Neighborhood Analysis

Juma Hamdan
FALL, 2020
jumahamdan@lewisu.edu

Abstract

Families moving to a new neighborhood face many hardships, from understanding the community and its people, learning the area, finding a job, and finding a place to live. The objective of this project is to help people that are planning to move by finding the best fit neighborhood at the new residence. People will sometimes have little knowledge about the new residence and its surroundings when relocating. Therefore, using socioeconomic data and venue data from different neighborhoods will help find neighborhood similarity based on those two data sources. It would be helpful to have a comparison between the most important neighborhoods in where they are moving to and where they used to reside. This will help find out which are similar.

***Index Terms* — Beautiful Soup, Folium, Json, K-Means Clustering, K-Nearest Neighbors, Principal Component Analysis, Web Scrapping**

I. INTRODUCTION

According to U.S. Census Bureau, average Americans move 11.7 times in their lifetime based upon the current age structure [1]. In cases of cross-country relocations, people sometimes have little knowledge about the new residence and its surroundings. Some people prefer neighborhoods with certain requirements, for example, young families, certain entertainment areas, etc.… A way to alleviate this problem that people have choosing neighborhoods is to find a neighborhood or neighborhoods like that of the persons' previous residence.

The idea of this project is to design and implement an application that will recommend nearby neighborhoods to a person based on their priorities in life. This neighborhood analysis will algorithmically determine which neighborhood would suit the person based on importance of housing cost, available means of public transportation, closeness to points of interests (venues), walking as a viable means of getting to work, and finally, school rating.

This analysis will take look at each neighborhood individually then consider neighborhood matching by looking at graphs to make a candidate selection [2]. This analysis will be conducted by gathering a list of neighborhoods from city-data which also provide information at a neighborhood level, for example, area, population, and average household size. Venue data will also be gathered from FourSquare. FourSquare will provide certain venues within a certain range of a certain neighborhood. With these sources combined will allow a person to find a similar neighborhood that of the previous residence.

In the future sections, we explain the use of clustering to help recommend a neighborhood or neighborhoods. We will also discuss the methodology, the results of the analysis along with a discussion, and a conclusion.

II. DATA EXPLORATION ANALYSIS

A. Data Description

The exclusive source of neighborhood data was Foursquare. Furthermore, we also collect and analyze socioeconomic data on neighborhood residents from city data which provides this information at

neighborhood level [8]. There are two primary stages of data acquisition. First, experimenting with neighborhoods to try and understand applicability of methods, for example, convert neighborhood name to coordinates using nominatim, pull venue data from coordinates: New York, New York, OH and Chicago, Chicago, IL, and scrape socioeconomic data from www.city-data.com. The second stage is the actual data acquisition, scrape neighborhood names in Chicago, IL from city data website, scrape socioeconomic data from city data website, pull venue data from FourSquare, and finally, join socioeconomic data and venues data.

Foursquare will provide data about venues within a mile radius for all Chicago neighborhoods. Socioeconomic data and venue data will join and make the full dataset to be analyzed [5].

The city data website will provide a list of neighborhoods near Chicago. For instance, the city data website for the state of Illinois will show 1 page out of 7 pages total of neighborhoods in the Illinois area. The list will further be filtered to contain only neighborhoods mentioning “Chicago”. The following attributes for each neighborhood will be collected.

B. Data Cleansing

This section focuses on cleaning, preparation of previously collected data. Below is the list of items in this section, more detail on each individual operation is in the text below.

Cleaning and preparation of the data especially the socioeconomic data needs to be cleaned and prepared by renaming fields, removing extra symbols in the data, and replacing and fixing NaN.

C. Geographical Data (Venues Data)

FourSquare will provide data about venues within a little less than 200-meter radius for New York, Ohio [6] and all Chicago, Illinois [5] neighborhoods. The venues were obtained using the FourSquare API for each city in both states. FourSquare API account has been created and unique credentials have been assigned. We have fetched the data from FourSquare API and converted it to a data frame. Venue data joined together with socioeconomic data will make the full dataset to be analyzed [7].

Table 1. DATA DICTIONARY GEOGRAPHICAL DATA

Attribute	Type	Example Value	Description
venue.name	String	“Millennium Park”	The name of the venue
venue.location.city	String	“Chicago”	The name of the city where the venue is located
venue.location.lat	Float	41.882598	The venue latitude
venue.location.lng	Float	-87.624126	The venue longitude
venue.categories	String	“Park”	The venue category
venue.location.address	String	“201 E Randolph St”	The address of the venue

D. Socioeconomic Data (Neighborhood Data)

Socioeconomic data on neighborhood residents was also collected and analyzed from http://www.city-data.com [4] which provides this information at neighborhood level.

Table 2 DATA DICTIONARY SOCIOECONOMIC DATA

Attribute	Type	Example Value	Description
Area	Integer	12.979	The part of region
Population	Integer	25,173	Number of people living in area
Average household size	Float	5.8	Number of people in a home
Average number of cars apartments	Float	1.1	Average number of cars apartments
Average number of cars houses	Float	1.6	Average number of cars houses
Number of males	Float	24255.5	Number of males
Median age females	Float	36.9	Median age females
Median age males	Float	36.4	Median age males
Median household income	Float	59415.0	Median household income in area
Median rent	Float	869.0	Median rent of area
Percent born in another U.S. state	Float	14.3	Percentage of people born in another U.S. state
Percent born in state	Float	43.0	Percentage of people born in this state
Percent families with children	Float	25.1	Percentage of married-couple families with children
Percent family household	Float	52.1	Percentage of family households
Percent foreign born residents	Float	40.6	Percentage of foreign-born residents
Percent married couple	Float	47.2	Percentage of married-couple families
Percent native residents born outside us	Float	2.1	Percentage of native residents but born outside the U.S.
Percent never married females, 15	Float	20.6	Percentage of never married females

years old and older			15 years old and over
Percent never married males, 15 years old and older	Float	22.5	Percentage of never married males 15 years old and over
Percent do not speak English well	Float	16.2	Percentage of people that speak English not well or not at all
Percent single mother households	Float	6.3	Percentage of single-mother households
Percent units' mortgage	Float	72.5	Percentage of units with a mortgage

III. Methodology

A. Tools Used

The tools used to perform this project were Jupyter Notebook, Nominatim, FourSquare API, and a custom-built web scrapper. Nominatim uses OpenStreetMap data to find locations on Earth by name and address (geocoding). It can also do the reverse, by finding an address for any location on the planet. Using credentials from FourSquare API, venues will be extracted by defining a specific city, state, and the radius of the extract. A web scrapper was built to extract specific socioeconomic data from city data website to be combined with the venues extracted from FourSquare.

B. Neighborhood Data

For each city, data that describes the names of its neighborhoods and their coordinates is needed. A dataset that specifies the neighborhood data for Chicago was provided by web scrapping the city data website using Beautiful Soup. The dataset is originally in HTML format that specifies the name of each neighborhood, its coordinates – latitude and longitude, and its socioeconomic data. Figure 1 shows a part of this HTML format.

```
<div class="content-item row"><br/><b>Area:</b> 1.915 <b>square miles</b><b
r/><br/>
<b>Population:</b> 48,511<br/><br/>
<b>Population density:</b><br/><div class="hgraph"><table><tr><td><b>Albany
Park:</b></td><td><p class="h" style="padding-left:150px;"></p>25,327 <b>pe
ople per square mile</b></td></tr><tr><td><b>Chicago:</b></td><td><p class
="a" style="padding-left:70px;"></p>11,909 <b>people per square mile</b></t
d></tr></table></div>
</div>
```

Figure 1 A part of the HTML format that describes Chicago Neighborhoods

```
{'Area:': '12.979',
 'Population:': '25,173',
 'avg household size': '3.9 people',
 'avg number of cars apts': '1.4',
 'avg number of cars houses': '2.3',
 'males': '12,397',
 'med age females': '47.8 years',
 'med age males': '42.2 years',
 'med household income': '$54,539',
 'med rent': '$732',
 'pct born in another us state': '14.7%',
 'pct born in state': '78.4%',
 'pct families with children': '28.1%',
 'pct family household': '44.7%',
 'pct foreign born residents': '5.3%',
 'pct married couple': '46.5%',
 'pct native residents born outside us': '2.0%',
 'pct never married females > 15': '14.1%',
 'pct never married males > 15': '18.0%',
 'pct not speak English well': '1.6%',
 'pct single mother': '9.7%',
 'pct units mortgage': '62.2%'}
```

Figure 2 Socioeconomic Data for Chicago Neighborhoods

To be able to use the data from this HTML format in the later parts of this project, it should be stored in a Pandas dataframe. Figure 3 shows the Python code used to process the HTML formatted data.

```
page_no = 1 # use only the first page for now
url = "http://www.city-data.com/indexes/neighborhoods/IL/%d/"%page_no

page = requests.get(url)
soup = BeautifulSoup(page.text, 'html.parser')
```

```
li = soup.find_all('li')
li[100] # show one of the neighborhoods
```

```
<li><a href="/neighborhood/Bushs-Corners-Libertyville-IL.html">Bushs Corner
s neighborhood in Libertyville, IL</a></li>
```

```
url_prefix = 'http://www.city-data.com'

# function returns neighborhood name and url for a list item
def get_neigh_url(li_item):
    value = url_prefix+li_item.find('a').get('href')
    key = li_item.text
    return (key,value)

print(get_neigh_url(li[100]))
```

```
('Bushs Corners neighborhood in Libertyville, IL', 'http://www.city-data.co
m/neighborhood/Bushs-Corners-Libertyville-IL.html')
```

Figure 3 The code used to process the HTML format

```
# Scrape all neighborhood names from one page identified by page_no
def scrape_page(page_no):
    url = "http://www.city-data.com/indexes/neighborhoods/IL/%d/"%page_no
    page = requests.get(url)
    soup = BeautifulSoup(page.text, 'html.parser')
    urls_dict = []
    for li_item in soup.find_all('li'):
        if li_item.text.find('Chicago, IL') >= 0: # Use only neighborhoods co
            urls_dict.append(get_neigh_url(li_item))
    return urls_dict

# Example -- scrape page 2 from the website
scrape_page(2)
```

```
[('Clearing neighborhood in Chicago, IL',
  'http://www.city-data.com/neighborhood/Clearing-Chicago-IL.html'),
 ('Cornerstone Lakes neighborhood in West Chicago, IL',
  'http://www.city-data.com/neighborhood/Cornerstone-Lakes-West-Chicago-IL.
html'),
 ('Cragin (Belmont Cragin) neighborhood in Chicago, IL',
  'http://www.city-data.com/neighborhood/Cragin-Chicago-IL.html'),
 ('Depaul neighborhood in Chicago, IL',
  'http://www.city-data.com/neighborhood/Depaul-Chicago-IL.html'),
 ('Douglas neighborhood in Chicago, IL',
  'http://www.city-data.com/neighborhood/Douglas-Chicago-IL.html'),
 ('Downtown West Chicago (Downtown) neighborhood in West Chicago, IL',
  'http://www.city-data.com/neighborhood/Downtown-West-Chicago-West-Chicago
-IL.html'),
 ('Dunning neighborhood in Chicago, IL',
  'http://www.city-data.com/neighborhood/Dunning-Chicago-IL.html'),
 ('East Garfield Park neighborhood in Chicago, IL',
  'http://www.city-data.com/neighborhood/East-Garfield-Park-Chicago-IL.htm
l'),
 ('East Side neighborhood in Chicago, IL',
  'http://www.city-data.com/neighborhood/East-Side-Chicago-IL.html'),
 ('Edgewater neighborhood in Chicago, IL',
  'http://www.city-data.com/neighborhood/Edgewater-Chicago-IL.html'),
 ('Edison Park neighborhood in Chicago, IL',
  'http://www.city-data.com/neighborhood/Edison-Park-Chicago-IL.html')]
```

Figure 4 The code used to scrape all neighborhood names from a single page

```
# convert scraped data to a Pandas dataframe
df = pd.DataFrame(l)
df.head(2).T
```

	0	1
Area:	1.915	0.321
Population:	48,511	894
avg household size	5.8 people	3.7 people
avg number of cars apts	1.1	2.5
avg number of cars houses	1.6	2.5
males	23,875	436
med age females	36.9 years	33.1 years
med age males	36.4 years	29.3 years
med household income	$59,415	$59,742
med rent	$869	$1,924
pct born in another us state	14.3%	9.4%
pct born in state	43.0%	57.2%
pct families with children	25.1%	17.8%
pct family household	52.1%	61.8%
pct foreign born residents	40.6%	33.0%
pct married couple	47.2%	66.7%
pct native residents born outside us	2.1%	0.4%
pct never married females > 15	20.6%	24.2%
pct never married males > 15	22.5%	25.5%
pct not speak English well	16.2%	20.5%
pct single mother	6.3%	2.6%
pct units mortgage	72.5%	88.9%
neighborhood	Albany Park neighborhood in Chicago, IL	Alta Vista Gardens neighborhood in West Chicag...

