# Tour Vancouver with OSM

Xingjian Ding
Zhi Feng
Gengtao Zhao

## Table of Contents

## I. Overview

To Better Help people have a good vacation in Vancouver, British Columbia, We analyze several aspects of the OSM data in Vancouver including Public Transport, Catering Service, and Tourists Attractions. By building the models using machine learning, we aim to provide useful recommendations for people who are struggling to choose where to live, where to dine, and where to visit.

Xingjian Ding's Experience Summary: To get ideal Airbnb locations with convenient public transport, I extract Public transport data from OSM using spark and use DBSCAN to find public transport dense areas and calculate their centroids. Lastly, get all the ideal housings.

Zhi Feng's Experience Summary：I mainly analyzed restaurant data in the Greater Vancouver area. "DBSCAN Clustering", "heat maps" and other methods were used to show the distribution and density of chain and non-chain restaurants. I compared the number, distribution, and density of chain and non-chain restaurants and drew some conclusions.

Gengtao Zhao's Experience Summary: Extract the ideal activities category from the data cluster. For accessing users' locations, use EXIF, Geopy, and Nominatim for geo-data transformation, and Tkinter for file access to read geotag images. Use KNneighborclassifier and haversine_distances for the nearest activity location around the user and its closest neighbors.

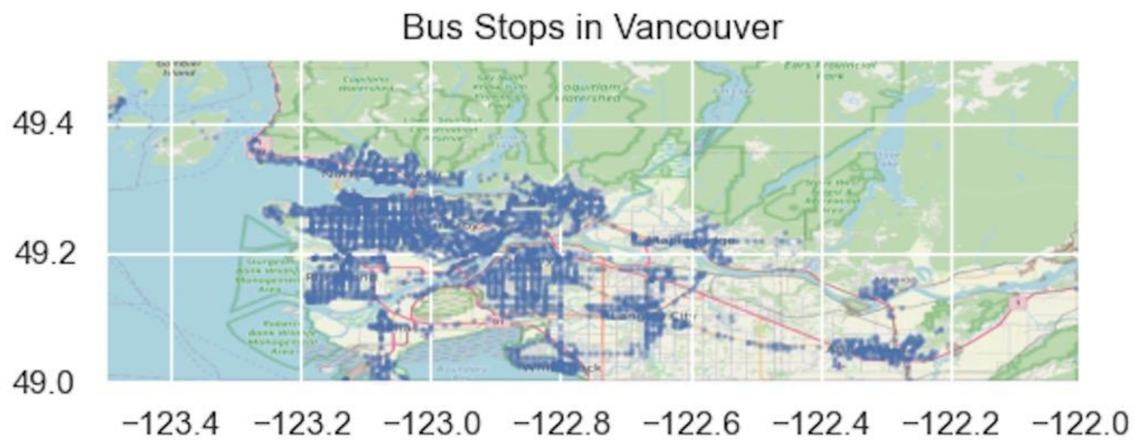
## II. Public Transport

- ***Introduction***

The common statistic to define convenient access to public transportation is the percentage of the population living within 500 meters of a public transport access point according to Statistics Canada[1]. Based on this definition, we would like to find places with dense public transport platforms including bus stops and Skytrain. They might represent the most convenient places in Vancouver and the Airbnbs located near their center could be great choices for people who are seeking convenient transportation.

