P134 Final Project: Recommending Restaurants

The goal of this project was to build a system to recommend a restaurant given different preferences from a user. For this project we narrowed our data to restaurants found in Isla Vista and the general Santa Barbara area, with the data being accessed using Yelp's Fusion API. After processing the data, we created two recommender systems, one based on using bag of words and another using tf-idf values from tokenzing multiple variables in the data set. The intent for the recommender systems is to:

1. Recommend restaurants given a particular cuisine from the user
2. Recommend restaurants similar to a restaurant given from the user

```
from flask import Flask, render_template, request  # Import Flask and related modules for web handling
import requests  # Import requests library to make HTTP requests
import http.client
from google.colab import files
import json
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import MinMaxScaler
```

Using Yelp's Fusion API

To access Yelp's Fusion API, one of our group members did have to create a membership to access the one-month free trial. The API request restricted the number of restaurants per request to 50 which motivated us to narrow the scope to the Isla Vista and general Santa Barbara area. To process the data from the API request, we created a helper function `get_restaurants` to extract the desired information and convert the data into a csv file we can use for our recommender system.

```
# Yelp API credentials
client_id = 'fOg_XdQrnZ96u3gwOQFPdw'
api_key = 'f-KxAdzDk5nbXKT9P9hBfefO7kpRW7EHFSxnxSNMF3AZJ5thYkeQBQRQ_sIDnEGCoafODEHteG2imdncOWZB72MAUWlTR5YLI6u5EOcaqtD15YpKS_GGD

# Yelp API endpoint for searching businesses
url = 'https://api.yelp.com/v3/businesses/search'
```

```
# function to return a dataframe of 50 restaurants in area
def get_restaurants(city, term='restaurants', limit = 50):
    # Construct the Yelp API URL
    url = 'https://api.yelp.com/v3/businesses/search'

    # Set up the headers for authentication
    headers = {
        'Authorization': f'Bearer {api_key}'
    }

    # Define the parameters for the request
    params = {
        'location': city,
        'term': term,
        'limit': limit
    }

    # Send a GET request to the Yelp API
    response = requests.get(url, headers=headers, params=params)

    # Check if the response was successful
    if response.status_code == 200:
        # Return the JSON data AS A DATAFRAME if the request was successful
        response_data = response.json()

        restaurants_df = pd.DataFrame()
        for i in range(len(response_data['businesses'])):
          # input restaurant information
          restaurants_df.loc[i, 'Business_ID'] = response_data['businesses'][i].get('id', None)
          restaurants_df.loc[i, 'Name'] = response_data['businesses'][i].get('name', None)
```

```
          restaurants_df.loc[i, 'NumReviews'] = response_data['businesses'][i].get('review_count', None)
          restaurants_df.loc[i, 'Rating'] = response_data['businesses'][i].get('rating', None)

          # input information from categories tab
          categories = response_data['businesses'][i].get('categories', [])
          if len(categories) > 0:
            restaurants_df.loc[i, 'Cuisine'] = categories[0]['title']
          if len(categories) > 1:
            restaurants_df.loc[i, 'Cuisine1'] = categories[1]['title']
          if len(categories) > 2:
            restaurants_df.loc[i, 'Cuisine2'] = categories[2]['title']

          restaurants_df.loc[i, 'PriceRange'] = response_data['businesses'][i].get('price', None)
          restaurants_df.loc[i, 'Takeout'] = ', '.join(response_data['businesses'][i].get('transactions', [])) # rewrite list as
          restaurants_df.loc[i, 'Waitlist_Needed'] = response_data['businesses'][i].get('attributes', {}).get('waitlist_reservat

          # business hours (check if any open hours)
          business_hours = response_data['businesses'][i].get('business_hours', [])
          if business_hours:
              open_hours = business_hours[0].get('open', [])
          else:
              open_hours = []
          days = ['Su', 'M', 'T', 'W', 'R', 'F', 'Sa']

          for j in range(7):
              if j < len(open_hours):
                  restaurants_df.loc[i, f'{days[j]}_start'] = open_hours[j].get('start', None)
                  restaurants_df.loc[i, f'{days[j]}_end'] = open_hours[j].get('end', None)
              else:
                  restaurants_df.loc[i, f'{days[j]}_start'] = None
                  restaurants_df.loc[i, f'{days[j]}_end'] = None

          restaurants_df.loc[i, 'Latitude'] = response_data['businesses'][i].get('coordinates', {}).get('latitude', None)
          restaurants_df.loc[i, 'Longitude'] = response_data['businesses'][i].get('coordinates', {}).get('longitude', None)
          restaurants_df.loc[i, 'Address'] = response_data['businesses'][i].get('location', {}).get('address1', None)
          restaurants_df.loc[i, 'City'] = response_data['businesses'][i].get('location', {}).get('city', None)
          restaurants_df.loc[i, 'State'] = response_data['businesses'][i].get('location', {}).get('state', None)
          restaurants_df.loc[i, 'Zip'] = response_data['businesses'][i].get('location', {}).get('zip_code', None)
          restaurants_df.loc[i, 'Menu'] = response_data['businesses'][i].get('attributes', {}).get('menu_url', None)

      return restaurants_df
  else:
      # Return None if there was an error
      return None
```