Figure 5 The code used to store Chicago neighborhood data into a dataframe

Having data of the coordinates of Chicago neighborhoods, it is possible to draw a map using Folium Python package of Chicago and its neighborhoods. Figure 6 shows this map; each circle represents the location of one neighborhood.

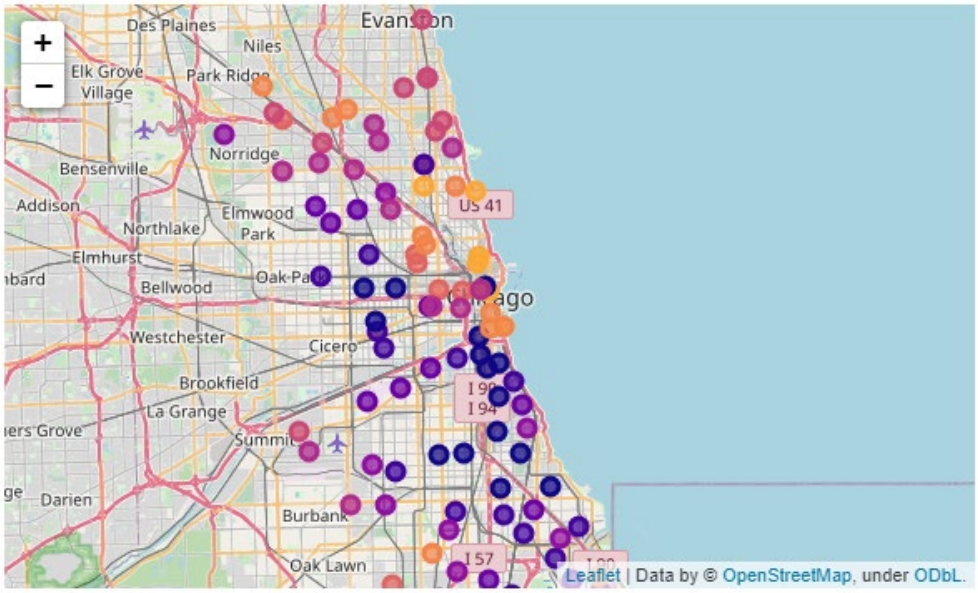

Figure 6 A map of Chicago and its neighborhoods

C. *Venues Data*

For each city, data that describes the venues of its neighborhoods and the categories of these venues is needed. Venues data will be retrieved from FourSquare which is a popular source of location and venue data. FourSquare API service will be utilized to access and download venues data.

To retrieve data from FourSquare using their API, a "client" should be constructed and used to request data related to a specific location. Figure 7 shows the Python implementation of this "client". In the example below, client indicates the API endpoint used, client_id and client_secret are credentials used to access the API service and are obtained when registering a FourSquare developer account, version indicates the API version to use.

```
def foursquare_api():
    with open('foursquare_keys.txt', 'r') as f:
        CLIENT_ID, CLIENT_SECRET = [lines.strip() for lines in f.readlines()]

    # VERSION = '20180605' # FOURSQUARE API VERSION
    VERSION = '20201112' # FOURSQUARE API VERSION

    # Construct the client object
    client = fs.Foursquare(client_id=CLIENT_ID,
                           client_secret=CLIENT_SECRET,
                           version=VERSION)
    return client
```

Figure 7 The code used to request data for venues


```
{'reasons': {'count': 0,
   'items': [{'summary': 'This spot is popular',
     'type': 'general',
     'reasonName': 'globalInteractionReason'}]},
  'venue': {'id': '4c47533649fa9521cb1f5e62',
   'name': 'Grant Park',
   'location': {'address': '337 E Randolph Dr',
    'crossStreet': 'at Michigan Ave',
    'lat': 41.8766257847394,
    'lng': -87.61926269478862,
    'labeledLatLngs': [{'label': 'display',
      'lat': 41.8766257847394,
      'lng': -87.61926269478862}],
    'distance': 443,
    'postalCode': '60601',
    'cc': 'US',
    'city': 'Chicago',
    'state': 'IL',
    'country': 'United States',
    'formattedAddress': ['337 E Randolph Dr (at Michigan Ave)',
     'Chicago, IL 60601',
     'United States']},
   'categories': [{'id': '4bf58dd8d48988d163941735',
     'name': 'Park',
     'pluralName': 'Parks',
     'shortName': 'Park',
     'icon': {'prefix': 'https://ss3.4sqi.net/img/categories_v2/parks_outdoors/park_',
      'suffix': '.png'},
     'primary': True}],
   'photos': {'count': 0, 'groups': []}},
  'referralId': 'e-0-4c47533649fa9521cb1f5e62-1'}
```


Figure 8 An example of venue data retrieved

Figure 9 shows the code used to create a function that takes a "client" which is created from the FourSquare API, a name of a city, radius is the maximum distance in meters between the specified location and the retrieved venues, and limit is used to limit the number of returned results if necessary. This function returns a dataframe with information about each neighborhood and its venues.

```
#https://developer.foursquare.com/docs/api-reference/venues/search/#parameters
def explore_venues(client, inputLocation, limit=100, radius=250):
    '''function to get n-places using explore in foursquare, where n is the limit when calling the
    This returns a pandas dataframe with name, city, categories, address, Latitude, Longitude.
    Arguments: client (foursquare_api()), inputLocation (city, state) , limit (defaults to 100), ra
    '''

    ll = get_coords(inputLocation)[0]
    params={'ll':ll,
            'limit':limit,
            'intent' : 'browse',
            'radius':radius,
           }
    venues = client.venues.explore(params)
    venues = venues['groups'][0]['items']
    venues = json_normalize(venues)
    filtered_cols = ['venue.name',
                     'venue.location.city',
                     'venue.categories',
                     'venue.location.address',
                     'venue.location.lat',
                     'venue.location.lng']
    venues = venues.loc[:, filtered_cols]
    venues['venue.categories'] = [v[0]['name'] for i, v in venues['venue.categories'].items()]
    venues.columns = [col.split(".")[-1] for col in venues.columns]


    return venues
```

Figure 9 Code used to build a venues dataframe for a city neighborhood

```
explore_venues(foursquare_api(), 'Chicago, IL', radius=100000)
```

	name	city	categories	address	lat	lng
0	Symphony Center (Chicago Symphony Orchestra)	Chicago	Concert Hall	220 S Michigan Ave	41.879275	-87.624680
1	Grant Park	Chicago	Park	337 E Randolph Dr	41.876626	-87.619263
2	The Art Institute of Chicago	Chicago	Art Museum	111 S Michigan Ave	41.879689	-87.623258
3	Millennium Park	Chicago	Park	201 E Randolph St	41.882598	-87.624126
4	Maggie Daley Park	Chicago	Park	337 E Randolph Dr	41.882905	-87.618846
...	...	...	...	...	...	...
95	Left Coast Food & Juice	Chicago	Vegetarian / Vegan Restaurant	2878 N Lincoln Ave	41.934139	-87.660919
96	Mariano's Fresh Market	Chicago	Grocery Store	3030 N Broadway St Ste 100	41.937335	-87.644776
97	Heritage Bicycles	Chicago	Coffee Shop	2959 N Lincoln Avenue	41.935760	-87.662830
98	City Press Juice & Bottle	Chicago	Juice Bar	2931 N Broadway St	41.935836	-87.644249
99	Bang Bang Pie Shop	Chicago	Pie Shop	2051 N California Ave	41.919014	-87.697173

Figure 10 Dataframe of venues

D. Exploratory Analysis

In this section, the datasets produced in the previous sections for both cities will be explored through effective visualizations to understand the data better. This section also focuses on checking field distributions, detecting any abnormalities such as percentage exceeding 100%, exploring correlation between fields, and discarding erroneous or not needed data.

Let us look at the categories that have most common venues. Figure 11 shows us the number of occurrences is counted for each venue category. After doing so, a bar plot can be used to visualize the popularity of the most common venue categories in each neighborhood.

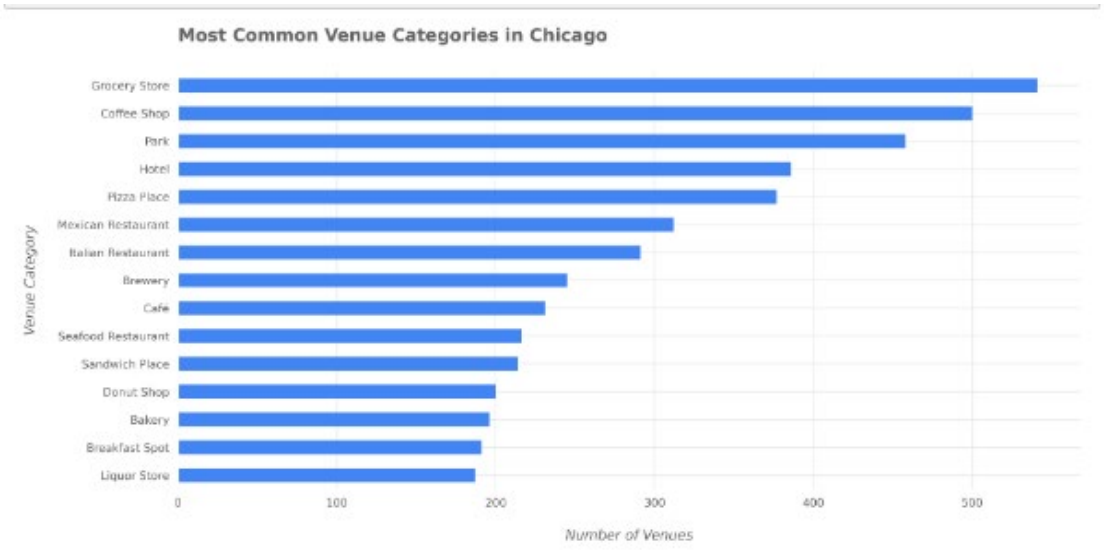

Figure 11 Most common venue categories in Chicago

Now another way of looking at the venue categories is to see which venue categories exist more in the neighborhood. To explain the difference with an example, suppose that there are 15 venues with the category “Park” and that these venues exist in 8 neighborhoods only out of 100 neighborhoods; also suppose that there are 10 venues with the category “Mediterranean Restaurant” and that these venues exist in 10 neighborhoods having each one of them in a different neighborhood. Then it can be said that the “Park” category is more common than “Mediterranean Restaurant” category because there are more venues under this category, and it can be that the “Mediterranean Restaurant” category is more widespread than the “Park” category because venues under this category exist in more neighborhoods than the other category.

