

FRAME - Food Recommendations for All Methodical Eaters

DATA 515 Technology Review - 21-Feb-2023

Zach Bowyer
Raman S V
Arjun Sharma
Adithyaa V

Background & Use Case

Background:

- Food delivery apps offer a lot of choices, leading to analysis paralysis and wasted time.
- Users often struggle to make confident decisions about what to order.

Use Case:

- Introducing a tool that provides a holistic approach to food ordering.
- Users can filter choices based on cuisine, dietary restrictions, price, allergens, and nutritional value.
- A curated list of 5-10 dishes and the restaurant from where it can be ordered is provided.
- Currently focusing on people living in the Seattle area who want to order food quickly and easily without being overwhelmed by options.

Why we do need libraries:

- Limited development time, no need to reinvent the wheel for proven algorithms/technologies when it comes to maps, data processing, data organization, etc.

Wireframe

Cuisine

Food Type

Misc. preference

Max Distance

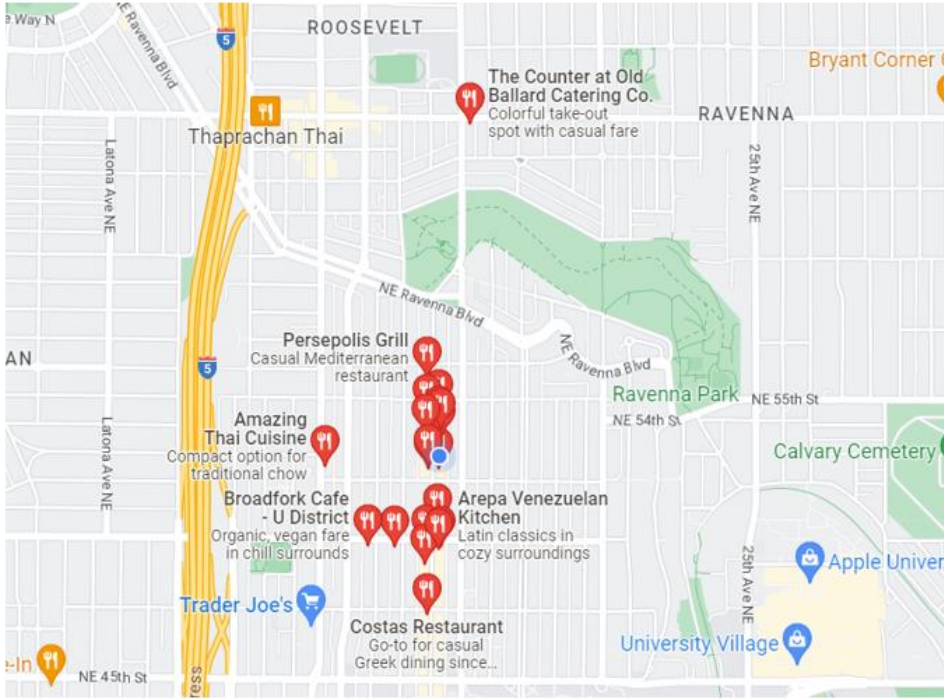
Price Range

Restaurant Rating

Search

Map View

List View



The Counter at Old Ballard Catering Co.
Colorful take-out spot with casual fare

Thaprachan Thai

Persepolis Grill
Casual Mediterranean restaurant

Amazing Thai Cuisine
Compact option for traditional chow

Broadfork Cafe - U District
Organic, vegan fare in chill surrounds

Trader Joe's

Costas Restaurant
Go-to for casual Greek dining since...

Arepa Venezuelan Kitchen
Latin classics in cozy surroundings

Ravenna Park

Calvary Cemetery

Apple University Village

University Village

Python Package Choices

Streamlit: Streamlit Inc.



- Framework that allows you to create web applications with Python.
- To create interactive and dynamic data visualization tools and dashboards.
- Has built-in mechanisms for handling user inputs and outputs, which could be useful for filtering food choices.

Yelp API: Yelp Inc.



- Allows us to access Yelp's vast database of business listings and reviews.
- To retrieve restaurant information such as location, hours of operation, and reviews,
- Provides the ability to filter search results based on various parameters such as price, rating, and cuisine type, which could be used to narrow down food choices.

Scikit-learn: Scikit-learn developers



- Machine learning library.
- Includes algorithms for clustering, classification, and regression, as well as tools for model selection and evaluation.
- Useful for developing the recommendation algorithm that powers the tool.

