y
    else:
        return x


# data merging step
def average_cols(x, y):
    if math.isnan(x):
        if math.isnan(y):
            return 0
        else:
            return y
    else:
        if math.isnan(y):
            return x
        else:
            return math.floor((int(x) + int(y)) / 2)


def min_check(x, y):
    if math.isnan(x):
        if math.isnan(y):
            return 0
        else:
            return y
    else:
        if math.isnan(y):
            return x
        else:
            return min(x, y)


def max_check(x, y):
    if math.isnan(x):
        if math.isnan(y):
            return 0
        else:
            return y
    else:
        if math.isnan(y):
            return x
        else:
            return max(x, y)


def max_length(x, y):
    if pd.isnull(x):
        if pd.isnull(y):
            return ''
        else:
            return y
    else:
        if pd.isnull(y):
            return x
        else:
            if len(str(x)) > len(str(y)):
                return x
            else:
                return y


def add_if_different(x, y):
    if pd.isnull(x):
        if pd.isnull(y):
            return ''
        else:
            return str(y)
    else:
        if pd.isnull(y):
            return str(x)
        else:
            if str(x) != str(y):
                return str(x) + '; ' + str(y)
            else:
                return str(x)
```

```
def retain_if_not_na(x, y):
    if pd.isnull(x):
        if pd.isnull(y):
            return 'NOT RATED'
        else:
            return str(y)
    else:
        if pd.isnull(y):
            return str(x)
        else:
            if str(x) != 'NOT RATED':
                return str(x)
            elif str(y) != 'NOT RATED':
                return str(y)
            else:
                return str(x)  # return NOT RATED


def pick_release_date(x, y):
    if pd.isnull(x):
        if pd.isnull(y):
            return ''
        else:
            return y
    else:
        return x


# generating the new columns from X(A) and Y(B).
merged_df['Title'] = merged_df.apply(lambda x: pick_title(x['Title_x'], x['Title_y']), axis=1)
merged_df['Score'] = merged_df.apply(lambda x: average_cols(x['Score_x'], x['Score_y']), axis=1)
merged_df['Directed By'] = merged_df.apply(lambda x: max_length(x['Directed By_x'], x['Directed By_y']),
axis=1)
merged_df['Written By'] = merged_df.apply(lambda x: max_length(x['Written By_x'], x['Written By_y']), axis=1)
merged_df['Runtime'] = merged_df.apply(lambda x: min_check(x['Runtime_x'], x['Runtime_y']), axis=1)
merged_df['Box Office'] = merged_df.apply(lambda x: max_check(x['Box Office_x'], x['Box Office_y']), axis=1)
merged_df['Genre'] = merged_df.apply(lambda x: add_if_different(x['Genre_x'], x['Genre_y']), axis=1)
merged_df['Studio'] = merged_df.apply(lambda x: add_if_different(x['Studio_x'], x['Studio_y']), axis=1)
merged_df['Rating'] = merged_df.apply(lambda x: retain_if_not_na(x['Rating_x'], x['Rating_y']), axis=1)
merged_df['Release Date'] = merged_df.apply(lambda x: pick_title(x['Release Date_x'], x['Release Date_y']),
axis=1)
merged_df['ID_new'] = merged_df.apply(lambda x: add_if_different(x['ID_x'], x['ID']), axis=1)

# basic select operation - ID_y is the ID of the matches table
Z = merged_df[['ID_new', 'Title', 'Score', 'Rating', 'Genre', 'Directed By', 'Written By', 'Box Office',
'Release Date', 'Runtime', 'Studio']]
# rename to match the required schema
E = Z.rename(columns={'ID_new': 'ID'})
print(list(Z))
print(list(E))

print(E.head(3))

path_E = './../data/E.csv'
E.to_csv(path_E, index=False)
```