- ***Data:***

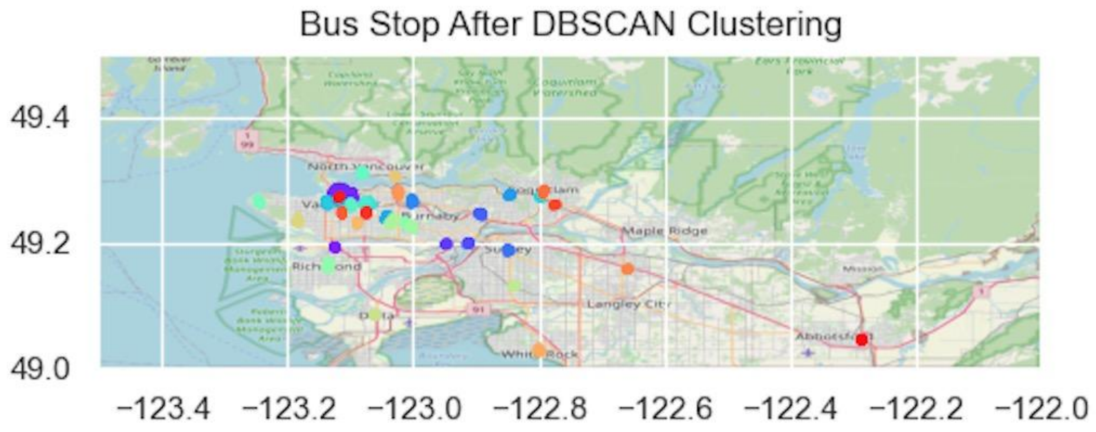
(1). ***Transport_stop-vancouver.json:*** The data used in this problem is from the planet.osm which contains all the information across the world along with all the Airbnb listings in Vancouver. Due to the huge volume of the OSM file in HDFS /courses/datasets/openstreetmaps , we select the nodes with the tag field "public transport" using spark. Then, we restrict the latitude and longitude for roughly Metro Vancouver, resulting in a JSON file with about 9000 nodes representing nearly 9000 bus stops as the graph "Bus Stops In Vancouver" is shown below. These two ETL tasks are done by osm-transport_stop.py and just-vancouver.py
(2). ***Listings.csv:*** For the Airbnb listing, we specify certain requirements to simulate a person traveling alone for one bedroom with an accommodation budget below 200$ per night. This is achieved by converting the values in the price column from type string to type float[4].



- ***Model:***

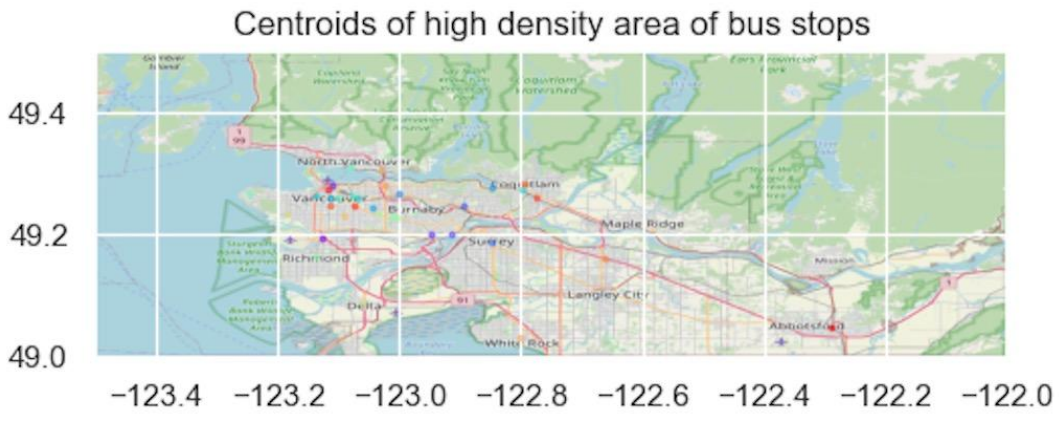
It is obvious that the distribution of bus stops is unbalanced. To capture the densest area, we use the DBSCAN clustering model to label every point so that points with the same label can form a cluster. The parameters epsilon and minimum samples are carefully chosen, reaching 35 clusters.

It is worth noting that parameters algorithm='ball_tree' and metric='haversine' help us deal with the latitude and longitude which are angles and cannot be handled using the Euclidean Distance Formula[5]. Looking at the data in the graph "Bus Stop after DBSCAN Clustering", we find that the downtown area forms the largest cluster with almost 200 bus stops and the nearest cluster from SFU is the Lougheed area.