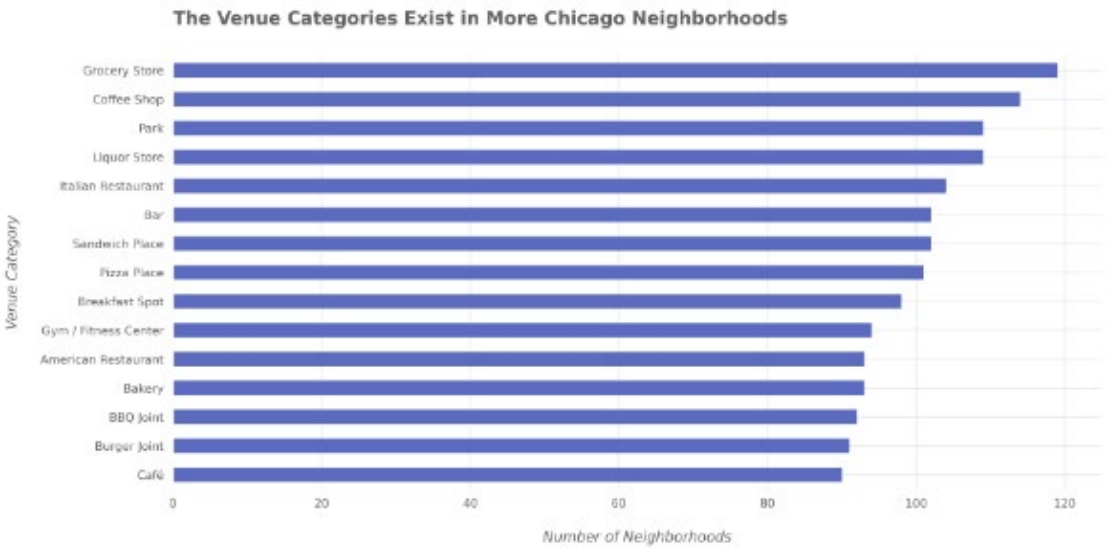

Figure 12 Most widespread venue categories in Chicago

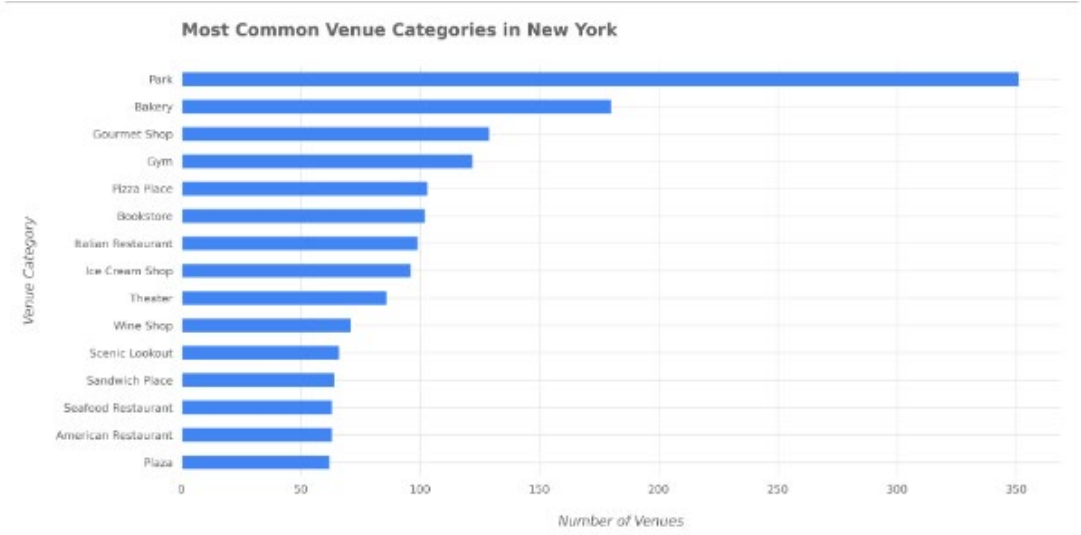

Figure 13 Most common venue categories in New York

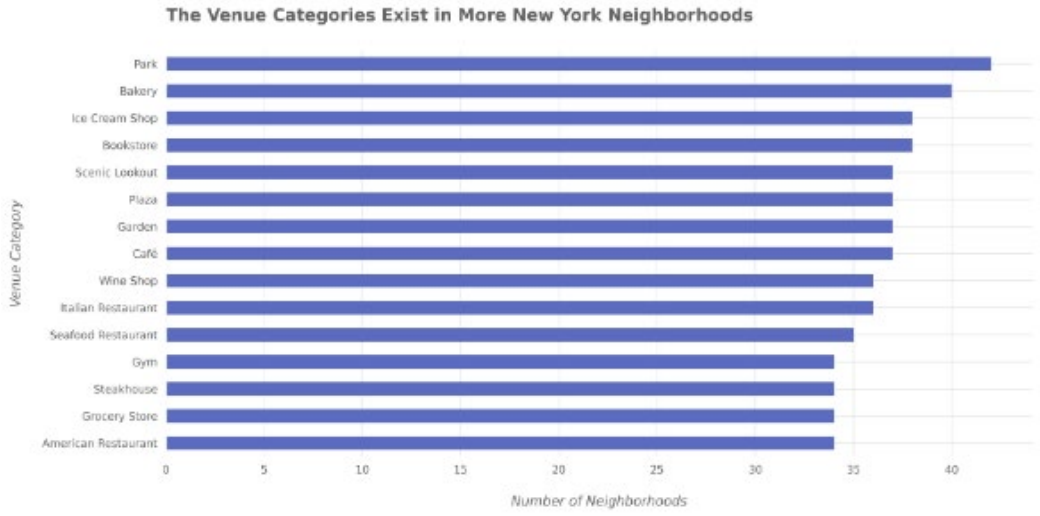

Figure 14 Most widespread venue categories in New York

Here we will look at the neighborhood analysis of area in miles and acres, average household size, average number of cars in apartments and houses, median age, median household income, median rent, percent of families with children, percent of family household, percent of married couple, percent of never married females older than 15, percent of never married males older than 15, percent of residents born in other states or outside of the United States, percent born in Illinois, percent of foreign born residents, percent of native residents born outside of the United States, percent of residents who do not speak English well, percent of single mother households, and finally, percentage of units that have a mortgage.

