Recommender Systems for Evanston Restaurants

Hongli Peng

May 1st, 2023

1 Introduction

In this project, we aim to build a recommendation system for restaurants in Evanston using a file 'RestaurantReviews.xlsx' that contains 2 sheets, one is called **'Restaurants'** and another is **'Reviews'**. The first sheet is about the information for the restaurants in Evanston. The second sheet includes information about reviewers' demographic information and their reviews and rating to the restaurants in Evasnton. We will explore various techniques to analyze the data and predict restaurant scores, such as popularity matching, content-based filtering, natural language analysis, collaborative filtering, and predictive modeling.

2 Data Overview

The dataset **'Restaurants'** has the shape of (63, 7) and the 7 columns are **Restaurant Name**, **Cuisine**, **Latitude**, **Longitude**, **Average Cost**, **Open After 8pm?**, and **Brief Description**. There are 63 Restaurant Name, 24 levels for Cuisine, 5 levels of Average Cost, and 2 levels of Open After 8pm?.

The dataset **'Reviews'** has the shape of (1444, 14) and the 14 columns are **Reviewer Name**, **Restaurant Name**, **Rating**, **Review Text**, **Date of Review**, **Birth Year**, **Marital Status**, **Has Children?**, **Vegetarian?**, **Weight (lb)**, **Height (in)**, **Average Amount Spent**, **Preferred Mode of Transport** and **Northwestern Student?**. There are 1047 levels of Reviewer Name, 67 levels of Restaurant Name, 5 levels of Rating, 4 levels of Marital Status, 2 levels of Has Children?, Vegetarian?, and Northwestern Student?, and 3 levels of Average Amount Spent and Preferred Mode of Transport.

Our outcome variable in this report is **'Rating'** that 5 levels are 1.0, 2.0, 3.0, 4.0 and 5.0.

3 Exploratory Data Analysis (EDA)

We started by exploring the dataset to understand its structure and identify any potential issues such as missing or invalid data.

3.1 Missing values

We found that there is no missing values in the 'Restaurants' dataset, and the variables that have missing values in the 'Reviews' dataset are shown in Table 1. We devided three parts to deal with the missing values, 1) Review Text, 2) Numerical Variables, and 3) Categorical variables.

In the first part, there are 550 missing values 'Review Text' and the missing proportion is about 38.09%. Since this is a considerable proportion, dropping the missing values could result in losing a significant amount of information. So, we filled in the missing values with empty string ''.

In the second part, we dealt with numerical variables 'Birth Year', 'Weight (lb)', and 'Height (in)'. We thought that some reviewers might only provide their weight or only provide their height. We got that there is only 1 observation that only provided the weight. And there are 44 observations that only provided the height. Therefore, to use a **linear regression model** approach to fill in the missing values for 'Weight (lb)' and 'Height (in)' may be a good approach, which is better than removing or

Variable	Missing Values
Review Text	550
Birth Year	2
Marital Status	35
Has Children?	38
Vegetarian?	1350
Weight (lb)	97
Height (in)	54
Average Amount Spent	2
Preferred Mode of Transport	7
Northwestern Student?	1

Table 1: Summary of missing values in the Reviews dataset.

simply replacing the missing values. Here, we only considered 'Weight' and 'Height' are related each other.

We first split the dataset into two subsets, one with missing values that either 'Weight (lb)' or 'Height (in)' is missing and another with non-missing values. Using the non-missing dataset to train our linear model allowed us to predict the missing values of 'Weight (lb)' and 'Height (in)'. For example, at first, we trained our model with x = Height (in) and y = Weight (lb). Secondly, we dropped the missing values for 'Height (in)' because we can not predict a null value, and this was the missing dataset that only contains 'Height (in)'. Thirdly, we used this to predict the missing 'Weight (lb)' in a linear model trained by the non-missing dataset. Similarly, we dropped the missing values for 'Weight (lb)' and predict the missing 'Height (in)'. Finally, we successfully predicted as many the missing 'Height (in)' and 'Weight (lb)' as possible given the information of non-missing Height (in) or Weight (lb).

Now, we imputed all the possible weight and height by using linear regression models. The 53 left missing values for both variables would be then replaced by the median. Also, there are only 2 missing values for birth year, we would simply replace it by the median.

In the third part, we dealt with categorical variables: 'Average Amount Spent', 'Marital Status', 'Has Children?', 'Vegetarian?', 'Preferred Mode of Transport', and 'Northwestern Student?'.

Variable	Missing (%)
Marital Status	2.42
Has Children?	2.63
Vegetarian?	93.49

Table 2: Proportion of missing values for these 3 variables.

1. Since there are only one 2 missing values for 'Average Amount Spent', 7 for 'Preferred Mode of Transport', and 1 for 'Northwestern Student?', we can use a simple method, replacing by the mode.

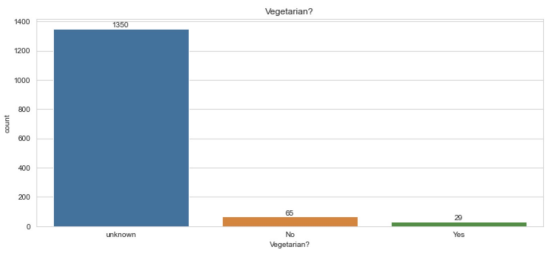
2. From table 2, since the the proportion of 'Vegetarian?' is 93.49% that there are many missing values, we created a new category 'unknown'.

3. From table 2, the proportion of missing 'Martial Status' and 'Has Children?' are 2.42% and 2.63%. We also created a new category 'unknown' for retaining as many information as possible.