Not shown: Since we're working in a shared google drive using collab, there are a couple more lines not in here where we downloaded the csv files to the shared drive.

Data Cleaning

After accessing the restaurant information, we combined our csv files, one for Isla Vista and another for Santa Barbara, and changed some of variables to be processed in the recommender systems. Some of variable changes include changing a restaurants takeout options into a categorical variable, encoding the price ranges as a categorical variable, and creating a variable that determines when restaurants are open. With the updated variables, our original csv files are in dataframes ready to use for our recommender systems.

```
restaurants = pd.concat([iv, sb], ignore_index=True)
```

```
# string split takeouts into list objects
restaurants['Takeout'] = restaurants['Takeout'].apply(lambda x: x.split(', ') if isinstance(x, str) else [])

# one-hot encode takeout cateogires
takeout_encoded = restaurants['Takeout'].apply(lambda x: pd.Series(1, index=x)).fillna(0)

restaurants = pd.concat([restaurants, takeout_encoded], axis=1)
```

```
restaurants['PriceRange'].unique()

def clean_price(x):
    if x == '$':
        return 'cheap' # budget friendly, affordable
    elif x == '$$':
```

```
        return 'moderate' # mid-range
    elif x == '$$$':
        return 'high' # high-end, upscale, fine-dining
    elif x == '$$$$':
        return 'expensive' # luxury, premium experience

restaurants['price_descrip'] = restaurants['PriceRange'].apply(clean_price)
```

