Capstone 3: Final Report

Problem Statement and Context

Using a database containing user ratings of recipes and the associated recipe database construct a model that can provide relevant recipe recommendations personalized for each user. These should be recipes that the user has not yet cooked.

Recommendation systems are essential for many online businesses. A well-designed, personalized recommendation system can increase customer engagement, improve satisfaction, and ultimately drive higher revenues. From streaming services to online retailers, effective recommendations contribute significantly to business success. On the other hand, an inaccurate system can harm customer satisfaction, potentially reducing engagement and revenue.

In this project, I will construct a recommendation system to deliver personalized recipe recommendations to users. These recommendations will be based on users' previous interactions with recipes, along with the ratings they provided for recipes they reviewed.

Data Wrangling

The data consists of two datasets:

1. The **recipe dataset** contains information regarding each of the 231,637 recipes. There is one row per recipe. The dataset contains the following features:
    a. **Recipe name:** the name of the recipe
    b. **Recipe ID:** a unique identifier for each recipe
    c. **Minutes:** the estimated number of minutes to execute the recipe
    d. **Contributor ID:** the id of the person who contributed the recipe
    e. **Submitted:** the date the recipe was submitted
    f. **Tags:** generic tags associated with the recipe (e.g., 15 minutes or less)
    g. **Nutrition:** nutrition information for each recipe. This contain calories, total fat, sugar, sodium, protein, saturated fat, and carbohydrates
    h. **N_steps:** the number of steps in the recipe
    i. **Steps:** text of the actual steps of the recipe. I.e., the directions
    j. **Description:** a short description of the recipe
    k. **Ingredients:** a list of the ingredients needed for the recipe
    l. **N_ingredients:** the number of ingredients in the recipe

2. The user **interaction dataset** contains user reviews and ratings for the recipes contained in the recipe dataset. The following features are in the dataset:

a. **User_id:** the unique identifier for each user. This column contains duplicate values since a user can write multiple reviews
b. **Recipe_id:** the id of the recipe that the user is reviewing/rating. This recipe_id matches the id from the recipe dataset
c. **Date:** the date which the user submitted the recipe review
d. **Rating:** the rating the user gave the recipe
e. **Review:** full text of the user review of the recipe

The dataset was pretty clean, but still required some action to deal with some null values. One recipe name was missing which I filled in manually based on the recipe description. Other null values were in the full text fields – the recipe description and the user review of the recipe (e.g., the user only submitted a rating but not a full text review). The total amount of null values for the full text fields were ~200 for the review and ~5,000 for the description. The fact that these full text fields have null values is not of concern. If I use these features I will fill them in with a standard string to indicate they weren't filled in.

Exploratory Data Analysis

The recipe dataframe underwent the following exploratory analysis:

1. Confirmed that there were no duplicate values for the ids
2. Computed the summary statistics for the numerical features:

	count	mean	std	min	25%	50%	75%	max
id	231637.0	2.220147e+05	1.412066e+05	38.0	99944.0	207249.0	333816.0	5.377160e+05
minutes	231637.0	9.398546e+03	4.461963e+06	0.0	20.0	40.0	65.0	2.147484e+09
contributor_id	231637.0	5.534885e+06	9.979141e+07	27.0	56905.0	173614.0	398275.0	2.002290e+09
n_steps	231637.0	9.765499e+00	5.995128e+00	0.0	6.0	9.0	12.0	1.450000e+02
n_ingredients	231637.0	9.051153e+00	3.734796e+00	1.0	6.0	9.0	11.0	4.300000e+01