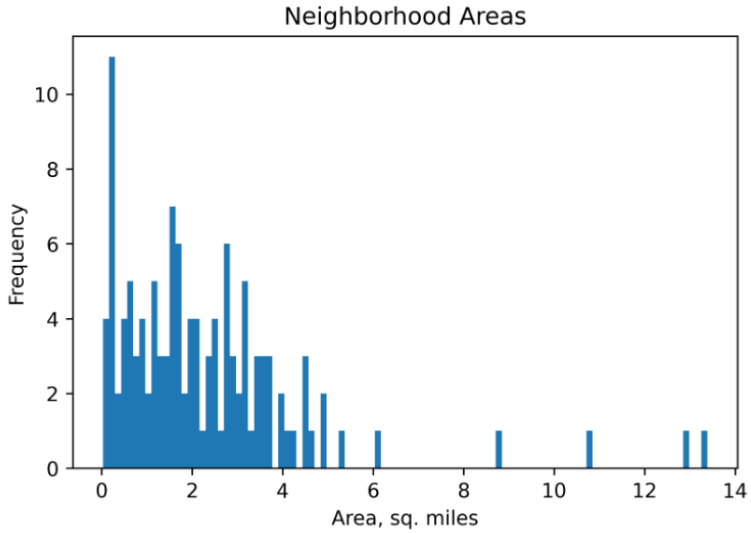

Figure 15 Neighborhood area in square miles

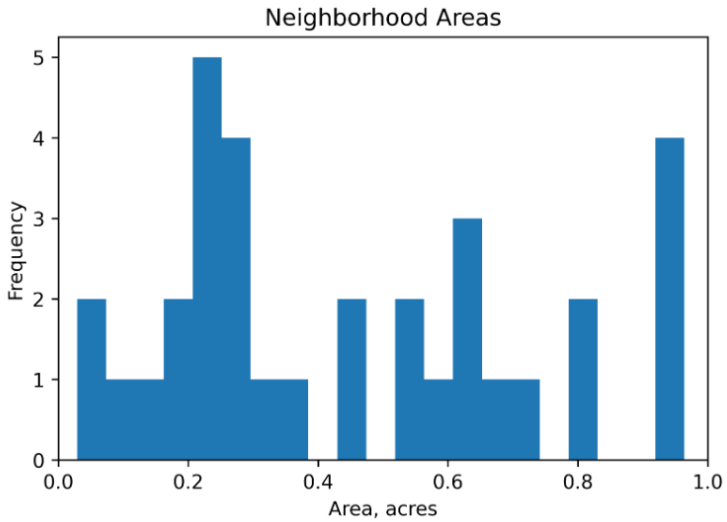

Figure 16 Neighborhood area in acres

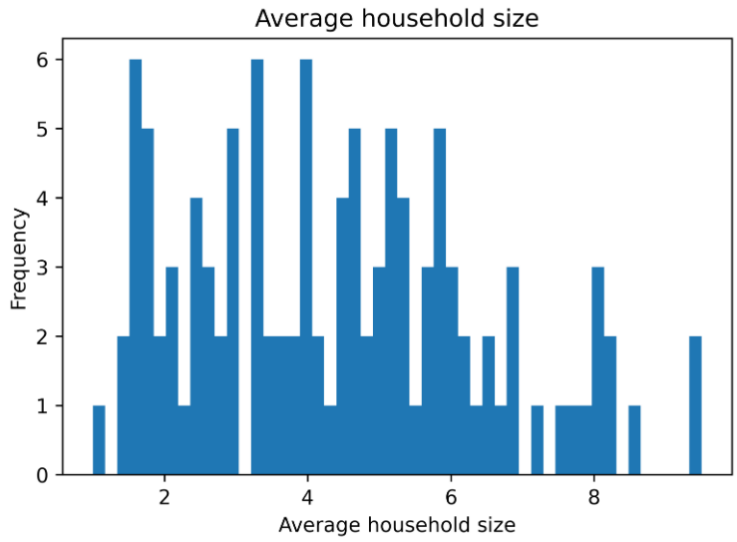

Figure 17 Average household size

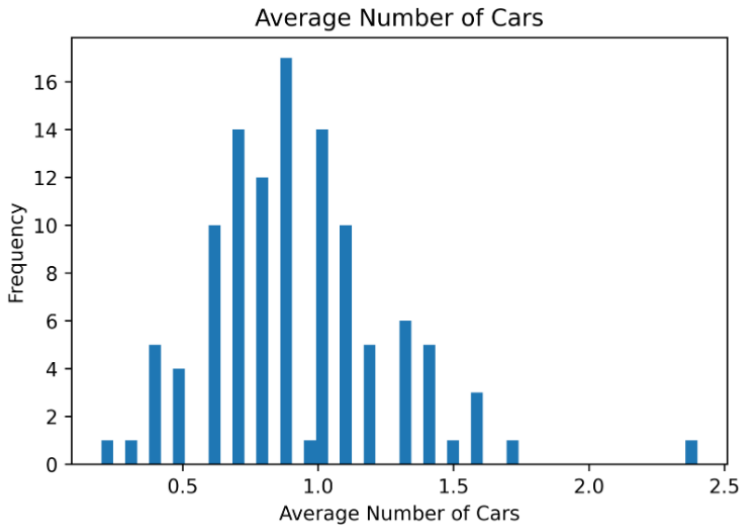

Figure 18 Average number of cars

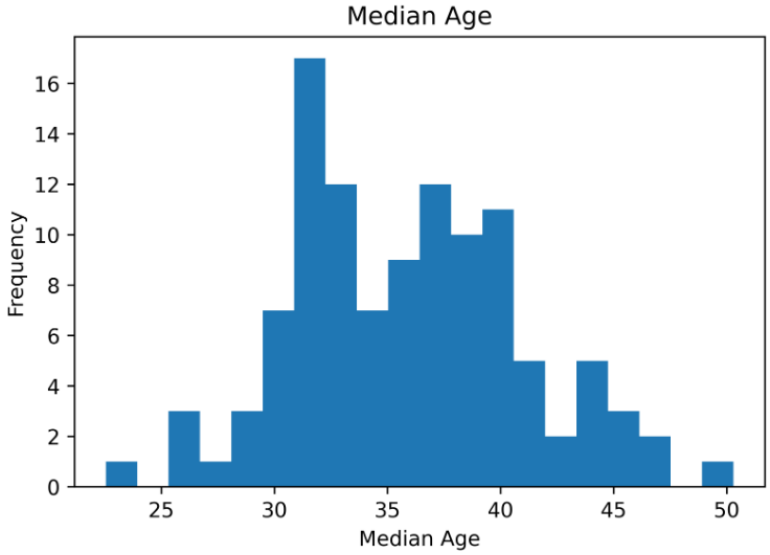

Figure 19 Median age of residents

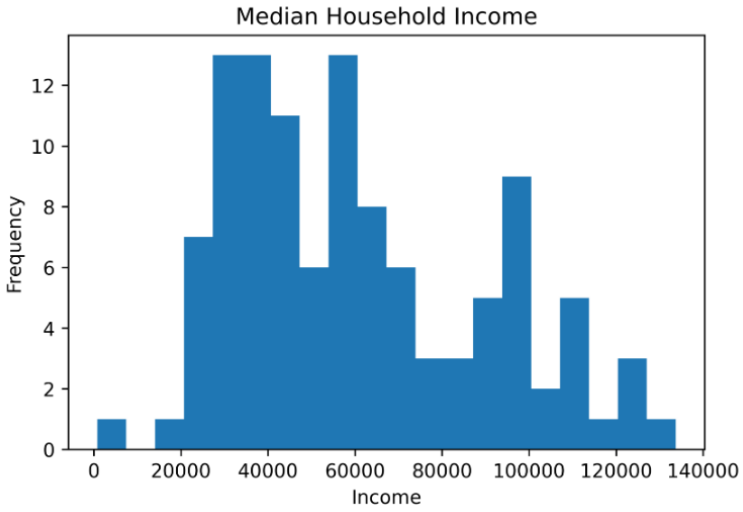

Figure 20 Median household income

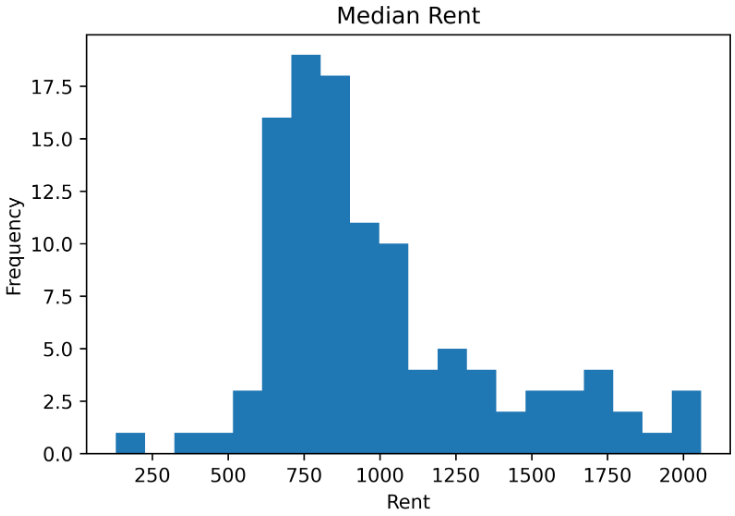

Figure 21 Median rent in neighborhood

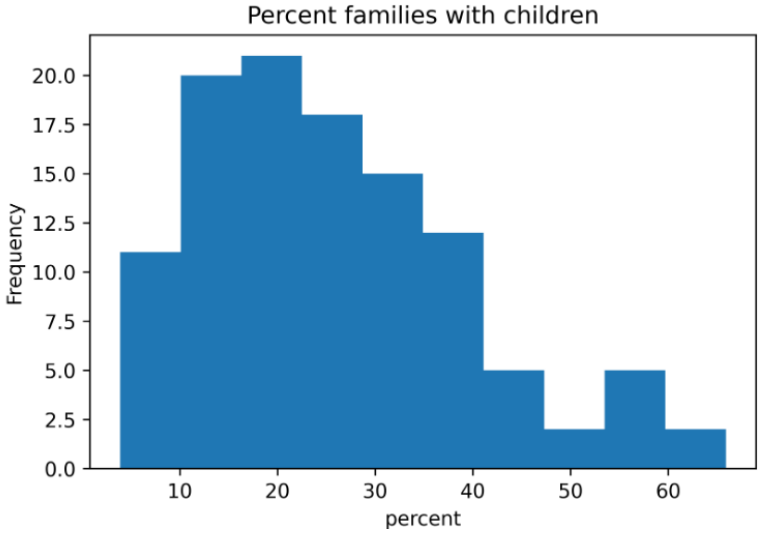

Figure 22 Percent of families that have children

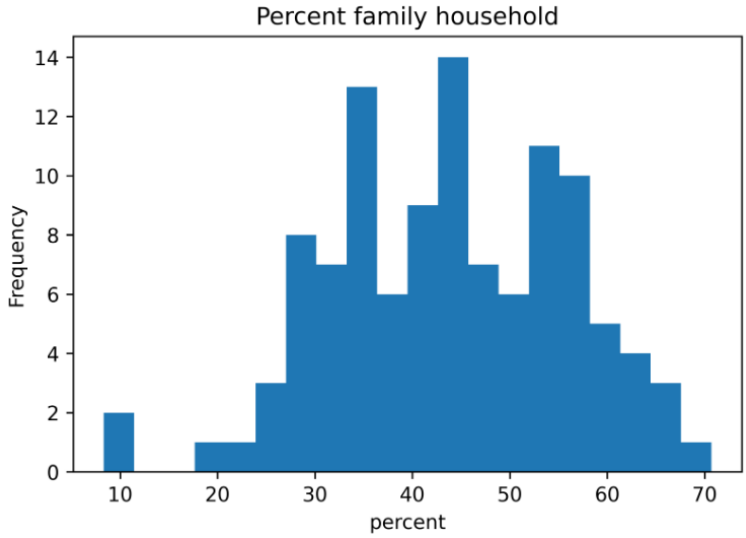

Figure 23 Percent of family households

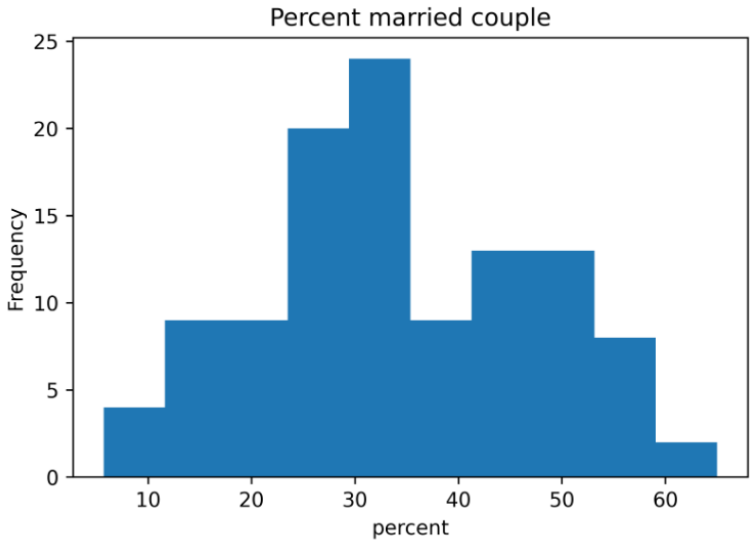

Figure 24 Percent of married couple

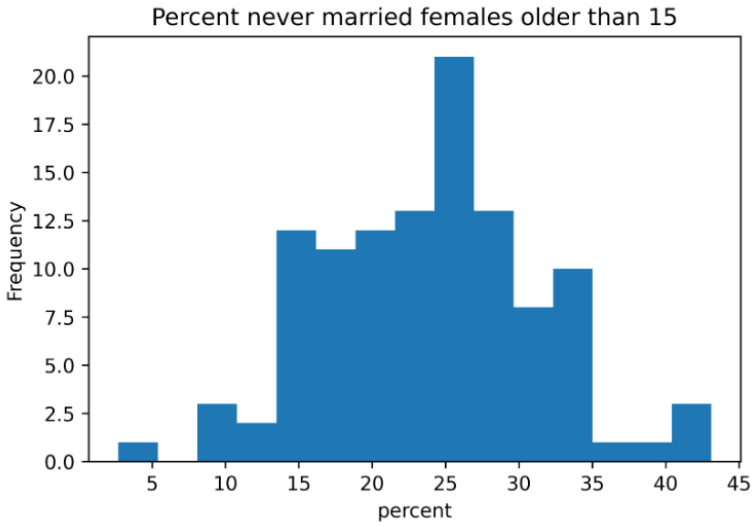

Figure 25 Percent of never married females older than 15

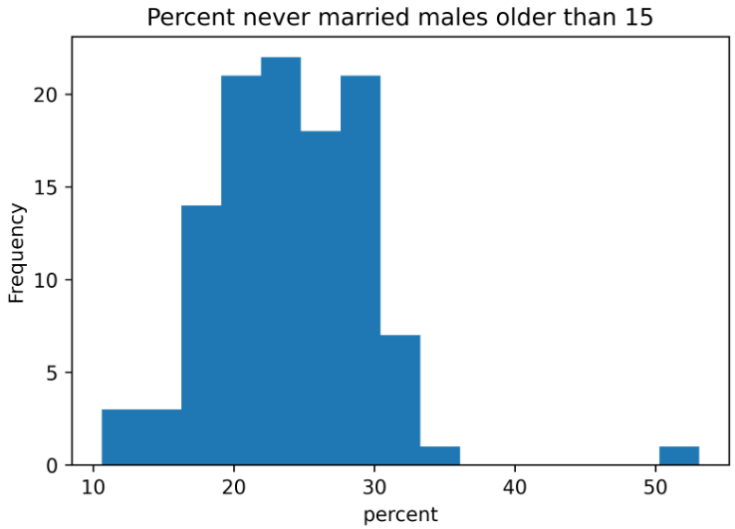

Figure 26 Percent of never married males older than 15

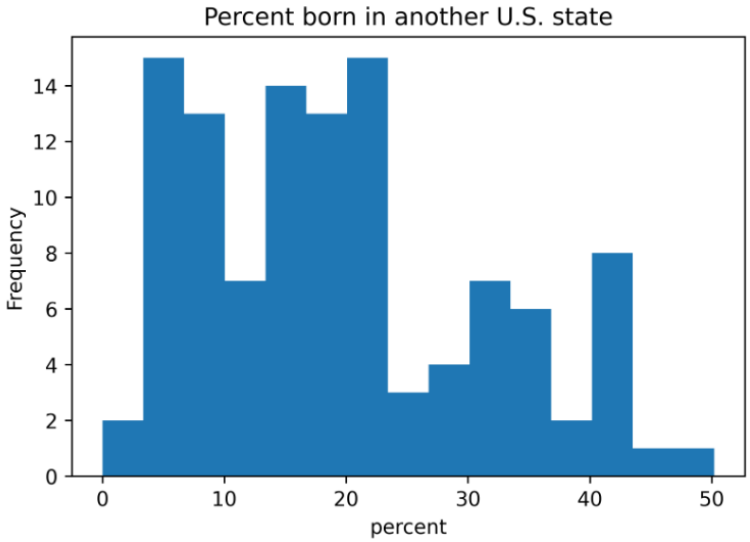

Figure 27 Percent of residents born in another U.S. state

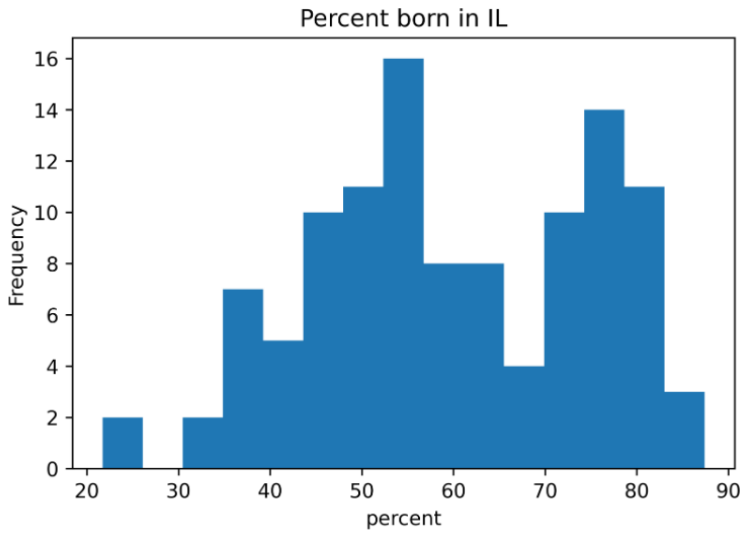

Figure 28 Percent of residents born in Illinois

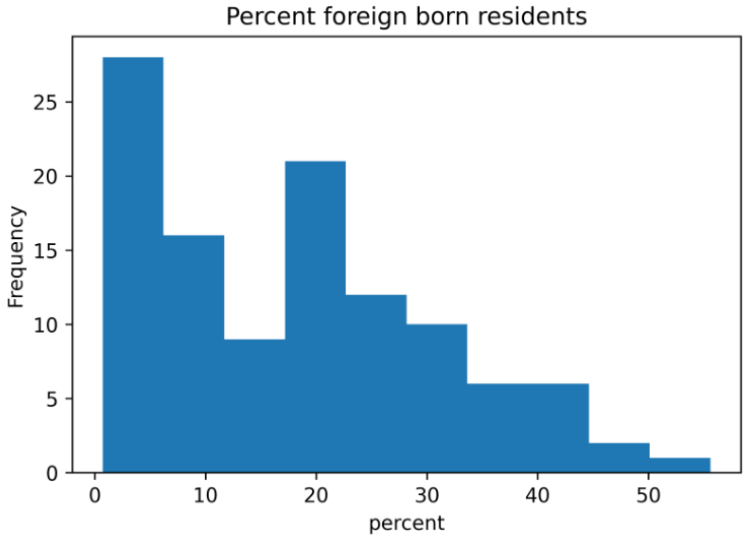

Figure 29 Percent of foreign-born residents

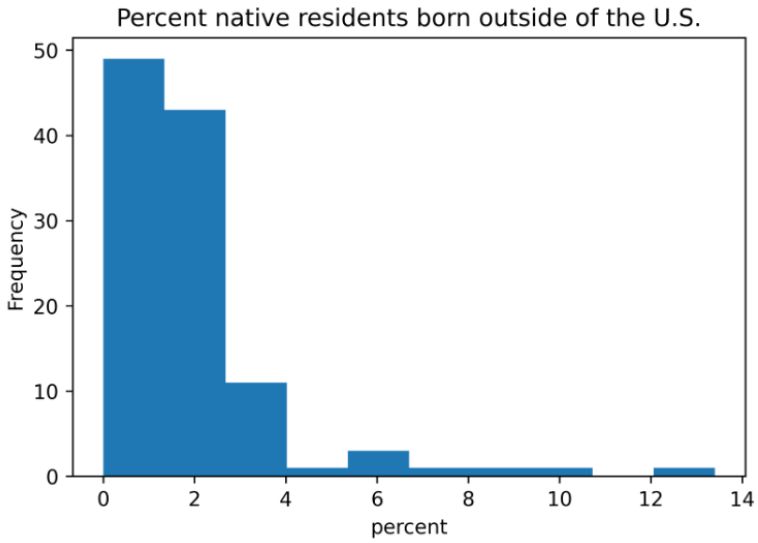

Figure 30 Percent of native residents born outside the United States

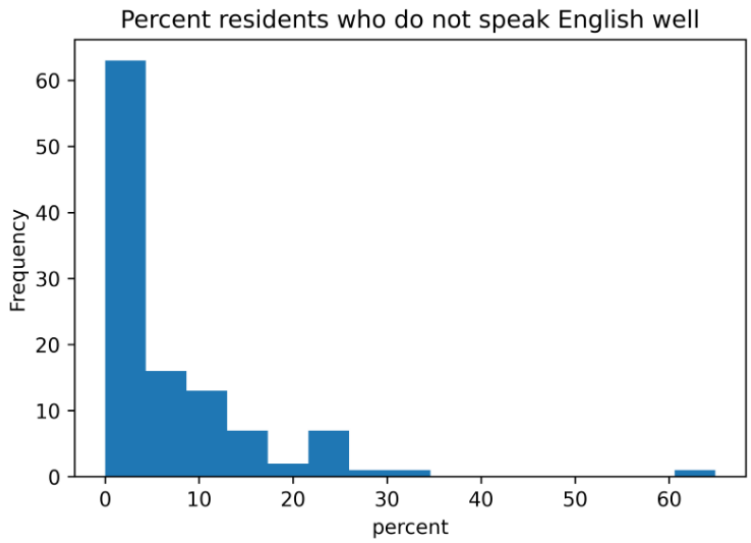

Figure 31 Percent of residents who do not speak English well

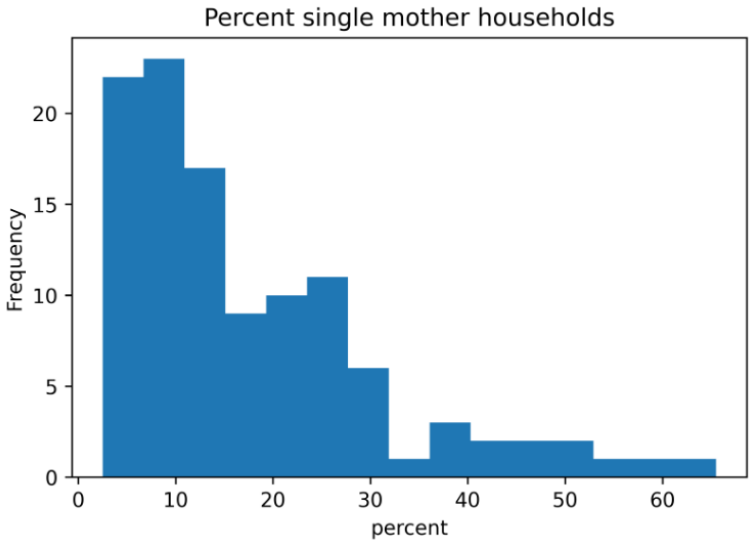

Figure 32 Percent of single mother households

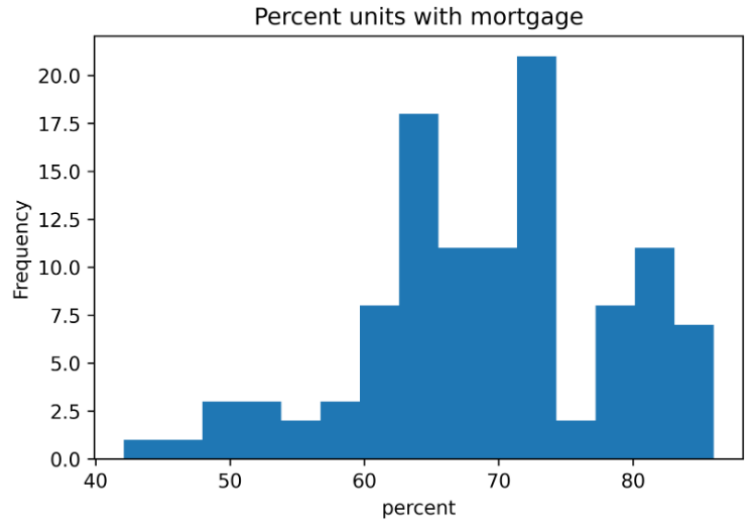

Figure 33 Percent of units that have a mortgage

E. Clustering of Neighborhoods

