University of Victoria

CSC - 501

REPORT (ASSIGNMENT – 1)

THE RELATIONAL DATA MODELS

SUBMITTED BY - GROUP B

Malik, Mona	V00935224
Singh, Jasbir	V00915704
Sannath Reddy Vemula	V00949217

Submitted To - Prof . Sean Chester

schester@uvic.ca

SECTION - 1

Data Modelling

- ### ER diagram

We are majorly using four tables as the core base for our Data Analysis. The tables are Movies, Rating, Links and tags. In order to have a many to many relation between the genre and the movies table, new tables called genre map (genreID, genre) and genre (genreID, genre) are created to reduce the query time. In the movies table implementations, the genres, a multi valued attribute has been normalized using 1st Normal form (atomic attribute only) for the sake of optimization. In the movies table we have created a separate column "year" (extracting year value from the title) during the data processing.

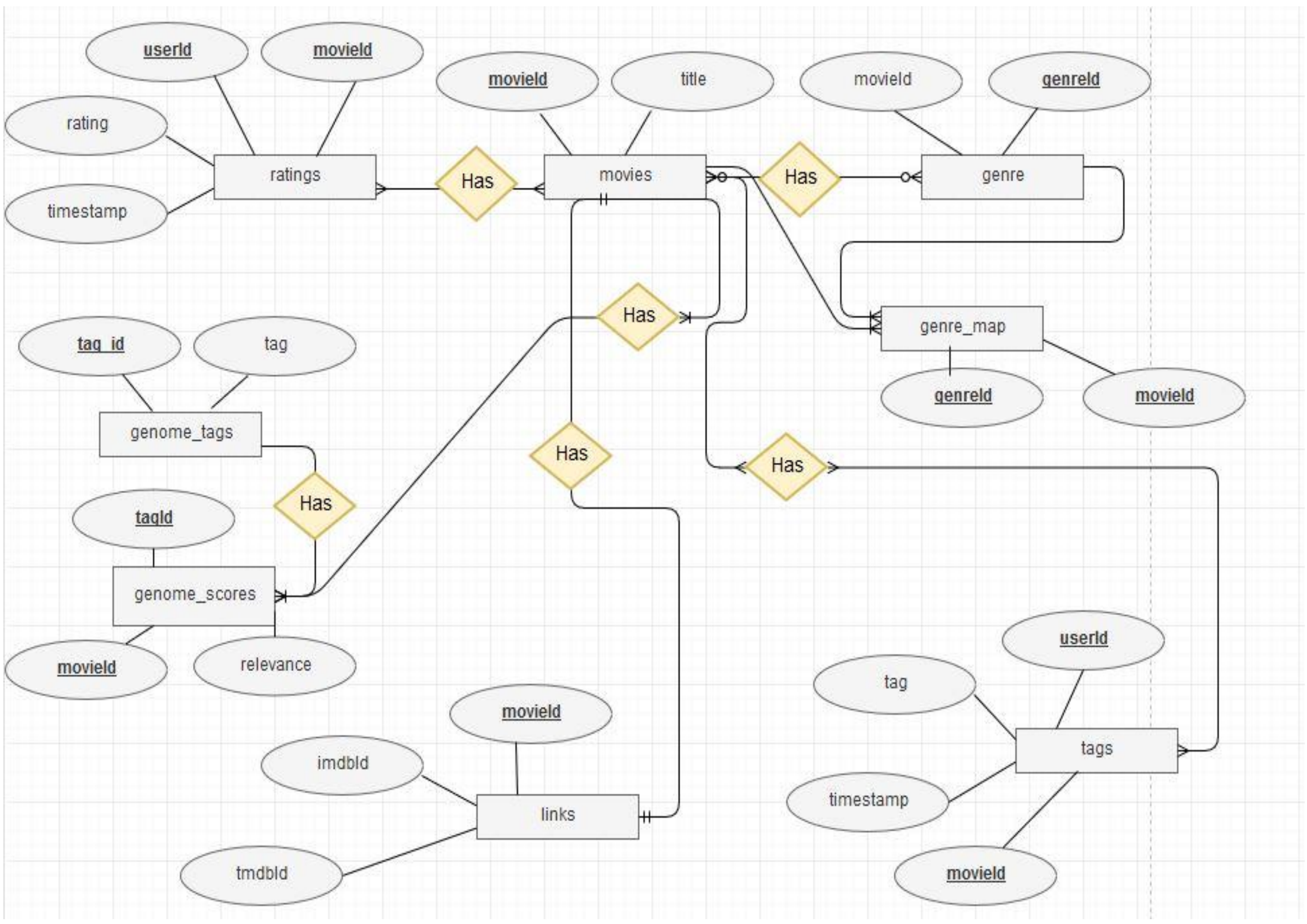
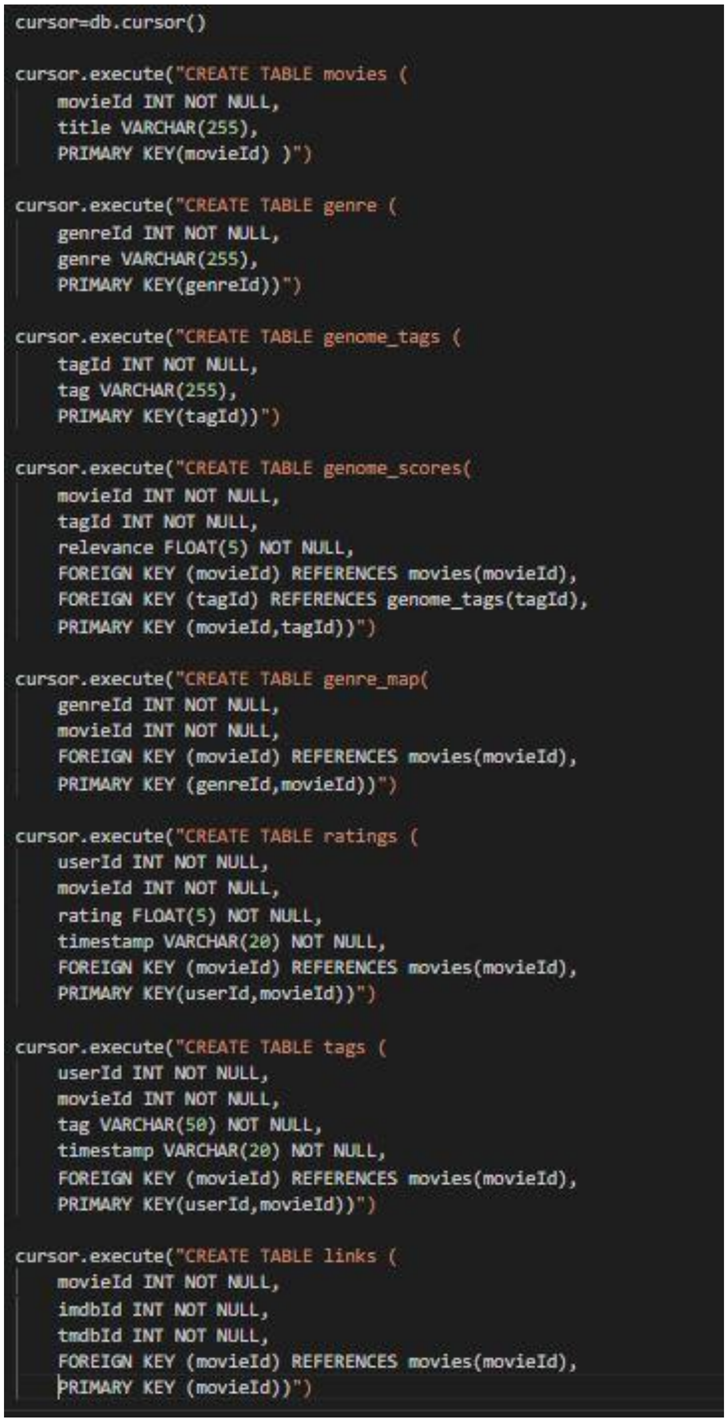