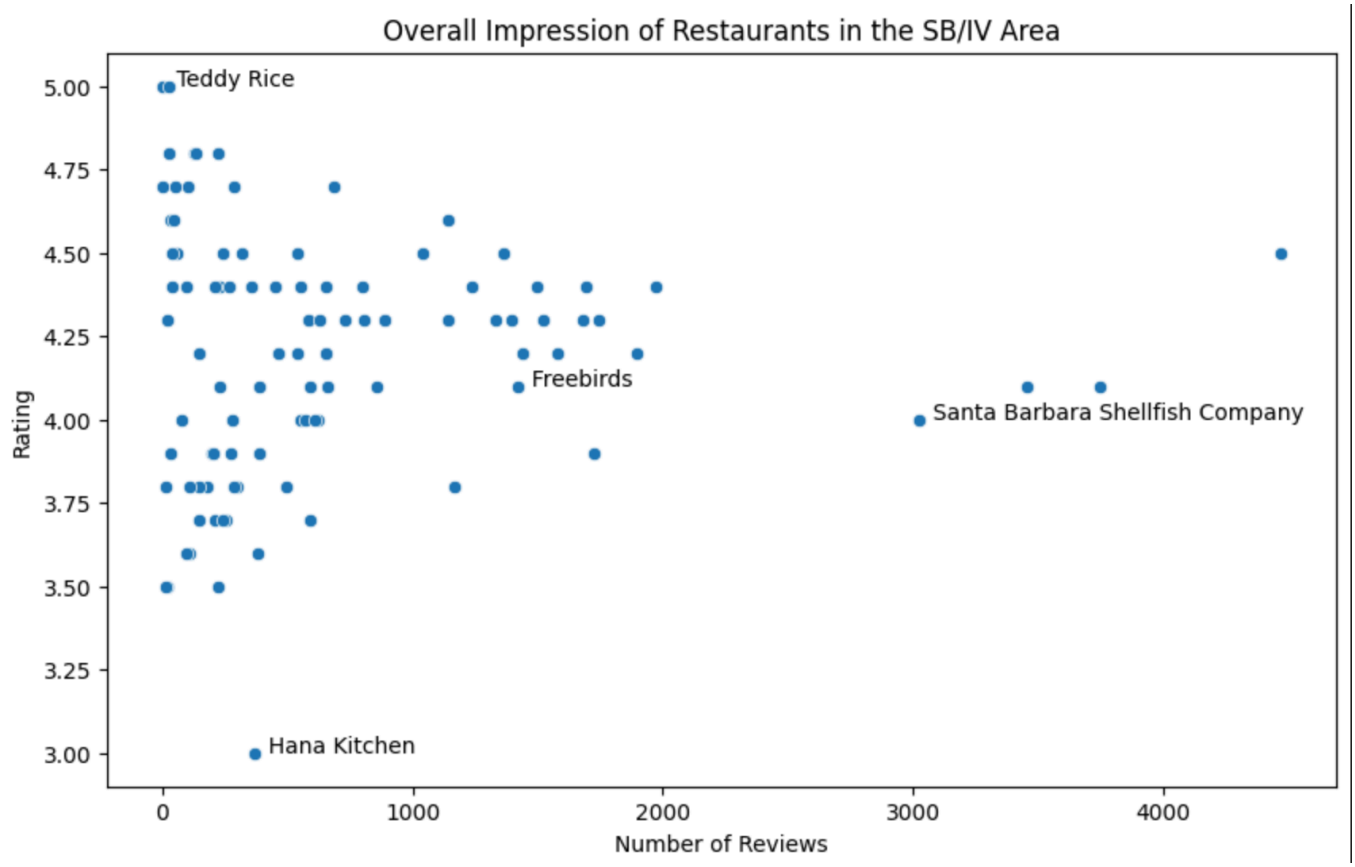
```
def is_open(start, end, meal_start, meal_end):
    # Normalize start/end times to the range of minutes from midnight
    start_minutes = (start // 100) * 60 + (start % 100)
    end_minutes = (end // 100) * 60 + (end % 100)
    meal_start_minutes = (meal_start // 100) * 60 + (meal_start % 100)
    meal_end_minutes = (meal_end // 100) * 60 + (meal_end % 100)

    # Check if the restaurant's hours overlap with the meal time
    return 1 if (start_minutes < meal_end_minutes and end_minutes > meal_start_minutes) else 0

# Meal time windows (in 24-hour format)
meal_times = {
    'Breakfast': (700, 1000),
    'Lunch': (1100, 1400),
    'Dinner': (1700, 2100)
}

# Apply the function for each day of the week
for day in ['Su', 'M', 'T', 'W', 'R', 'F', 'Sa']:
    for meal, (meal_start, meal_end) in meal_times.items():
        # Create new columns indicating whether the restaurant is open during each meal time
        restaurants[f'{day}_{meal}'] = restaurants.apply(lambda x: is_open(x[f'{day}_start'], x[f'{day}_end'], meal_start, meal_

restaurants['Cuisine'] = restaurants['Cuisine'].fillna('')
restaurants['Cuisine1'] = restaurants['Cuisine1'].fillna('')
restaurants['Cuisine2'] = restaurants['Cuisine2'].fillna('')
restaurants['Description'] = restaurants['Description'].fillna('')

restaurants['descriptors'] = restaurants.apply(lambda x: str(x['Cuisine']).strip() + ' ' + str(x['Cuisine1']).strip() + ' ' + st
```

```
# making final data frame
restaurants_cleaned = pd.DataFrame({'business_id': restaurants['Business_ID'],
                                    'name': restaurants['Name'],
                                    'num_reviews': restaurants['NumReviews'],
                                    'rating': restaurants['Rating'],
                                    'descriptors': restaurants['descriptors'],
                                    'price_descrip': restaurants['price_descrip'],
                                    'su_breakfast': restaurants['Su_Breakfast'],
                                    'su_lunch': restaurants['Su_Lunch'],
                                    'su_dinner': restaurants['Su_Dinner'],
                                    'm_breakfast': restaurants['M_Breakfast'],
                                    'm_lunch': restaurants['M_Lunch'],
                                    'm_dinner': restaurants['M_Dinner'],
                                    't_breakfast': restaurants['T_Breakfast'],
                                    't_lunch': restaurants['T_Lunch'],
                                    't_dinner': restaurants['T_Dinner'],
                                    'w_breakfast': restaurants['W_Breakfast'],
                                    'w_lunch': restaurants['W_Lunch'],
                                    'w_dinner': restaurants['W_Dinner'],
                                    'r_breakfast': restaurants['R_Breakfast'],
                                    'r_lunch': restaurants['R_Lunch'],
                                    'r_dinner': restaurants['R_Dinner'],
                                    'f_breakfast': restaurants['F_Breakfast'],
                                    'f_lunch': restaurants['F_Lunch'],
                                    'f_dinner': restaurants['F_Dinner'],
                                    'sa_breakfast': restaurants['Sa_Breakfast'],
                                    'sa_lunch': restaurants['Sa_Lunch'],
                                    'sa_dinner': restaurants['Sa_Dinner'],
                                    'pickup': restaurants['pickup'],
                                    'delivery': restaurants['delivery'],
                                    'restaurant_reservation': restaurants['restaurant_reservation']})
```

