In [1]:
```
#import relevant packages
from IPython.display import Image
```

Fictional Context For My Data Science Project as a Springboard Data Science Fellow for my Capstone 3 Project as of November 2nd, 2022 (2022-11-02)

Recommendation System For Movie Streaming Service

1) Problem Formulation & Data Source

A new movie streaming service is looking to increase the amount of time subscribers spend on their service by implementing a recommendation system that can providing new movies for users to watch. Their primary purpose is to increase time spent on the platform as this has been shown previously to both decrease the churn rate and be a key KPI to present to investors. The clients primary concern is to show a prototype model that can provide a RMSE below 1.5 in order to justify further investment and development of a more powerful model.

The data set we will be using is currently locked by the sponsor, but we are free to consider any models we wish. One note is that the sponsor wants to avoid using (or collecting) any personal data about the user.

The data source for our project can be found here as provided by the project sponsor. The data set is relatively clean but does require some data wrangling to consider before we can move forward to modeling.

2) Data Wrangling

For the full workthrough, please check the following notebook here

The data set has 100,002 observations for us to work through with 944 unique users in the dataset rating 1,682. We were given two tables. One providing rating information and the other provided movie title information. The primary features given were:

- user_id: The associated ID given to a unique user
- move_id: The associated ID given to a unique movie
- rating: The rating given by a user to a specific movie (done on scale of 1 (low) to 5 (high))

- timestamp: the Unix epoch timestamp for when a rating was provided

The main data wrangling & cleaning procedures that were utilized:

- Combine the data into a single DataFrame that included the names of the movie
- Exporting the final DataFrame as a unified data set going forward

Note on outliers: For a data set like this where all ratings are on a very condensed scale of 1 to 5, it's difficult to really assess what is an "outlier." This means we'll have to keep on eye on outliers throughout the process as we explore the dataset but they would be nearly as identifiable as they typically would be with a more traditional data set

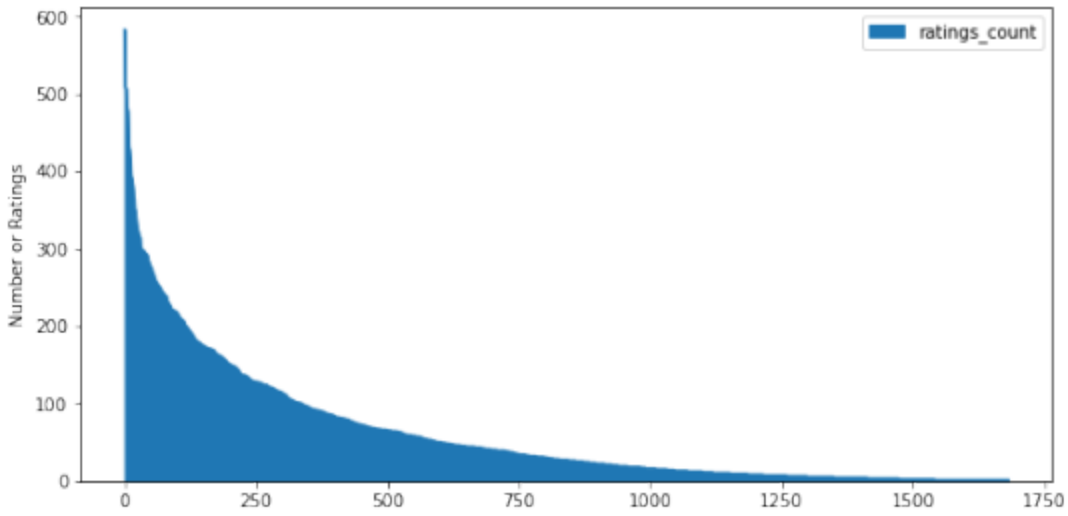
3) Exploratory Data Analysis

For the full-workthrough, please check the following notebook here

Once our was wrangling an put into a proper form, we began exploring it to get a better sense of it and to develop considerations for modeling. The first aspect to consider is to get a sense of how many ratings we have for each movie