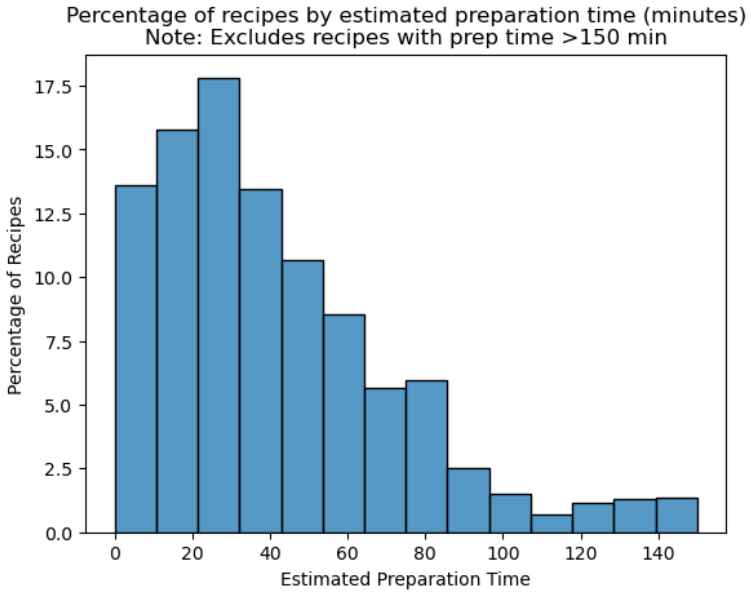
a. From the above summary statistics, a few things stand out:
    i. Minutes: the maximum minutes is 2e+09, this seems to be an error and will have to be investigated and fixed
        1. I modified some of the outliers here to correct inaccurate fields. For example, the recipe with the max number of minutes (id # 261647) has a tag that indicates the recipe should take 60 minutes or less, so I adjusted minutes to 60.

2. There are over 2000 recipes whose prep time takes over 800 minutes. Looking at recipes that take over 100,000 minutes and reading through some of their descriptions above, it seems like the minutes are actually accurate. A lot of these recipes require extended waiting times, especially for liquor recipes. I will keep these in the data set. It would be nice to parse out minutes between "active time" and "idle time" but don't think that it's worth it at this point in time. I will keep note of this as we go through the analysis.

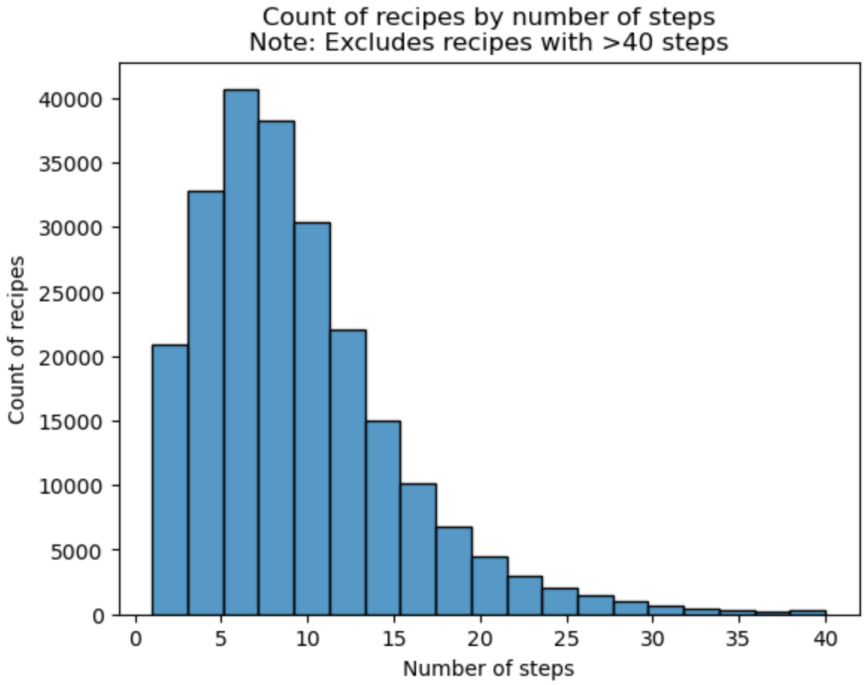
ii. n_steps: similarly for number of steps, the max is 145. This could be an error as well and will have to be looked at
    1. The number of steps is accurate for this case and it seems to be a true outlier. Looking at the steps column, the written steps make sense...this is truly a lengthy recipe.

iii. n_steps: the min number of steps is 0. This can't be right as a recipe should have a minimum of 1 step
    1. There is only one recipe or which there is 0 steps. It is a bread recipe. I've imputed the missing value with the average number of steps for recipes in which the name contains the string "bread". In this case, it is 11 steps.