Python Package Choices (Continued)

Pandas: Pandas Development Team



- Library for data manipulation .
- Provides data structures for efficiently storing and manipulating large datasets, tools for filtering, sorting, and summarizing data.
- Useful for organizing and filtering the restaurant and menu data.

Folium: Folium Development Team



- Allows static and interactive world maps with street information.
- Allows overlaid images, tooltips, etc.
- Can embed graphs in tooltips

Google Maps: Alphabet Inc.



- Can be used to retrieve directions from point A to B
- Can generate static maps
- Can generate dynamic maps via ipython
- Can verify that addresses still exist
- Data pulled in real time

Package Comparison: Streamlit vs Flask

Parameters	Streamlit	Flask
Addressing requirements	Designed specifically for building data science and machine learning tools	More general purpose, offers more flexibility for complex requirements
Compatibility within project	Compatible with other Python libraries and frameworks	Has a large ecosystem of plugins and extensions
Ease of use	Simple and intuitive API, easy to use and learn for beginners.	Requires some knowledge of web development and backend tech
Computational efficiency	Efficient for handling smaller-scale datasets	Efficient for handling larger datasets and complex computations
Availability of relevant examples	Has as a growing community of users and a large number of examples available online	Has a large and active community with extensive documentation and resources

Package Comparison: Pandas vs Dask

Parameters	Pandas	Dask
Addressing requirements	Both can handle large datasets and provide a way to manipulate and analyze them using a variety of functions and methods	
Compatibility within project	Both are compatible with many other Python packages and can be integrated easily	
Ease of use	Easier to use, has a more straightforward API	Requires a high level of setup and configuration
Computational efficiency	Computationally less efficient, especially when dealing with very large datasets	Designed specifically to handle larger-than-memory datasets and can distribute computations across multiple cores or even multiple machines
Availability of relevant examples	Been around longer than Dask and is more widely used, so there are likely more examples available	Has a growing community and many resources are available as well

Package Comparison: Scikit-learn vs XGBoost

Parameters	Scikit-learn	XGBoost
Addressing requirements	Provides a wide range of algorithms	Focuses on gradient boosting methods
Compatibility within project	Both can be used with other packages and tools	
Ease of use	Has a user-friendly and consistent API which makes it easy to use	Can be a bit harder to use as it has more configuration options and hyperparameters to tune
Computational efficiency	Efficient enough, but may not perform as well as XGBoost on very large datasets	Has a high computational efficiency, especially when dealing with large datasets
Availability of relevant examples	Has a large and well-documented set of examples and tutorials	Less commonly used in some industries and domains

Package Comparison: 'Google Maps' vs 'Folium'

Parameters	Google Maps (connects to api)	Folium
Addressing requirements	Can be used to create static/interactive maps, provide directions, and verify if locations still exist.	Can be used to generate interactive maps, show tooltips, display markers, circles, polygons, etc.
Compatibility within project	Can give directions from point A to B in coordinates, which can be overlaid on a map.	Generated HTML files allow users to see where restaurants are located relative to themselves.
Ease of use	Requires api key on limited free trial. Have to use third party library due to no official support. API much more complicated.	Extremely simple
Computational efficiency	Partly depends on internet speed.	Millions of overlaid polygons will have performance issues.
Availability of relevant examples	Has an extremely large general user base. However many examples are ipython specific.	Has a solid community with decent documentation.

Package Choices

- Decided to use both Google Maps for coordinate data and folium for the actual visualization (Efficient)
- Streamlit: Easy, intuitive, interactive, Efficient enough
- Scikit-learn: Explainability, Robustness, Flexibility, Documentation
- Pandas: Simplicity, Compatibility, Familiarity

Drawbacks and Concerns

- Google Maps directions data only return the coordinate points at which turns are made. This means for long stretches of road we currently do not know how to construct directions with polylines.
- Google Maps API key is on a limited free plan right now, and its possible we may reach a request limit
- Removing elements from a folium map may require either browser code or a complete remake of the map.
- Streamlit: Deployment could be an issue
- Scikit-learn: Performance when compared to other models, Speed
- Pandas: We are very used to it and might not overlook. Speed, scalability, parallelization

Demonstration: Google Maps and Folium

Imports and API key

```
import folium
import json
import requests
import googlemaps
from datetime import datetime
gmaps = googlemaps.Client(key='your key here')
```

Get directions via Google Maps API

```
# Request directions via car  
now = datetime.now()  
directions = gmaps.directions("4555 Roosevelt Way NE, Seattle, WAS 98105",  
                              "6226 Seaview Ave NW, Seattle, WA, 98107",  
                              departure_time=now)  
  
print(directions)
```

As you can see, output needs to be parsed!

