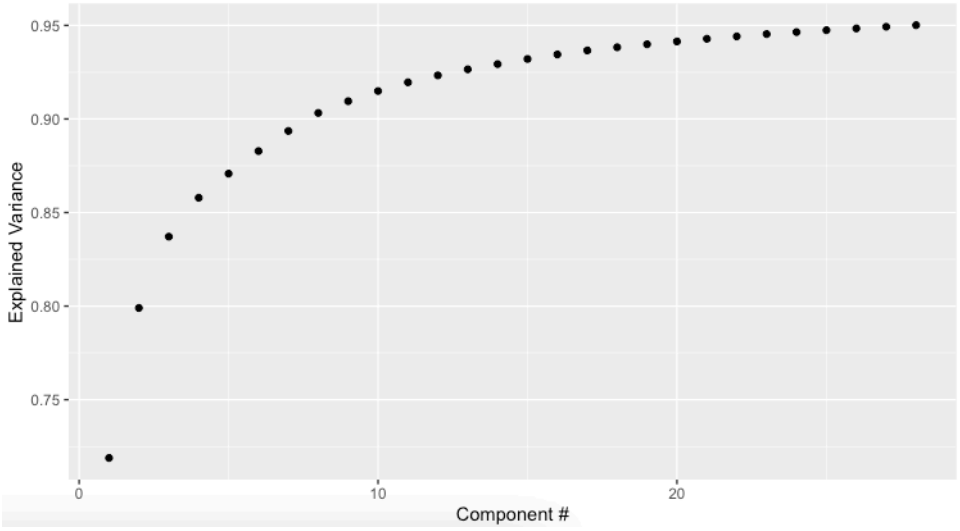
Using collaborative and content filtering methods has proven to be one of the most accurate ways to create a recommendation system, as proven in Netflix's recommendation system competition. However, when considering such a large dataset, and the fact that users will not buy most of the products that are available, a major problem is that our dataset will be incredibly sparse with many missing values. This would lead to an overfitting, and simply inaccurate model for predicting user purchases.
To overcome these problems, I will apply dimensionality reduction using the matrix factorization method SVD. For this project, I plan to use k-fold cross validation and use the RMSE to determine the accuracy. After calculating the error, I also plan to apply my own stochastic gradient descent algorithm from scratch that will run on the resulting error function in hopes that I can minimize it.

Data Wrangling:
I first obtained a dataset from Instacart, a grocery delivery service, that contains information on products ordered across 3 million unique orders by customers. This dataset included the product names/ids, order ids, and whether or not it was reordered. For each item to item relationship, I assigned a score calculated from whether the two items had been purchased together before and whether they were also reordered. This was done by grouping the total orders by the order id, calculating the score for each pair in the order, and then combining all the results for each order. At the end, any duplicate relationships from separate orders were combined together, leaving me with the total score for each relationship across all orders. I also removed the most uncommonly purchased items by removing rows for items that only appeared a single time. Since my data was still extremely skewed to the right, I then scaled the values between 0 and 100 and took the logarithmic function for each value to reduce the skewing effect. Finally, these ratings were then rescaled back to 0 and 100 to represent the final score.

SVD:
The first step I did was to calculate the number of components I wanted to keep to determine the dimensionality of my output data. To do this, I plotted the explained variance ratio for each number of components until the variance reached my set threshold of 95% . The explained variance ratio gives us the amount of variance attributed by each component, and by calculating

the sum, we get the total amount of explained variance for each number of components.



In this case, I chose 28 components, which would give me approximately 95% of explained variance.

For testing, I decided to run k-fold cross validation across 3 folds so I would not have to initially split my data into a training and testing set. Instead I can run cross validation multiple times with different combinations of a training and test split and get a final average of an RMSE that isn't as biased. The results show both the MSE and RMSE for each fold.

```
Fold 1
{'rmse': 0.9959928071014337, 'mse': 0.9920016717977936}
Fold 2
{'rmse': 1.0026745593648374, 'mse': 1.0053562719974707}
Fold 3
{'rmse': 1.0201294347723628, 'mse': 1.0406640636889803}
```

The average RMSE across these three folds comes out to be about 1.006.