In this section, clustering will be applied on Chicago and New York neighborhoods to find similar neighborhoods in the two cities. Clustering is the process of finding similar items in a dataset based on the characteristics (features) of items in the dataset. In particular, K-Means clustering algorithm of the Scikit-learn Python library will be used. To be able to perform clustering, we need to feed the clustering algorithm with features in appropriate format. The data format that we saw above shown also below is not suitable for the clustering algorithm.

	Neighborhood	Neighborhood Latitude	Neighborhood Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
0	Albany Park neighborhood in Chicago, IL	41.879275	-87.62468	Potbelly Sandwich Shop	41.879333	-87.624655	Sandwich Place
1	Albany Park neighborhood in Chicago, IL	41.879275	-87.62468	Cliff Dwellers Club	41.879364	-87.624722	Social Club
3	Albany Park neighborhood in Chicago, IL	41.879275	-87.62468	Symphony Center (Chicago Symphony Orchestra)	41.879275	-87.624680	Concert Hall
4	Albany Park neighborhood in Chicago, IL	41.879275	-87.62468	Womens Health Consulting	41.879304	-87.624915	Doctor's Office
5	Albany Park neighborhood in Chicago, IL	41.879275	-87.62468	The Art Institute of Chicago	41.879689	-87.623258	Art Museum

Figure 34 Neighborhood and venue data for Chicago

The goal of clustering is to cluster neighborhoods based on the similarity of venue categories in the neighborhoods. This means that the two things of interest here are the neighborhoods and the venue categories in the neighborhood. Thus, the following two features will be selected out of the dataframe "Neighborhood" and "Venue Category". However, still after that, the data is not ready for the clustering algorithm because the algorithm works with numerical features.

