New_Facility_Location_Selection_Report_v2

December 30, 2019

1 Final Project:

1.1 New Facility Location Selection

1.1.1 by: Jeffrey Dupree

1.2 Table of contents

1.3 Introduction: Business Problem

The owner of several successful gyms wants to open a new facility in Tampa, FL. They want to ensure that the gym's location is in an area not already saturated with gyms and other businesses that might compete with a new gym. This initial analysis will be to determine to a neighborhood level, where to consider placing the new gym facility. Later analysis and research of available real estate will be required to select the final location. That is beyond the scope of this analysis.

In order to conduct this analysis, we must collect: * Zip Codes in Tampa, FL * Zip Code locations (latitude/longitude) * Zip Code boundaries * Business type and frequency

[1]:
```
%%capture

# If you don't have these packages available, uncomment the appropriate lines␣
 ↪below to install them.
import sys
#!{sys.executable} -m pip install beautifulsoup4
#!{sys.executable} -m pip install lxml
#!{sys.executable} -m pip install requests
#!{sys.executable} -m pip install --upgrade geopy==1.20.0 #Must specify version␣
 ↪1.20.0 for country_codes argument to work.
#!{sys.executable} -m pip install geojson
#!conda install -c conda-forge folium=0.5.0 --yes

from bs4 import BeautifulSoup #For scaping and rendering webpages.
```

```
import requests
import pandas as pd
import numpy as np
import re #This will allow use of regular expressions (regex)
from tqdm import tqdm   #This will allow a progress bar to show that there is␣
 ↪progress being made. This is helpful when an
                        #iterative process may take more than a few seconds.
import geopy
from geopy.geocoders import Nominatim
from geopy.extra.rate_limiter import RateLimiter #This will get around getting␣
 ↪shut down for too many request errors.
from functools import partial #This will allow multiple arguments to be passed␣
 ↪to RateLimiter and geocode.
import json #Library to handle JSON files
from pandas.io.json import json_normalize #Tranform JSON file into a pandas␣
 ↪dataframe
import matplotlib.cm as cm #Matplotlib and associated plotting modules
import matplotlib.colors as colors
import folium #Map rendering library
import urllib
import geojson
from geojson import Feature, Point, FeatureCollection, MultiPolygon
```

1.4 Data

1.4.1 Zip Codes

To begin with, the analysis will need specific Zip Code data for Tampa, FL.

Step one: Identify the list of Zip Codes that correspond to Tampa, FL. For that, this notebook will scrape information from a ZIP-CODES.COM page https://www.zip-codes.com/state/fl.asp#zipcodes to create a dataframe consisting of the Zip Code, the City name, County name and the Zip Code type. Use the BeautifulSoup package to scrape the information from the webpage. I used the lxml parsing method, but you can use any you like. Find the table using `soup.find` from BeautifulSoup. Initially, the analyst must display the structure and content of the table (a portion shown below). Once the analyst understands the structure, they can develop the logic required to extract the desired elements in the next steps.

[2]:
```
source = requests.get('https://www.zip-codes.com/state/fl.asp#zipcodes').text
```

[3]:
```
soup = BeautifulSoup(source, 'lxml')
```

[4]:
```
table = soup.find(id="tblZIP")
print(table.prettify()[0:487]) # Only printing the first 987 characters in the␣
 ↪string as an example. Remove [0:487] to see entire output.
```

```
<table border="0" cellpadding="0" cellspacing="0" class="statTable" id="tblZIP"
title="All Florida ZIP Codes, City, County, Classification, and Area Codes."
```

```
width="99%">
 <tr>
  <td class="label" title="All ZIP Codes for Florida">
   <strong>
    ZIP Code
   </strong>
  </td>
  <td class="info" title="The official city name as designated by the USPS.">
   <strong>
    City
   </strong>
  </td>
  <td class="info" title="The primary county or parish this ZIP Code serves.">
   <strong>
```

