Open Street Map Tour Planner

CMPT 353 Computational Data Science Final Project

Jiangpei Chen, 301326516
Nontawat Janpongsri, 301311427
December 11th, 2020

Introduction

OpenStreetMap (OSM) is a collaborative project to create a free editable map of the world[1]. It contains numerous data information of places such as points of interest (POIs), water fountains, restaurants, libraries, and bus stops. For this project, we decided to use the OSM data in Metro Vancouver which was provided by professor [2]. The JSON file from the OSM contains the latitude and longitude, timestamp (when the point was added), amenity type, and related tag information. We will be mainly using this OSM, but we will also be collecting some other adds-on dataset as supplements.
In this project, we will be applying inferential statistics, machine learning, and other data analysis skills to the data to formulate a list of attractions, determine if there is a relationship between 2 groups, and find the area in Vancouver with the most resting spots.

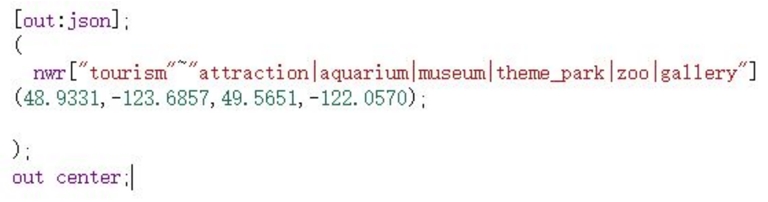
Idea Creation

Tour planning is always a painful process. With the idea "if I was planning a tour of the city, where should I go?" provided on the project page [2], we have decided to come up with a program that will do all the tour planning for you in the city of Vancouver. The program will take an image and travel mode as input from the user to analyze what the user might be interested in for the tour. For the image input, the program will decide whether the user would like to visit the attraction first or the restaurant first. This is done by checking the location that the picture had been taken. If taken near a restaurant then assume the user would like to visit the restaurant first. Otherwise, assume that the user would like to visit the attraction first. The restaurant and attraction type in the list would be calculated based on the user travel mode. If the user decides to walk, the program will return any type of restaurants and attractions in the data list that are the closest to them. If the travel mode is a bike, then the restaurant type would be fast food, and the attraction would be a park. If a car, then the restaurant type would be fine dining, pub or bar and the attraction would be a famous tourist attraction.

Project Step analyze

Based on the scale of the project, we divided our project into roughly eight parts.

1) **Data Collection:**
   In the project, we are mainly working on the data that was provided by the professor, but after opening the amenities-vancouver.json dataset, we found that it does not contain much tourism information. So we decided to add more data to our main dataset.
   Since we are mainly focusing on Vancouver, we went to the City of Vancouver's open data website [3] and found the park information dataset.
   We write a python file (read_park_data.py) to parse the park dataset and then append it to our main dataset.
   Besides the park data, we also want to add more tourism spots such as museums, attractions, theme_park, so we used the Overpass-API [4], restricting the area to Metro Vancouver.
   The script we used to extract the tourism information.



```
[out:json];
(
  nwr["tourism"~"attraction|aquarium|museum|theme_park|zoo|gallery"]
(48.9331,-123.6857,49.5651,-122.0570);

);
out center;
```

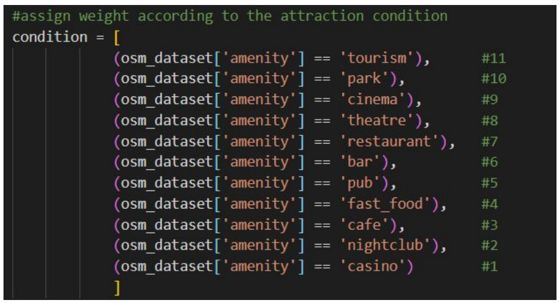
   It returns the data in 3 different elements, Nodes, Ways, as well as Relations.
   We write a python file (read_attractive.py)to parse all of those tourism POIs and then append it to our main dataset.
   So our main dataset is combining the amenities-vancouver.json, park data in Vancouver, tourism POIs in Metro Vancouver. (concatenate_data.py)

2) **Finding location from the image that the user inputs:**
   We want to know user interests based on the user's input image's location, so we need to extract the EXIF information from the image.
   We used an external library to help us to extract the information from the user's image. The python file (extract_gps.py) will read a geotagged image from the command line by the user's input and will return a (latitude, longitude) tuple of the image's location.

3) **Finding the distance between the user image input location and points in dataset**
   We wrote a python function (distance_calculate.py) using the Haversine formula to help us calculate the distance between two geographic coordinates (user's image and points in the dataset) [5].