For that, one-hot encoding will be applied on the "Venue Category" feature and the result of the encoding will be used for clustering. One-hot encoding will be applied on the Chicago data and on New York data then, as will be explained later, the data of the two cities will be combined. After applying one-hot encoding on Chicago data, the resulting dataframe looks like the one shown below in Figure 35.

	Neighborhood	ATM	Accessories Store	African Restaurant	Airport Lounge	Airport Service	Airport Terminal	American Restaurant	Amphitheater	Animal Shelter	...	Waterfront	Whisky Bar	Wine Bar	Wine Shop
0	Albany Park, Chicago, IL	0.00	0.0	0.00	0.0	0.00	0.00	0.00	0.0	0.0	...	0.00	0.0	0.0	0.00
1	Andersonville, Chicago, IL	0.00	0.0	0.00	0.0	0.00	0.00	0.01	0.0	0.0	...	0.00	0.0	0.0	0.00
2	Appletree, West Chicago, IL	0.00	0.0	0.00	0.0	0.00	0.01	0.03	0.0	0.0	...	0.00	0.0	0.0	0.00
3	Archer Heights, Chicago, IL	0.00	0.0	0.00	0.0	0.01	0.00	0.04	0.0	0.0	...	0.00	0.0	0.0	0.00
4	Armour Square, Chicago, IL	0.00	0.0	0.01	0.0	0.00	0.00	0.01	0.0	0.0	...	0.01	0.0	0.0	0.00

Figure 35 The result of one hot encoding on Chicago data

	Neighborhood	ATM	Accessories Store	African Restaurant	Airport Lounge	Airport Service	Airport Terminal	American Restaurant	Amphitheater	Animal Shelter	...	Waterfront	Whisky Bar	Wine Bar	Wine Shop
156	Wicker Park, Chicago, IL	0.00	0.0	0.0	0.0	0.0	0.0	0.00	0.0	0.0	...	0.00	0.0	0.0	0.00
157	Woodland, West Chicago, IL	0.01	0.0	0.0	0.0	0.0	0.0	0.04	0.0	0.0	...	0.00	0.0	0.0	0.00
158	Woodlawn, Chicago, IL	0.00	0.0	0.0	0.0	0.0	0.0	0.01	0.0	0.0	...	0.01	0.0	0.0	0.00
159	World Trade Center (WTC), New York, NY	0.00	0.0	0.0	0.0	0.0	0.0	0.01	0.0	0.0	...	0.00	0.0	0.0	0.02
160	Yorkville, New York, NY	0.00	0.0	0.0	0.0	0.0	0.0	0.00	0.0	0.0	...	0.01	0.0	0.0	0.00

Figure 36 The result of one hot encoding on Chicago and New York Data

The next step is aggregating the values for each neighborhood so that each neighborhood becomes represented by only one row. The aggregation will be done by grouping rows by neighborhood and by taking the mean of the frequency of occurrence of each category. Therefore, for example, if the Albany Park neighborhood has 30 venues (i.e., 30 rows in the dataframe of Figure 35) and 8 of these venues are of the "Park" category (i.e., the "Park" column in Figure 35 has a value of 1 for 8 of Albany Park rows), then Albany Park row in the aggregated dataframe will have the value of 8/30 = 0.26 for the "Park" column. Figure 36 shows how the aggregated dataframe looks like for both cities.

After producing the aggregated dataframes for each of the cities, these dataframes should be combined before applying the clustering algorithm. However, in order to distinguish Chicago neighborhoods from New York City neighborhoods in the new dataframe, a text string is appended to each of the neighborhood names before merging the dataframes: for Chicago, the string to be appended is ", Chicago, IL".

	Neighborhood	ATM	Accessories Store	African Restaurant	Airport Lounge	Airport Service	Airport Terminal	American Restaurant	Amphitheater	Animal Shelter	...	Waterfront	Whisky Bar	Wine Bar	Wine Shop
156	Wicker Park, Chicago, IL	0.00	0.0	0.0	0.0	0.0	0.0	0.00	0.0	0.0	...	0.00	0.0	0.0	0.00
157	Woodland, West Chicago, IL	0.01	0.0	0.0	0.0	0.0	0.0	0.04	0.0	0.0	...	0.00	0.0	0.0	0.00
158	Woodlawn, Chicago, IL	0.00	0.0	0.0	0.0	0.0	0.0	0.01	0.0	0.0	...	0.01	0.0	0.0	0.00
159	World Trade Center (WTC), New York, NY	0.00	0.0	0.0	0.0	0.0	0.0	0.01	0.0	0.0	...	0.00	0.0	0.0	0.02
160	Yorkville, New York, NY	0.00	0.0	0.0	0.0	0.0	0.0	0.00	0.0	0.0	...	0.01	0.0	0.0	0.00

Figure 37 The combination of Chicago and New York City aggregated dataframes

The most common categories for each neighborhood using the dataframe from the Figure above, another dataframe is created to specify the 7 most common categories for each neighborhood in Chicago and New York City. This dataframe is created by retrieving the 7 categories with the largest values for each neighborhood in the Figure above. The Figure below shows this dataframe.

	Neighborhood	1st Most Common Category	2nd Most Common Category	3rd Most Common Category	4th Most Common Category	5th Most Common Category	6th Most Common Category	7th Most Common Category
0	Albany Park, Chicago, IL	Convenience Store	Brewery	Grocery Store	Discount Store	Farmers Market	Ethiopian Restaurant	Event Space
1	Andersonville, Chicago, IL	Coffee Shop	Grocery Store	Brewery	Pizza Place	Italian Restaurant	Gym	Beach
2	Appletree, West Chicago, IL	Park	Pizza Place	Breakfast Spot	Coffee Shop	Mexican Restaurant	Hot Dog Joint	Pharmacy
3	Archer Heights, Chicago, IL	Mexican Restaurant	Pizza Place	Taco Place	Grocery Store	Hotel	American Restaurant	Park
4	Armour Square, Chicago, IL	Park	Grocery Store	Coffee Shop	Café	Art Gallery	Bakery	Lounge

Figure 38 Most common categories for each neighborhood

	Neighborhood	1st Most Common Category	2nd Most Common Category	3rd Most Common Category	4th Most Common Category	5th Most Common Category	6th Most Common Category	7th Most Common Category
159	World Trade Center (WTC), New York, NY	Park	Bakery	Gourmet Shop	Bookstore	Scenic Lookout	Thai Restaurant	Ice Cream Shop
160	Yorkville, New York, NY	Park	Bakery	Gym	Theater	Pizza Place	Plaza	Exhibit

Figure 39 Most common categories for each neighborhood

F. K-Means Clustering

By obtaining the dataframe of the Figures above, it is now possible to apply the clustering algorithm. Figure 39 shows the code used to perform clustering using the K-Means algorithm of Scikit-learn library.

The variable named *df_scaled* is the neighborhood data being scaled. Standard Scalar removes the mean and scales each feature to unit variance.

```
df_scaled = StandardScaler().fit_transform(df)
df_scaled
```

```
array([[-0.05316508,  0.02590917,  0.09648052, ..., -0.58222251,
        -0.54321448, -1.02766652],
       [-0.48744172, -0.35045306,  0.17289116, ..., -0.58222251,
        -0.54321448,  0.87377739],
       [-0.65966071, -0.53319655, -0.28557266, ..., -0.58222251,
         0.88725031,  0.23996276],
       ...,
       [-0.56643193, -0.16507055, -0.89685775, ...,  0.56766695,
         2.3177151 ,  0.23996276],
       [-0.69051097, -0.51140923, -1.12608966, ...,  2.86744586,
         2.3177151 ,  1.50759203],
       [-0.53558168, -0.1253992 , -0.13275139, ...,  0.56766695,
        -0.54321448, -0.39385188]])
```

Figure 40 Code used to scale the dataframe

```
max_k = 50
squared_distance = np.zeros(max_k)

for k in range(2, max_k):
    kmeans = KMeans(n_clusters = k, random_state = 0)
    kmeans = kmeans.fit(df_scaled)
    squared_distance[k] = kmeans.inertia_
    print(k, sep=' ', end = ' ', flush = True)
```

```
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
46 47 48 49
```

Figure 41 Code used to find the optimal number of clusters

```
plt.figure(figsize=(6, 4))
plt.plot(squared_distance)
plt.xlabel('K')
plt.ylabel('Avg distance')
plt.title('Average distances from points to their cluster centers')
plt.xlim([2,max_k])
plt.show()
```

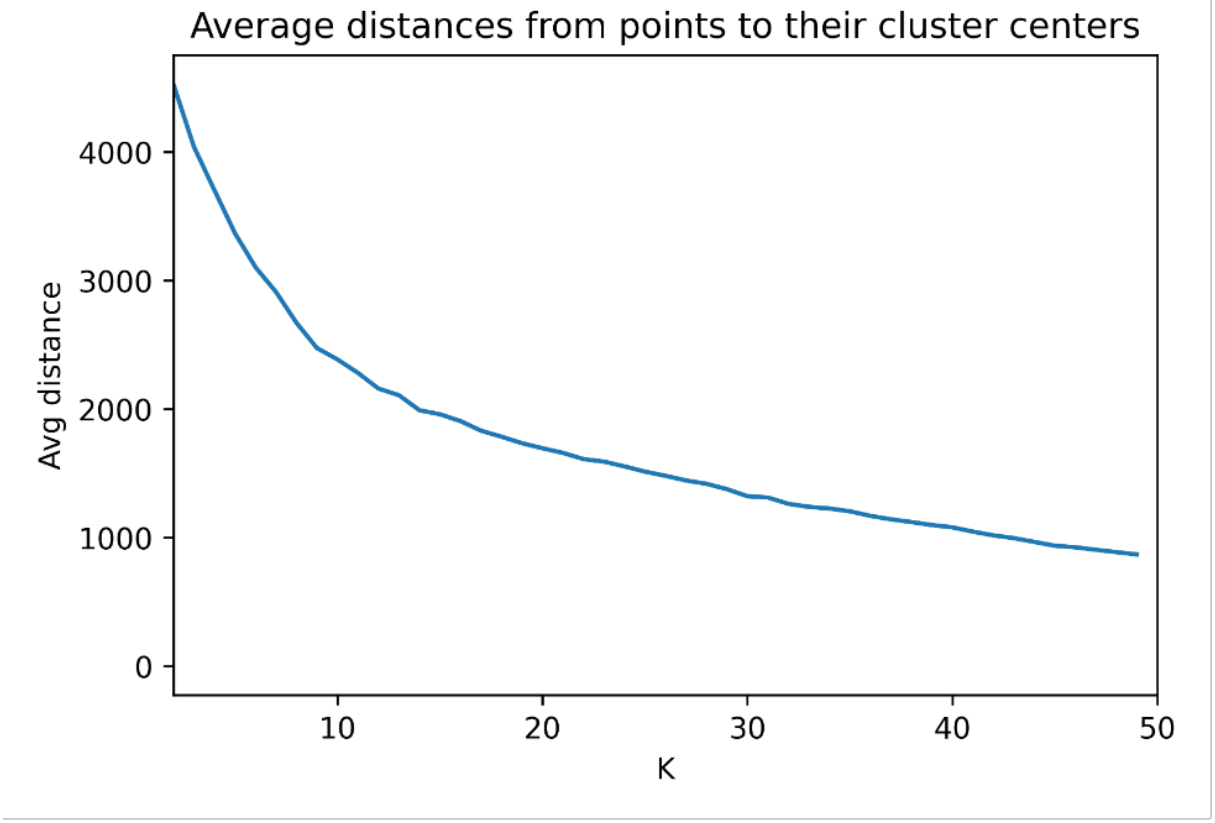

Figure 42 Elbow plot used to find optimal number of clusters

```
kclusters = 13 # the optimum number of clusters
kmeans = KMeans(n_clusters = kclusters, random_state = 0).fit(df)
df['cluster'] = kmeans.labels_
```

```
df.T.tail()
```

