1. Project details: Name of your project, your team, and team members.

Project Name: ***Movie Recommendations System***

- **Edward Kim (eykim5): eykim5@buffalo.edu**
- **Mohammed Samsuddin (Mshmsudd): mshmsudd@buffalo.edu**
- **Cory Chaise (Corychai): corychai@buffalo.edu**

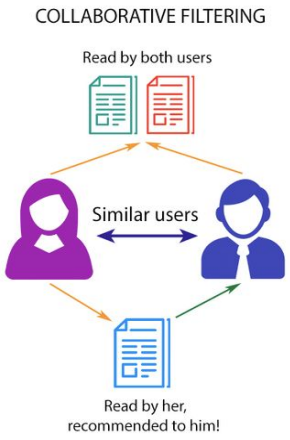
2. Problem Statement: What problem are you tackling, and what's the setting you're considering?

Problem: Improved Movie Recommendations System based on Collaborative Filtering

Entertainment services such Netflix, Amazon Prime, or Voot often utilize recommendation algorithms to suggest new TV shows and films to their customers. These systems are capable of analyzing users' watch history and providing them with relevant suggestions. The recommendation system is classified into two types: A **Content-Based** or **Collaborative-Based** recommendation system. For our project, we will implement a collaborative-based movie recommendation system. The theory behind collaborative filtering involves working with user or movie ID. For example, suppose there are two users, **A** and **B**

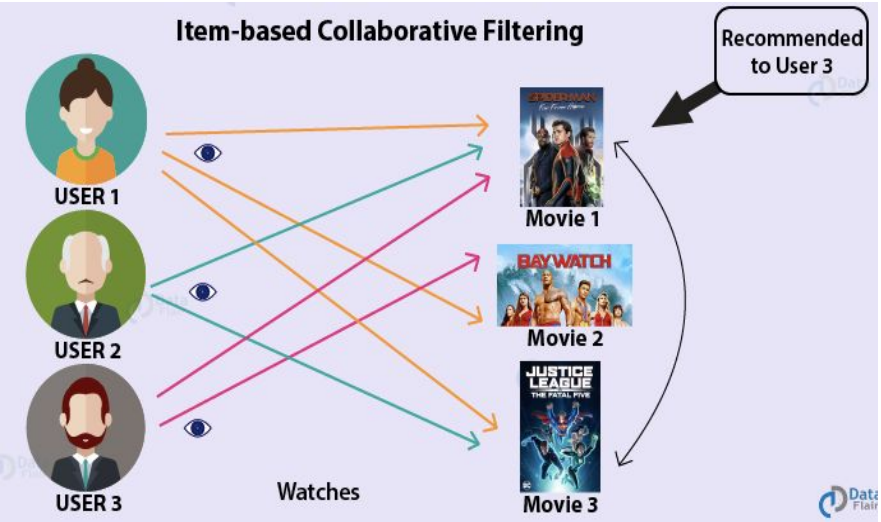
- User **A** likes movie **P, Q, R, S**
- User **B** likes movie **Q, R, S, T**

Since **A** and **B** both like **Q, R,** and **S**, the system will recommend **P** to user **B,** and **T** to user **A**.



There are two types of Collaborative filtering Algorithm: **User-based filtering** and **Item-based filtering**. We will use an Item-based filtering algorithm for our system. This algorithm finds the Movie look-alike. It picks from your previous list of 50-100 movies that share similar peoples with

the targeted movie, how much you will like targeted movies depends on how much the others liked those earlier movies. So, you tend to like this movie because you have liked those movies.



Item-based filtering algorithm is far less resource consuming than User-based filtering. For a new user this algorithm also takes far less time because we don't need all similarity scores between users. Once we have a movie look-alike Matrix, we can easily recommend similar movies to users who have rated the movie from the dataset.

3. Method: What machine learning techniques (definitely you should use some of the techniques that you are learning in class. However, you can also show results with other methods that you may feel would be appropriate for your problem) have you tried and why?

Method:
K-nearest neighbors (KNN): A simple Supervised ML algorithm. KNN works by finding the distances between a query and all the examples in the data, selecting the specified number examples (K) closest to the query, then votes for the most frequent label (in the case of classification) or averages the labels (in the case of regression).

In implementing KNN, the first step is to transform data points into feature vectors, or their mathematical value. The algorithm then works by finding the distance between the mathematical values of these points. The most common way to find this distance is the Euclidean distance, which we will be using here.

