Restaurant Recommendation System:
A Look into Collaborative Filtering vs. Content Based Filtering

Rahul Rajeev

09/24/2023

DSC 680-T302

Professor Chase Denton

Introduction

The topic of this project is to develop a recommendation system for restaurants. I decided to try my hand at two different types of recommendation systems: one with collaborative filtering and the second with content-based filtering.

Research Question

Recommendation systems are intricate mechanisms used by commercial businesses to sell similar products or services. There are two common versions of recommendation systems: one using collaborative filtering and the other using content-based filtering. I decided to try my hand at both. Given a clean user ratings dataset, the collaborative filtering system can be completed quite easily. But the content-based filtering, as I soon found out, was a much more daunting task. In short, I wish to build the two types of recommendation systems and potentially add a user interface to make it more interactive.

Background

From the two restaurant recommendation systems I've used in Google Maps and Yelp didn't have much in terms of user preference. Usually, you were allowed to search for a certain cuisine, maybe the days they were open, and the price range. Building a content-based filtering system seemed like a great addition to the typical collaborative-filtering systems that the two websites already have. I believe the user should be able to input more preferences including parking availability, accessibility, alcohol availability, smoking permissions, ambience, etc.

Data Explanation

I found this data on the UCI Machine Learning repository. The dataset is titled "Restaurant & consumer data" and was posted in 2012 as a starting point for a similar recommendation system. While the dataset consists of restaurants mainly located in Mexico, it has the structure required to build a recommendation system for other restaurants internationally that could be added as time goes.

The entire dataset is split amongst 9 csv files each having attributes either concerning the restaurant or a user's preferences. About five of them were about the restaurants, four about users, and then one that had user reviews based on overall rating, food rating, and service rating for each restaurant. I decided to work with most of the restaurant files instead of the user files because I wanted the user to enter their own preferences instead of using pre-existing profiles with user id numbers. The five restaurant CSVs were about cuisine, parking, payment methods, hours of operation, and overall information. I used the review file to start working on a collaborative filtering recommendation system that asks the user whether they prefer overall rating, food rating, or service rating.

Visualizations

To understand some trends in the data I decided to create a couple of visualizations including a heat map using the latitude and longitude, a horizontal bar chart with the frequency of cuisines, and two pie charts for the frequencies of parking and payment methods.

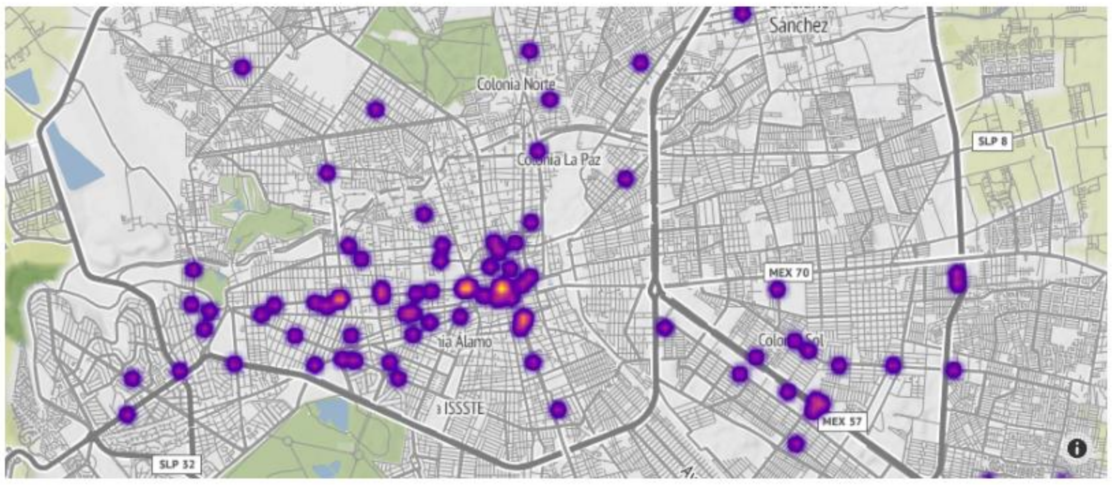


Figure 1: Locations of restaurants in San Lui Polosi

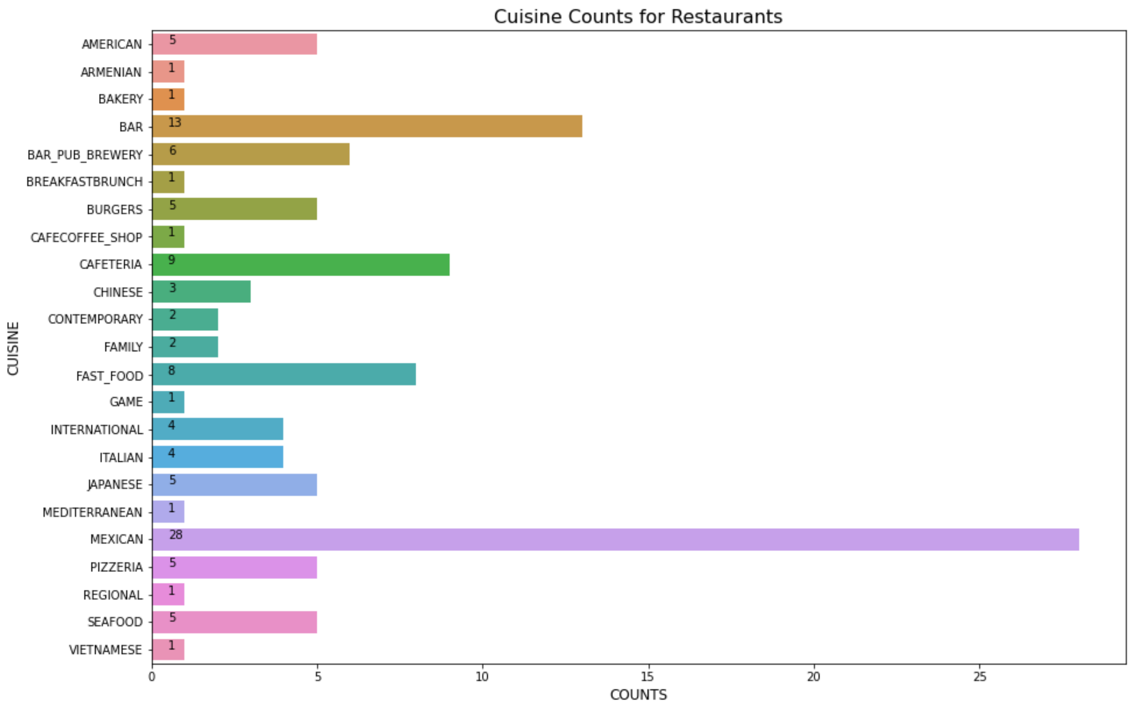


Figure 2: Cuisine Frequency Bar Chart

Most restaurants serve Mexican cuisine food, followed by restaurants that are bars, and then fast food. I was surprised by the variety of cuisine available since most of the restaurants are in the hearts of the cities. But that can be explained by the mesh of cultures in Mexico.

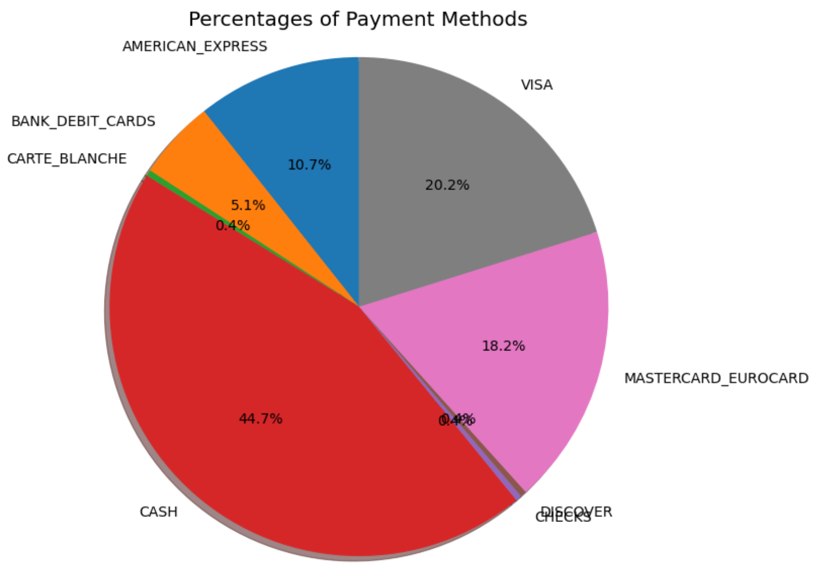


Figure 3: Pie Chart of Payment Methods

Most restaurants accept cash, followed by visa and master card. For some reason, some restaurants accept checks and Carte Blanche. I would expect most restaurants to be paid by cash especially if they are street-accessible restaurants and fast food.

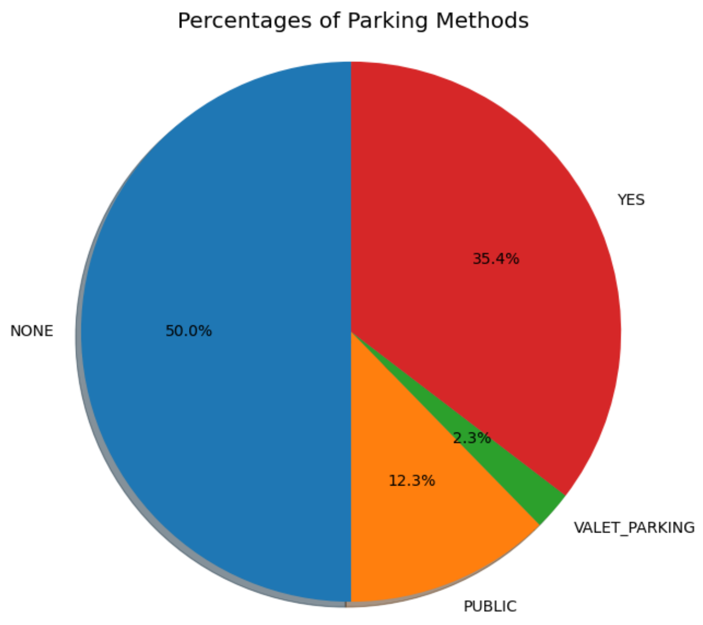
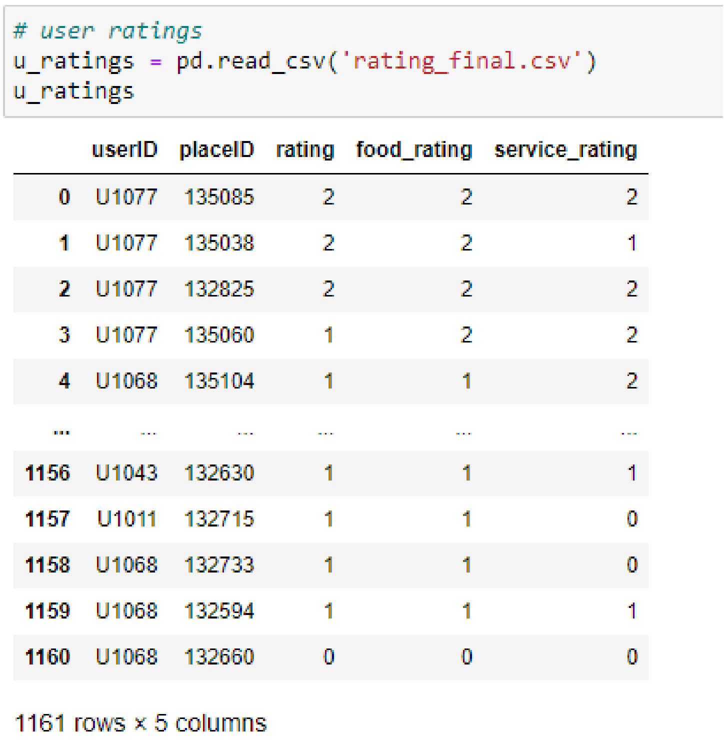


Figure 4: Pie Chart of Parking Methods

Most restaurants not having parking makes sense if the restaurants are in a dense city or in the middle of a streetway. Valet parking is a small section of parking for higher end restaurants. “Yes” and “public” could imply the same thing as parking is available regardless of the location of the restaurant.

The other features had much less variation than the ones above so creating visualizations for each of them didn’t seem necessary. However, if you are interested in seeing all of the possible responses, take a look at the unique responses dictionary in Appendix B. The appendices will also contain extensive code snippets with the mechanics for both recommendation systems and some of the formatting techniques.

Data Preparation and Formatting



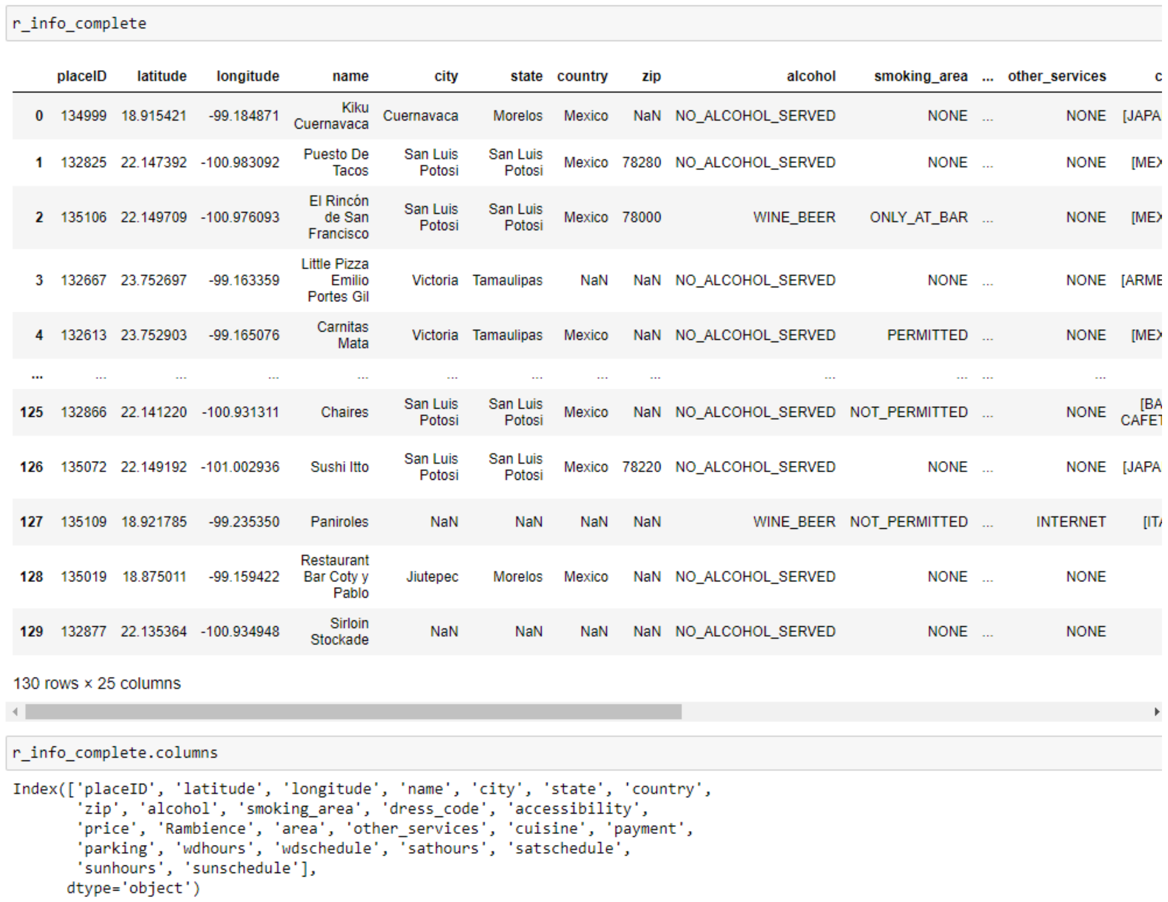
```
# user ratings
u_ratings = pd.read_csv('rating_final.csv')
u_ratings
```

	userID	placeID	rating	food_rating	service_rating
0	U1077	135085	2	2	2
1	U1077	135038	2	2	1
2	U1077	132825	2	2	2
3	U1077	135060	1	2	2
4	U1068	135104	1	1	2
...	...	...	...	...	...
1156	U1043	132630	1	1	1
1157	U1011	132715	1	1	0
1158	U1068	132733	1	1	0
1159	U1068	132594	1	1	1
1160	U1068	132660	0	0	0