[5]:
```
table_rows = table.find_all('tr')

res = []
for tr in table_rows:
    td = tr.find_all('td')
    row = [tr.text.strip() for tr in td if tr.text.strip()]
    if row:
        res.append(row)

# Label the columns.
df = pd.DataFrame(res[1:], columns=['Zip_Code','City','County','Type'])

# Remove the text 'Zip Code' from the records in the Zip Code column.
df['Zip_Code'] = df['Zip_Code'].str[-5:]

# Select only the Zip Codes for Tampa, FL.
df = df.loc[df['City'] == "Tampa"]
```

[6]:
```
# Remove rows with Type = "P.O. Box" and "Unique", and reset the index to start␣
 ↪at 0
df = df[df.Type == 'Standard']
df = df.reset_index(drop=True)
```

Now a pandas dataframe needs to be created. This will require looping through the elements from the table and assigning the elements to a list. The list can then be made into a dataframe using `pd.DataFrame`. The columns will need header names. I manually assigned these instead of pulling them from the BeautifulSoup object `table`. Next remove the rows where the type is "P.O. Box". The first five rows of the resulting dataframe look like this.

[7]:
```
df.head()
```

[7]:

	Zip_Code	City	County	Type
0	33602	Tampa	Hillsborough	Standard
1	33603	Tampa	Hillsborough	Standard
2	33604	Tampa	Hillsborough	Standard
3	33605	Tampa	Hillsborough	Standard
4	33606	Tampa	Hillsborough	Standard

Step two: The locations of the Zip Codes (latitude and longitude) will need to be collected. This will be accomplished through Nominatim in the Geopy library. This leverages the OpenStreetMap (OSM) dataset application programming interface (API) to geolocate each Zip Code. This adds two rows (location, point) to the dataframe. The first five rows are shown here.

[8]:
```
# @hidden_cell
user_agent = "JGD_20191006"
```

[9]:
```
tqdm.pandas(disable=True) #Change 'disable=True' to 'disable=False' to show␣
 ↪progress bar.
geolocator = Nominatim(user_agent=user_agent)
geocode = RateLimiter(geolocator.geocode, min_delay_seconds=0.5, max_retries=2,␣
 ↪error_wait_seconds=5.0, swallow_exceptions=True,␣
 ↪return_value_on_exception=None)
df['location'] = df['Zip_Code'].progress_apply(partial(geocode,␣
 ↪country_codes='us')) #Commented this out to troubleshoot. If uncommented␣
 ↪removed '))' precedimg '#'.
df['point'] = df['location'].apply(lambda loc: tuple(loc.point) if loc else␣
 ↪None)
df.head()
```

```
C:\Users\JeffDupree\Anaconda3\lib\site-packages\tqdm\std.py:648: FutureWarning:
The Panel class is removed from pandas. Accessing it from the top-level
namespace will also be removed in the next version
  from pandas import Panel
```

[9]:

	Zip_Code	City	County	Type	\
0	33602	Tampa	Hillsborough	Standard	
1	33603	Tampa	Hillsborough	Standard	
2	33604	Tampa	Hillsborough	Standard	
3	33605	Tampa	Hillsborough	Standard	
4	33606	Tampa	Hillsborough	Standard	

	location	\
0	(Ybor City, Tampa, Hillsborough County, Florid…	
1	(Tampa, Hillsborough County, Florida, 33603, U…	
2	(Sulphur Springs, Tampa, Hillsborough County, …	
3	(East Ybor, Tampa, Hillsborough County, Florid…	
4	(Hyde Park, Tampa, Hillsborough County, Florid…	

```
                                        point
0                (27.9516574, -82.449638, 0.0)
1  (27.9823952329372, -82.4629461755015, 0.0)
2               (28.0127051, -82.4665599, 0.0)
3                 (27.96589, -82.4209639, 0.0)
4               (27.9341317, -82.4680636, 0.0)
```

[10]:
```
df[['Latitude','Longitude','3']] = pd.DataFrame(df['point'].tolist(), index=df.
↪index)
df = df.drop(columns=['point','3'])
```

Now the latitude and longitude values for each of the postal codes are separated out into respective columns. Next we take the first portion of the location string, removing everything after the first comma, then renaming the column "Neighborhood". The dataframe now looks like this.

