CS 839 Project-4

by Vishal Agarwal, Roshan Lal and Pratham Desai

Merging Tables:

We had two tables T1 (imdb) and T2 (allmovie) containing movie information, from www.imdb.com and www.allmovie.com respectively.

We used different merging rules for different columns in the two tables. For “Name” and “Director” column, we kept longer of the strings found in T1 and T2. For columns Year and Certificate, we preferred T1’s data over T2. For Runtime, we took average of the two. For Genre and SubGenre columns, we took a union of the genres/subgenres mentioned in the two tables. Rating and votes information was significantly different in the two tables, so we retained data from both the sources. Columns for Producer, Country and OtherRating are taken from T2. Columns for Gross and Metascore are taken from T1.

Detailed merging rules for every field is described below:

Column name	Merging rule
Key	Key of the matched pair returned by py_entitymatching
Key1	ID of the movie on www.imdb.com
Key2	ID of the movie on www.allmovie.com
Name	Longer of the strings T1.name and T2.name

Year	if T1.Year is present
	  Year = T1.year
	else
	  Year = T2.Year

Certificate	if T1.Certificate is present
	  Certificate = T1.Certificate
	else
	  Certificate = T2.Certificate

Runtime	if both T1.Runtime and T2.Runtime are present
	  Runtime = (T1.Runtime + T2.Runtime) / 2.0
	else if T1.Runtime is present
	  Runtime = T1.Runtime
	else if T2.Runtime is present

	Runtime = T2.Runtime else Runtime = ‘’
Genre	Union of genres given by T1.Genre and T2.Genre (represented using a ‘|’ separated string)
SubGenre	Union of subgenres given by T1.SubGenre and T2.SubGenre (represented using a ‘|’ separated string)
rating1	T1.rating
rating2	T2.rating
votes1	T1.votes
votes2	T2.votes
Director	Longer of the strings T1.Director and T2.Director
Producer	T2.Producer
Country	T2.Country
Gross	T1.Gross
OtherRating	T2.OtherRating
Metascore	T1.Metascore

Statistics:

In the previous section we presented a detailed description of the schema.
Our table has 1931 tuples.

Few sample tuples:

	Example 1	Example 2	Example 3	Example 4
Key	318244	308786	280335	273306
Key 1	tt0406375	tt0055608	tt0120176	tt0073629
Key 2	zathura-v310130	voyage-to-the-bottom-of-the-sea-v53041	the-spanish-prisoner-v158678	the-rocky-horror-picture-show-v41864
Name	Zathura: A Space Adventure	Voyage to the Bottom of the Sea	The Spanish Prisoner	The Rocky Horror Picture Show
Year	2005	1961	1997	1975
Certificate	PG	PG	PG	R
Runtime	101	105.5	111	100
Genre	fantasy|children's/family|comedy|action|adventure	science fiction|sci-fi|action|adventure	drama|thriller|mystery	science fiction|musical|comedy|horror
SubGenre	children's fantasy|space adventure	sea adventure|adventure drama|sci-fi disaster film	crime thriller|post-noir (modern noir)	sex comedy|horror comedy|rock musical
Rating 1	6.2	6.1	7.3	7.4
Rating 2	7	7	8	8
Votes 1	77676	4780	19697	115333
Votes 2	127	38	47	214
Director	Jon Favreau	Irwin Allen	David Mamet	Jim Sharman
Producer	Columbia TriStar	20th Century Fox	Jean Doumanian Productions; Sweetland Films	20th Century Fox
Country	USA	USA	USA	UK

