MOVIELENS

Table of Contents:

Tools used:

I have used Snowflake and SQL developer for database design, data storage and analysis of data using queries. Also used tableau to visually analyze the data.

Executive Summary:

This extensive data warehousing project intends to provide a solid basis for perceptive analytics and well-informed decision-making by centralizing and analyzing enormous MovieLens platform datasets gathered from user interactions. Building an effective and scalable data warehouse that compiles information, such as user reviews, ratings, watching trends, and demographic data, is the main goal of the project. With help cutting-edge data warehousing technologies like "Snowflake", our architecture guarantees the best possible data processing, retrieval, and storage. On the other hand, Robust Extract, Transform, Load (ETL) processes are implemented to seamlessly integrate data into the data warehouse using snowflake. This ensures data consistency, accuracy, and timeliness.

This project also leverages the use of advance analytics by applying sophisticated algorithms to user behavior data, we can generate more accurate movie recommendations, personalized content suggestions, and predictive models for user engagement. Finally, Interactive dashboards and real-time reporting tools are implemented to provide stakeholders with immediate access to key performance indicators. This empowers decision-makers with timely insights into user behavior, content popularity, and overall platform performance.

Problem Statement:

In the era of information overload, users often face difficulties in discovering movies that align with their preferences. Hence, the contemporary landscape of the entertainment industry, data-driven decision-making has become paramount for optimizing user experiences and ensuring the success of digital platforms. However, building an advanced recommendation system goes beyond traditional collaborative and content-based filtering. To address this challenge, our goal is to develop a robust and accurate movie recommendation system using the MovieLens dataset. MovieLens is a popular dataset containing user ratings and movie metadata, making it an ideal resource for building personalized recommendation algorithms. The system should not only provide accurate predictions but also offer insights into the reasoning behind the recommendations. The success of the project will be measured based on the accuracy of movie recommendations, user satisfaction, and the system's ability to handle scalability challenges.

Literature Review:

The project uses the MovieLens dataset which involves in summarizing and analyzing relevant research papers, articles, and publications in the field of recommender systems, collaborative filtering, and related topics.

One direct research site we could learn was MovieLens site run by “GroupLens Research at the University of Minnesota” which uses "collaborative filtering" technology to make recommendations of movies that you might enjoy, and to help you avoid the ones that you won't. Based on your movie ratings, MovieLens generates personalized predictions for movies you haven't seen yet. MovieLens is a unique research vehicle for dozens of undergraduates and graduate students researching various aspects of personalization and filtering technologies.

We have also used LensKit for research recommender algorithms, evaluation techniques, or user experience.

Michael D. Ekstrand. 2020. LensKit for Python: Next-Generation Software for Recommender Systems Experiments. In Proceedings of the 29th ACM International Conference on Information and Knowledge Management *(CIKM '20). doi 10.1145/3340531.3412778. arXiv:1809.03125 [cs.IR].*

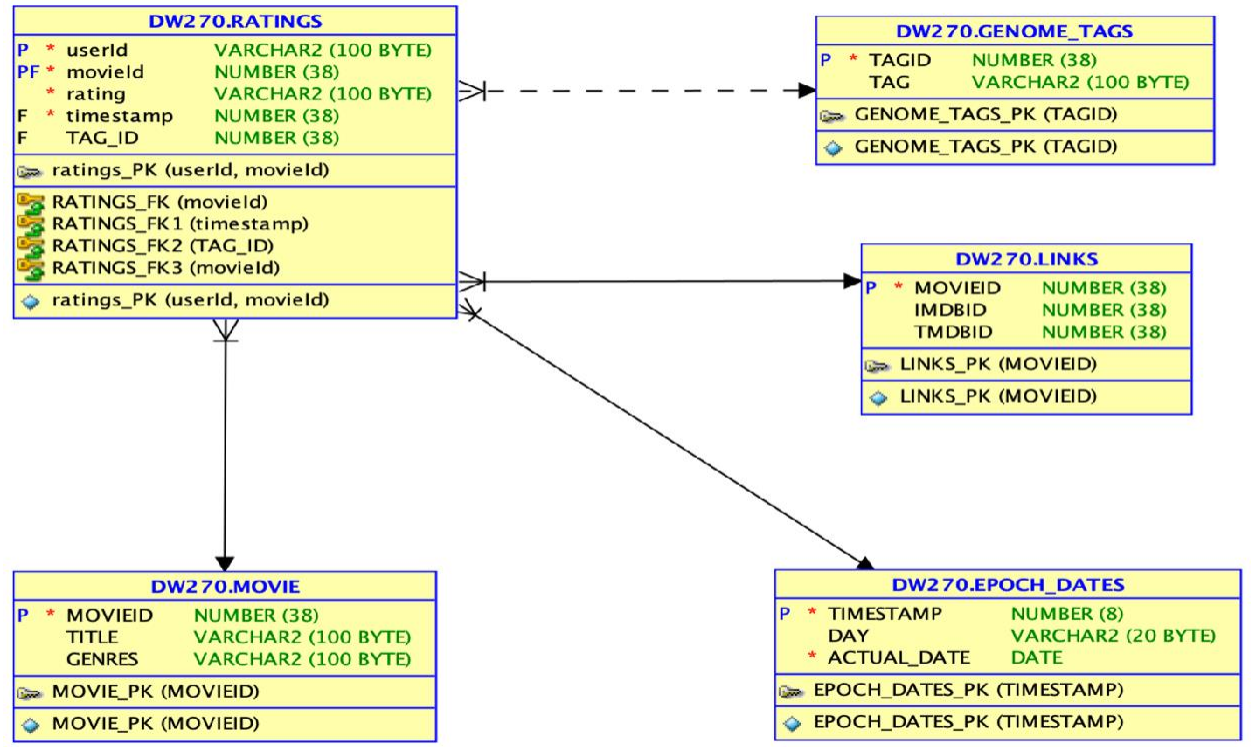
Data Collection and Preparation:

Major source of data that was utilized in this project was “MovieLens”. The dataset describes 5-star rating and free-text tagging activity. It contains 25000095 ratings and 1093360 tag applications across 62423 movies. These data were created by 162541 users between January 09, 1995, and November 21, 2019. This dataset was generated on November 21, 2019. Users were selected at random for inclusion. All selected users had rated at least 20 movies. No demographic information is included. Each user is represented by an id, and no other information is provided.

I created a pipeline in “Talend” and imported the data into the snowflake’s cloud Data Warehouse. I initially performed Data transformation, cleaning, and validations for prepare the data to be ready for our analysis.

Database Design:

ER Diagram shows relationships among different entity sets that are stored in MovieLens schema database.



Ratings: This is a fact table, which contains the actual data of ratings provided by the users. We have USERID (Unique id for user), MOVIEID (Unique id for a movie), RATING (rating in numeric value), TIMESTAMP (time when the rating was provided by user in EPOCH time format).

Movie: This is a dimension table which contains information of a movie. It contains MOVIEID (Unique id), Title and Genre of the movie.

Genome tags: This is a dimension table that contains the description of each TAGID.

Links: Links is another dimension table which provides IMDBID and TMDBIB for a given MOVIEID.

EPOCH Dates: This dimension table contains actual dates of the epoch timestamps.

Analytical Queries:

1. Aggregates movies for a sample of users

```
SELECT TITLE, round (AVG("rating"),2) AS avg_rating, COUNT (*) AS observations
FROM ratings
INNER JOIN MOVIES ON RATINGS."movieId"=MOVIES.MOVIEID
WHERE "userId" IN (SELECT "userId"
                   FROM RATINGS SAMPLE (10)
                   GROUP BY MOVIEID, TITLE
                   HAVING (COUNT (*) > 5) AND (AVG("rating") > 3))
ORDER BY avg_rating DESC;
```

This query aggregates movies for a sample of users, where the average rating is greater than 3. The count is given as observations. This aggregation analysis can help in movie recommendations to the users.

Output:

	TITLE ···	AVG_RATING	OBERVATIONS
1	Forsyte Saga, The (1967)	4.5	6
2	Letter from Siberia (1957)	4.5	7
3	Limuzins Janu nakts krasa (1981)	4.5	6
4	New York: A Documentary Film (1999)	4.5	6
5	The Criminal Excellency Fund (2018)	4.5	6
6	Planet Earth II (2016)	4.49	1119
7	Planet Earth (2006)	4.47	1744
8	Love on the Sidelines (2016)	4.43	7
9	Love 911 (2012)	4.43	7
10	Michael Jackson: Dangerous Tour (Bucharest, 1992) (1992)	4.43	7

2. Aggregates all the user ratings.

```
SELECT
MOVIEID, TITLE, round (AVG("rating"),2) AS avg_rating
FROM MOVIES
INNER JOIN RATINGS on RATINGS."movieId"=MOVIES.MOVIEID
GROUP BY MOVIEID, TITLE
ORDER BY avg_rating DESC;
```

This query aggregates the user ratings based on average movie rating for each movie. This analysis can help us know, some good movies to recommend the users based on the average ratings the movie has gained from the ratings provided by the users.