Table Structure of the ER DIAGRAM



```
cursor=db.cursor()

cursor.execute("CREATE TABLE movies (
    movieId INT NOT NULL,
    title VARCHAR(255),
    PRIMARY KEY(movieId) )")

cursor.execute("CREATE TABLE genre (
    genreId INT NOT NULL,
    genre VARCHAR(255),
    PRIMARY KEY(genreId))")

cursor.execute("CREATE TABLE genome_tags (
    tagId INT NOT NULL,
    tag VARCHAR(255),
    PRIMARY KEY(tagId))")

cursor.execute("CREATE TABLE genome_scores(
    movieId INT NOT NULL,
    tagId INT NOT NULL,
    relevance FLOAT(5) NOT NULL,
    FOREIGN KEY (movieId) REFERENCES movies(movieId),
    FOREIGN KEY (tagId) REFERENCES genome_tags(tagId),
    PRIMARY KEY (movieId,tagId))")

cursor.execute("CREATE TABLE genre_map(
    genreId INT NOT NULL,
    movieId INT NOT NULL,
    FOREIGN KEY (movieId) REFERENCES movies(movieId),
    PRIMARY KEY (genreId,movieId))")

cursor.execute("CREATE TABLE ratings (
    userId INT NOT NULL,
    movieId INT NOT NULL,
    rating FLOAT(5) NOT NULL,
    timestamp VARCHAR(20) NOT NULL,
    FOREIGN KEY (movieId) REFERENCES movies(movieId),
    PRIMARY KEY(userId,movieId))")

cursor.execute("CREATE TABLE tags (
    userId INT NOT NULL,
    movieId INT NOT NULL,
    tag VARCHAR(50) NOT NULL,
    timestamp VARCHAR(20) NOT NULL,
    FOREIGN KEY (movieId) REFERENCES movies(movieId),
    PRIMARY KEY(userId,movieId))")

cursor.execute("CREATE TABLE links (
    movieId INT NOT NULL,
    imdbId INT NOT NULL,
    tmdbId INT NOT NULL,
    FOREIGN KEY (movieId) REFERENCES movies(movieId),
    PRIMARY KEY (movieId))")
```

The basic SQL operations like querying, making joins, finding and dropping null values , sorting table ,indexing were used for internal data modelling using pandas. The merge function of pandas helped to make all type of joins possible in SQL. Although it was not an issue with the smaller data sets to use those operations with the help of pandas, but for larger databases, pandas suffered from high query times. Later we decided to shift to SQLite for these operations. The possible explanation we found for this was that pandas fetched data to the code, while SQLite performed the coding operations on the data in place and did not require any data shifting

SECTION - 2

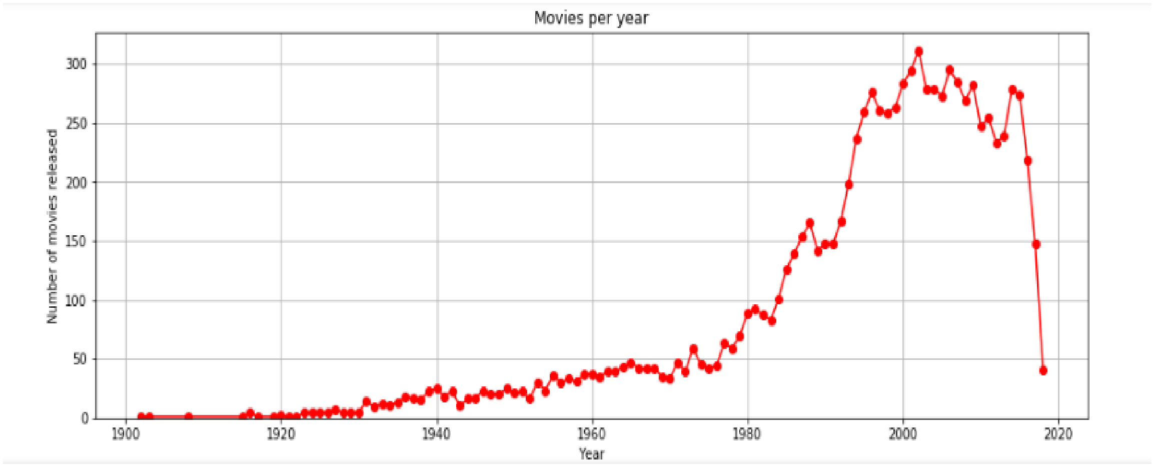
Visualization (Transforming Raw Data to Insights)