1161 rows × 5 columns

Figure 5: Ratings dataset

For the collaborative-filtering method, I created three pivot tables of restaurant ID against user ID for review vectors. Each pivot table followed a specific type of rating either overall rating, food rating, or service rating. I used the pandas built in method for pivot table and saved the results as the data frames to be passed into the model.



```
r_info_complete
```

	placeID	latitude	longitude	name	city	state	country	zip	alcohol	smoking_area	...	other_services	c
0	134999	18.915421	-99.184871	Kiku Cuernavaca	Cuernavaca	Morelos	Mexico	NaN	NO_ALCOHOL_SERVED	NONE	...	NONE	[JAPA
1	132825	22.147392	-100.983092	Puesto De Tacos	San Luis Potosi	San Luis Potosi	Mexico	78280	NO_ALCOHOL_SERVED	NONE	...	NONE	[MEX
2	135106	22.149709	-100.976093	El Rincón de San Francisco	San Luis Potosi	San Luis Potosi	Mexico	78000	WINE_BEER	ONLY_AT_BAR	...	NONE	[MEX
3	132667	23.752697	-99.163359	Little Pizza Emilio Portes Gil	Victoria	Tamaulipas	NaN	NaN	NO_ALCOHOL_SERVED	NONE	...	NONE	[ARME
4	132613	23.752903	-99.165076	Carnitas Mata	Victoria	Tamaulipas	Mexico	NaN	NO_ALCOHOL_SERVED	PERMITTED	...	NONE	[MEX
...	...	...	...	...	...	...	...	...	...	...	...	...	
125	132866	22.141220	-100.931311	Chaires	San Luis Potosi	San Luis Potosi	Mexico	NaN	NO_ALCOHOL_SERVED	NOT_PERMITTED	...	NONE	[BA CAFET
126	135072	22.149192	-101.002936	Sushi Itto	San Luis Potosi	San Luis Potosi	Mexico	78220	NO_ALCOHOL_SERVED	NONE	...	NONE	[JAPA
127	135109	18.921785	-99.235350	Paniroles	NaN	NaN	NaN	NaN	WINE_BEER	NOT_PERMITTED	...	INTERNET	[IT
128	135019	18.875011	-99.159422	Restaurant Bar Coty y Pablo	Jiutepec	Morelos	Mexico	NaN	NO_ALCOHOL_SERVED	NONE	...	NONE	
129	132877	22.135364	-100.934948	Sirloin Stockade	NaN	NaN	NaN	NaN	NO_ALCOHOL_SERVED	NONE	...	NONE	

130 rows × 25 columns

```
r_info_complete.columns
```

```
Index(['placeID', 'latitude', 'longitude', 'name', 'city', 'state', 'country',
       'zip', 'alcohol', 'smoking_area', 'dress_code', 'accessibility',
       'price', 'Rambience', 'area', 'other_services', 'cuisine', 'payment',
       'parking', 'wdhours', 'wdschedule', 'sathours', 'satschedule',
       'sunhours', 'sunschedule'],
      dtype='object')
```

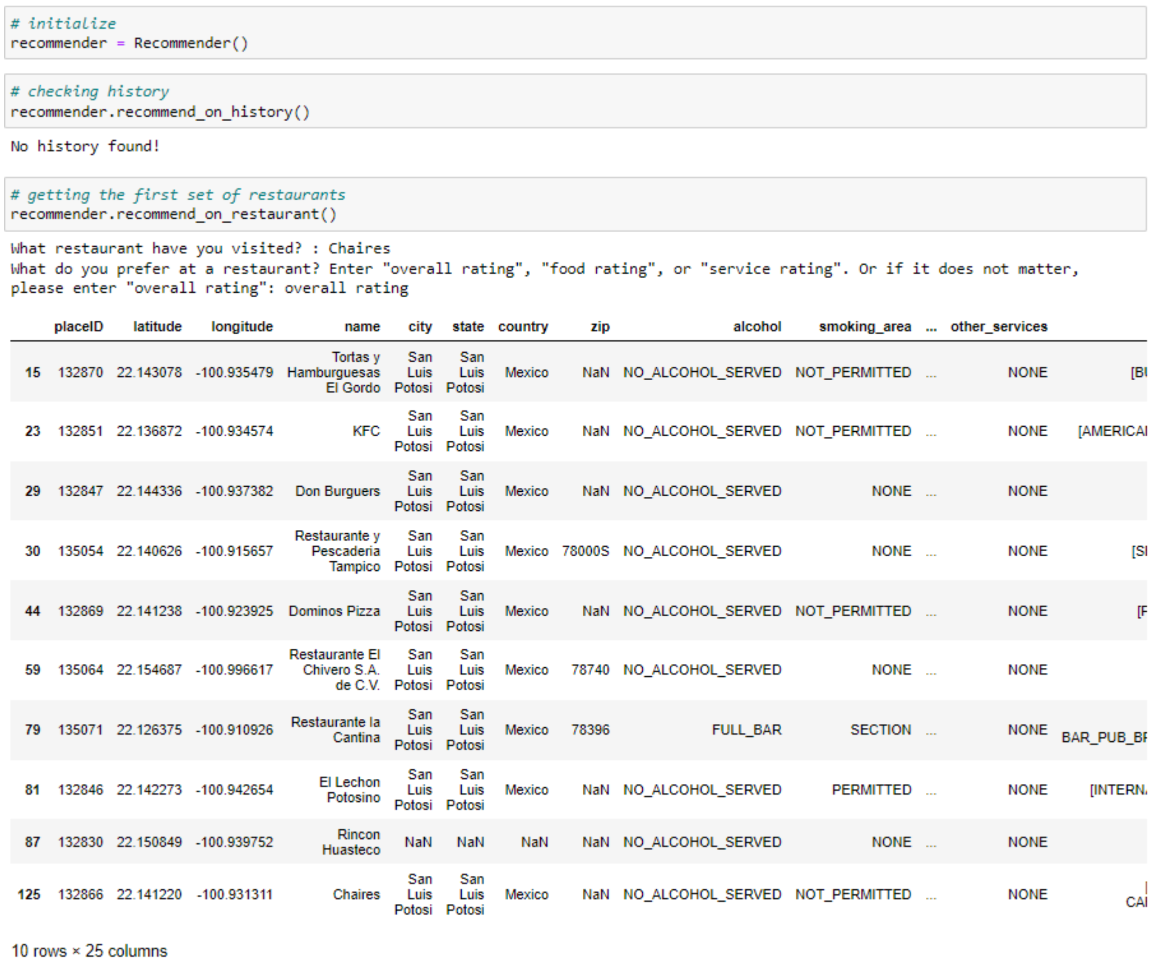
Figure 6: Formatted data frame for restaurant information

The data preparation method for the content-based filtering involved formatting the columns of data into lists of strings if necessary. The main way I decided to format strings was to replace any hyphens or spaces with underscores and capitalize everything for visibility. The cuisine dataset, the parking dataset, and the payment dataset were all "exploded" forms of some original dataset. Meaning they had multiple rows for each restaurant for extended values for each dataset. To format these datasets, I grouped by restaurant ID and created a list of the elements. The hours dataset required much more work, as I had to first splice the hours of operation string column into lists of intervals, and then create columns describing what days the restaurant was open or closed. The restaurant info dataset mainly had to have the strings formatted. After joining the data frames together, I had to replace the missing elements for the list columns with empty lists for search methods defined later on.

For passing this data into the model, I had to use pandas get dummies to transform the categorical columns into sparse data. I had to use the original cleaned versions of the parking, payment, and cuisine datasets as creating dummy variables from lists wasn’t possible. I also had to make sure that these three data sets were filtered by restaurants only included in the r_info_complete data set to have the most information and customization for preferences as possible. Then I added the remaining columns. The result included parking, payment, cuisine, weekday schedule, Saturday schedule, Sunday schedule, ambience, alcohol, smoking area, area, dress code, accessibility, price, and other miscellaneous services. After, it was on to creating the systems themselves.

Methods

Euclidean distance doesn’t work well in higher dimensions for sparse data so instead of clustering higher dimensional points, I instead delegated the rows of data to vectors and used the cosine similarity metric in Nearest Neighbors for the recommendation system (see Appendix A). I created a recommender class that had the a recommend_on_restaurant method, and a recommend_on_history method, and a show_history method that shows all restaurants visited so far. The recommend_on_restaurant method takes in a user input of a restaurant name, converts it to a restaurant id, and then runs the Nearest Neighbor model and returns the first ten restaurants closest to the entered restaurant vector. The recommend_on_history method looks at previous restaurants, calculates an average of the restaurant ids, and then recommends ten restaurants closest. The last method show_history returns the list of restaurants entered in previous calls. The combination of these methods could potentially be highlighted in some kind of user interface, although since I am not very familiar with it, I could possibly save that for a later time.



```
# initialize
recommender = Recommender()
```

```
# checking history
recommender.recommend_on_history()
```

```
No history found!
```

```
# getting the first set of restaurants
recommender.recommend_on_restaurant()
```

```
What restaurant have you visited? : Chaires
What do you prefer at a restaurant? Enter "overall rating", "food rating", or "service rating". Or if it does not matter,
please enter "overall rating": overall rating
```

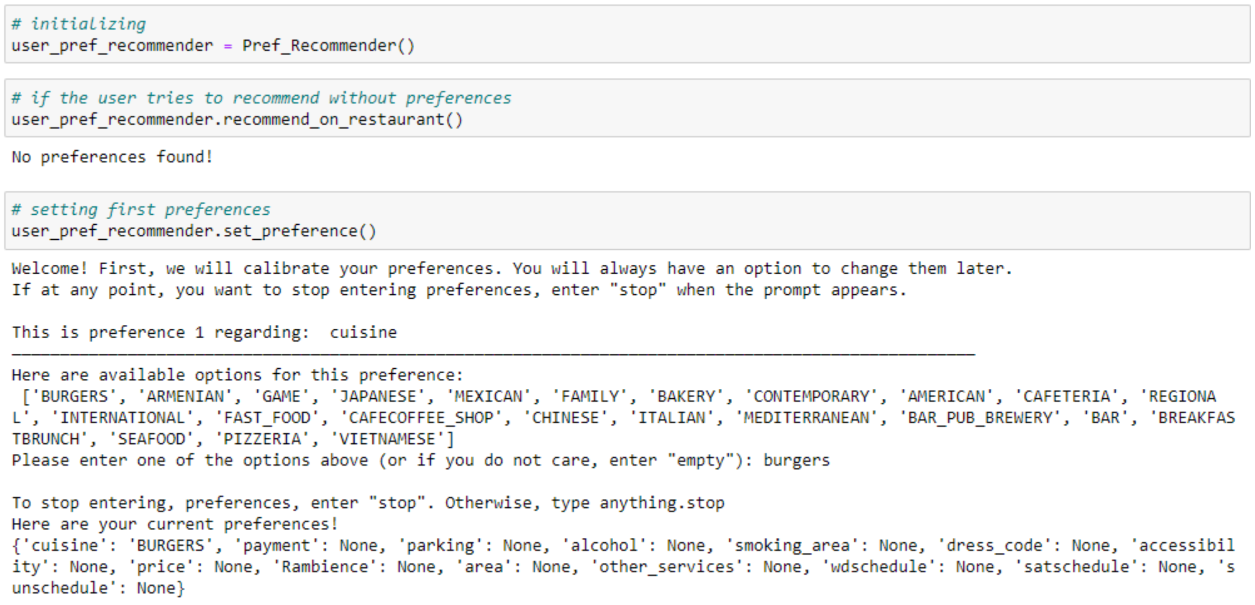
	placeID	latitude	longitude	name	city	state	country	zip	alcohol	smoking_area	...	other_services	
15	132870	22.143078	-100.935479	Tortas y Hamburguesas El Gordo	San Luis Potosi	San Luis Potosi	Mexico	NaN	NO_ALCOHOL_SERVED	NOT_PERMITTED	...	NONE	[BU
23	132851	22.136872	-100.934574	KFC	San Luis Potosi	San Luis Potosi	Mexico	NaN	NO_ALCOHOL_SERVED	NOT_PERMITTED	...	NONE	[AMERICAI
29	132847	22.144336	-100.937382	Don Burguers	San Luis Potosi	San Luis Potosi	Mexico	NaN	NO_ALCOHOL_SERVED	NONE	...	NONE	
30	135054	22.140626	-100.915657	Restaurante y Pescaderia Tampico	San Luis Potosi	San Luis Potosi	Mexico	78000S	NO_ALCOHOL_SERVED	NONE	...	NONE	[SI
44	132869	22.141238	-100.923925	Dominos Pizza	San Luis Potosi	San Luis Potosi	Mexico	NaN	NO_ALCOHOL_SERVED	NOT_PERMITTED	...	NONE	[F
59	135064	22.154687	-100.996617	Restaurante El Chivero S.A. de C.V.	San Luis Potosi	San Luis Potosi	Mexico	78740	NO_ALCOHOL_SERVED	NONE	...	NONE	
79	135071	22.126375	-100.910926	Restaurante la Cantina	San Luis Potosi	San Luis Potosi	Mexico	78396	FULL_BAR	SECTION	...	NONE	BAR_PUB_BI
81	132846	22.142273	-100.942654	El Lechon Potosino	San Luis Potosi	San Luis Potosi	Mexico	NaN	NO_ALCOHOL_SERVED	PERMITTED	...	NONE	[INTERN.
87	132830	22.150849	-100.939752	Rincon Huasteco	NaN	NaN	NaN	NaN	NO_ALCOHOL_SERVED	NONE	...	NONE	
125	132866	22.141220	-100.931311	Chaires	San Luis Potosi	San Luis Potosi	Mexico	NaN	NO_ALCOHOL_SERVED	NOT_PERMITTED	...	NONE	CAI

10 rows × 25 columns

Figure 7: Example results of the three methods in collaborative filtering.