Output:

	MOVIEID	TITLE	AVG_RATING
1	137849	9 Full Moons (2013)	5
2	147450	Amigo Undead	5
3	145330	Unidentified (2006)	5
4	143888	Gallows Road (2015)	5
5	179391	Detonator (2013)	5
6	142160	Carlos Spills the Beans (2014)	5
7	141777	White Creek (2014)	5
8	139703	Mona (2012)	5
9	139473	Lost Woods (2012)	5
10	139453	Hide (2011)	5

3. Find the top-rated movies based on average ratings.

```
SELECT "movieId", AVG("rating") AS avg_rating
FROM RATINGS
GROUP BY "movieId"
ORDER BY avg_rating DESC
LIMIT 10;
```

This query yields us the top 10 movies with highest average ratings provided by the users. For example, in a recommendation system where it suggests the top movies for the week or month or year, this analysis can be used.

Output:

	movieId	AVG_RATING
1	205277	5
2	208597	5
3	123569	5
4	183585	5
5	159048	5
6	197595	5
7	173413	5
8	157617	5
9	166576	5
10	200650	5

4. Identify movies with the most reviews.

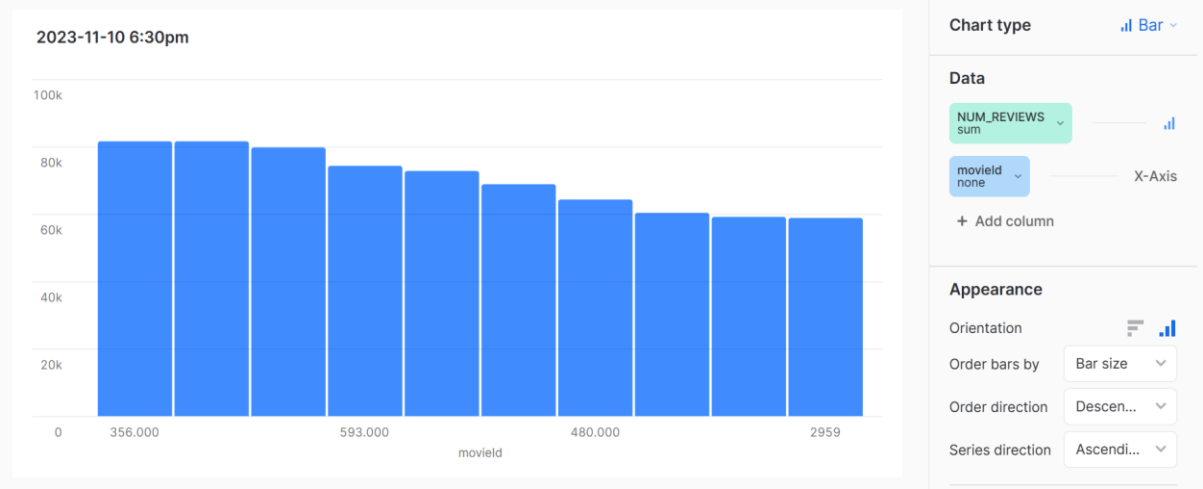
```
SELECT "movieId", COUNT("rating") AS num_reviews
FROM RATINGS
GROUP BY "movieId"
ORDER BY num_reviews DESC
LIMIT 10;
```

This query gives us the most rated movies, based on the number of user ratings. This analysis can gain us an insight of what are the movies that are watched more and being rated many times.

Output:

	movieId	···	NUM_REVIEWS
1	356		81491
2	318		81482
3	296		79672
4	593		74127
5	2571		72674
6	260		68717
7	480		64144
8	527		60411
9	110		59184
10	2959		58773

Below is the Snowflake's In-built visualization for the above query,



5. Analyze the distribution of movies across different genres.

```
SELECT TRIM(VALUE::STRING) AS genre, COUNT(m.movieId) AS num_movies
FROM MOVIES m,
LATERAL FLATTEN (INPUT => SPLIT(m.genres, '|')) AS genre
GROUP BY genre
ORDER BY num_movies DESC;
```

This query helps us know how many movies has been made from different genres. Total number of movies for each genre has been shown based on count of movies. This analysis can answer us research questions like what the top genres in the movie industry are.

Output:

	GENRE	NUM_MOVIES
1	Drama	25606
2	Comedy	16870
3	Thriller	8654
4	Romance	7719
5	Action	7348
6	Horror	5989
7	Documentary	5605
8	Crime	5319
9	(no genres listed)	5062
10	Adventure	4145

6. Explore user tagging behavior.

```
SELECT "userId", COUNT("tag") AS num_tags
FROM TAGS
GROUP BY "userId"
ORDER BY num_tags DESC
LIMIT 10;
```

This Query outputs the tagging behaviors for each user. How many tags each user has given for movies on a whole.

