**What were the main challenges faced during data preparation and how were these solved?**

The biggest challenge that we faced during data preparation was the size of the datasets that were used to build our recommendation system. The raw datasets that were available to download on Kaggle was **8GB** in total and this consisted of more than **5.2 million** unique reviews on **174,000** businesses. The datasets were so large that even reading the files into R Studio took some of our machines more than 5(?) minutes.

Very early on, we realized that it would not be feasible to suggest a recommendation engine using the entire dataset from Kaggle. The dataset consists of reviews from a wide variety of states in the U.S, hence it would not make sense from a user’s point of view to recommend businesses from “Nevada” when he could be at “North Carolina”! Hence, our group decided that we would only recommend business from “Nevada” for this project as this state contained the largest count of reviews. This resulted in the number of being unique reviews being cut to **1.75 million** and helped our machine to run the dataset much more easily. In an actual “production” recommendation engine, we would imagine that it would be less of a problem as the servers would be more than equipped with handling the load or otherwise, different models could be built for different states.

However, even after narrowing down the state we realized that our computers could still face problems building the recommendation system. The ratings matrix consisting of unique businesses and unique reviews would still exceed the current memory limit in our systems. Hence, our group then decided to narrow down the type of businesses in the reviews by examining the categories column. The initial analysis showed that there were more than 1160 categories in the reduced dataset, however, the term “Restaurants” appeared most frequently. Our team then decided to only use businesses that contained the term “Restaurants” in the categories. This also made sense from a recommendation point of view as we would not want “Hotels” to appear in the recommendation list when a user is only looking for places to eat.

With this filtering in place, the dataset still consisted of **800,000** reviews and **5000** restaurants which our group felt was still excessive for the purposes of this project. Our group decided to take a random sample of **2000** restaurants out of this dataset so that our machines would be able to run the data faster. We chose a random sample as we wanted to avoid any “biasness” during the recommendation. In order to ensure that all the members in the group obtained the same final 2000 restaurants, only one person in the group ran the sampling on his computer and exported it out as a .RData file. We believe that this would be better than setting a seed for random sampling as we were running different OS and R versions and the final sample might not have been the same across all our machines.

Another challenge that we faced during data preparation was determining what were the datasets that we actually needed to build the recommendation engine. One discussion our group had was whether we could actually use user profiles to build a recommendation engine by learning their preferences. This would have helped in the content-based recommendation model as we could examine the user’s likes or past visits. However, the user profile in the Kaggle dataset only provided information on whether other users liked his/her reviews, similar to that of a Facebook post. This provided no information of the preferences of our users and hence this idea had to be scrapped.

**What were the main challenges faced during solution building and how were these solved?**

One problem that we faced in building the recommendation system was the “Cold Start” problem that typically happens for new users. When building recommendation system models such as collaborative filtering, the model already assumes that each user or item have already been given some ratings, hence the model is able to infer the ratings of similar users or items even if those ratings are not available. However, for new users or items, there has been zero activity or ratings, making it difficult to recommend products.

One possible solution that we thought of was recommending the top 10 restaurants in terms of ratings when a new user signs up. However, these restaurants could be of various cuisines and not the one that the new user particularly likes. We looked at other recommendation systems in the market to come up with an alternative solution. One example is the recommendation engine used by Netflix, whereby a new user is prompted to select a series of genres and movies he or she is interested in before Netflix proceeds to give the recommendation. We implemented a new user input function for our recommender system similar to Netflix in order to give accurate recommendations the moment a new user signs up. We did this by asking what type of cuisine the user is interested in and what features they look out for when they visit a restaurant. By doing this, we were able to map our new user to the top 3 existing users who have similar tastes and preferences. These three users that were selected have rated the greatest number of restaurants that contain both the keywords and cuisines that our new user is interested in and gave us information about our new user that we otherwise didn’t have. This in turn allowed us to implement both collaborative filtering and content-based models on new users.

Another problem that we faced when building our final list of recommendations was how to build a hybrid list containing both recommendations from the content-based model and the collaborating filtering model. The first suggestion was to simply look at the predicted scores from both models and rank them in terms of total score. However, when we looked at the final predicted scores using this method, the content-based model produced very high scores only for the top few recommendations and lower scores for everything else. However, the collaborative filtering model produced scores that were more evenly distributed. This meant that both scores were not evenly weighted, and we would introduce bias if we were to simply take the total score from both models and divided it by 2. The next solution that we tried was to index the recommendations from the two different models separately and combine then. For example, if Restaurant ABC was ranked 2 in the collaborative filtering model and ranked 10 in the content-based model, we would rank it as (2+10)/2 = 6 in the final recommendation. However, this again failed to take in the score weighting of each model. In order to fix this, our team decided to incorporate weightings to the predicted score of each model. We did this by examining the Root Mean Square Errors (RMSE) of each model during the evaluation phase. We noticed that the collaborative filtering model has a lower RMSE than the content-based model, hence this would mean that the predicted score from the collaborating filtering model should be of greater importance or weightage than the other model. We converted these errors into weights for both of the models and ended up with a weight of 0.711 for the collaborative filtering model and 0.288 for the content-based model. Multiplying the weights to the predicted scores will then give the final predicted scores from the hybrid model.

**What conclusions and individual lessons have you learned?**

From this project, I have learnt that there are many aspects to building a recommendation system. Firstly, the data used for the recommendation engine has to be suitable. Without ratings data, it would not be possible to build a collaborative filtering model and without any product description or categories, it would not be possible to build a content-based model. Different companies might build different recommendation systems depending on the type of data that they collect. I have also learnt from this project that there is an explosion of data whenever Natural Language Processing is adopted. We expected in the beginning to be able to process all the reviews text and convert them into a document term matrix. However, we quickly found out that it was impossible and there had to be some form of filtering in order to trim the dataset down so that our machines could process the data.

Another lesson that I picked up over the course of this project is that there are many different types of libraries and algorithms that could be used to train recommendation models. After the first week of lessons, we decided to use the package recommenderlab to build the collaborative filtering model. We were able to use algorithms for User Based collaborative filtering as well as Item Based collaborative filtering to come up with two different recommendation lists. We believed that we would implement Item Based collaborative filtering as we would not have to update the model as frequently as compared to User Based. However, on the third week we were introduced to matrix factorization, specifically the Alternating Least Squares (ALS) algorithm which made use of latent features to represent users and items in a lower dimension space. While we did not fully understand how the algorithm works, we tested it out and found out that it produced lower errors compared to other algorithms which lead to our decision to implement ALS for collaborative filtering.