4) **Detect what the user is interested in visiting first base on the input image location:**
   We want to detect user interests based on the location of the image.
   For example, the user's input image location is in or near Canada Place, a tourism point. Then our program will provide a tour starting with tourism spots and a restaurant in between for a break. If the user's input image location is in

or near a restaurant, then our program will provide a tour starting/ending with a restaurant and with tourism spots in between.
To detect what the user is interested in visiting first, we did the following:
Step 1:
We assigned weight to each of the points in our main dataset, based on the amenity type.



```
#assign weight according to the attraction condition
condition = [
            (osm_dataset['amenity'] == 'tourism'),      #11
            (osm_dataset['amenity'] == 'park'),         #10
            (osm_dataset['amenity'] == 'cinema'),       #9
            (osm_dataset['amenity'] == 'theatre'),      #8
            (osm_dataset['amenity'] == 'restaurant'),   #7
            (osm_dataset['amenity'] == 'bar'),          #6
            (osm_dataset['amenity'] == 'pub'),          #5
            (osm_dataset['amenity'] == 'fast_food'),    #4
            (osm_dataset['amenity'] == 'cafe'),         #3
            (osm_dataset['amenity'] == 'nightclub'),    #2
            (osm_dataset['amenity'] == 'casino')        #1
            ]
```

Step 2:
we will select a point from the dataset that has a distance less or equal to 100m with the highest weight assigned to it as the user first interest
Step 3:
If the point that we have selected has a weight > 7 our program will mainly focus on tourism.

- Our program will create the tour list consist of 2Tourism + 1Restaurant + 2Tourism

If the point that we have selected has a weight <= 7 our program will mainly focus on food.

- Our program will create the tour list consist of 1Restaurant + 2Tourism + 1Restaurant + 1Tourism

5) **Cleaning data**
Since we are mainly focusing on food and tourism, there are many points in our dataset that we do not care about or are unable to work with it. These points would be necessary to remove from our dataset so that it would not be a burden for our future calculation. The points with the following conditions would be remove from our dataset:
   1) There is no name given for that point
   2) Depending on the traveling mode remove points that has the following conditions (for further information read the note down below):
      a) Walk -> weight = 0
      b) Bike -> weight != 10, 4, 3
      c) Drive -> tourism tag is empty and weight != 5,6,7
   3) Depending on the traveling mode remove points that has distance greater than:
      a) Walk -> 1km
      b) Bike -> 10km
      c) Drive -> 50km

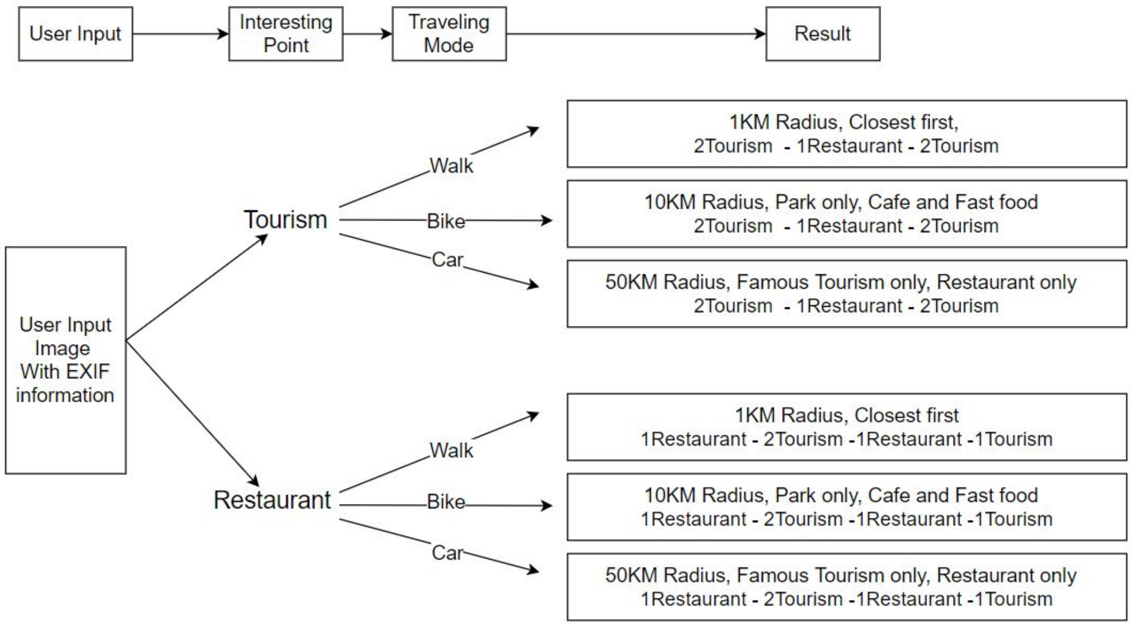
Note:

- If the user's travel mode is walking, our program will only remove the points that are neither an attraction nor a restaurant and are not within a 1km distance away from the user input.
- If the user's travel mode is biking, our program will assume that the user is interested in nature. So, it will remove any points that are not a park and fast food or cafe for a restaurant because the user would likely be on the go.
- If the user's travel mode is driving, our program will filter out the non-famous tourist attraction by using tags. If the point has a 'tourism' tag, our program will consider it as being famous. For restaurants, the program will only keep the one that has a weight that was assigned to it less than equal to 7 but more than 4 (fine dining only).