Parsing start/end destination coordinates, as well as trip coordinates (Lat, Lon)

Coordinate ranges: (-90 to 90), (-180 to 180)

```
endLocation = (directions[0]["legs"][0]["end_location"])
startLocation = (directions[0]["legs"][0]["start_location"])
startLat = startLocation['lat']
startLon = startLocation['lng']
endLat = endLocation['lat']
endLon = endLocation['lng']
print("Starting coordinates:", startLat, startLon)
print("Ending coordinates:", endLat, endLon)
```

```
Starting coordinates: 47.6629014 -122.3174096
Ending coordinates: 47.6748442 -122.4063584
```

Parsing start/end destination coordinates, as well as trip coordinates (Lat, Lon) Continued

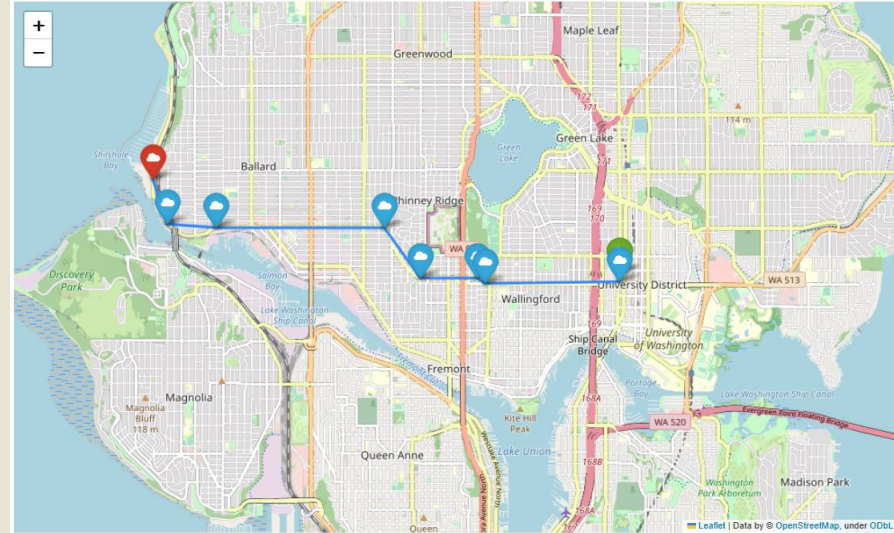
```
listCoords = []
steps = directions[0]['legs'][0]['steps']
counter = 1
listCoords.append([startLat, startLon])
for step in steps:
    #if(counter > 2):
    endlat = step["end_location"]["lat"]
    endlon = step["end_location"]["lng"]
    listCoords.append([endlat, endlon])
    #counter += 1
print(listCoords)
```

```
[[47.6629014, -122.3174096], [47.6616322, -122.3174549], [47.66139949999999, -122.3429996], [47.6621288, -122.3445939], [47.6621677, -122.3554293], [47.6686778, -122.3622056], [47.6686443, -122.394367], [47.6690321, -122.4036442], [47.6748442, -122.4063584]]
```

Adding markers/polylines on the folium map at the specified coordinates

```
m = folium.Map(location=[startLat, startLon])
counter = 1
for coords in listCoords:
    if(counter == len(listCoords)): continue
    folium.Marker(
        location=[coords[0], coords[1]],
        popup="Waypoint",
        icon=folium.Icon(icon="cloud"),
    ).add_to(m)
    counter += 1

# Polylines
folium.PolyLine(listCoords, tooltip="Coast").add_to(m)
```



Shows directions from start to finish. Markers can be customized to have different colors, icons, tooltips, hover rules, etc. The map is typically interactive but can be static if needed.

Pandas Demo

```
#Pandas Demo
import pandas as pd

data = {
    'name': ['Alice', 'Bob', 'Charlie', 'David', 'Peter'],
    'age': [25, 32, 18, 47, 28],
    'city': ['New York', 'London', 'Paris', 'Tokyo', 'Sydney'],
    'score': [85, 73, 92, 68, 79]
}

# create a pandas dataframe from the data
df = pd.DataFrame(data)
print('\n\nData:\n\n',df.head())

# filter the dataframe based on a condition
filtered_df = df[df['score'] > 80]
print('\n\nFiltered Data:\n\n',filtered_df)

# group the dataframe based on a column and calculate the mean
grouped_df = df.groupby('city').mean()
print('\n\nGrouped Data:\n\n',grouped_df)
```