Output:

	userId	NUM_TAGS
1	6550	183254
2	21096	20317
3	62199	13699
4	160540	12075
5	155146	11443
6	70092	10582
7	131347	10192
8	14116	10167
9	31047	8457
10	141263	7114

7. Calculate the average rating for movies over time.

```
SELECT YEAR(TRY_TO_DATE(DATE)) AS year, round(AVG("rating"),2) AS avg_rating
FROM RATINGS
GROUP BY year
ORDER BY year;
```

This query outputs the values of average ratings of all the movies over the years. Each year is considered and average of all the movie ratings of that year is provided. This can show which year has given good movies according to users. Comparing performance in years and look at the trend of movie ratings is possible.

Output:

	··· YEAR	AVG_RATING
1	1995	3.67
2	1996	3.55
3	1997	3.59
4	1998	3.51
5	1999	3.62
6	2000	3.58
7	2001	3.53
8	2002	3.49
9	2003	3.48
10	2004	3.43
11	2005	3.43
12	2006	3.47
13	2007	3.47
14	2008	3.54
15	2009	3.51

8. Calculate the total number of ratings in each quarter of years (ROLLUP).

```
SELECT
YEAR(TRY_TO_DATE(DATE)) AS year,
QUARTER(TRY_TO_DATE(DATE)) AS quarter,
COUNT(*) AS rating_count
FROM RATINGS
GROUP BY ROLLUP(year, quarter)
ORDER BY year, quarter;
```

Using rollup, we have found out the number of ratings of each quarter. This can make us evident that which quarter of the month users are actively watching movies or ratings the movies and look at the trend.

Output:

	YEAR	QUARTER	RATING_COUNT
1	1995	1	3
2	1995	null	3
3	1996	1	8605
4	1996	2	443305
5	1996	3	524204
6	1996	4	453979
7	1996	null	1430093
8	1997	1	249679
9	1997	2	234829
10	1997	3	68668
11	1997	4	73026
12	1997	null	626202
13	1998	1	53684
14	1998	2	39112
15	1998	3	124136

9. Calculate the total number of ratings in each quarter of years (CUBE).

```
SELECT
YEAR(TRY_TO_DATE(DATE)) AS year,
QUARTER(TRY_TO_DATE(DATE)) AS quarter,
COUNT(*) AS rating_count
FROM RATINGS
GROUP BY CUBE (year, quarter)
ORDER BY year, quarter;
```

Using Cube instead of rollup, we have found out the number of ratings of each quarter. This can make us evident that which quarter of the month users are actively watching movies or ratings the movies and look at the trend.

Output:

	YEAR	QUARTER	RATING_COUNT
113	2018	1	331744
114	2018	2	279675
115	2018	3	328969
116	2018	4	370373
117	2018	null	1310761
118	2019	1	383745
119	2019	2	322169
120	2019	3	331892
121	2019	4	162828
122	2019	null	1200634
123	null	1	6204397
124	null	2	5916668
125	null	3	6043372
126	null	4	6835658
127	null	null	25000095

10. Find the top genres with highest average ratings.

```
SELECT TRIM (VALUE::STRING) AS genre, round(AVG("rating"),2) as Average_Rating
FROM MOVIES m inner join RATINGS r on m.movieid=r."movieId",
LATERAL FLATTEN(INPUT => SPLIT(m.genres, '|')) AS genre
GROUP BY genre
ORDER BY Average_Rating DESC;
```