Stochastic Gradient Descent:

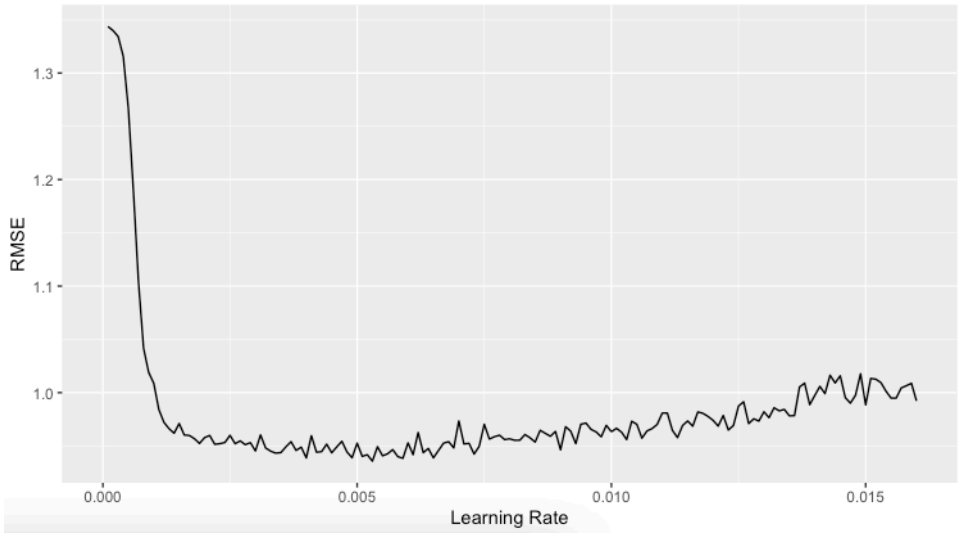
To improve upon my calculated error, I implemented a stochastic gradient descent algorithm to help reduce the error. I started out with a learning rate of 0.01 and set the number of epochs to 20, to see how the resulting error would behave.

```
Fold 1
{'rmse': 0.9754951912374052, 'mse': 0.9515908681273019}
Fold 2
{'rmse': 0.9715645460788228, 'mse': 0.9439376671973491}
Fold 3
{'rmse': 0.9483642305012011, 'mse': 0.8993947136941353}
```

When calculating the RMSE values across my cross validation with the stochastic gradient descent algorithm, the results show that the algorithm did indeed help to improve my error values slightly. The average error with this method is about 0.965.

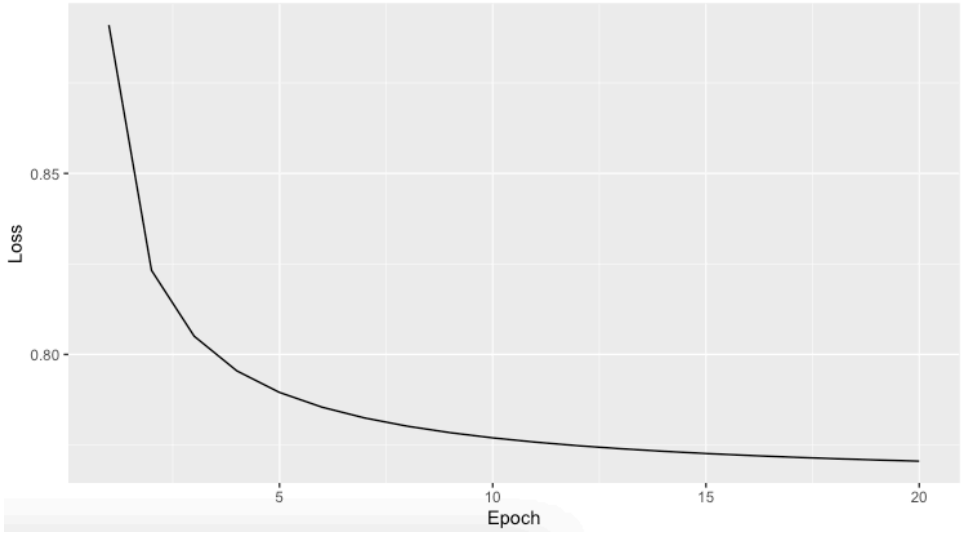
Tuning:

To experiment with different learning rates, I also calculated the different average RMSE values for various learning rates. This plot shows how the resulting RMSE changes based on a variable learning rate with epochs of 20. Too small of a learning rate would not be able to change the weights quickly enough, and too high of a learning rate meant it would not be able to reach the minimum.