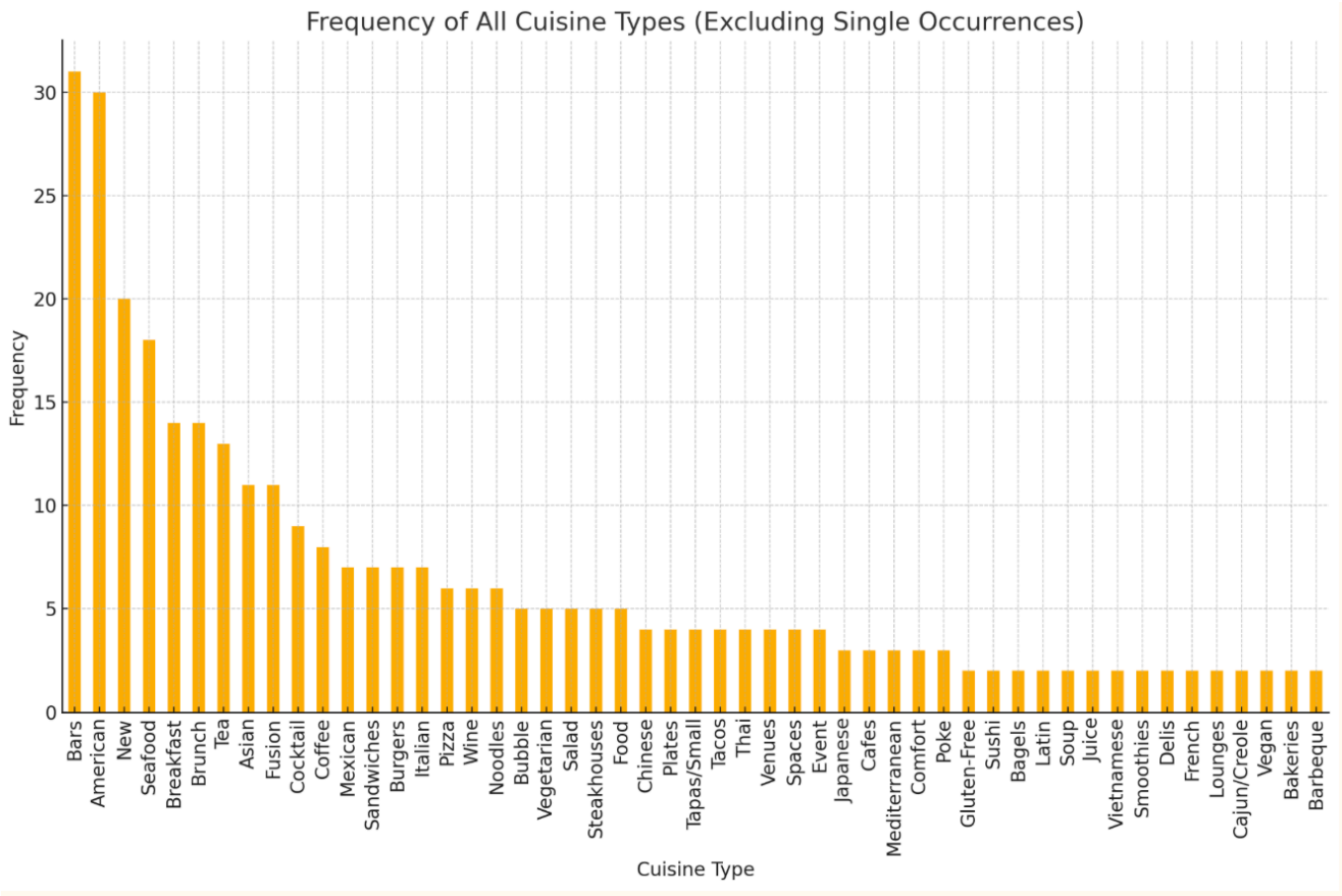
Not shown: Downloading `restaurants_cleaned` to a csv file and saving it in our shared google drive.