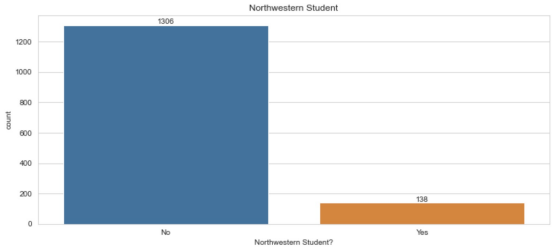
3.2 Histograms for interesting variables

From figure 1, we discovered 5 interesting histograms for our variables. The first four histograms showed that there are some unbalanced problem we should pay attention to. The first one is **Vegetarian?** that there are 1350 *'unknown'* much higher than other two categories. The second one is

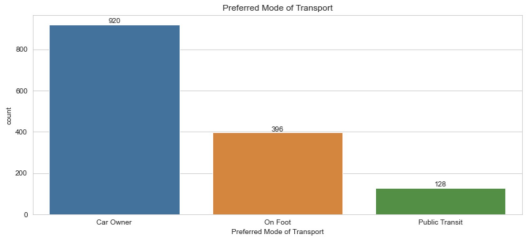
Northwestern Student? that there are 1306 *'No'* much higher than 138 'Yes'. The third one is **Preferred Mode of Transport** that 3 categories here are unbalanced that 'Car Owner' is larger than the sum of 'On food' and 'Public Transit'. The fourth plot is **Martial Status** that there are only 54 *'Widow'*. The fifth plot is **Cuisine** that we found that there are 8 *'American'* restaurants, which has the highest number among other cuisines.



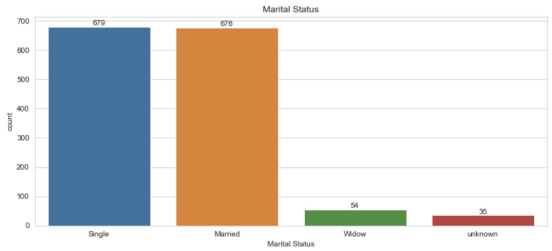
(a) Vegetarian?



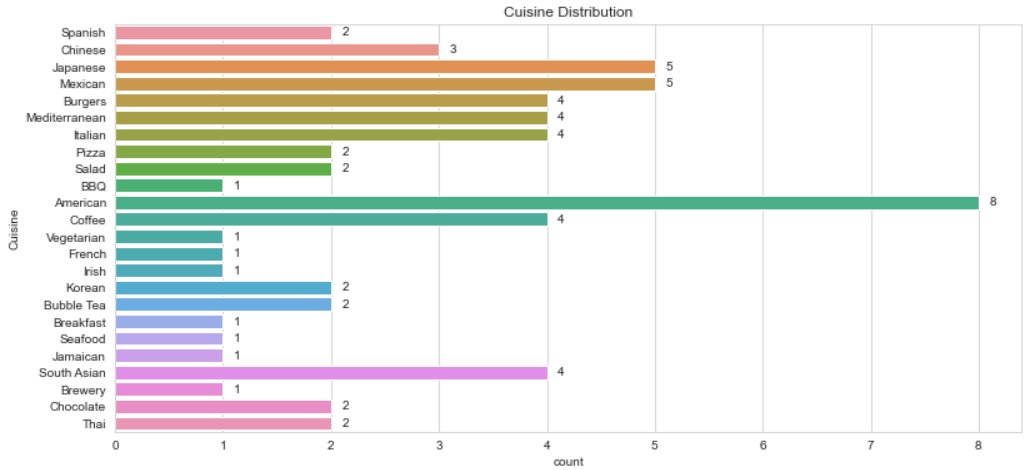
(b) Northwestern Student?



(c) Preferred Mode of Transport



(d) Martial Status



(e) Cuisine

Figure 1: Five Plots

3.3 Clustering the demographic data

In this section, we utilized the k-means to cluster the demographic data. The columns chosen for this demographic data are 'Birth Year', 'Weight (lb)', 'Height (in)', 'Marital Status', 'Has Children?', 'Vegetarian?', 'Average Amount Spent', 'Preferred Mode of Transport', and 'Northwestern Student?'. We followed by the following steps:

1. Converted the categorical variables to **one-hot encoding** to ensure that each category is represented by a separate feature, but for the binary variables, we transformed them to 0 and 1 in one column. It is because converting binary variables to one-hot encoding can increase the *dimensionality* of the data, which can be computationally expensive and may not improve the clustering performance.

2. Scaled the numerical variables using **standardization** to prevent some variables have a higher weight than others, which may yield bias to the model.

3. Applied **ELBOW** method to determine the optimal number of clusters needed in K-means. (*we used the code from pages 59 in Lecture 3 clustering*)
   Idea:
   It involved fitting the model with a range of cluster numbers K and plotting the Sum of Squared Errors (SSE, also called inertia) for each k. The optimal K is usually where the elbow of the curve occurs, indicating that adding more clusters does not significantly improve the model's performance.