[11]:
```
df['location'] = df['location'].astype(str)
df['location'] = df['location'].str.split(", ").str[0].tolist()
df = df.rename(columns={"location": "Neighborhood"})
```

[12]:
```
df.head()
```

[12]:
```
  Zip_Code   City        County      Type     Neighborhood   Latitude  \
0    33602  Tampa  Hillsborough  Standard        Ybor City  27.951657
1    33603  Tampa  Hillsborough  Standard            Tampa  27.982395
2    33604  Tampa  Hillsborough  Standard  Sulphur Springs  28.012705
3    33605  Tampa  Hillsborough  Standard        East Ybor  27.965890
4    33606  Tampa  Hillsborough  Standard        Hyde Park  27.934132

   Longitude
0 -82.449638
1 -82.462946
2 -82.466560
3 -82.420964
4 -82.468064
```

Step three: The last feature of Zip Code data needed are the boundaries of each Zip Code. These will be stored as latitudes and longitudes for the verices of polygons representing areas corresponding to each Zip Code. This data is downloaded as a GeoJSON file from https://opendata.arcgis.com/datasets/d356e19e0fb34524b54d189fafb0d675_0.geojson.

1.4.2 Business Data

Once the Zip Code data are collected, we need to collect the data on the surrounding businesses. We use the Foursquare API to collect data about the businesses near each Zip Code loaction.

1.5 Methodology

Locate Zip Codes Lacking Gyms We can start by visualizing the location of each zip code (based on the coordinates associated with it). The very first visualization is to plot the locations associated with each zip code to ensure that they fall within the intended area. These locations can also be labeled with information from the dataframe to make the graphic interactive. Selecting a point on the map reveals the Latitude, Longitude, and Neighborhood associated with that point.

[13]:
```
# Create map of Tampa using latitude and longitude values
tampa = geolocator.geocode({"state": "fl", "city": "tampa"})
map_tampa = folium.Map(location=[tampa.latitude, tampa.longitude],␣
 ↪zoom_start=11)

# add markers to map
for lat, lng, hood in zip(df['Latitude'], df['Longitude'], df['Neighborhood']):
    label = '{}'.format(hood)
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker(
        [lat, lng],
        radius=5,
        popup=label,
        color='blue',
        fill=True,
        fill_color='#3186cc',
        fill_opacity=0.7,
        parse_html=False).add_to(map_tampa)

map_tampa
```

[13]:
```
<folium.folium.Map at 0x2ba6df071c8>
```

[14]:
```
# @hidden_cell
CLIENT_ID = 'MAI43NUPMV0YXXNFKS2XVGUPBMIB5SBO5T5W5FV4ZND2VTJW' # your␣
 ↪Foursquare ID
CLIENT_SECRET = 'V1POSAELWQONIURPOW2C43LH2FTO5NJOVGYQXMSD2OGRLEND' # your␣
 ↪Foursquare Secret
VERSION = '20180604' # Foursquare API version
```

A query of the Foursquare API returns the top 150 venues within 1000 meters of the zip code locations. The query is passed as a url using the `get()` command and returns a json formatted response. After reviewing the structure of the JSON, a function must be created to extract the venue category types associated with each zip code. The venues can then be placed in a table with the venue name, category, latitude, and longitude as columns. The first five rows of the table are shown here.

[15]:
```
search_lat = df.Latitude[0]
search_lon = df.Longitude[0]
LIMIT = 150 # Limit of number of venues returned by Foursquare API
```

```
radius = 1000 # Define radius in meters

# Create URL
url = 'https://api.foursquare.com/v2/venues/explore?
 ↪&client_id={}&client_secret={}&v={}&ll={},{}&radius={}&limit={}'.format(
    CLIENT_ID,
    CLIENT_SECRET,
    VERSION,
    search_lat,
    search_lon,
    radius,
    LIMIT)
```

[16]:
```
results = requests.get(url).json()
#results # Uncomment to see structure of JSON returned by get(url).json().
```

[17]:
```
# Function that extracts the category of the venue
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']
```

[18]:
```
venues = results['response']['groups'][0]['items']
#venues # Uncomment to see the results, potentially large.
```