With the data fully cleaned we made visuals of some variables in the dataset to inform our choices when building the recommender systems.

```
# plotting revews and num_reviews
plt.figure(figsize = (10,6))

sns.scatterplot(data = restaurants_cleaned, x = 'num_reviews', y = 'rating')

# labeling specific restaurants in the plot
labeled_restaurants = ['Freebirds', 'Teddy Rice', 'Hana Kitchen', "Santa Barbara Shellfish Company"]
for i, row in restaurants.iterrows():
    if row['name'] in labeled_restaurants:
        plt.text(row['num_reviews'], row['rating'], '  ' + row['name'],
                 fontsize = 10, ha = 'left', color = 'black')

plt.title("Overall Impression of Restaurants in the SB/IV Area")
plt.xlabel("Number of Reviews")
plt.ylabel("Rating")

plt.show()
```




```
# plot of most common cuisine types
cuisine_counter = Counter(" ".join(data['descriptors'].dropna()).split())
common_cuisines = pd.DataFrame(cuisine_counter.most_common(10), columns=['Cuisine', 'Count'])
common_cuisines.set_index('Cuisine').plot(kind='bar')
plt.title('Top 10 Most Common Cuisine Types')
plt.ylabel('Frequency')
plt.show()
```



Recommender Systems

With our cleaned data, we're ready to implement recommender systems. We decided to test two different methods with one recommender using tf-idf values and another using bag of words.

TF-IDF Recommender

We first transform the price numerically and normalize the rating before combining the restaurant's name and listed cuisines into `Combined_Text`. This new variable is used to create a tf-idf matrix. This matrix is used with other factors to create the similarity scores shown in the top ranked restaurants.

```
price_map = {'$': 1, '$$': 2, '$$$': 3, '$$$$': 4}

restaurant_data['Price_Numeric'] = restaurant_data['PriceRange'].map(price_map).fillna(0)

restaurant_data['Rating_Normalized'] = restaurant_data['Rating'] / restaurant_data['Rating'].max()

restaurant_data['Combined_Text'] = (
    restaurant_data['Name'].fillna('') + ' ' +
    restaurant_data['Cuisine'].fillna('') + ' ' +
    restaurant_data['Cuisine1'].fillna('') + ' ' +
    restaurant_data['Cuisine2'].fillna('')
)

# Create a TfidfVectorizer instance and fit it on the restaurant data
vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = vectorizer.fit_transform(restaurant_data['Combined_Text'])
```

```
# Define a function to compute adjusted similarity considering price range and rating
def recommend_restaurants(user_input, tfidf_matrix, vectorizer, restaurant_data, preferred_price, top_n=5):
    # Transform the user input using the same vectorizer
    user_tfidf = vectorizer.transform([user_input])

    # Compute cosine similarity between the user input and restaurant data
```

```
    similarity_scores = cosine_similarity(user_tfidf, tfidf_matrix).flatten()

    # Adjust similarity based on price range
    preferred_price_numeric = price_map.get(preferred_price, 0)
    price_diff = abs(restaurant_data['Price_Numeric'] - preferred_price_numeric)
    price_penalty = 1 - (price_diff / max(price_map.values()))  # Scale to 0-1
    price_adjusted_scores = similarity_scores * price_penalty

    # Further adjust similarity based on ratings
    final_scores = price_adjusted_scores * restaurant_data['Rating_Normalized']

    # Get the top N restaurant indices sorted by final score
    top_indices = final_scores.argsort()[::-1][:top_n]

    # Get the top N restaurants with their final scores
    top_restaurants = [(restaurant_data.iloc[i], final_scores[i]) for i in top_indices]

    return top_restaurants
```

```
# Example user query and preferred price range
user_query = "I am in the mood for italian cuisine"
preferred_price = "$$"

# Get top recommendations based on the user query, price range, and rating
top_recommendations = recommend_restaurants(user_query, tfidf_matrix, vectorizer, restaurant_data, preferred_price)

# Display the top recommendations
print("Top restaurant recommendations:\n")
for i, (restaurant, score) in enumerate(top_recommendations):
    print(f"Rank {i + 1}: {restaurant['Name']} (Score: {score:.2f})")
    print(f"Location: {restaurant['City']}, {restaurant['State']}")
    print(f"Rating: {restaurant['Rating']}, Reviews: {restaurant['NumReviews']}")
    print(f"Price Range: {restaurant['PriceRange']}\n")
```

