A

PROJECT REPORT

ON

CITY SIMILARITY TEST

Submitted in partial fulfillment of their requirements for the award under the supervision of

Dr. B.D.K. PATRO

Assistant Professor and HOD of CSE Department

SUBMITED BY

AKASH TRIPATHI (1683910004)

GOPI GUPTA (1783910904)

SUMIT SINGH (1783910907)

(BATCH-2016-2020)

Submitted to

RAJKIYA ENGINEERING COLLEGE KANNAUJ

ACKNOWLEDGEMENT

I would like to express my deep and sincere gratitude to my supervisor Dr. **B. D. K. PATRO (Assistant Professor and HOD of REC Kannauj),** who gave me his full support and encouraged me to work in an innovative and challenging project for Educational field. His wide knowledge and logical thinking gave me right direction all the time.

I am deeply grateful my project coordinator for his help and support provided at every step of the project.

Akash Tripathi (1683910004)

DECLARATION

This is to certify that the project report entitled "**City Similarity Test**" is done by me is an authentic work carried out for the partial fulfillment of the requirements for the award of the Degree in **"(Computer Science and Engineering)"** under the guidance of **Dr. B. D. K. PATRO**. The matter embodied in this project work has not been submitted earlier for award of any degree to the best of my knowledge and belief.

`

AKASH TRIPATHI

INDEX

INTRODUCTION

A TATA, Wipro, New York Luxury Brand and other multi-national brands has built its business in several cities all over world. Due to its success and growing popularity in these cities, the CEO and his team wants to expand their business to other cities in the India, UK States, China, Japan and also explore their market in big cities in other countries. Now the CEO has hired a data scientist and assigned her a task to find out the similarity between different big cities in the world and group the cities into various clusters, so that the Board of Directors can make a better decision of which business mode to operate in new cities.

PROBLEM STATEMENT

Now the CEO has hired a data scientist and assigned her a task to find out the similarity between different big cities in the world and group the cities into various clusters, so that the Board of Directors can make a better decision of which business mode to operate in new cities. (For Example: If London has been grouped into the same cluster with Delhi, the business mode operated in Delhi market will be considered for London). The similarity test should be based on various factors, including but not limiting to geolocations, economic development, cultures, and population components and so on. In order to carry out the task, the data scientist should make a full use of FourSquare API and collect a dataset for at least 15 cities, including those in the United States and those in other countries.

OBJECTIVES OF THE PROJECT

- As a hired data scientist from CEO of multinationals companies our job is to find out the similarity between different big cities in the world.
- Group the cities into various clusters, so that the Board of Directors can make a better decision of which business mode to operate in new cities.
- There are 3 data features used for model which are venue category data, city GDP data and city temperature or climate data.
- To compare the different cities, we will obtain an extensive list of venues using the Foursquare API.
- We are also working to find out the various features such as average salary of each person in a city, population density and crime rates such as robbing that can influence the similarity between two cities and more variables could be included for higher accuracy of clustering results.

Brief of Base Reference Paper(Daniel Preo,tiuc-Pietro,Justin Cranshaw,Tae Yano,Exploring venue based city to city measures,2013)

- Imagine two hypothetical cities, Concentralia and Dispersia, that are exactly the same in nearly every way, having exactly the same venues—the same universities, restaurants and parks. Suppose further that they only differ in the spatial arrangement of these venues, so that the venues in Dispersia are distributed uniformly throughout the city,and the venues in Concentralia are positioned more naturally— organically shaped alongside Concentralia's economic, political, and cultural evolution.

- With the growth of smart-phones,and location-based social networks, data is being generated about human activity in urban areas at a level of detail not seen before.

- One issue in comparing spatial regions such as cities is the normalization of absolute data, since often raw data from two different contexts are incomparable. Another challenge is the question of how to account for spatial effects. In our hypothetical comparison, the only difference between Concentralia and Dispersia was in the spatial distribution of their venues.

- Here define the bag of venues representation of a spatial region r.

- For the venues, we collected data from the widely used location-based Social Network (LBSN) Foursquare. Users of Foursquare"check-in" to their current location on their mobile device by selecting it from a list of nearby named venues.

Languages, Libraries and Platform used in this Project

LANGUAGE USED IN PROJECT

PYTHON

LIBRARIES USED IN PROJECT

Pandas: library written for the Python programming language for data manipulation and analysis.
Numpy: which stands for Numerical Python, is a **library** consisting of multidimensional array objects and a collection of routines for processing those arrays.
Matplotlib: Matplotlib is a plotting **library** for the Python programming language and its numerical mathematics extension NumPy.
Geopy: geopy is a Python client for several popular geocoding web services. **geopy** makes it easy for Python developers to locate the coordinates of addresses, cities, countries, and landmarks across the globe using third-party geocoders and other data sources.
Folium: The **library** has a number of built-in tilesets from OpenStreetMap, Mapbox, and Stamen, and supports custom tilesets with Mapbox or Cloudmade API keys. **folium** supports both Image, Video, GeoJSON and TopoJSON overlays.
BeautifulSoup: Beautiful Soup is a Python **library** for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree.

PLATFORM USED IN PROJECT

IBM Watson Studio (Jupyter Notebook)
IBM CLOUD For storage of data

Data Acquisition and Cleaning

DATA SOURCE

We **selected twenty nine preferred** cities **within the** world and clustered them **supported 3** factors, venues distribution, **GDP** indicator, and climate **varieties**. **the situation data** of these cities, **as well as** latitudes and longitudes, are obtained by **victimization** geolocator package on python. The venues **data** is retrieved from FourSquare API and **at the most five hundred** venues are there **for every town**, while the **GDP data** and climate **kind** information are scraped from **on-line** Wikipedia pages.

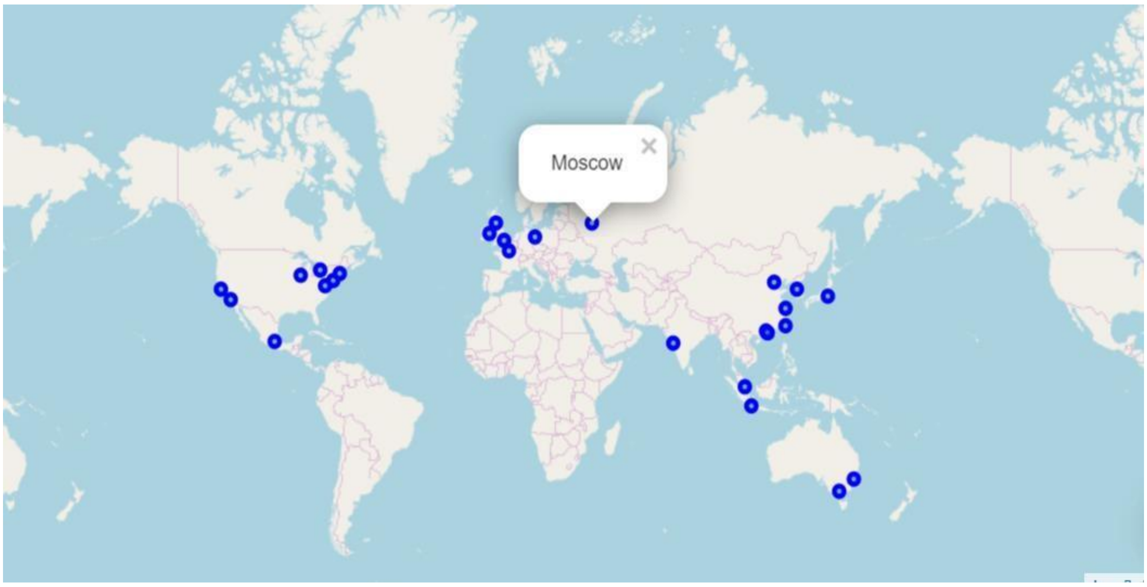


Figure 1. World map with location points

Data Cleaning

Data scraped from online sources contain extensive information that we might not need for analysis. Thus, we dropped out irrelevant data and only select those we need – venues category, annual GDP, and average annual temperature. Since venue categories are of the type string and need to be quantified for modeling, we apply one-hot coding to the venue category. The resulting data frame is as follows:

Table 1. City Venue Category (one-hot coding)

	City	Accessories Store	Adult Boutique	African Restaurant	American Restaurant	Amphitheater	Aquarium	Arcade	Ar Restau
0	New York	0	0	0	0	0	0	0	
1	New York	0	0	0	0	0	0	0	
2	New York	0	0	0	0	0	0	0	
3	New York	0	0	0	0	0	0	0	
4	New York	0	0	0	0	0	0	0	

Temperature and GDP data frame are as follows. The entry values are converted from strings to float numbers, and they are also normalized for modeling

	City	Normalized GDP	GDP
0	Tokyo	0.509116	1617.0
1	New York	0.441738	1403.0
2	Los Angeles	0.270930	860.5
3	Seoul	0.266333	845.9
4	London	0.263122	835.7

Table 2. City with GDP table

	City	Normalized Temperature	Temperature
0	Mumbai	0.307823	27.1
1	Singapore	0.306687	27.0
2	Jakarta	0.303279	26.7
3	Hong Kong	0.264659	23.3
4	Taipei	0.261252	23.0

Table 3. City with Temperature table

Data Flow Diagram

Introduction:-

DFD is an acronym for the word Data Flow Diagram. DFD is ppictorial representation of the system. DFD is a graphical representation of the flow of data through the information system. DFD are also used for the visualization of data processing (structured design). ADFD provides no information about the timings of the process, or about whether process will operate in parallel or sequence. DFD is an important technique for modeling system's high-level detail by showing how input details transformed to output results through a sequence of functional transformations. DFD reveal relationships among between the various components in a program or system. The strength of DFD lies in the fact that using few symbols we are able to express program design in an easier manner. ADFD can be used to represent the following:-

- External Entity sending and receiving data.
- Process that change the data.
- Flow of data within the system.
- Data Storage locations.

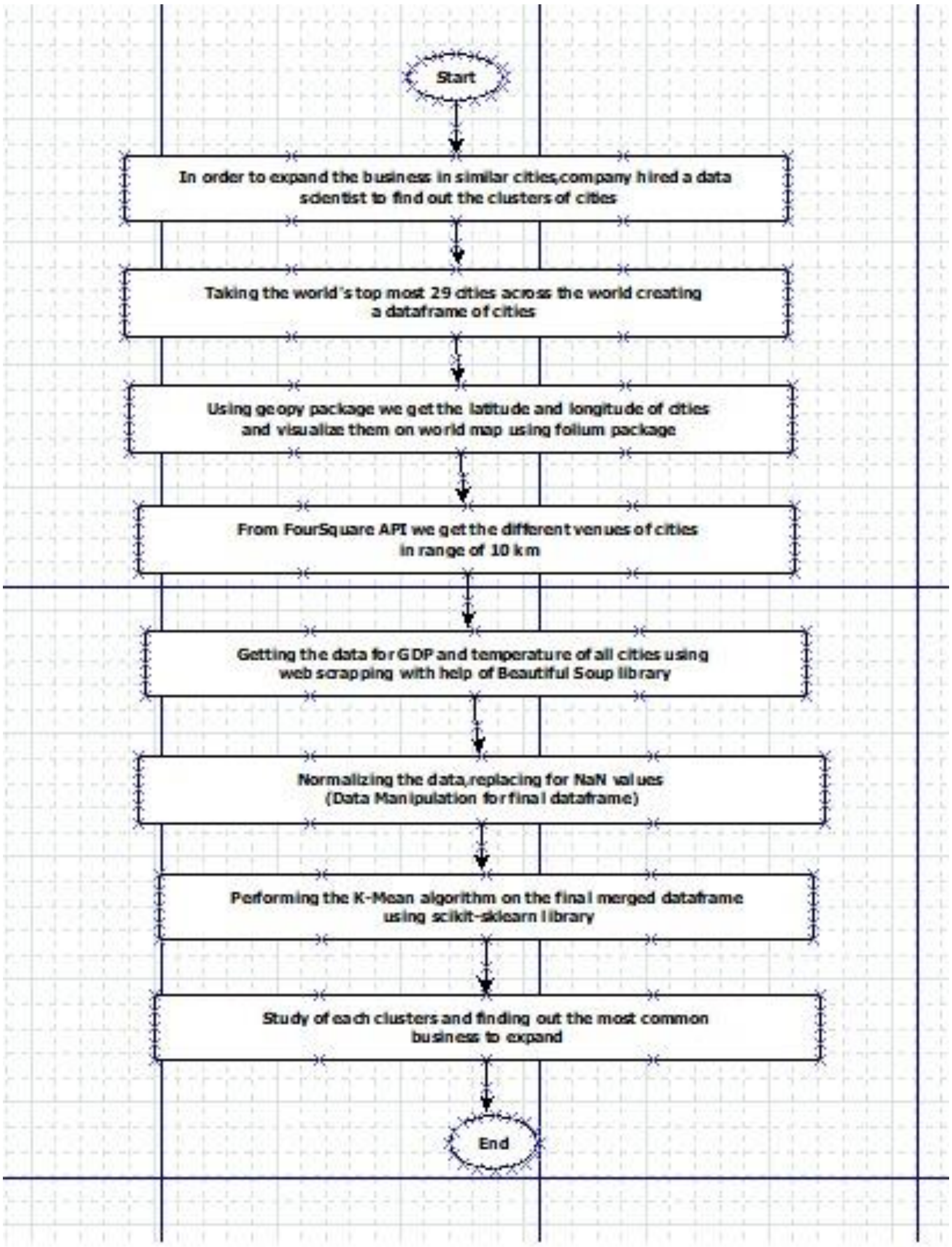
Uses of DFD:-

The main uses of data flow diagrams are as follows: -

DFD is a method of choice for representation of showing of information through a system because of the following reasons:-

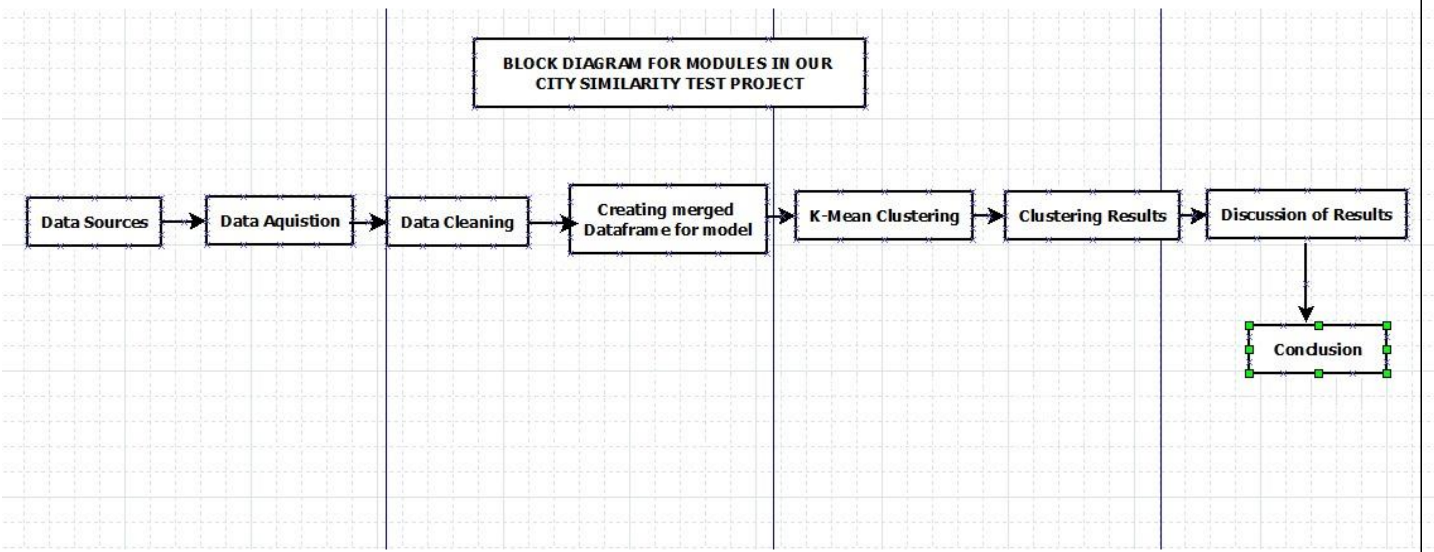
- DFDs are easier to understand by technical and non-technical audiences.
- DFDs can provide a high level system overview, complete with boundaries and connections to other system.
- DFDs can provide a detailed representation of system components.

DATA FLOW DIAGRAM



BLOCK DIAGRAM OF CITY SIMILARITY TEST

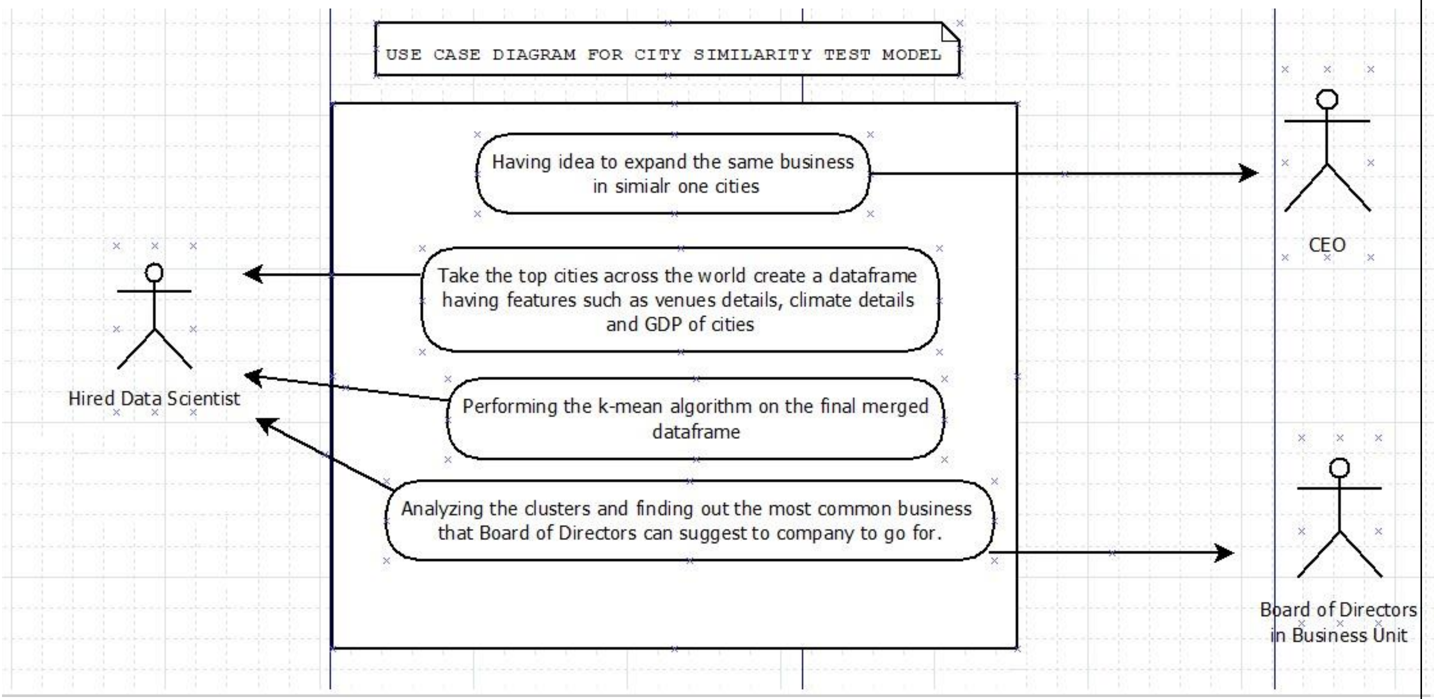
A **block diagram** is a **diagram** of a system in which the principal parts or functions are represented by **blocks** connected by lines that show the relationships of the **blocks**. They are heavily used in engineering in hardware design, electronic design, software design, and process flow **diagrams**.



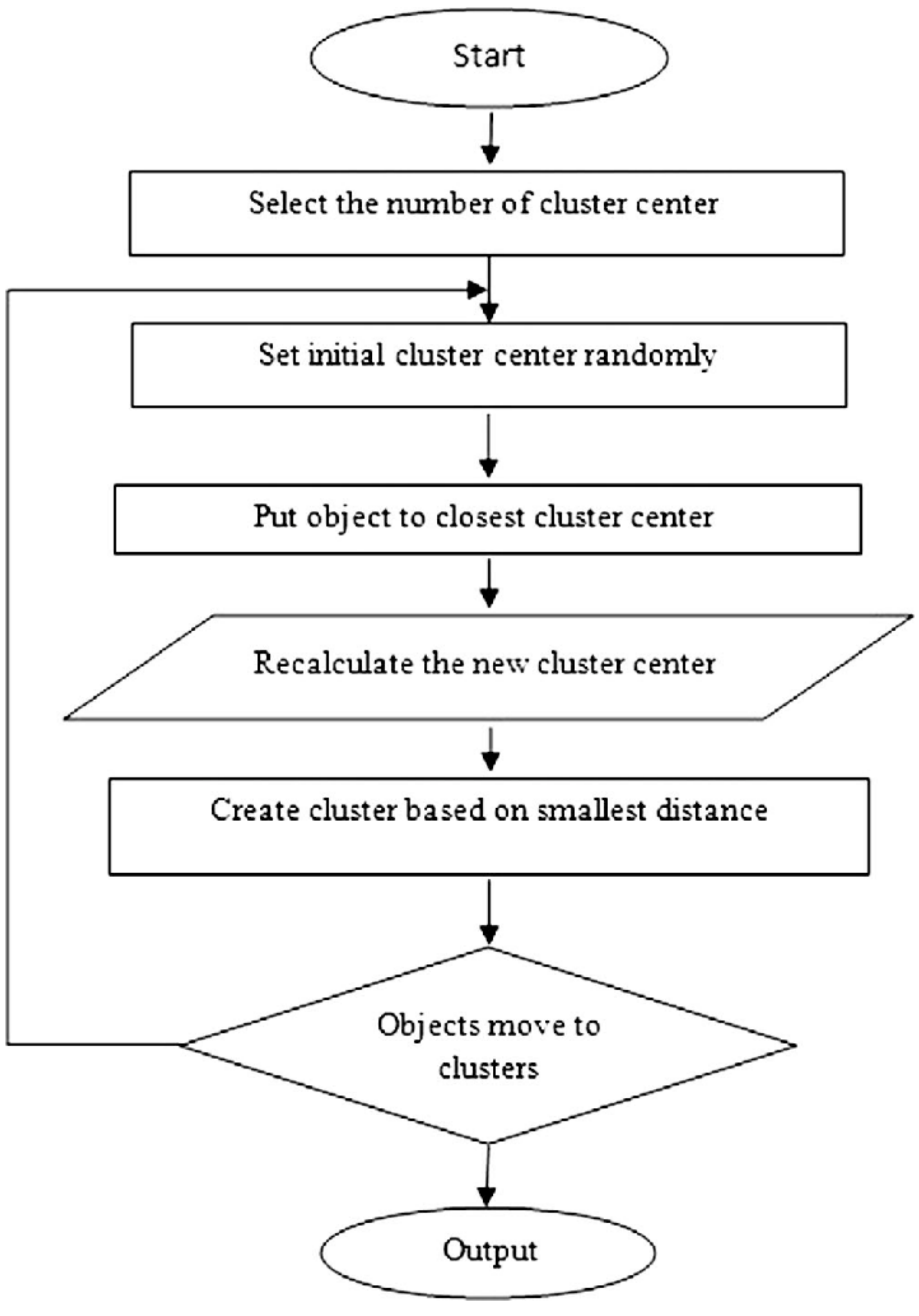
USE CASE DIAGRAM

A **use case diagram** is a dynamic or behavior **diagram** in UML. **Use case diagrams** model the functionality of a system **using** actors and **use** cases. **Use** cases are a set of actions, services, and functions that the system needs to perform. ... The "actors" are people or entities operating under defined roles within the system.

USE CASE DIAGRAM OF CITY SIMILARITY TEST



K-MEAN ALGORITHM USED FOR CLUSTERING IN THIS PROJECT



Data Section

We chose 29 most popular cities in the world and clustered them based on the three factors, venues distribution, GDP indicator, and climate types. The venues information is retrieved from FourSquare API and at most 500 venues were selected for each city, while the GDP information and climate type information are scraped from online Wikipedia pages.

Import Necessary Libraries

In [4]:
```
import pandas as pd
import numpy as np
import matplotlib as mpl
```

Set up a World City Dataframe

The world city dataframe includes information of city names, their locations features, and their country names.

In [6]:
```
## Create a dataframe of cities
City_data = {'City': ['New York', 'London','Edinburgh', 'Toronto', 'Sydney', 'Singapore',
                      'Melbourne', 'Hong Kong', 'Los Angeles',
                      'Chicago', 'Boston', 'San Francisco', 'Dublin', 'Washington','Beijing',
                      'Shanghai','Guangzhou', 'Shenzhen', 'Mumbai', 'Tokyo', 'Seoul-Incheon','Moscow','Paris'
,
                      'Taipei', 'Berlin', 'Jakarta', 'Mexico City','Delhi','Kolkata']}
City_df = pd.DataFrame(City_data)

## add up columns of 'Lat', 'Lng', 'Country'
## For lar, and lng, we use zero values first for later data fill-in
City_df.insert(1, 'Latitude', np.zeros(29))
City_df.insert(2, 'Longitude', np.zeros(29))
City_df.insert(3, 'Country', ['US', 'UK', 'UK', 'Canada', 'Australia', 'Singapore', 'Australia', 'China',
                              'US', 'US', 'US', 'US', 'Ireland', 'US', 'China', 'China', 'China', 'China', 'India', 'Japan',
                              'South Korea', 'Russia', 'France', 'China', 'Germany', 'Indon
esia', 'Mexico','India','India'])
```

In [10]:
```
City_df.iloc[0]
```

Out[10]:
```
City         New York
Latitude            0
Longitude           0
Country            US
Name: 0, dtype: object
```

In [11]:
```
City_df.head(29)
```
Out[11]:

	City	Latitude	Longitude	Country
0	New York	0.0	0.0	US
1	London	0.0	0.0	UK
2	Edinburgh	0.0	0.0	UK
3	Toronto	0.0	0.0	Canada
4	Sydney	0.0	0.0	Australia
5	Singapore	0.0	0.0	Singapore
6	Melbourne	0.0	0.0	Australia
7	Hong Kong	0.0	0.0	China
8	Los Angeles	0.0	0.0	US
9	Chicago	0.0	0.0	US
10	Boston	0.0	0.0	US
11	San Francisco	0.0	0.0	US
12	Dublin	0.0	0.0	Ireland
13	Washington	0.0	0.0	US
14	Beijing	0.0	0.0	China
15	Shanghai	0.0	0.0	China
16	Guangzhou	0.0	0.0	China
17	Shenzhen	0.0	0.0	China
18	Mumbai	0.0	0.0	India
19	Tokyo	0.0	0.0	Japan
20	Seoul–Incheon	0.0	0.0	South Korea
21	Moscow	0.0	0.0	Russia
22	Paris	0.0	0.0	France
23	Taipei	0.0	0.0	China
24	Berlin	0.0	0.0	Germany
25	Jakarta	0.0	0.0	Indonesia
26	Mexico City	0.0	0.0	Mexico
27	Delhi	0.0	0.0	India
28	Kolkata	0.0	0.0	India

Add in Latitude & Longitude Values

In [12]:
```
## Import necessary libraries
import geopy
from geopy.geocoders import Nominatim

## use geolocation package to retrieve location features (lat & lng) into the
dataframe for index, row in City_df.iterrows():
    city = row['City']
    geolocator = Nominatim(user_agent = "explorer2")
    location_city = geolocator.geocode(str(city))
    lat_city = location_city.latitude
    lng_city = location_city.longitude
    City_df.loc[index, 'Latitude'] = lat_city
    City_df.loc[index, 'Longitude'] =
    lng_city

City_df.head(29)
```

Out[12]:

	City	Latitude	Longitude	Country
0	New York	40.712728	-74.006015	US
1	London	51.507322	-0.127647	UK
2	Edinburgh	55.953346	-3.188375	UK
3	Toronto	43.653482	-79.383935	Canada
4	Sydney	-33.854816	151.216454	Australia
5	Singapore	1.357107	103.819499	Singapore
6	Melbourne	-37.814218	144.963161	Australia
7	Hong Kong	22.279328	114.162813	China
8	Los Angeles	34.053691	-118.242767	US
9	Chicago	41.875562	-87.624421	US
10	Boston	42.360253	-71.058291	US
11	San Francisco	37.779026	-122.419906	US
12	Dublin	53.349764	-6.260273	Ireland
13	Washington	38.894893	-77.036553	US
14	Beijing	39.906217	116.391276	China
15	Shanghai	31.232276	121.469207	China
16	Guangzhou	23.130196	113.259294	China
17	Shenzhen	22.555454	114.054330	China
18	Mumbai	18.938771	72.835335	India
19	Tokyo	35.682839	139.759455	Japan
20	Seoul–Incheon	37.440324	126.735400	South Korea
21	Moscow	55.479205	37.327330	Russia
22	Paris	48.856697	2.351462	France
23	Taipei	25.037520	121.563680	China
24	Berlin	52.517037	13.388860	Germany
25	Jakarta	-6.175394	106.827183	Indonesia
26	Mexico City	19.432630	-99.133178	Mexico
27	Delhi	28.651718	77.221939	India
28	Kolkata	22.545412	88.356775	India

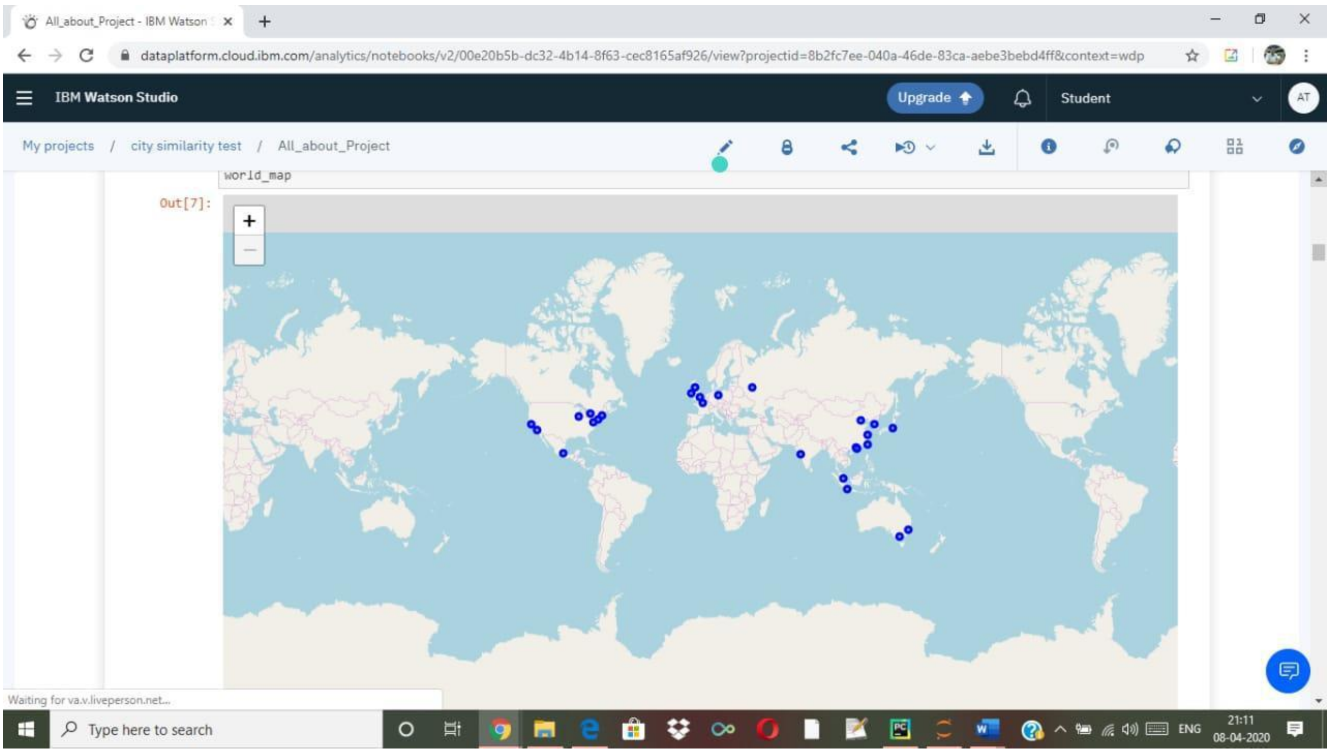
In [13]: ## Install relevant packages for visualization !conda install -c conda-forge folium=0.5.0 --yes

```
Solving environment: done

## Package Plan ##

  environment location: /opt/conda/envs/Python36

  added / updated specs:
    - folium=0.5.0


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    certifi-2020.4.5.1         |    py36h9f0ad1d_0         151 KB  conda-forge
    openssl-1.1.1f             |       h516909a_0         2.1 MB  conda-forge
    folium-0.5.0               |             py_0          45 KB  conda-forge
    vincent-0.4.4              |             py_1          28 KB  conda-forge
    ca-certificates-2020.4.5.1 |       hecc5488_0         146 KB  conda-forge
    python_abi-3.6             |          1_cp36m           4 KB  conda-forge
    branca-0.4.0               |             py_0          26 KB  conda-forge
    altair-4.1.0               |             py_1         614 KB  conda-forge
    ------------------------------------------------------------
                                           Total:         3.1 MB

The following NEW packages will be INSTALLED:

    altair:         4.1.0-py_1       conda-forge
    branca:         0.4.0-py_0       conda-forge
    folium:         0.5.0-py_0       conda-forge
    python_abi:     3.6-1_cp36m      conda-forge
    vincent:        0.4.4-py_1       conda-forge

The following packages will be UPDATED:

    ca-certificates: 2020.1.1-0                  -->   2020.4.5.1-hecc5488_0   conda-forge
    certifi:         2019.11.28-py36_0           --> 2020.4.5.1-py36h9f0ad1d_0 conda-forge
    openssl:         1.1.1e-h7b6447c_0           --> 1.1.1f-h516909a_0  conda-forge


Downloading and Extracting Packages
certifi-2020.4.5.1 | 151 KB | ##################################### | 100%
openssl-1.1.1f | 2.1 MB | ##################################### | 100% folium-
0.5.0 | 45 KB | ##################################### | 100% vincent-0.4.4 | 28
KB | ##################################### | 100% ca-certificates-2020 | 146 KB
| ##################################### | 100% python_abi-3.6 | 4 KB |
##################################### | 100% branca-0.4.0 | 26 KB |
##################################### | 100% altair-4.1.0 | 614 KB |
##################################### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

VISUALIZE THE CITIES ON WORLD MAP

```
In [14]: ## import necessary lib
         import folium

         ## create a world map
         world_map = folium.Map()

         ## add location marks on the world map
         for lati, lngi, city in zip(City_df['Latitude'], City_df['Longitude'], City_df['City']):
             label = '{}'.format(city)
             label = folium.Popup(label, parse_html=True)
             folium.CircleMarker(
                 [lati, lngi],
                 radius = 3,
                 popup = label,
                 color = 'blue',
                 fill = True,
                 fill_color = '#3186cc',
                 fill_opacity = 0.6,
                 parse_html = False
             ).add_to(world_map)

         world_map
```

Out[14]:



Scrape Data

We will collect and clean data of venues, GDP, and climates step by step in this section

Retrieve Venue Information for Cities

We retrieve at most 500 venues information for each city and add venue names and venue categories to the dataframe

In [15]:
```
## import necessary packages
import requests

## Client Information for Foursquare
CLIENT_ID = "YYZIJHKGABGQIHFD4SQH0RKMCD5E3JPUAIRCM1QLOANUILAU"
CLIENT_SECRET = "IQDDZ201VFA0XFRA2U1RKP30BDBYKL0XG42AXJ0LHKLTOPKX"
VERSION = '20190829'
LIMIT = 500
```

```
In [16]: ## Create a function to repeat process for all neighborhoods
         def getNearbyVenues(names, latitudes, longitudes, radius=10000):

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

                 # create the API request URL
                 url_city = 'https://api.foursquare.com/v2/venues/explore?&client_id={}&client_secret={}&v={}&ll={},{}
         &radius={}&limit={}'.format(
                     CLIENT_ID,
                     CLIENT_SECRET,
                     VERSION,
                     lat,
                     lng,
                     radius,
                     LIMIT)

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

                 # return only relevant information for each nearby
                 venue venues_list.append([(
                     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',
                           'City Latitude',
                           'City Longitude',
                           'Venue',
                           'Venue Latitude',
                           'Venue Longitude',
                           'Venue Category']
             return nearby_venues
```

In [17]:
```
## Fill in the location inforation of cities into the function and return a segregated dataframe of venues fo r all cities
world_venues = getNearbyVenues(names = City_df['City'], latitudes = City_df['Latitude'], longitudes = City_df
['Longitude'])
world_venues.head()
```

Out[17]:

	City	City Latitude	City Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
0	New York	40.712728	-74.006015	Korin	40.714824	-74.009404	Furniture / Home Store
1	New York	40.712728	-74.006015	Aire Ancient Baths	40.718141	-74.004941	Spa
2	New York	40.712728	-74.006015	9/11 Memorial North Pool	40.712077	-74.013187	Memorial Site
3	New York	40.712728	-74.006015	One World Trade Center	40.713069	-74.013133	Building
4	New York	40.712728	-74.006015	Washington Market Park	40.717046	-74.011095	Playground

In [18]:
```
## Check out the size of the dataset
world_venues.shape
```

Out[18]: (2898, 7)

In [20]:
```
## Apply onehot-coding to venue categories
world_onehot = pd.get_dummies(world_venues['Venue Category'], prefix = "", prefix_sep= "")
world_onehot.head()
```

Out[20]:

	Adult Boutique	American Restaurant	Amphitheater	Aquarium	Arcade	Argentinian Restaurant	Art Gallery	Art Museum	Arts & Crafts Store	Asian Restaurant	...	Wine Bar	Wine Shop	Winery
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0

5 rows × 344 columns

In [21]:
```
## Add city column back to dataframe
world_onehot[['City']] = world_venues[['City']]

# move city column to the first column
fixed_columns = [world_onehot.columns[-1]] + list(world_onehot.columns[:-1])
world_onehot_city = world_onehot[fixed_columns]

world_onehot_city.head()
```

Out[21]:

	City	Adult Boutique	American Restaurant	Amphitheater	Aquarium	Arcade	Argentinian Restaurant	Art Gallery	Art Museum	Arts & Crafts Store	...	Wine Bar	Wine Shop	Winery	Xin Resta
0	New York	0	0	0	0	0	0	0	0	0	...	0	0	0	
1	New York	0	0	0	0	0	0	0	0	0	...	0	0	0	
2	New York	0	0	0	0	0	0	0	0	0	...	0	0	0	
3	New York	0	0	0	0	0	0	0	0	0	...	0	0	0	
4	New York	0	0	0	0	0	0	0	0	0	...	0	0	0	

5 rows × 345 columns

In [13]:
```
cols=world_onehot.columns.to_list()
print(cols)
```

```
['Accessories Store', 'Adult Boutique', 'African Restaurant', 'American Restaurant', 'Amphitheater', 'Aquariu m', 'Arcade', 'Arepa Restaurant', 'Argentinian Restaurant', 'Art Gallery', 'Art Museum', 'Arts & Crafts Stor e', 'Asian Restaurant', 'Athletics & Sports', 'Australian Restaurant', 'Austrian Restaurant', 'Auto Worksho p', 'Awadhi Restaurant', 'BBQ Joint', 'Bagel Shop', 'Bakery', 'Bar', 'Baseball Stadium', 'Basketball Stadiu m', 'Bath House', 'Beach', 'Beach Bar', 'Bed & Breakfast', 'Beer Bar', 'Beer Garden', 'Beer Store', 'Beijing Restaurant', 'Belgian Restaurant', 'Bengali Restaurant', 'Bike Rental / Bike Share', 'Bistro', 'Board Shop', 'Boat or Ferry', 'Bookstore', 'Botanical Garden', 'Boutique', 'Boxing Gym', 'Brazilian Restaurant', 'Breakfas t Spot', 'Brewery', 'Bridge', 'Bubble Tea Shop', 'Buddhist Temple', 'Buffet', 'Building', 'Bunsik Restauran t', 'Burger Joint', 'Burrito Place', 'Butcher', 'Cafeteria', 'Café', 'Cajun / Creole Restaurant', 'Canal', 'C andy Store', 'Cantonese Restaurant', 'Capitol Building', 'Castle', 'Caucasian Restaurant', 'Cha Chaan Teng', 'Chaat Place', 'Cheese Shop', 'Chinese Breakfast Place', 'Chinese Restaurant', 'Chocolate Shop', 'Church', 'C limbing Gym', 'Clothing Store', 'Club House', 'Cocktail Bar', 'Coffee Shop', 'College Library', 'College Qua d', 'Comedy Club', 'Comfort Food Restaurant', 'Comic Shop', 'Concert Hall', 'Construction & Landscaping', 'Co nvenience Store', 'Cosmetics Shop', 'Coworking Space', 'Creperie', 'Cricket Ground', 'Cupcake Shop', 'Cycle S tudio', 'Dance Studio', 'Deli / Bodega', 'Department Store', 'Dessert Shop', 'Dhaba', 'Dim Sum Restaurant', 'Diner', 'Discount Store', 'Dive Bar', 'Dog Run', 'Donburi Restaurant', 'Donut Shop', 'Drugstore', 'Dumpling Restaurant', 'Eastern European Restaurant', 'Electronics Store', 'Event Space', 'Exhibit', 'Falafel Restauran t', 'Farmers Market', 'Fast Food Restaurant', 'Field', 'Filipino Restaurant', 'Fish & Chips Shop', 'Fish Mark et', 'Flea Market', 'Flower Shop', 'Food & Drink Shop', 'Food Court', 'Food Truck', 'Fountain', 'French Resta urant', 'Fried Chicken Joint', 'Fruit & Vegetable Store', 'Furniture / Home Store', 'Garden', 'Gastropub', 'G ay Bar', 'General Entertainment', 'German Restaurant', 'Gift Shop', 'Golf Course', 'Gourmet Shop', 'Governmen t Building', 'Greek Restaurant', 'Grocery Store', 'Gym', 'Gym / Fitness Center', 'Gym Pool', 'Gymnastics Gy m', 'Hainan Restaurant', 'Halal Restaurant', 'Harbor / Marina', 'Hawaiian Restaurant', 'Historic Site', 'Hist ory Museum', 'Hobby Shop', 'Hockey Arena', 'Hong Kong Restaurant', 'Hookah Bar', 'Hostel', 'Hot Dog Joint', 'Hotel', 'Hotel Bar', 'Hotpot Restaurant', 'Ice Cream Shop', 'Indian Restaurant', 'Indian Sweet Shop', 'Indie Movie Theater', 'Indie Theater', 'Indonesian Meatball Place', 'Indonesian Restaurant', 'Irani Cafe', 'Irish P ub', 'Island', 'Israeli Restaurant', 'Italian Restaurant', 'Japanese Curry Restaurant', 'Japanese Restauran t', 'Javanese Restaurant', 'Jazz Club', 'Jewelry Store', 'Juice Bar', 'Kaiseki Restaurant', 'Karaoke Bar', 'K ebab Restaurant', 'Korean Restaurant', 'Lake', 'Latin American Restaurant', 'Library', 'Lingerie Store', 'Liq uor Store', 'Lounge', 'Manadonese Restaurant', 'Marijuana Dispensary', 'Market', 'Massage Studio', 'Mediterra nean Restaurant', 'Memorial Site', "Men's Store", 'Mexican Restaurant', 'Middle Eastern Restaurant', 'Mini Go lf', 'Modern European Restaurant', 'Molecular Gastronomy Restaurant', 'Monument / Landmark', 'Moroccan Restau rant', 'Mosque', 'Motel', 'Mountain', 'Movie Theater', 'Mughlai Restaurant', 'Multicuisine Indian Restauran t', 'Multiplex', 'Museum', 'Music Venue', 'Nature Preserve', 'Neighborhood', 'New American Restaurant', 'Nigh t Market', 'Nightclub', 'Noodle House', 'North Indian Restaurant', 'Opera House', 'Optical Shop', 'Organic Gr ocery', 'Other Great Outdoors', 'Other Nightlife', 'Outdoor Sculpture', 'Outlet Mall', 'Padangnese Restauran t', 'Park', 'Parsi Restaurant', 'Pastry Shop', 'Pedestrian Plaza', 'Peking Duck Restaurant', 'Pelmeni House', 'Performing Arts Venue', 'Peruvian Restaurant', 'Pet Café', 'Pet Service', 'Pet Store', 'Pie Shop', 'Pier', 'Pilates Studio', 'Pizza Place', 'Planetarium', 'Playground', 'Plaza', 'Poke Place', 'Pool', 'Portuguese Rest aurant', 'Post Office', 'Pub', 'Public Art', 'Ramen Restaurant', 'Record Shop', 'Recreation Center', 'Rental Car Location', 'Reservoir', 'Resort', 'Restaurant', 'River', 'Road', 'Rock Club', 'Roof Deck', 'Rugby Stadiu m', 'Russian Restaurant', 'Sake Bar', 'Salad Place', 'Salon / Barbershop', 'Samgyetang Restaurant', 'Sandwich Place', 'Sauna / Steam Room', 'Scandinavian Restaurant', 'Scenic Lookout', 'Schnitzel Restaurant', 'Science M useum', 'Scottish Restaurant', 'Sculpture Garden', 'Seafood Restaurant', 'Shaanxi Restaurant', 'Shabu-Shabu R estaurant', 'Shanghai Restaurant', 'Shanxi Restaurant', 'Shoe Store', 'Shopping Mall', 'Shopping Plaza', 'Shr ine', 'Skating Rink', 'Smoke Shop', 'Smoothie Shop', 'Snack Place', 'Soba Restaurant', 'Soccer Field', 'Socce r Stadium', 'Social Club', 'Soup Place', 'South Indian Restaurant', 'Souvenir Shop', 'Spa', 'Spanish Restaura nt', 'Speakeasy', 'Spiritual Center', 'Sporting Goods Shop', 'Sports Club', 'Stadium', 'State / Provincial Pa rk', 'Stationery Store', 'Steakhouse', 'Street Art', 'Street Food Gathering', 'Sukiyaki Restaurant', 'Superma rket', 'Sushi Restaurant', 'Synagogue', 'Taco Place', 'Tailor Shop', 'Taiwanese Restaurant', 'Tapas Restauran t', 'Tattoo Parlor', 'Tea Room', 'Temple', 'Tennis Stadium', 'Thai Restaurant', 'Theater', 'Theme Park', 'The me Restaurant', 'Tibetan Restaurant', 'Tonkatsu Restaurant', 'Tour Provider', 'Toy / Game Store', 'Trail', 'T rain Station', 'Trattoria/Osteria', 'Travel Agency', 'Turkish Restaurant', 'Udon Restaurant', 'Used Bookstor e', 'Vacation Rental', 'Vegetarian / Vegan Restaurant', 'Veterinarian', 'Video Store', 'Vietnamese Restauran t', 'Wagashi Place', 'Watch Shop', 'Water Park', 'Waterfront', 'Whisky Bar', 'Wine Bar', 'Wine Shop', 'Winer y', 'Wings Joint', 'Xinjiang Restaurant', 'Yakitori Restaurant', 'Yoga Studio', 'Yoshoku Restaurant', 'Yunnan Restaurant', 'Zhejiang Restaurant', 'Zoo']
```

```
In [22]: ## Group the dataset by the city names to check out the percentage of each venue
         categores world_grouped = world_onehot_city.groupby('City').mean().reset_index()
         world_grouped.head()
```

Out[22]:

	City	Adult Boutique	American Restaurant	Amphitheater	Aquarium	Arcade	Argentinian Restaurant	Art Gallery	Art Museum	Arts & Crafts Store	...	Wine Bar	Wine Shop	Winery	Re
0	Beijing	0.0	0.00	0.00	0.0	0.00	0.0	0.02	0.00	0.00	...	0.00	0.00	0.0	
1	Berlin	0.0	0.00	0.00	0.0	0.00	0.0	0.01	0.00	0.02	...	0.02	0.00	0.0	
2	Boston	0.0	0.03	0.00	0.0	0.00	0.0	0.00	0.00	0.00	...	0.01	0.02	0.0	
3	Chicago	0.0	0.01	0.01	0.0	0.00	0.0	0.00	0.01	0.00	...	0.00	0.00	0.0	
4	Delhi	0.0	0.00	0.00	0.0	0.01	0.0	0.01	0.01	0.00	...	0.00	0.00	0.0	

5 rows × 345 columns

```
In [23]: ## Define a function that sorts the values in rows

         def return_most_common_venues(row, num_top_venues):
             row_categories = row.iloc[1:]
             row_categories_sorted = row_categories.sort_values(ascending=False)

             return row_categories_sorted.index.values[0:num_top_venues]
```

```
In [24]: ## create columns according to number of top
         venues columns = ['City']
         for ind in np.arange(10):
             try:
                 columns.append('{}{} Most Common Venue'.format(ind+1, indicators[ind]))
             except:
                 columns.append('{}th Most Common Venue'.format(ind+1))

         ## Create a dataframe
         ## set up the column names for the dataframe
         City_venue_sorted = pd.DataFrame(columns = columns)

         ## set the column of "City"
         City_venue_sorted['City'] = world_grouped['City']

         ## Set the other column values -- the top 10 venue
         names for ind in np.arange(world_grouped.shape[0]):
             City_venue_sorted.iloc[ind, 1:] = return_most_common_venues(world_grouped.iloc[ind, :], 10)

         City_venue_sorted
```

Out[24]:

	City	1th Most Common Venue	2th Most Common Venue	3th 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	Beijing	Historic Site	Hotel	Park	Café	Shopping Mall	Brewery	Coffee Shop	Dumpling Restaurant	Temple	Cocktail Bar
1	Berlin	Park	Coffee Shop	Bookstore	Gourmet Shop	Ice Cream Shop	Monument / Landmark	Chocolate Shop	Hotel	Concert Hall	Bakery
2	Boston	Park	Bakery	Italian Restaurant	Seafood Restaurant	Salad Place	Gym	Gastropub	French Restaurant	Coffee Shop	Hotel
3	Chicago	Hotel	New American Restaurant	Coffee Shop	Park	Sandwich Place	Grocery Store	Donut Shop	Bar	Seafood Restaurant	Café
4	Delhi	Indian Restaurant	Hotel	Café	Lounge	Monument / Landmark	Deli / Bodega	Park	Bar	South Indian Restaurant	Coffee Shop
5	Dublin	Park	Café	Pub	Coffee Shop	Burger Joint	Plaza	Restaurant	Stadium	Ice Cream Shop	Historic Site
6	Edinburgh	Park	Coffee Shop	Café	Pub	Beer Bar	Hotel	Bar	Scenic Lookout	Cocktail Bar	Historic Site
7	Guangzhou	Hotel	Coffee Shop	Shopping Mall	Park	Turkish Restaurant	Cantonese Restaurant	Café	Electronics Store	Snack Place	Cocktail Bar
8	Hong Kong	Hotel	Italian Restaurant	Japanese Restaurant	Thai Restaurant	Bar	Chinese Restaurant	Café	Scenic Lookout	Gym / Fitness Center	Park
9	Jakarta	Hotel	Coffee Shop	Shopping Mall	Steakhouse	Indonesian Restaurant	Restaurant	Lounge	Bakery	Fast Food Restaurant	Clothing Store
10	Kolkata	Chinese Restaurant	Hotel	Café	Shopping Mall	Indian Restaurant	Indian Sweet Shop	Dhaba	Bookstore	Mughlai Restaurant	Multiplex
11	London	Hotel	Cocktail Bar	Park	Art Gallery	Art Museum	Coffee Shop	Theater	Lounge	Department Store	Hotel Bar
12	Los Angeles	Coffee Shop	Taco Place	Brewery	Art Gallery	Food Truck	American Restaurant	Sushi Restaurant	Theater	Italian Restaurant	Bakery
13	Melbourne	Café	Coffee Shop	Park	Bar	Ice Cream Shop	Cocktail Bar	Wine Bar	Plaza	Asian Restaurant	Monument / Landmark
14	Mexico City	Ice Cream Shop	Park	Art Museum	Bakery	Coffee Shop	Hotel	Mexican Restaurant	Taco Place	Asian Restaurant	Public Art
15	Moscow	Park	Café	Restaurant	Supermarket	Hotel	Convenience Store	Rest Area	Housing Development	Stables	Soccer Field
16	Mumbai	Indian Restaurant	Hotel	Café	Lounge	Fast Food Restaurant	Cricket Ground	Dessert Shop	Scenic Lookout	Ice Cream Shop	Restaurant
17	New York	Park	Bookstore	Ice Cream Shop	Gourmet Shop	Bakery	Movie Theater	Scenic Lookout	Pier	Furniture / Home Store	Sandwich Place
18	Paris	Plaza	French Restaurant	Hotel	Wine Bar	Art Museum	Bakery	Fountain	Italian Restaurant	Garden	Restaurant
19	San Francisco	Park	Coffee Shop	Bakery	Ice Cream Shop	Grocery Store	Pizza Place	Gym	New American Restaurant	Marijuana Dispensary	Yoga Studio
20	Seoul–Incheon	Coffee Shop	Park	Korean Restaurant	Market	BBQ Joint	Multiplex	Bakery	Fast Food Restaurant	Café	Golf Course
21	Shanghai	Hotel	Coffee Shop	Shopping Mall	Bakery	Dumpling Restaurant	Spa	Gym / Fitness Center	Italian Restaurant	Hotpot Restaurant	Restaurant
22	Shenzhen	Hotel	Shopping Mall	Coffee Shop	Electronics Store	Café	Park	Chinese Restaurant	Bar	Hotpot Restaurant	New American Restaurant
23	Singapore	Hotel	Japanese Restaurant	Ice Cream Shop	Shopping Mall	Park	Italian Restaurant	Clothing Store	Dessert Shop	Greek Restaurant	Chinese Restaurant
24	Sydney	Park	Café	Scenic Lookout	Coffee Shop	Theater	Bakery	Pizza Place	Ice Cream Shop	Garden	Hotel
25	Taipei	Hotel	Bakery	Café	Noodle House	Dumpling Restaurant	Dessert Shop	Chinese Restaurant	Taiwanese Restaurant	Bookstore	Park
26	Tokyo	Hotel	Art Museum	Chinese Restaurant	Ramen Restaurant	Wagashi Place	Tonkatsu Restaurant	Coffee Shop	Sake Bar	Garden	BBQ Joint
27	Toronto	Coffee Shop	Park	Bakery	Café	Mexican Restaurant	Hotel	Vegetarian / Vegan Restaurant	Sandwich Place	Farmers Market	Diner
28	Washington	Monument / Landmark	Art Museum	Hotel	Park	History Museum	Ice Cream Shop	Coffee Shop	Garden	Theater	American Restaurant

SCRAPE GDP DATA ONLINE

In [25]:
```python
## Import necessary lib
import pandas as pd
import requests
from bs4 import BeautifulSoup

## scrape datasets from website -- wikipedia page table
res = requests.get("https://en.wikipedia.org/wiki/List_of_cities_by_GDP")
soup = BeautifulSoup(res.content, 'lxml')
table = soup.find_all('table')[0]
City_GDP = pd.read_html(str(table))
City_GDP_dp = City_GDP[0][['City proper /Metropolitan area', 'Brookings Institution[5]2014 est.PPP-
adjustedGD P ($BN)']]
City_GDP_dp.head()
```

Out[25]:

	City proper /Metropolitan area	Brookings Institution[5]2014 est.PPP-adjustedGDP ($BN)
0	Aachen-Liège-Maastricht	99.7
1	Aberdeen	NaN
2	Abidjan	NaN
3	Abu Dhabi	178.3
4	Addis Ababa	NaN

In [26]:
```python
## Change the column names for convenience
City_GDP_dp.columns = ['City', 'GDP']

## convert city names in world_grouped dataframe into a
list city_list = world_grouped['City'].tolist()

## Filter out data for relevant cities
gdp_filtered = []
for index, row in City_GDP_dp.iterrows():
    if row['City'] in city_list:
        gdp_filtered.append([row['City'], row['GDP']])

## print out city names that match >> turn out there are two cities that are not matched in the
 dataframe >> Seoul-Incheon & Washington
gdp_filtered
```

Out[26]:
```
[['Beijing', '506.1'],
 ['Berlin', '157.7'],
 ['Boston', '360.1'],
 ['Chicago', '563.2'],
 ['Delhi', '293.6'],
 ['Dublin', '90.1'],
 ['Edinburgh', '32.5'],
 ['Guangzhou', '380.3'],
 ['Hong Kong', '416.0'],
 ['Jakarta', '321.3'],
 ['Kolkata', '150'],
 ['London', '835.7'],
 ['Los Angeles', '860.5'],
 ['Melbourne', '178.4'],
 ['Mexico City', '403.6'],
 ['Moscow', '553.3'],
 ['Mumbai', '150.9'],
 ['New York', '1403'],
 ['Paris', '715.1'],
 ['San Francisco', '331.0'],
 ['Shanghai', '594.0'],
 ['Shenzhen', '363.2'],
 ['Singapore', '365.9'],
 ['Sydney', '223.4'],
 ['Taipei', '327.3'],
 ['Tokyo', '1617'],
 ['Toronto', '276.3']]
```

In [27]:
```python
## Add up the two cities -- Seoul-Incheon & Washington into the
list for index, row in City_GDP_dp.iterrows():
    if row['City'] in ['Washington, DC', 'Seoul']:
        gdp_filtered.append([row['City'], row['GDP']])

## Convert the list into a dataframe
gdp_filtered_df = pd.DataFrame(gdp_filtered)
gdp_filtered_df.columns = ['City', 'GDP']

## drop repeated rows in the dataframe and convert GDP column into float
gdp_filtered_df = pd.DataFrame(gdp_filtered_df.drop_duplicates())
gdp_filtered_df['GDP'] = pd.to_numeric(gdp_filtered_df['GDP'])
```

In [30]:
```python
gdp_filtered_df.head(29)
```

Out[30]:

	City	GDP
0	Beijing	506.1
1	Berlin	157.7
2	Boston	360.1
3	Chicago	563.2
4	Delhi	293.6
5	Dublin	90.1
6	Edinburgh	32.5
7	Guangzhou	380.3
8	Hong Kong	416.0
9	Jakarta	321.3
10	Kolkata	150.0
11	London	835.7
12	Los Angeles	860.5
13	Melbourne	178.4
14	Mexico City	403.6
15	Moscow	553.3
16	Mumbai	150.9
17	New York	1403.0
18	Paris	715.1
19	San Francisco	331.0
20	Shanghai	594.0
21	Shenzhen	363.2
22	Singapore	365.9
23	Sydney	223.4
24	Taipei	327.3
25	Tokyo	1617.0
26	Toronto	276.3
27	Seoul	845.9
28	Washington, DC	442.2

Rank Cities in GDP Values

In [31]:
```
## Rank Cities in GDP values and sort values
gdp_filtered_sorted = gdp_filtered_df.sort_values('GDP', ascending = False)
gdp_filtered_sorted.head(10)
```

Out[31]:

	City	GDP
25	Tokyo	1617.0
17	New York	1403.0
12	Los Angeles	860.5
27	Seoul	845.9
11	London	835.7
18	Paris	715.1
20	Shanghai	594.0
3	Chicago	563.2
15	Moscow	553.3
0	Beijing	506.1

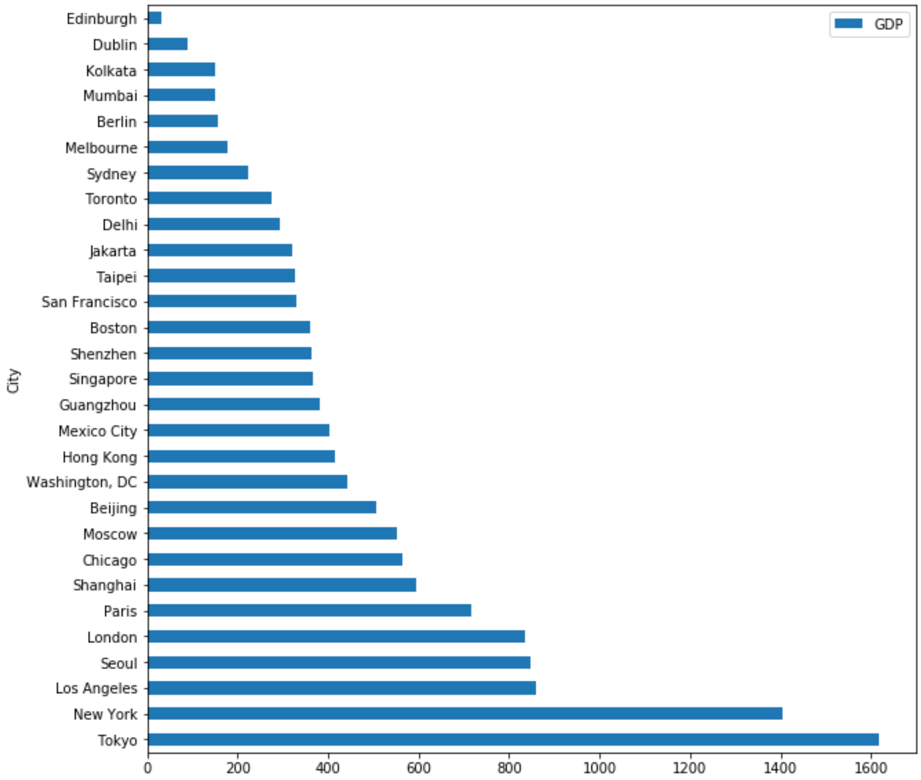
In [32]:
```
## The last 5 cities in rank of GDP
gdp_filtered_sorted.tail(5)
```

Out[32]:

	City	GDP
1	Berlin	157.7
16	Mumbai	150.9
10	Kolkata	150.0
5	Dublin	90.1
6	Edinburgh	32.5

In [34]:
```
## Visualize the ranking with a bar chart
gdp_visualize = gdp_filtered_sorted
gdp_visualize = gdp_visualize.set_index('City')
import matplotlib as mpl
gdp_visualize.plot(kind = 'barh',
                        figsize = (10, 10))
```

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x7f443f46d8d0>



In [35]:
```
## reset index for GDP dataset
gdp_filtered_sorted1 = gdp_filtered_sorted.reset_index().drop(['index'], axis = 1)
```

In [36]:
```
## import necessary libs
from sklearn import preprocessing

## Standardize datasets
scaler = preprocessing.StandardScaler()
gdp_array = np.array(gdp_filtered_sorted['GDP'])
gdp_normalized_array = preprocessing.normalize([gdp_array])

## add the normalized gdp back into the dataframe
gdp_column = pd.DataFrame(gdp_normalized_array).transpose()
gdp_filtered_sorted1.insert(1, 'Normalized GDP', gdp_column)
gdp_filtered_sorted1.head()
```

Out[36]:

	City	Normalized GDP	GDP
0	Tokyo	0.506395	1617.0
1	New York	0.439377	1403.0
2	Los Angeles	0.269482	860.5
3	Seoul	0.264910	845.9
4	London	0.261716	835.7

In [37]:
```
## scrape datasets from website -- wikipedia page
table import requests
from bs4 import BeautifulSoup
import pandas as pd

## scrape data from the Wikipedia avergae temperature page
res1 = requests.get("https://en.wikipedia.org/wiki/List_of_cities_by_average_temperature")
soup1 = BeautifulSoup(res1.content, 'lxml')

## scrape Asia table
table_Asia = soup1.find_all('table')[1]
Asia_temp = pd.read_html(str(table_Asia))[0]

## scrape Europe table
table_Europe = soup1.find_all('table')[2]
Europe_temp = pd.read_html(str(table_Europe))[0]

## scrape America table
table_America = soup1.find_all('table')[3]
America_temp = pd.read_html(str(table_America))[0]

## scrape Australia table
table_Australia = soup1.find_all('table')[4]
Australia_temp = pd.read_html(str(table_Australia))[0]

Australia_temp.tail()
```

Out[37]:

	Country	City	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	
21	New Zealand	Wellington	16.9(62.4)	17.2(63.0)	15.8(60.4)	13.7(56.7)	11.7(53.1)	9.7(49.5)	8.9(48.0)	9.4(48.9)	10.8(51.4)	12.0(53.6)	13
22	Papua New Guinea	Port Moresby	27.4(81.3)	27.3(81.1)	27.1(80.8)	27.0(80.6)	26.9(80.4)	26.1(79.0)	25.7(78.3)	26.1(79.0)	26.5(79.7)	27.5(81.5)	27
23	Solomon Islands	Honiara	26.7(80.1)	26.6(79.9)	26.6(79.9)	26.5(79.7)	26.6(79.9)	26.4(79.5)	26.1(79.0)	26.2(79.2)	26.5(79.7)	26.5(79.7)	26
24	United States	Honolulu	22.9(73.2)	22.9(73.2)	23.7(74.7)	24.6(76.3)	25.6(78.1)	26.8(80.2)	27.4(81.3)	27.8(82.0)	27.6(81.7)	26.8(80.2)	25
25	Vanuatu	Port Vila	26.4(79.5)	26.5(79.7)	26.3(79.3)	25.3(77.5)	24.1(75.4)	23.0(73.4)	22.1(71.8)	22.0(71.6)	22.7(72.9)	23.4(74.1)	24

In [46]:
```
## set up a list to store relevant data
temp_list = []

## Filter out data for relevant cities >> in
Asia for index, row in Asia_temp.iterrows():
    if row['City'] in city_list:
        temp_list.append([row['City'], row['Year']])

## Filter out data for relevant cities >> in
Europe for index, row in Europe_temp.iterrows():
    if row['City'] in city_list:
        temp_list.append([row['City'], row['Year']])

 ## Filter out data for relevant cities >> in
America for index, row in America_temp.iterrows():
    if row['City'] in city_list:
        temp_list.append([row['City'], row['Year']])

## Filter out data for relevant cities >> in Australia for
index, row in Australia_temp.iterrows():
    if row['City'] in city_list:
        temp_list.append([row['City'], row['Year']])

## check if data for all cities are successfully
extracted len(temp_list)
```

Out[46]: 22

```
In [47]: ## check out which cities are missing
         temp_list
```

```
Out[47]: [['Beijing', '12.9(55.2)'],
          ['Shanghai', '16.7(62.1)'],
          ['Hong Kong', '23.3(73.9)'],
          ['Kolkata', '26.7(80.1)'],
          ['Mumbai', '27.1(80.8)'],
          ['Jakarta', '26.7(80.1)'],
          ['Tokyo', '15.4(59.7)'],
          ['Singapore', '27(81)'],
          ['Taipei', '23.0(73.4)'],
          ['Paris', '12.3(54.1)'],
          ['Berlin', '10.3(50.5)'],
          ['Dublin', '9.8(49.6)'],
          ['Moscow', '5.8(42.4)'],
          ['Edinburgh', '9.3(48.7)'],
          ['London', '10.3(50.5)'],
          ['Toronto', '9.4(48.9)'],
          ['Mexico City', '17.5(63.5)'],
          ['Boston', '10.9(51.7)'],
          ['Chicago', '9.8(49.7)'],
          ['Los Angeles', '18.6(65.4)'],
          ['Melbourne', '15.1(59.2)'],
          ['Sydney', '17.7(63.9)']]
```

```
In [48]: ## add up the missing cities
         ## Seoul
         for index, row in Asia_temp.iterrows():
             if row['City'] in ['Seoul']:
                 temp_list.append(['Seoul-Incheon', row['Year']])

         ## Washington,D.C., San Francisco, New York City for
         index, row in America_temp.iterrows():
             if row['City'] in ['New York City' ]:
                 temp_list.append(['New York', row['Year']])

         ## Manually add up the rest from online sources
         temp_list.append(['San Francisco', '14.6()'])
         temp_list.append(['Washinton DC', '14.6()'])
         temp_list.append(['Shenzhen', '22.9()'])
         temp_list.append(['Guangzhou', '22.2()'])
         temp_list.append(['Delhi', '29.2()'])

         len(temp_list)
```

```
Out[48]: 29
```

```
In [49]: ## convert temp_list into a dataframe
         temp_df = pd.DataFrame(temp_list)
         temp_df.columns = ['City', 'Temperature']

         ## drop out the F temp in the ()
         for index, row in temp_df.iterrows():
             row['Temperature'] = row['Temperature'].split('(')[0]

         ## convert temperature values into int
         temp_df['Temperature'] = pd.to_numeric(temp_df['Temperature'])

         temp_df.head()
```

Out[49]:

	City	Temperature
0	Beijing	12.9
1	Shanghai	16.7
2	Hong Kong	23.3
3	Kolkata	26.7
4	Mumbai	27.1

Rank Cities in Temperature Values

In [50]:
```python
## Rank Cities in Temperature values and sort values
temp_sorted = temp_df.sort_values('Temperature', ascending = False)
temp_sorted.head(10)
```

Out[50]:

	City	Temperature
28	Delhi	29.2
4	Mumbai	27.1
7	Singapore	27.0
3	Kolkata	26.7
5	Jakarta	26.7
2	Hong Kong	23.3
8	Taipei	23.0
26	Shenzhen	22.9
27	Guangzhou	22.2
19	Los Angeles	18.6

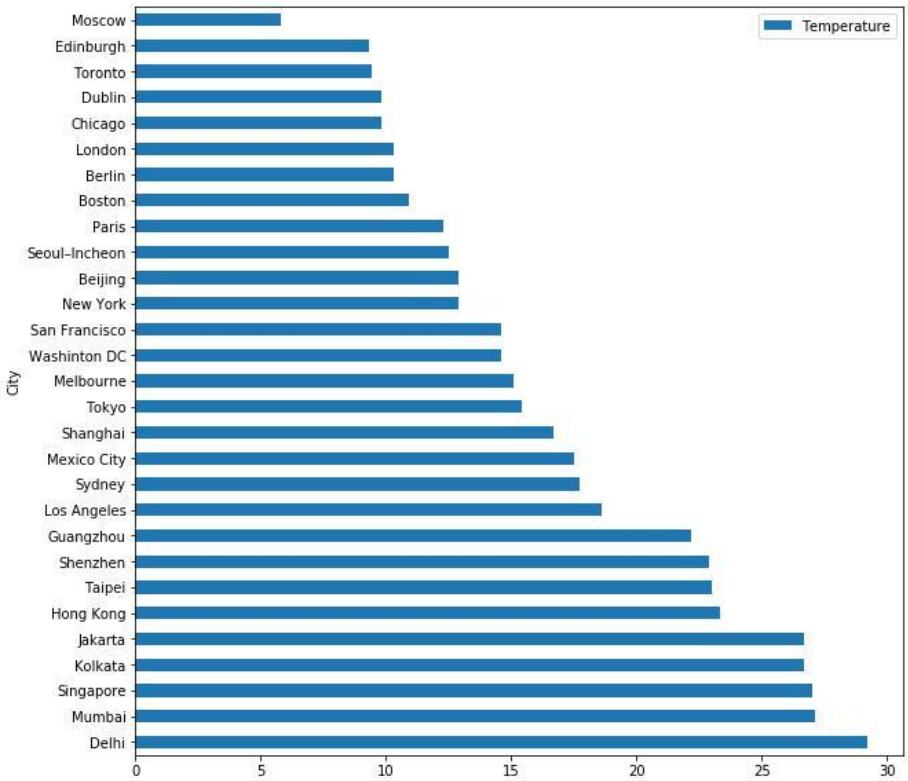
In [51]:
```python
## The last 10 cities in rank of tempature
temp_sorted.tail(10)
```

Out[51]:

	City	Temperature
22	Seoul–Incheon	12.5
9	Paris	12.3
17	Boston	10.9
10	Berlin	10.3
14	London	10.3
18	Chicago	9.8
11	Dublin	9.8
15	Toronto	9.4
13	Edinburgh	9.3
12	Moscow	5.8

In [52]:
```python
## Visualize the ranking with a bar chart
temp_visualize = temp_sorted
temp_visualize = temp_visualize.set_index('City')
temp_visualize.plot(kind = 'barh',
                    figsize = (10, 10))
```

Out[52]: <matplotlib.axes._subplots.AxesSubplot at 0x7f443c023f98>



Normalize Temperature data for modeling

In [53]:
```
## reset index for Temperature dataset
temp_sorted = temp_sorted.reset_index().drop(['index'], axis = 1)
```

In [54]:
```
## Standardize datasets
scaler_temp = preprocessing.StandardScaler()
temp_array = np.array(temp_sorted['Temperature'])
temp_normalized_array = preprocessing.normalize([temp_array])

## add the normalized gdp back into the dataframe
temp_column = pd.DataFrame(temp_normalized_array).transpose()
temp_sorted.insert(1, 'Normalized Temperature', temp_column)
temp_sorted.head()
```

Out[54]:

	City	Normalized Temperature	Temperature
0	Delhi	0.302161	29.2
1	Mumbai	0.280430	27.1
2	Singapore	0.279395	27.0
3	Kolkata	0.276291	26.7
4	Jakarta	0.276291	26.7

Merge All Features -- Venue Distribution, GDP & Climate

We will merge all data into one dataframe for convenience

In [55]:
```
## make sure the city names are the same
gdp_filtered_sorted1= gdp_filtered_sorted1.replace('Seoul-Incheon', 'Seoul')
gdp_filtered_sorted1 = gdp_filtered_sorted1.replace('Washington, DC', 'Washington')
temp_sorted = temp_sorted.replace('Washinton DC', 'Washington')
temp_sorted = temp_sorted.replace('Seoul-Incheon', 'Seoul')
world_grouped = world_grouped.replace('Seoul-Incheon', 'Seoul')
```

In [56]:
```
## merge GDP data
world_merged_cluster = world_grouped
world_merged_cluster = world_merged_cluster.join(gdp_filtered_sorted1.set_index('City'), on = 'City')

# merge Temperature data
world_merged_cluster = world_merged_cluster.join(temp_sorted.set_index('City'), on = 'City')
world_merged_cluster.head()
```

Out[56]:

	City	Adult Boutique	American Restaurant	Amphitheater	Aquarium	Arcade	Argentinian Restaurant	Art Gallery	Art Museum	Arts & Crafts Store	...	Yakitori Restaurant	Yoga Studio	Y Res
0	Beijing	0.0	0.00	0.00	0.0	0.00	0.0	0.02	0.00	0.00	...	0.0	0.00	
1	Berlin	0.0	0.00	0.00	0.0	0.00	0.0	0.01	0.00	0.02	...	0.0	0.00	
2	Boston	0.0	0.03	0.00	0.0	0.00	0.0	0.00	0.00	0.00	...	0.0	0.01	
3	Chicago	0.0	0.01	0.01	0.0	0.00	0.0	0.00	0.01	0.00	...	0.0	0.02	
4	Delhi	0.0	0.00	0.00	0.0	0.01	0.0	0.01	0.01	0.00	...	0.0	0.00	

5 rows × 349 columns

In [57]:
```
## Drop GDP and Temperature columns
world_merged_cluster = world_merged_cluster.drop(['GDP', 'Temperature'], axis = 1)
world_merged_cluster.head()
```

Out[57]:

	City	Adult Boutique	American Restaurant	Amphitheater	Aquarium	Arcade	Argentinian Restaurant	Art Gallery	Art Museum	Arts & Crafts Store	...	Winery	Xinjiang Restaurant	Res
0	Beijing	0.0	0.00	0.00	0.0	0.00	0.0	0.02	0.00	0.00	...	0.0	0.01	
1	Berlin	0.0	0.00	0.00	0.0	0.00	0.0	0.01	0.00	0.02	...	0.0	0.00	
2	Boston	0.0	0.03	0.00	0.0	0.00	0.0	0.00	0.00	0.00	...	0.0	0.00	
3	Chicago	0.0	0.01	0.01	0.0	0.00	0.0	0.00	0.01	0.00	...	0.0	0.00	
4	Delhi	0.0	0.00	0.00	0.0	0.01	0.0	0.01	0.01	0.00	...	0.0	0.00	

5 rows × 347 columns

Add weights to different factors

In order for different factors to be weighted diferently in the model, we will adjust the scale for normalized GDP and normalized temperature. We will
assign 2 times to normalized GDP and 2 times to normalized Temperature

In [58]:
```
## 10 times to normalized GDP
world_merged_cluster['Normalized GDP'] = world_merged_cluster['Normalized GDP']*1.5
world_merged_cluster['Normalized Temperature'] = world_merged_cluster['Normalized
Temperature']*1.5 world_merged_cluster
```

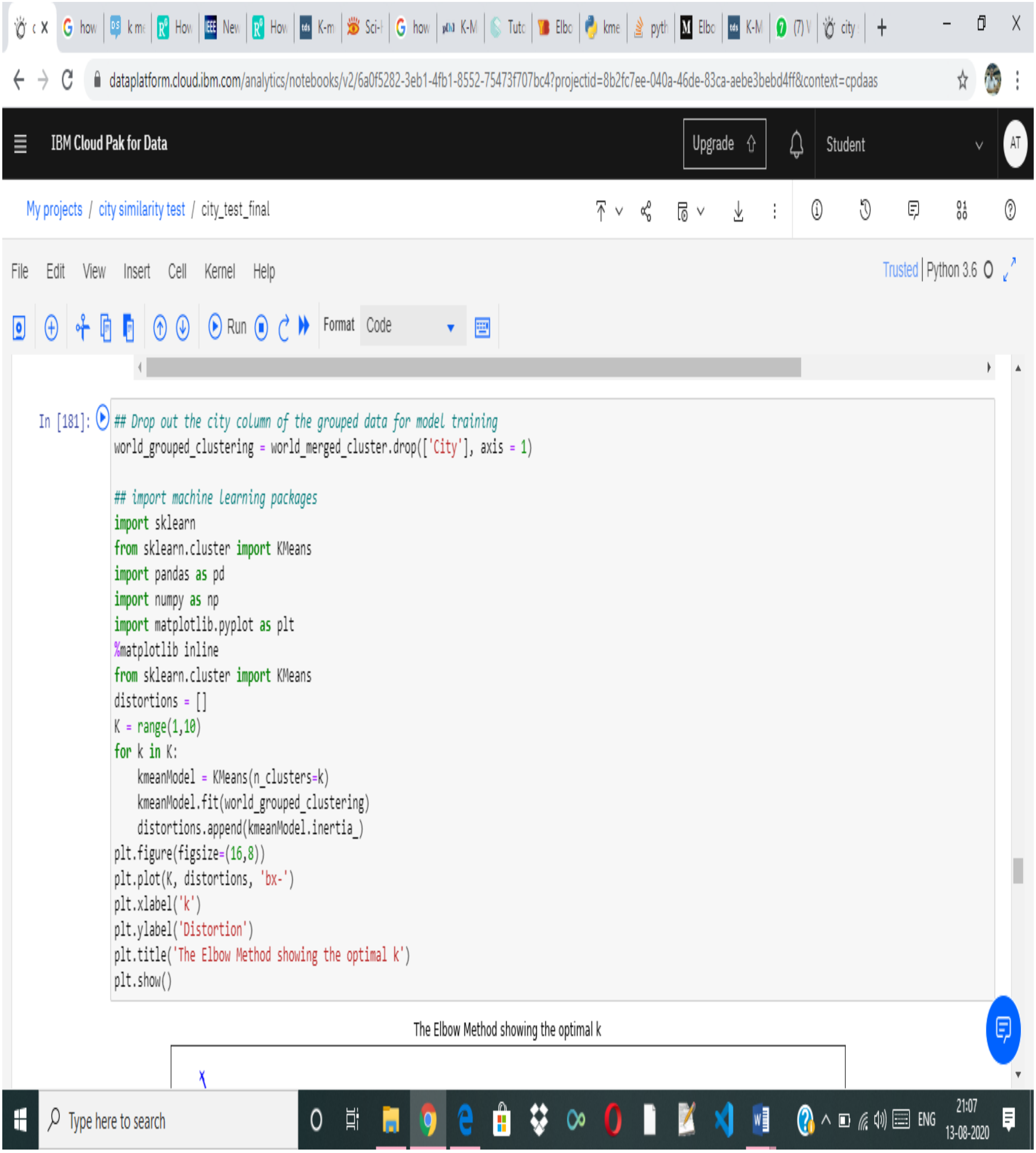
Out[58]:

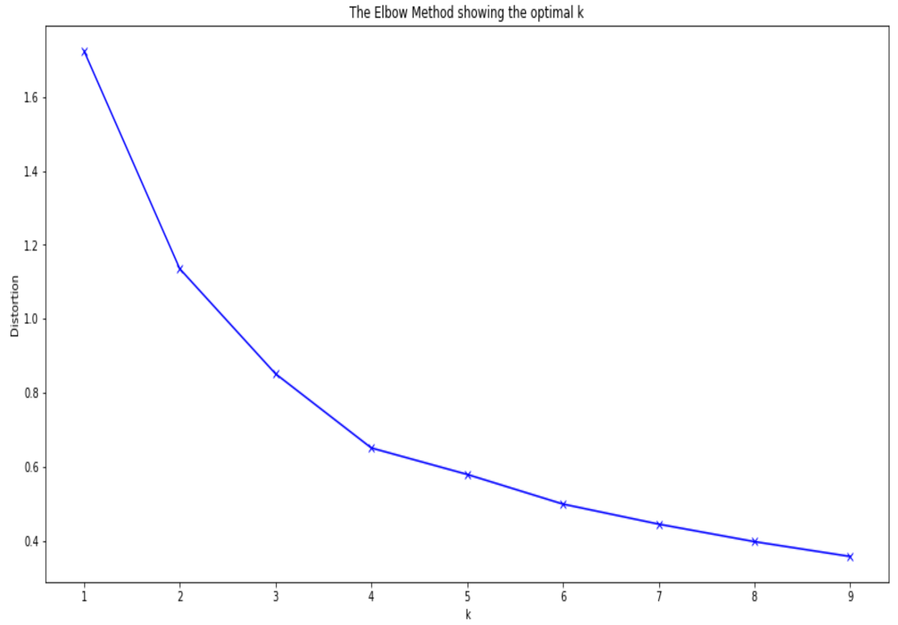
City	Adult Boutique	American Restaurant	Amphitheater	Aquarium	Arcade	Argentinian Restaurant	Art Gallery	Art Museum	Arts & Crafts Store	...	Winery	Xinjiang Restaurant

0	Beijing	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	...	0.00	0.01
1	Berlin	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.02	...	0.00	0.00
2	Boston	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.00	0.00
3	Chicago	0.00	0.01	0.01	0.00	0.00	0.00	0.00	0.01	0.00	...	0.00	0.00
4	Delhi	0.00	0.00	0.00	0.00	0.01	0.00	0.01	0.01	0.00	...	0.00	0.00
5	Dublin	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.00	0.00
6	Edinburgh	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.00	...	0.00	0.00
7	Guangzhou	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.00	0.00
8	Hong Kong	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	...	0.00	0.00
9	Jakarta	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.00	0.00
10	Kolkata	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.01	...	0.00	0.00
11	London	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.04	0.00	...	0.00	0.00
12	Los Angeles	0.00	0.03	0.00	0.00	0.00	0.00	0.04	0.01	0.00	...	0.01	0.00
13	Melbourne	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.00	...	0.00	0.00
14	Mexico City	0.00	0.01	0.00	0.00	0.00	0.00	0.01	0.05	0.00	...	0.00	0.00
15	Moscow	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.00	0.00
16	Mumbai	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	...	0.00	0.00
17	New York	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	...	0.00	0.00
18	Paris	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.04	0.00	...	0.00	0.00
19	San Francisco	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	...	0.00	0.00
20	Seoul	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.00	0.00
21	Shanghai	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.00	0.00
22	Shenzhen	0.00	0.01	0.00	0.00	0.00	0.00	0.01	0.00	0.00	...	0.00	0.00
23	Singapore	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.00	0.00	...	0.00	0.00
24	Sydney	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.01	0.00	...	0.00	0.00
25	Taipei	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.00	0.00
26	Tokyo	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.05	0.00	...	0.00	0.00
27	Toronto	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.00	0.00	...	0.00	0.00
28	Washington	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.07	0.00	...	0.00	0.00

29 rows × 347 columns

SETUP AND TRAIN THE MODEL USING KMEAN AND ELBOW METHOD






```
In [182]: ## Drop out the city column of the grouped data for model training
world_grouped_clustering = world_merged_cluster.drop(['City'], axis = 1)

## import machine learning packages
import sklearn
from sklearn.cluster import KMeans

## Create and fit a kmeans model
model_kmeans = KMeans(n_clusters = 7, random_state = 0)
model_kmeans.fit(world_grouped_clustering)

## Check out the labels
kmeans_labels = model_kmeans.labels_
kmeans_labels
```

```
Out[182]: array([2, 1, 1, 2, 5, 0, 0, 4, 4, 4, 5, 2, 6, 0, 1, 2, 5, 3, 2, 1, 6, 2,
           4, 4, 0, 4, 3, 1, 1], dtype=int32)
```

```
In [ ]:
```

```
In [183]: City_venue_sorted.insert(0, 'Cluster Labels', kmeans_labels)
City_venue_sorted.head()
```

Out[183]:

Update the Dataframe with Cluster Labels and Location Features

We will add the cluster labels to the dataframe -- City_venue_sorted

In [60]:
```
## Add Clustering labels
City_venue_sorted.insert(0, 'Cluster Labels', kmeans_labels)
City_venue_sorted.head()
```

Out[60]:

	Cluster Labels	City	1th Most Common Venue	2th Most Common Venue	3th 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	1	Beijing	Historic Site	Hotel	Park	Café	Shopping Mall	Brewery	Coffee Shop	Dumpling Restaurant	Temple	Cocktail Bar
1	3	Berlin	Park	Coffee Shop	Bookstore	Gourmet Shop	Ice Cream Shop	Monument / Landmark	Chocolate Shop	Hotel	Concert Hall	Bakery
2	3	Boston	Park	Bakery	Italian Restaurant	Seafood Restaurant	Salad Place	Gym	Gastropub	French Restaurant	Coffee Shop	Hotel
3	1	Chicago	Hotel	New American Restaurant	Coffee Shop	Park	Sandwich Place	Grocery Store	Donut Shop	Bar	Seafood Restaurant	Café
4	0	Delhi	Indian Restaurant	Hotel	Café	Lounge	Monument / Landmark	Deli / Bodega	Park	Bar	South Indian Restaurant	Coffee Shop

In [61]:
```
## Check out the shape of the City_venue_sorted
City_venue_sorted.shape
```

Out[61]: (29, 12)

In [62]:
```
## Check out the shape of the Toronto_selected
City_df.shape
```

Out[62]: (29, 4)

In [63]:
```
## Since the two dataframes have the same shape, we can merge them on the Postal Code
City_venue_sorted = City_venue_sorted.replace('Seoul-Incheon', 'Seoul')
world_merged = City_df
world_merged = world_merged.replace('Seoul-Incheon', 'Seoul')
world_merged = world_merged.join(City_venue_sorted.set_index('City'), on = 'City')

world_merged
```

Out[63]:

City	LatitudeLongitudeCountry	Cluster Labels	1th Most Common Venue	2th Most Common Venue	3th Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7 C

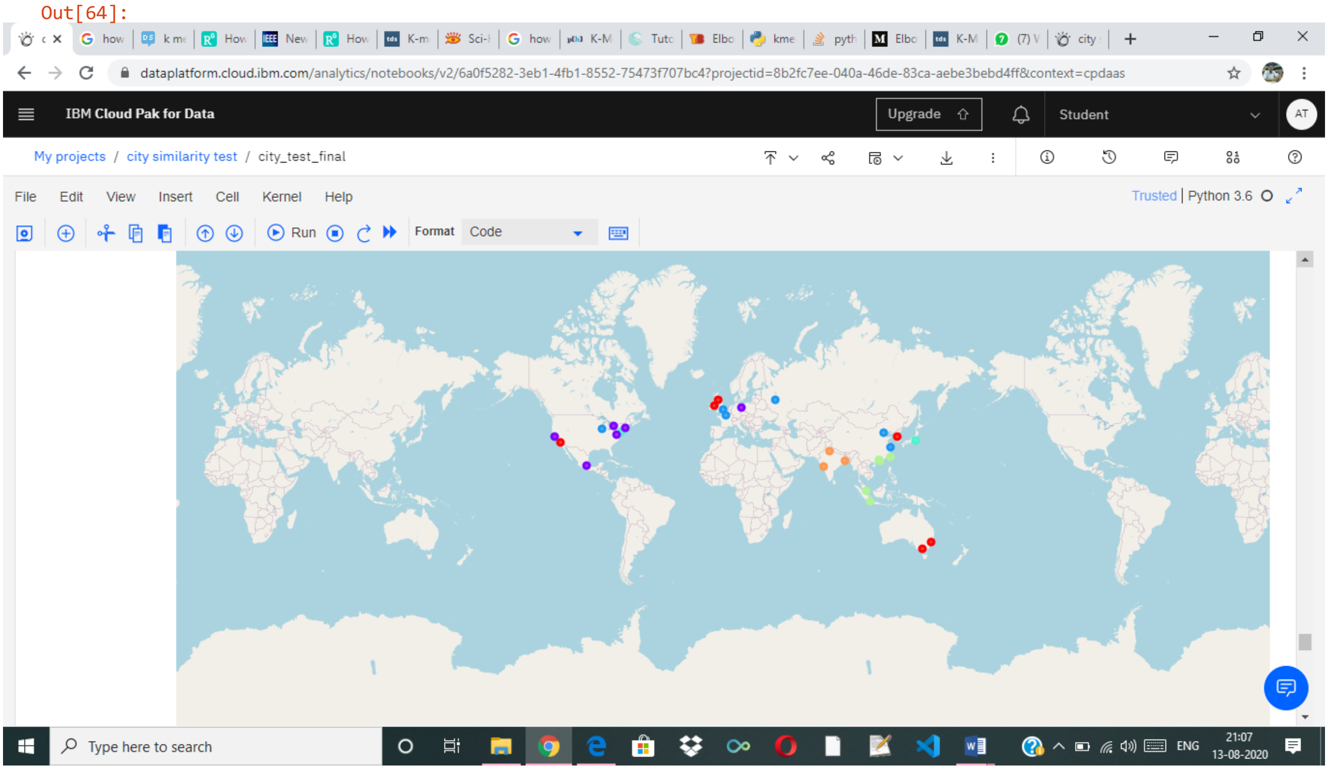
0	New York	40.712728	-74.006015	US
1	London	51.507322	-0.127647	UK
2	Edinburgh	55.953346	-3.188375	UK
3	Toronto	43.653482	-79.383935	Canada
4	Sydney	-33.854816	151.216454	Australia
5	Singapore	1.357107	103.819499	Singapore
6	Melbourne	-37.814218	144.963161	Australia
7	Hong Kong	22.279328	114.162813	China
8	Los Angeles	34.053691	-118.242767	US
9	Chicago	41.875562	-87.624421	US
10	Boston	42.360253	-71.058291	US
11	San Francisco	37.779026	-122.419906	US
12	Dublin	53.349764	-6.260273	Ireland
13	Washington	38.894893	-77.036553	US
14	Beijing	39.906217	116.391276	China
15	Shanghai	31.232276	121.469207	China
16	Guangzhou	23.130196	113.259294	China
17	Shenzhen	22.555454	114.054330	China
18	Mumbai	18.938771	72.835335	India
19	Tokyo	35.682839	139.759455	Japan
20	Seoul	37.440324	126.735400	South Korea
21	Moscow	55.479205	37.327330	Russia
22	Paris	48.856697	2.351462	France
23	Taipei	25.037520	121.563680	China
24	Berlin	52.517037	13.388860	Germany
25	Jakarta	-6.175394	106.827183	Indonesia
26	Mexico City	19.432630	-99.133178	Mexico
27	Delhi	28.651718	77.221939	India
28	Kolkata	22.545412	88.356775	India

Visualize the Cluster Results on the Map

We will visualize the map with different clusters results with different colors

```
In [64]: ## import necessary lib and packages
import matplotlib.cm as cm
import matplotlib.colors as colors

## Create map
map_clusters = folium.Map()

## set color scheme for the
clusters x = np.arange(6)
ys = [i + x + (i*x)**2 for i in range(6)]
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 = []
for lat, lon, city, country,cluster in zip(world_merged['Latitude'], world_merged['Longitude'],
world_merged[ 'City'], world_merged['Country'],world_merged['Cluster Labels']):
    label = folium.Popup(str(city) + ',' + str(country) + ' Cluster ' + str(cluster),
    parse_html=True) folium.CircleMarker(
        [lat, lon],
        radius=3,
        popup=label,
        color=rainbow[int(cluster)-1], fill=True,
        fill_color=rainbow[int(cluster)-1],
        fill_opacity=0.7).add_to(map_clusters)


map_clusters
```

Out[64]:



In [188]:

```
## Filter out the cluster 0 cities and change the column name to cluster 0
Cluster0 = pd.DataFrame(world_merged[world_merged['Cluster Labels'] == 0][['City', 'Country']])
Cluster0.columns = ['City', 'Country']
Cluster0 = pd.DataFrame(world_merged[world_merged['Cluster Labels'] == 0][['City', 'Country']]).replace('Seoul-Incheon','Seoul')
Cluster0.columns = ['City', 'Country']
Cluster0 = Cluster0.join(City_venue_sorted.set_index('City'), on = 'City')
Cluster0 = Cluster0.join(gdp_filtered_sorted1.set_index('City'), on = 'City')
Cluster0 = Cluster0.join(temp_sorted.set_index('City'), on = 'City')
Cluster0
```

Out[188]:

	City	Country	Cluster Labels	1th Most Common Venue	2th Most Common Venue	3th 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	Normalized GDP	GDP	Normalized Temperature	Tempera
2	Edinburgh	UK	0	Hotel	Café	Park	Coffee Shop	Pub	Beer Bar	Cocktail Bar	Bar	Restaurant	Mexican Restaurant	0.010178	32.5	0.093751	
4	Sydney	Australia	0	Park	Café	Theater	Hotel	Scenic Lookout	Bakery	Pub	Coffee Shop	Thai Restaurant	Cocktail Bar	0.069962	223.4	0.178428	
6	Melbourne	Australia	0	Café	Coffee Shop	Park	Cocktail Bar	Bar	Plaza	Whisky Bar	Performing Arts Venue	Burger Joint	Hotel	0.055869	178.4	0.152219	
12	Dublin	Ireland	0	Café	Pub	Coffee Shop	Park	Restaurant	Hotel	Plaza	Gastropub	Brewery	Museum	0.028216	90.1	0.098791	

This cluster has 4 cities, There Cafe,coffee shops,Hotel,Bakery and Pub the most famous.

```
Cluster1 = Cluster1.join(gdp_filtered_sorted1.set_index('City'), on = 'City')
Cluster1 = Cluster1.join(temp_sorted.set_index('City'), on = 'City')
Cluster1
```

Out[189]:

	City	Country	Cluster Labels	1th Most Common Venue	2th Most Common Venue	3th 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	Normalized GDP	GDP	Normalized Temperature	Te
3	Toronto	Canada	1	Park	Café	Coffee Shop	Bakery	Italian Restaurant	Restaurant	Sandwich Place	Farmers Market	French Restaurant	Japanese Restaurant	0.086528	276.3	0.094759	
10	Boston	US	1	Bakery	Park	Italian Restaurant	Hotel	Seafood Restaurant	Historic Site	Gym	Gastropub	Pizza Place	Ice Cream Shop	0.112772	360.1	0.109880	
11	San Francisco	US	1	Park	Coffee Shop	Bakery	Grocery Store	Pizza Place	Wine Bar	Art Museum	Sushi Restaurant	Marijuana Dispensary	Ice Cream Shop	0.103659	331.0	0.147178	
13	Washington	US	1	Hotel	Monument / Landmark	Art Museum	Ice Cream Shop	Coffee Shop	Park	American Restaurant	History Museum	Falafel Restaurant	Bakery	0.138483	442.2	0.147178	
24	Berlin	Germany	1	Coffee Shop	Park	Bookstore	Ice Cream Shop	Indie Movie Theater	Café	Wine Bar	Concert Hall	Monument / Landmark	Bakery	0.049387	157.7	0.103831	
26	Mexico City	Mexico	1	Ice Cream Shop	Mexican Restaurant	Bakery	Park	Art Museum	Asian Restaurant	Seafood Restaurant	Museum	Taco Place	Hotel	0.126394	403.6	0.176412	

This cluster has 7 cities, There coffee shops,Bakery,Pub and Cafe are the most famous.

```
Cluster2 = Cluster2.join(temp_sorted.set_index('City'), on = 'City')
Cluster2
```

Out[190]:

	City	Country	Cluster Labels	1th Most Common Venue	2th Most Common Venue	3th 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	Normalized GDP	GDP	Normalized Temperature	Tempe
1	London	UK	2	Hotel	Art Museum	Park	Bookstore	Coffee Shop	Department Store	Boutique	Cocktail Bar	Gym / Fitness Center	Café	0.261714	835.7	0.103831	
9	Chicago	US	2	Hotel	Park	Yoga Studio	Grocery Store	Coffee Shop	Theater	Italian Restaurant	New American Restaurant	Gym	Seafood Restaurant	0.176376	563.2	0.098791	
14	Beijing	China	2	Hotel	Historic Site	Park	Café	Shopping Mall	Coffee Shop	Dumpling Restaurant	Brewery	Bookstore	Temple	0.158494	506.1	0.130041	
15	Shanghai	China	2	Hotel	Coffee Shop	Shopping Mall	Dumpling Restaurant	Lounge	Park	Bakery	Italian Restaurant	French Restaurant	Cocktail Bar	0.186022	594.0	0.168348	
21	Moscow	Russia	2	Hotel	Pizza Place	Yoga Studio	Park	Art Gallery	Theater	Coffee Shop	Plaza	Road	Cocktail Bar	0.173276	553.3	0.058468	
22	Paris	France	2	Plaza	Wine Bar	Hotel	Bookstore	Cocktail Bar	Art Museum	Italian Restaurant	Historic Site	Garden	Bakery	0.223946	715.1	0.123993	

These 6 cities are from all across the world. One common feature among them is that theaters are pretty popular in these cities. London, Chicago, and Moscow are four out of three cities among all with theaters in the top 3 most common venues, and they all have developed arts and entertainment industries. Besides, Hotels are popular in these cities and all 6 cities all have close GDP, which is nearly 2 times higher than that of cluster 0 cities. However, their climates are pretty different.

have close GDP, which is nearly 2 times higher than that of cluster 0 cities. However, their climates are pretty different.

In [191]:

```
## Filter out the cluster 3 cities and change the column name to cluster 3
Cluster3 = pd.DataFrame(world_merged[world_merged['Cluster Labels'] == 3][['City', 'Country']])
Cluster3.columns = ['City', 'Country']
Cluster3 = pd.DataFrame(world_merged[world_merged['Cluster Labels'] == 3][['City', 'Country']]).replace('Seoul-Incheon','Seoul')
Cluster3.columns = ['City', 'Country']
Cluster3 = Cluster3.join(City_venue_sorted.set_index('City'), on = 'City')
Cluster3 = Cluster3.join(gdp_filtered_sorted1.set_index('City'), on = 'City')
Cluster3 = Cluster3.join(temp_sorted.set_index('City'), on = 'City')
Cluster3
```

Out[191]:

	City	Country	Cluster Labels	1th Most Common Venue	2th Most Common Venue	3th 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	Normalized GDP	GDP	Normalized Temperature	Temperatu
0	New York	US	3	Park	Ice Cream Shop	Bookstore	Bakery	Scenic Lookout	Seafood Restaurant	Thai Restaurant	French Restaurant	Sandwich Place	Cycle Studio	0.439374	1403.0	0.130041	12
19	Tokyo	Japan	3	Hotel	Ramen Restaurant	Wagashi Place	Sake Bar	Art Museum	Sushi Restaurant	Tonkatsu Restaurant	Coffee Shop	BBQ Joint	Chinese Restaurant	0.506392	1617.0	0.155243	15

In this cluster has only 2 cities .These two cities have very different culture so setting business their will be having individual risks

In [192]:

```
## Filter out the cluster 4 cities and change the column name to cluster 4
Cluster4 = pd.DataFrame(world_merged[world_merged['Cluster Labels'] == 4][['City', 'Country']])
```

In [193]:

```
## Filter out the cluster 5 cities and change the column name to cluster 5
Cluster5 = pd.DataFrame(world_merged[world_merged['Cluster Labels'] == 5][['City', 'Country']])
Cluster5.columns = ['City', 'Country']
Cluster5 = pd.DataFrame(world_merged[world_merged['Cluster Labels'] == 5][['City', 'Country']]).replace('Seoul-Incheon','Seoul')
Cluster5 = Cluster5.join(City_venue_sorted.set_index('City'), on = 'City')
Cluster5 = Cluster5.join(gdp_filtered_sorted1.set_index('City'), on = 'City')
Cluster5 = Cluster5.join(temp_sorted.set_index('City'), on = 'City')
Cluster5
```

Out[193]:

	City	Country	Cluster Labels	1th Most Common Venue	2th Most Common Venue	3th 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	Normalized GDP	GDP	Normalized Temperature	Tempe
18	Mumbai	India	5	Indian Restaurant	Hotel	Scenic Lookout	Dessert Shop	Café	Coffee Shop	Pizza Place	Restaurant	Deli / Bodega	Park	0.047257	150.90	0.273187	
27	Delhi	India	5	Indian Restaurant	Hotel	Café	Bar	BBQ Joint	Lounge	Coffee Shop	Deli / Bodega	Tibetan Restaurant	Fast Food Restaurant	0.091946	293.60	0.294356	
28	Kolkata	India	5	Café	Chinese Restaurant	Indian Restaurant	Hotel	Shopping Mall	Mughlai Restaurant	Multiplex	Indian Sweet Shop	Italian Restaurant	Dhaba	0.047088	150.36	0.269155	

These 3 cities are all located in the southern Asian areas and in India with similar climates and temperatures. Their GDP are close too and are lower than those of the cluster 1 cities. 6 of them have hotel as the most common venue and coffee shops/cafe are very popular too. This shows that tourism might be an essential source of income for these cities

In [194]:

```
## Filter out the cluster 6 cities and change the column name to cluster 6
Cluster6 = pd.DataFrame(world_merged[world_merged['Cluster Labels'] == 6][['City', 'Country']])
Cluster6.columns = ['City', 'Country']
Cluster6 = pd.DataFrame(world_merged[world_merged['Cluster Labels'] == 6][['City', 'Country']]).replace('Seoul-Incheon','Seoul')
Cluster6 = Cluster6.join(City_venue_sorted.set_index('City'), on = 'City')
Cluster6 = Cluster6.join(gdp_filtered_sorted1.set_index('City'), on = 'City')
Cluster6 = Cluster6.join(temp_sorted.set_index('City'), on = 'City')
Cluster6
```

Out[194]:

	City	Country	Cluster Labels	1th Most Common Venue	2th Most Common Venue	3th 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	Normalized GDP	GDP	Normalized Temperature	Temperatur
8	Los Angeles	US	6	Coffee Shop	Taco Place	Korean Restaurant	Bakery	Ice Cream Shop	American Restaurant	Food Truck	Art Gallery	Theater	Park	0.269481	860.5	0.187501	18.
20	Seoul	South Korea	6	Coffee Shop	Korean Restaurant	Park	Market	Multiplex	Golf Course	Fast Food Restaurant	Bakery	BBQ Joint	Outlet Mall	0.264909	845.9	0.126009	12.

In this cluster has only two cities one from US and other from korean country having totally different lifestyles.There coffee shops ,Bakery and Fast food Restaurant are famous and can be established in order to gain profit if qualities are much better than others.

In [ ]:

ACCURACY OF THE MODEL

Kmeans clustering is one of the most popular clustering algorithms and usually the first thing practitioners apply when solving clustering tasks to get an idea of the structure of the dataset. The goal of kmeans is to group data points into distinct non-overlapping subgroups.
Unlike supervised learning, clustering is considered an unsupervised learning method since we don't have the ground truth to compare the output of the clustering algorithm to the true labels to evaluate its performance. We only want to try to investigate the structure of the data by grouping the data points into distinct subgroups.
In this model we have used kmean algorithm which is the best one and number of cluster is defined by Elbow Method here.

CONCLUSION

In this assignment we have built up a clustering model to segment the major big cities into different groups. The result could be a valuable reference to the Board of Directors when they are making decisions on their business expansions into these cities. This results is straight-forward and takes different factors into account. However, there is also room for improvement, as there are a lot of features that influence the similarity between two cities and more variables could be included for higher accuracy of the clustering results.

REFERENCES

1. J. Cranshaw, R. Schwartz, J. Hong, and N. Sadeh. The livehoods project: Utilizing social media to understand the dynamics of a city. ICWSM 2012, 2012.

2. Daniel Preo,tiuc-Pietro,Justin Cranshaw,Tae Yano,Exploring venue based city to city measures,2013

3. https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a

4. New efficient clustering quality indexes Jean-Charles Lamirel, Nicolas Dugu´e, Pascal Cuxac

5. https://towardsdatascience.com/

PROJECT LINK(GITHUB)

https://github.com/akashtri19298/City-Similarity-Test