[19]:
```
# Flatten JSON
nearby_venues = json_normalize(venues)
```

[20]:
```
# Filter columns
filtered_columns = ['venue.name', 'venue.categories', 'venue.location.lat',␣
 ↪'venue.location.lng']
nearby_venues = nearby_venues.loc[:, filtered_columns]
```

[21]:
```
# Filter the category for each row
nearby_venues['venue.categories'] = nearby_venues.apply(get_category_type,␣
 ↪axis=1)
```

[22]:
```
# Clean columns
nearby_venues.columns = [col.split(".")[-1] for col in nearby_venues.columns]
```

[23]:
```
nearby_venues.head()
```

[23]:

	name	categories	lat	lng
0	Pour House at Grand Central	Bar	27.951357	-82.447740
1	Crunch - Channelside	Gym / Fitness Center	27.951152	-82.447940
2	Cena	Italian Restaurant	27.951569	-82.447869
3	Publix - Channelside	Grocery Store	27.952128	-82.448741
4	City Dog Cantina	Mexican Restaurant	27.951118	-82.447726

We then create a function that uses the Foursquare API to find the nearby venues for all of the neighborhoods, by zip code. The `getNearbyVenues` function can then be applied to the dataframe to create a dataframe of the venues near the grid associated with each zip code. The first five rows of the resulting dataframe will look like this.

[24]:
```
def getNearbyVenues(names, latitudes, longitudes, radius=1000):

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

        # 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 = ['Zip_Code',
                  'Zip Latitude',
                  'Zip Longitude',
                  'Venue',
```

```
                  'Venue Latitude',
                  'Venue Longitude',
                  'Venue Category']

    return(nearby_venues)
```

[26]:
```
tampa_venues = getNearbyVenues(names=df['Zip_Code'],
                               latitudes=df['Latitude'],
                               longitudes=df['Longitude']
                              )
```

[27]:
```
#print(tampa_venues.shape)
tampa_venues.head()
```

[27]:
```
   Zip_Code  Zip Latitude  Zip Longitude                        Venue  \
0     33602     27.951657     -82.449638  Pour House at Grand Central
1     33602     27.951657     -82.449638         Crunch - Channelside
2     33602     27.951657     -82.449638                         Cena
3     33602     27.951657     -82.449638         Publix - Channelside
4     33602     27.951657     -82.449638             City Dog Cantina

   Venue Latitude  Venue Longitude        Venue Category
0       27.951357       -82.447740                   Bar
1       27.951152       -82.447940  Gym / Fitness Center
2       27.951569       -82.447869    Italian Restaurant
3       27.952128       -82.448741         Grocery Store
4       27.951118       -82.447726    Mexican Restaurant
```

Once all of the venues have been associated with neighborhoods by proximity, the frequency of venue types can be determined. However, before the frequency of each venue can be calculated, a list of the unique venue categories must be created and evaluated. This can be a very long list of more than 100 categories. Many of these categories can be very similar. For example the categories "Gym / Fitness Center" and "Gym" appear in the list. These two categories could be considered to be the same. Another venue that would compete with a gym is 'Military Base'. Military bases have gyms and fitness centers for military members at no cost. This could reduce the need for another gym in the area. We will need to recode any gym-like categories with a common category name (i.e., gym). This will require examining the list of unique categories and creating a list of the categories that should be recoded. We use one-hot encoding to determine if a venue type exists in a neighborhood. One-hot encoding will create a column for each of the unique categories, and assign a value of 1 if that venue type exists in the neighborhood or 0 otherwise for each row. A portion of that table would look like this.