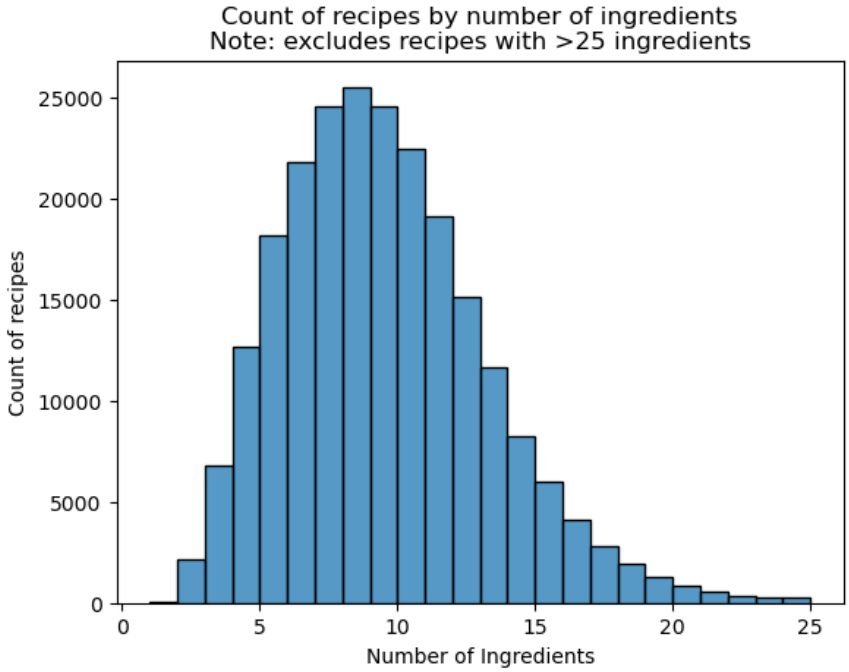
3. I then plotted histograms to visualize the dataset.
    a. **Percentage of recipes by estimated prep time (minutes).** This shows that the significant majority of recipes are completed in 60 minutes or less.



b. **Count of recipes by number of steps.** Most of the recipes are 10 steps or less.



c. **Count of recipes by number of ingredients.** Most of the recipes have between 6 and 12 ingredients.

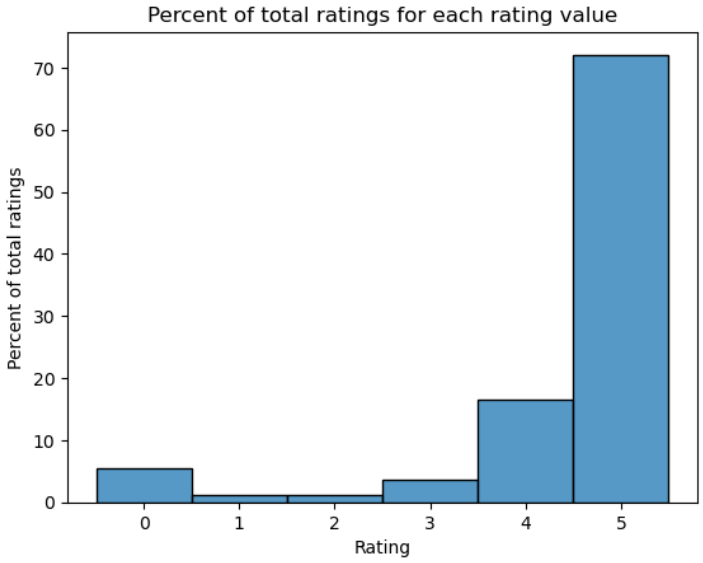


I then performed the following EDA on the interactions dataframe:

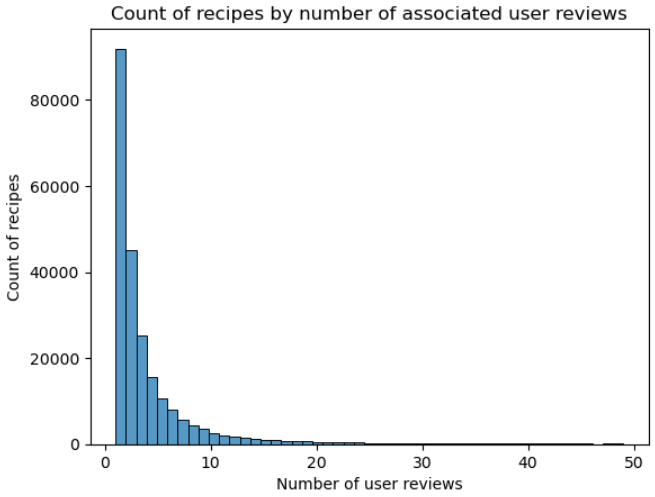
1. Computed the summary statistics for the dataframe. Specifically the 'rating' column.

	count	mean	std	min	25%	50%	75%	max
rating	1132367.0	4.411016	1.264752	0.0	4.0	5.0	5.0	5.0

2. Plotted a histogram of user ratings. **Over 70% of ratings are 5 stars.**

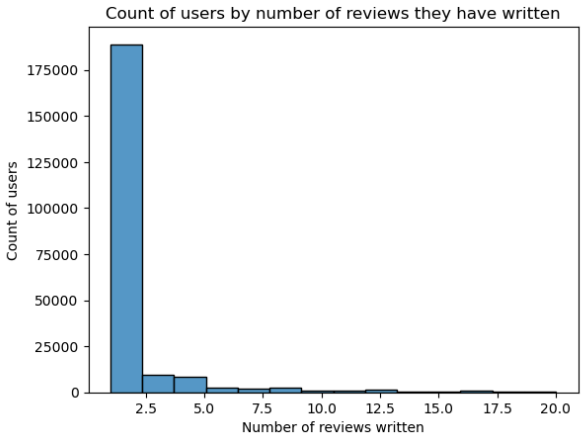


I then created some new features to get some more insights into the data. I wanted to see how many reviews each recipe has. I merged the recipe dataframe with the value counts of recipe id from the interactions dataframe. This created a new column in the recipe dataframe indicating how many times that recipe had been reviewed. I then plotted the resulting column as a histogram:



The majority of recipes have less than 5 reviews. This shows that there are very few recipes which have large amounts of reviews (the max is over 1600 reviews for one recipe) and many recipes which have very small number of reviews. This may present some challenges in training the model. Regardless of the model that is chosen, a higher level of user engagement would generate a higher performing model.

I next looked at a histogram of how many reviews each user generated. I did this by creating a pivot table of the interactions dataframe. The rows were the user ids and the values are the count of the unique recipe ids that each user reviewed.



I also used the describe method to get a breakdown of the summary statistics for number of reviews per user. Over 50% of the users have only reviewed one recipe, but there are some major outliers. One user has reviewed over 7600 recipes! I will need to take this into account when developing the recommendation model.

	count	mean	std	min	25%	50%	75%	max
Number of recipes reviewed per user	226570.0	4.997868	49.663111	1.0	1.0	1.0	2.0	7671.0

I then calculated the sparcity of the dataset, via the following equation:

$$Sparcity = 1 - \frac{Number\ of\ Interactions}{Total\ Possible\ Interactions} = 1 - \frac{length\ of\ interactions\ df}{n_{users} * n_{recipes}} = 99.9978\%$$

This is a high level of sparcity. Only 0.0022% of possible user-item interactions have actual values. This level of sparcity can be challenging for traditional methods of recommendation systems to find patterns. I will also have to be cognizant of the cold-start problem in which the model might struggle to have accurate recommendations for users or recipes with few interactions.