Gross	29258869		10200000	139876417
OtherRating	7	6	7	8
Metascore	67		70	58

Script to merge tables:

```
result_cols =
['Key','Key1','Key2','Name','Year','Certificate','Runtime','Genre','SubGenr
e','rating1','rating2','votes1','votes2','Director','Producer','Country','G
ross','OtherRating','Metascore']

def read_csv(fname):
      with open(fname, 'r') as f:
            content=f.readlines()
            col_heads = content[0].strip().split(',')
            rows = [x.strip().split(',') for x in content[1:] if
x.strip()!='']
            return col_heads, rows

def get_preferred(val1, val2, pref):

      if pref == 1:
            if val1.strip() != '':
                  return val1.strip()

            return val2.strip()

      if pref == 2:

            if val2.strip() != '':
                  return val2.strip()

            return val1.strip()

def get_longer(val1, val2):

      if len(val1.strip()) > len(val2.strip()):
            #print("Longer of " + val1.strip() + " and " + val2.strip())
```

```
            return val1.strip()
      else:
            return val2.strip()

def get_average(val1, val2):

      val1_ = val1.strip()
      val2_ = val2.strip()

      if val1_ == '':
            return val2_

      if val2_ == '':
            return val1_

      try:
            num1 = float(val1_)
      except ValueError:
            num1=-1
            print(val1_ + ' not convertible to float')

      try:
            num2 = float(val2_)
      except ValueError:
            num2 = -1
            print(val2_ + ' not convertible to float')

      if num1 >= 0 and num2 >= 0:
            return str((num1+num2)/2.0)
      elif num1 >= 0:
            return str(num1)
      elif num2 >= 0:
            return str(num2)
      else:
            print('both values invalid')
            return ''

def get_union(val1, val2):
      list1 = [x for x in val1.lower().strip('').split('|') if x != '']
      list2 = [x for x in val2.lower().strip('').split('|') if x != '']
      union_set = set(list1) | set(list2)
      return '|'.join(union_set)
```

```
def merge_rows(pair_key, row1, col_names_1, row2, col_names_2):

      key1 = row1[0]
      key2 = row2[0]

      merged_row = []

      # pair key
      merged_row.append(pair_key)

      # key1
      merged_row.append(key1)

      # key2
      merged_row.append(key2)

      # name
      name1 = row1[col_names_1.index('Name')]
      name2 = row2[col_names_2.index('Name')]
      merged_name = get_longer(name1, name2)
      merged_row.append(merged_name)

      # year
      year1 = row1[col_names_1.index('Year')]
      year2 = row2[col_names_2.index('Year')]
      year = get_preferred(year1, year2, 1)
      merged_row.append(year)

      # certificate
      certificate1 = row1[col_names_1.index('Certificate')]
      certificate2 = row2[col_names_2.index('Certificate')]
      certificate = get_preferred(certificate1, certificate2, 1)
      merged_row.append(certificate)

      # runtime
      runtime1 = row1[col_names_1.index('Runtime')]
      runtime2 = row2[col_names_2.index('Runtime')]
      runtime = get_average(runtime1, runtime2)
      merged_row.append(runtime)

      # Genre     - union of two sets
```

```
        genre1 = row1[col_names_1.index('Genre')]
        genre2 = row2[col_names_2.index('Genre')]
        genre = get_union(genre1, genre2)
        merged_row.append(genre)

        # SubGenre - union of two sets
        subgenre1 = row1[col_names_1.index('SubGenre')]
        subgenre2 = row2[col_names_2.index('SubGenre')]
        subgenre = get_union(subgenre1, subgenre2)
        merged_row.append(subgenre)

        # rating1 - imdb_rating
        rating1 = row1[col_names_1.index('Rating')]
        merged_row.append(rating1)

        # rating2 - allmovie_rating
        rating2 = row2[col_names_2.index('Rating')]
        merged_row.append(rating2)

        # votes1 - imdb_votes
        votes1 = row1[col_names_1.index('Votes')]
        merged_row.append(votes1)

        # votes2 - allmovie_votes
        votes2 = row2[col_names_2.index('Votes')]
        merged_row.append(votes2)

        # Director - longer of the two if exists
        director1 = row1[col_names_1.index('Director')]
        director2 = row2[col_names_2.index('Director')]
        merged_director = get_longer(director1, director2)
        merged_row.append(merged_director)

        # Producer - allmovie if exists
        producer2 = row2[col_names_2.index('Producer')]
        merged_row.append(producer2)

        # Country - allmovie if exists
        country2 = row2[col_names_2.index('Country')]
        merged_row.append(country2)

        # Gross     - imdb if exists
```

```
	gross1 = row1[col_names_1.index('Gross')]
	merged_row.append(gross1)

	# OtherRating - allmovie if exists
	otherRating2 = row2[col_names_2.index('OtherRating')]
	merged_row.append(otherRating2)

	# Metascore - imdb if exists
	metascore1 = row1[col_names_1.index('Metascore')]
	merged_row.append(metascore1)

	return merged_row

def main():

	col_names, matched_keys = read_csv('../data/P2.csv')
	col_names_1, data_1 = read_csv('../data/imdb.csv')
	col_names_2, data_2 = read_csv('../data/allmovie.csv')

	data1_dict = {}
	for row in data_1:
		data1_dict[row[0]] = row

	data2_dict = {}
	for row in data_2:
		data2_dict[row[0]] = row

	output_column_heads = result_cols
	merged_table_file = open('merged_table.csv','w')
	merged_table_file.write(','.join(output_column_heads)+'\n')

	for matched_pair in matched_keys:

		pair_key = matched_pair[0]

		key1 = matched_pair[1]
		key2 = matched_pair[2]

		row1 = data1_dict[key1]
		row2 = data2_dict[key2]
```

```
            merged_row = merge_rows(pair_key, row1, col_names_1, row2,
col_names_2)
            #merged_table_file.write(','.join(col_names_1)+'\n')
            #merged_table_file.write(','.join(row1)+'\n')
            #merged_table_file.write(','.join(col_names_2)+'\n')
            #merged_table_file.write(','.join(row2)+'\n')
            merged_table_file.write(','.join(merged_row)+'\n')
            #merged_table_file.write('-------------------------\n\n')

      merged_table_file.close()

if __name__ == "__main__":
      main()
```