[28]:
```
#tampa_venues['Venue Category'].unique()
```

[29]:
```
gym = ['Gym / Fitness Center', 'Park', 'Martial Arts Dojo', 'Gym', 'Pool',␣
 ↪'Tennis Court', 'Disc Golf', 'Volleyball Court',
```

```
      'Soccer Field', 'Basketball Court', 'Yoga Studio', 'College Basketball␣
↪Court', 'College Gym','College Track',
      'Dance Studio', 'Military Base', 'Athletics & Sports', 'Golf Course',␣
↪'Baseball Field', 'Trail', 'Hockey Arena',
      'Hockey Field', 'Track', 'Water Park', 'Outdoors & Recreation', 'State /␣
↪Provincial Park', 'Playground']
```

[30]:
```
tampa_venues['Venue Category'].replace(to_replace = gym, value = "Gym", inplace␣
↪= True)
```

[31]:
```
# One hot encoding
tampa_onehot = pd.get_dummies(tampa_venues[['Venue Category']], prefix="",␣
↪prefix_sep="")

# Add zip code column back to dataframe
tampa_onehot['Zip_Code'] = tampa_venues['Zip_Code']

# Move zip code column to the first column
fixed_columns = [tampa_onehot.columns[-1]] + list(tampa_onehot.columns[:-1])
tampa_onehot = tampa_onehot[fixed_columns]

tampa_onehot.head()
```

[31]:

	Zip_Code	Accessories Store	Airport	Airport Lounge	Airport Service	\
0	33602	0	0	0	0	
1	33602	0	0	0	0	
2	33602	0	0	0	0	
3	33602	0	0	0	0	
4	33602	0	0	0	0	

	American Restaurant	Antique Shop	Aquarium	Arcade	Art Gallery	…	\
0	0	0	0	0	0	…	
1	0	0	0	0	0	…	
2	0	0	0	0	0	…	
3	0	0	0	0	0	…	
4	0	0	0	0	0	…	

	Vegetarian / Vegan Restaurant	Video Game Store	Video Store	\
0	0	0	0	
1	0	0	0	
2	0	0	0	
3	0	0	0	
4	0	0	0	

	Vietnamese Restaurant	Waste Facility	Wine Bar	Wings Joint	\
0	0	0	0	0	
1	0	0	0	0	

```
2                     0               0         0            0
3                     0               0         0            0
4                     0               0         0            0

   Women's Store  Zoo  Zoo Exhibit
0              0    0            0
1              0    0            0
2              0    0            0
3              0    0            0
4              0    0            0

[5 rows x 178 columns]
```

With the one-hot encoded data, we can determine the frequency with which each venue type occurs in each borough. This results in a dataframe with a column for each unique venue type and a row for each unique borough.

[32]:
```
tampa_grouped = tampa_onehot.groupby('Zip_Code').mean().reset_index()
tampa_grouped.head()
```

[32]:
```
  Zip_Code  Accessories Store  Airport  Airport Lounge  Airport Service  \
0    33602                0.0      0.0             0.0              0.0
1    33603                0.0      0.0             0.0              0.0
2    33604                0.0      0.0             0.0              0.0
3    33605                0.0      0.0             0.0              0.0
4    33606                0.0      0.0             0.0              0.0

   American Restaurant  Antique Shop  Aquarium  Arcade  Art Gallery  …  \
0             0.033333      0.000000  0.011111     0.0     0.011111  …
1             0.000000      0.055556  0.000000     0.0     0.111111  …
2             0.025000      0.000000  0.000000     0.0     0.000000  …
3             0.083333      0.000000  0.000000     0.0     0.000000  …
4             0.040000      0.000000  0.000000     0.0     0.000000  …

   Vegetarian / Vegan Restaurant  Video Game Store  Video Store  \
0                          0.000               0.0         0.00
1                          0.000               0.0         0.00
2                          0.025               0.0         0.00
3                          0.000               0.0         0.00
4                          0.000               0.0         0.02

   Vietnamese Restaurant  Waste Facility  Wine Bar  Wings Joint  \
0                    0.0             0.0     0.000          0.0
1                    0.0             0.0     0.000          0.0
2                    0.0             0.0     0.025          0.0
3                    0.0             0.0     0.000          0.0
4                    0.0             0.0     0.020          0.0
```

```
   Women's Store    Zoo  Zoo Exhibit
0           0.00  0.000     0.011111
1           0.00  0.000     0.000000
2           0.00  0.025     0.250000
3           0.00  0.000     0.000000
4           0.02  0.000     0.000000

[5 rows x 178 columns]
```

Next we will determine the five most frequent venues within a borough to describe a neighborhood 'type', and group the borough by type symilarity. We begin by creating a function that will return the most common venues for each zip code.