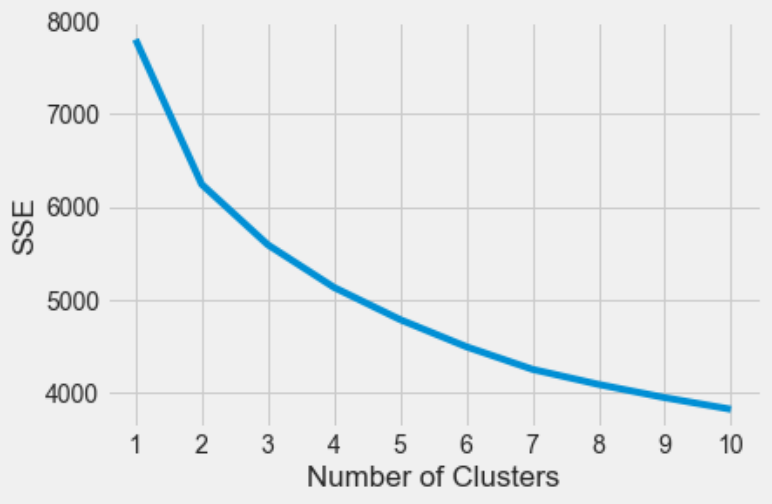


Figure 2: Elbow Method

From the plot 6, the optimal K cluster is between 2 and 5. We could use the **KneeLocator** from the **kneed** library that is used to identify the elbow point programmatically in the SSE curve. (*from pages 61 in lecture 3 clustering*) Finally, the optimal number of cluster k is 4 in our 'Reviews' dataset.

4. Fitted the **K-means** model that is an iterative clustering algorithm with the optimal number of clusters = 4.
   Idea: (*Idea is from pages 10 in lecture 3 clustering. The code is from pages 30 in lecture 3 clustering*)
   1) Initialize: Pick K random points as cluster centers, 2) Alternate: 1. Assign data points to closets cluster center, 2. Change the cluster center to the average of its assigned points, 3) Stop when no points' assignments change.

5. We created a column **'Cluster'** that is equal to 'kmeans.labels_' to our 'Reviews' data.

6. Analyzed each cluster.
   We grouped by clusters to calculate the mean of numerical variables and the average rating for

Table 3: Average Rating, Birth Year, Weight, and Height for each Cluster

Rating	Birth Year	Weight (lb)	Height (in)	Count
3.96	1994	183.02	160.50	383
3.92	1990	191.15	186.60	349
3.69	1960	179.19	165.91	346
3.45	1970	260.57	176.53	366

each cluster. From the table 3, we got the average rating, birth year, weight and height for each cluster. From the result, there might exist a **slight** trend that younger people might tend to give a higher rating compared to elder people. The properties for the clusters might be different according to what clustering methods we applied and how we processed the data.

We also grouped by clusters and each categorical variables to calculate the average rating for each cluster and the count of each category in each cluster, in order to identify the difference and trend. However, there are no obvious trends here.

4 Popularity Matching

In this section, we applied **Popularity Matching**, the simplest type of recommendation system. There are two ways to score popularity. 1) **User engagement** that is the total number of clicks or views or purchases; 2) **User reviews** that is upvote or downvote and rating from 1 to 5. (*From pages 10-12 in lecture 4 recommondation system*)

4.1 Option 1 User Engagement

For the option 1, we identify the restaurants that have received the largest quantity of review, and the median number of reviews. We grouped the data by 'Restaurant Name' to count the number of ratings for each restaurant. From the table 5, the 'Campagnola' has received the largest quantity of

Table 4: Top 5 restaurants by count

Restaurant Name	Count
Campagnola	48
Chipotle	41
Cozy Noodles and Rice	38
Taco Diablo	37
Tealicious	34

48 reviews. From this aspect, it is most popular. We also got the median number of reviews is 23, which allows us to have a better sense of the distribution of reviews counts for each restaurant.

4.2 Option 2 User Reviews

For option 2, we first grouped by restaurants to calculate the *mean rating*, **median rating**, and *reviews counts* for each restaurant, and sorted by the decreasing mean rating. We derived the top 6 restaurants and the results are stored in table 5.

Table 5: Summary statistics for restaurant ratings

Restaurant Name	Mean	Median	Count
Fonda Cantina	5.00	5.0	6
LeTour	5.00	5.0	4
World Market	5.00	5.0	2
Evanston Games & Cafe	5.00	5.0	1
La Principal	5.00	5.0	1
Zentli	4.76	5.0	17

From the table 5, From the table, 'Fonda Cantina', 'LeTour', 'World Market', 'Evanston Games & Cafe', and 'La Principal' have *the highest mean rating* of 5.0. The the median scores for these restau-

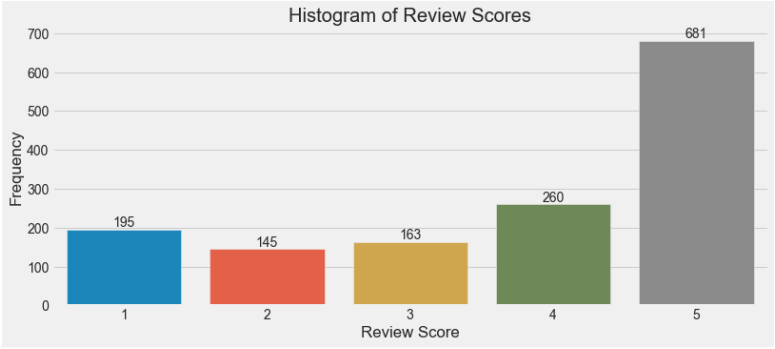


Figure 3: Distribution for reviews scores

rants are all 5.0, which is really good. What's more, the **'Fonda Cantina'** has the *highest reviews counts* among these 5 restaurants, which is the most popular one among the these five restaurants.

Secondly, we calculated the *average reviews score* and the *median score* among all restaurants. From table 6, the average reviews score is 3.753 and the median is 4.0.

Table 6: Summary statistics for overall review scores

Statistic	Value
Average review score	3.753
Median review score	4.0

We also plotted the **histogram of reviews scores** to visualize its distribution. (Note: Rating has five categories, from 1 to 5.) From the figure 3, we found that the most frequent rating is 5.0 and the count is 681, which means that most people are willing to give 5.0 rating to the restaurants.