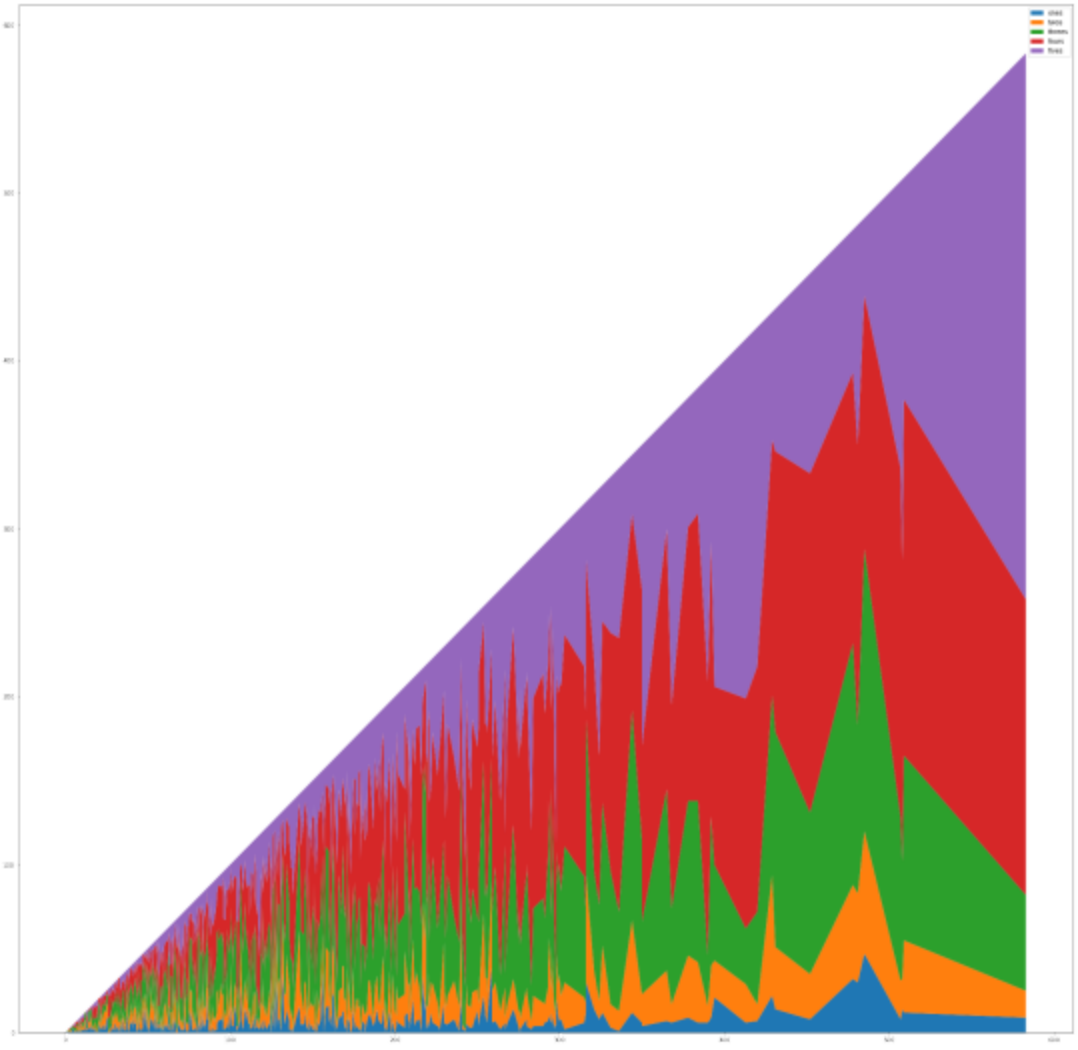
Ratings Count Grouped By Movie ID			
Count:	1,682.00	25%:	6.00
Mean:	59.45	50%:	27.00
STD:	80.39	75%:	80.00
Min:	1.00		
Max:	583.00		

This table provides our first consideration regarding the number of ratings per movie. Since the 25% percentile is a 6 (meaning 25% of the movies in the database have 6 or less ratings), it may be difficult to simply only use movies with "a high number" of ratings. As such, we'll need to consider how we can have a functioning model that can consider movies with a smaller number of ratings. We can also visualize this data to get a better sense of the distribution with number of ratings on the y-axis.