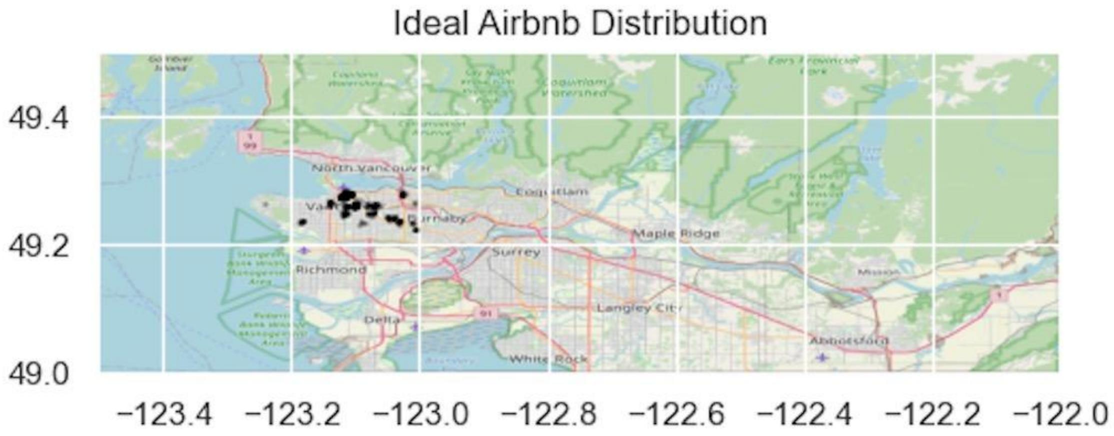
- ***Further Data Processing:***

DBSCAN differs from other clustering methods like K-Means in that it does not produce any centroids by itself. Thus we generated 35 theoretical "centroids" with the mean of all points' latitude and longitude. This way, we could limit our searches to housing in the Airbnb listings near these centroids. The function "distanceToNearestCluster(point_lon,point_lat, centroids)" takes every housing's coordinates and all the cluster centroids as input and calculates the distance between the housing and its nearest centroid using function distance(lat1, lon1, lat2, lon2)[3]. Then we apply the function to every Airbnb housing and put them into one data frame "listings".



- ***Results:***

With the definition of convenient access to public transport in mind, we define a threshold of 0.5 km for the “distanceToCluster” column of data frame “listings” to filter out those located more than 500 meters away from dense area centroids.
Lastly, we output the ideal Airbnb listing as a CSV file that provides good Airbnb choices for users.



- ***Visualization:***

The visualization is done by combining a scatter plot with a Vancouver image exported from OpenStreetMap[6]. To illustrate different cluster points, we use different colors to represent clusters and centroids using cm’s rainbow method in matplotlib to iterate 35 colors[2].

## III. Restaurant Analysis

- ***Data Preprocessing***

1) Find Chain Info For Restaurant

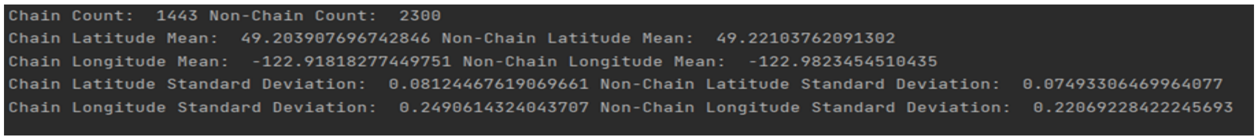
First, we read data from “amenities-vancouver.json.gz” and then add a chain or non-chain information to those restaurants. Those data are the basis for subsequent analysis.

2) Clean Data

Drop all empty data and duplicate data here.