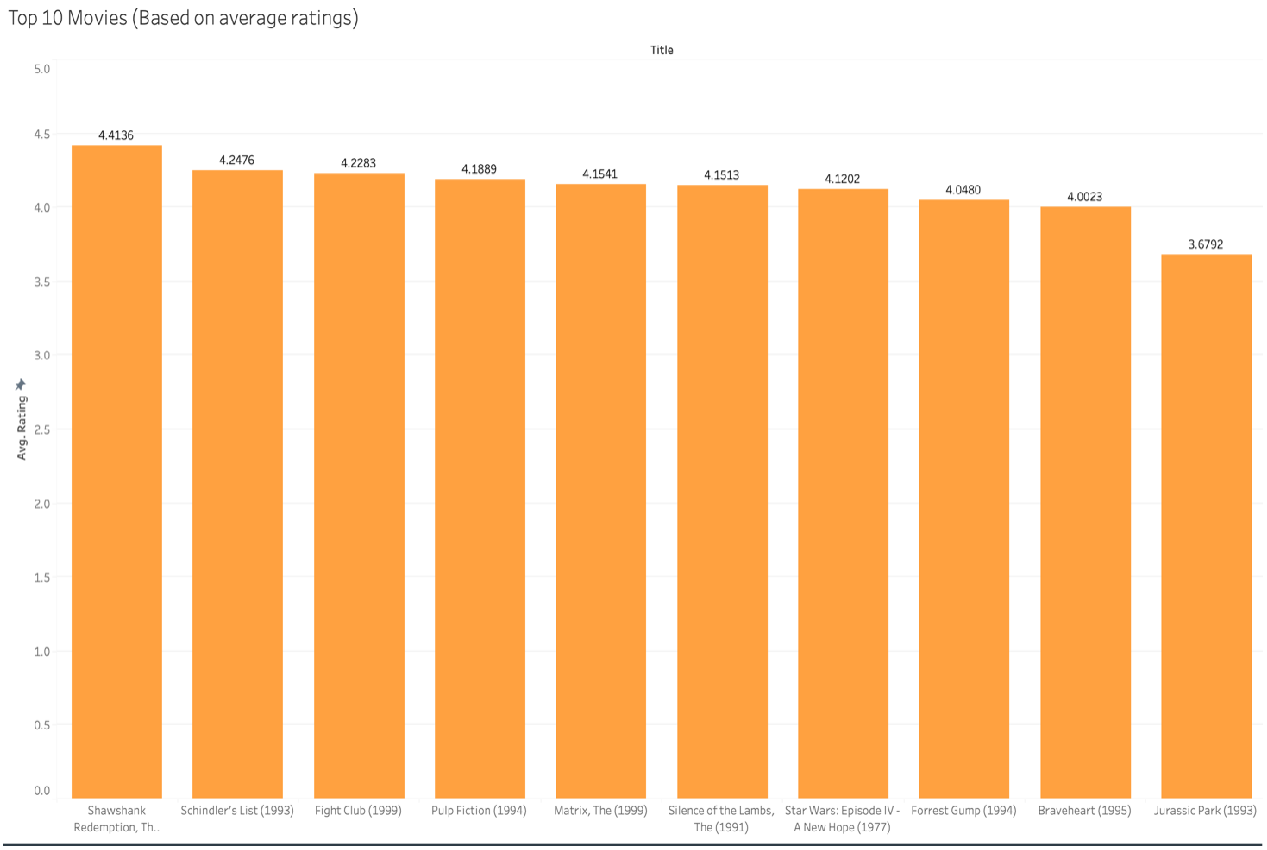
This query gives the genres with highest ratings. It is calculates based on grouping movies of each genre and taking of their average ratings. Aggregation on the rating of all movies and grouping of genres is considered here. This can showcase what time of genres have been performing better in the industry according to the users' ratings.

Output:

	GENRE	···	AVERAGE_RATING
1	Film-Noir		3.93
2	War		3.79
3	Documentary		3.71
4	Crime		3.69
5	Drama		3.68
6	Mystery		3.67
7	Animation		3.61
8	IMAX		3.6
9	Western		3.59
10	Musical		3.55

Data Visualization:

1. Presenting the top 10 movies based on average ratings given by the users.



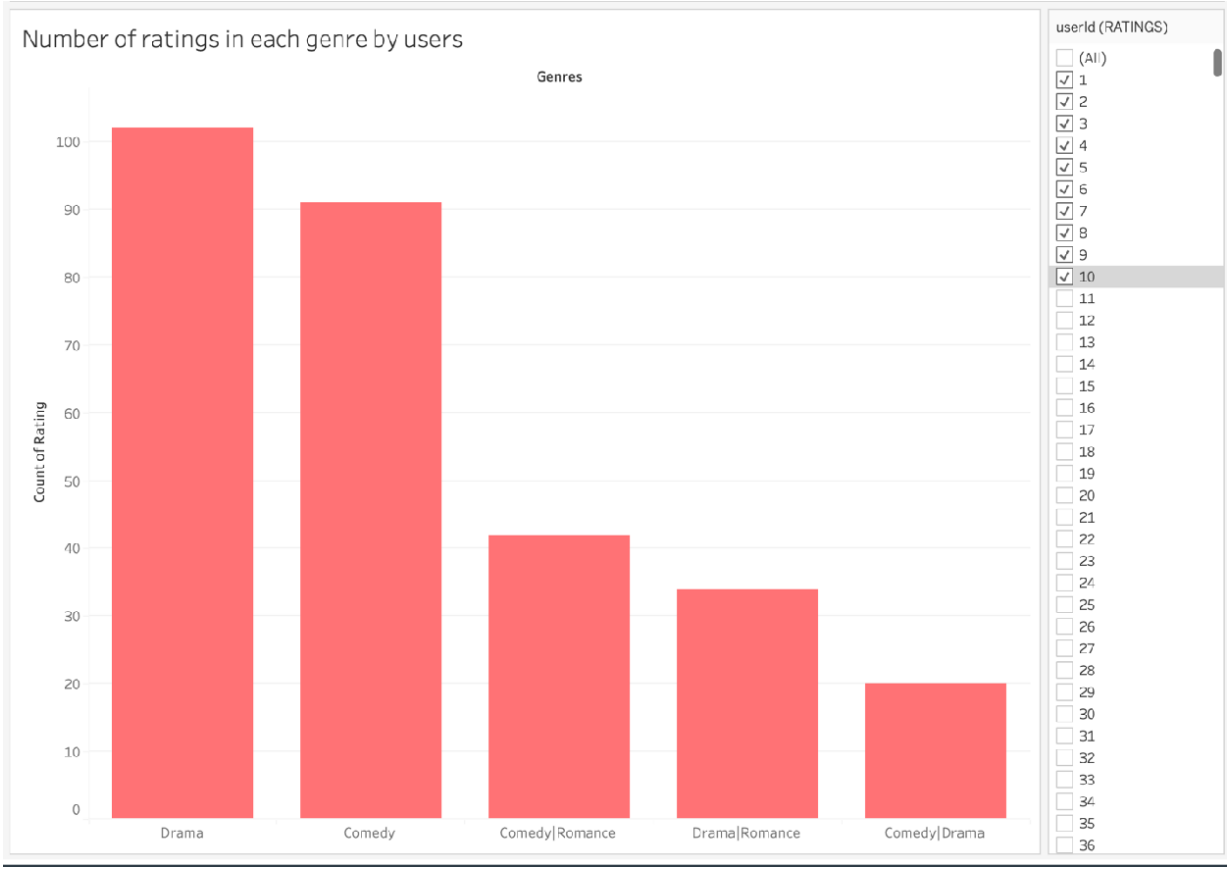
2. Presenting the top 10 movies based on Total Number of user ratings.