The content-based filtering uses the same methodology save for the type of user input. Instead of directly entering a restaurant name that the user has visited or wants to visit, it instead takes in a set of preferences stored in a dictionary. Then the restaurant with those specific preferences is found. That will be the equivalent of entering a restaurant from the collaborative filtering system. Then it should have similar methods: recommend_on_restaurant, recommend_on_history, and the show_history method.

In addition to the similar methods in the collaborative filtering method, I will also add a couple of more methods. User_preference, edit_preference, and filter_time will be the names. The general purpose of the first two methods is for the user to enter in a set of preferences which is saved in the class and can be edited further whenever the user requires. The filter_time method takes the recommended restaurants or the filtered restaurants given after sifting through the entire restaurant info data frame by day of the week and hours. See Appendix B for code snippets.



```
# initializing
user_pref_recommender = Pref_Recommender()
```

```
# if the user tries to recommend without preferences
user_pref_recommender.recommend_on_restaurant()
```

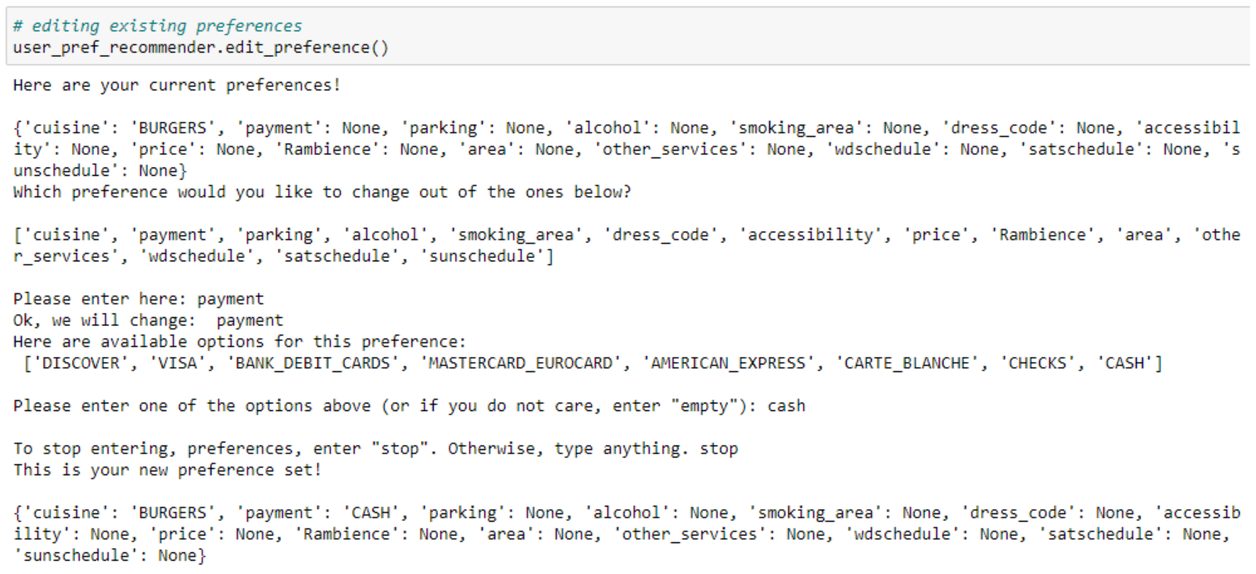
```
No preferences found!
```

```
# setting first preferences
user_pref_recommender.set_preference()
```

```
Welcome! First, we will calibrate your preferences. You will always have an option to change them later.
If at any point, you want to stop entering preferences, enter "stop" when the prompt appears.

This is preference 1 regarding:  cuisine
________________________________________________________________________________

Here are available options for this preference:
 ['BURGERS', 'ARMENIAN', 'GAME', 'JAPANESE', 'MEXICAN', 'FAMILY', 'BAKERY', 'CONTEMPORARY', 'AMERICAN', 'CAFETERIA', 'REGIONA
L', 'INTERNATIONAL', 'FAST_FOOD', 'CAFECOFFEE_SHOP', 'CHINESE', 'ITALIAN', 'MEDITERRANEAN', 'BAR_PUB_BREWERY', 'BAR', 'BREAKFAS
TBRUNCH', 'SEAFOOD', 'PIZZERIA', 'VIETNAMESE']
Please enter one of the options above (or if you do not care, enter "empty"): burgers

To stop entering, preferences, enter "stop". Otherwise, type anything.stop
Here are your current preferences!
{'cuisine': 'BURGERS', 'payment': None, 'parking': None, 'alcohol': None, 'smoking_area': None, 'dress_code': None, 'accessibil
ity': None, 'price': None, 'Rambience': None, 'area': None, 'other_services': None, 'wdschedule': None, 'satschedule': None, 's
unschedule': None}
```

Figure 8: Example results of methods in content-based filtering (Part 1)

First, I must initialize the recommender. If the user initially tries to get recommendations without setting preferences, the recommender will error out and return a print statement. I can call the set_preference method to enter the user input feed back loop to fill out the preference dictionary. As you can see, I stopped the loop after one preference, but you can enter up to preferences in the Sunday schedule. Although I don’t think you would have much success with many preferences and being super specific for this small dataset of 100 restaurants, but if I were to include restaurants internationally, specific restaurants could exist.

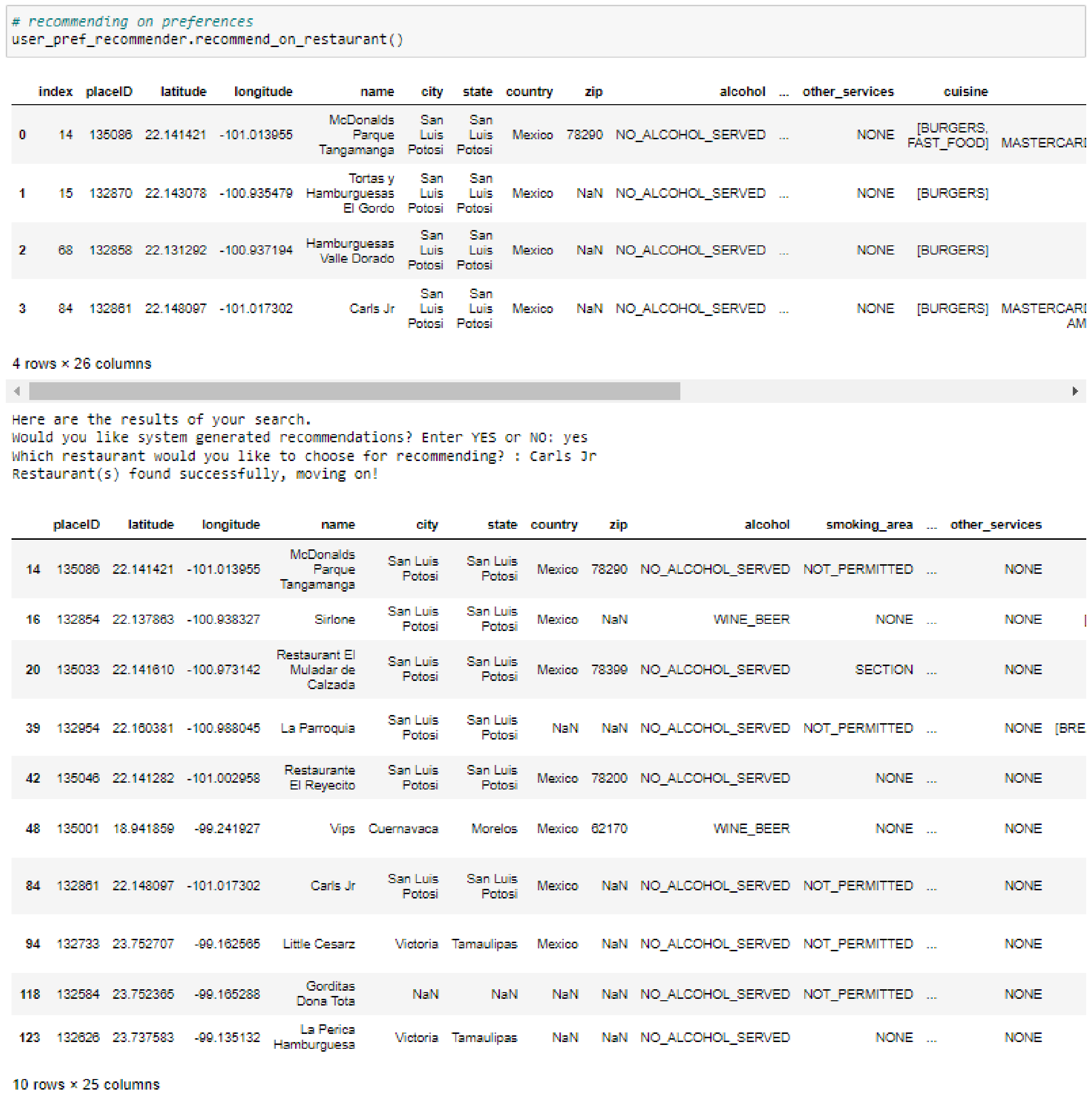


```
# editing existing preferences
user_pref_recommender.edit_preference()
```

```
Here are your current preferences!

{'cuisine': 'BURGERS', 'payment': None, 'parking': None, 'alcohol': None, 'smoking_area': None, 'dress_code': None, 'accessibil
ity': None, 'price': None, 'Rambience': None, 'area': None, 'other_services': None, 'wdschedule': None, 'satschedule': None, 's
unschedule': None}
Which preference would you like to change out of the ones below?

['cuisine', 'payment', 'parking', 'alcohol', 'smoking_area', 'dress_code', 'accessibility', 'price', 'Rambience', 'area', 'othe
r_services', 'wdschedule', 'satschedule', 'sunschedule']

Please enter here: payment
Ok, we will change:  payment
Here are available options for this preference:
 ['DISCOVER', 'VISA', 'BANK_DEBIT_CARDS', 'MASTERCARD_EUROCARD', 'AMERICAN_EXPRESS', 'CARTE_BLANCHE', 'CHECKS', 'CASH']

Please enter one of the options above (or if you do not care, enter "empty"): cash

To stop entering, preferences, enter "stop". Otherwise, type anything. stop
This is your new preference set!

{'cuisine': 'BURGERS', 'payment': 'CASH', 'parking': None, 'alcohol': None, 'smoking_area': None, 'dress_code': None, 'accessib
ility': None, 'price': None, 'Rambience': None, 'area': None, 'other_services': None, 'wdschedule': None, 'satschedule': None,
'sunschedule': None}
```

Figure 9: Example results of methods in content-based filtering (Part 2)

Next I called the edit preference which asks the user which preference they would like to change so the user can pick it and an appropriate response out of all the available ones in the r_info_complete data frame. Here I adjusted the payment method to change from an empty preference to “CASH”. Next, the recommend_on_restaurant method is called. Before it performs the search, there is a function that converts the dictionary to a “cleaned” version where only the preferences that aren’t none are selected. Then it is passed into a another function find_rows which checks whether rows contain all the necessary preferences with a counter (see appendix B). While the number of restaurant ids returned with the preference filter is zero, the user is prompted to edit their preferences until the list is greater than zero. So if the user has an ultra-specific set of preferences and finds no luck finding a restaurant that works for them, they will be forced to change each and almost every single one of their preferences until it returns at least one restaurant.



```
# recommending on preferences
user_pref_recommender.recommend_on_restaurant()
```

	index	placeID	latitude	longitude	name	city	state	country	zip	alcohol	...	other_services	cuisine	
0	14	135086	22.141421	-101.013955	McDonalds Parque Tangamanga	San Luis Potosi	San Luis Potosi	Mexico	78290	NO_ALCOHOL_SERVED	...	NONE	[BURGERS, FAST_FOOD]	MASTERCARI
1	15	132870	22.143078	-100.935479	Tortas y Hamburguesas El Gordo	San Luis Potosi	San Luis Potosi	Mexico	NaN	NO_ALCOHOL_SERVED	...	NONE	[BURGERS]	
2	68	132858	22.131292	-100.937194	Hamburguesas Valle Dorado	San Luis Potosi	San Luis Potosi	Mexico	NaN	NO_ALCOHOL_SERVED	...	NONE	[BURGERS]	
3	84	132861	22.148097	-101.017302	Carls Jr	San Luis Potosi	San Luis Potosi	Mexico	NaN	NO_ALCOHOL_SERVED	...	NONE	[BURGERS]	MASTERCARI AM

4 rows × 26 columns

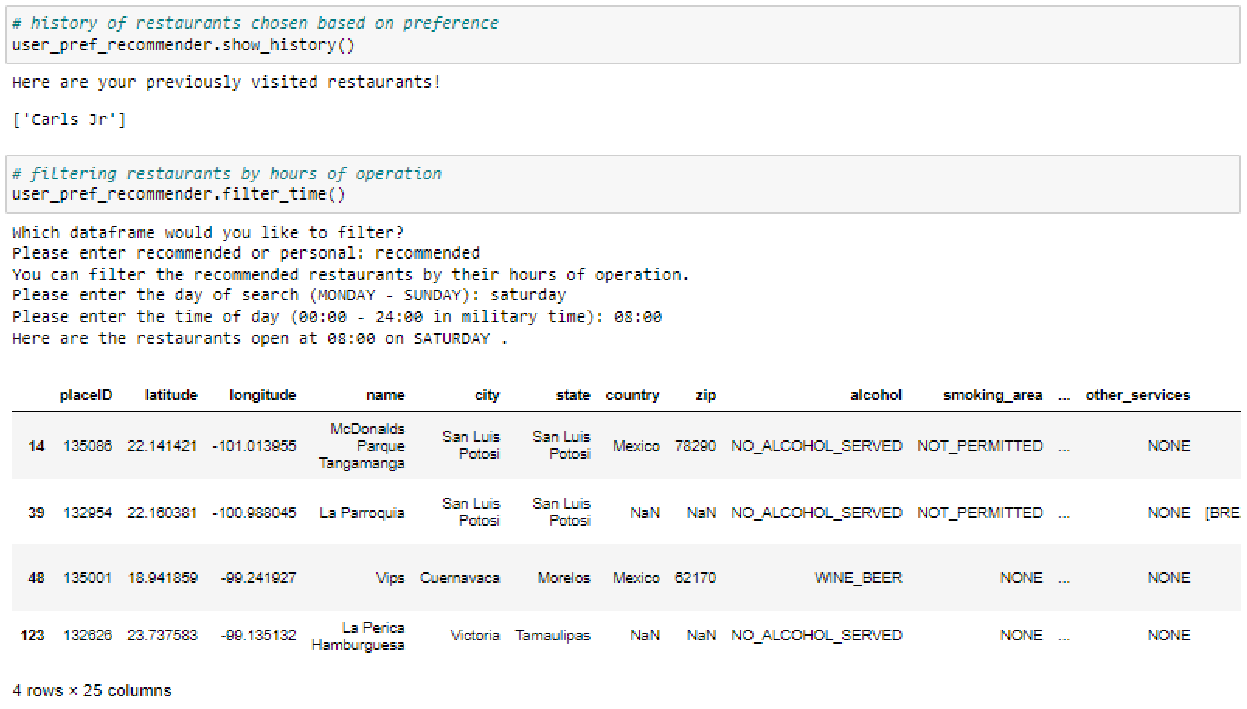
```
Here are the results of your search.
Would you like system generated recommendations? Enter YES or NO: yes
Which restaurant would you like to choose for recommending? : Carls Jr
Restaurant(s) found successfully, moving on!
```