PLOT – 1 : No. of Movies per year

FILES OR TABLE USED – movies.csv

To achieve the desired result we have extracted the year from the movie title and created a new column "years" internal to pandas. We also used internal pandas data processing instructions like group by , sorting , checking and dropping null values and other useful operations to move towards accuracy.

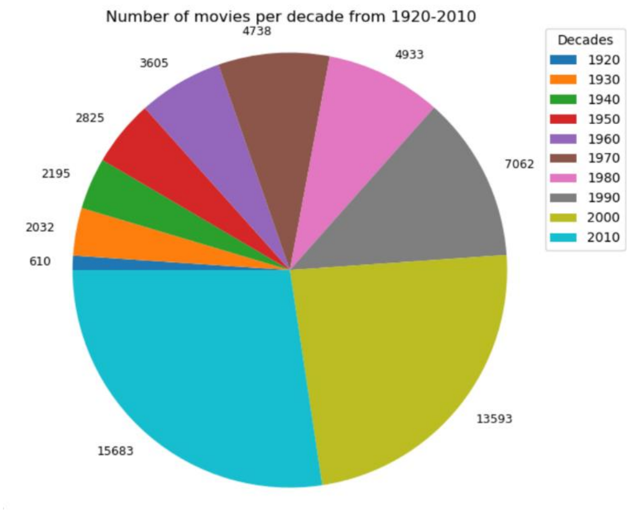
INSIGHTS – From the visualization we can derive that the number of movies released increased almost exponentially until 2000 might be due to the improvement in the photography technology and widespread use of television, digital distributions like CDs, then flatten and drop significantly in 2014.



PLOT – 2 : Visualization for the number of movies per decade (Pie chart)

FILES OR TABLE USED – movies.csv

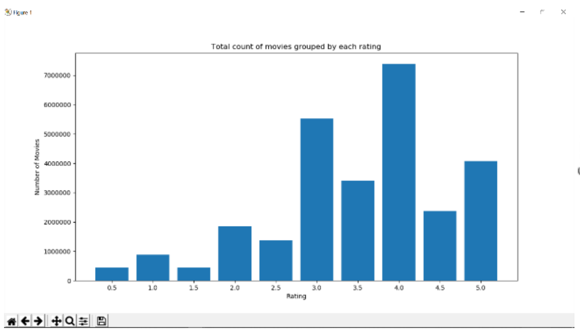
INSIGHTS - Aughts and 10's are the decades during which more than 50% of the movies has been released during the last century.