6) **Start searching for POIs**

After our program finalized the user's interest (either Tourism or Restaurant), and the candidate POIs have been filtered out with the corresponding range and travel method (1km for a walk, 10km for a bike, 50km for a car), our program will start searching for POIs.We are going to use the greedy algorithm to determine which POIs to select for our tour list. The POIs would be chosen based on which one has the shortest distance from the previous POI.

Algorithm flow chart


User Input
Interesting Point
Traveling Mode
Result
User Input Image With EXIF information
Tourism
Walk
Bike
Car
1KM Radius, Closest first, 2Tourism - 1Restaurant - 2Tourism
10KM Radius, Park only, Cafe and Fast food 2Tourism - 1Restaurant - 2Tourism
50KM Radius, Famous Tourism only, Restaurant only 2Tourism - 1Restaurant - 2Tourism
Restaurant
Walk
Bike
Car
1KM Radius, Closest first 1Restaurant - 2Tourism -1Restaurant -1Tourism
10KM Radius, Park only, Cafe and Fast food 1Restaurant - 2Tourism -1Restaurant -1Tourism
50KM Radius, Famous Tourism only, Restaurant only 1Restaurant - 2Tourism -1Restaurant -1Tourism

7) **Using the google api find the path that the user will take from each location**

After our algorithm finishes POI searching, it will return 5 POIs that our program suggested to go. It contains the name, geographic coordinate, relative tags of those five POIs.

Then we used Google Direction API [6] to help users to plan their route for those POIs with different traveling modes. Calling the API with start location, a destination location, traveling mode, then Google Direction API will return the path information from the start location to the destination location as a JSON file.
Because our program returns 5 geographic coordinates, to let the user visit all the POIs at once, we decided to use the google direction API waypoints feature. We set the image location as the start location in google API, then adding each of the POIs' geographic coordinates as waypoints, add the last POI as the destination. In the end, we set the traveling mode with the user's input traveling mode. Finally, our API call URL has been built.

One sample API call:

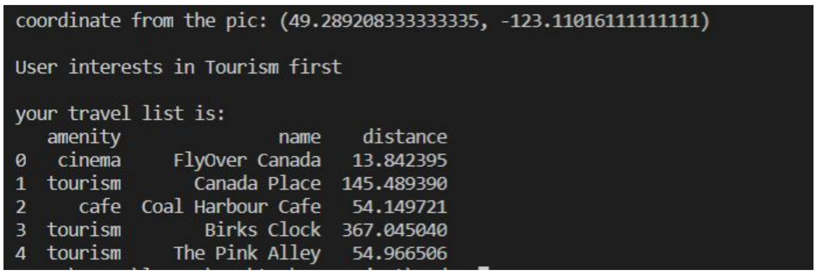
```
https://maps.googleapis.com/maps/api/directions/json?origin=49.288452138888886,-123.
11507413888889&destination=49.285911,-123.120948&waypoints=49.2887562,-123.1150926|4
9.2893832,-123.117621|49.288042,-123.119169|49.2875974,-123.1191416|&mode=bicycling&
language=en&key=AIzaSyC_OqhNkCgXZJlg5O3JEIGN4kHlF7YxKKo"
```

8) **Parse the Json file returned by Google API and display on map.**
After we use the API URL to call the Google API, it will return the turning points that we were supposed to take along the path in the JSON file. By connecting those turning points, we will be able to display our whole pathways on the map. To visualize the result, we decided to output the result as a geojson file, so that the user can view it using any online geojson file viewers (www.geojson.io). Because we have both line segments which represent the touring path, and points that represent visiting points. We save those line segments as LineString Objects from geojson library, and those points as Point Objects from geojson library. At the end we use the GeometryCollection as a container to save the paths and points, then output as a geojson file.

Result Case study

1) User input image
   Image taken in Flyover Canada (Near Canada Place) (49.289208, -123.1101611)
   Travel mode: walk
   Running command: **python .\main.py img2.jpg walk**