	placeID	latitude	longitude	name	city	state	country	zip	alcohol	smoking_area	...	other_services	
14	135086	22.141421	-101.013955	McDonalds Parque Tangamanga	San Luis Potosi	San Luis Potosi	Mexico	78290	NO_ALCOHOL_SERVED	NOT_PERMITTED	...	NONE	
16	132854	22.137863	-100.938327	Sirlone	San Luis Potosi	San Luis Potosi	Mexico	NaN	WINE_BEER	NONE	...	NONE	|
20	135033	22.141610	-100.973142	Restaurant El Muladar de Calzada	San Luis Potosi	San Luis Potosi	Mexico	78399	NO_ALCOHOL_SERVED	SECTION	...	NONE	
39	132954	22.160381	-100.988045	La Parroquia	San Luis Potosi	San Luis Potosi	NaN	NaN	NO_ALCOHOL_SERVED	NOT_PERMITTED	...	NONE	[BRE
42	135046	22.141282	-101.002958	Restaurante El Reyecito	San Luis Potosi	San Luis Potosi	Mexico	78200	NO_ALCOHOL_SERVED	NONE	...	NONE	
48	135001	18.941859	-99.241927	Vips	Cuernavaca	Morelos	Mexico	62170	WINE_BEER	NONE	...	NONE	
84	132861	22.148097	-101.017302	Carls Jr	San Luis Potosi	San Luis Potosi	Mexico	NaN	NO_ALCOHOL_SERVED	NOT_PERMITTED	...	NONE	
94	132733	23.752707	-99.162565	Little Cesarz	Victoria	Tamaulipas	Mexico	NaN	NO_ALCOHOL_SERVED	NOT_PERMITTED	...	NONE	
118	132584	23.752365	-99.165288	Gorditas Dona Tota	NaN	NaN	NaN	NaN	NO_ALCOHOL_SERVED	NOT_PERMITTED	...	NONE	
123	132626	23.737583	-99.135132	La Perica Hamburguesa	Victoria	Tamaulipas	NaN	NaN	NO_ALCOHOL_SERVED	NONE	...	NONE	

10 rows × 25 columns

Figure 10: Example results of methods in content-based filtering (Part 3)

The first dataframe is the filtered dataframe using user preferences. Here the user has the option to either stick with the set of restaurants they have or try their luck at system recommended set of restaurants that won’t exactly follow their specific preferences. You would pick a restaurant out of the filtered one, enter the name, and it will provide results using the same methodology as the collaborative filtering except for the dummy data frame containing restaurant ids their attributes.



```
# history of restaurants chosen based on preference
user_pref_recommender.show_history()
```

```
Here are your previously visited restaurants!

['Carls Jr']
```

```
# filtering restaurants by hours of operation
user_pref_recommender.filter_time()
```

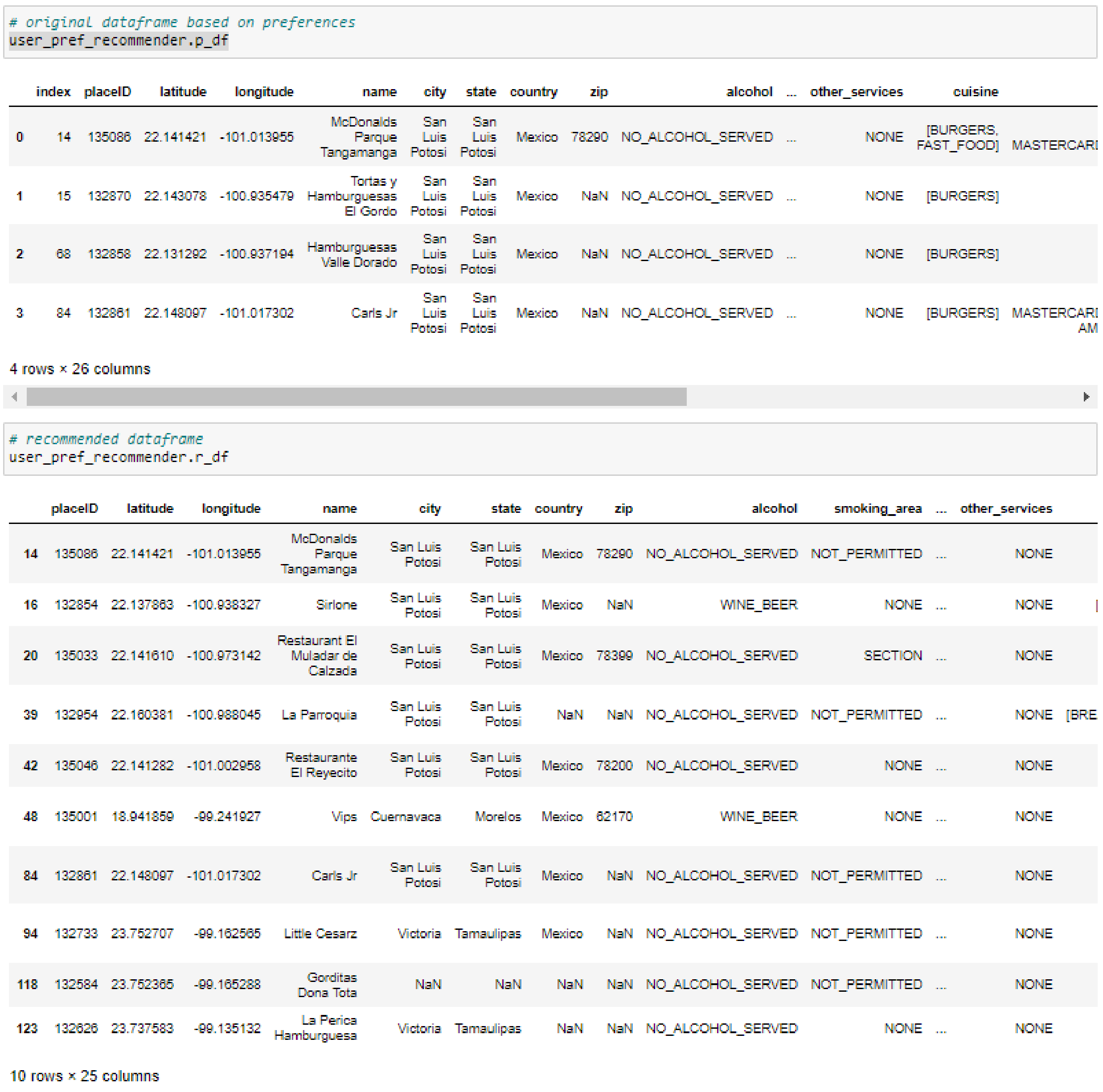
```
Which dataframe would you like to filter?
Please enter recommended or personal: recommended
You can filter the recommended restaurants by their hours of operation.
Please enter the day of search (MONDAY - SUNDAY): saturday
Please enter the time of day (00:00 - 24:00 in military time): 08:00
Here are the restaurants open at 08:00 on SATURDAY .
```

	placeID	latitude	longitude	name	city	state	country	zip	alcohol	smoking_area	...	other_services	
14	135086	22.141421	-101.013955	McDonalds Parque Tangamanga	San Luis Potosi	San Luis Potosi	Mexico	78290	NO_ALCOHOL_SERVED	NOT_PERMITTED	...	NONE	
39	132954	22.160381	-100.988045	La Parroquia	San Luis Potosi	San Luis Potosi	NaN	NaN	NO_ALCOHOL_SERVED	NOT_PERMITTED	...	NONE	[BRE
48	135001	18.941859	-99.241927	Vips	Cuernavaca	Morelos	Mexico	62170	WINE_BEER	NONE	...	NONE	
123	132626	23.737583	-99.135132	La Perica Hamburguesa	Victoria	Tamaulipas	NaN	NaN	NO_ALCOHOL_SERVED	NONE	...	NONE	

4 rows × 25 columns

Figure 11: Example results of methods in content-based filtering (Part 4)

The history method can be called to access all of the restaurants used for recommendations. The recommend_on_history method is also available from the collaborative filtering method that takes an average of the restaurant ids and returns the ten closest restaurants. I didn’t include that method in this picture as I displayed the usage in the collaborative filtering recommendation system results. The filter by time asks for user preference of day and time of day open and then filters either the preferred restaurants data frame or the recommended restaurants data frame. That choice is also made from the user using an input. In this example I decided to filter the recommended set for restaurants open on Saturday at 08:00 AM. I found that having the times in the military format and integer form the easiest to work with for searching methods.



```
# original dataframe based on preferences
user_pref_recommender.p_df
```

	index	placeID	latitude	longitude	name	city	state	country	zip	alcohol	...	other_services	cuisine	
0	14	135086	22.141421	-101.013955	McDonalds Parque Tangamanga	San Luis Potosi	San Luis Potosi	Mexico	78290	NO_ALCOHOL_SERVED	...	NONE	[BURGERS, FAST_FOOD]	MASTERCARI
1	15	132870	22.143078	-100.935479	Tortas y Hamburguesas El Gordo	San Luis Potosi	San Luis Potosi	Mexico	NaN	NO_ALCOHOL_SERVED	...	NONE	[BURGERS]	
2	68	132858	22.131292	-100.937194	Hamburguesas Valle Dorado	San Luis Potosi	San Luis Potosi	Mexico	NaN	NO_ALCOHOL_SERVED	...	NONE	[BURGERS]	
3	84	132861	22.148097	-101.017302	Carls Jr	San Luis Potosi	San Luis Potosi	Mexico	NaN	NO_ALCOHOL_SERVED	...	NONE	[BURGERS]	MASTERCARI AM

4 rows × 26 columns

```
# recommended dataframe
user_pref_recommender.r_df
```

	placeID	latitude	longitude	name	city	state	country	zip	alcohol	smoking_area	...	other_services	
14	135086	22.141421	-101.013955	McDonalds Parque Tangamanga	San Luis Potosi	San Luis Potosi	Mexico	78290	NO_ALCOHOL_SERVED	NOT_PERMITTED	...	NONE	
16	132854	22.137863	-100.938327	Sirlone	San Luis Potosi	San Luis Potosi	Mexico	NaN	WINE_BEER	NONE	...	NONE	|
20	135033	22.141610	-100.973142	Restaurant El Muladar de Calzada	San Luis Potosi	San Luis Potosi	Mexico	78399	NO_ALCOHOL_SERVED	SECTION	...	NONE	
39	132954	22.160381	-100.988045	La Parroquia	San Luis Potosi	San Luis Potosi	NaN	NaN	NO_ALCOHOL_SERVED	NOT_PERMITTED	...	NONE	[BRE
42	135046	22.141282	-101.002958	Restaurante El Reyecito	San Luis Potosi	San Luis Potosi	Mexico	78200	NO_ALCOHOL_SERVED	NONE	...	NONE	
48	135001	18.941859	-99.241927	Vips	Cuernavaca	Morelos	Mexico	62170	WINE_BEER	NONE	...	NONE	
84	132861	22.148097	-101.017302	Carls Jr	San Luis Potosi	San Luis Potosi	Mexico	NaN	NO_ALCOHOL_SERVED	NOT_PERMITTED	...	NONE	
94	132733	23.752707	-99.162565	Little Cesarz	Victoria	Tamaulipas	Mexico	NaN	NO_ALCOHOL_SERVED	NOT_PERMITTED	...	NONE	
118	132584	23.752365	-99.165288	Gorditas Dona Tota	NaN	NaN	NaN	NaN	NO_ALCOHOL_SERVED	NOT_PERMITTED	...	NONE	
123	132626	23.737583	-99.135132	La Perica Hamburguesa	Victoria	Tamaulipas	NaN	NaN	NO_ALCOHOL_SERVED	NONE	...	NONE	

10 rows × 25 columns

Figure 12: Example results of methods in content-based filtering (Part 5)

Lastly the user has the option to either call their preference data frame or recommended data frame separately. In order for these to change, the user would either have to change their preferences or recommend on a new restaurant or based on history. While the notebook doesn't display this information in full, with the help of an app interface, the user will be able to view all attributes simultaneously a lot easier. To follow the full dataframes and process, please check out the notebook file.

Analysis

I used one type of model for both recommendation systems, so there wasn’t any analysis to be done. Once I finish what’s left in the project, I could research into different methods for recommendation systems and try applying them. I will need to perform research about metrics of ranking models against each other and possibly reduce the time complexity of the code down. There are a lot of if and while loops in the code that may run into issues like timing out when using a larger dataset.

Conclusion

As the last test cell was run, I was thinking about how much effort and time went into this project. Since I tried to make it as user friendly as possible, there were a lot of user input feedback loops that had to be secure. Avoiding misspells, spaces, and the user inputting values outside of the existing list and then moving on to the data formatting struggles. There was a lot of data mis-formatted, missing, and just filled with random symbols. Correcting everything by hand took a while, and I used python to clear up general patterns. The super specific ones had to be done on the excel sheet. I couldn't get around to formatting the names as I wasn't sure if the names of the restaurants incorporated lower case and upper-case characters in real life. Then came formatting the schedule data frame. Splitting the hours column which initially were strings into integer lists so that I can easily search for restaurants open at that time and day.

As far as the recommendation systems go, the collaborative filtering was very easily accomplished since I had prior experience creating one as an assignment in DSC 630. I was able to get that finished in the very first week of this project. Everything after was done for the user preference filtering which took a lot longer to get finished. Countless nights were spent figuring out the best ways to bring my pseudocode into fruition. I had lots of ideas, some cut, and some successfully accomplished. The final user preference recommendation system can ask the user for their preferences, search for restaurants with those specific preferences, and if none exist, asks them to edit until something does work.

Assumptions

Coming into the project, I didn't have many assumptions about the project other than the formatting of the data. Turns out there were a lot of issues such as having string values in different forms like abbreviations, lowercase vs. uppercase, and extra punctuation. Formatting the data was a large time commitment, of which could have been instead focused on the actual system. But formatting and storing the data into various structures are all part of the data science process, so I didn't mind it. Although I was a bit intimidated at first because there were many helper functions created to format bits and pieces of the strings.

The other assumption I had made was that there would be complete information for each of the restaurants. While there were about 760 unique instances of restaurants for the cuisine, and several lower for each other dataset, the complete info dataset had the lowest of around 130 instances. To make the content-filtered system as accurate as possible, it would have been nice to have information about all the 760 unique instances.