I decided to choose a learning rate of 0.006 because it approximately results in the lowest average RMSE for my algorithm.

With a 0.006 learning rate, we can see how the loss approaches a minimum as the epochs pass. This plot tracks the loss across 20 epochs for a randomly selected cell in the matrix, calculated by the difference between the actual score and the current prediction.

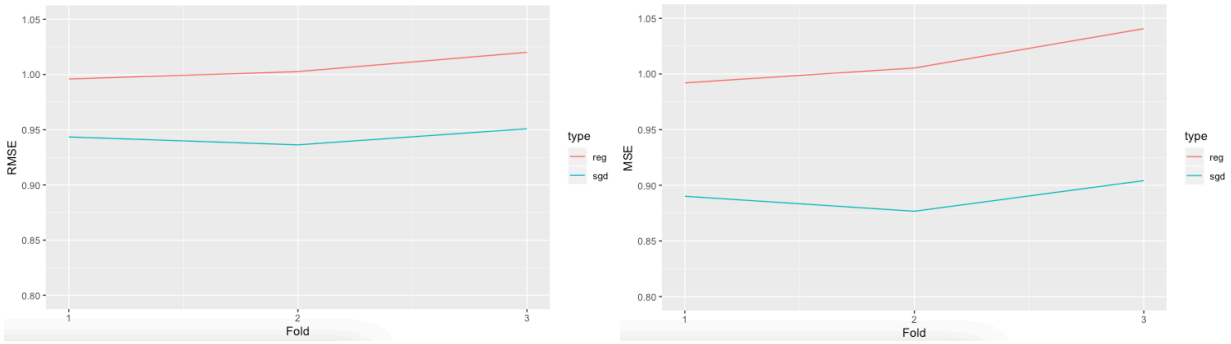


Results:
The RMSE values for a learning rate of 0.006 are shown below.

```
Fold 1
{'rmse': 0.9434727679155506, 'mse': 0.8901408637982303}
Fold 2
{'rmse': 0.9363185810655914, 'mse': 0.8766924852486824}
Fold 3
{'rmse': 0.9508962551259426, 'mse': 0.9042036880125416}
```

The average RMSE across the folds comes out to be 0.943.

I created two plots showing the error rates across the five folds for both RMSE and MSE. We can see that when using stochastic gradient descent, the average RMSE drops by about 0.05, while the MSE drops by about 0.13. The blue line represents the RMSE with stochastic gradient descent and the red line represents without.



I can then easily predict which products are most similar to each other by reconstructing a matrix similar to my original, instead filled with all the predicted ratings. A simple search for an item would then be able to return the predictions for that item.

For example, for an input of ‘Banana’, I printed the ten items with the highest predicted ratings.

```
('Organic Baby Spinach', 74.02328630754995)
('Organic Strawberries', 72.96793996937267)
('Organic Avocado', 67.18199690935198)
('Large Lemon', 50.8044932870653)
('Organic Hass Avocado', 42.62190628050047)
('Limes', 42.239628934759075)
('Cucumber Kirby', 40.38067987549162)
('Bag of Organic Bananas', 36.98234593849853)
('Organic Fuji Apple', 35.426625271599974)
('Honeycrisp Apple', 34.42962055414525)
```

From these ten items, the ratings seem quite reasonable, since it would make sense that with a purchase of a banana, the ten highest predicted ratings would also be fruits or vegetables.

Challenges:

One of the challenges I faced was determining what formula I would use to calculate the ‘score’ for each item pairing. Given the available data, it seemed that the most reasonable method would be to rate each pair on how many times they had been purchased together. However, this meant that some item pairs only had 2 as a score and some item pairs had scores in the hundreds. To

combat this problem I decided to first scale the values between 0 and 100, take the logarithmic function of each score, and then rescale back to 0 and 100. This helped to alleviate some of the skewing effects my original data had. Another challenge was the size of the dataset. Since there were over 3 million rows of data I had to parse through, it would take a long time to run. I was able to solve this problem by utilizing the computer labs on campus, which made my data processing part of the project much easier.

Tools:

Most of my code for this project was written in Python, with the help of libraries such as pandas and numpy to help with the data. I also used the sci-kit learn library to help with calculating the SVD part of the project. For the plots, I was originally going to use matplotlib in Python, but I decided to use R as well as the ggplot2 library for more versatility.