Output from Terminal

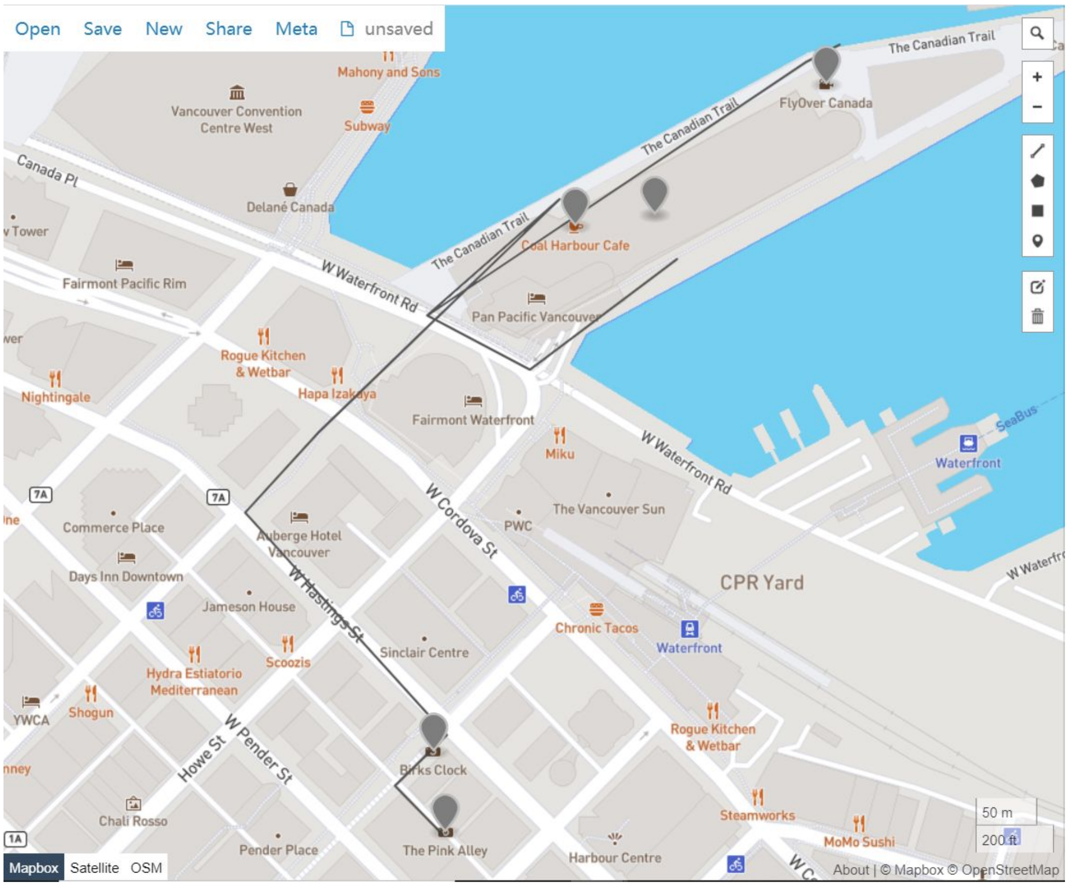


```
coordinate from the pic: (49.289208333333335, -123.11016111111111)

User interests in Tourism first

your travel list is:
   amenity               name    distance
0   cinema     FlyOver Canada   13.842395
1  tourism       Canada Place  145.489390
2     cafe  Coal Harbour Cafe   54.149721
3  tourism        Birks Clock  367.045040
4  tourism     The Pink Alley   54.966506
```

Viewing the geojson file

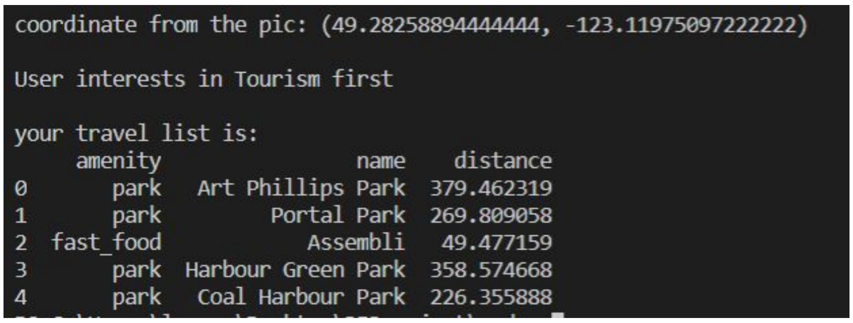


Case analyze:
Based on the algorithm chart, the image was taken near a tourism spot. So, the tour will be mainly focusing on Tourism. Because the user chooses to walk, the POIs will be restricted in a 1KM radius of the input image location.

So our program provided a route for the user to visit as:
1 FlyOver Canada (Tourism)
2 Canada Place (Tourism)
2 Coal Harbour Cafe (Cafe)
3 Birks Clock (Tourism)
4 The Pink Alley (Tourism)

2) Image taken in Vancouver Art Gallery (49.2825889, -123.11975097)
Travel mode: bike
Running command: **python .\main.py img15.jpg bike**