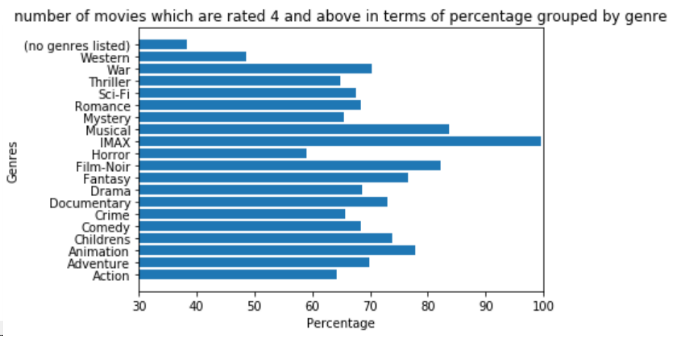


PLOT – 3: **Total Count of Movies for each Rating**

FILES OR TABLE USED - Ratings.csv

INSIGHTS - After analyzing users rating for the movies, it was found that approximately half of the movies were rated 4 or higher by the users. In order to find the users' taste of the movies, the ratings were analyzed against the movies' genres. It was found that they liked approximately every IMAX movie while western movies were greeted with low ratings. Although some users might be inclined to rate every movie low while the other ones being quite generous, but we did not take this bias into account.