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

KNN runs this formula to compute the distance between each data point and the test data. It then finds the probability of these points being similar to the test data and classifies it based on which points share the highest probabilities.

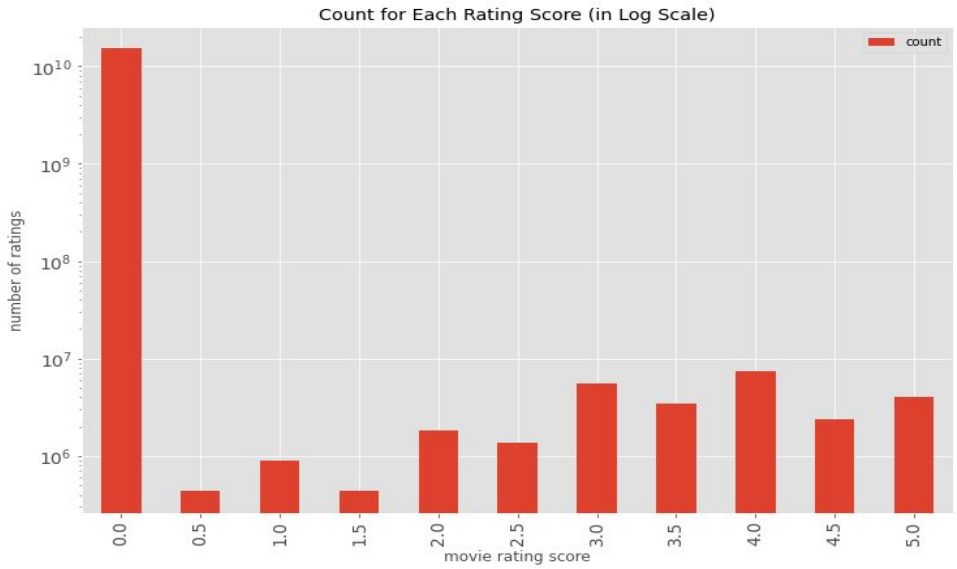
4. Preliminary Results: Results in the state of the art methods and your preliminary achievements

K-nearest neighbors (KNN):

1. Exploratory Data Analysis: We used movies.csv, tags.csv, and ratings.csv for our KNN.
    - Plot the counts of each rating: Using ratings.csv we plotted count of each rating in Log because zero rating count is simply too big to compare with others.

        1. Rating Count in Log and Linear:

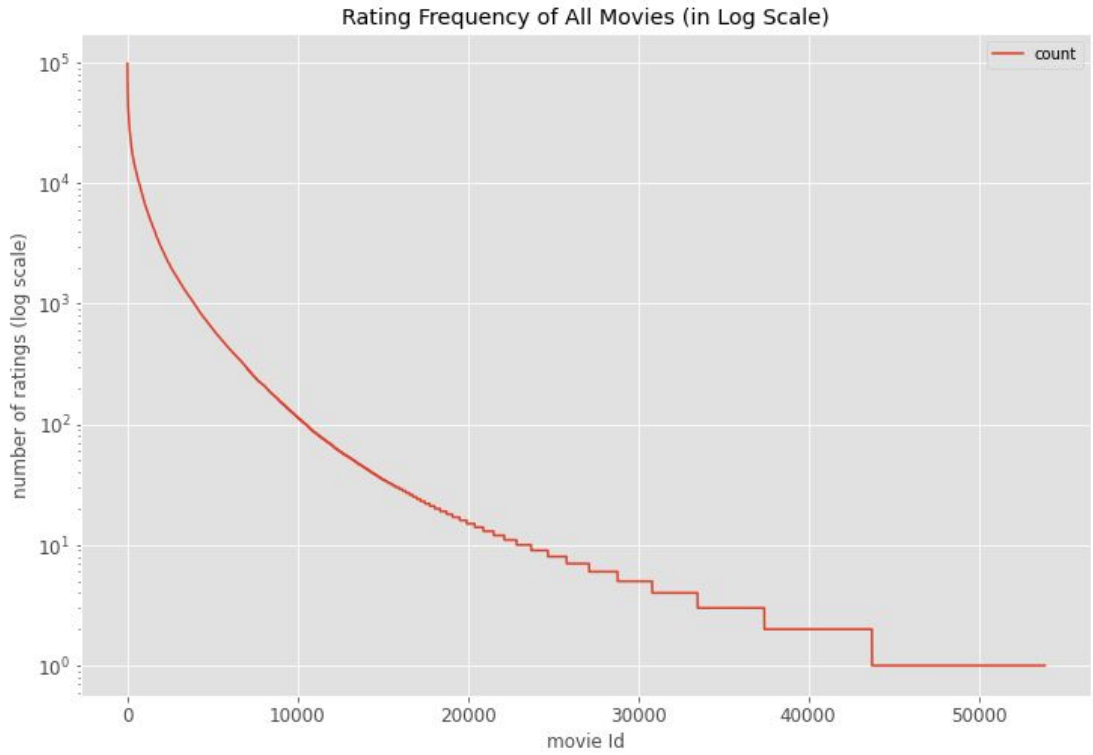
	count	log_count
0.0	15235120248	23.446869
0.5	442388	12.999943
1.0	886233	13.694735
1.5	441354	12.997603
2.0	1850627	14.431035
2.5	1373419	14.132814
3.0	5515668	15.523103
3.5	3404360	15.040568
4.0	7394710	15.816275
4.5	2373550	14.679897
5.0	4071135	15.219432