Output from Terminal

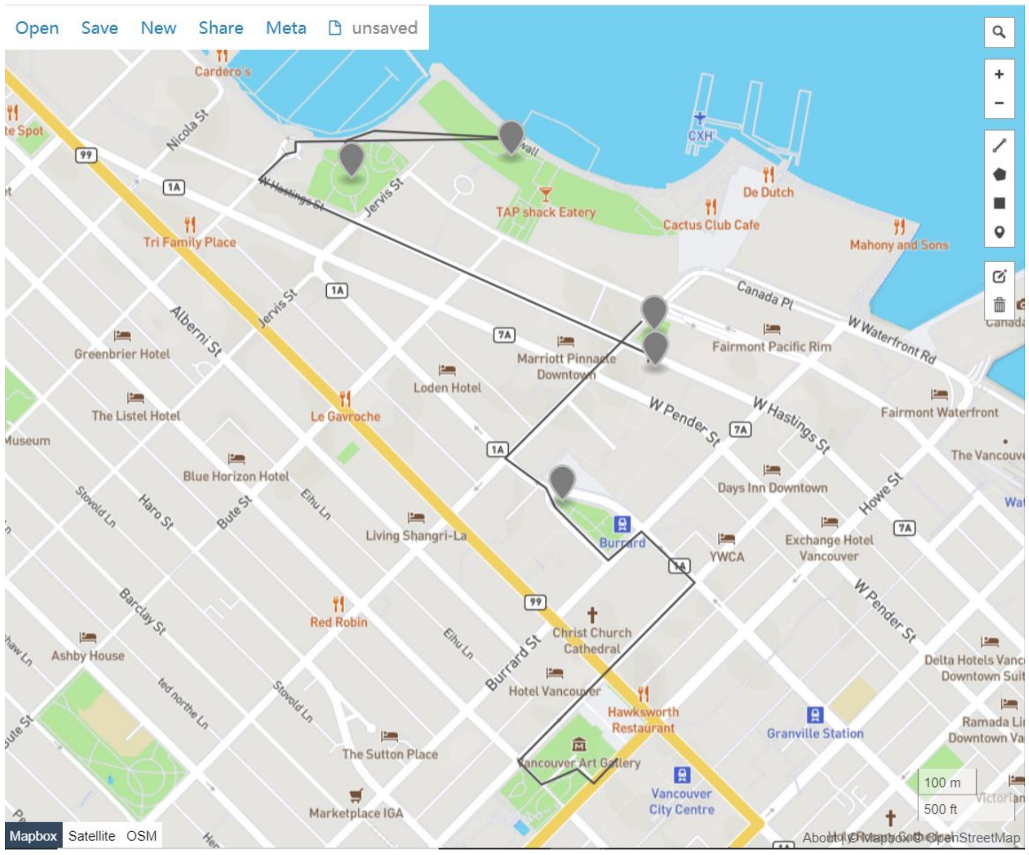


```
coordinate from the pic: (49.28258894444444, -123.11975097222222)

User interests in Tourism first

your travel list is:
     amenity                name    distance
0       park   Art Phillips Park  379.462319
1       park         Portal Park  269.809058
2  fast_food            Assembli   49.477159
3       park  Harbour Green Park  358.574668
4       park    Coal Harbour Park  226.355888
```

Viewing the geojson file



Case analyze:

Based on the algorithm chart, the image was taken near a tourism spot. So, the tour will mainly focus on Tourism. Because the user chooses to bike, the POIs will be restricted to parks and fast food within a 10KM radius of the input image location.

So our program provided a route for the user to visit as:

1 Art Phillips Park (Tourism)

2 Portal Park (Tourism)

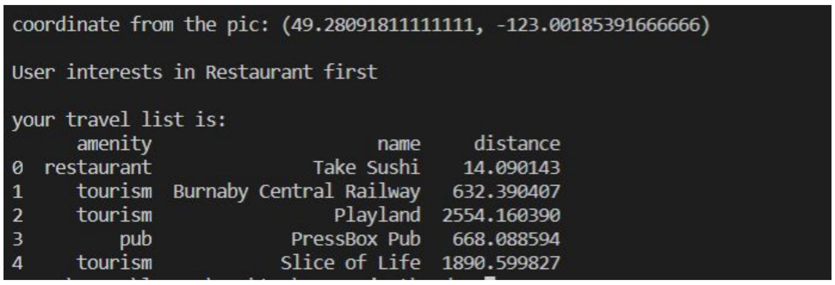
3 Assembli (Restaurant)

4 Harbour Green Park (Tourism)

5 Coal Harbour Park (Tourism)

3) Image taken near Hasting Street, Burnaby (49.280918, -123.0018539)
Travel mode: Drive
Running command: **python .\main.py img12.jpg drive**