Data Analysis

Based on domain knowledge and the available attributes present in the schema we had a bunch of hypothesis which we wanted to validate. For instance, we expected to see a correlation between the movie rating and the amount of money the movie made. We also expected to see an increase in the amount of money made by movies over the years. In order to validate these hypothesis we relied on two data analysis techniques namely (1) correlation discovery and (2) OLAP style querying. In the following sections we elaborate on what we have concluded from our analysis.

Correlation between attributes

The following table describes correlation between pairs of the numeric attributes in our merged table. Greener values mean that the attribute-pair is highly correlated. Red color indicates that they are inversely correlated.

	Year	Runtime	rating1	rating2	votes1	votes2	Gross	OtherRating	Metascore
Year	1.000	-0.022	-0.212	-0.123	0.310	0.206	0.196	-0.248	-0.228
Runtime	-0.022	1.000	0.285	0.230	0.210	0.206	0.236	0.168	0.146
rating1	-0.212	0.285	1.000	0.744	0.407	0.418	0.156	0.581	0.597
rating2	-0.123	0.230	0.744	1.000	0.314	0.376	0.087	0.585	0.564
votes1	0.310	0.210	0.407	0.314	1.000	0.879	0.623	0.127	0.142
votes2	0.206	0.206	0.418	0.376	0.879	1.000	0.572	0.236	0.206
Gross	0.196	0.236	0.156	0.087	0.623	0.572	1.000	-0.013	-0.014
OtherRating	-0.248	0.168	0.581	0.585	0.127	0.236	-0.013	1.000	0.635
Metascore	-0.228	0.146	0.597	0.564	0.142	0.206	-0.014	0.635	1.000