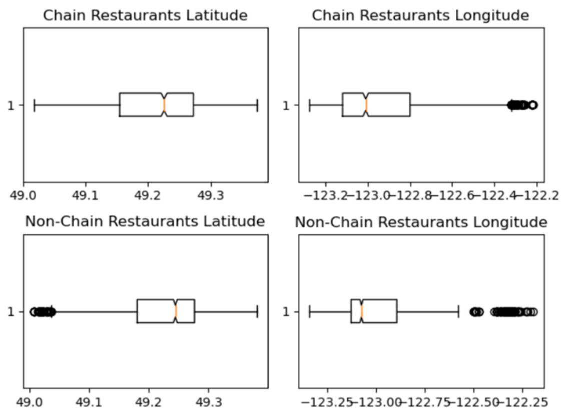
3) Data Description

Divide those restaurants into the chain and non-chain restaurants, and calculate the mean and variance of their latitude and longitude respectively.



```
Chain Count:  1443 Non-Chain Count:  2300
Chain Latitude Mean:  49.203907696742846 Non-Chain Latitude Mean:  49.22103762091302
Chain Longitude Mean:  -122.91818277449751 Non-Chain Longitude Mean:  -122.9823454510435
Chain Latitude Standard Deviation:  0.08124467619069661 Non-Chain Latitude Standard Deviation:  0.07493306469964077
Chain Longitude Standard Deviation:  0.2490614324043707 Non-Chain Longitude Standard Deviation:  0.22069228422245693
```

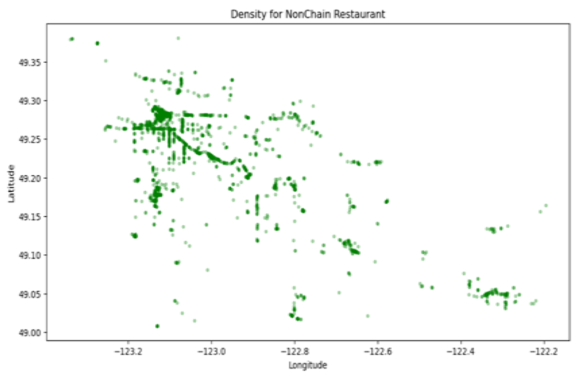
4) Find Outliers

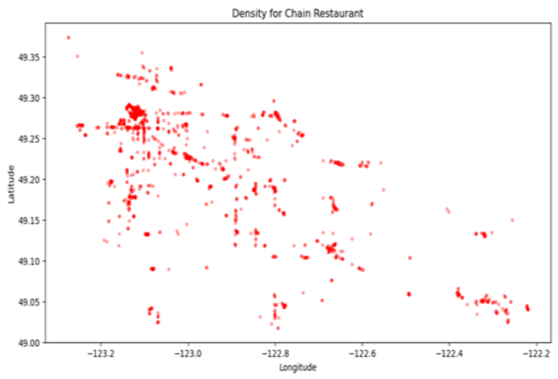


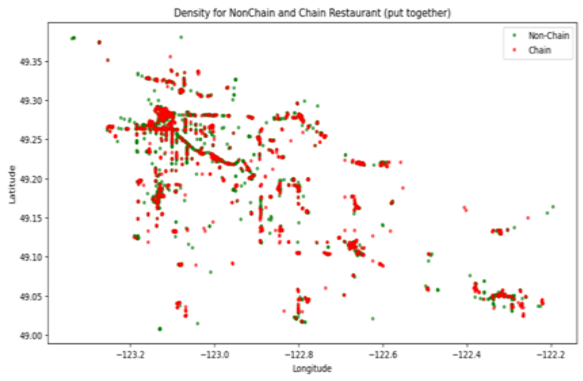
Except for chain restaurants' latitude, all the other data have some points that would be considered outliers.

But we shouldn't just ignore them. It makes sense to remove data if it's genuinely an error in measurement, or isn't fundamentally a true measurement of what you're looking at. But here those outliers are perfectly valid samples and should be included in the subsequent analysis so we shouldn’t remove them.