Output from Terminal

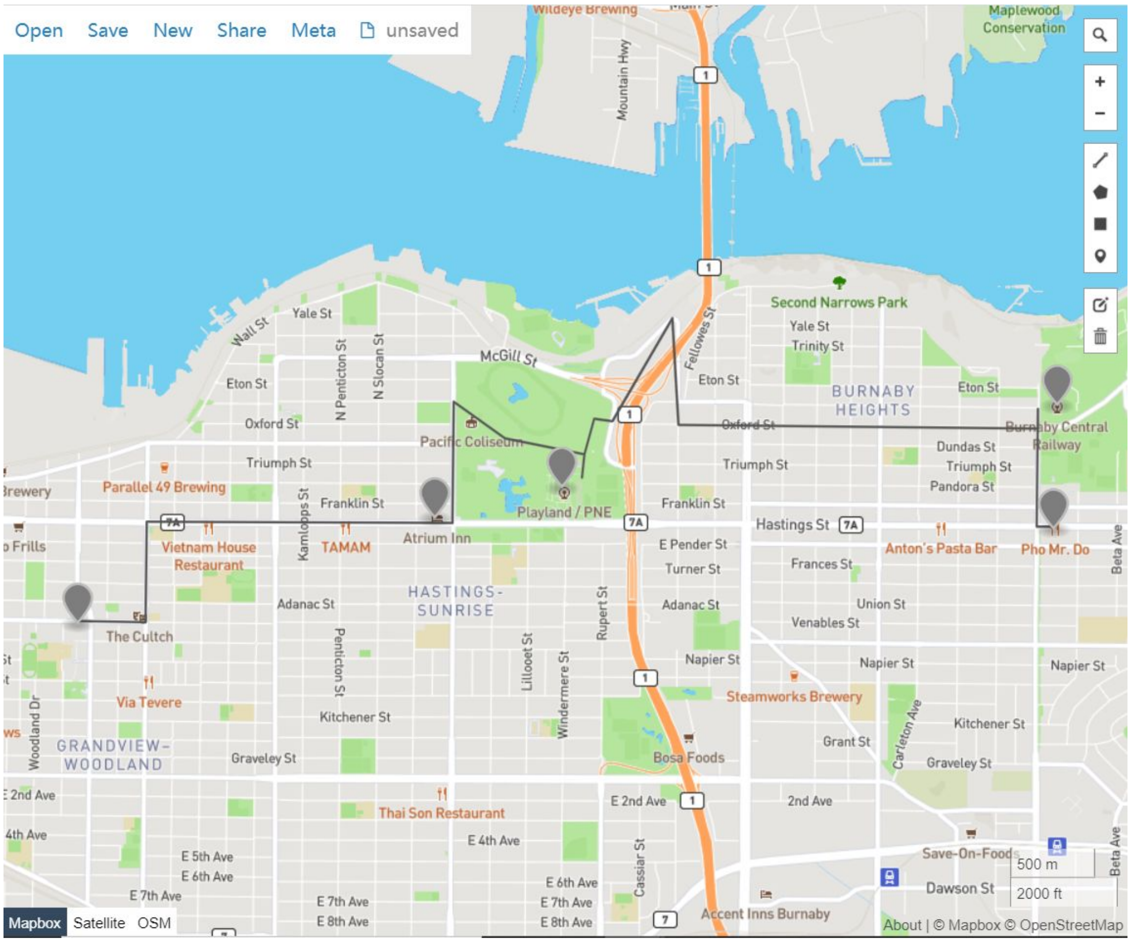


```
coordinate from the pic: (49.28091811111111, -123.00185391666666)

User interests in Restaurant first

your travel list is:
      amenity                   name     distance
0  restaurant             Take Sushi    14.090143
1     tourism  Burnaby Central Railway   632.390407
2     tourism               Playland  2554.160390
3         pub            PressBox Pub   668.088594
4     tourism          Slice of Life  1890.599827
```

Viewing the geojson file



Case analyze:
Based on the algorithm chart, the image was taken near a Restaurant, so that the tour will mainly focus on Restaurant. Because the user chooses to drive, the POIs will be restricted to famous tourist spots and fine dining within a 10KM radius of the input image location.

So our program provided a route for the user to visit as:
1 Take Sushi (Restaurant)
2 Burnaby Central Railway (Tourism)
3 Playland (Tourism)
4 PressBox Pub (Restaurant)
5 Slice of Life (Tourism)