Neighborhood	Albany Park, Chicago, IL	Andersonville, Chicago, IL	Appletree, West Chicago, IL	Archer Heights, Chicago, IL	Armour Square, Chicago, IL	Ashburn, Chicago, IL	Auburn Gresham, Chicago, IL	Austin, Chicago, IL	Avalon Park, Chicago, IL	Avondale, Chicago, IL	...	Turtle Bay, New York, NY	Two Bridges, New York, NY	Upper East Side (UES), New York, NY
Scenic Lookout	0.0	0.00	0.00	0.00	0.0	0.00	0.00	0.00	0.00	0.00	...	0.01	0.02	0.01
Thai Restaurant	0.0	0.00	0.01	0.00	0.0	0.01	0.01	0.00	0.00	0.00	...	0.00	0.02	0.00
Ice Cream Shop	0.0	0.03	0.02	0.03	0.0	0.05	0.01	0.03	0.01	0.05	...	0.01	0.04	0.01
cluster	5.0	4.00	4.00	7.00	12.0	5.00	0.00	8.00	7.00	5.00	...	9.00	0.00	6.00
map_cluster	5.0	7.00	7.00	4.00	0.0	5.00	1.00	2.00	4.00	5.00	...	11.00	1.00	9.00

Figure 43 Chicago and New York City neighborhoods and their clusters.

```
kmeans.labels_
```

```
array([ 5,  4,  4,  7, 12,  5,  0,  8,  7,  5,  8,  4,  7,  0,  0, 12,  9,
       12, 12, 12,  0,  9, 12,  5,  9,  8,  2, 12,  7,  5,  0,  0,  5,  4,
        0, 12,  9, 12,  0,  5,  4,  0,  0,  4,  7,  7,  4,  8,  7,  5,  7,
       12,  7,  2,  4,  8,  2,  4,  7,  5,  9, 12,  0,  7,  0,  7,  5,  6,
        4,  6,  0,  9,  2,  9, 12,  0,  7,  5,  7, 12,  4,  7,  5, 12,  4,
        9,  9,  9, 12,  4,  4,  9,  0,  4,  7,  0,  8,  9,  0,  9,  7,  2,
        5,  5,  0, 12,  7,  0, 12,  7,  9,  5,  0,  6,  9,  4,  0,  9,  4,
        7,  2,  7,  4,  2,  3,  8,  4,  9,  9,  5, 10,  5,  9,  9,  4,  0,
       12,  4,  0,  6,  6,  4,  9,  9,  4, 11, 11, 11,  9,  0,  6,  1,  2,
       10,  3,  9,  9,  9])
```

Figure 44 Cluster labels for each neighborhood in the data

As can be seen from the Figure above, the clustering algorithm produced cluster labels; these labels denote the cluster of each record (i.e., each neighborhood) in the data. Using these labels and the dataframe from above, another dataframe is constructed to show the neighborhoods of Chicago and New York City, and the cluster to each neighborhood.

The output of the clustering operation is 13 clusters with cluster labels 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12. Each cluster is expected to contain a group of similar neighborhoods based on the categories of the venues in each neighborhood and the socioeconomic data for each neighborhood. The clustering algorithm was run on more than 100 neighborhoods in Chicago and New York City.

IV. Results

The output of the clustering operation is 13 clusters with cluster labels 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12. Each cluster is expected to contain a group of similar neighborhoods based on the categories of the venues in each neighborhood and the socioeconomic data for each neighborhood. The clustering algorithm was run on more than 100 neighborhoods in Chicago and New York City.

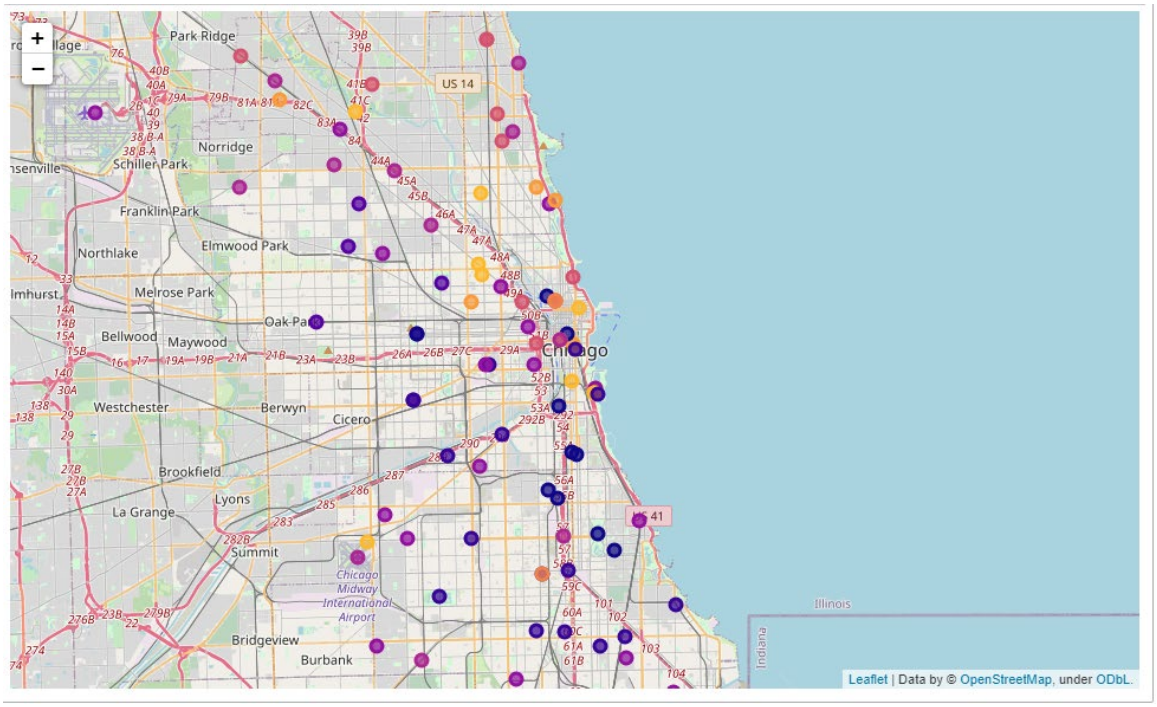

Figure 45 Map illustrating recommended neighborhoods for Chicago

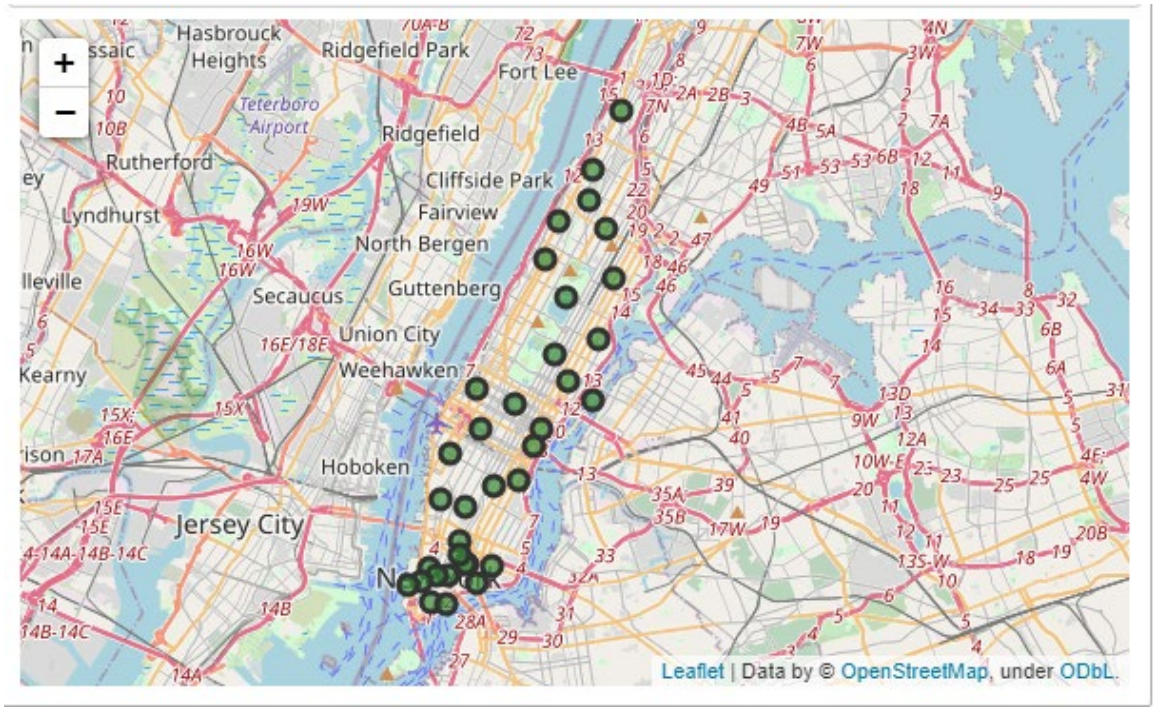

Figure 46 Map illustrating recommended neighborhoods for New York City

Neighborhood	lat	lng	map_cluster	med household income
Armour Square, Chicago, IL	41.788776	-87.601684	0.0	32295.0
Bronzeville, Chicago, IL	41.831272	-87.626548	0.0	30979.0
Burnside, Chicago, IL	41.724485	-87.598360	0.0	40373.0
Cabrini-Green (Cabrini Green), Chicago, IL	41.898805	-87.640937	0.0	41250.0
Canaryville, Chicago, IL	41.814756	-87.640329	0.0	38663.0
Chinatown, Chicago, IL	41.851203	-87.633525	0.0	38219.0
Douglas, Chicago, IL	41.830361	-87.623862	0.0	36665.0
Fair Meadows, West Chicago, IL	42.135445	-87.831782	0.0	28546.0
Fuller Park, Chicago, IL	41.811377	-87.634645	0.0	46708.0
Jeffery Manor, Chicago, IL	41.715035	-87.570046	0.0	37633.0
Madison Square, Chicago, IL	41.882171	-87.628681	0.0	30357.0
Manhattan, New York, NY	40.789624	-73.959894	0.0	89.0
North Center, North Chicago, IL	42.325578	-87.841182	0.0	42895.0
Oakland, Chicago, IL	42.679706	-83.297526	0.0	39527.0
Pullman, Chicago, IL	41.694537	-87.606464	0.0	44459.0
Riverdale, Chicago, IL	41.654029	-87.609837	0.0	37832.0
Washington Park, Chicago, IL	41.795928	-87.611151	0.0	24678.0
West Garfield Park, Chicago, IL	41.882088	-87.715917	0.0	24967.0

Figure 47 Map Cluster 0

As we can notice from Figure 47 *map_cluster 0* can only recommend 1 neighborhood outside of Chicago which is Manhattan, New York.