- ***Plot Restaurants on the map***



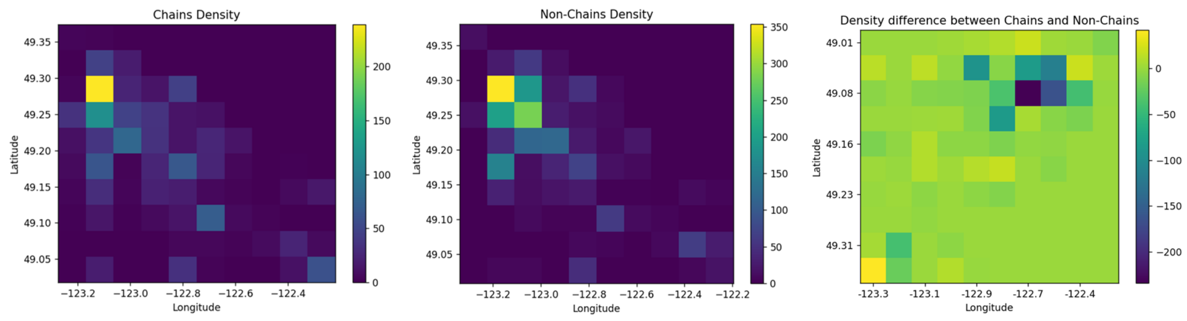




As we can see from the pictures above, chain and non-chain restaurants have a similar distribution in cities intuitively. The location of a restaurant doesn't seem to have anything to do with whether it's a chain or not.

- ***Visualize Density***

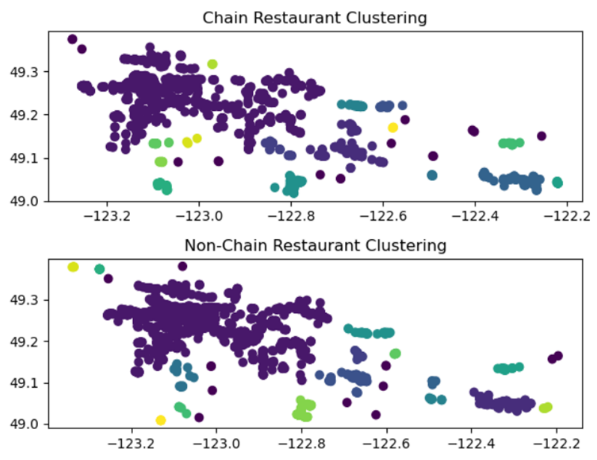
We plot three heat maps for the density of chain and non-chain restaurants and their density differences.



From the first two figures, we can see that chains and non-chains have similar densities. From the third figure, we can find the difference between chains and non-chains. In most places, they are not much different except in the northeast where the number of chain restaurants is much smaller than non-chain restaurants.

- ***DBSCAN Clustering***

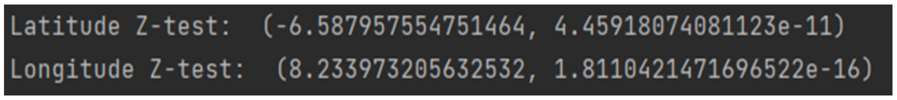
We use the "DBSCAN" method to cluster restaurants. "DBSCAN" is a density-based clustering algorithm. Unlike "K-means", we do not choose the number of clusters, so "DBSCAN" fits this problem better. When using "DBSCAN" we choose epsilon (the distance between points to consider) and min samples (the number of points to be considered a center). One difference from K-means is that not every point needs to be assigned to a cluster, some points can be considered noise.



Chain and non-chain restaurants have similar clustering results. They are concentrated in the same area.

- ***Z-Test***

Do Z-Test for the latitude and longitude of chain and non-chain restaurants.



```
Latitude Z-test:  (-6.587957554751464, 4.45918074081123e-11)
Longitude Z-test:  (8.233973205632532, 1.8110421471696522e-16)
```

The p-values are significantly less than 0.05. So the means of the latitude and longitude of chain and non-chain restaurants are not equal statistically.

- ***Summary***

After counting, the number of chain restaurants is smaller than that of non-chain restaurants in Vancouver. We used a scatter graph to map all the chain and non-chain restaurants in Vancouver and used the “DBSCAN” method to cluster restaurants. Through the results shown in the figures, we can see that they have a similar distribution. But with the Z-test, we find that chains and non-chains have different mean longitudes and latitudes.

In order to compare their densities visually, we plot some heat maps to show the densities of chains, and non-chains and the difference between them. They are not much different in most places except in northeast Vancouver where the chain restaurants are much less than non-chain restaurants.