Limitations

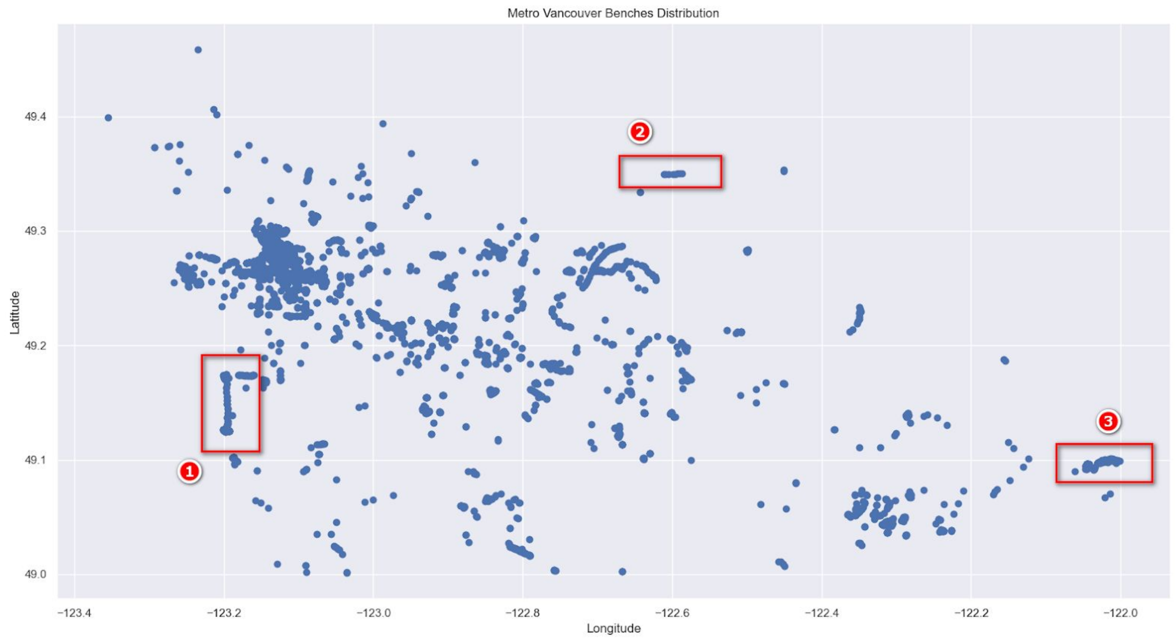
1) Dataset is imperfect. For example, we can only find the park data for Vancouver. And, in some cities such as Burnaby and Richmond, we could not find many useful datasets.
2) The tour path that is illustrated on the map is not smooth enough, because we are connecting the start point and destination point into a line segment object in geojson. So, if the road is not straight, the path will not match the road exactly. If we have more time, we might find some other way to make the paths more smooth.

Side Question Analyze

During the data collection and data analysis part of our project, we have some questions that we want to investigate further. Here is our Side-Question analysis part.

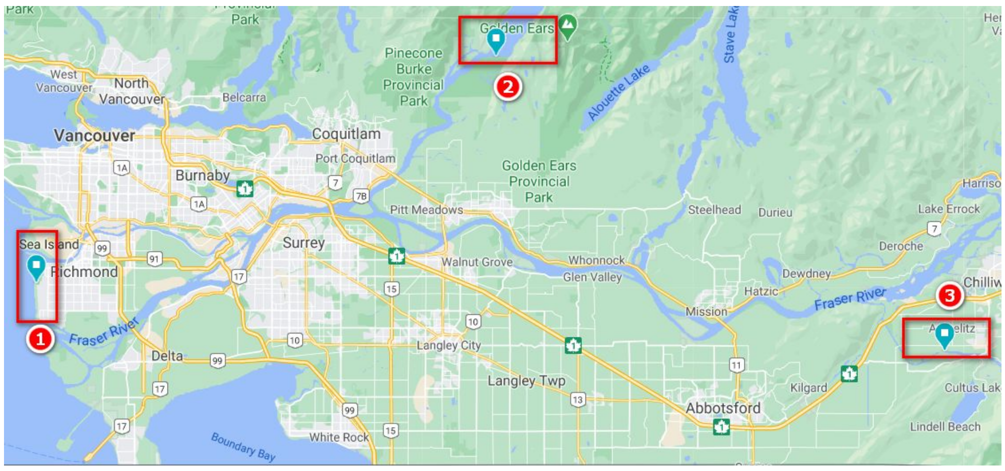
1) During the data collection part, I found that there are many data points having amenity = benches, most of them only have related geographic location, so they are not too useful in our main project, so we decided to filter them out when we were doing our main project.
As Metro Vancouver is a relatively large region, I want to know where those benches are located in Metro Vancouver? Which region of Metro Vancouver has more benches? Are we able to visualize them?

Scatter plot of Metro Vancouver bench Distribution (Data from OSM)



With the scatter plot, I found that there are some interesting points that I should take a look. On the scatter plot, you can see there are several small clusters, the benches inside of those small clusters almost align into a line shape, why are they lined up in this way compared with the rest of the benches? I chose 1 point inside of each red box, record the geographic location of them, then search up on the map.