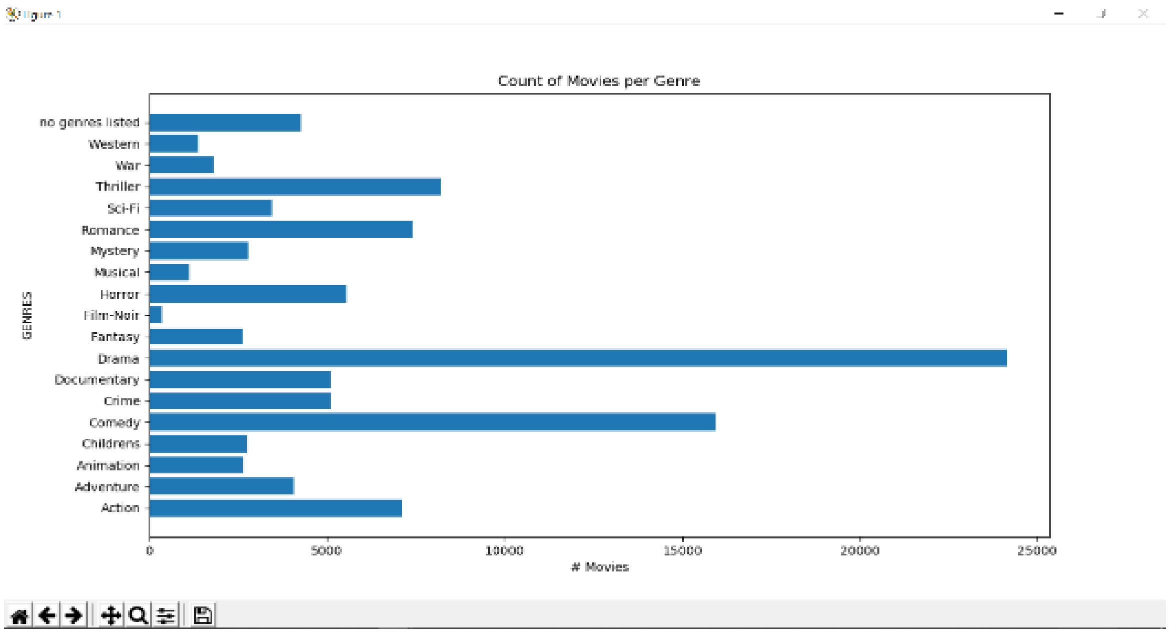


#

PLOT – 4: **Count of Movies per Genre**

FILES OR TABLE USED – movie.csv

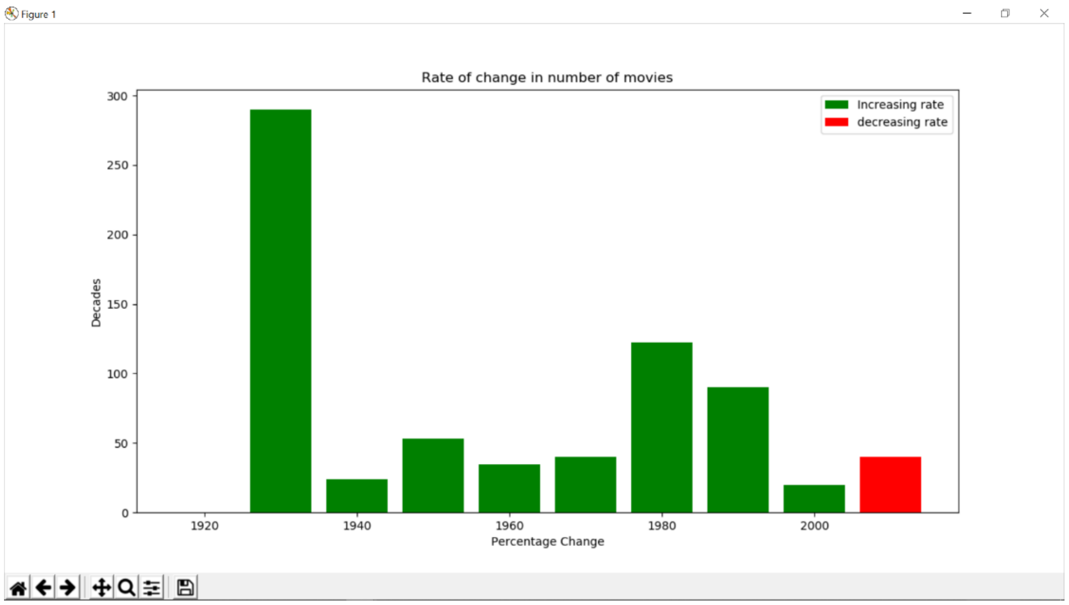
INSIGHTS - After analyzing all movies, it was found that most number of movies released were related to drama or comedy. But in contrast, the percentage of high ratings for these genres were quite low. Because most of the movies related to these genres were released before 2000 and rating only started after 2000, it might be a possible bias for this difference.



PLOT – 5 : Percentage Change in movies

FILES OR TABLE USED – movies.csv

INSIGHTS – For 10k data set, decreasing rate can be seen during 2010 . For 60k data set we analyze there was no decreasing rate through the entire decade . Additionally, 1930's is the decade during which the increasing rate was highest for both the data sets.

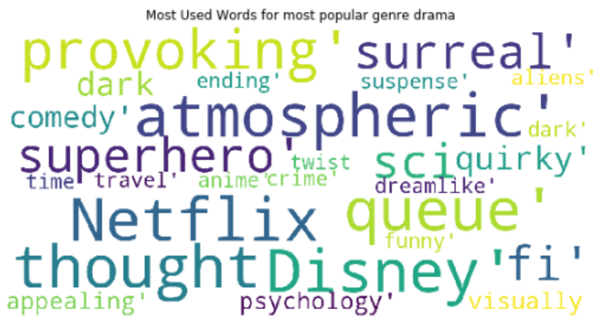


PLOT – 6 : Popular tags given by users to two most popular genre – DRAMA and Comedy

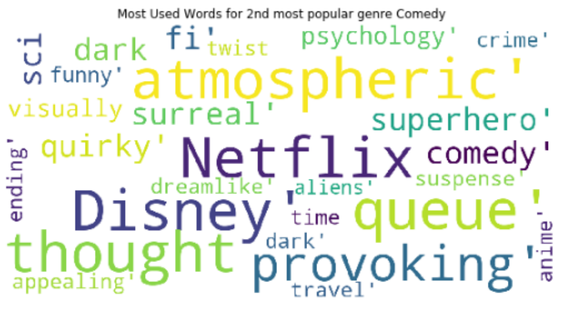
FILES OR TABLE USED – movies.csv , tags.csv

To get the desired result we have merge the movies and tag file and then achieved the 1st normal form by making multivalued genre attribute to contain atomic value instead.

INSIGHTS – From the word cloud visualization we can derive that the most popular tags for DRAMA are (suspense, provoking, dark, appealing, quirky, funny etc.) and in COMEDY are (quirky, funny, comedy, dreamlike, twist etc.) . From the tags for respective genre we can see the most popular tag given by user justify with genre domain.