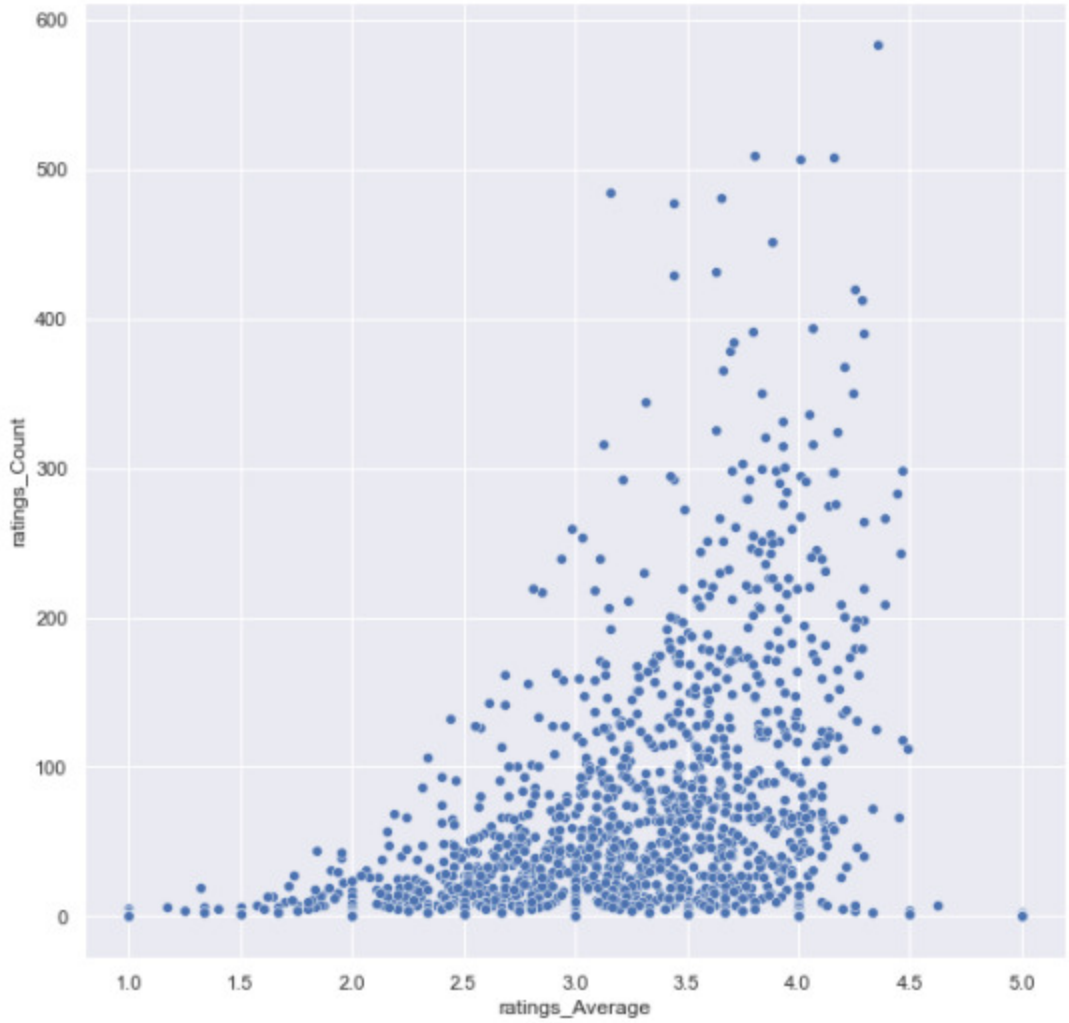
As a note, 141 of the movies have only a single rating and 68 of the movies only have two ratings.

