

IBM Data Science Professional Certification on [Coursera](https://www.coursera.org/account/accomplishments/professional-cert/SVLJCF45AA4F) (<https://www.coursera.org/account/accomplishments/professional-cert/SVLJCF45AA4F>)

This is the work I did for the final Capstone project to complete the 9 course certification.

As the first significant Python project that I have completed from scratch, this makes a good showcase for some of my new skills.

Libraries: pandas, numpy, sci-kit learn, folium, matplotlib, geopy, BeautifulSoup

Competencies: web-scraping, mapping, data wrangling, K-means clustering, RESTful APIs

Project: Relocation

You can see the broad remit for the Capstone Project [here](https://www.coursera.org/learn/applied-data-science-capstone#syllabus) (<https://www.coursera.org/learn/applied-data-science-capstone#syllabus>). I chose to take things further than necessary as I came up with a problem to solve that I found interesting.

My wife and I have two big passions, these being travel and the outdoors. We have been lucky enough to have done a lot of backpacking and trekking in beautiful places all around the world. Now that we have an 18 month old toddler, we will better enjoy new places by living in them for a while. We will need to work and earn enough money to be able to enjoy life when not working. We want to feel safe and usually (not always!) the wealth of a country can reasonably indicate how safe it is. We want to go hiking in the mountains at the weekends so we need to be close to at least one mountain. This is the starting point.

Where should we go?

NB. This is a brief outline. All sources and the approach are detailed above the code cells further down the page.

1. Limit the list of possible countries to those with a minimum GDP per capita.

- In the GDP per capita list, Chile is the nation with the lowest figure in which I've spent time and where I could imagine myself living and working.
- If I am to consider a country, it must have a GDP per capita at least as high as Chile's.

2. Find cities in those countries that are within 100km of a mountain.

- As we will be working full time, we want the mountains within an hour or so for easy access.

3. Of those cities, which are highly rated for both ease of getting settled and quality of urban living?

4. Using the Foursquare API and a K-Means clustering algorithm to find out about the neighborhoods of the remaining cities.

Final Thoughts

- This project was constrained by time.
- Decisions were made at every stage that could be questioned.
- I am interested to know where this might go if I develop it further.
- Some questions that I would like to answer:
 - Is GDP per Capita the right metric? Should I use it in conjunction with others?
 - Do we really need to be within 100km of a mountain? A national park or forest could be good enough.
- Regarding the narrowing of the neighborhoods into those suitable for living, I've barely scratched the surface. E.g.:
 - How easy it is for foreigners to get work visas?
 - What is the availability and cost of rental properties?
 - What are the local crime rates?
 - Are the schools of good quality?

The Project

You should be able to rerun this yourself. Other than this notebook and an environment in which to run the code (I use Anaconda), you'll need to add some keys to the configuration file, 'jh.cfg':

1. You can get your own api key from <https://opencagedata.com> (<https://opencagedata.com>). Add the key to the 'OTHER' section of the file.
2. Foursquare (*UK readers - this is similar to TripAdvisor*). Sign up [here](https://foursquare.com/developers/signup) (<https://foursquare.com/developers/signup>) and add your ID and secret to the 'FOURSQUARE' section of the file.

Step 1 - import the packages, modules and functions.

```
In [7]: from bs4 import BeautifulSoup # for handling websites.
import numpy as np
import pandas as pd
import requests # for handling websites. use with BeautifulSoup
import csv # writing to a csv
import geopy.distance as gd # calculate distance between two sets of coordinates
import wikipedia as wp # alternative to BeautifulSoup for wikipedia
from geopy.geocoders import Nominatim # convert an address into Latitude and Longitude values
from opencage.geocoder import OpenCageGeocode # alternative source for Latitude and Longitude values
import folium # map rendering library
# import k-means from clustering stage
from sklearn.cluster import KMeans
# Matplotlib and associated plotting modules
import matplotlib.cm as cm
import matplotlib.colors as colors
from IPython.display import display, HTML
from opencage.geocoder import OpenCageGeocode
import configparser # to work with the configuration file

config = configparser.ConfigParser()
config.read('jh.cfg') # edit this file to include your own values.












key=config['OTHER']['GEOCODE_KEY']

client_id = config['FOURSQUARE']['CLIENT_ID'] # your Foursquare ID
client_secret = config['FOURSQUARE']['CLIENT_SECRET'] # your Foursquare Secret
```

Get the countries GDP per capita data from the source and store it in a CSV.

[https://en.wikipedia.org/wiki/List_of_countries_by_GDP_\(PPP\)_per_capita](https://en.wikipedia.org/wiki/List_of_countries_by_GDP_(PPP)_per_capita)
([https://en.wikipedia.org/wiki/List_of_countries_by_GDP_\(PPP\)_per_capita](https://en.wikipedia.org/wiki/List_of_countries_by_GDP_(PPP)_per_capita))

Top 10

Rank ↕	Country/Territory ↕	Int\$ ↕
1	 Qatar	138,910
—	 Macau	113,352
2	 Luxembourg	112,045
3	 Singapore	105,689
4	 Ireland	86,988
5	 Brunei	85,011
6	 Norway	79,638
7	 United Arab Emirates	70,441
8	 Kuwait	67,891
9	 Switzerland	67,558
10	 United States	67,426

```
In [ ]: #Main function
def getContent(link, filename='what.csv', whichtable=0):
    result1 = requests.get(link)
    src1 = result1.content
    soup = BeautifulSoup(src1, 'html.parser')
    table = soup.find_all('table')[whichtable]
    with open(filename, 'w', newline='') as f:
        writer = csv.writer(f)
        for tr in table('tr'):
            row = [t.get_text(strip=True) for t in tr(['td', 'th'])]
            writer.writerow(row)

getContent('https://en.wikipedia.org/wiki/List_of_countries_by_GDP_(PPP)_per_capita', 'gdp_ppp.csv', whichtable=2)
```

Create a dataframe from the CSV. Do some formatting and filtering.

Remove those countries without a dollar value. Find the figure for Chile and remove all countries where the 'Dollars' figure is lower.

```
In [ ]: colNames = ['Rank', 'Country', 'Dollars']
IMF = pd.read_csv('gdp_ppp.csv', encoding='ISO-8859-1')
IMF.columns = colNames
# remove those without a dollar value
IMF = IMF[IMF['Dollars'].notna()]
IMF["Dollars"] = IMF["Dollars"].str.replace(",", "").astype(float)
# get the figure for Chile
ch_dollars = IMF[IMF['Country']=='Chile']['Dollars'].to_numpy()
ch_dollars = ch_dollars[0]
# remove countries with dollar value lower than that of Chile
IMF = IMF[IMF['Dollars'].astype(float) >= ch_dollars]
IMF = IMF[['Country', 'Dollars']]

print('This is the top 5 countries by GDP Per Capita:')
display(HTML(IMF.head().to_html()))

print('We are left with', len(IMF), 'countries in total.')
```

Get the mountains data

Wikipedia is a great source for this data. There is a page containing easily scraped tables of data for every mountain in the world.

https://en.wikipedia.org/wiki/List_of_mountains_by_elevation (https://en.wikipedia.org/wiki/List_of_mountains_by_elevation)

This is the first table:

Mountain	Metres	Feet	Range	Location and Notes
Mount Everest	8,848	29,029	Himalayas	Nepal/China
K2	8,611	28,251	Karakoram	Pakistan
Kangchenjunga	8,586	28,169	Himalayas	Nepal/India
Lhotse	8,516	27,940	Himalayas	Nepal/China – Climbers ascend Lhotse Face in climbing Everest
Makalu	8,485	27,838	Himalayas	Nepal/China
Cho Oyu	8,201	26,906	Himalayas	Nepal/China – Considered "easiest" eight-thousander
Dhaulagiri	8,167	26,795	Himalayas	Nepal – Presumed world's highest from 1808-1838
Manaslu	8,163	26,781	Himalayas	Nepal
Nanga Parbat	8,126	26,660	Himalayas	Pakistan
Annapurna	8,091	26,545	Himalayas	Nepal – First eight-thousander to be climbed (1950)
Gasherbrum I (Hidden peak; K5)	8,080	26,509	Karakoram	Pakistan/China – Originally named K5
Broad Peak	8,051	26,414	Karakoram	Pakistan/China – Originally named K3
Gasherbrum II (K4)	8,035	26,362	Karakoram	Pakistan/China – Originally named K4
Shishapangma	8,027	26,335	Himalayas	China

The next code cell stores each of the tables from the webpage as a separate dataframe.

```
In [ ]: link = 'https://en.wikipedia.org/wiki/List_of_mountains_by_elevation'
#store the tables as DataFrames (this creates a list)
dataFrames = pd.read_html(link)
```

Now we need to pull all of the mountain dataframes into one dataframe.

```
In [ ]: df_mountains = dataFrames[0][['Mountain', 'Metres', 'Location and Notes']]
for i in range(1,9):
    df_mountains = pd.concat([
        df_mountains,
        pd.DataFrame(dataFrames[i][['Mountain', 'Metres', 'Location and Notes']])
    ], ignore_index=True, sort=False)

print('This is the 5 highest mountains:')
display(HTML(df_mountains.head().to_html()))
print('This is the 5 shortest mountains:')
display(HTML(df_mountains.tail().to_html()))
```

We can see that some of these so-called 'mountains' are of very low height. We'll deal with that later.

Which country is each mountain in?

1. We have this information in the 'Location and Notes' column of the 'df_mountains' dataframe and need to perform some string manipulation to extract it.
2. Once we've done this, we filter out all mountains in countries that do not meet the GDP criteria previously established.

```
In [ ]: pat = "|".join(IMF.Country)
df_mountains.insert(0, 'Country', df_mountains['Location and Notes'].str.extract("(" + pat + ")", expand=False))
df_mountains = df_mountains[df_mountains['Country'].notna()].reset_index(drop=True) #remove rows and reset index
df_mountains.head()
```

Pass the names of the mountains back to wikipedia to get their coordinates.

We return to https://en.wikipedia.org/wiki/List_of_mountains_by_elevation (https://en.wikipedia.org/wiki/List_of_mountains_by_elevation) and follow the link for each mountain to its own wikipedia page where we find the coordinates.

This is the relevant section of the page for Everest:

Mount Everest



Mount Everest as viewed from Kalapatthar.

Highest point

Elevation 8,848 m (29,029 ft) [\[1\]](#)
Ranked 1st

Prominence
Ranked 1st
(Notice special definition for Everest)

Listing Seven Summits
Eight-thousander
Country high point
Ultra

Coordinates [27°59′17″N 86°55′31″E](#) [\[2\]](#)


```

In [ ]: wp2 = 'https://en.wikipedia.org/wiki/List_of_mountains_by_elevation'
page = requests.get(wp2).text
html = BeautifulSoup(page)
tables = html.findAll('table', 'wikitable')
#create empty dataframe
mountain_details = pd.DataFrame(columns=['Name', 'Lat', 'Lon', 'LatLon'])

# go through the tables, adding them one-by-one into the new dataframe
for i in range(0,9):

    # Loop through the rows of the table to find the link for each mountain
    for lks in tables[i].findAll('a'):
        sub_link = lks.get('href')

        # alter the link to create one that we can work with
        link2 = str(sub_link).replace('/wiki/', 'https://en.wikipedia.org/wiki/')
        Name = sub_link.replace('/wiki/', '').replace('_', ' ')

        # ensure that the mountain is in the 'df_mountains' dataframe
        if Name.strip() in df_mountains['Mountain'].to_string():
            try:
                r = requests.get(link2)
                rText = r.text
                soup = BeautifulSoup(rText, 'html.parser')

                # in the HTML source code, the coordinates are called 'wgCoordinates'
                # we apply some string manipulation to extract the Latitude and Longitude values
                strString = rText[rText.find("wgCoordinates")+22:rText.find("wgCoordinates")+150]
                Lat = strString[0:strString.find("lon")-3]
                Lon = strString[strString.find("lon")+5:strString.find("wgWikibase")-3]
                try:
                    # check that we have a number
                    if isinstance(float(Lat), float):
                        mountain_details.loc[-1] = [Name, Lat, Lon, Lat+', '+Lon] # insert a row
                        mountain_details.index = mountain_details.index + 1 # shifting index
                        mountain_details = mountain_details.sort_index() # sorting by index
                except:
                    pass
            except:
                pass

```

```
mountain_details.head()
```

Next we will copy the coordinates into the 'df_mountains' dataframe and do some filtering.

Mountains without their own wikipedia page are excluded by design. Any mountain worthy of the name *surely* has its own wikipedia entry...

Also dropped are any mountains under 500 metres in height. I admit that this is somewhat arbitrary and you can have a good day out amongst smaller peaks but we have to start somewhere.

```
In [ ]: new_columns = df_mountains.columns.values
new_columns[1] = 'Name'
df_mountains.columns = new_columns # align column name with 'mountain_details'
# excluding mountains without their own wikipedia page. To include them, add this before the closing bracket: , how='left'
df_mountains = df_mountains.merge(mountain_details, on='Name')
df_mountains.drop(df_mountains.columns[[3,6]], axis=1, inplace=True)
df_mountains = df_mountains[df_mountains['Metres'] >= 500] # imposing minimum of 500m height

print('The top 5 remaining mountains with their coordinates:')
df_mountains.head()
```

To re-cap, so far we have:

1. Filtered the countries to leave only those with the minimum GDP per capita. These are in a dataframe called, 'IMF'.
2. Found all of the 500m+ mountains within those countries. These are in a dataframe called, 'df_mountains'.

The next step is to find the cities in those countries. The search is restricted to cities as that's where the vast majority of large companies and therefore jobs are concentrated. There are many possible sources for this data. The first page of the one I went with is [here](https://data.mongabay.com/cities_pop_01.htm) (https://data.mongabay.com/cities_pop_01.htm). The cities are displayed in descending order of population size. This is the top 10 as it appears on the webpage:

	WIKIPEDIA SOURCES						UNITED NATIONS SOURCES						
City/Country	World Rank	2008 estimate	Area extent (sq km)	Pop Growth Rate (%/yr)	Sources of Pop. / Area	Estimate includes / Notes	City population	Surface area (sq km)	Urban area population	Surface area (sq km)	Data Source	Date of Census	UN Notes
TOKYO, Japan	1	34,400,000	7,835	0.15	C / B	Yokohama [2]	8,489,653	621	12,576,601	2,187	CDFC	2005-10-01	(61),(62),(63),(65)
JAKARTA, Indonesia	2	21,800,000	2,720	2.38	F / B	[3]	8,640,184	740	ESDF	2003-07-01	
New York (NY), United States	3	20,090,000	11,264	0.24	H / H	[4]	8,143,197	783	ESDJ	2005-07-01	(16),(17)
SEOUL, South Korea	4	20,010,000	1,943	0.43	C / B	Incheon [5]	10,020,123	605	ESDF	2006-07-01	
MANILA, Philippines	5	19,550,000	1,425	2.31	C / B	[6]	1,581,082	614	CDJC	2000-10-01	
Mumbai (Bombay), India	6	19,530,000	777	2	C / B	[7]	11,914,398	466	16,368,084	1,041	CDFC	2001-03-01	(53)
Sao Paulo, Brazil	7	19,140,000	2,590	0.78	C / B	[8]	11,016,703	1,493	ESDF	2005-07-01	(11)
MEXICO CITY, Mexico	8	18,430,000	2,137	0.6	C / B	[9]	18,204,964	...	ESDJ	2006-07-01	(12),(13)
Delhi, India	9	18,000,000	1,425	2.4	C / B	[10]	9,817,439	431	12,791,458	624	CDFC	2001-03-01	(53),(54)
Osaka, Japan	10	17,270,000	2,720	0.04	C / B		2,628,811	222	CDFC	2005-10-01	(61),(62),(63)

```
In [ ]: links = ['https://data.mongabay.com/cities_pop_01.htm'
                , 'https://data.mongabay.com/cities_pop_02.htm'
                , 'https://data.mongabay.com/cities_pop_03.htm'
                ]

citiesFrames = {}
for i in range(0,3):
    r = requests.get(links[i])
    rContent = r.content
    rText = rContent.decode('utf8') #this ensures we pick up special characters correctly
    soup = BeautifulSoup(rText,"html.parser")
    table = soup.find("table",{"class":"boldtable"})
    citiesFrames[i] = pd.read_html(str(table))[0]
    citiesFrames[i].columns = citiesFrames[i].iloc[1]
```

We now have 3 dataframes stored in the 'citiesFrames' dictionary. It would be handy to have all 3 in the same place.

```
In [ ]: df_cities = citiesFrames[0][['CityCountry', 'Citypopulation']]
        for i in range(1,3):
            df_cities = pd.concat([
                df_cities,
                pd.DataFrame(citiesFrames[i][['CityCountry', 'Citypopulation']])
            ], ignore_index=True, sort=False)

        print('The first 5 cities:')
        display(HTML(df_cities.head().to_html()))
        print('We have', len(df_cities), 'cities in total.')
```

We can see some invalid data in the head of the 'df_cities' dataframe above. Let's get rid of it and check the data again.

```
In [ ]: cond1 = df_cities['CityCountry'].notna()
        cond2 = df_cities['CityCountry'] != 'CityCountry'
        df_cities = df_cities[cond1 & cond2]
        print('The first 5 cities:')
        display(HTML(df_cities.head().to_html()))
        print('We have', len(df_cities), 'cities in total.')
```

'df_cities' contains a column called 'CityCountry' which contains both the name of the city and the name of the country. The next task is to use this column to create separate 'City' and 'Country' columns. We can also do some cleaning.

```
In [ ]: df_cities = pd.concat([df_cities, df_cities['CityCountry'].str.split(' ', expand=True)], axis=1)
        new_cols = ['City and Country', 'Population', 'City', 'Country']
        df_cities.rename(columns=dict(zip(df_cities.columns[[0,1,2,3]], new_cols)), inplace=True) #rename columns
        df_cities = df_cities[['City', 'Country', 'City and Country', 'Population']]
        df_cities['City'] = df_cities['City'].str.replace(r"\(.*\)", "").str.strip().str.title() #remove text in brackets (e.g. (
        df_cities.reset_index(drop=True)
        print('The first 5 cities:')
        display(HTML(df_cities.head().to_html()))
```

As we did with the mountains data, we will now remove from 'df_cities' those cities that are not in our list of countries.

```
In [ ]: df_cities = df_cities.loc[((df_cities.Country.isin(IMF['Country']))),:].reset_index(drop=True)
print('The first 5 cities:')
display(HTML(df_cities.head().to_html()))
print('We can see that Jakarta, Seoul and Manila are no longer in the top 5. They have been replaced by Osaka, Los Angel
```

We now need to ascertain which of the 975 cities in 'df_cities' are within 100km of a mountain. We will use an API to get the coordinates of each city.

```
In [ ]: geocoder = OpenCageGeocode(key)
df_cities['Lat'] = '1' #add new columns to hold the coordinates
df_cities['Lon'] = '1'
df_cities['LatLon'] = '1'

for i in range(0,976):
    try:
        query = df_cities['City and Country'][i]
        results = geocoder.geocode(query)
        lat = results[0]['geometry']['lat']
        lon = results[0]['geometry']['lng']
        df_cities['Lat'][i]= lat
        df_cities['Lon'][i]= lon
        df_cities['LatLon'][i]= lat,lon
    except:
        pass
print('The top 5 cities with their coordinates:')
df_cities.head()
```

Calculate the distance between the cities and the mountains.

We will run the calculations only for cities and mountains in the same country.

```
In [ ]: df_city_mountain = df_cities.merge(df_mountains, on='Country')
df_city_mountain = df_city_mountain[['City', 'Country', 'Lat_x', 'Lon_x', 'Name', 'Metres', 'Lat_y', 'Lon_y']]
df_city_mountain.columns = ['City', 'Country', 'Lat_x', 'Lon_x', 'Mountain', 'Metres', 'Lat_y', 'Lon_y']
df_city_mountain['Distance'] = '1' #insert new column to hold the distance
df_city_mountain['Lon_y'] = df_city_mountain['Lon_y'].str.replace('\n', '') #There are some rows with '\n' in the Lon_y f
```

Find the distances, removing those city/mountain combinations where the distance is greater than 100km.

```
In [ ]: for i in range(0,len(df_city_mountain)+1):
        try:
            coords_1 = df_city_mountain['Lat_x'][i],df_city_mountain['Lon_x'][i]
            coords_2 = df_city_mountain['Lat_y'][i],df_city_mountain['Lon_y'][i]
            #print (coords_1,coords_2)
            df_city_mountain['Distance'][i] = gd.distance(coords_1, coords_2).km
        except:
            pass

df_city_mountains_final = df_city_mountain[df_city_mountain['Distance'].astype(int) <= 100]

print('The first 5 city/mountains:')
display(HTML(df_city_mountains_final.head().to_html()))
print('We have a total of',len(df_city_mountains_final),'city/mountains.')
```

Next we will filter for quality of urban living and ease of getting settled.

I will use the Internations survey from late 2019 as my source. Internations hosts expat communities around the world. The survey's respondents were expats in the cities concerned.

Links:

The write-up of the survey: <https://www.internations.org/expat-insider/2019/the-best-and-worst-cities-for-expats-39894>
(<https://www.internations.org/expat-insider/2019/the-best-and-worst-cities-for-expats-39894>)

The write-up of the 'Urban Living' section: <https://www.internations.org/expat-insider/2019/quality-of-urban-living-index-39895>
(<https://www.internations.org/expat-insider/2019/quality-of-urban-living-index-39895>)

The write-up of the 'Getting Settled' section: <https://www.internations.org/expat-insider/2019/getting-settled-index-39896>
(<https://www.internations.org/expat-insider/2019/getting-settled-index-39896>)

The pdf containing the ordered lists (Quality of Urban Living on page 10): https://cms-internationsgmbh.netdna-ssl.com/sites/default/files/2019-12/Expat-Insider-2019_Expats-City-Ranking_0.pdf (https://cms-internationsgmbh.netdna-ssl.com/sites/default/files/2019-12/Expat-Insider-2019_Expats-City-Ranking_0.pdf)

Next step:

I'll manually create a dataframe of the top 29 for each. I've changed the city names where appropriate to fit with those in the existing data (e.g. I have changed Zurich to Zürich).

```
In [ ]: df_urban_living = pd.DataFrame(['Zug'
                                         , 'Tokyo'
                                         , 'Taipei'
                                         , 'Zürich'
                                         , 'Vienna'
                                         , 'Basel'
                                         , 'Bern'
                                         , 'Madrid'
                                         , 'Singapore'
                                         , 'Lisbon'
                                         , 'Prague'
                                         , 'Barcelona'
                                         , 'München'
                                         , 'Lausanne'
                                         , 'Helsinki'
                                         , 'Genève'
                                         , 'Dusseldorf'
                                         , 'Hamburg'
                                         , 'Lugano'
                                         , 'Seoul'
                                         , 'Tallinn'
                                         , 'Vancouver'
                                         , 'Oslo'
                                         , 'Calgary'
                                         , 'Sydney'
                                         , 'Berlin'
                                         , 'The Hague'
                                         , 'Dubai'
                                         , 'Copenhagen'
                                         ], columns=['City'])

df_getting_settled = pd.DataFrame(['Kuala Lumpur'
                                     , 'Manama'
                                     , 'Nairobi'
                                     , 'Muscat'
                                     , 'Mexico City'
                                     , 'Lisbon'
                                     , 'Singapore'
                                     , 'Miami'
                                     , 'Jakarta'
                                     , 'Calgary'])
```



```

, 'Abu Dhabi'
, 'Madrid'
, 'Barcelona'
, 'Houston'
, 'Ho Chi Minh City'
, 'Sydney'
, 'Dubai'
, 'Taipei'
, 'Buenos Aires'
, 'Melbourne'
, 'Cape Town'
, 'Doha'
, 'Vancouver'
, 'Toronto'
, 'Luxembourg City'
, 'Athens'
, 'Dublin'
, 'Chicago'
, 'Montréal'
], columns=['City'])

df_city_mountains_final = pd.merge(pd.merge(df_city_mountains_final, df_getting_settled, how='inner'), df_urban_living, how='inner')

print('These are the remaining city/mountains after all filtering:')
pd.DataFrame(df_city_mountains_final.groupby(['Country', 'City'])['Mountain'].count()).sort_values(by=['City'])

```

Hmmm. I've been running this through prior to publishing on Github and this is the first time that Dubai has appeared in the output...

```
In [ ]: df_city_mountains_final[df_city_mountains_final['City']=='Dubai'][['City', 'Mountain']]
```

I've discovered that Jabel Yibir is not accessible to the public owing to military checkpoints. I'm going to exclude to stick with Calgary and Vancouver.

Let's get the postal codes for Vancouver from wikipedia. The data isn't as friendly as the other data pages we've looked at in terms of layout. The [page \(https://en.wikipedia.org/wiki/List_of_postal_codes_of_Canada:_V\)](https://en.wikipedia.org/wiki/List_of_postal_codes_of_Canada:_V) covers the whole province of British Columbia. Therefore, some filtering will be required to get the codes for Vancouver. This is what the page looks like:

List of postal codes of Canada: V

From Wikipedia, the free encyclopedia

This is a list of [postal codes in Canada](#) where the first letter is V. Postal codes beginning with V are located within the Canadian province of [British Columbia](#). Only the first three characters are listed, corresponding to the Forward Sortation Area.

[Canada Post](#) provides a free postal code look-up tool on its website,^[1] via its [mobile apps](#) for such [smartphones](#) as the [iPhone](#) and [BlackBerry](#),^[2] and sells hard-copy directories and [CD-ROMs](#). Many vendors also sell validation tools, which allow customers to properly match addresses and postal codes. Hard-copy directories can also be consulted in all post offices, and some libraries.

British Columbia - 192 FSAs [edit]

Urban [edit]

V1A Kimberley	V2A Penticton	V3A Langley Township (Langley City)	V4A Surrey Southwest	V5A Burnaby (Government Road / Lake City / SFU / Burnaby Mountain)	V6A Vancouver (Strathcona / Chinatown / Downtown Eastside)	V7A Richmond South	V8A Powell River	V9A Victoria (Vic West / Esquimalt) Canadian Forces (MARPAAC)
V1B Vernon East	V2B Kamloops Northwest	V3B Port Coquitlam Central	V4B White Rock	V5B Burnaby (Parkcrest-Aubrey / Ardingley-Sprott)	V6B Vancouver (NE Downtown / Gastown / Harbour Centre / International Village / Victory Square / Yaletown)	V7B Richmond (Sea Island / YVR)	V8B Squamish	V9B Victoria (West Highlands / North Langford / View Royal)
V1C Cranbrook	V2C Kamloops Central and Southeast	V3C Port Coquitlam South	V4C Delta Northeast	V5C Burnaby (Burnaby Heights / Willingdon Heights / West Central Valley)	V6C Vancouver (Waterfront / Coal Harbour / Canada Place)	V7C Richmond Northwest	V8C Kitimat	V9C Victoria (Colwood / South Langford / Metchosin)
V1E Salmon Arm	V2E Kamloops South and West	V3E Coquitlam North	V4E Delta East	V5E Burnaby (Lakeview-Mayfield / Richmond Park / Kingsway-Beresford)	V6E Vancouver (SE West End / Davie Village)	V7E Richmond Southwest	V8E Whistler	V9E Victoria (East Highlands / NW Saanich)
V1G Dawson Creek	V2G Williams Lake	V3G Abbotsford East	V4G Delta East Central	V5G Burnaby (Cascade-Schou / Douglas-Gilpin)	V6G Vancouver (NW West End / Stanley Park)	V7G North Vancouver (district municipality) Outer East	V8G Terrace	V9G Ladysmith

```
In [ ]: html = wp.page("List_of_postal_codes_of_Canada:_V").html().encode("UTF-8")

df_v = pd.read_html(html)[0]
df_v2 = df_v.stack().reset_index()
df_v2 = df_v2[[0]]
df_v2.columns = ['Text']
df_v2['Postal_Code'] = df_v2['Text'].str[:3].str.strip()
df_v2['Location'] = df_v2['Text'].str[3:200].str.strip()
# remove text in brackets, remove white space
df_v2['City'] = df_v2['Location'].str.replace(r"\(.*\)", "").str.strip()
# The webpage contains postal codes for all of British Columbia. We only want Vancouver.
df_v2 = df_v2[df_v2['City']=='Vancouver']
df_v2['Neighborhood'] = df_v2.Location.str.extract(r"(\(.*\))", expand=False).str.replace("(", "").str.replace(")", "").str
df_v2 = df_v2[['Postal_Code', 'City', 'Neighborhood']].sort_values(by=['Postal_Code']).reset_index(drop = True)
print('This is the first 5 rows of the Vancouver postal codes:')
df_v2.head()
```

Calgary is in the province of Alberta. The [wikipedia page for Alberta postal codes \(https://en.wikipedia.org/wiki/List_of_postal_codes_of_Canada:_T\)](https://en.wikipedia.org/wiki/List_of_postal_codes_of_Canada:_T) looks similar to the British Columbia page so we'll follow a similar process:

```
In [ ]: html = wp.page("List_of_postal_codes_of_Canada:_T").html().encode("UTF-8")

df_c = pd.read_html(html)[0]
df_c2 = df_c.stack().reset_index()
df_c2 = df_c2[[0]]
df_c2.columns = ['Text']
df_c2['Postal_Code'] = df_c2['Text'].str[:3].str.strip()
df_c2['Location'] = df_c2['Text'].str[3:200].str.strip()
df_c2['City'] = df_c2['Location'].str.replace(r"\(.*\)", "").str.strip() #remove text in brackets, remove white space
df_c2 = df_c2[df_c2['City']=='Calgary'] #The webpage contains postal codes for the province of Alberta. We only want tho
df_c2['Neighborhood'] = df_c2.Location.str.extract(r"(\(.*\))", expand=False).str.replace("(", "").str.replace(")", "").str
df_c2 = df_c2[['Postal_Code', 'City', 'Neighborhood']].sort_values(by=['Postal_Code']).reset_index(drop = True)
print('This is the first 5 rows of the Calgary postal codes:')
df_c2.head()
```

Let's merge the Vancouver and Calgary postal code data into one dataframe.

```
In [ ]: df_postal_codes = pd.concat([
        df_v2,
        df_c2
    ], ignore_index=True, sort=False)
df_postal_codes
```

Get the coordinates for each postal code. We will use these later to find nearby amenities and schools.

```

In [43]: geocoder = OpenCageGeocode(key)
df_postal_codes['Lat'] = '1' # add new columns to hold the coordinates
df_postal_codes['Lon'] = '1'
df_postal_codes['LatLon'] = '1'

for i in range(0,len(df_postal_codes)):
    try:
        query = df_postal_codes['Postal_Code'][i] + ', ' + df_postal_codes['City'][i] + ', Canada'
        results = geocoder.geocode(query)
        lat = results[0]['geometry']['lat']
        lon = results[0]['geometry']['lng']
        df_postal_codes['Lat'][i]= lat
        df_postal_codes['Lon'][i]= lon
        df_postal_codes['LatLon'][i]= lat,lon
    except:
        pass

print('We can see the coordinates with the postal code data:')
df_postal_codes

```

We can see the coordinates with the postal code data:

```

Out[43]:

```

	Unnamed: 0	Postal_Code	City	Neighborhood	Lat	Lon	LatLon
0	0	V5K	Vancouver	North Hastings-Sunrise	49.282336	-123.040000	(49.2823358, -123.0399998)
1	1	V5L	Vancouver	North Grandview-Woodland	49.278839	-123.066843	(49.2788393, -123.0668426)
2	2	V5M	Vancouver	South Hastings-Sunrise , North Renfrew-Colling...	49.258053	-123.040160	(49.2580527, -123.0401597)
3	3	V5N	Vancouver	South Grandview-Woodland , NE Kensington-Cedar...	49.253451	-123.066314	(49.253451, -123.0663142)
4	4	V5P	Vancouver	SE Kensington-Cedar Cottage , Victoria-Fraserview	49.222370	-123.068315	(49.22237, -123.0683154)
...
57	57	T3J	Calgary	Martindale , Taradale , Falconridge , Saddle R...	51.113492	-113.945300	(51.113492, -113.9452998)
58	58	T3K	Calgary	Sandstone , MacEwan Glen , Beddington , Harves...	51.156300	-114.057200	(51.1563, -114.0572)
59	59	T3L	Calgary	Tuscany , Scenic Acres	51.125665	-114.262136	(51.1256654, -114.2621363)
60	60	T3M	Calgary	Cranston , Auburn Bay , Mahogany	50.870031	-113.980977	(50.8700306, -113.9809766)
61	61	T3P	Calgary	Symons Valley	51.207400	-114.134800	(51.2074, -114.1348)

62 rows × 7 columns

It's time to draw some city maps.

```

In [3]: addresses = ['Calgary, Canada', 'Vancouver, Canada']
cities = ['Calgary', 'Vancouver']
maps = {0: 'MapCalgary',
        1: 'MapVancouver'} #if we create a dictionary for the maps then we can add to them easily in the 'for' loop.

for i in range(0, len(addresses)):

    geolocator = Nominatim() #creating maps using Nominatim in deference to Coursera suggestions.
    location = geolocator.geocode(addresses[i])
    latitude = location.latitude
    longitude = location.longitude
    print(f'The geograpical coordinates of {addresses[i]} are {latitude}, {longitude}.')
    maps[i] = folium.Map(location=[latitude, longitude], zoom_start=9)

    # add neighborhood markers to map
    for lat, lng, neighborhood in zip(df_postal_codes[df_postal_codes['City']==cities[i]]['Lat'],
                                      df_postal_codes[df_postal_codes['City']==cities[i]]['Lon'],
                                      df_postal_codes[df_postal_codes['City']==cities[i]]['Neighborhood']):

        label = '{}'.format(neighborhood)
        label = folium.Popup(label, parse_html=True)
        folium.CircleMarker(
            [lat, lng],
            radius=5,
            popup=label,
            color='blue',
            fill=True,
            fill_color='#3186cc',
            fill_opacity=0.7,
            parse_html=False).add_to(maps[i])

    for lat, lng, mountain in zip(df_city_mountains_final[df_city_mountains_final['City']==cities[i]]['Lat_y'].astype(float),
                                  df_city_mountains_final[df_city_mountains_final['City']==cities[i]]['Lon_y'].astype(float),
                                  df_city_mountains_final[df_city_mountains_final['City']==cities[i]]['Mountain']):

        label = f'{mountain}'
        label = folium.Popup(label, parse_html=True)
        folium.Marker(
            [lat, lng],
            popup=label,
            icon=folium.Icon(color="green", icon="fa-tree", prefix='fa')
        ).add_to(maps[i])

```

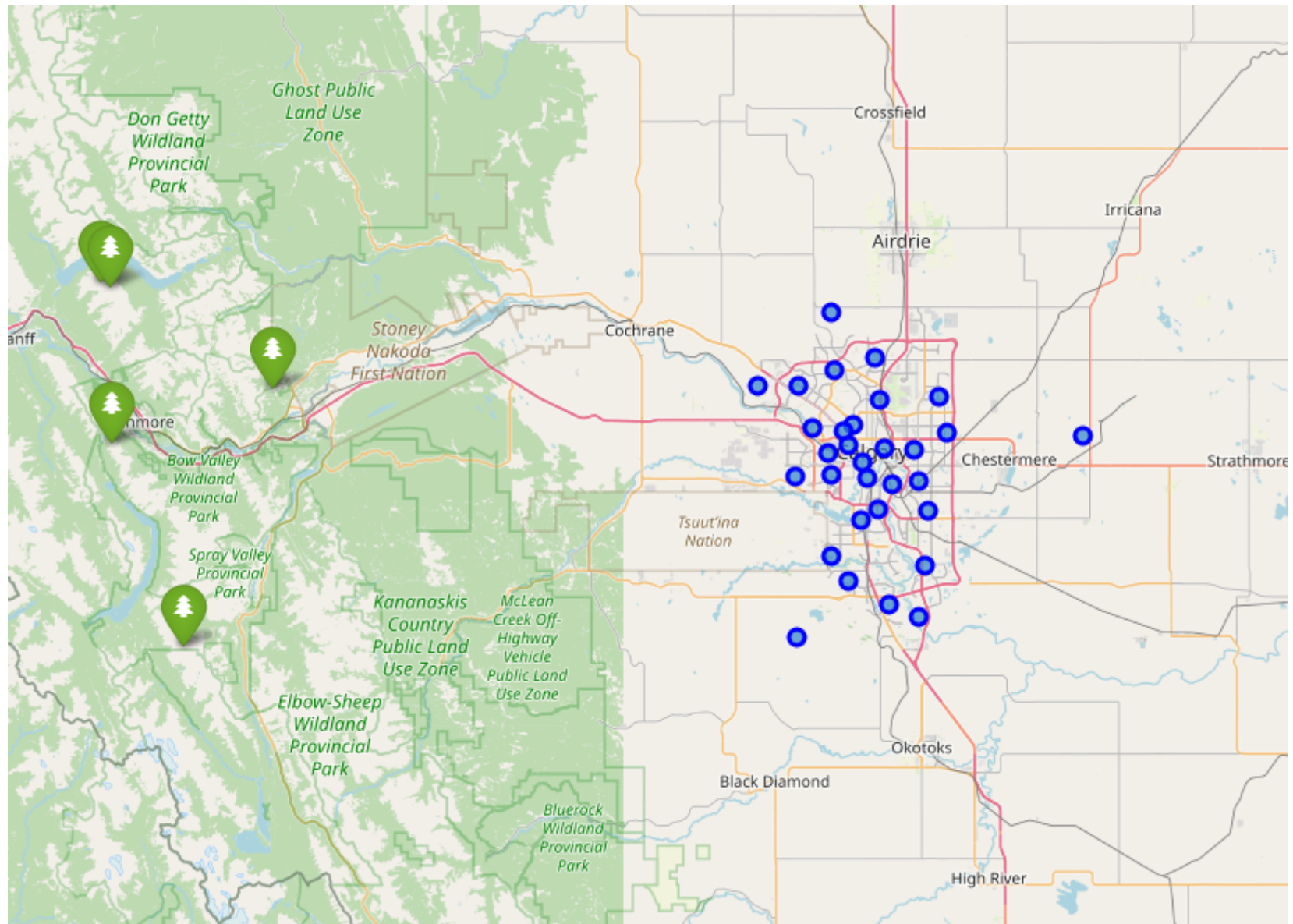
```
C:\Users\johnh\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: DeprecationWarning: Using Nominatim with the default "geopy/1.21.0" `user_agent` is strongly discouraged, as it violates Nominatim's ToS https://operations.osmfoundation.org/policies/nominatim/ (https://operations.osmfoundation.org/policies/nominatim/) and may possibly cause 403 and 429 HTTP errors. Please specify a custom `user_agent` with `Nominatim(user_agent="my-application")` or by overriding the default `user_agent`: `geopy.geocoders.options.default_user_agent = "my-application"`. In geopy 2.0 this will become an exception.
```

The geographical coordinates of Calgary, Canada are 51.0534234, -114.0625892.
The geographical coordinates of Vancouver, Canada are 49.2608724, -123.1139529.

The map of Calgary

The mountains are marked by trees while the neighborhoods are blue.

I've included images of all maps above each code cell as I realise you may well not be running this yourself and maps are not retained in the file.

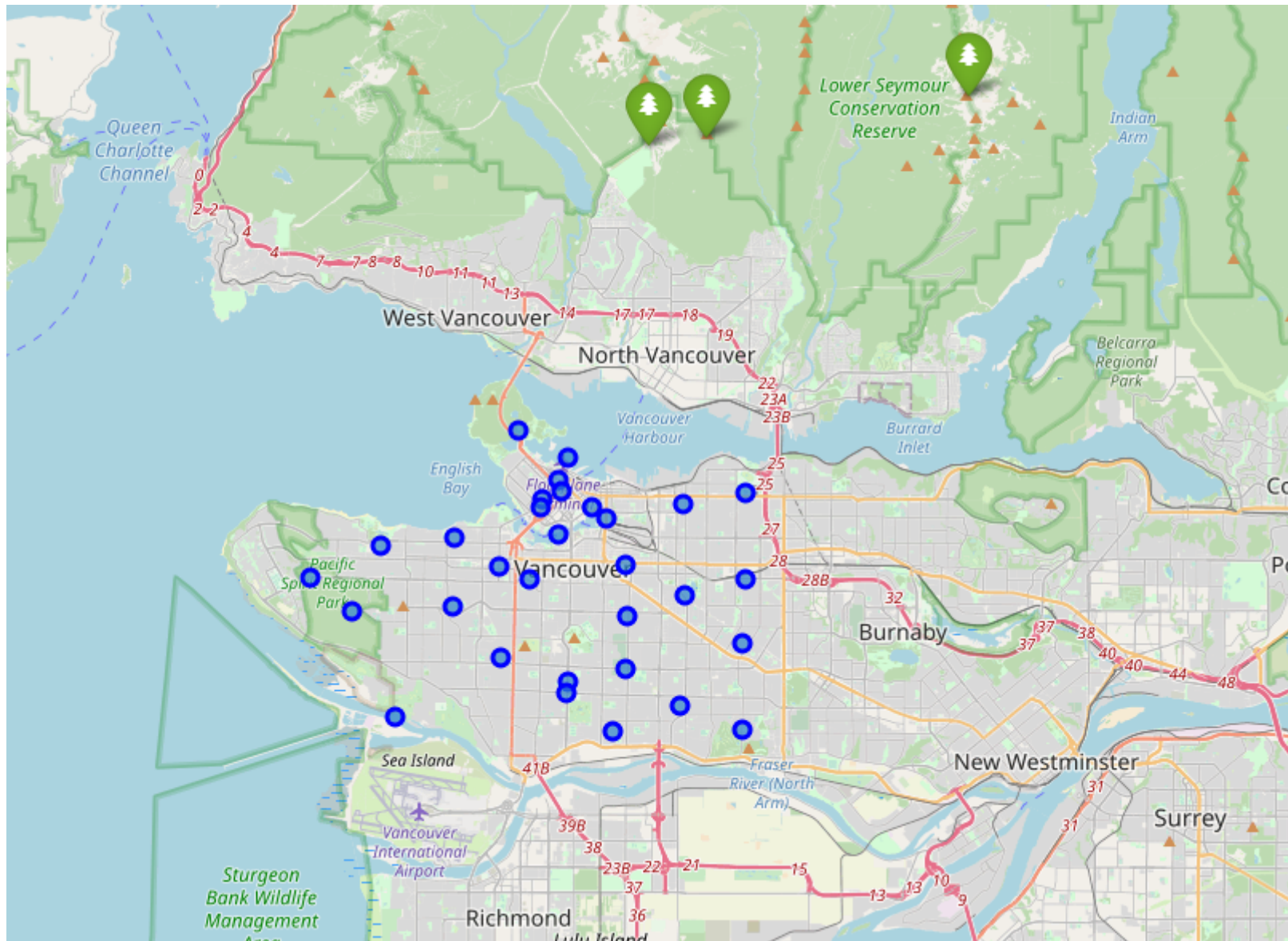


If you are running the code then you can click the icons in the map you generate to see the names. The maps are moveable and zoomable.

In [4]: `maps[0]`

Out[4]:

Vancouver:



```
In [5]: maps[1]
```

```
Out[5]:
```

Use the Foursquare API to look at what amenities are available in each neighborhood.

First, we build a function:

```

In [8]: fs_version = '20180605' # Foursquare API version

def getNearbyVenues(cities, names, latitudes, longitudes, radius=500, limit = 100):

    venues_list=[]
    for city, name, lat, lng in zip(cities, names, latitudes, longitudes):

        # create the API request URL
        url = 'https://api.foursquare.com/v2/venues/explore?&client_id={}&client_secret={}&v={}&ll={},{}&radius={}&limit={}&offset={}'
        client_id,
        client_secret,
        fs_version,
        lat,
        lng,
        radius,
        limit)

        # make the GET request
        results = requests.get(url).json()["response"]["groups"][0]["items"]

        # return only relevant information for each nearby venue
        venues_list.append([(
            city, #added city in as we have more than one.
            name,
            lat,
            lng,
            v['venue']['name'],
            v['venue']['location']['lat'],
            v['venue']['location']['lng'],
            v['venue']['categories'][0]['name']) for v in results])

    nearby_venues = pd.DataFrame([item for venue_list in venues_list for item in venue_list])
    nearby_venues.columns = ['City', #added city in as we have more than one.
                            'Neighborhood',
                            'Neighborhood Latitude',
                            'Neighborhood Longitude',
                            'Venue',
                            'Venue Latitude',
                            'Venue Longitude',
                            'Venue Category']

```

```
return(nearby_venues)
```

Call the function to obtain results.

We call the function only once, passing all the relevant data from the postal codes dataframe together. The function then iterates through the separate values, returning all results to one dataframe.

```
In [12]: df_neighborhoodVenues = getNearbyVenues(cities=df_postal_codes['City'],
                                                names=df_postal_codes['Neighborhood'],
                                                latitudes=df_postal_codes['Lat'],
                                                longitudes=df_postal_codes['Lon']
                                                )

print('This is what the neighborhood/venue dataframe looks like:')
df_neighborhoodVenues.head()
```

This is what the neighborhood/venue dataframe looks like:

```
Out[12]:
```

	Unnamed: 0	City	Neighborhood	Neighborhood Latitude	Neighborhood Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category	Category
0	0	Vancouver	North Hastings-Sunrise	49.282336	-123.040000	The Fair at the PNE	49.282971	-123.042109	Fair	Entertainment
1	1	Vancouver	North Hastings-Sunrise	49.282336	-123.040000	Livestock Barns	49.284037	-123.039278	Farm	Other
2	2	Calgary	Hawkwood , Arbour Lake , Citadel , Ranchlands ...	51.157034	-114.231181	Butterfield Acres	51.158593	-114.235227	Farm	Other
3	3	Vancouver	North Hastings-Sunrise	49.282336	-123.040000	Wooden Roller Coaster	49.281744	-123.035128	Theme Park Ride / Attraction	Entertainment
4	4	Vancouver	North Hastings-Sunrise	49.282336	-123.040000	Crazy Beach Party	49.282196	-123.036135	Theme Park Ride / Attraction	Entertainment

Take a closer look at Vancouver:

```
In [41]: df_neighborhoodVenues[df_neighborhoodVenues['City']=='Vancouver'][['Neighborhood','Venue']].groupby('Neighborhood').count
```

```
Out[41]:
```

	Venue
Neighborhood	
Bentall Centre	46
Central Kitsilano , Greektown	29
East Fairview , South Cambie	27
NE Downtown , Gastown , Harbour Centre , International Village , Victory Square , Yaletown	89
NW Shaughnessy , East Kitsilano , Quilchena	33
North Grandview-Woodland	34
North Hastings-Sunrise	25
Pacific Centre	98
SE Kensington-Cedar Cottage , Victoria-Fraserview	21
SE Riley Park-Little Mountain , SW Kensington-Cedar Cottage , NE Oakridge , North Sunset	21
SE West End , Davie Village	99
SW Downtown	65
Strathcona , Chinatown , Downtown Eastside	32
UBC	43
Waterfront , Coal Harbour , Canada Place	26
West Fairview , Granville Island , NE Shaughnessy	25
West Kensington-Cedar Cottage , NE Riley Park-Little Mountain	25
West Mount Pleasant , West Riley Park-Little Mountain	55

We can see that there are some neighborhoods with very few venues. It's going to be tough to infer much about them so we will remove them. I'll only include neighbourhoods with at least 20 venues in further analysis.

Calgary:


```
In [57]: df_neighborhoodVenues[df_neighborhoodVenues['City']=='Vancouver'][['City', 'Neighborhood', 'Venue']].groupby(['City', 'Neig
```

Out[57]:

		Venue
City	Neighborhood	
Vancouver	Bentall Centre	46
	Central Kitsilano , Greektown	29
	East Fairview , South Cambie	27
	East Mount Pleasant	18
	Killarney	17
	NE Downtown , Gastown , Harbour Centre , International Village , Victory Square , Yaletown	89
	NW Arbutus Ridge , NE Dunbar-Southlands	4
	NW Dunbar-Southlands , Chaldecutt , South University Endowment Lands	2
	NW Shaughnessy , East Kitsilano , Quilchena	33
	NW West End , Stanley Park	19
	North Grandview-Woodland	34
	North Hastings-Sunrise	25
	Pacific Centre	98
	SE Kensington-Cedar Cottage , Victoria-Fraserview	21
	SE Kerrisdale , SW Oakridge , West Marpole	4
	SE Oakridge , East Marpole , South Sunset	3
	SE Riley Park-Little Mountain , SW Kensington-Cedar Cottage , NE Oakridge , North Sunset	21
	SE West End , Davie Village	99
	SW Downtown	65
	South Grandview-Woodland , NE Kensington-Cedar Cottage	14
	South Hastings-Sunrise , North Renfrew-Collingwood	15
	South Renfrew-Collingwood	6
	South Shaughnessy , NW Oakridge , NE Kerrisdale , SE Arbutus Ridge	6

City	Venue	
	Neighborhood	
	Strathcona , Chinatown , Downtown Eastside	32
	UBC	43
	Waterfront , Coal Harbour , Canada Place	26
	West Fairview , Granville Island , NE Shaughnessy	25
	West Kensington-Cedar Cottage , NE Riley Park-Little Mountain	25
	West Kerrisdale , South Dunbar-Southlands , Musqueam	1
	West Kitsilano , West Point Grey , Jericho	2
	West Mount Pleasant , West Riley Park-Little Mountain	55

```
In [53]: df_neighborhoodVenues[df_neighborhoodVenues['City']=='Calgary'][['City', 'Neighborhood', 'Venue']].groupby(['City', 'Neighborhood']).count().reset_index()
```

Out[53]:

City	Neighborhood	Venue
Calgary	Braeside , Cedarbrae , Woodbine	3
	Brentwood , Collingwood , Nose Hill	7
	Bridgeland , Greenview , Zoo , YYC	3
	City Centre , Calgary Tower	1
	Connaught , West Victoria Park	81
	Cranston , Auburn Bay , Mahogany	2
	Dalhousie , Edgemont , Hamptons , Hidden Valley	5
	Discovery Ridge , Signal Hill , West Springs , Christie Estates , Patterson , Cougar Ridge	1
	Douglas Glen , McKenzie Lake , Copperfield , East Shepard	31
	Elbow Park , Britannia , Parkhill , Mission	1
	Forest Lawn , Dover , Erin Woods	1
	Hawkwood , Arbour Lake , Citadel , Ranchlands , Royal Oak , Rocky Ridge	2
	Highfield , Burns Industrial	7
	Inglewood , Burnsland , Chinatown , East Victoria Park , Saddledome	3
	Kensington , Westmont , Parkdale , University	5
	Lakeview , Glendale , Killarney , Glamorgan	6
	Lynnwood Ridge , Ogden , Foothills Industrial , Great Plains	4
	Martindale , Taradale , Falconridge , Saddle Ridge	5
	Midnapore , Sundance	5
	Montgomery , Bowness , Silver Springs , Greenwood	3
	Mount Pleasant , Capitol Hill , Banff Trail	4
	Oak Ridge , Haysboro , Kingsland , Kelvin Grove , Windsor Park	5
	Penbrooke Meadows , Marlborough	11

City	Neighborhood	Venue
	Rosscarrock , Westgate , Wildwood , Shaganappi , Sunalta	2
	Rundle , Whitehorn , Monterey Park	5
	Sandstone , MacEwan Glen , Beddington , Harvest Hills , Coventry Hills , Panorama Hills	22
	Symons Valley	1
	Thornccliffe , Tuxedo Park	8
	Tuscany , Scenic Acres	1

There are far fewer venues in most Calgary neighborhoods. Here, we'll put the venue minimum at 5.

In the next cell, I apply these filters:

```
In [138]: v = df_neighborhoodVenues['Neighborhood'][df_neighborhoodVenues['City']=='Vancouver'][df_neighborhoodVenues.groupby(['Neig
c = df_neighborhoodVenues['Neighborhood'][df_neighborhoodVenues['City']=='Calgary'][df_neighborhoodVenues.groupby(['Neig
vc = v.append(c, ignore_index=True)
df_neighborhoodVenues2 = pd.merge(df_neighborhoodVenues, vc, on='Neighborhood', how='inner')
print('The remaining neighborhoods:')
df_neighborhoodVenues2[['City', 'Neighborhood']].drop_duplicates().sort_values(by=['City', 'Neighborhood']).reset_index(dr
```

The remaining neighborhoods:

Out[138]:

	City	Neighborhood
0	Calgary	Brentwood , Collingwood , Nose Hill
1	Calgary	Connaught , West Victoria Park
2	Calgary	Dalhousie , Edgemont , Hamptons , Hidden Valley
3	Calgary	Douglas Glen , McKenzie Lake , Copperfield , E...
4	Calgary	Highfield , Burns Industrial
5	Calgary	Kensington , Westmont , Parkdale , University
6	Calgary	Lakeview , Glendale , Killarney , Glamorgan
7	Calgary	Martindale , Taradale , Falconridge , Saddle R...
8	Calgary	Midnapore , Sundance
9	Calgary	Oak Ridge , Haysboro , Kingsland , Kelvin Grov...
10	Calgary	Penbrooke Meadows , Marlborough
11	Calgary	Rundle , Whitehorn , Monterey Park
12	Calgary	Sandstone , MacEwan Glen , Beddington , Harves...
13	Calgary	Thorncliffe , Tuxedo Park
14	Vancouver	Bentall Centre
15	Vancouver	Central Kitsilano , Greektown
16	Vancouver	East Fairview , South Cambie
17	Vancouver	NE Downtown , Gastown , Harbour Centre , Inter...
18	Vancouver	NW Shaughnessy , East Kitsilano , Quilchena

	City	Neighborhood
19	Vancouver	North Grandview-Woodland
20	Vancouver	North Hastings-Sunrise
21	Vancouver	Pacific Centre
22	Vancouver	SE Kensington-Cedar Cottage , Victoria-Fraserview
23	Vancouver	SE Riley Park-Little Mountain , SW Kensington-...
24	Vancouver	SE West End , Davie Village
25	Vancouver	SW Downtown
26	Vancouver	Strathcona , Chinatown , Downtown Eastside
27	Vancouver	UBC
28	Vancouver	Waterfront , Coal Harbour , Canada Place
29	Vancouver	West Fairview , Granville Island , NE Shaughnessy
30	Vancouver	West Kensington-Cedar Cottage , NE Riley Park-...
31	Vancouver	West Mount Pleasant , West Riley Park-Little M...

Look at the venue types across remaining neighbourhoods.

We use 'one hot' encoding to move each venue type to its own column. This will make clustering much more simple later on.

```
In [143]: # one hot encoding
neighborhood_onehot = pd.get_dummies(df_neighborhoodVenues2[['Venue Category']], prefix="", prefix_sep="")

# add city and neighborhood columns back to dataframe
neighborhood_onehot['City'] = df_neighborhoodVenues2['City']

# move City column to the first column
fixed_columns = [neighborhood_onehot.columns[-1]] + list(neighborhood_onehot.columns[:-1])
neighborhood_onehot = neighborhood_onehot[fixed_columns]
neighborhood_onehot['Neighborhood'] = df_neighborhoodVenues2['Neighborhood']

# move neighborhood column to the second column
neigh = neighborhood_onehot['Neighborhood']
neighborhood_onehot.drop(labels=['Neighborhood'], axis=1, inplace = True)
neighborhood_onehot.insert(1, 'Neighborhood', neigh)

print('A sample of the re-organised neighborhood data for Vancouver:')
display(HTML(neighborhood_onehot.head().to_html()))
```

A sample of the re-organised neighborhood data for Vancouver:

Let's look at the same for Calgary:

```
In [145]: neighborhood_onehot.tail()
```

Out[145]:

	City	Neighborhood	Accessories Store	Airport Service	Airport Terminal	American Restaurant	Amphitheater	Art Gallery	Arts & Crafts Store	Asian Restaurant	...	Thrift / Vintage Store	Toy / Game Store	Trade School	Trai
991	Calgary	Martindale , Taradale , Falconridge , Saddle R...	0	0	0	0	0	0	0	0	...	0	0	0	(
992	Calgary	Martindale , Taradale , Falconridge , Saddle R...	0	0	0	0	0	0	0	0	...	0	0	0	(
993	Calgary	Martindale , Taradale , Falconridge , Saddle R...	0	0	0	0	0	0	0	0	...	0	0	0	(
994	Calgary	Martindale , Taradale , Falconridge , Saddle R...	0	0	0	0	0	0	0	0	...	0	0	0	(
995	Calgary	Martindale , Taradale , Falconridge , Saddle R...	0	0	0	0	0	0	0	0	...	0	0	0	(

5 rows × 200 columns



We can see from the output above that there are 200 columns. Most of these are different venue categories. This is too many for clustering. We can group them into a smaller number of categories such as 'Restaurants' and 'Nightlife' as below.


```
In [146]: df_venue_categories = {'Venue Category':[
'Accessories Store',
'Airport Service',
'Airport Terminal',
'American Restaurant',
'Amphitheater',
'Aquarium',
'Art Gallery',
'Arts & Crafts Store',
'Asian Restaurant',
'Athletics & Sports',
'Bagel Shop',
'Bakery',
'Bank',
'Bar',
'BBQ Joint',
'Beach',
'Beer Bar',
'Beer Garden',
'Belgian Restaurant',
'Bike Rental / Bike Share',
'Boat or Ferry',
'Bookstore',
'Boutique',
'Bowling Alley',
'Breakfast Spot',
'Brewery',
'Bubble Tea Shop',
'Building',
'Burger Joint',
'Burrito Place',
'Bus Station',
'Bus Stop',
'Business Service',
'CafÃ©',
'Cafeteria',
'Cajun / Creole Restaurant',
'Camera Store',
'Cantonese Restaurant',
'Caribbean Restaurant',
'Casino',
```

```
'Child Care Service',  
'Chinese Restaurant',  
'Chocolate Shop',  
'Church',  
'Circus',  
'Clothing Store',  
'Cocktail Bar',  
'Coffee Shop',  
'Concert Hall',  
'Construction & Landscaping',  
'Convenience Store',  
'Cosmetics Shop',  
'Cruise',  
'Dance Studio',  
'Deli / Bodega',  
'Department Store',  
'Dessert Shop',  
'Dim Sum Restaurant',  
'Diner',  
'Disc Golf',  
'Discount Store',  
'Dive Bar',  
'Dog Run',  
'Donut Shop',  
'Electronics Store',  
'Ethiopian Restaurant',  
'Event Space',  
'Exhibit',  
'Fair',  
'Falafel Restaurant',  
'Farm',  
'Farmers Market',  
'Fast Food Restaurant',  
'Field',  
'Filipino Restaurant',  
'Financial or Legal Service',  
'Fish & Chips Shop',  
'Food & Drink Shop',  
'Food Court',  
'Food Truck',  
'French Restaurant',  
'Fried Chicken Joint',
```

```
'Frozen Yogurt Shop',  
'Furniture / Home Store',  
'Garden',  
'Gas Station',  
'Gastropub',  
'Gay Bar',  
'General Entertainment',  
'German Restaurant',  
'Gift Shop',  
'Golf Course',  
'Gourmet Shop',  
'Greek Restaurant',  
'Grocery Store',  
'Gun Shop',  
'Gym',  
'Gym / Fitness Center',  
'Gym Pool',  
'Hawaiian Restaurant',  
'Health Food Store',  
'High School',  
'Himalayan Restaurant',  
'Historic Site',  
'History Museum',  
'Hobby Shop',  
'Hockey Arena',  
'Hockey Field',  
'Home Service',  
'Hookah Bar',  
'Hostel',  
'Hot Dog Joint',  
'Hotel',  
'Hotel Bar',  
'Hotpot Restaurant',  
'Ice Cream Shop',  
'Indian Restaurant',  
'Indie Movie Theater',  
'Indonesian Restaurant',  
'Inn',  
'Insurance Office',  
'Irish Pub',  
'IT Services',  
'Italian Restaurant',
```

```
'Japanese Curry Restaurant',  
'Japanese Restaurant',  
'Jewelry Store',  
'Juice Bar',  
'Karaoke Bar',  
'Kids Store',  
'Kitchen Supply Store',  
'Korean Restaurant',  
'Lake',  
'Laundromat',  
'Leather Goods Store',  
'Lebanese Restaurant',  
'Light Rail Station',  
'Lingerie Store',  
'Liquor Store',  
'Locksmith',  
'Lounge',  
'Malay Restaurant',  
'Market',  
'Massage Studio',  
'Mattress Store',  
'Medical Center',  
'Men's Store',  
'Mexican Restaurant',  
'Middle Eastern Restaurant',  
'Miscellaneous Shop',  
'Mobile Phone Shop',  
'Moroccan Restaurant',  
'Motorcycle Shop',  
'Movie Theater',  
'Museum',  
'Music Store',  
'Music Venue',  
'Neighborhood',  
'New American Restaurant',  
'Nightclub',  
'Noodle House',  
'Office',  
'Optical Shop',  
'Other Great Outdoors',  
'Outdoor Sculpture',  
'Paper / Office Supplies Store',
```

```
'Park',  
'Performing Arts Venue',  
'Pet Store',  
'Pharmacy',  
'Photography Studio',  
'Physical Therapist',  
'Pie Shop',  
'Pizza Place',  
'Playground',  
'Plaza',  
'Poke Place',  
'Pool',  
'Portuguese Restaurant',  
'Poutine Place',  
'Pub',  
'Ramen Restaurant',  
'Rental Car Location',  
'Restaurant',  
'Salad Place',  
'Salon / Barbershop',  
'Sandwich Place',  
'Scandinavian Restaurant',  
'Scenic Lookout',  
'Science Museum',  
'Seafood Restaurant',  
'Shoe Store',  
'Shopping Mall',  
'Skating Rink',  
'Snack Place',  
'Soccer Field',  
'Soccer Stadium',  
'Spa',  
'Speakeasy',  
'Sporting Goods Shop',  
'Sports Bar',  
'Stadium',  
'Steakhouse',  
'Supermarket',  
'Sushi Restaurant',  
'Taco Place',  
'Tanning Salon',  
'Tapas Restaurant',
```

```
'Tea Room',
'Tex-Mex Restaurant',
'Thai Restaurant',
'Theater',
'Theme Park',
'Theme Park Ride / Attraction',
'Thrift / Vintage Store',
'Toy / Game Store',
'Trade School',
'Trail',
'Vegetarian / Vegan Restaurant',
'Video Game Store',
'Vietnamese Restaurant',
'Warehouse Store',
'Water Park',
'Wine Shop',
"Women's Store",
'Yoga Studio',
'Zoo Exhibit'], 'Category':[
'Retail',
'Services',
'Travel / Transport',
'Restaurant',
'Entertainment',
'Entertainment',
'Entertainment',
'Retail',
'Restaurant',
'Retail',
'Restaurant',
'Retail',
'Services',
'Nightlife',
'Restaurant',
'Outdoors',
'Nightlife',
'Nightlife',
'Restaurant',
'Outdoors',
'Travel / Transport',
'Retail',
'Retail',
```

```
'Entertainment',  
'Cafes and Coffee Shops',  
'Nightlife',  
'Cafes and Coffee Shops',  
'Retail',  
'Restaurant',  
'Restaurant',  
'Travel / Transport',  
'Travel / Transport',  
'Services',  
'Cafes and Coffee Shops',  
'Cafes and Coffee Shops',  
'Restaurant',  
'Retail',  
'Restaurant',  
'Restaurant',  
'Nightlife',  
'Services',  
'Restaurant',  
'Retail',  
'Other',  
'Entertainment',  
'Retail',  
'Nightlife',  
'Cafes and Coffee Shops',  
'Entertainment',  
'Services',  
'Retail',  
'Retail',  
'Travel / Transport',  
'Entertainment',  
'Cafes and Coffee Shops',  
'Retail',  
'Cafes and Coffee Shops',  
'Restaurant',  
'Restaurant',  
'Entertainment',  
'Retail',  
'Nightlife',  
'Services',  
'Cafes and Coffee Shops',  
'Retail',
```

```
'Restaurant',  
'Services',  
'Entertainment',  
'Entertainment',  
'Restaurant',  
'Other',  
'Retail',  
'Restaurant',  
'Outdoors',  
'Restaurant',  
'Services',  
'Restaurant',  
'Retail',  
'Restaurant',  
'Restaurant',  
'Restaurant',  
'Restaurant',  
'Cafes and Coffee Shops',  
'Retail',  
'Outdoors',  
'Services',  
'Nightlife',  
'Nightlife',  
'Entertainment',  
'Restaurant',  
'Retail',  
'Sport',  
'Retail',  
'Restaurant',  
'Retail',  
'Retail',  
'Sport',  
'Sport',  
'Sport',  
'Restaurant',  
'Retail',  
'Restaurant',  
'Restaurant',  
'Restaurant',  
'Restaurant',  
'Retail',  
'Sport',
```



```
'Sport',  
'Services',  
'Nightlife',  
'Hotel',  
'Restaurant',  
'Hotel',  
'Nightlife',  
'Restaurant',  
'Cafes and Coffee Shops',  
'Restaurant',  
'Entertainment',  
'Restaurant',  
'Nightlife',  
'Services',  
'Nightlife',  
'Services',  
'Restaurant',  
'Restaurant',  
'Restaurant',  
'Retail',  
'Cafes and Coffee Shops',  
'Nightlife',  
'Retail',  
'Retail',  
'Restaurant',  
'Outdoors',  
'Services',  
'Retail',  
'Restaurant',  
'Travel / Transport',  
'Retail',  
'Retail',  
'Retail',  
'Nightlife',  
'Restaurant',  
'Retail',  
'Sport',  
'Retail',  
'Services',  
'Retail',  
'Restaurant',  
'Restaurant',
```

```
'Retail',  
'Retail',  
'Restaurant',  
'Retail',  
'Entertainment',  
'Entertainment',  
'Retail',  
'Entertainment',  
'Other',  
'Restaurant',  
'Nightlife',  
'Restaurant',  
'Services',  
'Retail',  
'Retail',  
'Entertainment',  
'Retail',  
'Outdoors',  
'Entertainment',  
'Retail',  
'Retail',  
'Retail',  
'Services',  
'Retail',  
'Restaurant',  
'Outdoors',  
'Outdoors',  
'Nightlife',  
'Sport',  
'Restaurant',  
'Restaurant',  
'Nightlife',  
'Restaurant',  
'Travel / Transport',  
'Restaurant',  
'Restaurant',  
'Retail',  
'Restaurant',  
'Restaurant',  
'Outdoors',  
'Entertainment',  
'Restaurant',
```

```
'Retail',  
'Retail',  
'Sport',  
'Cafes and Coffee Shops',  
'Sport',  
'Sport',  
'Entertainment',  
'Nightlife',  
'Retail',  
'Restaurant',  
'Sport',  
'Restaurant',  
'Retail',  
'Restaurant',  
'Restaurant',  
'Retail',  
'Restaurant',  
'Cafes and Coffee Shops',  
'Restaurant',  
'Restaurant',  
'Entertainment',  
'Entertainment',  
'Entertainment',  
'Retail',  
'Retail',  
'Services',  
'Outdoors',  
'Restaurant',  
'Retail',  
'Restaurant',  
'Retail',  
'Entertainment',  
'Retail',  
'Retail',  
'Sport',  
'Entertainment']}]}
```

```
In [148]: df_venue_categories = pd.DataFrame(df_venue_categories)
print('These are the new venue categories:')
df_venue_categories['Category'].drop_duplicates().reset_index(drop=True)
```

These are the new venue categories:

```
Out[148]: 0          Retail
1          Services
2    Travel / Transport
3          Restaurant
4          Entertainment
5          Nightlife
6          Outdoors
7    Cafes and Coffee Shops
8              Other
9              Sport
10             Hotel
Name: Category, dtype: object
```

Next, we need to merge the new venue categories dataframe with the neighborhood venues dataframe.

```
In [149]: df_neighborhoodVenues3 = df_neighborhoodVenues2.merge(df_venue_categories)
print('This is the first 5 rows of the merged dataset:')
df_neighborhoodVenues3.head()
```

This is the first 5 rows of the merged dataset:

```
Out[149]:
```

	Unnamed: 0	City	Neighborhood	Neighborhood Latitude	Neighborhood Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category	Category
0	0	Vancouver	North Hastings-Sunrise	49.282336	-123.04	The Fair at the PNE	49.282971	-123.042109	Fair	Entertainment
1	1	Vancouver	North Hastings-Sunrise	49.282336	-123.04	Livestock Barns	49.284037	-123.039278	Farm	Other
2	3	Vancouver	North Hastings-Sunrise	49.282336	-123.04	Wooden Roller Coaster	49.281744	-123.035128	Theme Park Ride / Attraction	Entertainment
3	4	Vancouver	North Hastings-Sunrise	49.282336	-123.04	Crazy Beach Party	49.282196	-123.036135	Theme Park Ride / Attraction	Entertainment
4	7	Vancouver	North Hastings-Sunrise	49.282336	-123.04	Playland	49.281924	-123.036258	Theme Park	Entertainment

Now we will re-run the one hot encoding on the new categories.

```
In [150]: # one hot encoding
neighborhood_onehot = pd.get_dummies(df_neighborhoodVenues[['Category']], prefix="", prefix_sep="")

# add city and neighborhood columns back to dataframe
neighborhood_onehot['City'] = df_neighborhoodVenues['City']
# move City column to the first column
fixed_columns = [neighborhood_onehot.columns[-1]] + list(neighborhood_onehot.columns[:-1])
neighborhood_onehot = neighborhood_onehot[fixed_columns]
neighborhood_onehot['Neighborhood'] = df_neighborhoodVenues['Neighborhood']
# move neighborhood column to the second column
neigh = neighborhood_onehot['Neighborhood']
neighborhood_onehot.drop(labels=['Neighborhood'], axis=1, inplace = True)
neighborhood_onehot.insert(1, 'Neighborhood', neigh)
print('This looks much more sensible:')
neighborhood_onehot.head()
```

This looks much more sensible:

Out[150]:

	City	Neighborhood	Cafes and Coffee Shops	Entertainment	Hotel	Nightlife	Other	Outdoors	Restaurant	Retail	Services	Sport	Travel / Transport
0	Vancouver	North Hastings-Sunrise	0	1	0	0	0	0	0	0	0	0	0
1	Vancouver	North Hastings-Sunrise	0	0	0	0	1	0	0	0	0	0	0
2	Calgary	Hawkwood , Arbour Lake , Citadel , Ranchlands ...	0	0	0	0	1	0	0	0	0	0	0
3	Vancouver	North Hastings-Sunrise	0	1	0	0	0	0	0	0	0	0	0
4	Vancouver	North Hastings-Sunrise	0	1	0	0	0	0	0	0	0	0	0

Now, let's group rows by neighborhood and by taking the mean of the frequency of occurrence of each category.

The sum for each row (neighborhood) is 1. Each value represents the proportion of all of the venues in that neighborhood made up by that type of venue. For example, hotels in Connaught / West Victoria Park make up 0.024691 or about 2.5% of all venues there.

```
In [160]: neighborhood_grouped = neighborhood_onehot.groupby(['City', 'Neighborhood']).mean().reset_index()
neighborhood_grouped[neighborhood_grouped['City']=='Calgary'].head()
```

Out[160]:

	City	Neighborhood	Cafes and Coffee Shops	Entertainment	Hotel	Nightlife	Other	Outdoors	Restaurant	Retail	Services	Sport	Travel / Transport
0	Calgary	Braeside , Cedarbrae , Woodbine	0.000000	0.333333	0.000000	0.000000	0.0	0.000000	0.333333	0.000000	0.333333	0.000000	0.0
1	Calgary	Brentwood , Collingwood , Nose Hill	0.142857	0.000000	0.000000	0.142857	0.0	0.000000	0.428571	0.285714	0.000000	0.000000	0.0
2	Calgary	Bridgeland , Greenview , Zoo , YYC	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	1.000000	0.000000	0.000000	0.000000	0.0
3	Calgary	City Centre , Calgary Tower	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	1.000000	0.000000	0.0
4	Calgary	Connaught , West Victoria Park	0.123457	0.000000	0.024691	0.160494	0.0	0.024691	0.432099	0.197531	0.012346	0.024691	0.0

Get the 10 most common venue types for each neighborhood and store the data in a dataframe:

```

In [162]: # create a function to get the most common venues
def return_most_common_venues(row, num_top_venues):
    row_categories = row.iloc[2:]
    row_categories_sorted = row_categories.sort_values(ascending=False)

    return row_categories_sorted.index.values[0:num_top_venues]

num_top_venues = 10

indicators = ['st', 'nd', 'rd'] # 1st, 2nd, 3rd

# create columns according to number of top venues
columns = ['City', 'Neighborhood']
for ind in np.arange(num_top_venues):
    try:
        columns.append(f'{ind+1}{indicators[ind]} Most Common Venue')
    except: # 4th, 5th etc
        columns.append(f'{ind+1}th Most Common Venue')

# create a new dataframe
neighborhoods_venues_sorted = pd.DataFrame(columns=columns)
neighborhoods_venues_sorted['City'] = neighborhood_grouped['City']
neighborhoods_venues_sorted['Neighborhood'] = neighborhood_grouped['Neighborhood']

for ind in np.arange(neighborhood_grouped.shape[0]): # can't use Len(neighborhood_grouped) as 'int' object is not iterable
    neighborhoods_venues_sorted.iloc[ind, 2:] = return_most_common_venues(neighborhood_grouped.iloc[ind, :], num_top_venues)

nvs = {0: 'Calgary',
        1: 'Vancouver'}
nvs[0] = neighborhoods_venues_sorted[neighborhoods_venues_sorted['City']=='Calgary']

print('View data for Calgary:')
display(HTML(nvs[0].to_html()))

```

View data for Calgary:

City	Neighborhood	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7th Most Common Venue	8th Most Common Venue	9th Most Common Venue	10th Most Common Venue
<hr/>											

	City	Neighborhood	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7th Most Common Venue	8th Most Common Venue	9th Most Common Venue	10th Most Common Venue
0	Calgary	Braeside , Cedarbrae , Woodbine	Services	Restaurant	Entertainment	Travel / Transport	Sport	Retail	Outdoors	Other	Nightlife	Hotel
1	Calgary	Brentwood , Collingwood , Nose Hill	Restaurant	Retail	Nightlife	Cafes and Coffee Shops	Travel / Transport	Sport	Services	Outdoors	Other	Hotel
2	Calgary	Bridgeland , Greenview , Zoo , YYC	Restaurant	Travel / Transport	Sport	Services	Retail	Outdoors	Other	Nightlife	Hotel	Entertainment
3	Calgary	City Centre , Calgary Tower	Services	Travel / Transport	Sport	Retail	Restaurant	Outdoors	Other	Nightlife	Hotel	Entertainment
4	Calgary	Connaught , West Victoria Park	Restaurant	Retail	Nightlife	Cafes and Coffee Shops	Sport	Outdoors	Hotel	Services	Travel / Transport	Other
5	Calgary	Cranston , Auburn Bay , Mahogany	Services	Travel / Transport	Sport	Retail	Restaurant	Outdoors	Other	Nightlife	Hotel	Entertainment
6	Calgary	Dalhousie , Edgemont , Hamptons , Hidden Valley	Retail	Services	Travel / Transport	Sport	Restaurant	Outdoors	Other	Nightlife	Hotel	Entertainment
7	Calgary	Discovery Ridge , Signal Hill , West Springs ,Christie Estates , Patterson , Cougar Ridge	Entertainment	Travel / Transport	Sport	Services	Retail	Restaurant	Outdoors	Other	Nightlife	Hotel
8	Calgary	Douglas Glen , McKenzie Lake , Copperfield , East Shepard	Restaurant	Retail	Cafes and Coffee Shops	Nightlife	Travel / Transport	Sport	Services	Outdoors	Other	Hotel
9	Calgary	Elbow Park , Britannia , Parkhill , Mission	Restaurant	Travel / Transport	Sport	Services	Retail	Outdoors	Other	Nightlife	Hotel	Entertainment

	City	Neighborhood	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7th Most Common Venue	8th Most Common Venue	9th Most Common Venue	10th Most Common Venue
10	Calgary	Forest Lawn , Dover , Erin Woods	Services	Travel / Transport	Sport	Retail	Restaurant	Outdoors	Other	Nightlife	Hotel	Entertainment
11	Calgary	Hawkwood , Arbour Lake , Citadel , Ranchlands , Royal Oak , Rocky Ridge	Sport	Other	Travel / Transport	Services	Retail	Restaurant	Outdoors	Nightlife	Hotel	Entertainment
12	Calgary	Highfield , Burns Industrial	Retail	Restaurant	Travel / Transport	Sport	Services	Outdoors	Other	Nightlife	Hotel	Entertainment
13	Calgary	Inglewood , Burnsland , Chinatown , East Victoria Park , Saddledome	Services	Restaurant	Travel / Transport	Sport	Retail	Outdoors	Other	Nightlife	Hotel	Entertainment
14	Calgary	Kensington , Westmont , Parkdale , University	Sport	Outdoors	Travel / Transport	Services	Retail	Restaurant	Other	Nightlife	Hotel	Entertainment
15	Calgary	Lakeview , Glendale , Killarney , Glamorgan	Retail	Restaurant	Travel / Transport	Sport	Services	Outdoors	Other	Nightlife	Hotel	Entertainment
16	Calgary	Lynnwood Ridge , Ogden , Foothills Industrial , Great Plains	Restaurant	Cafes and Coffee Shops	Travel / Transport	Sport	Services	Retail	Outdoors	Other	Nightlife	Hotel
17	Calgary	Martindale , Taradale , Falconridge , Saddle Ridge	Retail	Services	Outdoors	Travel / Transport	Sport	Restaurant	Other	Nightlife	Hotel	Entertainment
18	Calgary	Midnapore , Sundance	Sport	Services	Retail	Outdoors	Travel / Transport	Restaurant	Other	Nightlife	Hotel	Entertainment

	City	Neighborhood	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7th Most Common Venue	8th Most Common Venue	9th Most Common Venue	10th Most Common Venue
19	Calgary	Montgomery , Bowness , Silver Springs , Greenwood	Sport	Restaurant	Outdoors	Travel / Transport	Services	Retail	Other	Nightlife	Hotel	Entertainment
20	Calgary	Mount Pleasant , Capitol Hill , Banff Trail	Restaurant	Outdoors	Cafes and Coffee Shops	Travel / Transport	Sport	Services	Retail	Other	Nightlife	Hotel
21	Calgary	Oak Ridge , Haysboro , Kingsland , Kelvin Grove , Windsor Park	Restaurant	Sport	Retail	Cafes and Coffee Shops	Travel / Transport	Services	Outdoors	Other	Nightlife	Hotel
22	Calgary	Penbrooke Meadows , Marlborough	Restaurant	Retail	Cafes and Coffee Shops	Entertainment	Travel / Transport	Sport	Services	Outdoors	Other	Nightlife
23	Calgary	Rosscarrock , Westgate , Wildwood , Shaganappi , Sunalta	Nightlife	Cafes and Coffee Shops	Travel / Transport	Sport	Services	Retail	Restaurant	Outdoors	Other	Hotel
24	Calgary	Rundle , Whitehorn , Monterey Park	Restaurant	Services	Retail	Travel / Transport	Sport	Outdoors	Other	Nightlife	Hotel	Entertainment
25	Calgary	Sandstone , MacEwan Glen , Beddington , Harvest Hills , Coventry Hills , Panorama Hills	Retail	Restaurant	Cafes and Coffee Shops	Services	Outdoors	Nightlife	Travel / Transport	Sport	Other	Hotel
26	Calgary	Symons Valley	Services	Travel / Transport	Sport	Retail	Restaurant	Outdoors	Other	Nightlife	Hotel	Entertainment
27	Calgary	Thornccliffe , Tuxedo Park	Restaurant	Retail	Nightlife	Entertainment	Travel / Transport	Sport	Services	Outdoors	Other	Hotel
28	Calgary	Tuscany , Scenic Acres	Retail	Travel / Transport	Sport	Services	Restaurant	Outdoors	Other	Nightlife	Hotel	Entertainment

Same for Vancouver:

```
In [165]: nvs[1] = neighborhoods_venues_sorted[neighborhoods_venues_sorted['City']=='Vancouver']
display(HTML(nvs[1].reset_index(drop=True).to_html()))
```

	City	Neighborhood	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7th Most Common Venue	8th Most Common Venue	9th Most Common Venue
0	Vancouver	Bentall Centre	Retail	Cafes and Coffee Shops	Restaurant	Hotel	Outdoors	Nightlife	Sport	Entertainment	Travel / Transport
1	Vancouver	Central Kitsilano , Greektown	Restaurant	Retail	Cafes and Coffee Shops	Services	Outdoors	Nightlife	Travel / Transport	Sport	Other
2	Vancouver	East Fairview , South Cambie	Restaurant	Cafes and Coffee Shops	Retail	Outdoors	Travel / Transport	Services	Sport	Other	Nightlife
3	Vancouver	East Mount Pleasant	Restaurant	Retail	Nightlife	Entertainment	Outdoors	Travel / Transport	Sport	Services	Other
4	Vancouver	Killarney	Restaurant	Retail	Cafes and Coffee Shops	Services	Travel / Transport	Sport	Outdoors	Other	Nightlife

The final step of the project is to build and run a K-Means Clustering algorithm. The aim of this is to see which neighborhoods are similar to each other - those that are similar will be placed in the same cluster. We can then examine the types of venues in each cluster to get some insight on the kind of area we are looking at and whether or not we might be interested in living there.

```
In [166]: neighborhood_grouped_clustering = {}

neighborhood_grouped_clustering[0]= neighborhood_grouped[neighborhood_grouped['City'] == 'Calgary'].drop(['City', 'Neighborhood'])
neighborhood_grouped_clustering[1]= neighborhood_grouped[neighborhood_grouped['City'] == 'Vancouver'].drop(['City', 'Neighborhood'])
```

```
In [167]: # set number of clusters
kclusters = 8

kmeans = {0: 'Calgary'
          ,1: 'Vancouver'} #dictionary to store models in

for i in kmeans:

    # run k-means clustering
    kmeans[i] = KMeans(n_clusters=kclusters, random_state=0).fit(neighborhood_grouped_clustering[i])

    # check cluster labels generated for each row in the dataframe
    kmeans[i].labels_[0:10]

    # add clustering labels to nvs dataframes
    nvs[i].insert(0, 'Cluster Labels', kmeans[i].labels_)

#merge KMeans results
neighborhoods_venues_sorted = pd.concat([
                                         nvs[0],
                                         nvs[1]
                                         ], ignore_index=True, sort =False)
```

Let's create a new dataframe that includes the cluster as well as the top 10 venues for each neighborhood.

```
In [168]: neighborhood_merged = df_postal_codes[['City', 'Neighborhood', 'Lat', 'Lon']]
del neighborhood_merged['City'] # we don't need this here and we would need to alter the next line if left in.
neighborhood_merged = neighborhood_merged.join(neighborhoods_venues_sorted.set_index('Neighborhood'), on='Neighborhood')

neighborhood_merged.head()
```

Out[168]:

	City	Neighborhood	Lat	Lon	Cluster Labels	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7th Most Common Venue	8th Cor \
0	Vancouver	North Hastings-Sunrise	49.282336	-123.040000	3.0	Entertainment	Sport	Restaurant	Services	Retail	Nightlife	Outdoors	
1	Vancouver	North Grandview-Woodland	49.278839	-123.066843	6.0	Restaurant	Retail	Cafes and Coffee Shops	Nightlife	Entertainment	Travel / Transport	Sport	Se
2	Vancouver	South Hastings-Sunrise , North Renfrew-Colling...	49.258053	-123.040160	5.0	Retail	Restaurant	Services	Cafes and Coffee Shops	Travel / Transport	Sport	Outdoors	
3	Vancouver	South Grandview-Woodland , NE Kensington-Cedar...	49.253451	-123.066314	3.0	Sport	Retail	Outdoors	Services	Entertainment	Other	Travel / Transport	Resti
4	Vancouver	SE Kensington-Cedar Cottage , Victoria-Fraserview	49.222370	-123.068315	6.0	Restaurant	Retail	Services	Outdoors	Travel / Transport	Sport	Other	Ni

Next, add cluster descriptions.

These were created manually in Excel based on analysis of the neighborhood_merged dataframe.

```
In [169]: cluster_descriptions = pd.read_csv('cluster_labels_upload.csv')
cluster_descriptions.head()
```

```
Out[169]:
```

	City	Cluster Labels	Cluster Name
0	Calgary	0	Shopping
1	Calgary	0	Shopping
2	Calgary	0	Shopping
3	Calgary	0	Shopping
4	Calgary	1	Nightlife

Merge this dataframe with the neighborhood data:

```
In [171]: neighborhood_merged = neighborhood_merged.merge(cluster_descriptions)
neighborhood_merged.head()
```

Out[171]:

	City	Neighborhood	Lat	Lon	Cluster Labels	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7th Most Common Venue	8th Most Common Venue
0	Vancouver	North Hastings-Sunrise	49.282336	-123.040000	3.0	Entertainment	Sport	Restaurant	Services	Retail	Nightlife	Outdoors	Coffee Shop
1	Vancouver	North Hastings-Sunrise	49.282336	-123.040000	3.0	Entertainment	Sport	Restaurant	Services	Retail	Nightlife	Outdoors	Coffee Shop
2	Vancouver	North Hastings-Sunrise	49.282336	-123.040000	3.0	Entertainment	Sport	Restaurant	Services	Retail	Nightlife	Outdoors	Coffee Shop
3	Vancouver	South Grandview-Woodland, NE Kensington-Cedar...	49.253451	-123.066314	3.0	Sport	Retail	Outdoors	Services	Entertainment	Other	Travel / Transport	Restaurant
4	Vancouver	South Grandview-Woodland, NE Kensington-Cedar...	49.253451	-123.066314	3.0	Sport	Retail	Outdoors	Services	Entertainment	Other	Travel / Transport	Restaurant

Draw some more maps so that we can see where the clusters are:


```

In [172]: addresses = ['Calgary, Canada', 'Vancouver, Canada']
cities = ['Calgary', 'Vancouver']
maps_clusters = {0: 'MapCalgary',
                  1: 'MapVancouver'} #if we create a dictionary for the maps then we can add to them easily in the 'for' loop.

for i in range(0, len(addresses)):

    geolocator = Nominatim()
    location = geolocator.geocode(addresses[i])
    latitude = location.latitude
    longitude = location.longitude
    maps_clusters[i] = folium.Map(location=[latitude, longitude], zoom_start=11)

    # set color scheme for the clusters
    x = np.arange(kclusters)
    ys = [i + x + (i*x)**2 for i in range(kclusters)]
    colors_array = cm.rainbow(np.linspace(0, 1, len(ys)))
    rainbow = [colors.rgb2hex(i) for i in colors_array]

    # add markers to the map
    markers_colors = []
    #no foursquare venues found in some postal codes, hence these were not clustered. drop those records.
    neighborhood_merged = neighborhood_merged[~neighborhood_merged['Cluster Labels'].isnull()]
    for lat, lon, neigh, cluster, clusterN in zip(neighborhood_merged[neighborhood_merged['City']==cities[i]]['Lat'],
                                                neighborhood_merged[neighborhood_merged['City']==cities[i]]['Lon'],
                                                neighborhood_merged[neighborhood_merged['City']==cities[i]]['Neighborhood'],
                                                neighborhood_merged[neighborhood_merged['City']==cities[i]]['Cluster Labels'],
                                                neighborhood_merged[neighborhood_merged['City']==cities[i]]['Cluster Name']):
        label = folium.Popup(str(clusterN) + ': ' + str(neigh), parse_html=True)
        folium.CircleMarker(
            [lat, lon],
            radius=5,
            popup=label,
            color=rainbow[int(cluster)-1],
            fill=True,
            fill_color=rainbow[int(cluster)-1],
            fill_opacity=0.7).add_to(maps_clusters[i])

    #add the mountains to the maps
    for lat, lng, mountain in zip(df_city_mountains_final[df_city_mountains_final['City']==cities[i]]['Lat_y'],
                                df_city_mountains_final[df_city_mountains_final['City']==cities[i]]['Lon_y'],

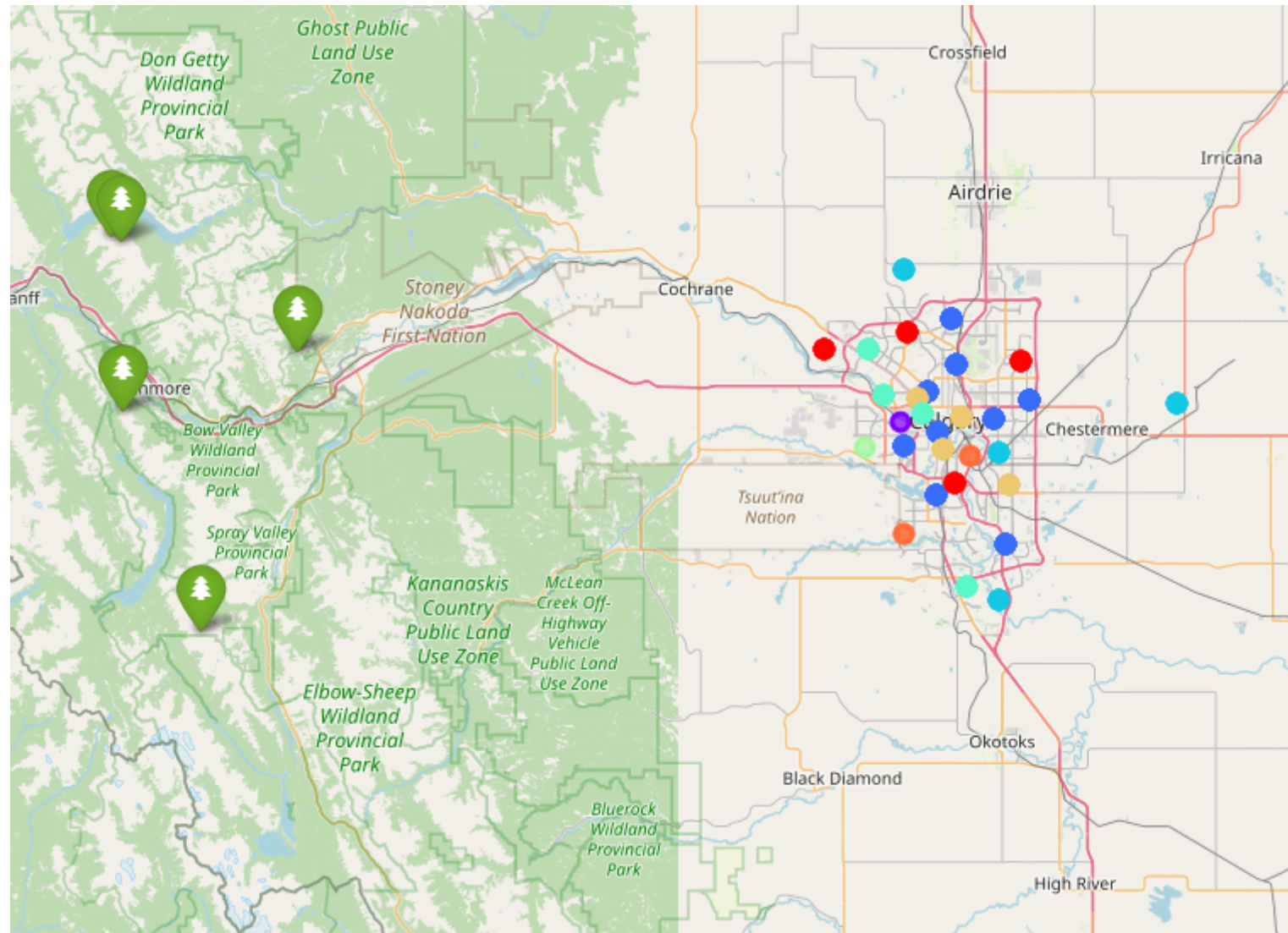
```

```
df_city_mountains_final[df_city_mountains_final['City']==cities[i]]['Mountain']  
label = '{}'.format(mountain)  
label = folium.Popup(label, parse_html=True)  
folium.Marker(  
    [lat, lng],  
    popup=label,  
    icon=folium.Icon(color="green", icon="fa-tree", prefix='fa')).add_to(maps_clusters[i])
```

C:\Users\johnh\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: DeprecationWarning: Using Nominatim with the default "geopy/1.21.0" `user_agent` is strongly discouraged, as it violates Nominatim's ToS <https://operations.osmfoundation.org/policies/nominatim/> (<https://operations.osmfoundation.org/policies/nominatim/>) and may possibly cause 403 and 429 HTTP errors. Please specify a custom `user_agent` with `Nominatim(user_agent="my-application")` or by overriding the default `user_agent`: `geopy.geocoders.options.default_user_agent = "my-application"`. In geopy 2.0 this will become an exception.

The cluster map for Calgary

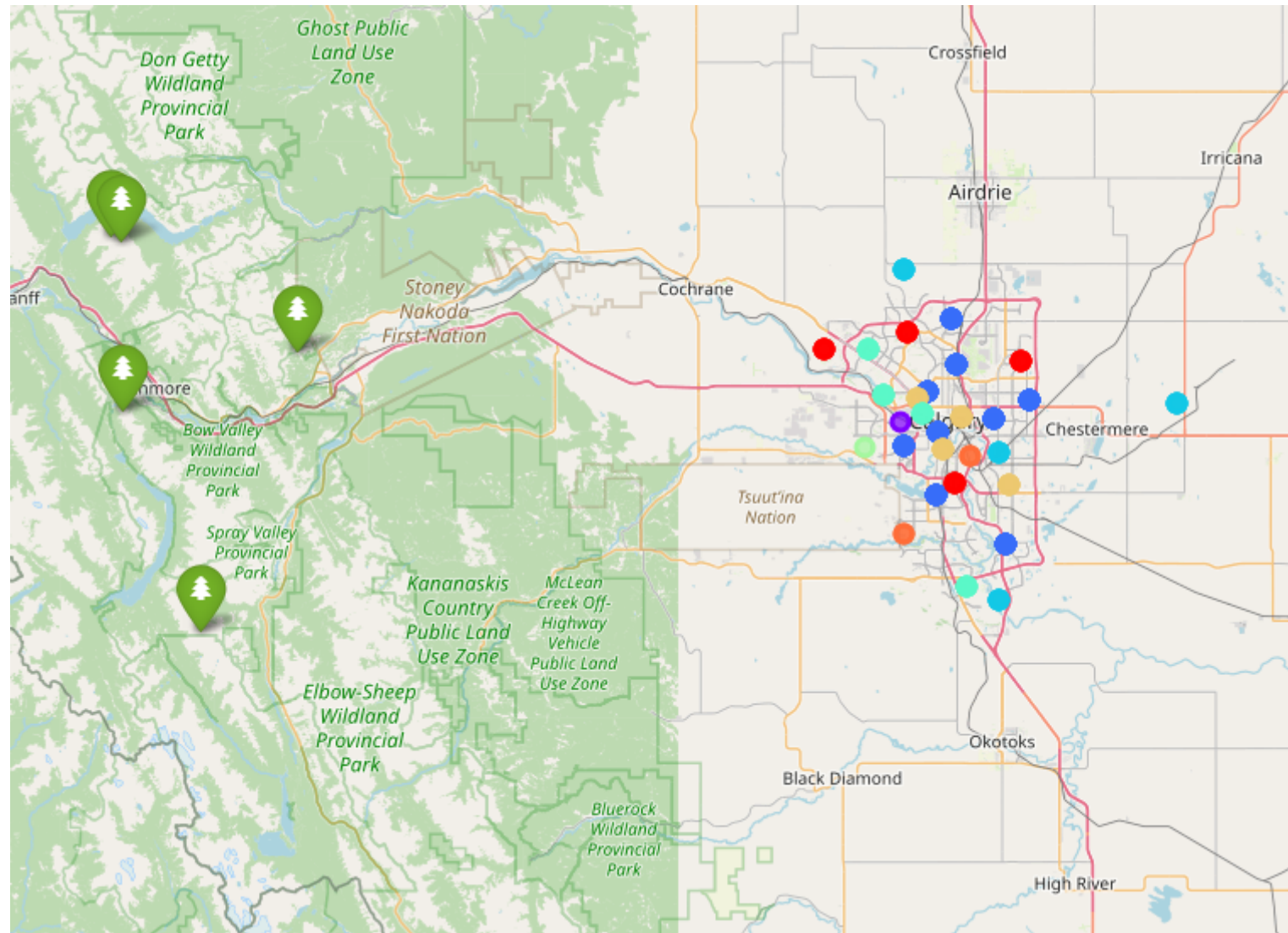
Again, images included for those who don't wish to run this themselves.



```
In [173]: maps_clusters[0]
```

```
Out[173]:
```

This is the cluster map for Vancouver



In [174]: `maps_clusters[1]`

Out[174]:

[Certificate \(https://www.coursera.org/account/accomplishments/specialization/certificate/SVLJCF45AA4F\)](https://www.coursera.org/account/accomplishments/specialization/certificate/SVLJCF45AA4F)



9 Courses

- What is Data Science?
- Tools for Data Science
- Data Science Methodology
- Python for Data Science and AI
- Databases and SQL for Data Science
- Data Analysis with Python
- Data Visualization with Python
- Machine Learning with Python
- Applied Data Science Capstone



05/13/2020

John Hills

has successfully completed the online, non-credit Professional Certificate

IBM Data Science

In this Professional Certificate learners developed and honed hands-on skills in Data Science and Machine Learning. Learners started with an orientation of Data Science and its Methodology, became familiar and used a variety of data science tools, learned Python and SQL, performed Data Visualization and Analysis, and created Machine Learning models. In the process they completed several labs and assignments on the cloud including a Capstone Project at the end to apply and demonstrate their knowledge and skills.

The online specialization named in this certificate may draw on material from courses taught on-campus, but the included courses are not equivalent to on-campus courses. Participation in this online specialization does not constitute enrollment at this university. This certificate does not confer a University grade, course credit or degree, and it does not verify the identity of the learner.

Rav Ahuja
AI & Data Science
Program Director
IBM Skills Network

Verify this certificate at:
coursera.org/verify/professional-cert/SVLJCF45AA4F

Output to CSV for next time.

```
In [ ]: df_neighborhoodVenues.to_csv('df_neighborhoodVenues.csv')
df_postal_codes.to_csv('df_postal_codes.csv')
df_city_mountains_final.to_csv('df_city_mountains_final.csv')
df_cities.to_csv('df_cities.csv')
df_mountains.to_csv('df_mountains.csv')
IMF.to_csv('IMF.csv')
neighborhood_merged.to_csv('neighborhood_merged.csv')
```

Upload CSVs for next time.

```
In [10]: df_neighborhoodVenues = pd.read_csv('df_neighborhoodVenues.csv')
df_postal_codes = pd.read_csv('df_postal_codes.csv')
df_city_mountains_final = pd.read_csv('df_city_mountains_final.csv')
df_cities = pd.read_csv('df_cities.csv')
df_mountains = pd.read_csv('df_mountains.csv')
IMF = pd.read_csv('IMF.csv')
neighborhood_merged = pd.read_csv('neighborhood_merged.csv')
```

```
In [ ]:
```