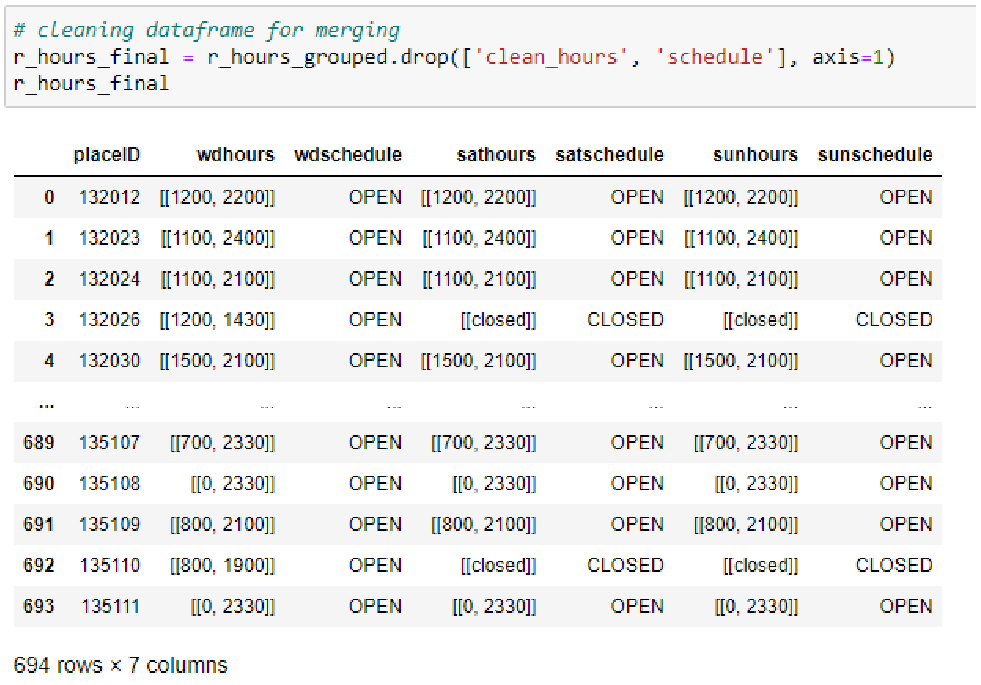
Limitations

The primary limitation was mentioned in the assumptions section in regard to the amount of additional information available about each restaurant. Only the 130 restaurants with complete information can be viewed for users with very specific preferences. The rest are available only for generalized preferences in payment, parking, cuisine, and hours available.

The second limitation is that restaurants are only available in Mexico and South America. This means that many of the options that exist there could possibly not exist here, or information is not readily available. The model is only built on this specific set of restaurants so generalizing the model is not possible now.

Challenges

The main challenges I faced were data formatting. I had already created a recommendation system before, so the logic of it made sense to me. Since there was data missing, in inconsistent forms, or I had to perform data engineering, most of the time and effort went into data formatting. Missing data was dealt with by replacing '?' strings with NaN values. The other data formatting process I had to run through was mentioned in the data preparation and explanation section: replacing any hyphens or spaces with underscores and capitalizing everything for visibility. There were some columns that were dropped due to a lack of variety or just mostly missing values. Later, I had to perform data engineering, most of which was done on the hours of operation dataset.



```
# cleaning dataframe for merging
r_hours_final = r_hours_grouped.drop(['clean_hours', 'schedule'], axis=1)
r_hours_final
```

	placeID	wdhours	wdschedule	sathours	satschedule	sunhours	sunschedule
0	132012	[[1200, 2200]]	OPEN	[[1200, 2200]]	OPEN	[[1200, 2200]]	OPEN
1	132023	[[1100, 2400]]	OPEN	[[1100, 2400]]	OPEN	[[1100, 2400]]	OPEN
2	132024	[[1100, 2100]]	OPEN	[[1100, 2100]]	OPEN	[[1100, 2100]]	OPEN
3	132026	[[1200, 1430]]	OPEN	[[closed]]	CLOSED	[[closed]]	CLOSED
4	132030	[[1500, 2100]]	OPEN	[[1500, 2100]]	OPEN	[[1500, 2100]]	OPEN
...	...	...	...	...	...	...	...
689	135107	[[700, 2330]]	OPEN	[[700, 2330]]	OPEN	[[700, 2330]]	OPEN
690	135108	[[0, 2330]]	OPEN	[[0, 2330]]	OPEN	[[0, 2330]]	OPEN
691	135109	[[800, 2100]]	OPEN	[[800, 2100]]	OPEN	[[800, 2100]]	OPEN
692	135110	[[800, 1900]]	OPEN	[[closed]]	CLOSED	[[closed]]	CLOSED
693	135111	[[0, 2330]]	OPEN	[[0, 2330]]	OPEN	[[0, 2330]]	OPEN

694 rows × 7 columns

Figure 15: Formatted hours dataset

For example, the hours dataset had a column which referenced hours of operation in a semicolon split form. I then had to split that string by semicolon, split them again by the colon marker, and then convert the string numbers to integer form to work towards having an option for the user to filter restaurants by time. In addition to the hourly schedule, I then took this data and iterated through each range of hours to figure out whether the restaurant was open on the weekdays or the weekend. Of course, this work could have been done later since I don't think I'll be able to finish the filtering by time or for that matter, filter by location, by the deadline of this project. But I wanted to get my hands dirty with data formatting now rather than later so I could move forward with plans if I do decide to pick this project up later.

Additional Applications

Moving forward with the project in the next week and possibly in the future, I believe that this recommendation system should be used by any industry interested in helping the users get restaurants recommended for wherever they live, whatever time, and their unique set of preferences. This could include Google Maps and Yelp. On the other hand, a similar pair of recommendation systems could be built for other objects such as books, movies, etcetera. I think one I am interested in would be the books recommendation system as I would like to have recommendations for people in a group and their genre preferences.

Recommendations

For future recommendations, there are two primary objectives to complete. They are implementing the filtering restaurant results by closest location and having a user interface.

The closest location filter can be done given the user input coordinates and compare them using Euclidean distance to the coordinates of the restaurant given in the dataset. Unfortunately, again we run into the limitation of only the 130 restaurants with coordinate information, so if the user's preferences are specific enough to avoid these 130 restaurants, then it may not return the best results. I think the closest location filter should only be applied once the dataset is generalized to a set of international restaurants outside of Mexico.

The user interface is something I've referred to in the project proposal as well as earlier in this report. I've been wanting to achieve it but with the lack of knowledge and the time constraint to move on to other projects, I will have to hold off on this one till the future.

The primary benefit of having user input is the periodic cleaning of printed text created by the recommendation systems. The back tracking of the user in a notebook is very unclean as it requires a lot of text to be displayed and the user would feel crowded with all their choices on the screen. I believe having a nice overhead of your preferences as you choose and edit them instead of printing them every single time would be a great choice in a user interface. Secondly, with a user interface it would be possible to autocorrect the users and give them options in the form of a drop-down menu instead of typing. That altogether would solve input error. Thirdly a switch that takes you back and forth from the collaborative filtering and preference filtering would be very handy and a global history list would allow the user to see the choices they make regardless of the recommendation system they use.

Implementation Plan

A typical implementation plan would be to take the two recommendation systems and create a user interface where the user would be able to flawlessly pick out the recommendation system they would like to use and enter preferences easily. User input will have the option to autocorrect and provide suggestions via a dropdown menu. Users will also have a calibrated profile that can be adjusted at any time to get referred to other restaurants out of their comfort zone. The data wouldn’t be stored locally on the user’s computer so there might have to be some kind of API call to gather enough information about the restaurants a user might prefer.

Ethical Assessment

The only ethical implication I would possibly see would be the lack of information about certain allergens not included in the data files. A user might be recommended to eat at a restaurant where they could be allergic to the environment, the dishes, or certain culinary methods like using peanut oil or not cleaning equipment after each dish. The only way to possibly remedy this would be to ask the restaurants a little more about their dishes and common allergens. Then the next task would be to add that as a feature to the existing dataset so that the user can add preferences for avoiding allergens.

Audience Questions and Answers:

1) What is the best method to filter the restaurants by preference, and possibly time and location?
    a) Filtering by preference was a challenge and involved a counter going up everytime a condition was met in the row, and if the counter was equal for the length of the preference dictionary for that row, the row was saved. Filtering by time required iterating through each list of times as integers and making sure that the entered time was in that interval. In addition to the time, the user is also asked to specify what day they would like the restaurant to be open and whether they would like to filter the preferred list or recommended list of restaurants. The location filter was not implemented as the dataset was centralized around Mexico. There was not a large variety of locations.
2) How would you ask foreign restaurants to provide more information if you were to implement this system in real life?
    a) I would show them the ideal version of entered values for each of the attributes from an existing data frame, but if that is too hard to follow, I will ask them to submit their information in document-based (json) form, csv, or even text and I will hand format it unless it is a consistent format.
3) How do you know that "Nearest Neighbors" creates the best recommendation systems?
    a) With some research I found that nearest neighbors are the easiest to train and the most reliable in terms of recommendation systems (Liao, 2018).
4) Following the previous questions, what other possible models are there for recommendation systems that can be tried?

   a) There are projects using SVM but I'm not sure if SVM can be as versatile as KNN for matrix vectorization.

5) Might there be a way to condense the helper functions in data formatting into one clean one?
   a) I tried my hand at formatting the helper functions into one, but they ended up either taking up too much space and looking cluttered. I still prefer having smaller helper functions that I can test individually before gathering them into one larger function that can be run for general purposes.
6) What types of visualizations can help users get more comfortable with the recommendation system?
   a) This is honestly a great question, and one where I still haven't thought of a sure-fire answer for. However, I think maybe having loading metrics as their data is loaded (there's no load time running locally) could exist to ease tension and helpful pictures explaining why the specific set of preferences may not return an exact set of restaurants.
7) Would you consider adding an image feature where pictures of the restaurants, the location, and the menu are provided along with the other information?
   a) Adding pictures of the restaurants is a great idea. I could use google maps as a way for the user to also explore the area around the restaurant to scout for potential parking or issues they may have with the area. The location should be easily accessible by GPS and provided. Recently updated menus should also be provided.
8) Do you believe that if the data was formatted, that this project would've taken a lot less time?
   a) 100% yes. About half of my project was spent on hard editing the existing data files, especially formatting the time. But I'm glad I got that over with earlier because the filter

by time method for the content-based filtering recommendation system worked exactly as intended.

9) If the data was already formatted, what would you have accomplished instead?
   a) I want to say that I would at least have some kind of prototype user interface or at least a method to make there be less print in the notebook file if the user chooses to run the recommendation through the file.

10) Any ideas on how to accomplish the user interface?
    a) I didn't have much time to learn about user interfaces for the duration of this project with the workload of classes and my job. However, I have some ideas written down here in the report and on the notebook file that can be explored at my leisure. I look forward to polishing up this project when I can.

References

Liao, K. (2018a, November 19). *Prototyping a recommender system step by Step Part 1: Knn Item-based collaborative filtering*. Medium. https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-1-knn-item-based-collaborative-filtering-637969614ea

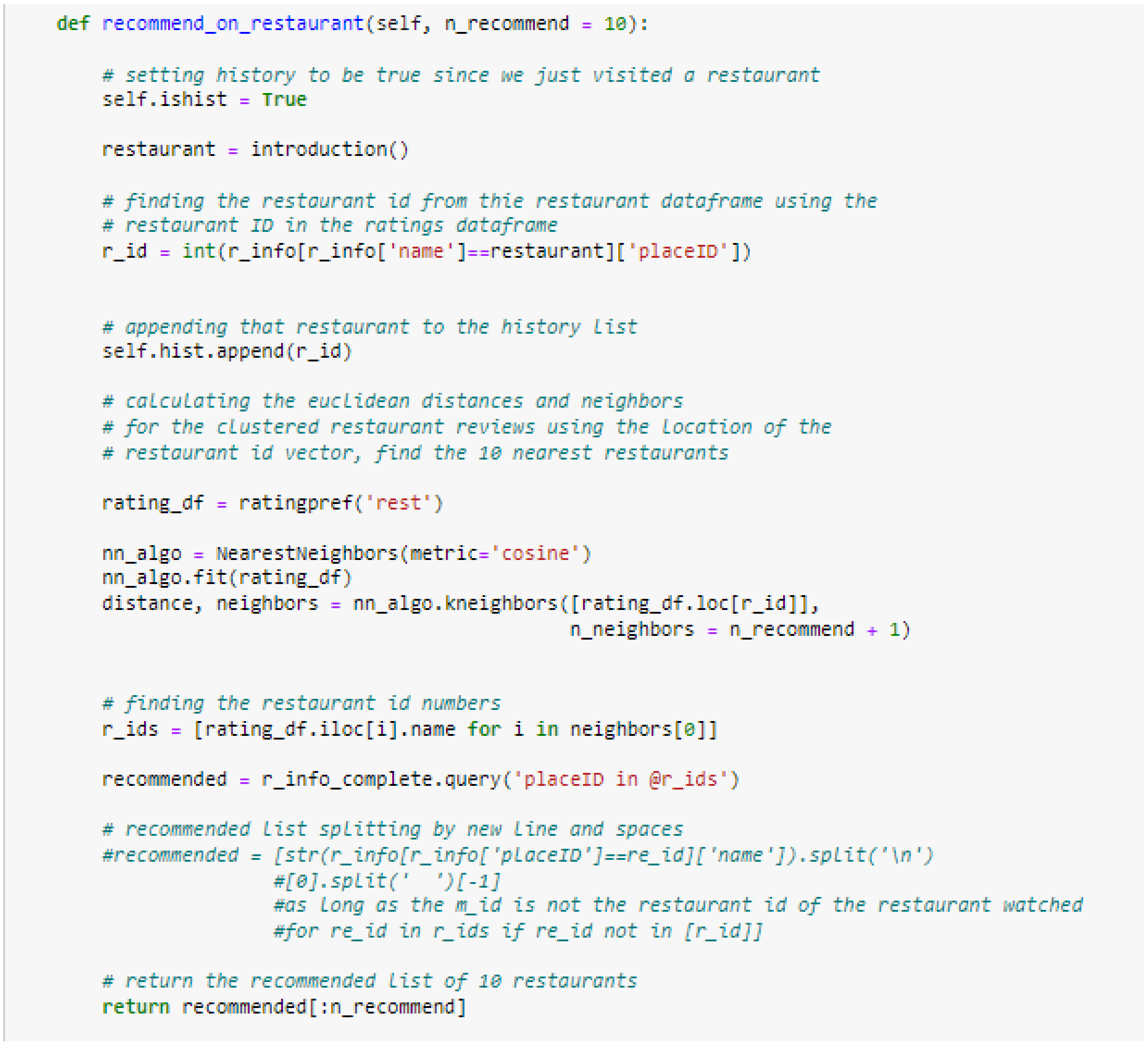
M, D. (2022, July 7). What is cosine similarity and how is it used in machine learning?. Analytics India Magazine. https://analyticsindiamag.com/cosine-similarity-in-machine learning/#:~:text=Cosine%20similarity%20is%20used%20as,of%20texts%20in%20the 20document.