We can also get more detailed with our ratings visualizaiton and consider how many of "each" rating (1,2,3,4 or 5) a specific movie receieved



From this plot, we can see that it appears that the ratings proportionally grow in terms of the different types. It's only once you really hit the 400+ ratings count mark that you see some shifts (for instance, the red 4's takes a dive and there is a basket of 5's) and then once you get above 500+ ratings, we see that 5 ratings become the greater majority. This makes sense as "positive buzz" about a movie may encourage a momentum effect where people positively rate and speak about a movie which may cause more people to see the movie, who in turn, positively rate and speak about the movie in a self-amplifying cycle.

We can get another summarization of the data from a scatterplot that plots the "average" rating of a movie against the number of ratings for a given movie



From this scatterplot, we can see that there is a moderate direct correlation between the average rating of a movie and the number of ratings a movie receives which further supports our "positive buzz momentum" observation previously. One thing to know however is that both the extremes (1's and 4.5+'s) seem to suffer from a "small population" distoration issue where there just aren't many ratings. For the low ratings, if a movie gets some initial bad ratings, this may push it into "movie purgatory" where people may not watch it (which suggests that our sponsor may want to be consider the number of ratings a movie receives before making the information public). For the high ratings, this may be that they are just "starting" the positive buzz process and yet to fully blossom yet.

In terms of summary key findings:

- There is a correlation between the number of ratings and the average rating of a movie
- We have a fair amount of movies with a "low" number of ratings so we'll need to consider how to adjust for that

4) Preprocessing

For the full workthrough, please check the following notebook here

Scaling & Normalization

Since our data is in integer from ranging from 1 to 5, it has been effectively scaled and we can move forward data set splitting between training and testing sets.

Training & Testing Data Set Creation

When it comes to recommendation systems, doing a stratified sample based on the movie is important to ensure that you have enough training observations to be able to generate a meaningful model. For this problem, we attempt to stick to a 80/20 training/testing split as "closely" as possible. As such, we use the following strategy:

- Any movie with only one or two ratings is dropped from the set
- Any movie with only three ratings is split into two training observations and one testing observation at random
- Any movie with only four ratings is split into three training observations and one testing observation at random
- Any movie with five or more ratings is approached as follows:
    - For each multiple of five, four of the observations are training obseravations and one observations is a testing observation
    - For the remainder (e.g., 27 divided by 5 has a remainder of two), we approach it as follows: remainder of 1 is placed in training set, remainder of 2 is split one in training and one in testing, remainder of 3 is split two for training and one for testing, and remainder of 4 is split two for training and two for testing

After splitting our data, we get 79,223 training observations and 20,502 testing observations, which is a 79.4% / 20.6% split in terms of training/testing which is fairly close to the 80/20 split we were targeting.

5) Modeling

For the full workthrough, please check the following notebook here

Performance Criteria

We use the Root Mean Squared Error (RMSE) metric as our method of evaluating model:

RMSE: $\sqrt{\frac{\sum(\hat{y}-y)^2}{n}}$

This means that the worst score our model can get is a "4" which gives us a frame of reference for "absolute" worst model.

For the purposes of developing a prototype, we will examine four main types of estimators:

- A naive estimator that simply predicts "3" for every movie
- An estimator that takes the unweighted average of a movie from the training set
- An estimator that takes a weighted average based on euclidean similiarity of each user against the test user (based on a movie)
- An estimator that takes a weighted average based on cosine similiarity of each user against the test user (based on a movie)

Naive Estimator of "3" for every movie

We begin our analysis by simply predicting a rating of "3" for every test observation (regardless of movie and user) for setting a baseline of estimate quality. We use the value of "3" because it is the median value of the possible ratings and generally will split the difference between the "best" and "worst" values. As such, it technically would set that our "worst" RMSE using the value of "3" should be 2 at worst (assuming all test value ratings are either 5 or 1). In addition, this is a relatively simple estimator to implement.