Modeling

The first approach I attempted to make in regards to making a model was to use an item-based collaborative filter. I chose this approach for the following reasons:

- The items – in this case, the recipes – have a more robust dataset. Therefore the model is more likely to pick up on similarities between the recipes with all the available features.
- On average, there were more interactions per recipe than there were per user. This allows for better estimations of similarity between recipes, even if users have only rated a small number of items
- Better recommendations could be produced even if a user is a "cold-start", which in this case, many users would be (having only written one review)

To execute this approach I had the following plan:

1. **Construct a user-recipe interaction table:** Each user will have a row and each recipe will be a feature. The values will be the ratings that the recipe was given by that user.
2. **Calculate item similarities:** I will explore using different similarity metrics to calculate item similarities. This can include cosine similarity, pearson correlation, or implement a KNN model to find similar items.
3. **Calculate predicted ratings of recipes:** Using KNN, I can select a number of nearest neighbors for each recipe and use a weighted average to calculate the predicted rating of that particular recipe for that particular user.
4. **Generate recommendations:** Select the highest predicted ratings for the recipes that the user has not interacted with yet as recommendations.

In order to construct a user-recipe interaction table, I tried to break the interactions_df into chunks of 10,000 rows. For each chunk I would create a pivot table where the user id was the row, the recipe id was a column and the values would be the rating. I would then concatenate the chunks together and group by the user id pulling in the max rating for each recipe id. I used the SparseDtype built into pandas to limit the memory usage of the dataframe.

This approach quickly ran into memory-constrain problems. Given that there are 231,673 unique recipes and 226,570 unique users, the user-item matrix would require 52,490,151,610 cells. My machine could not handle this memory load even with all the mitigation measures I

was implementing. Therefore, I decided to adjust the approach and use a less memory intensive option.

The second approach I decided upon was using a neural network. This approach had several advantages:

- **Handles large datasets efficiently.** This would be essential given my hardware constraints.
- **Handles high levels of dimensionality well.** Using embeddings to represent users and items would reduce dimensionality and allow the model to quickly capture similarities and relationships between users and items.
- **Easier customization.** I could easily tailor the model by using different loss functions or modifying the training objectives. TensorFlow has some easily used pre-build modules for building neural -network based recommendation systems.

I used the following approach to build the neural network recommendation system:

1. **Preprocess data:** convert categorical features in both recipe and user dfs into numerical values using tokenization and embeddings. I will also convert the datasets into TensorFlow datasets.
2. **Split the data:** split the data into training and test sets
3. **Define and create the model architecture:** Create input layers for both user interactions and recipes, define how many hidden layers we want, which activation function to use, the loss function to optimize, which optimizer we want to use
4. **Train the model:** Train the model on the training and validation data
5. **Evaluate the model:** Evaluate the model using the test set using a pre-defined evaluation metric such as RMSE
6. **Optimize the model:** Tune the parameters to improve evaluation metrics
7. **Deploy the model:** Deploy the model to be able to recommend recipes to new users