4.3 Build a Recommendation Engine

We built an recommendation engine wherein a user can input a cuisine type and receive a rec commendation based on popularity score. And we used this to give recommendations for Spanish food, Chinese food, Mexican food, and Coffee. We built the engine with the following procedures:

1. **Merged** the 'Reviews' with the 'Restaurants' datasets to get the restaurants information along with their ratings.

   We found that the total number of restaurants in 'Reviews' is 67 and the total number of restaurants in 'Restaurants' is 63. The different restaurants between two datasets are 'World Market', 'Todoroki Sushi', 'La Principal', and 'Clare's Korner'. Hence, we got a table 7 that their review counts are really small number. We could use **imputation method** for their cuisine type and other information. But for ***simplicity*** , we used **inner join** for merging this two datasets and ignore these 4 restaurants. The shape of the this merge dataset is (1436, 21) and we have 63 restaurants.

   Table 7: Summary statistics for 4 different restaurants

Restaurant Name	Mean	Median	Count
World Market	5.00	5.0	2
Todoroki Sushi	3.25	3.0	4
La Principal	5.00	5.0	1
Clare's Korner	1.00	1.0	1

2. Grouped by 'Restaurant Name' and 'Cuisine' and sorted by mean rating and then count, then compute the mean rating and count of reviews for each restaurant. We have 63 restaurants here.

3. Defined the **shrinkage estimator** function:(Using the formula from page 16 in lecture 4 recommendation-systems) $\frac{N_\mu * \mu_s + N_p * \mu_p}{N_\mu + N_p}$, where the shrinkage never goes away entirely. Let μ_s be the overall mean rating and N_μ be the overall mean counts of ratings. μ_p is the mean rating for the p-th restaurant and the N_p is the the number of ratings for p-th restaurant.
4. We got the overall mean ratings is 3.753 and the overall mean review count is 22.
5. Applied the shrinkage estimator function to the restaurant ratings.
6. Defined a function to recommend a restaurant based on the input **cuisine type** and the highest shrinkage rating.
7. Used this function to recommend the best restaurant for Spanish food, Chinese food, Mexican food, and Coffee.
   We finally got the best restaurant for these 4 Cuisine types and the results are sotred in table 7.

Table 8: Recommended restaurants for the 4 Cuisine types

Cuisine	Restaurant Name
Spanish	Tapas Barcelona
Chinese	Joy Yee Noodle
Mexican	Zentli
Coffee	Philz Coffee

4.4 Analyze the Shrinkage Method

We created a new column **'shrinkage_diff'** that is the difference between mean rating for each restaurant and its shrinkage rating. Then this allows us to find the restaurant that benefit the most and is hurt the most from shrinkage method. The table 9 showed the top 5 restaurants that benefit the most from shrinkage method. The table 10 showed the top 5 restaurants that are hurt the most from shrinkage method.

Table 9: Top 5 restaurants that benefit the most from shrinkage

Restaurant Name	Cuisine	Shrinkage Diff
Burger King	Burgers	0.914
Evanston Chicken Shack	American	0.621
Cross Rhodes	Mediterranean	0.525
Cozy Noodles and Rice	Thai	0.498
Mumbai Indian Grill	South Asian	0.433

Table 10: top 5 restaurants that are hurt the most from shrinkage

Restaurant Name	Cuisine	Shrinkage Diff
Evanston Games & Cafe	Coffee	-1.193
LeTour	French	-1.055
Fonda Cantina	Mexican	-0.980
Zentli	Mexican	-0.571
Philz Coffee	Coffee	-0.504

We also made a histogram plot to better visualize it, the figure 4. The yellow means it benefits the most and the blue means it is hurt the most. The y axis is the restaurants and the x axis is the

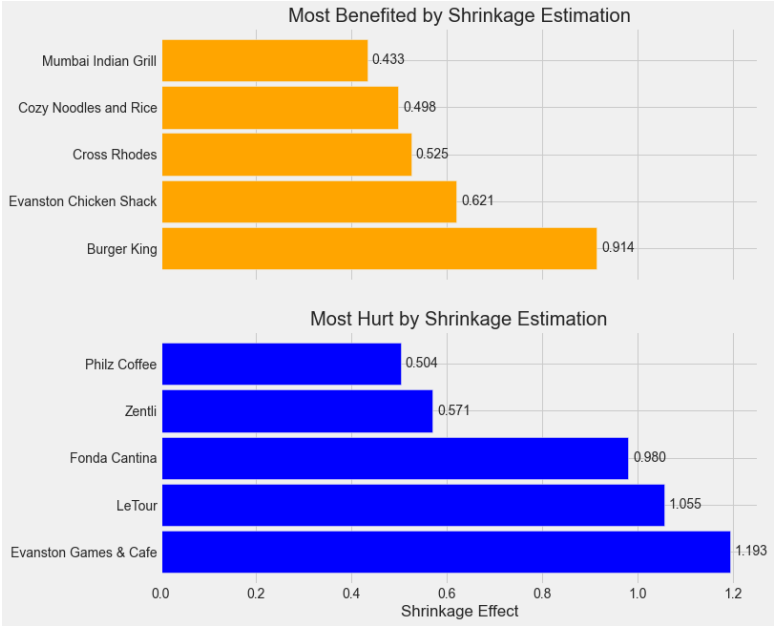


Figure 4: Restaurants that are influenced the most by Shrinkage

shrinkage difference where we take the absolute value. Finally, the **Evanston Games & Cafe** is hurt the most and **Burger King** benefits the most.