Top restaurant recommendations:

Rank 1: Ca' Dario - Goleta (Score: 0.24) Location: Goleta, CA Rating: 4.0, Reviews: 282 Price Range: $$

Rank 2: TAP Thai Cuisine (Score: 0.20) Location: Goleta, CA Rating: 3.6, Reviews: 97 Price Range: $$

Rank 3: Cajun Kitchen Cafe (Score: 0.00) Location: Goleta, CA Rating: 4.1, Reviews: 661 Price Range: $$

Rank 4: Choi's Oriental Market (Score: 0.00) Location: Santa Barbara, CA Rating: 4.5, Reviews: 241 Price Range: $$

Rank 5: Uniboil (Score: 0.00) Location: Goleta, CA Rating: 3.9, Reviews: 275 Price Range: $$

The number one recommended restaurant is indeed an italian restaurant, matching the cuisine we put in the query. However after that we see that it recommends TAP, which doesn't have pasta on its menu. Followed by three more restaurants that all don't have a single italian dish on their menu. This lack of results can be explained by the small size of our data sets.

Bag of Words Recommender

The bag of words recommender system uses cosine similarity scores from a count vectorizer on the descriptors plus the price description of the restaurants. These scores were used in the recommender system.

```
restaraunts['combined_features'] = restaraunts['descriptors'].fillna('') + ' ' + restaraunts['price_descrip'].fillna('')

count_vectorizer = CountVectorizer(stop_words='english')
count_matrix = count_vectorizer.fit_transform(restaraunts['combined_features'])
cosine_sim = cosine_similarity(count_matrix)
```

```
def recommend_restaurants(restaurant_name, top_n=5):
    # Get the index of the restaurant
    idx = restaraunts[restaraunts['name'] == restaurant_name].index[0]

    # Get pairwise similarity scores for the restaurant
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort restaurants by similarity score
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the top-n most similar restaurants
```

```
    sim_scores = sim_scores[1:top_n + 1]  # Exclude the restaurant itself

    # Get restaurant indices
    restaurant_indices = [i[0] for i in sim_scores]

    # Return the top-n most similar restaurants
    return restaraunts.iloc[restaurant_indices][['name', 'descriptors', 'price_descrip', 'rating']]
```

```
restaurant_name = "Zocalo"
top_recommendations = recommend_restaurants(restaurant_name, top_n=5)
print(top_recommendations)
```

name, descriptors, price_descrip, rating

1. The Shop, Brunch New American Coffee & Tea Breakfast & Brunch, moderate, 4.3
2. Helena Avenue Bakery, Bakeries Breakfast & Brunch Coffee & Tea, moderate, 4.3
3. Jeannine's American Bakery, Restaurant Coffee & Tea American, moderate, 4.4
4. On the Alley - Goleta, American Breakfast & Brunch, moderate, 3.8
5. Freebirds, Mexican, moderate, 4.1

Instead of using a query, we passed through a restaurant already contained in our data set. The first four recommendations focus on the breakfast and brunch aspect of Zocalo while the final recomomendation, Freebirds, focuses on Mexican cuisine.

Results and Going Forward

The goal of this project was to solve the timeless problem of friends coming up with a place to eat together. The recommender systems we built do well in providing recommendations that match a couple words in `descriptors` but not the entire description. The results both the tf-idf and bag of words systems mostly recommended breakfast places when `Zocalo` is input. While the recommenders did correctly recommend breakfast and brunch places in response, the recommendations are mostly lacking of the cuisine of the restaurant.

These results are proof that better, more elaborate systems with more variables could be built to improve recommendations to users. Without the query limitations of Yelp's Fusion API, we could expand the list of restaurants to a large area and begin to factor distance into the recommender. Using Yelp's reviews from the API we could also tokenize user's reviews to gather more descriptors for the restaurants that would aide the recommendation process. Additionally, we could expand the recommender to consider multiple users which would also take allergies and restaurant's hours to more accurately reflect real-world situations.