Data:

	name	age	city	score
0	Alice	25	New York	85
1	Bob	32	London	73
2	Charlie	18	Paris	92
3	David	47	Tokyo	68
4	Peter	28	Sydney	79

Filtered Data:

	name	age	city	score
0	Alice	25	New York	85
2	Charlie	18	Paris	92

Grouped Data:

	age	score
city		
London	32	73
New York	25	85
Paris	18	92
Sydney	28	79
Tokyo	47	68

Scikit-learn Demo:

```
#Scikit-learn Demo
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

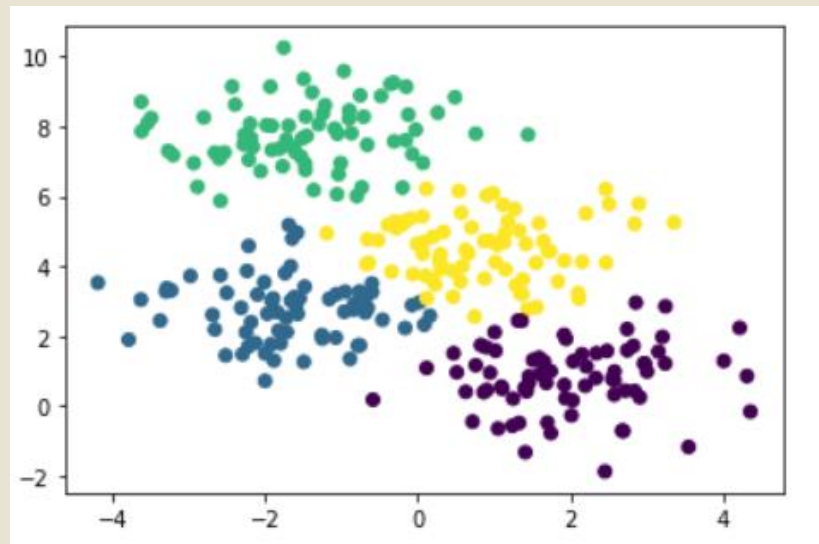
# Generate some data to cluster using make_blobs data
X, y = make_blobs(n_samples=300, centers=4, random_state=0)

# Initialize the KMeans algorithm
kmeans = KMeans(n_clusters=4, random_state=0)

# Fit the model to the data
kmeans.fit(X)

# Get the predicted labels for each data point
labels = kmeans.predict(X)

# Plot the data points with different colors based on their labels
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.show()
```



Streamlit Demonstration

```
# Importing Libraries

import numpy as np
import pandas as pd
import streamlit as st
import pgeocode

def zip_to_coordinates(zip_code):
    country = pgeocode.Nominatim('US')
    lat, long = country.query_postal_code(zip_code).latitude, country.query_postal_code(zip_code).longitude
    return lat, long

def recommend_food(zip_input, max_dist_input, allergy_input, diet_input, price_input, misc_input, rating_input):
    latitude, longitude = zip_to_coordinates(zip_input)
    return

def main():

    st.title("FRAME - Food Recommendations for ALL Methodical Eaters")
    html_temp = """
    <div style = 'background-color: tomato; padding: 60px>
    <h3 style = 'color: white; text-align: center;'></h3>
    <h3 style = 'color: white; text-align: center;'>Hungry but don't know what you want?</h3>
    <h3 style = 'color: white; text-align: center;'>Enter your preferences below, get recommendations!</h3>
    </div>
    """
    st.markdown(html_temp, unsafe_allow_html = True)

    st.subheader("")
    st.text_input("Enter your Name: ", 'John Smith', key="name")

    st.subheader("Please enter the required information below.")
```

FRAME - Food Recommendations for All Methodical Eaters

Hungry but don't know what you want?

Enter your preferences below, get recommendations!

Enter your Name:

John Smith

Please enter the required information below.

Enter your zip code:

00000

Enter your maximum distance preference (miles):

0.0

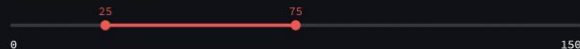
Any Allergies?

Choose an option

Enter if you have any dietary preferences:

- ☒ Vegetarian
☐ Vegan
☐ None
☐ Other

Enter your preferred price range:



Miscellaneous Preferences:

Choose an option

Enter restaurant rating preferences (on a scale of 5):

★ & Up

Get FRAMed!

Success! We've got some delicious recommendations for you!



587820666