❖ Description of Data

➢ Data requirements

In order to build the descriptive model, two datasets will be needed.

1) The geographical location of each neighborhood to visualize and segment the boroughs on the map.

2) The venues within each neighborhood (along with its category) to explore the characteristics of each borough.

➢ Data collection

1) The geographical location of each neighbor is available from the website. The link for the neighborhood data is: https://geo.nyu.edu/catalog/nyu_2451_34572 [1]. We can also download the dataset from the following address: https://cocl.us/new_york_dataset, as shown in Figure 1.

```
[3]: !wget -q -O 'newyork_data.json' https://cocl.us/new_york_dataset
```

Figure 1. Downloading the geographical data.

After opening the json file, we can find that the neighborhood dataset contains both the latitude and longitude data, as well as what borough it belongs to, as shown in Figure 2.

```
[4]: with open('newyork_data.json') as json_data:
         newyork_data = json.load(json_data)
```

Take *features* from the dataset.

```
[5]: neighborhoods_data = newyork_data['features']
     neighborhoods_data[0]
```

```
[5]: {'type': 'Feature',
      'id': 'nyu_2451_34572.1',
      'geometry': {'type': 'Point',
       'coordinates': [-73.84720052054902, 40.89470517661]},
      'geometry_name': 'geom',
      'properties': {'name': 'Wakefield',
       'stacked': 1,
       'annoline1': 'Wakefield',
       'annoline2': None,
       'annoline3': None,
       'annoangle': 0.0,
       'borough': 'Bronx',
       'bbox': [-73.84720052054902,
        40.89470517661,
        -73.84720052054902,
        40.89470517661]}}
```

Figure 2. Explore the New York neighborhood dataset.

Then, an empty Pandas [2] dataframe is created. We loop through the data and fill the dataframe one row at a time, as shown in Figure 3.

```
[6]: # define the dataframe columns
     column_names = ['Borough', 'Neighborhood', 'Latitude', 'Longitude']

     # instantiate the dataframe
     neighborhoods = pd.DataFrame(columns=column_names)
     neighborhoods
```

```
[6]:   Borough  Neighborhood  Latitude  Longitude
```

```
[7]: for data in neighborhoods_data:
         borough = data['properties']['borough']
         neighborhood_name = data['properties']['name']
         neighborhood_latlon = data['geometry']['coordinates']
         neighborhood_lat = neighborhood_latlon[1]
         neighborhood_lon = neighborhood_latlon[0]

         neighborhoods = neighborhoods.append({'Borough': borough,
                                               'Neighborhood': neighborhood_name,
                                               'Latitude': neighborhood_lat,
                                               'Longitude': neighborhood_lon}, ignore_index=True)
```

```
[8]: print('The dataframe has {} boroughs and {} neighborhoods.'.format(len(neighborhoods['Borough'].unique()),
             neighborhoods.shape[0]))
```

```
The dataframe has 5 boroughs and 306 neighborhoods.
```

Figure 3. Load the geographical data into the pandas dataframe.

Consequently, the resulted dataframe is given as follows:

```
neighborhoods.head()
```

[7]:

	Borough	Neighborhood	Latitude	Longitude
0	Bronx	Wakefield	40.894705	-73.847201
1	Bronx	Co-op City	40.874294	-73.829939
2	Bronx	Eastchester	40.887556	-73.827806
3	Bronx	Fieldston	40.895437	-73.905643
4	Bronx	Riverdale	40.890834	-73.912585

Figure 4. New York neighborhood dataframe.

Also, the geographical location of each borough is obtained by executing the following code:

[9]:
```
address = 'New York City, NY'
geolocator = Nominatim(user_agent="ny_explorer")
location = geolocator.geocode(address)
latitude = location.latitude
longitude = location.longitude
```

10]:
```
manhattan_data = neighborhoods[neighborhoods['Borough'] == 'Manhattan'].reset_index(drop=True)
brooklyn_data = neighborhoods[neighborhoods['Borough'] == 'Brooklyn'].reset_index(drop=True)
queens_data = neighborhoods[neighborhoods['Borough'] == 'Queens'].reset_index(drop=True)
bronx_data = neighborhoods[neighborhoods['Borough'] == 'Bronx'].reset_index(drop=True)
staten_island_data = neighborhoods[neighborhoods['Borough'] == 'Staten Island'].reset_index(drop=True)
```

Figure 5. Retrieve the geographical location of each borough.

2) Next, the venue data within each neighborhood is obtained from Foursquare API [3]. Foursquare is a technology company that build a massive dataset of accurate location data, and its API has been used by over 100,000 developers. First, we load the personal client_id and client_sceret data from a saved json file, as shown in Figure 6.

[19]:
```
secrets = json.load(open('credential.json'))
CLIENT_ID = secrets['CLIENT_ID']
CLIENT_SECRET = secrets['CLIENT_SECRET']
VERSION = '20200101'
```

Figure 6. Load client id and secret from the json file.

Then we set the maximum number of venues returned by Foursquare API as 100.

Limit of number of venues returned by Foursquare API

[20]:
```
LIMIT = 100
```

Figure 7. Limit the number of venues returned from API.

We define a function to extract venue category, as shown in Figure 8.

Define function to extract the category of the venue.

```
[21]: def get_category_type(row):
          try:
              categories_list = row['categories']
          except:
              categories_list = row['venue.categories']

          if len(categories_list) == 0:
              return None
          else:
              return categories_list[0]['name']
```

Figure 8. Define a function to extract the venue category.

After that, a function is created to get the nearby venues of each neighborhood that are within 500-meter radius, as shown in Figure 9.

```
def getNearbyVenues(names, latitudes, longitudes, radius=500):

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

        # create the API request URL
        url = '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).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 = ['Neighborhood',
                  'Neighborhood Latitude',
                  'Neighborhood Longitude',
                  'Venue',
                  'Venue Latitude',
                  'Venue Longitude',
                  'Venue Category']

    return(nearby_venues)
```

Figure 9. Define a function to get the nearby venues of each neighborhood.

We get the nearby venues' information of each borough and save them in the dataframe.

```
[23]: manhattan_venues = getNearbyVenues(names=manhattan_data['Neighborhood'],
                                         latitudes=manhattan_data['Latitude'],
                                         longitudes=manhattan_data['Longitude']
                                        )

[24]: brooklyn_venues = getNearbyVenues(names=brooklyn_data['Neighborhood'],
                                        latitudes=brooklyn_data['Latitude'],
                                        longitudes=brooklyn_data['Longitude']
                                       )

[25]: queens_venues = getNearbyVenues(names=queens_data['Neighborhood'],
                                      latitudes=queens_data['Latitude'],
                                      longitudes=queens_data['Longitude']
                                     )

[26]: bronx_venues = getNearbyVenues(names=bronx_data['Neighborhood'],
                                     latitudes=bronx_data['Latitude'],
                                     longitudes=bronx_data['Longitude']
                                    )

[27]: staten_island_venues = getNearbyVenues(names=staten_island_data['Neighborhood'],
                                             latitudes=staten_island_data['Latitude'],
                                             longitudes=staten_island_data['Longitude']
                                            )
```

Figure 10. Get the nearby venue data of each borough.

Finally, we check the size of the resulting dataframe and print how many unique categories can be curated from all the returned venues, as shown in

```
[28]: print('The size of manhattan dataframe is {}.'.format(manhattan_venues.shape))
      print('The size of brooklyn dataframe is {}.'.format(brooklyn_venues.shape))
      print('The size of queens dataframe is {}.'.format(queens_venues.shape))
      print('The size of bronx dataframe is {}.'.format(bronx_venues.shape))
      print('The size of staten island dataframe is {}.'.format(staten_island_venues.shape))

      The size of manhattan dataframe is (3309, 7).
      The size of brooklyn dataframe is (2788, 7).
      The size of queens dataframe is (2119, 7).
      The size of bronx dataframe is (1215, 7).
      The size of staten island dataframe is (827, 7).

[29]: print('There are {} uniques categories in manhattan.'.format(len(manhattan_venues['Venue Category'].unique())))
      print('There are {} uniques categories in brooklyn.'.format(len(brooklyn_venues['Venue Category'].unique())))
      print('There are {} uniques categories in queens.'.format(len(queens_venues['Venue Category'].unique())))
      print('There are {} uniques categories in bronx.'.format(len(bronx_venues['Venue Category'].unique())))
      print('There are {} uniques categories in staten island.'.format(len(staten_island_venues['Venue Category'].unique())))

      There are 337 uniques categories in manhattan.
      There are 287 uniques categories in brooklyn.
      There are 271 uniques categories in queens.
      There are 168 uniques categories in bronx.
      There are 184 uniques categories in staten island.
```

Figure 11. Check the returned data.

❖ References

[1] New York Department of City Planning, "2014 New York City Neighborhood Names," [Online] available: https://geo.nyu.edu/catalog/nyu_2451_34572.

[2] Wes McKinney, “Data Structures for Statistical Computing in Python,” Proceedings of the 9th Python in Science Conference, pp. 51-56, 2010.

[3] Foursquare, [Online] available: https://developer.foursquare.com/.