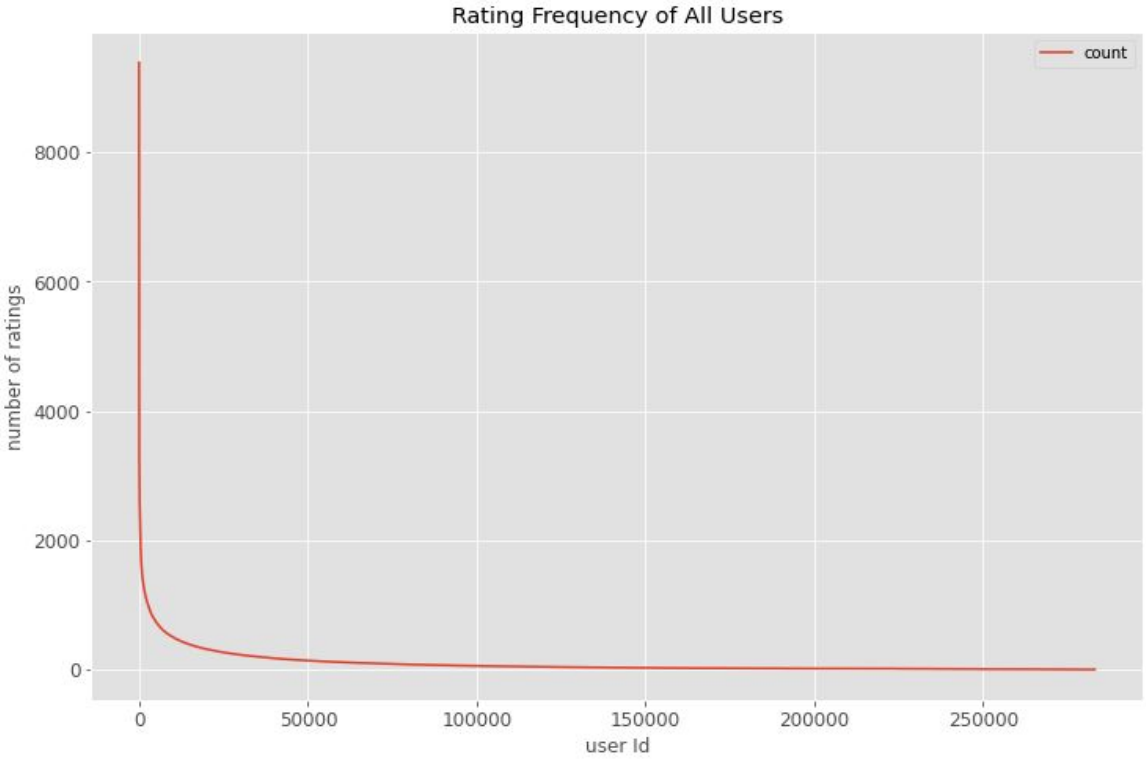
Based on the Graph we can see people are giving ratings of 4 more than others. And 1.5 is the least popular rating score.

2. Plot rating frequency of all movies:

movieId	count	log_count
1	68469	11.134136
2	27143	10.208874
3	15585	9.654064
4	2989	8.002694
5	15474	9.646916
...	...	...
193876	1	0.000000
193878	1	0.000000
193880	1	0.000000
193882	1	0.000000
193886	2	0.693147



We can see that roughly 10,000 out of 53,889 movies are rated more than 100 times. More interestingly, roughly 20,000 out of 53,889 movies are rated less than only 10 times.