DRAMA



COMEDY

SECTION - 3

Scalability

Processor - intel i7 7th gen CPU 2.7GHz - 2.9GHz 64-bit OSx64 based processor
RAM - 8 gb
Tool/software - Anaconda version 2019.07
Language – Python 3.7

Average Time taken for the execution of the above presented graphs:

- Movies per decade
  for 10k movies dataset - 3.4 seconds
  for 60k movies dataset - 5.9 seconds

- Rate of change in movies released per decade
  for 10k movies dataset - 3.3 seconds
  for 60k movies dataset - 3.74 seconds

- Movie Count grouped by Ratings
  i. Loading file (CSV with 27million records) to Database and converting the file into table in the form of chunks:
     when executed in python IDLE terminal - 5 minutes (approx..)
     when executed with Anaconda in jupyter notebook -10 minutes (approx.)

  ii. Executing SQL statements to fetch from table (14 million records)
     with pandas - 2 minutes (approx.) (for an instance it took 1min 20sec)
     with sqlite3 - under a sec (for an instance it took 3.59ms)

- Movie Count per rating
  for 10k dataset - 4seconds (approx.)
  for 60k dataset - 5seconds (approx.)

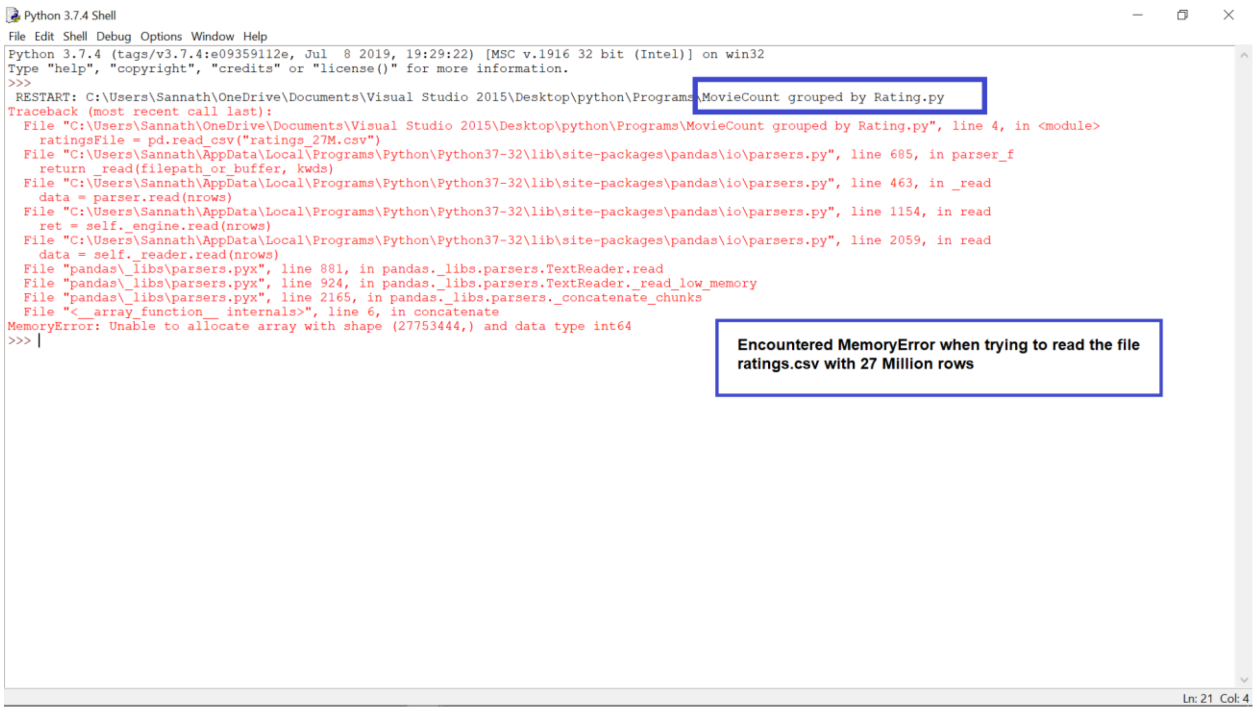
We successfully tested our code for different amount of data; like 10k, 60k, 100k, 20 million and 27 million rows. We were successful in fetching the records quicker using the SQLite.