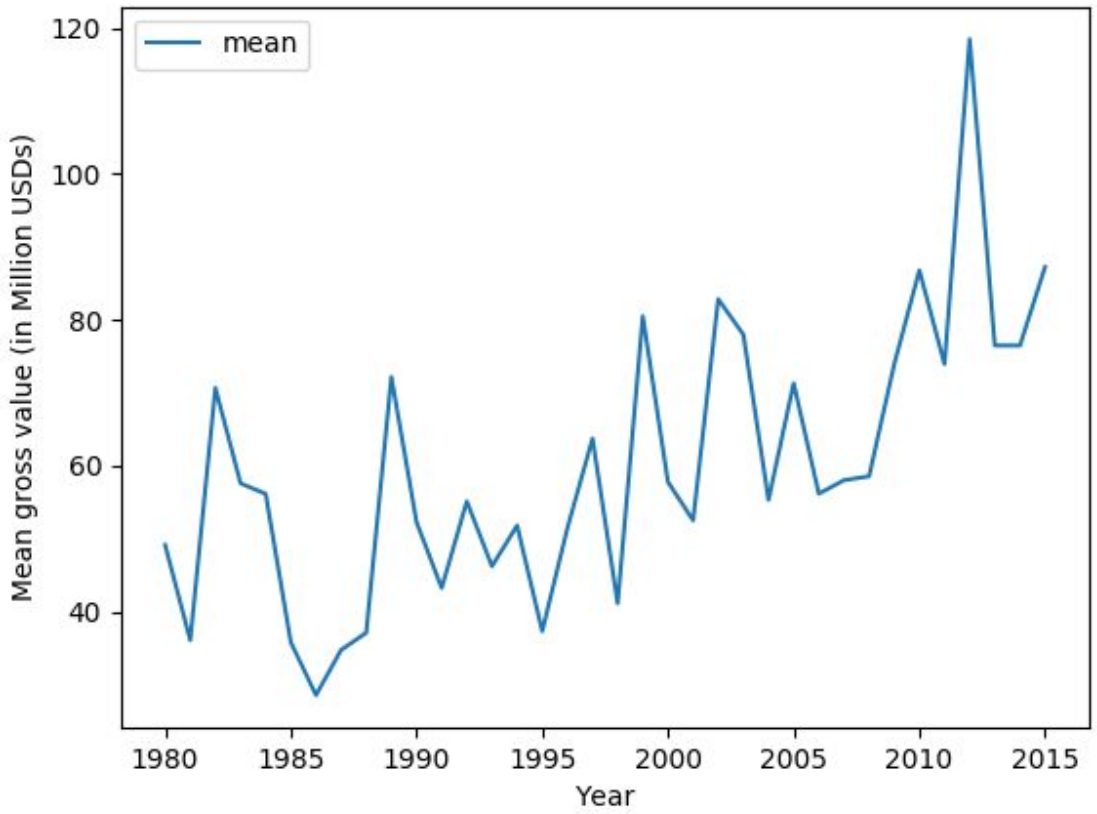
From the table, it is evident that the rating values on the two platforms have high correlation among them. The same thing is true for votes as well.

Metascore column represents a score given by critics. The metascore values and rating values are correlated, but the correlation is not that strong. This makes sense, because not all the movies liked by critics are also loved by general audience, and vice-versa.