We can see that the distribution of ratings by users is very similar to the distribution of ratings among movies. They both have long-tail property. Only a very small fraction of users are very actively engaged with rating movies that they watched. Vast majority of users aren't interested in rating movies. So we can limit users to the top 40%, which is about 113,291 users. And drop 75% of movies in our dataset. We still have quite a large data set for our KNN.

2. Train KNN model for item-based collaborative filtering:
    - Reshaping the Data:
      We convert our table to a 2D matrix, where each row is a movie and each column is a different user. And fill the missing values with zeros (since we will calculate distances between rating vectors).

userId	4	5	10	14	15	18	19	26	31	34	35	36	38	39	42	43	45
movieId																	
1	4.0	0.0	5.0	4.5	4.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	5.0	0.0	4.0	5.0	0.0
2	4.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.5	3.0	0.0	0.5
3	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
189363	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
189713	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
191351	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
191799	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
192283	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

We then transform the values(ratings) of the matrix data frame into a scipy sparse matrix for more efficient calculations.

```
(0, 0)        4.0
(0, 2)        5.0
(0, 3)        4.5
(0, 4)        4.0
(0, 8)        5.0
(0, 12)       5.0
(0, 14)       4.0
(0, 15)       5.0
(0, 20)       4.0
(0, 22)       5.0
(0, 23)       2.5
(0, 26)       4.0
(0, 27)       4.0
(0, 28)       4.0
(0, 30)       4.0
(0, 33)       5.0
(0, 40)       4.5
(0, 41)       4.0
(0, 43)       5.0
(0, 44)       5.0
(0, 47)       4.0
(0, 50)       5.0
(0, 51)       5.0
(0, 53)       3.5
(0, 56)       3.0
:     :
```

- Fitting the Data:
  We use unsupervised algorithms with sklearn.neighbors. The algorithm we use to compute the nearest neighbors is “brute”, and we specify “metric=cosine”, and nearest neighbor “n_neighbor = 20” so that the algorithm will calculate the cosine similarity between rating vectors. Finally, we fit the model.

3. Use this trained model to make Movie Recommendations:
   The KNN algorithm measures distance to determine the “closeness” of instances. It then classifies an instance by finding its nearest neighbors, and picks the most popular class among the neighbors. The code for the Movie Recommendation system is provided. After closely inspecting our data, we found out that 98.35% of entries in our data is zero. This explains why the distance between similar items or opposite items are both pretty large.

5. Dataset: What are the datasets that you have used, and why did you decide to choose them?

Currently we are using MovieLens Dataset: https://grouplens.org/datasets/movielens/latest/
It contains 27,000,000 ratings and 1,100,000 tag applications across 58,000 movies. These data were created by 280,000 users between January 09, 1995 and September, 2018. This dataset was generated on September 26, 2018.

The data that are contained in the files are: genome-scores.csv, genome-tags.csv, links.csv, movies.csv, ratings.csv and tags.csv.

6. Plan : Given your preliminary results, what are the next steps that you're considering? How to submit Save your report as Milestone1.pdf and upload to UBLearn. Only the team leader (Member 1) needs to submit the file.

So far we have completed K-nearest neighbors (KNN). The problem with KNN is that the vast majority of our rating entries are zero. With too many zero values in our data, the data sparsity becomes a real issue for KNN models and the distance in KNN models starts to fall apart. We are looking forward to finishing UI for our WebApp by the end of the semester.

Team Members Contribution:
Mohammed Samsuddin: I wrote the Method and Preliminary results of this document and Also Designed and implemented the KNN model in python.

Edward: Researched the initial topic for the dataset. Edited the Milestone Report, recorded demo video as well as some additional writing for the Results section.

Cory: Researched topics for project subjects and found datasets to use. Helped come up with the problem statement; contributed to the written report. Created frontend website for project.

Github Repository Link:

References:
https://towardsdatascience.com/how-did-we-build-book-recommender-systems-in-an-hour-part-2-k-nearest-neighbors-and-matrix-c04b3c2ef55c
https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26

https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-1-knn-item-based-collaborative-filtering-637969614ea

https://medium.com/capital-one-tech/k-nearest-neighbors-knn-algorithm-for-machine-learning-e883219c8f26