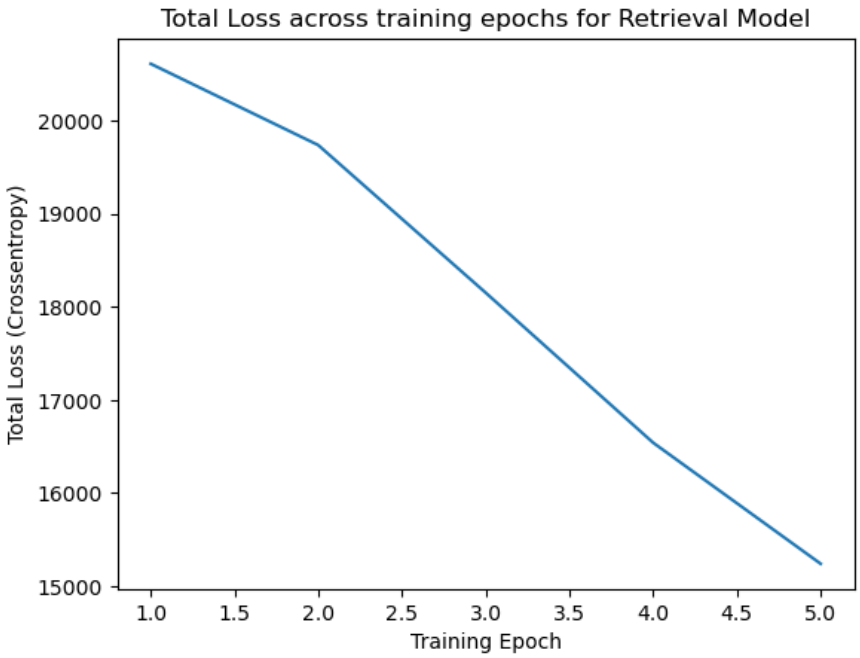
The model consists of two submodels:

Retrieval model – this model will take a user_id and output a short list of recipes that the user is likely to enjoy. This short-list is based on the similarities the model learned by being trained on the user-item interactions (e.g., which users interacted with which recipes.)

8. The model is trained by being fed the user ids and which recipes they have interacted with. These are treated as positive interactions. Recipes which they have not interacted with are treated as negative interactions.
9. The model seeks to minimize crossentropy loss. This loss function works by comparing a user's embedding with the positive recipes and a set of randomly selected negative recipes. The model learns to make the positive recipe closer to the user in the embedding space and push the negative recipes further away. During training, the

model tries to maximize the dot product between the user embedding and the positive recipe embedding. Similarly, it minimizes the dot product of negative recipes.

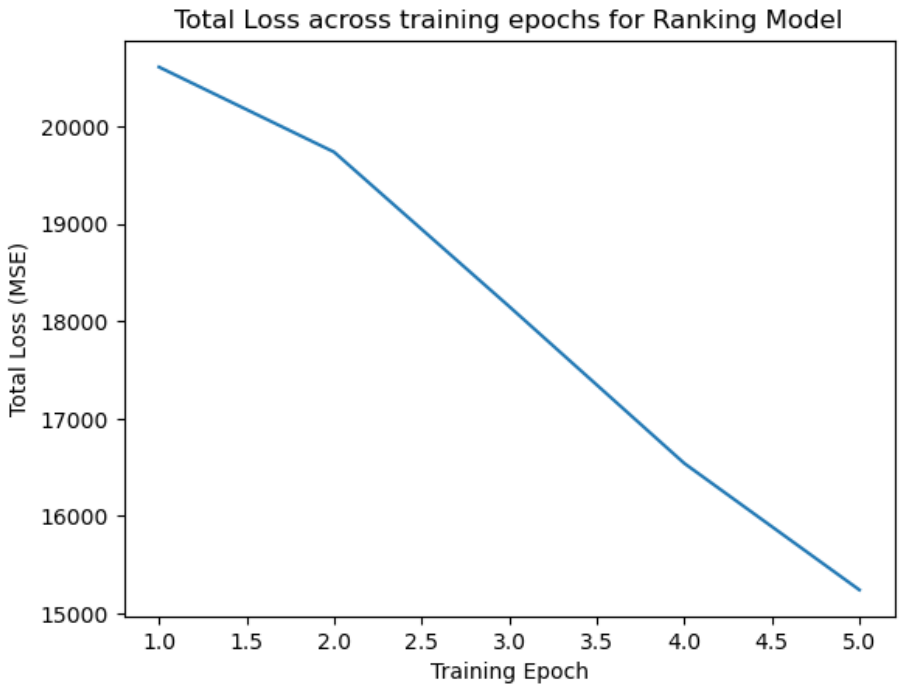
10. To evaluate this model, I'm using factorized top K as a metric. This metric checks if the truly relevant items (in this case, recipes the user has interacted with) have appeared within the top K results.
11. The following hyperparameters were used:
    a. Loss function – crossentropy loss
    b. Training epochs – 5
    c. Optimizer – Adaptive gradient (adagrad)
        i. This optimizer is commonly used in recommendation systems. It adjusts the learning rate for each parameter based on the magnitude of past gradients.
    d. Learning Rate – 0.1
    e. Train / test split – 70/30%
12. This model achieved the following loss function minimization during training:



- The model achieved a factorized top K of top 50 categorical accuracy of 0.0126 and a top 100 categorical accuracy of 0.0194. This means that on average about 1.26% of the time does the model include a relevant (positive) recipe in the top 50 recommendations and 1.94% of the time, does the model include a positive recipe in the top 100. This isn't necessarily surprising given that most users only have 1 recipe reviewed out of >200K recipes. Ideally this metric could be improved, and the training methodology could be refined in later iterations to improve this metric.

Ranking model – this model will take the same user_id as an input as well as the short list of recipes generated by the retrieval model to predict the rating of the recipes on the shortlist for that particular user.

13. To train this model, I created a dataset that included the recipe_id, the user_id, and the ratings.
14. I used a mean squared error (MSE) loss function to train the model. This will seek to minimize the mean squared error between the predicted rating and the true rating.
15. For the evaluation metric, I will use root mean squared error (RMSE).
16. The following hyperparameters were used:
    a. Loss function – MSE
    b. Training epochs – 5
    c. Optimizer – Adagrad
    d. Learning rate – 0.1
    e. Train / test split – 70/30
17. This model achieved the following loss function minimization during training:



- The ranking model achieved a RMSE of 1.228 on the test data set. This means that on average the model was about 1.2 stars off between the predicted rating and the actual rating. For such a sparse data set, this isn't terrible, but there is some room for improvement.
- The baseline which this can be compared is the RMSE for if we had predicted a rating of all 5. In this case, the RMSE would be 1.395, so the model achieves a significant improvement over the baseline.

Making Recommendations

To actually make recommendations, I created functions which would execute the following:

1. Take a user_id as input that feeds into the retrieval model to generate a list of 30 recipes the user is likely to enjoy but has not yet made
2. Take the same user_id and the list of 30 recipes and feed those as inputs into the ranking model to generate a predicted ratings for all 30 recipes for that user.
3. Sort the predicted ratings and use the recipe_ids to lookup the recipe names from the recipe dataframe
4. Print out the recipe names with the top 5 highest predicted ratings as recipe recommendations

Takeaways and Next Steps

Using neural network, I managed to take a large and sparse dataset and use efficient computing methods to create a recommendation system. This could help drive user interaction. In turn, increased user interaction would help improve the recommendation system accuracy. The following would be next steps for the recommendation system:

1. **Deploy the model –** This model could be deployed easily. A new user could answer a few questions regarding their recipe preferences (e.g., choose which recipe most appeals to you) to use as a baseline input. That user id and recipe preferences could be fed into the retrieval and ranking models to produce a list of 5 recommended recipes.
2. **Track user behavior / interaction –** an experiment should be run to see if / how the recommendation system changes user interactions. Does it have a positive impact? Are users more likely to interact with the website of recipes if they have these recommendations presented to them? These experiments can help refine the model. Key indicators could be engagement metrics like clicks or recipe saves and how they change with recommendations.
3. **Refine the model as more user interaction data is obtained –** there is much room for improvement on the model. We can include more features to train the ranking model. We can refine the model, deploy new models, and run experiments to determine which model is most effective at improving user interactions. Incorporating additional features like tags, review texts, nutrition may help greatly improve the model. However, this would take computational time and power.