Neighborhood	lat	lng	map_cluster	med household income
Andersonville, Chicago, IL	41.977139	-87.669273	7.0	85830.0
Appletree, West Chicago, IL	41.858798	-88.189060	7.0	81237.0
Bellevue, New York, NY	42.893392	-78.733919	7.0	95679.0
Beverly, Chicago, IL	41.721483	-87.665896	7.0	100631.0
City Hall, New York, NY	40.713282	-74.006978	7.0	112731.0
Edison Park, Chicago, IL	42.002484	-87.818219	7.0	116503.0
Garment District, New York, NY	40.753694	-73.990517	7.0	113561.0
Gold Coast, Chicago, IL	41.906699	-87.625331	7.0	102502.0
Greektown, Chicago, IL	41.878445	-87.646932	7.0	105481.0
Hillside, West Chicago, IL	41.881111	-88.198889	7.0	96435.0
Lakeview East (East LakeView), Chicago, IL	42.234253	-87.945024	7.0	104994.0
Lincoln Square, Chicago, IL	40.148032	-89.363308	7.0	88737.0
Little Italy, New York, NY	40.719273	-73.998215	7.0	83985.0
Manhattan Valley, New York, NY	40.799776	-73.967772	7.0	97826.0
Morningside Heights, New York, NY	40.810000	-73.962500	7.0	94786.0
Near South Side, Chicago, IL	41.896471	-87.635719	7.0	111120.0
Old Town, Chicago, IL	41.896198	-87.655358	7.0	106789.0
Ravenswood, Chicago, IL	41.965591	-87.666724	7.0	88524.0
Rogers Park, Chicago, IL	42.009574	-87.675550	7.0	81657.0
Rogers Park, Chicago, IL	42.009574	-87.675550	7.0	81657.0
Sauganash, Chicago, IL	41.990036	-87.742289	7.0	102053.0
Stuyvesant Town (Stuyvesant Park), New York, NY	42.390361	-73.781513	7.0	100091.0
Woodland, West Chicago, IL	41.669845	-87.630629	7.0	96435.0

Figure 48 Map Cluster 7

As we can notice from Figure 48 *map_cluster 7* is able to recommend many neighborhoods outside of Chicago.

Figure 49 below, shows the number of Chicago neighborhoods and the number of New York City neighborhoods in each cluster of the 12 resulting clusters.

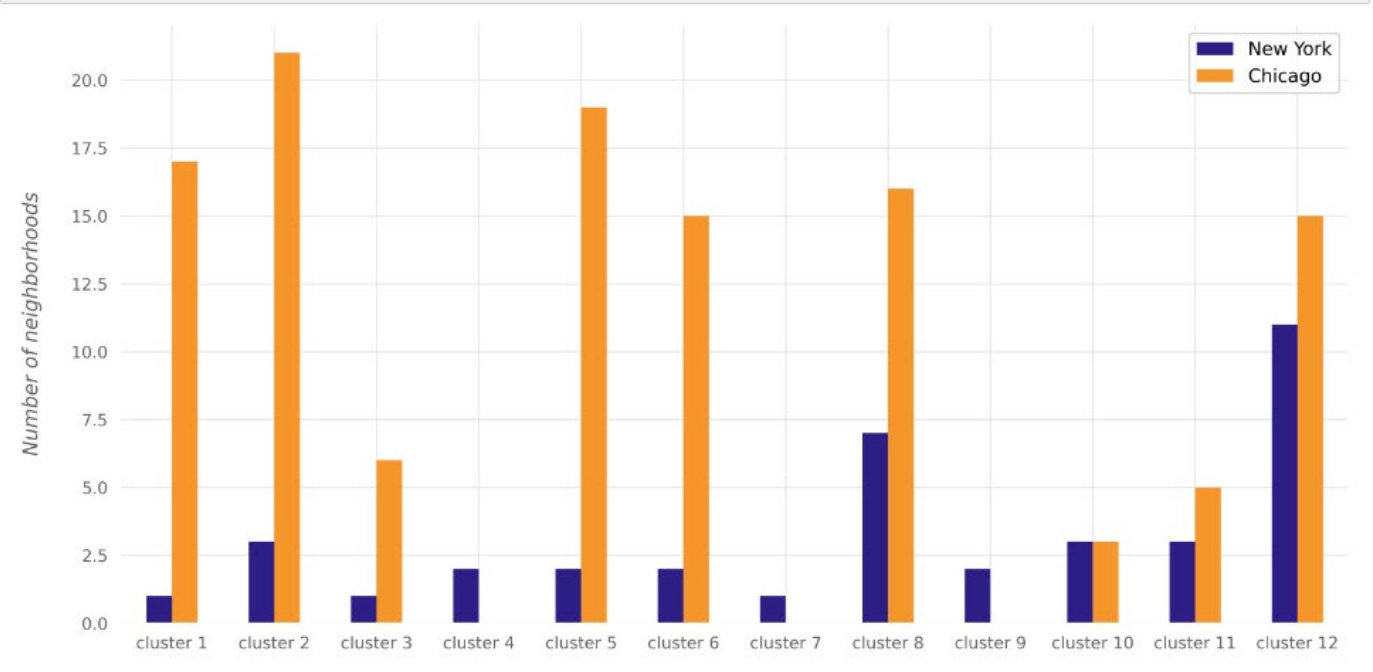

Figure 49 The number of Chicago and New York City neighborhoods in each cluster

Using Principal Component Analysis which is a dimensionality-reduction method that is used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set. Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity.

```
def pca2(data, pc_count = None):
    return PCA(n_components = 13).fit_transform(data)
```

```
pca = pca2(df)
pca
```

```
array([[ 2.51424836e+03, -1.70706004e+04,  3.65166395e+02, ...,
        -3.63281501e+00,  2.43938893e+00, -2.27616525e+00],
       [-3.16147541e+04,  3.03697812e+03,  1.16594455e+03, ...,
        -4.95573449e+00,  8.36095095e+00,  1.56596515e+00],
       [-4.83350689e+04, -1.55297880e+03,  1.36739782e+03, ...,
         5.47869173e-01,  1.90822481e+01, -9.01265491e+00],
       ...,
       [-1.48783294e+04,  6.96969037e+04,  1.19411161e+02, ...,
         5.91789664e+00,  3.05180040e-01, -2.55471741e+00],
       [-4.63944666e+04,  7.47445946e+04,  7.24489895e+02, ...,
         4.35461703e+01,  5.83262723e+00,  4.07751112e+00],
       [-1.25036195e+04,  3.88659341e+04, -2.11208421e+03, ...,
         4.58237548e+00,  6.32977683e-01,  1.00538759e+00]])
```

Figure 50 Using PCA to get 13 components

The purpose of using 13 components is to match the number of clusters used. The below Figure shows the code written to create a recommendation based on giving the neighborhood you currently live in and in return it will recommend a similar neighborhood.

```
def recommend():
    neighborhood_name =  'Manhattan, New York, NY'
    num_recommendations = 15
    recommendations = get_recommendations(df_pca, neighborhood_name, num_recommendations)
    return recommendations
```

Figure 51 Code used to make recommendations

```
recommend()
```

```
[('Fair Meadows, West Chicago, IL', 28498.173175822187),
 ('Washington Park, Chicago, IL', 29411.433970779424),
 ('Madison Square, Chicago, IL', 30313.469969383303),
 ('Bronzeville, Chicago, IL', 30923.485506637116),
 ('Armour Square, Chicago, IL', 33055.076989112255),
 ('West Garfield Park, Chicago, IL', 36708.64362658058),
 ('Riverdale, Chicago, IL', 38081.89914872363),
 ('Jeffery Manor, Chicago, IL', 38671.04682465364),
 ('Chinatown, Chicago, IL', 39675.58484430577),
 ('Burnside, Chicago, IL', 40319.1259946908),
 ('Oakland, Chicago, IL', 40655.502457405695),
 ('Douglas, Chicago, IL', 40953.29306045632),
 ('Cabrini-Green (Cabrini Green), Chicago, IL', 41191.06208830154),
 ('Canaryville, Chicago, IL', 41545.5450628833),
 ('North Center, North Chicago, IL', 44102.21725597759)]
```

Figure 52 Recommendation of neighborhoods similar of Manhattan New York

Based providing your current neighborhood “Manhattan New York” the recommendation system will recommend the above neighborhoods that can be noted as similar to “Manhattan”.

V. Conclusion

The project analyzes neighborhood similarity based on two data sources: socioeconomic data obtained from city-data.com and venues data from Foursquare. The socioeconomic data web scraping code can pull data for other neighborhoods. In fact, some neighborhoods in Chicago area belong to other cities like West Chicago. Therefore, it would be nice to include these into the analysis in the future work.

The data needed heavy cleaning as it contained nonexistent neighborhoods, missing data for neighborhoods, or neighborhoods consisting of a single household. Some socioeconomic data were not trustable either, such as neighborhoods with average number of vehicles exceeding 10 apartments.

The venues data was usable; however, we wish Foursquare had returned more venues as the data was very sparse and difficult to use for clustering. In future work, it may also be useful to include data pertaining to schools, universities, and such.

Perhaps, another source of data worth considering is police and nuisance reports, accidents data, or may be sexual offender’s residences.

VI. References

[1] Bureau, US Census. “Calculating Migration Expectancy Using ACS Data.” *The United States Census Bureau*, 19 July 2020, www.census.gov/topics/population/migration/guidance/calculating-migration-expectancy.html.

[2] Wu, Yuting, et al. “Neighborhood Matching Network for Entity Alignment.” *ACL Anthology*, www.aclweb.org/anthology/2020.acl-main.578.

[3] M. D. Porter and R. Smith, "Network neighborhood analysis," 2010 IEEE International Conference on Intelligence and Security Informatics, Vancouver, BC, 2010, pp. 31-36, doi: 10.1109/ISI.2010.5484781.

[4] Stats about all US cities - real ESTATE, relocation info, crime, house prices, cost of living, Races, home VALUE Estimator, recent sales, INCOME, photos, Schools, Maps, Weather, neighborhoods, and more. (n.d.). Retrieved February 14, 2021, from http://www.city-data.com/

[5] Albany Park neighborhood in Chicago, Illinois (il), 60625, 60630 detailed profile. (n.d.). Retrieved February 14, 2021, from http://www.city-data.com/neighborhood/Albany-Park-Chicago-IL.html

[6] New York neighborhood in NEW YORK, OHIO (oh), 44130, 44135, 44142 detailed profile. (n.d.). Retrieved February 14, 2021, from http://www.city-data.com/neighborhood/Brook-Park-Brook-Park-OH.html

[7] Ganesh, S. (2020, June 23). Popular places near me - data visualization using python and FourSquare api. Retrieved February 14, 2021, from https://towardsdatascience.com/popular-places-near-me-data-visualization-using-python-and-foursquare-api-4d1683cd62d1

[8] “Where matters.,” FourSquare. Retrieved February 14, 2021, from https://foursquare.com/

[9] Dong, L., Ratti, C., & Zheng, S. (2019, July 30). Predicting neighborhoods' socioeconomic attributes using restaurant data. Retrieved February 14, 2021, from https://www.pnas.org/content/116/31/15447