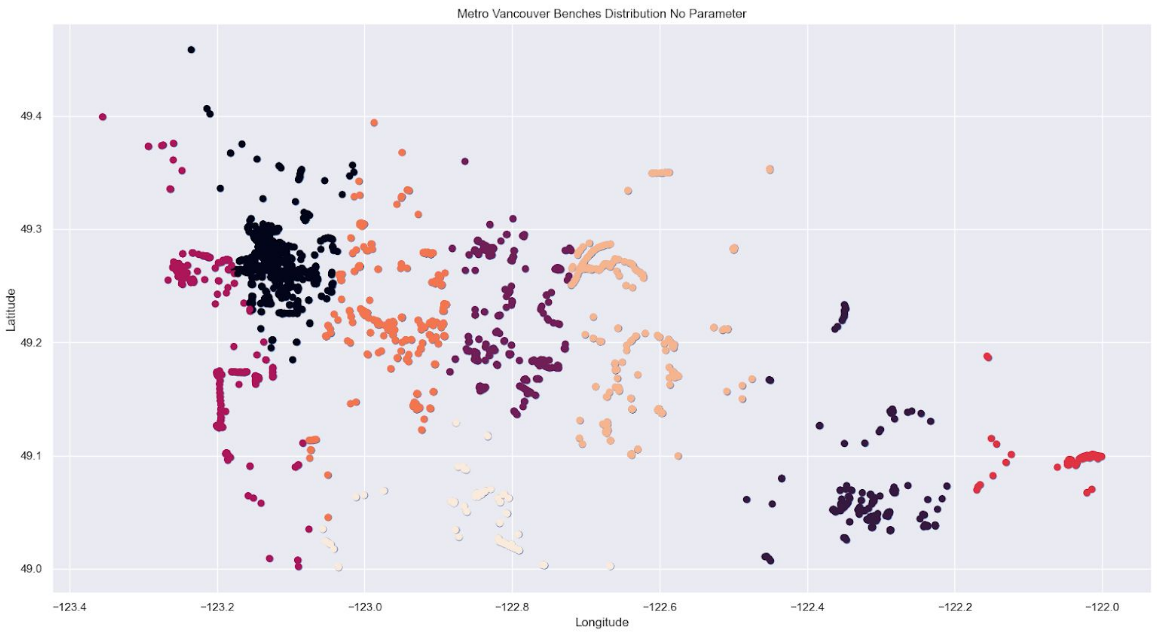
Box #	Coordinate	Area
1	49.158400,-123.19700	Richmood Beach
2	49.350200,-122.60000	Golden Ears
3	49.100300,-122.01700	Chilliwack River Shore



After plotting those bench clusters on the map, now I know the reason why they are aligned into a-line shape because those benches are located near the beach or river shore. Box 1 is in the beach of Richmond, Box 2 is in a reserved area in Golden Ears, and Box 3 is in the Chilliwack River Shore.

In order to figure out the question, “Which region of Metro Vancouver has more benches?” I decided to run *KMeans algorithms* for the bench data.

K Means without Parameter specific the cluster size



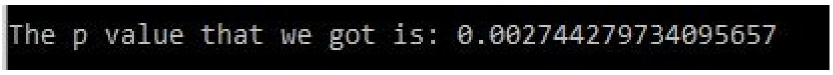
After running the KMeans algorithm, we can see that the density of benches is higher in the black cluster compared with the rest. The black cluster, which is Downtown Vancouver. **So we have a conclusion, that is, in Metro Vancouver, with the current OSM data, Downtown Vancouver has relatively more benches than other areas of Metro Vancouver.**

2) Because there are so many benches on our dataset, I would like to find out if the quality of the bench depends on the area that you are currently in or not. The quality of the bench can be divided into two groups, one with a backrest and the other without. We can determine if the bench has a backrest or not by using its tag. If there is a backrest tag in the bench data point, that means the bench has a backrest. For the area, I decided to compare a park with the central city area. The benches data for the park group are collected within a 5km radius around Stanley Park. The benches data for the central city area are collected within a 5km radius around a restaurant called Gotham.

After cleaning and collecting our data i manage to find this:

	With backrest	**Without backrest**
Park	445	703
Central city	657	814

By using one of the inferential statistics tests called Chi-Square, I can determine if the area has any effect on the bench quality.
The resulted p-value after running the test is:



```
The p value that we got is: 0.002744279734095657
```

Because p < 0.05, we can conclude that the area has some effect on the bench quality you get.

Project Experience Summary

Jiangpei Chen

- Dataset collection from Vancouver Open data website and overpass API.
- Data extraction and Data Concatenate.
- Implements the greedy algorithm for POI searching.
- Explore the google direction API and understand the API query string.
- Create structure of the report and write a comprehensive report.
- Side-question: Bench Density Analysis.

Nontawat Janpongsri

- Implements an algorithm to detect the place that the user would like to visit first.
- Implements data cleaning to the data to accommodate all the different travel modes for the future used.
- Parse the json file and save the paths and points into geojson file.
- Wrote a report with a detailed explanation for the reader to read.
- Refine the report so that the reader would have an easier time reading it.
- Using inferential statistics to determine the existence of the relationship between the city area and bench quality.

Reference Page

[1] https://en.wikipedia.org/wiki/OpenStreetMap OSM info
[2] https://coursys.sfu.ca/2020fa-cmpt-353-d1/pages/ProjectTour Project page
[3] https://opendata.vancouver.ca/explore/dataset/parks/information/ park info
[4] https://overpass-turbo.eu/ Tourism data
[5] https://stackoverflow.com/a/21623206 Distance calculate
[6] https://developers.google.com/maps/documentation/directions/overview Direction api