The shrinkage estimator is used in popularity matching to address the issue of limited sample sizes for some items and to avoid overestimating the popularity of an item based on a few highly positive or highly negative reviews. This technique, also known as **Bayesian estimation**, helps to produce more reliable and stable popularity rankings, especially when dealing with sparse data or items with a limited number of reviews.

5 Content-Based Filtering

From pages 18 lecture 4 recommendation systems, the idea is to recommend users items that are **similar** to items they have previously interacted with. In our context, we should found the **similarity** between restaurants. From 'Restaurants' dataset, we should only include columns **'Cuisine'**, **'Average Cost'**, and **'Open After 8pm?'** to calculate the similarity. We excluded the numerical columns **'Longitude'** and **'Latitude'** because when observing the summary statistics, these two variables made no difference across restaurants.

By checking the uniqueness for the variables, the number of categories for 'Cuisine' is 24. The number of categories for 'Average Cost' is 5. The number of categories for 'Open After 8pm?' is 2. Therefore, we should transform 'Cuisine' to **one-hot encoding** and 'Open After 8pm?' to a binary variable 0 and 1 in its column. Although **'Average Cost'** has five integers 13, 20, 27, 50, and 40, it represents an ***ordinal variable*** with a natural order (higher values indicate a higher cost). One-hot encoding is not necessary in this case because the numeric values already capture the ordinal relationship between the categories. Another reason is that one-hot might *increase the dimensionality*. As a result, we transformed 'Average Cost' by standardizing.

5.1 Similarity and Distance

There are two ways of calculating the similarity between restaurants. One is **Euclidean Distance** and another is **Cosine Similarity**. (Formulas are from notes 25 in Lecture 4, codes are from pages 34 and pages 35)

1. **Euclidean Distance** formula is $d(\mathbf{A}, \mathbf{B}) = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2}$, then the **Similarity** is $1 - d(\mathbf{A}, \mathbf{B})$.

2. **Cosine Similarity** formula is

$$\cos(\mathbf{X}, \mathbf{Y}) = \cos(\theta) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}} \tag{1}$$

   Then the **Cosine Distance** is $1 - cos(\mathbf{X}, \mathbf{Y})$

5.2 Built a Script for Content Based Filtering

This script takes a user, finds restaurants the user liked, and then find similar restaurants using euclidean and cosine distances. We defined a function **'get_similar_restaurants'** with inputs of (user_name, distance_metric, K, reverse), where user_name is the reviewer, distance_metric should be either Cosine similarity or Euclidean distance, K is the top 5 restaurants that has the highest similarity, and reverse means the way of sorting the metric. For cosine similarity, reverse should be True(from the highest to the lowerest) and for Euclidean distance, reverse should be False.

Idea:

1. Found the restaurant that the user rating the most and the highest.

2. Calculated the distance between this restaurant and other restaurants with the metric we chose.

3. Sorted the distance from lowest to highest and the similarity from highest to lowest.

For example, **get_similar_restaurants**('Jacquelyn Rigatti', euclidean_distance, 5, reverse=False) would give us the top 5 restaurants that has the lowest euclidean distance with the restaurants 'Jacquelyn Rigatti' like the most. Cosine similarity is the same except the reverse=True. Finally, our result showed that both metrics provided the **same** recommendations 'Alcove', 'LeTour', 'Oceanique', 'Graduate Homestead Room', and 'Kansaku'.

6 Natural Language Analysis

We would focus on **'Brief Description'** in 'Restaurants' dataset and adopt **Jaccard Similarity**, **TF-IDF score**, and **BERT** to make recommendations. The main procedure here is that we made a new column **'Augmented Description'** that attaches the restaurant's **'Cuisine"** type to the end of the **'Brief Description'**.

6.1 Jaccard Matrix

We computed the **Jaccard Matrix** using the elements of 'Augemented Description'. The formula is $J(\text{doc1}, \text{doc2}) = \frac{\text{length(intersection)}}{\text{length(union)}}$. (code is from pages 55 in lecture 4) The idea is that you look at all of the words that two descriptions have in common and divide that by the total number of unique words in both descriptions. (idea is from pages 46 in lecture 4) Higher similarity is better. Finally, we got a 63x63 matrix where rows and columns are resturants' names and the entries are **Jaccard Similarity**. The diagonal entry should be 1 because the restaurants are the same.

6.2 TF-IDF Score for each Restaurant's Augmented Description

From pages 61 to 68 in lecture 4 notes, **TF-IDF** can deal with the problem of frequent unmeaning words like **"is"** and **"the"** that might be problematic in Jaccard similarity. IF-IDF is Term Frequency-Inverse Document Frequency, which is a **corpus-wide** metric.

1. IF-IDF = IF * IDF
2. Tf(t) = $\frac{Number\ of\ times\ t\ appears\ in\ document}{Total\ number\ of\ terms\ in\ the\ document}$, which measures how many times does a term occur in a document.
3. IDF(t)= ln($\frac{Total\ number\ of\ documents}{Total\ number\ of\ documennts\ with\ term\ t}$), which measures how important or unique a term is overall.

Words with higher TF-IDF scores are more important in the document and are less common across the entire corpus. Words with lower TF-IDF scores are less important in the document and are more common across the entire corpus. We used the package **TfidfVectorizer** from sklearn to calculate the TF-IDF scores for each word in each restaurant.

We defined a function **find_highest_tfidf(word)**. Firstly, we found the restaurant index with the highest TF-IDF score for the given word. Secondly, we filtered the restaurant name and the highest TF-IDF score. For instance, for **'cozy'** and **'chinese'**, the restaurant with the *highest* TF-IDF score for the word **'cozy'** is **'Taste of Nepal'** with a score of 0.288. And the restaurant with the *highest* TF-IDF score for the word **'chinese'** is **'Lao Sze Chuan'** with a score of 0.420. This makes sense that **Lao Sze Chuan'** is indeed a chinese restaurant which means our model performed good.