Restaurant & Consumer Data. UCI Machine Learning Repository. (n.d.). https://archive.ics.uci.edu/dataset/232/restaurant+consumer+data

Sachinsarkar. (2021, November 6). Movielens Movie Recommendation System. Kaggle. https://www.kaggle.com/code/sachinsarkar/movielens-movie-recommendation system/notebook

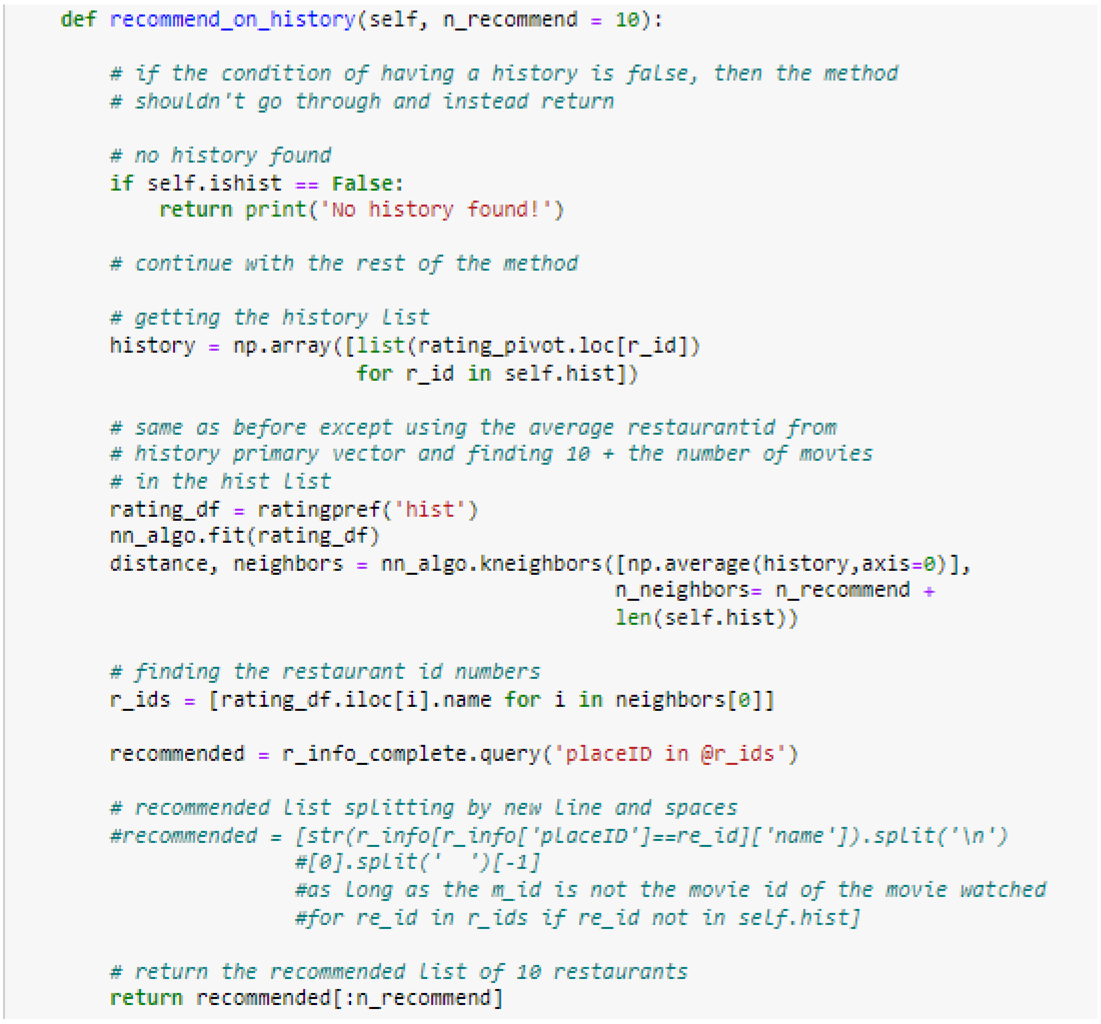
WilliamVorhies. (2017, January 17). 5 types of recommenders. Data Science Central. https://www.datasciencecentral.com/5-types-of-recommenders/

Appendix A: *Recommendation System Structure*



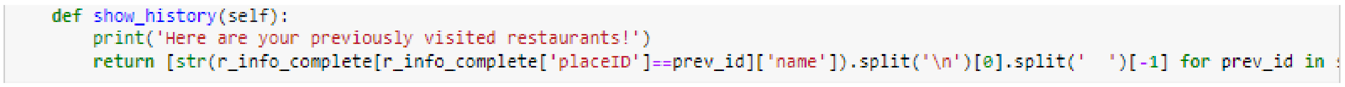
```python
def recommend_on_restaurant(self, n_recommend = 10):

    # setting history to be true since we just visited a restaurant
    self.ishist = True

    restaurant = introduction()

    # finding the restaurant id from thie restaurant dataframe using the
    # restaurant ID in the ratings dataframe
    r_id = int(r_info[r_info['name']==restaurant]['placeID'])


    # appending that restaurant to the history list
    self.hist.append(r_id)

    # calculating the euclidean distances and neighbors
    # for the clustered restaurant reviews using the location of the
    # restaurant id vector, find the 10 nearest restaurants

    rating_df = ratingpref('rest')

    nn_algo = NearestNeighbors(metric='cosine')
    nn_algo.fit(rating_df)
    distance, neighbors = nn_algo.kneighbors([rating_df.loc[r_id]],
                                             n_neighbors = n_recommend + 1)


    # finding the restaurant id numbers
    r_ids = [rating_df.iloc[i].name for i in neighbors[0]]

    recommended = r_info_complete.query('placeID in @r_ids')

    # recommended list splitting by new line and spaces
    #recommended = [str(r_info[r_info['placeID']==re_id]['name']).split('\n')
                   #[0].split('  ')[-1]
                   #as long as the m_id is not the restaurant id of the restaurant watched
                   #for re_id in r_ids if re_id not in [r_id]]

    # return the recommended list of 10 restaurants
    return recommended[:n_recommend]
```

Figure A1: Part 1 of Recommendation System Structure



```
    def recommend_on_history(self, n_recommend = 10):

        # if the condition of having a history is false, then the method
        # shouldn't go through and instead return

        # no history found
        if self.ishist == False:
            return print('No history found!')

        # continue with the rest of the method

        # getting the history list
        history = np.array([list(rating_pivot.loc[r_id])
                            for r_id in self.hist])

        # same as before except using the average restaurantid from
        # history primary vector and finding 10 + the number of movies
        # in the hist list
        rating_df = ratingpref('hist')
        nn_algo.fit(rating_df)
        distance, neighbors = nn_algo.kneighbors([np.average(history,axis=0)],
                                                 n_neighbors= n_recommend +
                                                 len(self.hist))

        # finding the restaurant id numbers
        r_ids = [rating_df.iloc[i].name for i in neighbors[0]]

        recommended = r_info_complete.query('placeID in @r_ids')

        # recommended list splitting by new line and spaces
        #recommended = [str(r_info[r_info['placeID']==re_id]['name']).split('\n')
                       #[0].split('  ')[-1]
                       #as long as the m_id is not the movie id of the movie watched
                       #for re_id in r_ids if re_id not in self.hist]

        # return the recommended list of 10 restaurants
        return recommended[:n_recommend]
```

Figure A2: Part 2 of Recommendation System Structure



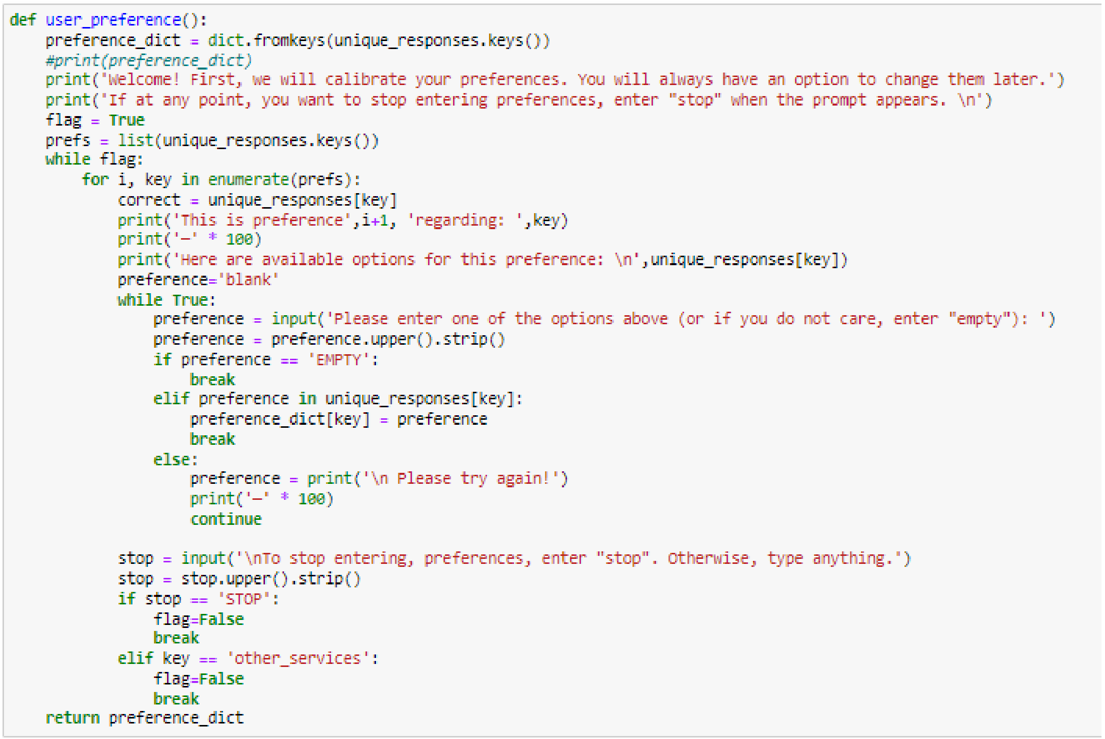
```
    def show_history(self):
        print('Here are your previously visited restaurants!')
        return [str(r_info_complete[r_info_complete['placeID']==prev_id]['name']).split('\n')[0].split('  ')[-1] for prev_id in
```

Figure A3: Part 3 of Recommendation System Structure

Appendix B: *User Preference (Content-Based Filtering)*

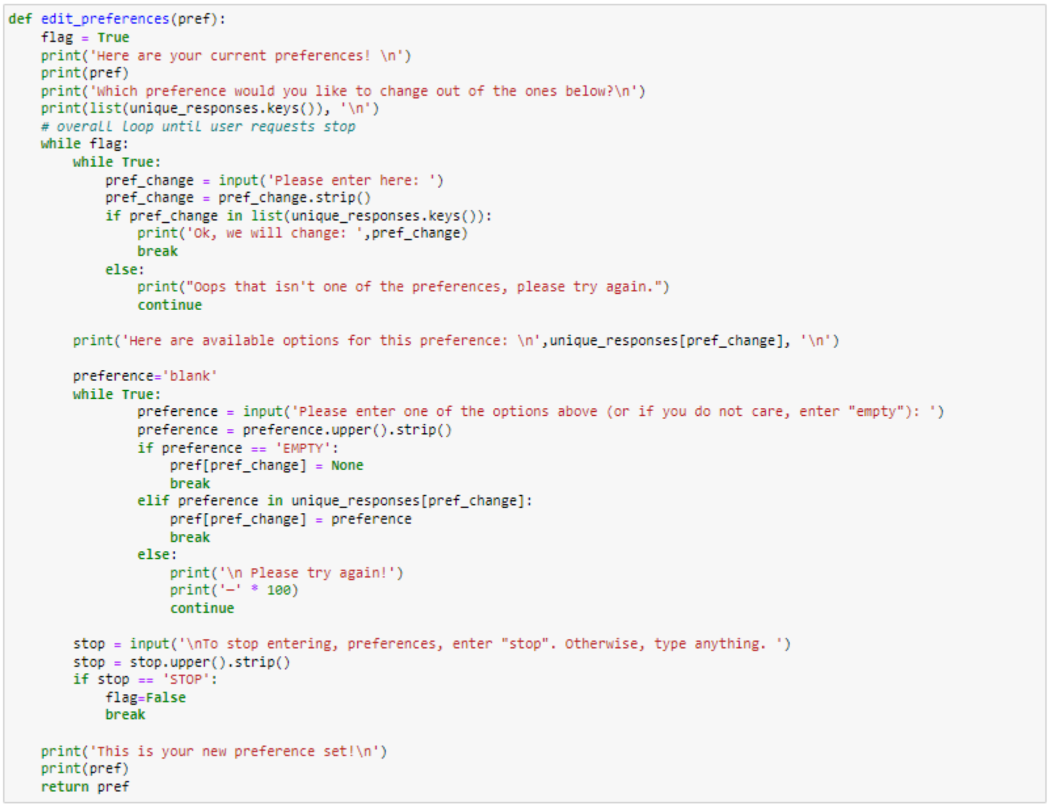
```
{'cuisine': ['BURGERS','ARMENIAN','GAME','JAPANESE','MEXICAN','FAMILY',
        'BAKERY','CONTEMPORARY','AMERICAN','CAFETERIA','REGIONAL',
        'INTERNATIONAL',
        'FAST_FOOD','CAFECOFFEE_SHOP','CHINESE','ITALIAN','MEDITERRANEAN','
        BAR_PUB_BREWERY','BAR','BREAKFASTBRUNCH','SEAFOOD','PIZZERIA',
        'VIETNAMESE'],
 'payment': ['DISCOVER','VISA','BANK_DEBIT_CARDS','MASTERCARD_EUROCARD',
        'AMERICAN_EXPRESS','CARTE_BLANCHE','CHECKS','CASH'],
 'parking': ['YES', 'PUBLIC', 'VALET_PARKING', 'NONE'],
 'alcohol': ['NO_ALCOHOL_SERVED', 'WINE_BEER', 'FULL_BAR'],
 'smoking_area': ['NONE','ONLY_AT_BAR','PERMITTED','SECTION',
'NOT_PERMITTED'],
 'dress_code': ['INFORMAL', 'CASUAL', 'FORMAL'],
 'accessibility': ['NO_ACCESSIBILITY', 'COMPLETELY', 'PARTIALLY'],
 'price': ['MEDIUM', 'LOW', 'HIGH'],
 'Rambience': ['FAMILIAR', 'QUIET'],
 'area': ['CLOSED', 'OPEN'],
 'other_services': ['NONE', 'INTERNET', 'VARIETY'],
 'wdschedule': ['OPEN', 'CLOSED'],
 'satschedule': ['OPEN', 'CLOSED'],
 'sunschedule': ['OPEN', 'CLOSED']}
```