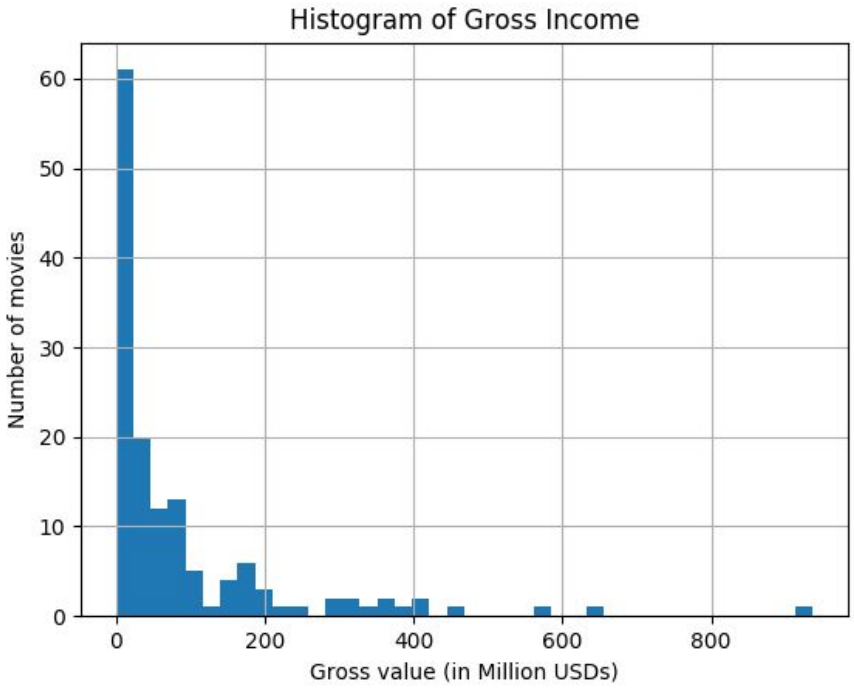
We can also see some correlation between ratings and votes. If we consider number of votes as how many people watched a movie, this suggests that it is related to the gross.

There seems to be a negative correlation between rating and year. This suggests that people used to like older movies slightly more than they do the new ones.

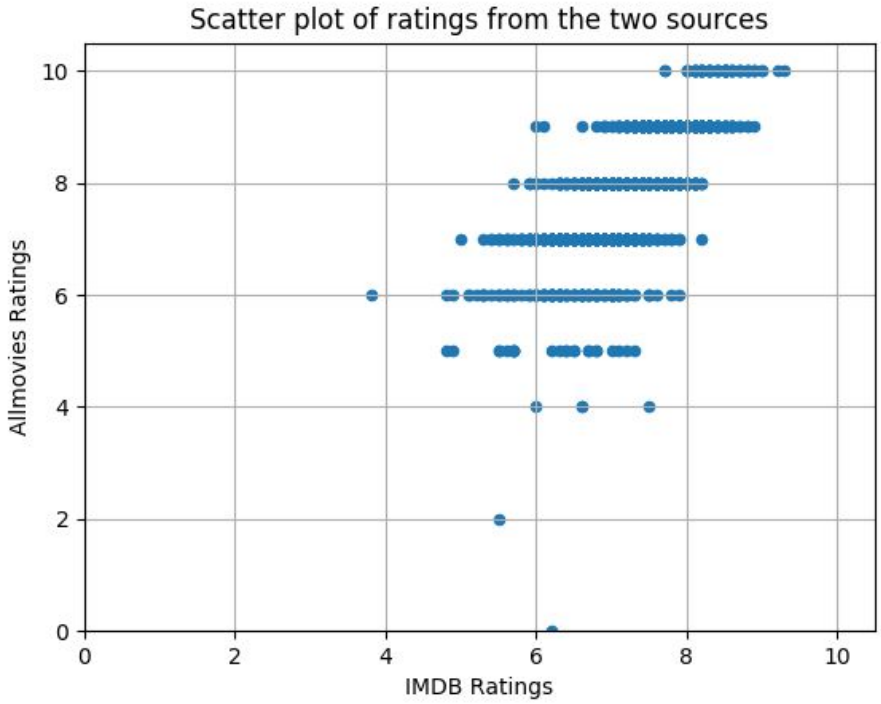
OLAP



Over the years the amount of money made by movies has been increasing.



We bucketed the movies by the gross money made by them to see if anything interesting comes up. However, we were not able to find anything interesting here.



As was evident from the correlation analysis we observe a high correlation between the user ratings on the two platforms.