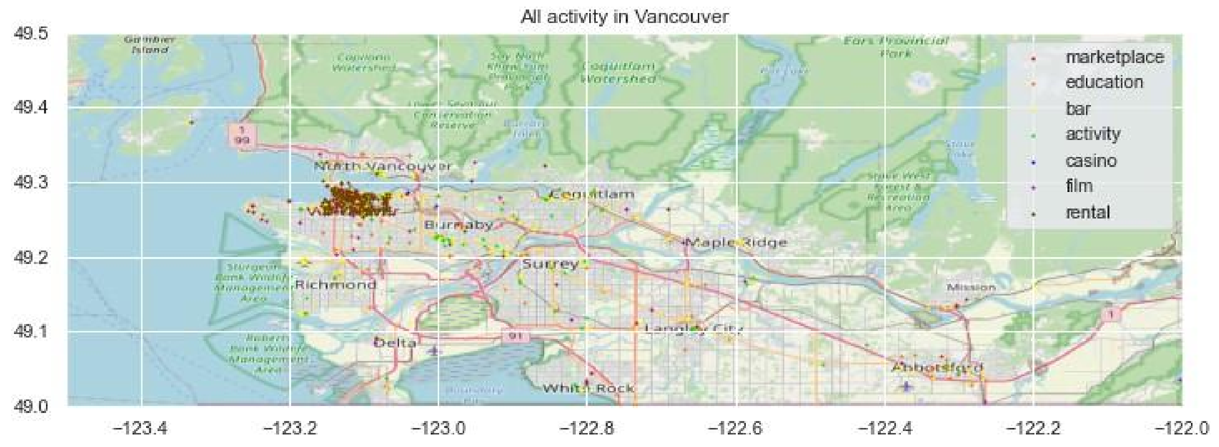
## IV. Tourists Attractions

- ***Introduction***

Besides all those famous attractions such as Stanley park, the science world, and the pacific national exhibition, where are those activities surrounding the communities which people have easy access, and how to access such information without much effort? Those are the problem that we will try our best effort to solve.

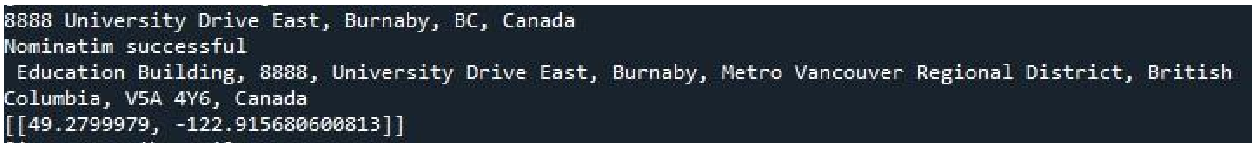
- ***Data (scenery_extract.py)***

There are two main focuses on the question. The first one is the overall information about the activities that exist in the region. The raw data we used for the information is ***amenities-vancouver.json***. The original version of the data contains many locations that are not suitable for being attractions including benches, post offices, waste disposal, and other places that are more focused on community utility purposes rather than entertainment purposes. Therefore, we extracted different types of locations that we considered and merge them into different categories including market, education, bar, casino, film, other activities, and vehicle rental for essential travel purposes. We formed a data frame for each category and assign them in different colors. Finally, we used a scatter plot to plot each location on the ***map.png*** with the same x and y-axis indicating the actual latitude and longitude. (***All activities in Vancouver.png***)

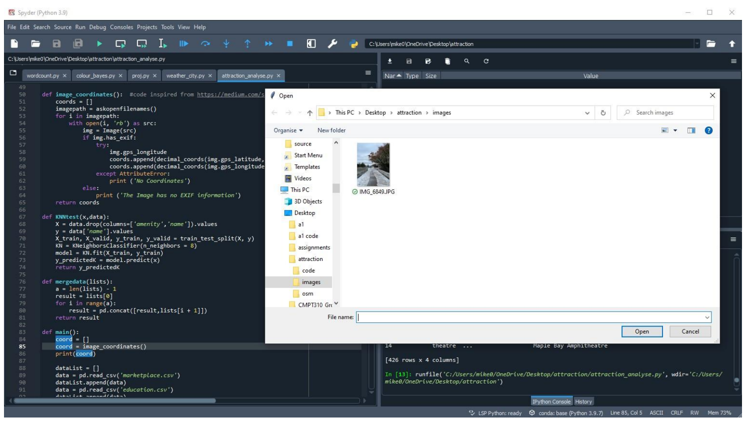


- ***Application (attraction_analyse.py)***

The second core issue for the task is the accessibility of the information. The basic idea of accessibility is to provide any activities near the location of the customer. The first attempt of getting the user's location is to use geopy to translate the street name, city, and county which the user inserted into geographic location values like latitude and longitude.[7]



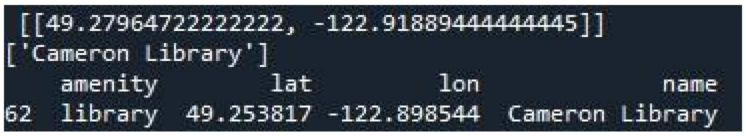
```
8888 University Drive East, Burnaby, BC, Canada
Nominatim successful
 Education Building, 8888, University Drive East, Burnaby, Metro Vancouver Regional District, British
Columbia, V5A 4Y6, Canada
[[49.2799979, -122.915680600813]]
```



```
def image_coordinates():  #code inspired from https://medium.com/s
    coords = []
    imagepath = askopenfilenames()
    for i in imagepath:
        with open(i, 'rb') as src:
            img = Image(src)
            if img.has_exif:
                try:
                    img.gps_longitude
                    coords.append(decimal_coords(img.gps_latitude,
                    coords.append(decimal_coords(img.gps_longitude
                except AttributeError:
                    print ('No Coordinates')
            else:
                print ('The Image has no EXIF information')
    return coords

def KNNtest(x,data):
    X = data.drop(columns=['amenity','name']).values
    y = data['name'].values
    X_train, X_valid, y_train, y_valid = train_test_split(X, y)
    KN = KNeighborsClassifier(n_neighbors = 8)
    model = KN.fit(X_train, y_train)
    y_predictedK = model.predict(x)
    return y_predictedK

def mergedata(lists):
    a = len(lists) - 1
    result = lists[0]
    for i in range(a):
        result = pd.concat([result,lists[i + 1]])
    return result

def main():
    coord = []
    coord = image_coordinates()
    print(coord)

    dataList = []
    data = pd.read_csv('marketplace.csv')
    dataList.append(data)
    data = pd.read_csv('education.csv')
```

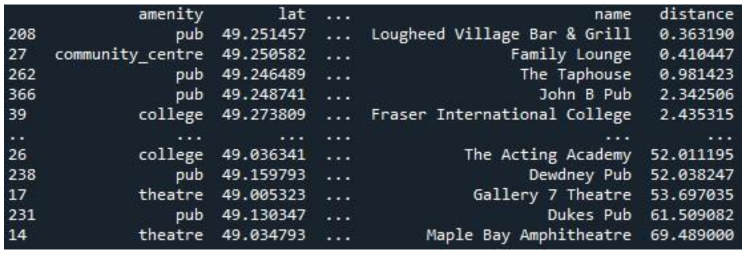
```
14          theatre  ...                Maple Bay Amphitheatre

[426 rows x 4 columns]

In [13]: runfile('C:/Users/mike0/OneDrive/Desktop/attraction/attraction_analyse.py', wdir='C:/Users/
mike0/OneDrive/Desktop/attraction')
```

Even though, the transformation of the street name to latitude and longitude is quite successful. We still think it is not convenient enough as the user might not be able to find his location with no street signs around. Therefore, We introduced a more convenient method to indicate the users' location by extracting the geographic information from the geotag of the picture uploaded by the users[8]. The users only need to take pictures with their phones as the location setting is on, then select the image from their file directories. If there is no geographic information in the pictures, the users can still insert their location in the location.txt file.



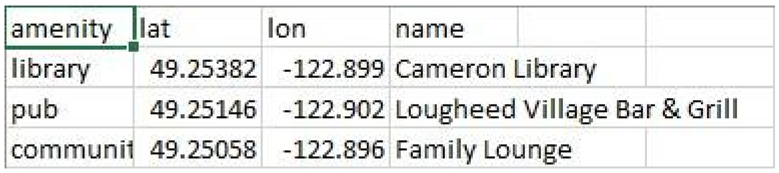
```
 [[49.27964722222222, -122.91889444444445]]
['Cameron Library']
    amenity        lat         lon            name
62  library  49.253817 -122.898544  Cameron Library
```

The next thing to solve after getting the users' location is to find the closest activity for them. We used KNeighborsClassifier (***KNNtest***(x,data)) to locate the closest neighbor of the inserted latitude and longitude value.



```
              amenity        lat  ...                       name   distance
208               pub  49.251457  ...  Lougheed Village Bar & Grill   0.363190
27   community_centre  49.250582  ...              Family Lounge   0.410447
262               pub  49.246489  ...               The Taphouse   0.981423
366               pub  49.248741  ...                 John B Pub   2.342506
39            college  49.273809  ...  Fraser International College   2.435315
..                ...        ...  ...                        ...        ...
26            college  49.036341  ...          The Acting Academy  52.011195
238               pub  49.159793  ...                Dewdney Pub  52.038247
17            theatre  49.005323  ...          Gallery 7 Theatre  53.697035
231               pub  49.130347  ...                  Dukes Pub  61.509082
14            theatre  49.034793  ...     Maple Bay Amphitheatre  69.489000
```

The next step of convenience is to plan the schedule for the travelers after we find the closest activity for them. We assumed that each traveler is able to go through three attractions at a max of a day with traveling between locations considered into the factors. The travelers also do not wish to go to the same activities on the same day. Therefore, we need to arrange three locations that are the closest to each other while remaining different types of activities. After we solve the first location by using KNeighborsClassifier, we then exclude that type of activity from the data frame. By using ***haversine_distances***[9], we are able to calculate the distance between every location from the new data frame to the first activity(***day_trip***(matched,allVA)). Then we select the next location with the smallest distance. We further apply the function on the newly formed data frame for the third location. Finally, we gather all three locations and formed a CSV file for the users. (***closest_route.csv***)



| amenity | lat | lon | name | |
|---|---|---|---|---|
| library | 49.25382 | -122.899 | Cameron Library | |
| pub | 49.25146 | -122.902 | Lougheed Village Bar & Grill | |
| communit | 49.25058 | -122.896 | Family Lounge | |

## V. Reflection

1. Running osm-transport_stop.py to extract all the public transport information is time-consuming. It might be better to export data from a smaller area from OSM.

2. The dataset Airbnb listings is not completed with lots of housing from Burnaby and Coquitlam missing. The model can get better results with a better dataset.

3. One of the disadvantage for using KNneighborclassifier is that it only finds the closest class but not necessarily the the closest point.

4. Our collaboration between teammates is not very efficient. We mostly focus on our own tasks without contributing much to each other’s work.

## VI. Source

1. https://www150.statcan.gc.ca/n1/daily-quotidien/200602/dq200602a-eng.htm
2. https://stackoverflow.com/questions/4971269/how-to-pick-a-new-color-for-each-plotted-line-within-a-figure-in-matplotlib
3. https://stackoverflow.com/questions/27928/calculate-distance-between-two-latitude-longitude-points-haversine-formula/21623206
4. https://stackoverflow.com/questions/32464280/converting-currency-with-to-numbers-in-python-pandas
5. https://geoffboeing.com/2014/08/clustering-to-reduce-spatial-data-set-size/
6. https://towardsdatascience.com/easy-steps-to-plot-geographic-data-on-a-map-python-11217859a2db
7. https://towardsdatascience.com/geocode-with-python-161ec1e62b89
8. https://medium.com/spatial-data-science/how-to-extract-gps-coordinates-from-images-in-python-e66e542af354
9. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.haversine_distances.html