6.3 TF-IDF score for 100 Most Popular Words for Restaurants

1. Count the occurrences of words in the corpus 'Augmented Description'. Here, we have removed the punctuation since they are not meaningful. Then we found the most popular words based on this.

Word	**Frequency**
for	64
known	62
a	28
in	27
and	19

Table 11: Frequencies of top 5 words

From table 11, the top 5 most popular words are **'for'**, **'knwon'**, **'a'**, **'in'**, and **'and'**.

2. Initialized a **TfidfVectorizer** object with the most 100 popular words and fitted the 'vectorizer' to the corpus and compute the TF-IDF scores. Then converted this sparse matrix to a dense array.
3. Finally, the matrix is 63*100 that is 63 restaurants and the 100 words. The entry is TF-IDF scores for the i-th restaurant and i-th words.

6.4 TF-IDF Distance Matrix

In this section, we compute the **'TF-IDF Distance Matrix'** where d_ij is the distance between the TF-IDF vectors for restaurants i and j. We chose the **Cosine Similarity** as our distance metric that the higher value means better. The reason for using the cosine similarity is that it's a commonly used metric for comparing text documents in the context of TF-IDF, and the magnitude doesn't matter here. The idea here is that we calculate the cosine similarity of each row vector for the previous 63*100 matrix to other 62 rows vectors, because each row vector is features for each restaurant. In the end, we got 63*63 matrix where the entry is the cosine similarity between restaurants.

6.5 BERT Embedding

We utilized **BERT** to embed the restaurant descriptions into a vectorized representation, then compute an Embedding-Distance matrix, where d_ij is the distance between embedding vectors of restaurants i and j. IF-IDF only consider a single word. It has issues when embedding an entire sentence. Word2vec & BERT do not compute the distance between two documents. Instead, they focus on learning an embedding f. (from pages 74 in lecture 4)

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained language model that is capable of understanding the context and semantics of a given text. It is based on the Transformer architecture and has been trained on a large corpus of text. In the provided code, we use BERT to generate embeddings (vectorized representations) for the Augmented Description of each restaurant. These embeddings can be used to compute the similarity between different restaurant descriptions. We also chose **'Cosine Similarity'** that the higher is better.
Steps:

1. Loaded the pre-trained BERT model and tokenizer **bert-base-uncased** since the tokenizer is responsible for converting the input text into tokens that BERT can understand.
2. Defined a function **get_bert_embedding(text)** to get BERT embeddings for a text, then we could get BERT embeddings for the 'Augmented Descriptions'.
3. Calculated the distance matrix using cosine similarity. As a result, we get a 63*63 matrix where the entry is cosine similarities between two restaurants.

6.6 Comparing Jaccard Distance, TF-IDF Distance, and BERT

In this section, we used the idea from pages 71 in lecutre 4.

1. Let the review score for restaurant i be denoted by s_i.
2. Computed the mean distance between a given restaurant and every other restaurant. Use this as a threshold d_{mean}. In our case, we computed the mean similarity between restaurants.
3. At restaurant i, denoted r_i, find the k closest restaurant (in terms for Jaccard, TF-IDF or BERT). Require that the distance is less than d_{mean}. In our case, we should have the **Cosine Similarity** that is higher than d_{mean}.
4. Compute $Z_{jac} = \frac{1}{k}\sum_{i=1}^{k} s_i$. Compute Z_{TF} and Z_{BERT} the similar way.
5. Finally, we would get the average rating for each recommendation systems and compare the final results.

For instance, we intended to find the 5 closet restaurants for **'Lao Sze Chuan'**. By using the steps above, our result are stored in the table 12. From the result, **Bert** performed the best with the highest average mean rating for the top 6 restaurants for **'Lao Sze Chuan'**.

Method	Average Mean Rating
Bert	4.050
TF-IDF	3.967
Jaccard	4.024

Table 12: Average mean ratings for top 6 restaurants among 3 recommondation systems

7 Collaborative Filtering

Collaborative filtering is a technique used to recommend items based on the preferences of similar users.

7.1 Demographic Data for each Reviewers

The final demographic vector is 'Reviewer Name', 'Birth Year', 'Weight (lb)', 'Height (in)', 'Preferred Mode of Transport', 'Average Amount Spent','Marital Status', 'Northwestern Student?', 'Has Children?', and 'Vegetarian?'

1. Dropped duplicates based on the **'Reviewer Name**, which we got the unique reviewer along with their demographic information.
2. Transformed the categorical variables that have more than 3 levels to one-hot encoding, and that have 2 levels to a binary variable 0 and 1. And we standardized the numerical variables.
3. In the end, the final demographic data has the shape of (1047, 21), where we have 1047 unique reviewers and 21 features.

7.2 Similar Users Using Cosine Distance

This is the option 1 of **'How can we find similar users?'** from pages 100 in lecture 4. The idea is that takes the demographic vectors first and then computes the cosine distance between users $d(x_i, x_j)$. We performed this idea with several steps:

1. Defined the **Cosine Distance** function, which is 1 - **Cosine Similarity**, that the lowerer is better.
2. Defined the function **user_distances(user_name, demographic_data)** that computes the cosine distance between the given user and every other user.
3. Defined **recommend_similar_users(user_name, df_reviews, demographic_data, K, K1)** function. This function finds the top K1 similar users based on the cosine distance between their demographic data. It then collects the reviews made by these similar users and calculates the average rating for each restaurant. Finally, it returns the top K restaurants based on the average ratings.