[33]:
```
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]
```

[34]:
```
num_top_venues = 5

indicators = ['st', 'nd', 'rd']

# Create columns according to number of top venues
columns = ['Zip_Code']
for ind in np.arange(num_top_venues):
    try:
        columns.append('{}{} Most Common Venue'.format(ind+1, indicators[ind]))
    except:
        columns.append('{}th Most Common Venue'.format(ind+1))

# Create a new dataframe
zip_venues_sorted = pd.DataFrame(columns=columns)
zip_venues_sorted['Zip_Code'] = tampa_grouped['Zip_Code']

for ind in np.arange(tampa_grouped.shape[0]):
    zip_venues_sorted.iloc[ind, 1:] = return_most_common_venues(tampa_grouped.
 ↪iloc[ind, :], num_top_venues)

# Define a function to color the text red if a venue type is "Gym"
def color_gym_red(val):
    color = 'red' if val == "Gym" else 'black'
    return 'color: %s' % color

# Display the first five records of the dataframe, with "Gym" highlighted red.
zip_venues_sorted.head().style.applymap(color_gym_red)
```

[34]:
```
<pandas.io.formats.style.Styler at 0x2ba6fb6e508>
```

Now that we can see what the five most common venues are in each Zip Code, we can eliminate those Zip Codes with 'gym' type venues in the top five.

[35]:
```
zip_venues_reduced = zip_venues_sorted[(zip_venues_sorted['1st Most Common␣
↪Venue'] != 'Gym') & (zip_venues_sorted['2nd Most Common Venue'] != 'Gym') &
                                        (zip_venues_sorted['3rd Most Common␣
↪Venue'] != 'Gym') & (zip_venues_sorted['4th Most Common Venue'] != 'Gym') &
                                        (zip_venues_sorted['5th Most Common␣
↪Venue'] != 'Gym')]
zip_venues_reduced
```

[35]:

	Zip_Code	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	\
5	33607	Scenic Lookout	Harbor / Marina	Food Truck	
6	33609	Clothing Store	Women's Store	Sandwich Place	
7	33610	Grocery Store	Discount Store	Restaurant	
8	33611	Turkish Restaurant	Sandwich Place	Korean Restaurant	
15	33618	Pizza Place	American Restaurant	Massage Studio	
19	33625	Fast Food Restaurant	Nail Salon	Gas Station	
20	33626	Insurance Office	Home Service	Zoo Exhibit	

	4th Most Common Venue	5th Most Common Venue
5	Karaoke Bar	Mobile Phone Shop
6	Lingerie Store	Department Store
7	Video Store	Spa
8	Food	Motel
15	Coffee Shop	Hobby Shop
19	Big Box Store	Coffee Shop
20	Event Service	Fountain

Now the list only includes Zip Codes where 'gym' type venues are not one of the five most frequent venue types. We can sort this list by descending frequency of gyms. Where the gym frequencies are equal, records are sorted by Zip_Code in ascending order.

[36]:
```
index = zip_venues_reduced.index
locations = tampa_grouped[['Zip_Code','Gym']].iloc[index].
↪sort_values(by=['Gym'], ascending=True)
locations
```

[36]:

	Zip_Code	Gym
5	33607	0.000000
6	33609	0.000000
7	33610	0.000000
19	33625	0.000000
20	33626	0.000000
15	33618	0.025641

```
8     33611  0.066667
```

Now that we have the reduced list of zip codes, we join it to our location dataframe, rename the 'Gym' column as 'Gym Frequency', and reset the indeces.