Figure B1: Unique Responses for User Preference Input



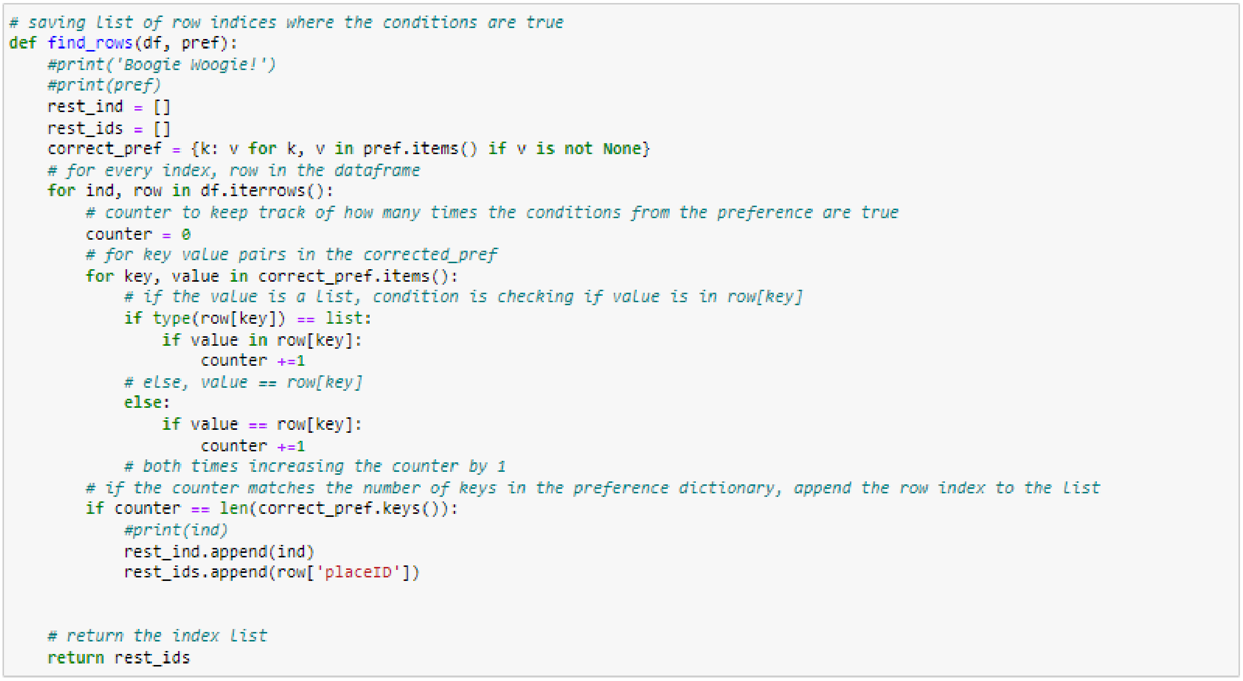
```python
def user_preference():
    preference_dict = dict.fromkeys(unique_responses.keys())
    #print(preference_dict)
    print('Welcome! First, we will calibrate your preferences. You will always have an option to change them later.')
    print('If at any point, you want to stop entering preferences, enter "stop" when the prompt appears. \n')
    flag = True
    prefs = list(unique_responses.keys())
    while flag:
        for i, key in enumerate(prefs):
            correct = unique_responses[key]
            print('This is preference',i+1, 'regarding: ',key)
            print('-' * 100)
            print('Here are available options for this preference: \n',unique_responses[key])
            preference='blank'
            while True:
                preference = input('Please enter one of the options above (or if you do not care, enter "empty"): ')
                preference = preference.upper().strip()
                if preference == 'EMPTY':
                    break
                elif preference in unique_responses[key]:
                    preference_dict[key] = preference
                    break
                else:
                    preference = print('\n Please try again!')
                    print('-' * 100)
                    continue

            stop = input('\nTo stop entering, preferences, enter "stop". Otherwise, type anything.')
            stop = stop.upper().strip()
            if stop == 'STOP':
                flag=False
                break
            elif key == 'other_services':
                flag=False
                break
    return preference_dict
```

Figure B2: Feedback Loop for User Preference



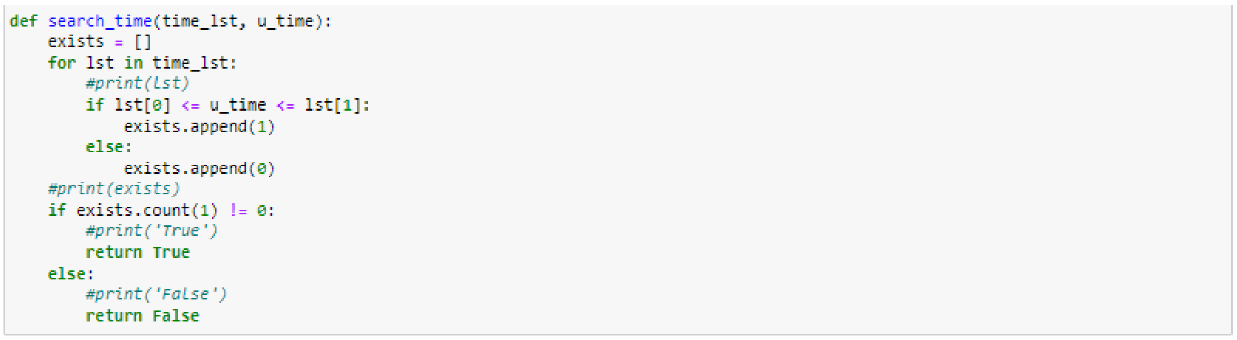
```python
def edit_preferences(pref):
    flag = True
    print('Here are your current preferences! \n')
    print(pref)
    print('Which preference would you like to change out of the ones below?\n')
    print(list(unique_responses.keys()), '\n')
    # overall loop until user requests stop
    while flag:
        while True:
            pref_change = input('Please enter here: ')
            pref_change = pref_change.strip()
            if pref_change in list(unique_responses.keys()):
                print('Ok, we will change: ',pref_change)
                break
            else:
                print("Oops that isn't one of the preferences, please try again.")
                continue

        print('Here are available options for this preference: \n',unique_responses[pref_change], '\n')

        preference='blank'
        while True:
                preference = input('Please enter one of the options above (or if you do not care, enter "empty"): ')
                preference = preference.upper().strip()
                if preference == 'EMPTY':
                    pref[pref_change] = None
                    break
                elif preference in unique_responses[pref_change]:
                    pref[pref_change] = preference
                    break
                else:
                    print('\n Please try again!')
                    print('-' * 100)
                    continue

        stop = input('\nTo stop entering, preferences, enter "stop". Otherwise, type anything. ')
        stop = stop.upper().strip()
        if stop == 'STOP':
            flag=False
            break

    print('This is your new preference set!\n')
    print(pref)
    return pref
```

Figure B3: Feedback Loop for Editing Preferences



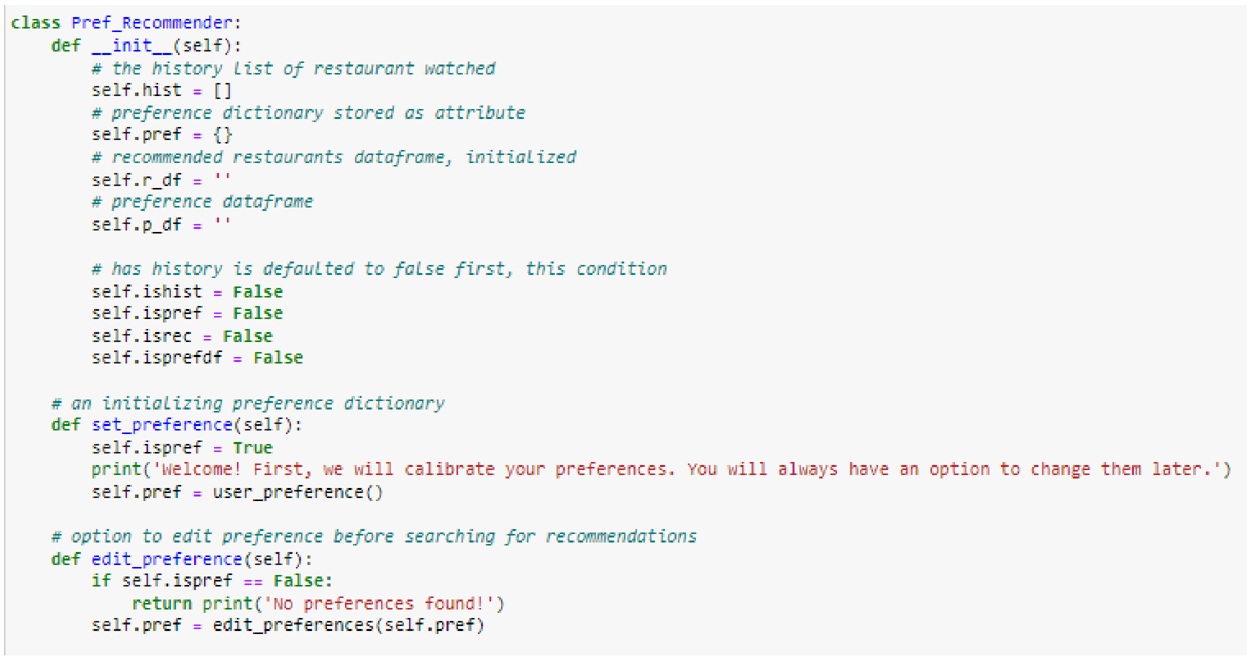
```python
# saving list of row indices where the conditions are true
def find_rows(df, pref):
    #print('Boogie Woogie!')
    #print(pref)
    rest_ind = []
    rest_ids = []
    correct_pref = {k: v for k, v in pref.items() if v is not None}
    # for every index, row in the dataframe
    for ind, row in df.iterrows():
        # counter to keep track of how many times the conditions from the preference are true
        counter = 0
        # for key value pairs in the corrected_pref
        for key, value in correct_pref.items():
            # if the value is a list, condition is checking if value is in row[key]
            if type(row[key]) == list:
                if value in row[key]:
                    counter +=1
            # else, value == row[key]
            else:
                if value == row[key]:
                    counter +=1
            # both times increasing the counter by 1
        # if the counter matches the number of keys in the preference dictionary, append the row index to the list
        if counter == len(correct_pref.keys()):
            #print(ind)
            rest_ind.append(ind)
            rest_ids.append(row['placeID'])


    # return the index list
    return rest_ids
```

Figure B4: Find Restaurant Ids Based on Preferences



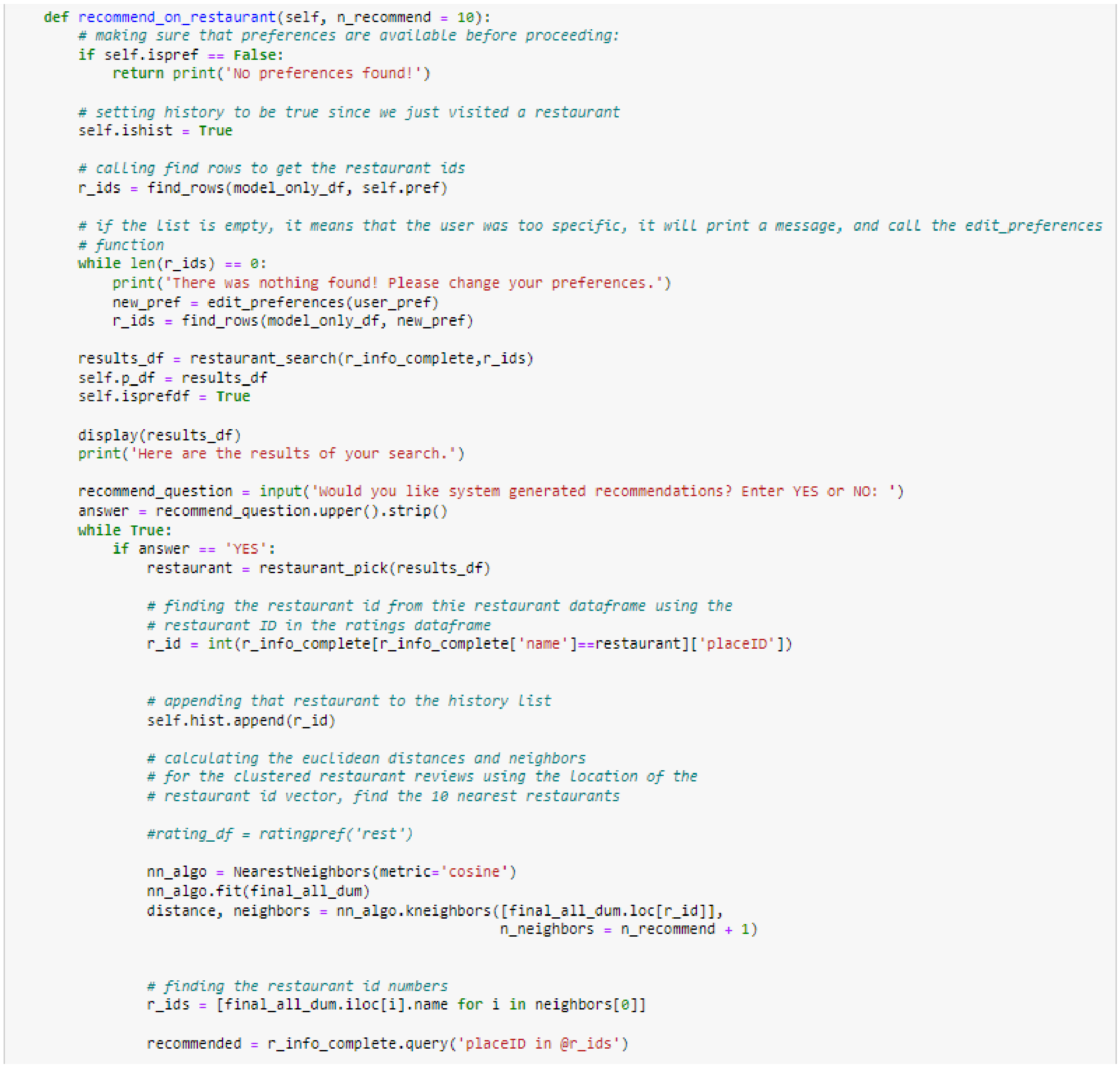
```python
def search_time(time_lst, u_time):
    exists = []
    for lst in time_lst:
        #print(lst)
        if lst[0] <= u_time <= lst[1]:
            exists.append(1)
        else:
            exists.append(0)
    #print(exists)
    if exists.count(1) != 0:
        #print('True')
        return True
    else:
        #print('False')
        return False
```