Top 10 Movies(Based on the number of user ratings)

Title	
Forrest Gump (1994)	81,491
Shawshank Redemption, The (1994)	81,482
Pulp Fiction (1994)	79,672
Silence of the Lambs, The (1991)	74,127
Matrix, The (1999)	72,674
Star Wars: Episode IV - A New Hope (1977)	68,717
Jurassic Park (1993)	64,144
Schindler's List (1993)	60,411
Braveheart (1995)	59,184
Fight Club (1999)	58,773

3. Presenting the top 5 genres the users rates based on count of ratings.

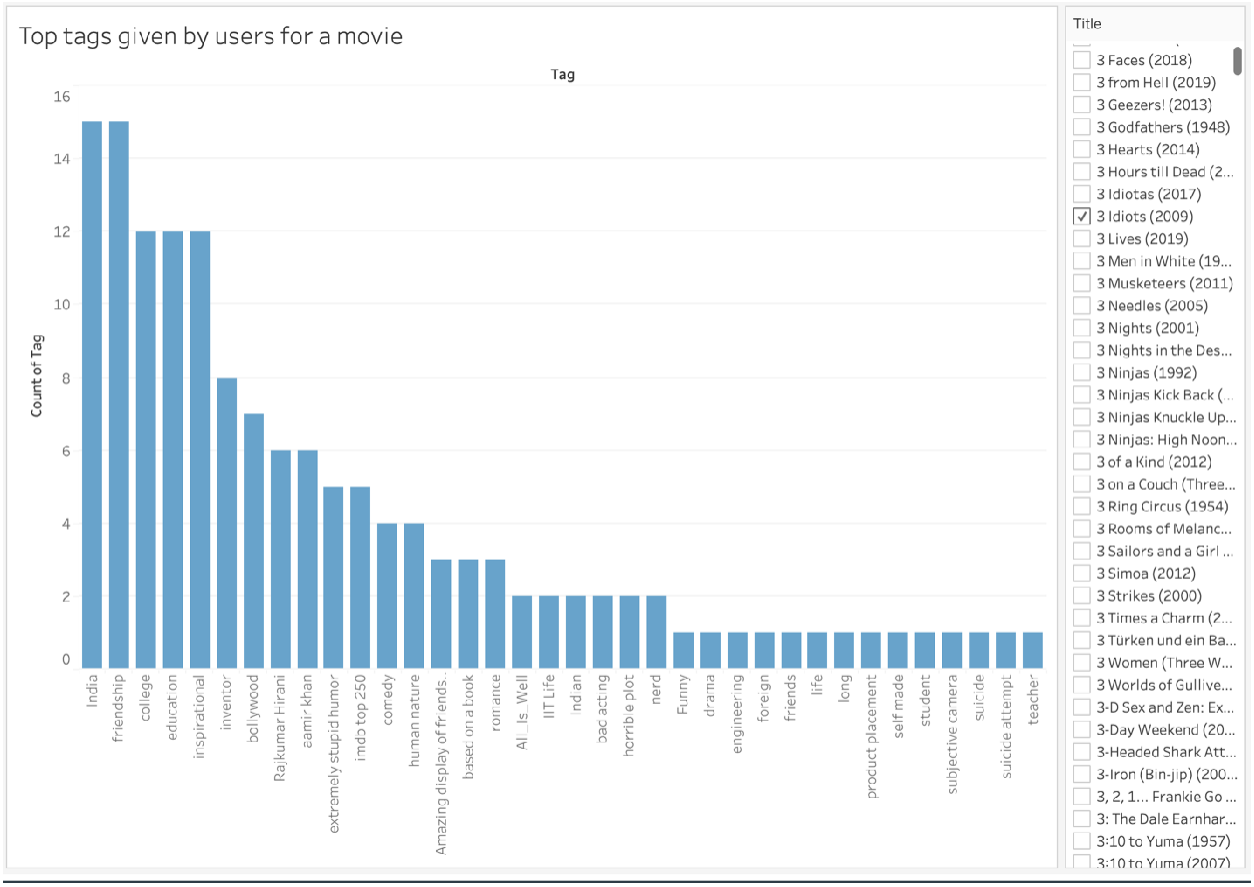
**UserID was used in filter to look at individual count of ratings provided.