Challenges and solutions:

We went through some of the challenges during the working of our project but here we are presenting some of the most important related to the scalability part and the solutions.

- ### Loading 27 m records:

We got out of memory exception when we were trying to load 27 million rows using pandas. The corresponding screenshot has been attached with this report. Initially we were using pandas to fetch the csv files and realized that pandas keeps everything in memory and could not work on larger files containing billion records. In order to process these larger files, we used database in the backend and used data chunking in order to load csv file into the data base. SQL Alchemy library was used to perform this operation. Because the data was loaded in chunks, smaller RAM did not restrict from processing larger files.



```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sannath\OneDrive\Documents\Visual Studio 2015\Desktop\python\Programs\MovieCount grouped by Rating.py
Traceback (most recent call last):
  File "C:\Users\Sannath\OneDrive\Documents\Visual Studio 2015\Desktop\python\Programs\MovieCount grouped by Rating.py", line 4, in <module>
    ratingsFile = pd.read_csv("ratings_27M.csv")
  File "C:\Users\Sannath\AppData\Local\Programs\Python\Python37-32\lib\site-packages\pandas\io\parsers.py", line 685, in parser_f
    return _read(filepath_or_buffer, kwds)
  File "C:\Users\Sannath\AppData\Local\Programs\Python\Python37-32\lib\site-packages\pandas\io\parsers.py", line 463, in _read
    data = parser.read(nrows)
  File "C:\Users\Sannath\AppData\Local\Programs\Python\Python37-32\lib\site-packages\pandas\io\parsers.py", line 1154, in read
    ret = self._engine.read(nrows)
  File "C:\Users\Sannath\AppData\Local\Programs\Python\Python37-32\lib\site-packages\pandas\io\parsers.py", line 2059, in read
    data = self._reader.read(nrows)
  File "pandas\_libs\parsers.pyx", line 881, in pandas._libs.parsers.TextReader.read
  File "pandas\_libs\parsers.pyx", line 924, in pandas._libs.parsers.TextReader._read_low_memory
  File "pandas\_libs\parsers.pyx", line 2165, in pandas._libs.parsers._concatenate_chunks
  File "<__array_function__ internals>", line 6, in concatenate
MemoryError: Unable to allocate array with shape (27753444,) and data type int64
>>> |
```

Encountered MemoryError when trying to read the file ratings.csv with 27 Million rows

- **To minimize the query time**

Although we were preprocessing the data and adding it to database, but we were fetching data from the database with the help of pandas. Only later we realized that performing the querying operations in SQLite was a lot faster than pandas and improved the performance significantly. Pandas took an average time of 2 minutes for each SQL select statement. Using SQLite3 library the time was brought down to 3.4 milliseconds.

SECTION - 4

Relationship to modern practise:

- We used a library called **SQL Alchemy** in order to load the data from csv file into the db. It uses data chunking as taught in the class in order to process larger data sets. In order to create database and load 27 million records, SQL Alchemy approach took around 5 minutes.

- We tried to use another library called **"Vaex"** as an alternative to SQL Alchemy which claims to read 1 billion record in just 1 second. As of now, the library is in its rudimentary form and also we have very limited reference resources to work on it.

- We tried to link our assignment solution with **Hill View paper** (the paper discussed in class) as it could process trillion of rows. But the problem we faced with it was that it required 8 clusters and a specific search engine for it. But due to the limited resources (commodity hardware), we decide it for a future topic as of now. So, we believe that for a commodity hardware using SQL Alchemy is a better choice as of now.