Table 13: Top 10 closest distances for **Jacquelyn Rigatti**

Name	Distance
Chris Jacobson	0.014
Sonya Morrow	0.022
Scott Stamps	0.023
Courtney Butler	0.025
Vanessa Paulseth	0.028
Stanley Rose	0.033
Douglas Osborn	0.034
Richard Paddock	0.037
Parker Castro	0.048
Lisa Riley	0.050

From the table 13, we derived the top K1=10 reviewers that has the closest distances with **Jacquelyn Rigatti**. 'Chris Jacobson' has the most similarity. From the table 14, we derived the top K=5 restaurants for **Jacquelyn Rigatti**

7.3 Similar Users Using the Scores themselves to Compute Cosine Distance

This is the option 2 of **'How can we find similar users?'** from pages 101 in lecture 4. We used the **scores** themselves to compute the cosine distance between users, which is often what people mean by collaborative filtering. There are several steps:

Table 14: Top 5 restaurant recommendations for Jacquelyn Rigatti

Restaurant Name	Rating
Le Peep	5.0
LeTour	5.0
Tealicious	5.0
Zentli	5.0
5411 Empanadas	4.0

1. Find a reviewer who has given at least 4 reviews. We found 'Jacquelyn Rigatti'.
2. Pivot the reviews dataframe to create a user_restaurant rating 1047*67 matrix where the row represents i-th of 1047 reviewers and the column is j-th of 67 restaurant. The entry is rating for i-th reviewer and j-th restaurant. The 0 entry means that the i-th reviewer made no rating for j-th restaurant.
3. For filling the blank entries, we can first use **K-means Clustering**. 1) Based on scores, cluster the users into K cluster. 2) Map user i to their closet cluster. 3) For a new item j, just rank it with the average score within the cluster. 4) For the rest of 0 entries, we replaced them by mean imputation.
4. Select row where 'Jacquelyn Rigatti', we are able to get 67*1 vectors that how 'Jacquelyn Rigatti' rated each restaurant.
5. Then we used cosine distance to calculate between user's review scores vectors
6. We are able to get the top K1=10 similar users and get the top 5 restaurants that have the highest average rating.

7.4 Comparing these two options

We just compared the average overall ratings for the restaurants these two option recommended for 'Jacquelyn Rigatti'. For option 1, the average overall rating is 4.26. For the option 2, the average rating is 3.45, which the option one is better. Usually the option 2 is better and the reason here may be there are many 0 entries for the option 2 which yields a high bias.

8 Predictive Modeling

For the predictive modeling, we used a linear model that takes demographic data and the cuisine type for a restaurant to predict the restaurant score. We compared the performance of the model with and without the review text embeddings.

8.1 Linear Model without Embeddings

Using demographic and cuisine type data, we trained a linear model and calculated the mean squared error (MSE) for both the training and testing sets. The results were as follows: We split the dataset to two parts, testing set (30% of the data), and training set (70% of the data).

- Training MSE: 1.8484
- Testing MSE: 1.9053

For exmaple, we take one review from the test set and use our model to take user demographics for this review and cusine type and predict a score. We selected one review for 'Peppercorns Kitchen'. We found that the actual rating of 'Peppercorns Kitchen' is 4.0 and the predicted rating of 'Peppercorns Kitchen': 4.0078125, which two values are close and the model performed excellent.

8.2 Adding L1 Lasso Regularization

We added L1 Lasso Regularization to the previous standard linear model becuase this can solve the over fitting problem.

1. Used the package **LassoCV**, trained the LassoCV model to find the **best alpha value**.
2. Trained the Lasso model with the **best alpha 0.00628**, and evaluated it on the test set.
3. Finally, the train mse is 1.8812 and test mse is 1.8438. We can know that our traing mse increased and the test mse decreased, which yields a better performance and solved the over-fitting problem.
4. Compared the coefficients of the Lasso model to identify the selected features. From table 15, the

Table 15: Coefficients for Linear and Lasso Models

Feature	Linear Model	Lasso Model
Marital Status_unknown	7.758e+12	-1.223
Cuisine_Burgers	3.764e+11	-1.118
Vegetarian?_No	3.845e+12	-0.891
Cuisine_Thai	3.764e+11	-0.690
Cuisine_American	3.764e+11	-0.539
Cuisine_Italian	3.764e+11	-0.383
Cuisine_Mediterranean	3.764e+11	-0.359
Cuisine_Coffee	3.764e+11	0.323
Has Children?_No	-1.502e+13	0.278
Marital Status_Married	7.758e+12	0.247

higher the absolute coefficient value, the more important the feature is in predicting the review score. So we sorted the coefficients by the absolute value of Lasso model coefficients and got the top 10 features. Small coefficients represent features that are less important, and their impact on the prediction is minimal. Negative coefficients show features that have a negative relationship with the target variable. As the feature value increases, the target variable decreases.

8.3 Linear Model with Embeddings Using BERT

Next, we incorporated the BERT embeddings into our dataset and trained a new linear model using the same process.

1) we loaded the BERT model for embedding the review text. Since the 'en_trf_bertbaseuncased_lg' model is not available, we'll use the 'sentence-transformers/bert-base-nli-mean-tokens' model from Hugging Face.

2) Create a function to get the BERT embeddings.

3) Embed the review texts: At first, ignore the unknown in 'Review Text'. Furthermore, convert the 'Review Text' to a list of strings before passing it to the get_bert_embeddings function:

4) Train a linear regression model to predict review scores. The results were:

- Shape for the dataset is (889, 768)) where we have 889 non empty 'Review Text' and 768 is the features for this BERT model.
- Training MSE with embeddings: 0.008
- Testing MSE with embeddings: 62.59

Removing the missing value for 'Review Text' would cause a over-fitting problem, which is not really good. And the testing mse is higher.