4. Presenting the top 5 genres the users rates based on count of ratings.

Tags given by users for each movie can be visualized here. It considers the total number of times a particular TAG was given for a movie. For example, 3 Idiots movie was given a tag “India” by 15 users.

** A filter was provided to know individual movie tags.



Predictive Modeling:

#setting the environment variables

```
import os
os.environ["BLAS"] = "openblas"
os.environ["MKL_NUM_THREADS"] = "1"
os.environ["MKL_THREADING_LAYER"]="tbb"
os.environ["OMP_NUM_THREADS"] = "1"
os.environ["OPENBLAS_NUM_THREADS"] = "1"
os.environ["NUMBA_NUM_THREADS"]="1"
os.environ["LK_NUM_PROCS"] = "1"
os.environ["LK_THREADING_LAYER"] = "tbb"
```

#installing the libraries

```
pip install lenskit
from lenskit import batch, topn, util
from lenskit import crossfold as xf
from lenskit.algorithms import Recommender, als, item_knn as knn
from lenskit import topn
import pandas as pd
%matplotlib inline
```

#reading the data from amazon s3 bucket to jupyter notebook on amazon sagemaker

```
specific_dataset=pd.read_csv('s3://sagemaker-studio-045163058014
qwcava9tzgb/ratings.csv',nrows=100000)
```

#setting up two algorithms

Item-Item Collaborative Filtering is a type of collaborative filtering algorithm that makes recommendations based on the similarity between items. The idea is to recommend items that are like the ones a user has shown interest in or interacted with. ALS is a matrix factorization algorithm used for collaborative filtering. It's particularly useful when dealing with large sparse matrices. Item-Item Collaborative Filtering relies on the similarity between items, while ALS involves matrix factorization.

```
algo_ii = knn.ItemItem(20)
algo_als = als.BiasedMF(50)
```

Defining a function to generate recommendations from one algorithm over a single partition of the data set. It will take an algorithm, a train set, and a test set, and return the recommendations.

Note: before fitting the algorithm, we clone it. Some algorithms misbehave when fit multiple times.

Note 2: our algorithms do not necessarily implement the Recommender interface, so we adapt them. This fills in a default candidate selector.

```
def eval (aname, algo, train, test):
    fittable = util.clone(algo)
    fittable = Recommender.adapt(fittable)
    fittable.fit(train)
    users = test.user.unique()


# now we run the recommender
    recs = batch.recommend(fittable, users, 100)

# add the algorithm name for analyzability
    recs['Algorithm'] = aname
    return recs

user_col_name = 'userId'


# Rename the 'userId' column to 'user'

specific_dataset.rename(columns={user_col_name: 'user','movieId': 'item'}, inplace=True)

#looping over the data and the algorithms, and generate recommendations

all_recs = []
test_data = []
for train, test in xf.partition_users(specific_dataset[['user', 'item', 'rating']], 5,
xf.SampleFrac(0.2)):
    test_data.append(test)
    all_recs.append(eval('ItemItem', algo_ii, train, test))
    all_recs.append(eval('ALS', algo_als, train, test))

#concatenate the results into a single data frame

all_recs = pd.concat(all_recs, ignore_index=True)
all_recs.head()
```

	item	score	user	rank	Algorithm
0	4117	5.521701	2	1	ItemItem
1	114554	5.429056	2	2	ItemItem
2	7071	5.398047	2	3	ItemItem
3	6776	5.304069	2	4	ItemItem
4	45503	5.296740	2	5	ItemItem

```
test_data = pd.concat(test_data, ignore_index=True)
```

#We analyze our recommendation lists with a RecListAnalysis.