Highest grossing movie per year (1997 - 2017):

Year	Name	Gross (in millions)
1997	Titanic	659.32
1998	There's Something About Mary	176.48
1999	Star Wars: Episode I - The Phantom Menace	474.54
2000	Cast Away	233.63
2001	Harry Potter and the Sorcerer's Stone	317.57
2002	Spider-Man	403.70
2003	Finding Nemo	380.84
2004	Spider-Man 2	373.58
2005	Star Wars: Episode III - Revenge of the Sith	380.26
2006	Pirates of the Caribbean: Dead Man's Chest	423.31
2007	Transformers	319.07
2008	The Dark Knight	534.85
2009	Avatar	760.50
2010	Toy Story 3	415.00
2011	Harry Potter and the Deathly Hallows; Part 2	381.01
2012	Marvel's The Avengers	623.35
2013	The Hunger Games: Catching Fire	424.66
2014	The Hunger Games: Mockingjay - Part 1	337.13
2015	Star Wars: The Force Awakens	936.66
2016	The Jungle Book	364.00
2017	Wonder Woman	412.56

Highest Rated Movies (1997 - 2017):

Year	Name	IMDB Ratings
1997	Life Is Beautiful	8.6
1998	Lock; Stock and Two Smoking Barrels	8.2
1999	Fight Club	8.8
2000	Memento	8.5
2001	The Lord of the Rings: The Fellowship of the Ring	8.8
2002	The Lord of the Rings: The Two Towers	8.7
2003	The Lord of the Rings: The Return of the King	8.9
2004	Black Friday	8.6
2005	Batman Begins	8.3
2006	The Prestige	8.5
2007	No Country for Old Men	8.1
2008	The Dark Knight	9
2009	The Secret in Their Eyes	8.2
2010	Inception	8.8
2011	The Intouchables	8.6
2012	The Dark Knight Rises	8.4
2013	12 Years a Slave	8.1
2014	Interstellar	8.6
2015	Mad Max: Fury Road	8.1
2016	La La Land	8.1
2017	Three Billboards Outside Ebbing; Missouri	8.3

Top 10 movies by gross:

Name	Year	Gross (in millions)
Star Wars: The Force Awakens	2015	936.66
Avatar	2009	760.50
Titanic	1997	659.32
Jurassic World	2015	652.27
Marvel's The Avengers	2012	623.35
Black Panther	2018	578.37
The Dark Knight	2008	534.85
Star Wars: Episode I - The Phantom Menace	1999	474.54
Avengers: Age of Ultron	2015	459.00
The Dark Knight Rises	2012	448.13

Top 10 movies by ratings:

Name	Year	IMDB Ratings
The Shawshank Redemption	1994	9.3
The Godfather	1972	9.2
The Godfather: Part II	1974	9.0
The Dark Knight	2008	9.0
Pulp Fiction	1994	8.9
The Good; the Bad and the Ugly	1966	8.9
The Lord of the Rings: The Return of the King	2003	8.9
Schindler's List	1993	8.9
Inception	2010	8.8
Star Wars: Episode V - The Empire Strikes Back	1980	8.8

Problems faced

- We started off with about 7000 movies in each of the source tables but were left with only about 2000 of them after matching. Our analysis would have been more concrete if we had more data in our merged table
- Attributes like genre, sub-genre and country were sets. This made it difficult to analyse them
- Data for movies before 1980 was not very reliable. While performing analysis it was evident that some data was incorrect

Future Work

In our analysis we did not make use of attributes like genre, sub-genre and country. This was mainly because of the way these attributes were stored (pipe separated in a single column). Given more time we would have liked to explode these attributes and ask interesting questions like the following:

- Do movies of some genre make more money than others
- Most popular genre based on country
- Are movies of a particular genre rated higher than other movies
- Do horror movies make less money because they are not prefered by a lot of people