8.4 Linear Model with Embeddings Using BERT and Demographic Data

In this section, we include the demographic data and BERT embedding to train a linear model. Finally, the results were:

- Shape for the dataset is (889, 816) where we have 889 non empty 'Review Text'.
- Training MSE with embeddings: 0
- Testing MSE with embeddings: 3.5844

The 0 training MSE may indicate an over-fitting problem and 3.5844 for Testing MSE

8.5 Comparisons for different model

We also calculated the training and testing MSE of both the linear model with BERT embeddings that contains empty 'Review Text' and the linear model with BERT embeddings and demographic data that contains empty 'Review Text'. We made a table to better compare training MSE and testing MSE.

From the table 16 comparing 6 different linear models, we are to conclude which model has a lower training or testing MSE. Comparing with or without empty 'Review Text', it is apparent that including the **empty** 'Review Text' will have a higher training MSE in BERT and even a lower one when incorporating the demographic data. And it has a large testing MSE 571913.3233 in BERT and even the one close to infinity. Therefore, containing all the empty 'Review Text' may be not a good choice to do. One might consider other technique to **'embedding the missing'** such as **'Average embedding'**, **'Median embedding'** and **'Cluster-based embedding'**.

Overall, for the 6 linear models we compared, the model Lasso Regression with best alpha has the lowest testing MSE. The BERT without empty text has a lower testing MSE than with empty text. And the BERT and demographic without empty text has a lower testing MSE than that does not contain the demographic data. There are still many other aspects we should consider to our models. Using the linear model, our predicting rating is continuous, but the true rating only has five levels, 1.0, 2.0, 3.0, 4.0 and 5.0. We could provide a **threshold** for our continuous predicting rating to transform them to 5 levels. What's more, we could use advanced models such as **random forest** and **Neural Network** for multiple classification.

Table 16: Mean Squared Errors (MSE) for different models

Model	Training MSE	Testing MSE
Standard linear model	1.8484	1.9053
Lasso regression with best alpha	1.8812	1.8438
Without empty BERT	0.0080	14.7393
With empty BERT	0.7671	571913.3233
Bert and demographic without empty	0.0000	3.5844
Bert and demographic with empty	0.4647	∞

9 Feature Importance for Coffee Shop

The **coffee shops** are 'Philz Coffee', 'Brothers K Coffeehouse', 'Pâtisserie Coralie', 'Evanston Games & Cafe' and the number of coffee shops is 4. The number of rating in the coffee shop data is 61. Then we used a linear model to predict rating for these 4 coffee shops. The result is:

- The shape for the final dataset is (61, 22) that is 61 reviews for 4 coffee shops.
- Training MSE: 1.0426
- Testing MSE: 1.2483

The top 5 features that has the largest weight of the linear model for the the coffee shops data were shown at table 17. And The top 5 features that has the smallest weight of the linear model for the the coffee shops data were shown at table 18. The variable with the highest **absolute coefficients** has the highest weight on the linear model. Smaller absolute coefficients mean less important and have a less weight on the linear model. The negative coefficients mean that the relationship between this feature and the 'Rating' are negative. So the demographic features selected are 'Has Children?_Yes', 'Northwestern Student?', 'Vegetarian?_unknown', 'Has Children?_No', and 'Marital Status_unknown'.

From the table 17, we can observe certain groups of people like or dislike the coffee. For people who has children, the coefficient is negative which means that they might tend to rate less and dislike the coffee shop. For Northwestern students or no children, the coefficient is positive, which means that this group of people may tend to rate higher and like the coffee shop.

Table 17: Top 5 variables that have the largest weight

Feature	Coefficients	Absolute Coefficients
Has Children?_Yes	-0.799090	0.799090
Northwestern Student?	0.585749	0.585749
Vegetarian?_unknown	-0.471785	0.471785
Has Children?_No	0.434038	0.434038
Marital Status_unknown	0.365052	0.365052

Features	**Coefficients**	**Absolute Coefficients**
Average Amount Spent_Low	-0.027466	0.027466
Birth Year	-0.028134	0.028134
Preferred Mode of Transport_Public Transit	-0.069143	0.069143
Marital Status_Married	-0.084605	0.084605
Height (in)	0.085108	0.085108

Table 18: Top 5 variables that have the smallest weight

We can also utilize step-wise include both forward and backward to select best features. The result features are 'Weight (lb)', 'Northwestern Student?', 'Vegetarian?_Yes', 'Vegetarian?_unknown', 'Has Children?_Yes', which they have common features as the table 17.

10 Interesting Finding

Statistic	Value
Mean	172.169161
Std	14.074959
Min	123.000000
25%	160.000000
50%	171.000000
75%	182.000000
Max	201.000000

Table 19: Summary statistics for Height (in)

From the summary of statsics of 'Height (in)', we can conclude that the reasonable unit for Height is cm instead of in. What's more, when I was doing EDA, I found that the spelling of the 'SIngle' in 'Martial Status' is wrong and it should be 'Single'.

11 Conclusion

In this report, we explored various techniques for building different recommendation systems for Evanston restaurants using a dataset containing reviewer, restaurant, and review information. We employed popularity matching, content-based filtering, natural language analysis, collaborative filtering, and predictive modeling to generate recommendations. Overall, this project provided valuable insights into the development of a recommendation system and demonstrated the benefits of incorporating natural language processing techniques in the analysis.