Figure B6: Filter by time value

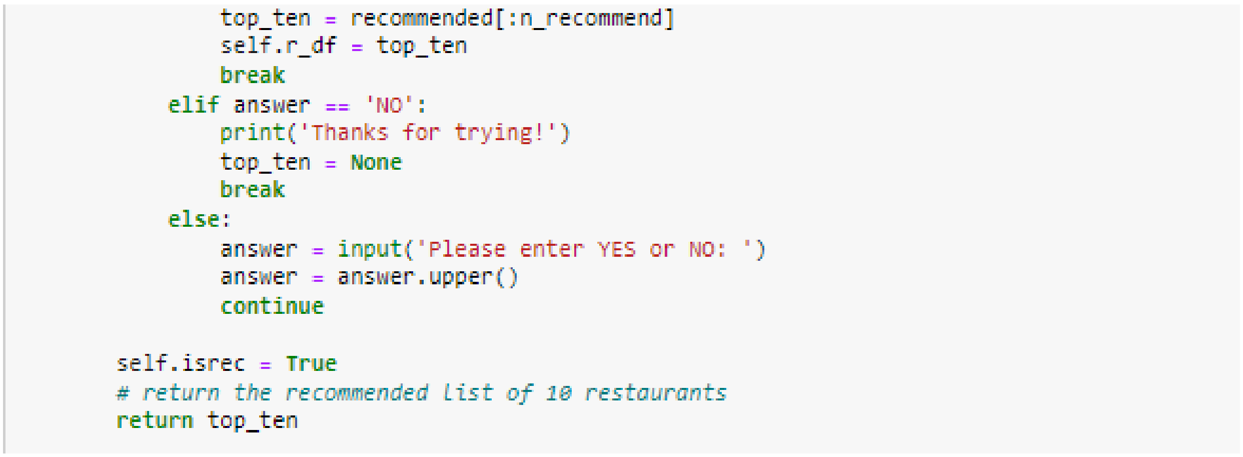


```python
class Pref_Recommender:
    def __init__(self):
        # the history list of restaurant watched
        self.hist = []
        # preference dictionary stored as attribute
        self.pref = {}
        # recommended restaurants dataframe, initialized
        self.r_df = ''
        # preference dataframe
        self.p_df = ''

        # has history is defaulted to false first, this condition
        self.ishist = False
        self.ispref = False
        self.isrec = False
        self.isprefdf = False

    # an initializing preference dictionary
    def set_preference(self):
        self.ispref = True
        print('Welcome! First, we will calibrate your preferences. You will always have an option to change them later.')
        self.pref = user_preference()

    # option to edit preference before searching for recommendations
    def edit_preference(self):
        if self.ispref == False:
            return print('No preferences found!')
        self.pref = edit_preferences(self.pref)
```

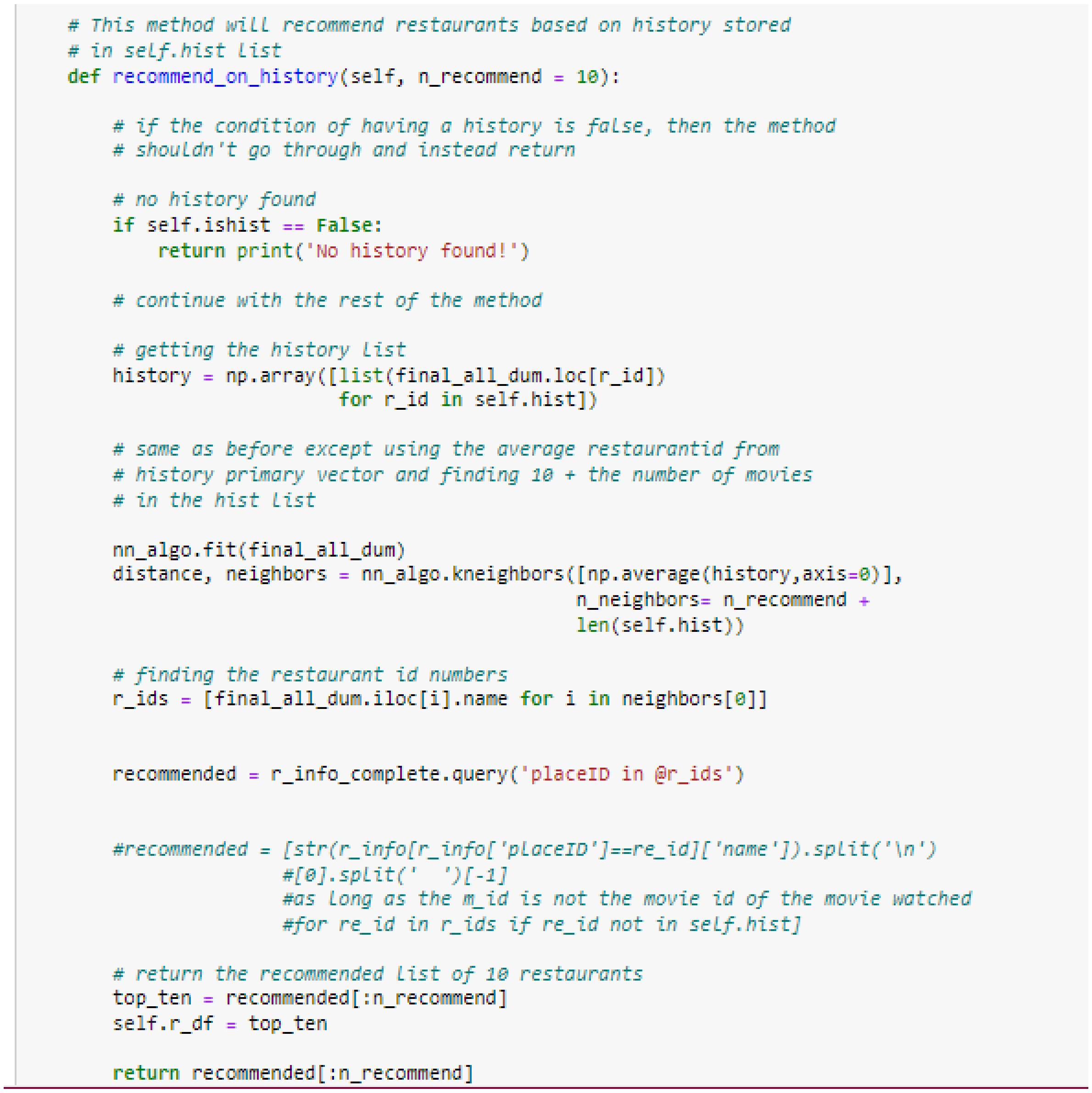
Figure B7-B10: Pref_Recommender Class with initial methods



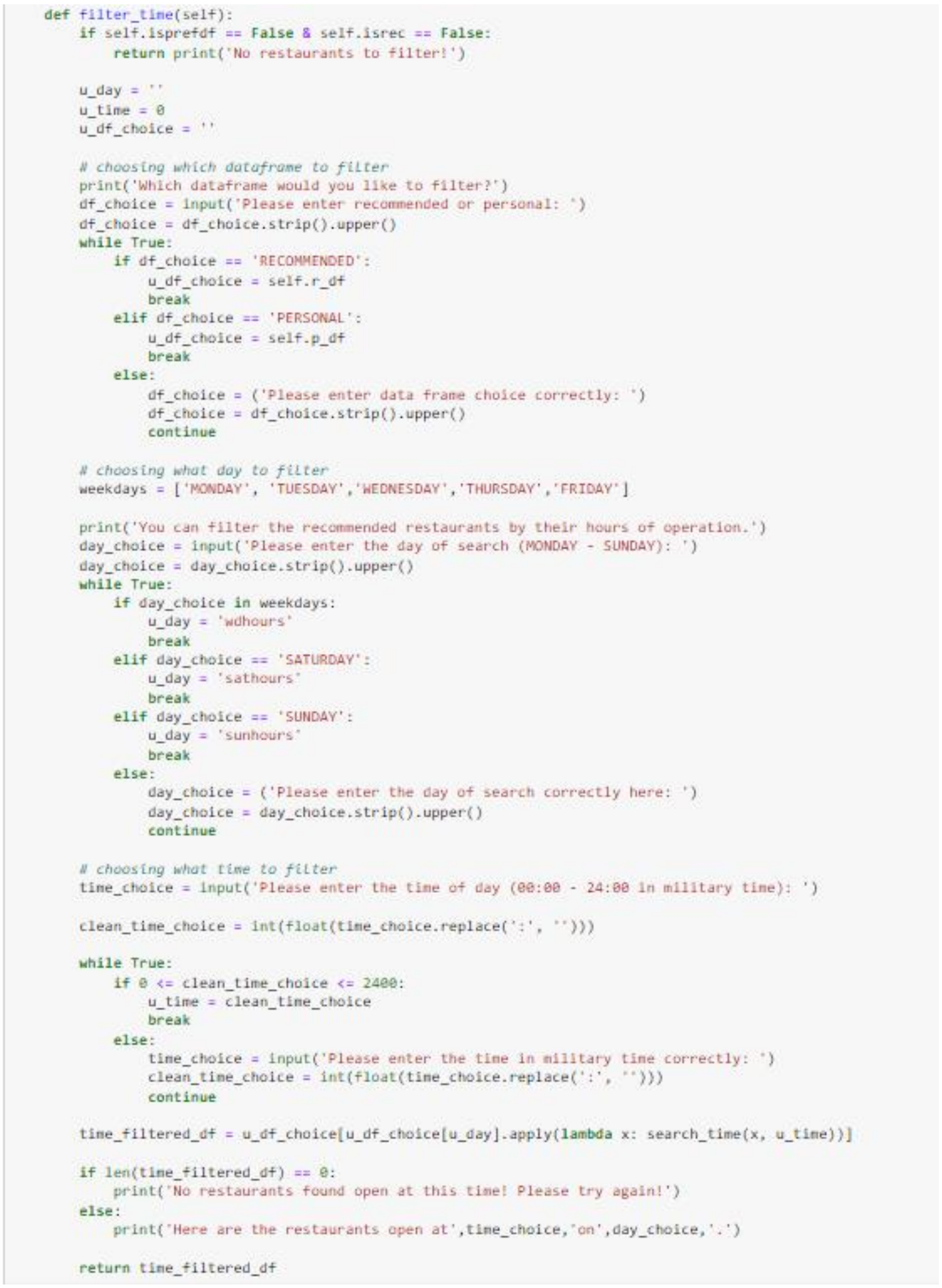
```
    def recommend_on_restaurant(self, n_recommend = 10):
        # making sure that preferences are available before proceeding:
        if self.ispref == False:
            return print('No preferences found!')

        # setting history to be true since we just visited a restaurant
        self.ishist = True

        # calling find rows to get the restaurant ids
        r_ids = find_rows(model_only_df, self.pref)

        # if the list is empty, it means that the user was too specific, it will print a message, and call the edit_preferences
        # function
        while len(r_ids) == 0:
            print('There was nothing found! Please change your preferences.')
            new_pref = edit_preferences(user_pref)
            r_ids = find_rows(model_only_df, new_pref)

        results_df = restaurant_search(r_info_complete,r_ids)
        self.p_df = results_df
        self.isprefdf = True

        display(results_df)
        print('Here are the results of your search.')

        recommend_question = input('Would you like system generated recommendations? Enter YES or NO: ')
        answer = recommend_question.upper().strip()
        while True:
            if answer == 'YES':
                restaurant = restaurant_pick(results_df)

                # finding the restaurant id from thie restaurant dataframe using the
                # restaurant ID in the ratings dataframe
                r_id = int(r_info_complete[r_info_complete['name']==restaurant]['placeID'])


                # appending that restaurant to the history list
                self.hist.append(r_id)

                # calculating the euclidean distances and neighbors
                # for the clustered restaurant reviews using the location of the
                # restaurant id vector, find the 10 nearest restaurants

                #rating_df = ratingpref('rest')

                nn_algo = NearestNeighbors(metric='cosine')
                nn_algo.fit(final_all_dum)
                distance, neighbors = nn_algo.kneighbors([final_all_dum.loc[r_id]],
                                                         n_neighbors = n_recommend + 1)


                # finding the restaurant id numbers
                r_ids = [final_all_dum.iloc[i].name for i in neighbors[0]]

                recommended = r_info_complete.query('placeID in @r_ids')
```



```
                top_ten = recommended[:n_recommend]
                self.r_df = top_ten
                break
            elif answer == 'NO':
                print('Thanks for trying!')
                top_ten = None
                break
            else:
                answer = input('Please enter YES or NO: ')
                answer = answer.upper()
                continue

        self.isrec = True
        # return the recommended list of 10 restaurants
        return top_ten
```



```
    # This method will recommend restaurants based on history stored
    # in self.hist list
    def recommend_on_history(self, n_recommend = 10):

        # if the condition of having a history is false, then the method
        # shouldn't go through and instead return

        # no history found
        if self.ishist == False:
            return print('No history found!')

        # continue with the rest of the method

        # getting the history list
        history = np.array([list(final_all_dum.loc[r_id])
                            for r_id in self.hist])

        # same as before except using the average restaurantid from
        # history primary vector and finding 10 + the number of movies
        # in the hist list

        nn_algo.fit(final_all_dum)
        distance, neighbors = nn_algo.kneighbors([np.average(history,axis=0)],
                                                 n_neighbors= n_recommend +
                                                 len(self.hist))

        # finding the restaurant id numbers
        r_ids = [final_all_dum.iloc[i].name for i in neighbors[0]]


        recommended = r_info_complete.query('placeID in @r_ids')


        #recommended = [str(r_info[r_info['placeID']==re_id]['name']).split('\n')
                       #[0].split('  ')[-1]
                       #as long as the m_id is not the movie id of the movie watched
                       #for re_id in r_ids if re_id not in self.hist]

        # return the recommended list of 10 restaurants
        top_ten = recommended[:n_recommend]
        self.r_df = top_ten

        return recommended[:n_recommend]
```



```
def filter_time(self):
    if self.isprefdf == False & self.isrec == False:
        return print('No restaurants to filter!')

    u_day = ''
    u_time = 0
    u_df_choice = ''

    # choosing which dataframe to filter
    print('Which dataframe would you like to filter?')
    df_choice = input('Please enter recommended or personal: ')
    df_choice = df_choice.strip().upper()
    while True:
        if df_choice == 'RECOMMENDED':
            u_df_choice = self.r_df
            break
        elif df_choice == 'PERSONAL':
            u_df_choice = self.p_df
            break
        else:
            df_choice = ('Please enter data frame choice correctly: ')
            df_choice = df_choice.strip().upper()
            continue

    # choosing what day to filter
    weekdays = ['MONDAY', 'TUESDAY','WEDNESDAY','THURSDAY','FRIDAY']

    print('You can filter the recommended restaurants by their hours of operation.')
    day_choice = input('Please enter the day of search (MONDAY - SUNDAY): ')
    day_choice = day_choice.strip().upper()
    while True:
        if day_choice in weekdays:
            u_day = 'wdhours'
            break
        elif day_choice == 'SATURDAY':
            u_day = 'sathours'
            break
        elif day_choice == 'SUNDAY':
            u_day = 'sunhours'
            break
        else:
            day_choice = ('Please enter the day of search correctly here: ')
            day_choice = day_choice.strip().upper()
            continue

    # choosing what time to filter
    time_choice = input('Please enter the time of day (00:00 - 24:00 in military time): ')

    clean_time_choice = int(float(time_choice.replace(':', '')))

    while True:
        if 0 <= clean_time_choice <= 2400:
            u_time = clean_time_choice
            break
        else:
            time_choice = input('Please enter the time in military time correctly: ')
            clean_time_choice = int(float(time_choice.replace(':', '')))
            continue

    time_filtered_df = u_df_choice[u_df_choice[u_day].apply(lambda x: search_time(x, u_time))]

    if len(time_filtered_df) == 0:
        print('No restaurants found open at this time! Please try again!')
    else:
        print('Here are the restaurants open at',time_choice,'on',day_choice,'.')

    return time_filtered_df
```

Figure B11: Hour and Day of Operation Filter