Once implemented, the naive estimator, we estimated a RMSE of 1.2401 which is approximately 62% of the maximum error possible using this estimator. This implicitly suggests that if every movie's "true" rating were

4.201 and you guessed "3", you'd get the same result. Finally, this sets what should be considered the "true" baseline (as opposed to "4") as it gives us a better understanding of what a very simply estimator can provide

Estimator of unweighted average across users in the training set for a given movie

Our first estimator beyond a naive estimator will be use the observations in the training set and to simply average the ratings of users for a given movie. The nice thing about this estimator is that it is quite simple but it comes with the inherit downside of treating each testing user the same (regardless of their rating profile).

Once implemented, the unweighted average estimator results in a RMSE of 1.0300, which is technically a 16.94% decrease relative to the naive estimator's RMSE which is a substantial improvement.

Estimator of weighted average across users in the training set for a given movie using Euclidean similarity

Euclidean similarity is given by the following formula:

$$sim(x, y) = \frac{1}{1 + \sqrt{\sum(x - y)^2}}$$

If two users have *exactly* the same profile, it will take on a maximum value of one and will decrease asympoticially towards zero as two users have differing profiles. Profiles in this case are the basket of ratings that users have in relation to each other. In the event that there is no rating for a specific movie, the rating is treated as a "zero" in order to faciliate calculation. One important point is that in order to do a similarity calculation, the complexity of implementation increases substantially because each training user must be compared against the test case user's movie rating.

Once implemented, the Euclidean similarity estimator gives us an RMSE of 1.0295, which is not nearly as stark an improvement (merely a 0.05% decrease) *but* is still an improvement and when it comes to a model that has a very narrow range for improvement, even the slightest improvement needs to be considered (especially if it still results in increased viewing time and/or reduced churn rate). We test one more measure of similarity for prototyping purposes.

Estimator of weighted average across users in the training set for a given movie using Cosine similarity

Cosine similarity is given by the following formula:

$$sim(x, y) = \frac{(x . y)}{\sqrt{(x . x)(y . y)}}$$

This gives us a different type of similarity measure based on the cocept of the cosine of a geometric shape. One thing to highlight for this measure is this measure is that it does have edge cases where it is undefined, which typically is the result of user only rating the movie in question and no other movies to build a similarity profile. Since we treat unrated movies as a zero, this can "break" the measure. In the event that the Cosine similarity would be undefined, we fall back on the average unweighted rating as the prediction. This highlights one of the issues with similarity measures in that they need to be robust enough to deal with edge cases.

Once implemented, the Cosine similarity estimator gives us an RMSE of 1.0355, which is technically worse than the RMSE we acheived, which means will stick with the Euclidean measure for now for the purposes of the prototype.

6) Conclusion & Next Steps

Conculsion

Given the charge of the sponsor, we were able to develop a prototype model that meet the KPI threshold. While we were capable of doing this with a simple naive guess, we improved upon this with by using a Euclidean Similarity Weighting Scheme to develop predictive ratings. Given the time restrictions of the project, we weren't able to conduct more analysis but this represents a good milestone to consider further development.

Next steps

The next step will be to build upon this success and consider some of the efficiencies we can consider which include:

- Testing other measures of similarity such as Manhattan or Jacobian similarity to see if we can decrease RMSE even further
- Consider increasing efficiency by seperating the computation arrays necessary for calculation from the actual similarity calculation. The first step is shared and then the second step is only specific to the similarity which can help speed up calculations
- Consider developing ensemble methods that blend methods we have developed (and potentially other methods) to increase RMSE
- Think about using an API tied to a database like IMDB to fill in more information about a movie to see if we can use defining features about a movie to help make our predictions more accurate (e.g., genre, release date, etc.) It's possible the company may have this information and we can have a discussion regarding incorporating it into their database
- Dicuss further methods of testing the effacacy of a model since the biggest goal is to increase viewership time, which isn't necessary captured by our current methodology. This may require some discussion as to how to implement and test

Final thoughts

This was definitely an interesting project to work on and I learned alot about setting up a proper Data Science Pipeline for analyzing and developing models. It especially highlighted how important it is to think as detailed and precisely as possible about how to establish efficient testing and modeling to be able to evaluate as many options as possible. As always, thank you everyone for helping me with the project! Cheers! Emre

In [ ]: