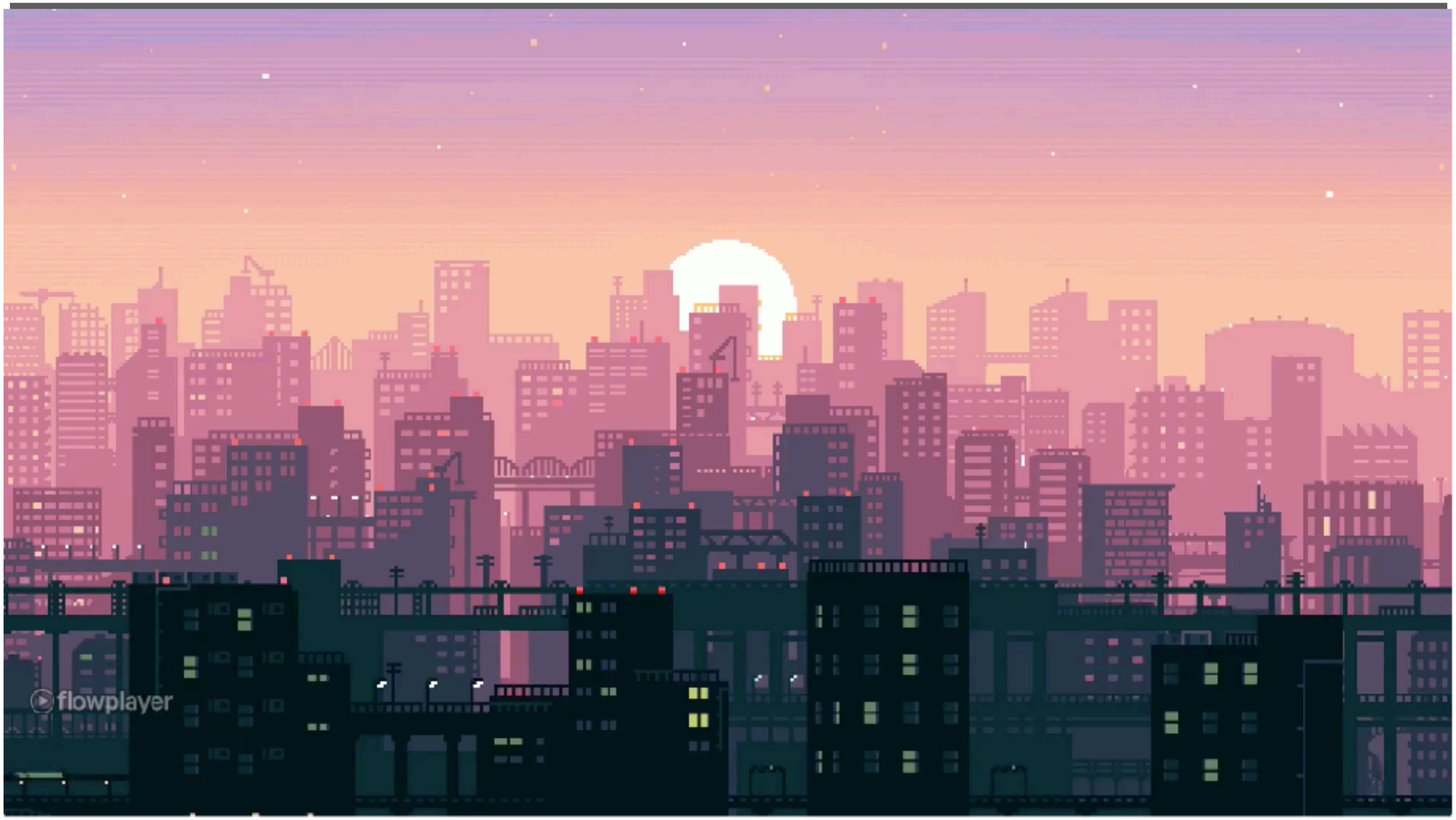
Corner Finder



Introduction

Find a similar neighborhood between two cities

According to the Census around 35.5 million Americans move each year to new cities.

The most stressful part of moving is finding a new neighborhood you can call home. The goal of this project is to find your corner every where you move or travel.

Finding your corner is targeted to people that are relocating to a new place and want to find the environment they already love in another city.

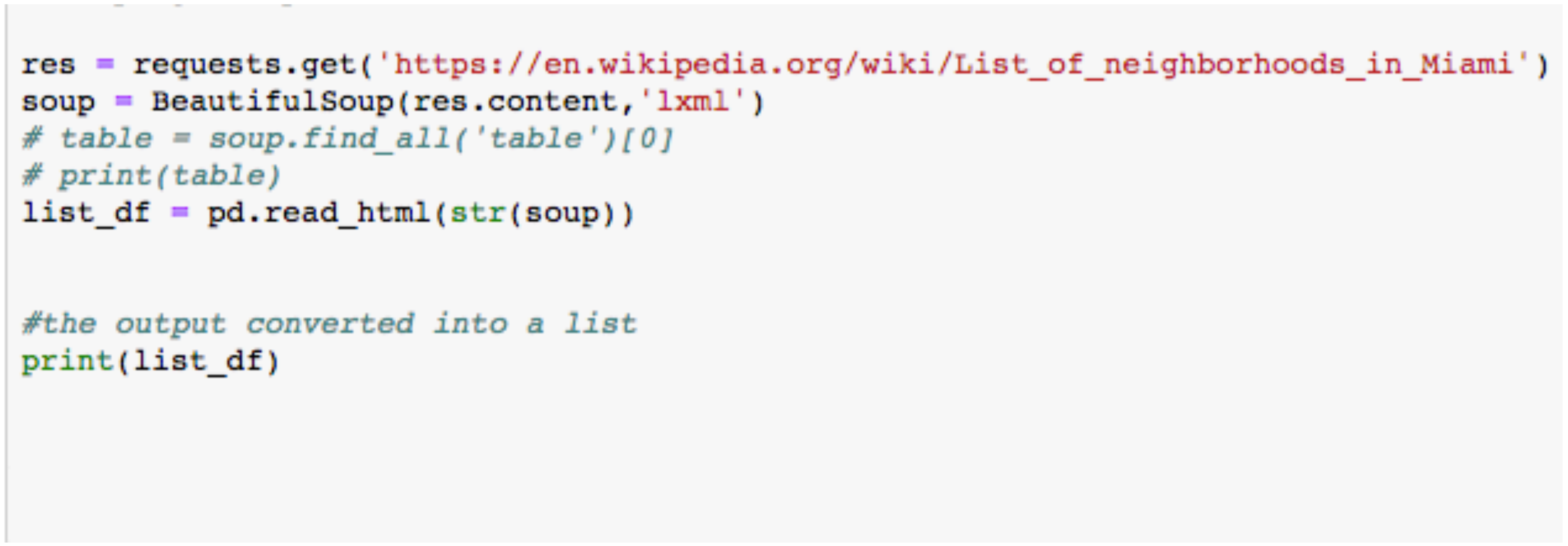
Similar neighborhoods in Miami and San Francisco

Data Acquisition and Cleaning

- The first one is comprised of acquiring and cleaning data to create a data frame with Miami's and San Francisco's neighborhoods and coordinates.
- The second phase is connecting to the Foursquare API to obtained the venues data in the neighborhoods.

Phase I

- The Miami neighborhoods data was scraped from wikipedia, obtaining the neighborhood names populations and coordinates for 25 neighborhoods.

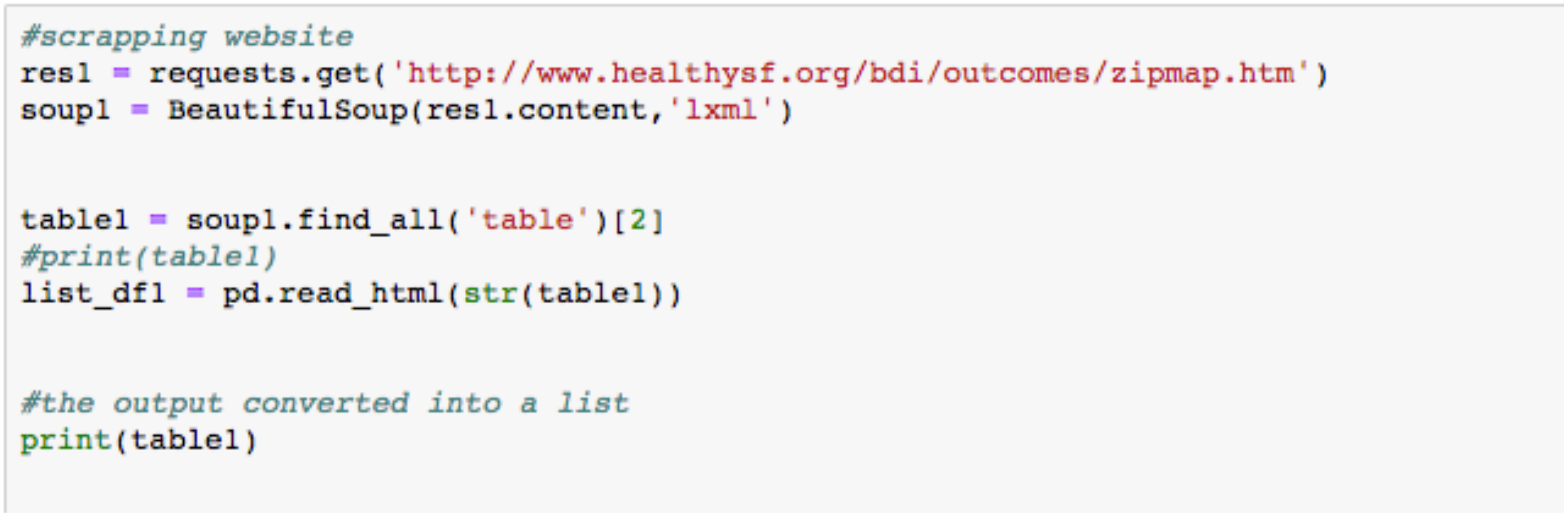


```
res = requests.get('https://en.wikipedia.org/wiki/List_of_neighborhoods_in_Miami')
soup = BeautifulSoup(res.content,'lxml')
# table = soup.find_all('table')[0]
# print(table)
list_df = pd.read_html(str(soup))


#the output converted into a list
print(list_df)
```

Data Acquisition and Cleaning (cont)

Phase I

- The SanFrancisco neighborhoods data was first scraped from the *San Francisco Burden of Disease & Injury Study* website, that was last updated on 2004. This website had the neighborhood information and corresponding zip codes. However, the website did not have the coordinates of each neighborhood. To obtained this coordinates the uszipcode packaged was utilized. The coordinates provided by the uszipcode packaged was incorrect for the 'Outer Richmond' neighborhood, the correct coordinates were obtained from wikipedia and changed in the data frame.
-

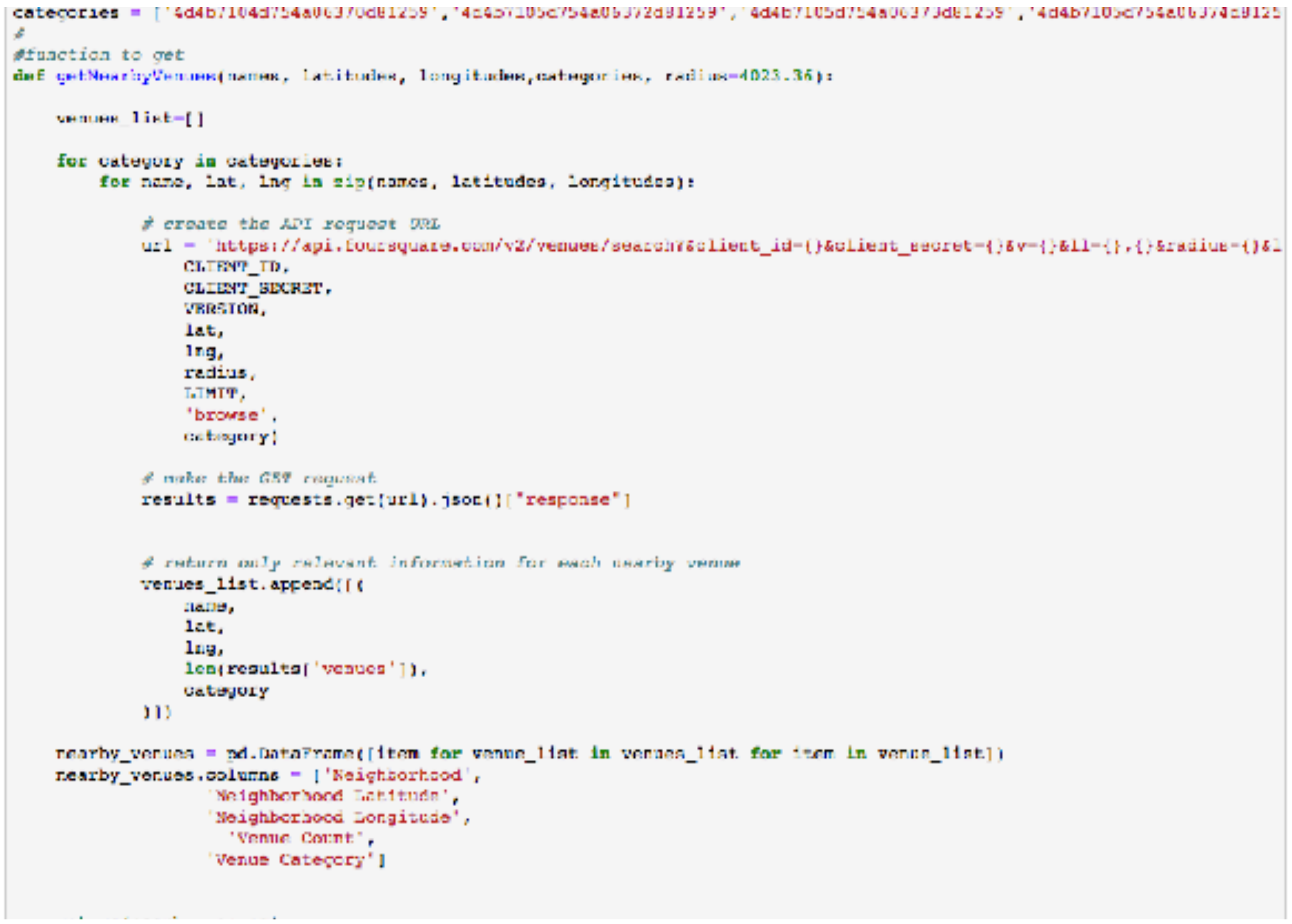


```
#scrapping website
res1 = requests.get('http://www.healthysf.org/bdi/outcomes/zipmap.htm')
soup1 = BeautifulSoup(res1.content,'lxml')


table1 = soup1.find_all('table')[2]
#print(table1)
list_df1 = pd.read_html(str(table1))


#the output converted into a list
print(table1)
```

Foursquare API

Phase II

- The Foursquare API offers a global database of venues data with more than 35 different end points.
- Request:

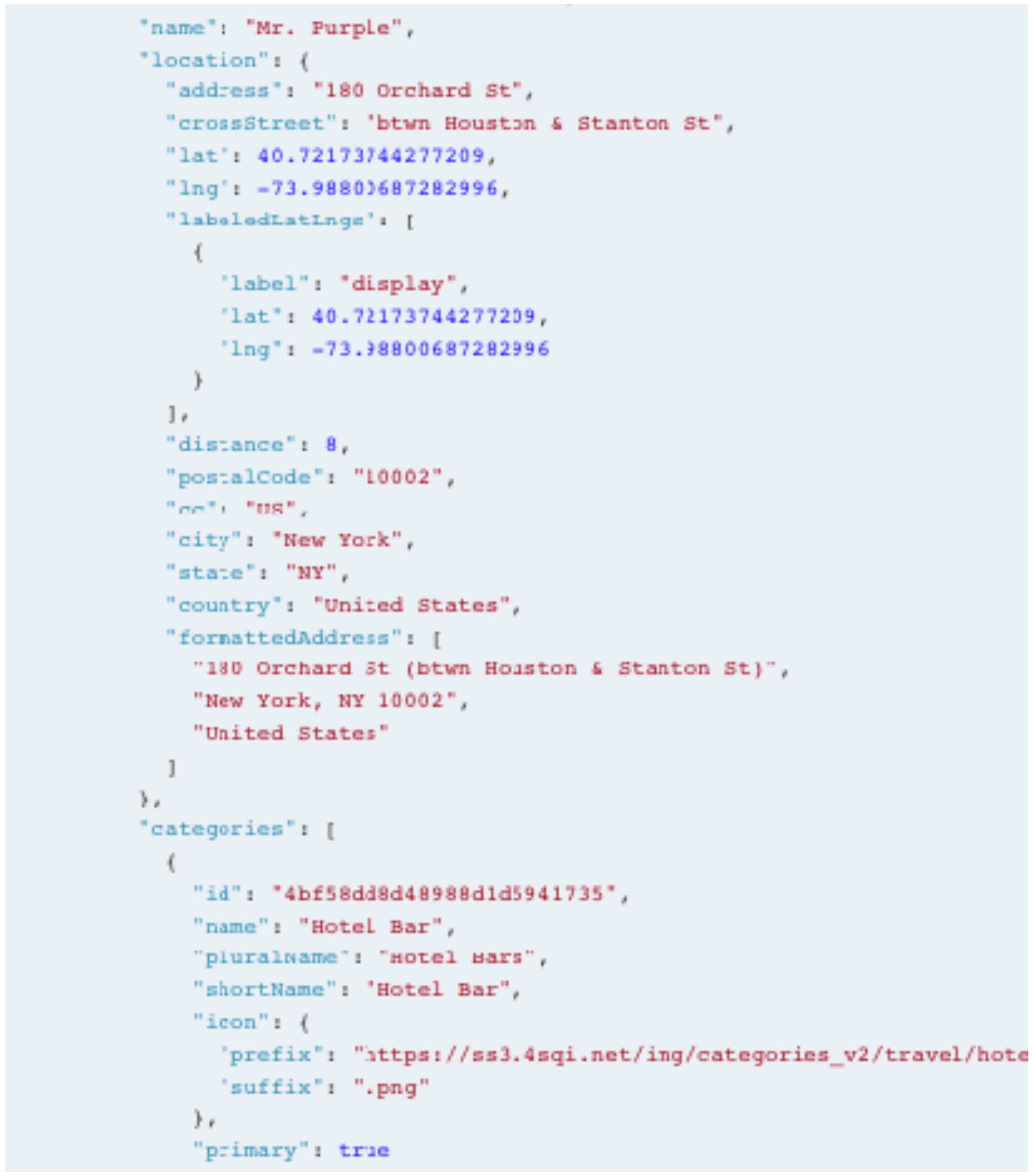


```
categories = ['4d4b7104d754a06370d81259','4d4b7105d754a06372d81259','4d4b7105d754a06373d81259','4d4b7105d754a06374d8125
#
#function to get
def getNearbyVenues(names, latitudes, longitudes,categories, radius=4023.36):

    venues_list=[]

    for category in categories:
        for name, lat, lng in zip(names, latitudes, longitudes):

            # create the API request URL
            url = 'https://api.foursquare.com/v2/venues/search?&client_id={}&client_secret={}&v={}&ll={},{}&radius={}&l
                CLIENT_ID,
                CLIENT_SECRET,
                VERSION,
                lat,
                lng,
                radius,
                LIMIT,
                'browse',
                category)

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


            # return only relevant information for each nearby venue
            venues_list.append([(
                name,
                lat,
                lng,
                len(results['venues']),
                category
            )])

    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 Count',
                  'Venue Category']
```

Foursquare API (cont)

Phase II

- Response:

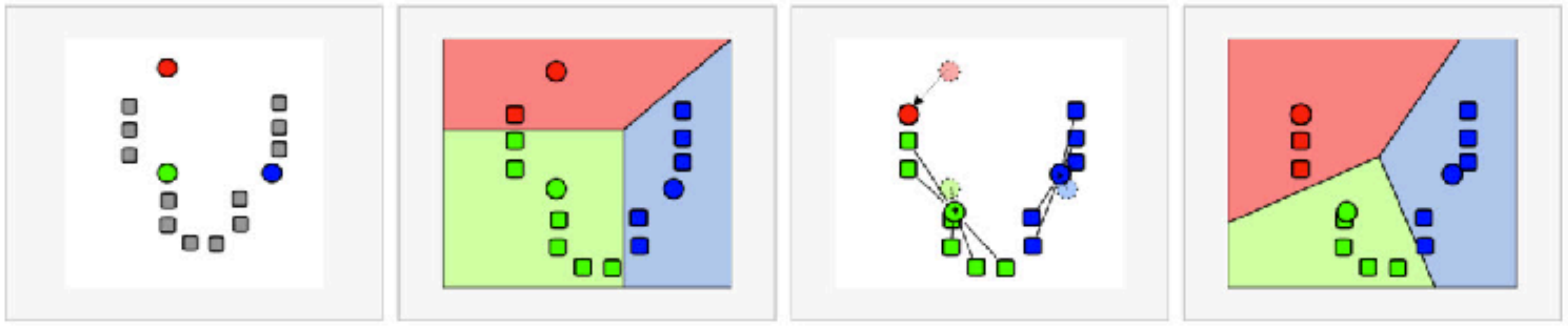


```
"name": "Mr. Purple",
"location": {
  "address": "180 Orchard St",
  "crossStreet": "btwn Houston & Stanton St",
  "lat": 40.72173744277209,
  "lng": -73.98800687282996,
  "labeledLatLngs": [
    {
      "label": "display",
      "lat": 40.72173744277209,
      "lng": -73.98800687282996
    }
  ],
  "distance": 8,
  "postalCode": "10002",
  "cc": "US",
  "city": "New York",
  "state": "NY",
  "country": "United States",
  "formattedAddress": [
    "180 Orchard St (btwn Houston & Stanton St)",
    "New York, NY 10002",
    "United States"
  ]
},
"categories": [
  {
    "id": "4bf58dd8d48988d1d5941735",
    "name": "Hotel Bar",
    "pluralName": "Hotel Bars",
    "shortName": "Hotel Bar",
    "icon": {
      "prefix": "https://ss3.4sqi.net/img/categories_v2/travel/hote
      "suffix": ".png"
    },
    "primary": true
```

Methodology

- The goal of this project is to compare neighborhoods in Miami and San Francisco and grouped them by similarity. The similarity are evaluated by the amount of venues in each category from the FourSquare API. The number of venues are then used to create a k-mean cluster. The k-mean cluster calculated with Python and the Scikit-learn library.
- The k-mean cluster algorithm was chosen because of it ease of understanding, implementation and runtime. It also aligns with the goal of the project effectively grouping neighborhoods based on its similarities. The k-means cluster algorithm groups similar data into *k* clusters. The goal of this algorithm is to put individual in the cluster closest to the cluster's mean value (Sterling etl, 2018). This is done by selecting k initial centroids at random and iterating trough: assigning the individual to the closest cluster minimizing the inter cluster Euclidian distance and maximizing the outer cluster distance until the cluster converges.
-

Demonstration of the standard algorithm



1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).

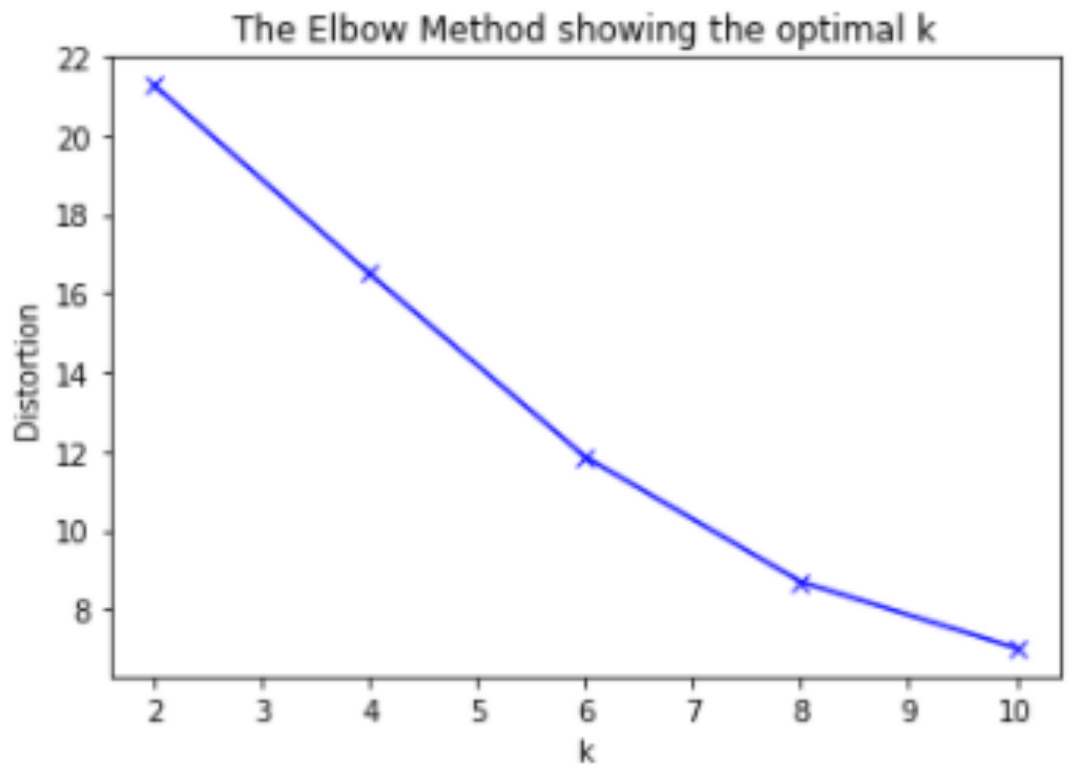
2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

3. The centroid of each of the *k* clusters becomes the new mean.

4. Steps 2 and 3 are repeated until convergence has been reached.

Results

- The amount clustered neighborhoods as mentioned on the methodology above was selected based on an the elbow method. The elbow methods works by plotting the number of clusters vs variance explained, this results on an elbow shaped figure depicting the right *k*. The Elbow method created with the data is below:



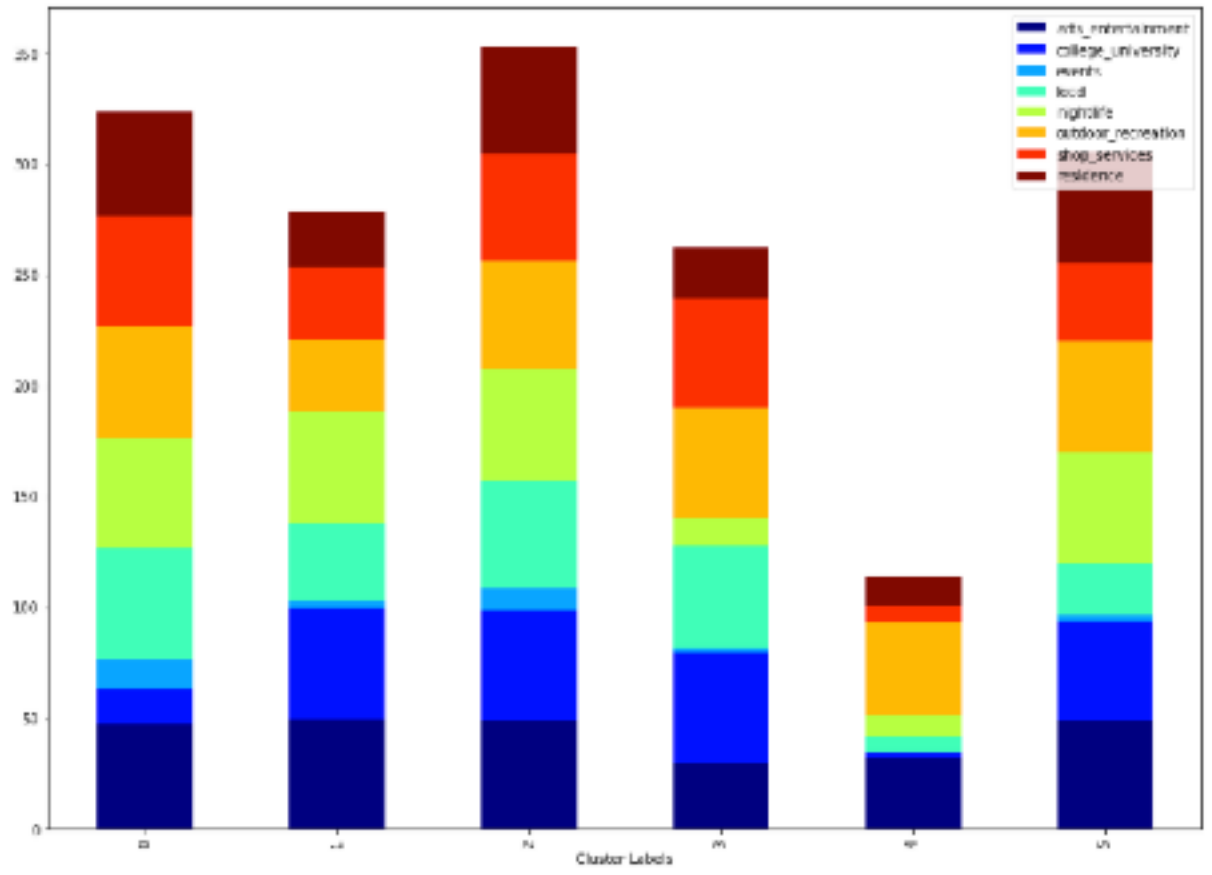
- In the table above, the "elbow" or inflection point is shown on 6 and 8. In this case 6 is chosen as an attempt of having the least clusters possible. The neighborhood data is then clustered in 6 different groups each with different number of neighborhoods.

Results (cont)

- The amount of neighborhoods in each cluster is below:

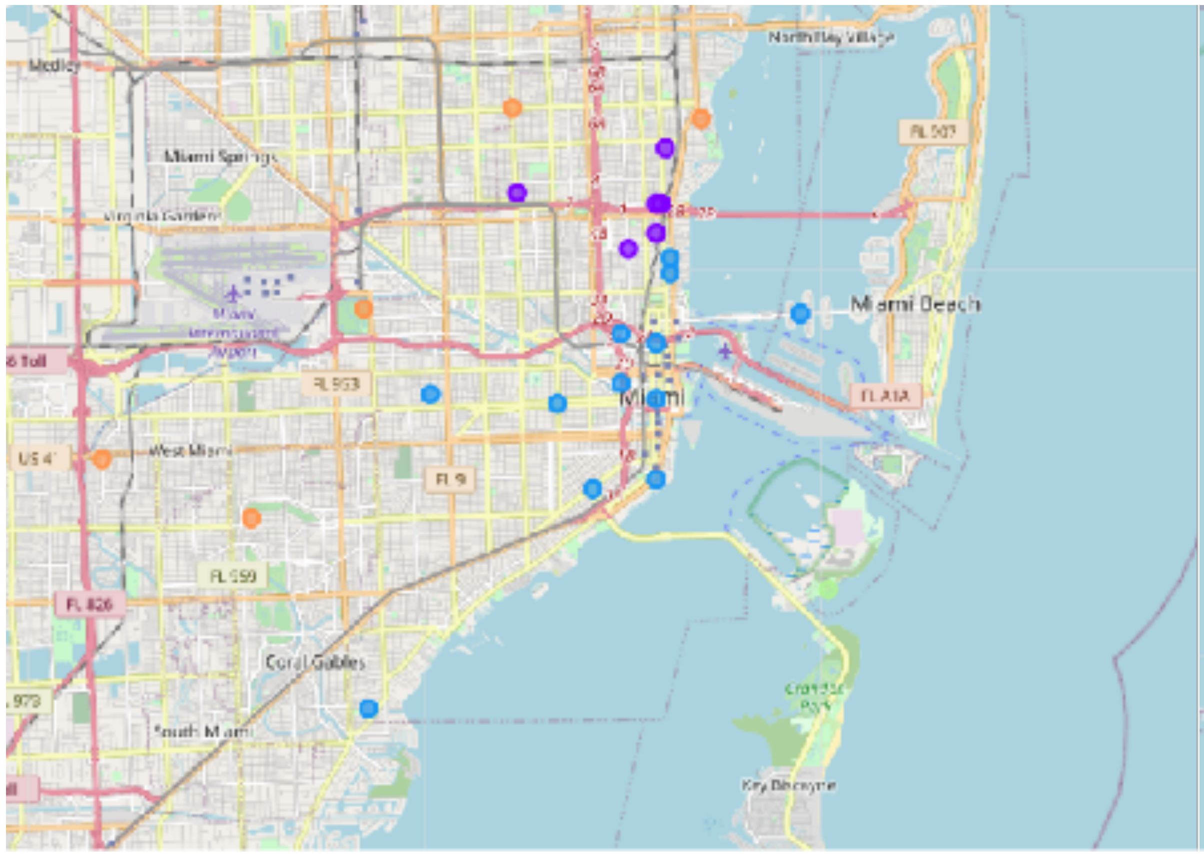
	Neighborhood
Cluster Labels	
0	4
1	6
2	24
3	4
4	1
5	5

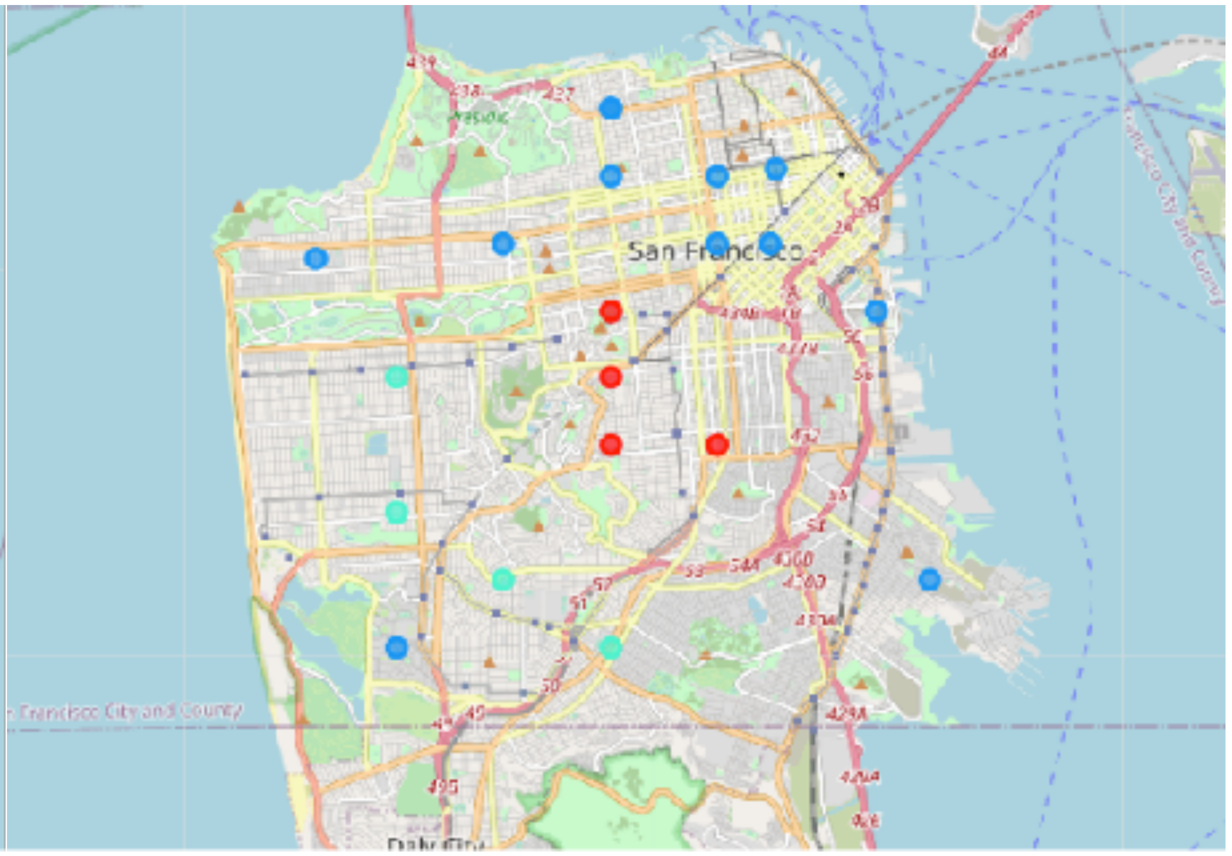
- The mean of each venues by cluster is below:



Results (cont)

- Miami clustered neighborhoods
- San Francisco clustered neighborhoods





Discussion

- In conclusion there is only one neighborhood that are similar in Miami and San Francisco.
- These type of neighborhoods (cluster 2) is recurrent and in most cases next to each other. However is visible in San Francisco and Miami map, that in some instances , the cluster 2 type of neighborhood can be found outside the center of the city. It will be interesting to understand what are the factors that drives high diverse and count of venues location outside of the city center.
- The cluster 2 type of neighborhoods as an archetype of city neighborhood.