It takes care of the hard work of making sure that the truth data (our test data) and the recommendations line up properly. We do assume here that each user only appears once per algorithm. Since our crossfold method partitions users, this is fine.

```
rla = topn.RecListAnalysis()
rla.add_metric(topn.ndcg)
results = rla.compute(all_recs, test_data)
results.head()
```

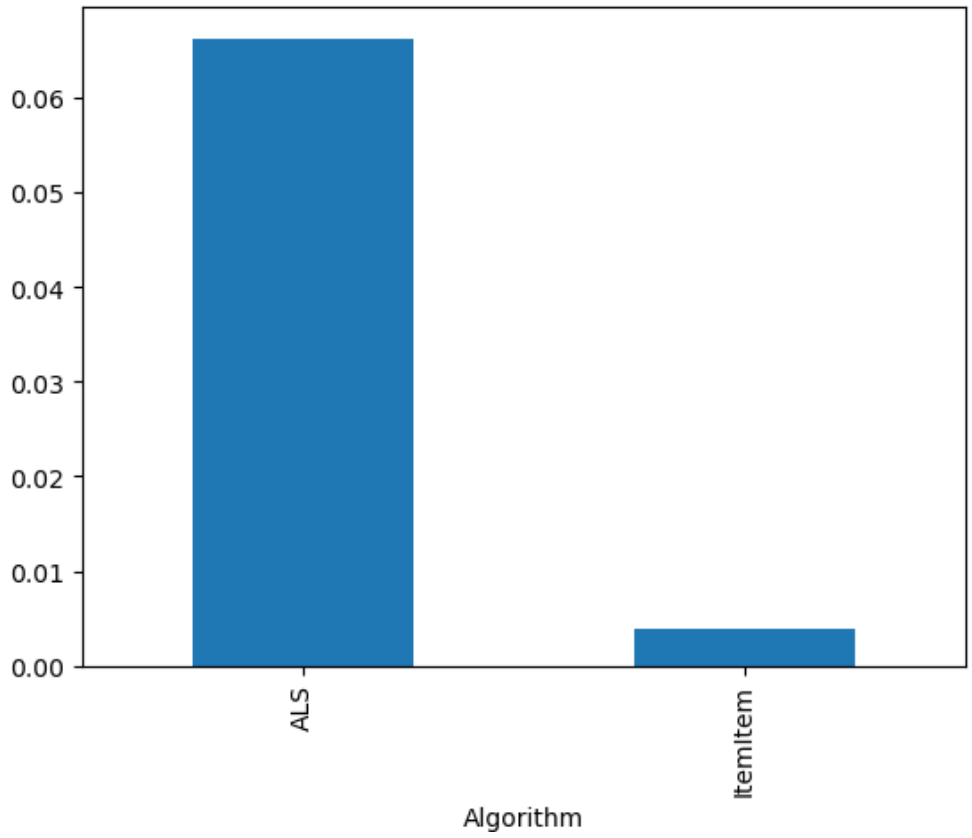
		nrecs	ndcg
Algorithm	**user**		
ItemItem	**2**	100	0.021936
	4	100	0.000000
	11	100	0.000000

		nrecs	ndcg
Algorithm	**user**		
	29	100	0.094621
	33	100	0.000000

```
results.groupby('Algorithm').ndcg.mean()
```

```
Algorithm
ALS        0.066264
ItemItem   0.003812
Name: ndcg, dtype: float64
```

```
results.groupby('Algorithm').ndcg.mean().plot.bar()
```



Normalized Discounted Cumulative Gain (nDCG) is an evaluation metric commonly used in recommendation systems to assess the quality of ranked recommendations. It considers both the relevance of items and their positions in the recommendation list. The relevance of items, whether rated or unrated, is a crucial aspect of nDCG calculation.

In the context of recommendation systems, nDCG considers both rated and unrated items in the evaluation. The algorithm's ability to accurately predict the relevance of items, whether they were explicitly rated by the user or not, contributes to the nDCG score. Higher nDCG values indicate better-quality recommendations.

Conclusion:

Right from collecting the data from different and storing them in a data warehouse by creating pipelines. Transforming the data into completely ready entity for analysis and using multiple approaches in analyzing the data and building predictive models to provide advanced recommendations, this project very much incorporates concepts of Data analysis, prediction models, Data visualization and Data Warehousing concepts in order to build a robust recommendation system. Hence, looking the present trend of data proliferation and the rising reliance on data-driven decision-making, data warehouses are critical.

References:

[1] https://grouplens.org/datasets/movielens/
[2] https://lenskit.org/research