[37]:
```
cols = ['Zip_Code']
locations = locations.join(df.set_index(cols), on=cols)
locations = locations.rename(columns={"Gym": "Gym Frequency"}).
 ↪reset_index(drop=True)
locations
```

[37]:

	Zip_Code	Gym Frequency	City	County	Type	Neighborhood	\
0	33607	0.000000	Tampa	Hillsborough	Standard	Tampa	
1	33609	0.000000	Tampa	Hillsborough	Standard	Palma Ceia	
2	33610	0.000000	Tampa	Hillsborough	Standard	Ybor City	
3	33625	0.000000	Tampa	Hillsborough	Standard	Hillsborough County	
4	33626	0.000000	Tampa	Hillsborough	Standard	Hillsborough County	
5	33618	0.025641	Tampa	Hillsborough	Standard	Mullis City	
6	33611	0.066667	Tampa	Hillsborough	Standard	Palma Ceia	

	Latitude	Longitude
0	27.973131	-82.585196
1	27.944813	-82.536276
2	27.977944	-82.442975
3	28.068327	-82.557302
4	28.057031	-82.610797
5	28.039589	-82.508293
6	27.880332	-82.498916

Now we can display the locations on a map. Selecting a marker on the map will display that zip code and the frequency of 'gym' type venues within 1km of the zip code central point.

[38]:
```
# Create map
map_locations = folium.Map(location=[tampa.latitude, tampa.longitude],␣
 ↪zoom_start=11)

# Set color scheme for the clusters
x = np.arange(locations.shape[0])
ys = [i + x + (i*x)**2 for i in range(locations.shape[0])]
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 = []
count = 0
for lat, lon, poi, freq, nbh in zip(locations['Latitude'],␣
 ↪locations['Longitude'], locations['Zip_Code'],
```

```
                                    locations['Gym Frequency'],␣
↪locations['Neighborhood']):
    label = folium.Popup('Neigborhood: ' + str(nbh) + ' / Zip Code: ' +␣
↪str(poi) + ' / Gym Frequency: ' + str(freq),
                         parse_html=False,)
    count = count + 1
    folium.CircleMarker(
        [lat, lon],
        radius=7,
        popup=label,
        color=rainbow[count-1],
        fill=True,
        fill_color=rainbow[count-1],
        fill_opacity=0.7).add_to(map_locations)

map_locations
```

[38]:
```
<folium.folium.Map at 0x2ba6e076608>
```

[39]:
```
url = 'https://opendata.arcgis.com/datasets/d356e19e0fb34524b54d189fafb0d675_0.
↪geojson'
with urllib.request.urlopen(url) as url:
    plt = json.loads(url.read().decode())
#plt #If uncommented the JSON will be displayed, and is potentially very large.
```

Using the GeoJSON file from https://opendata.arcgis.com/datasets/d356e19e0fb34524b54d189fafb0d675_0.geojso
polygons for the Zip Codes of intereset can be defined using the latitude and longitude coordinates. Below we create a list of coordinates for both latitudes and longitudes, then place these lists at the end of the dataframe.

[40]:
```
for i in reversed(range(len(plt['features']))):
    count = locations.shape[0]
    for j in range(locations.shape[0]):
        if plt['features'][i]['properties']['Zip_Code'] !=␣
↪locations['Zip_Code'][j]:
            count = count - 1
            if count == 0:
                del plt['features'][i]
                break
```

1.6 Results

Now the polygons for the areas represented by the zip code can be overlaid on the map.

[41]:
```
# Create map
map_test = folium.Map(location=[tampa.latitude, tampa.longitude], zoom_start=11)
```

```
# Add polygons to the map
for i in range(len(plt['features'])):
    ZIP = plt['features'][i]['properties']['Zip_Code']
    neighborhood = locations.Neighborhood[locations[locations.Zip_Code == ZIP].
 ↪index[0]]
    zip_code = locations.Zip_Code[locations[locations.Zip_Code == ZIP].index[0]]
    geojson = folium.GeoJson(
        plt['features'][i],
        name=neighborhood
    )
    popup = folium.Popup(neighborhood + " " + zip_code)
    popup.add_to(geojson)
    geojson.add_to(map_test)

folium.LayerControl().add_to(map_test)

map_test
```

[41]: <folium.folium.Map at 0x2ba700a94c8>

1.7 Discussion

Using this method the analyst is able to quickly gather and display location and venue information for the area of interest. With this data the analyst can categorize the areas by the types of venues in that are and the frequency with which they occur. This allows for a cursory analysis to narrow down the locations that may be good choices for a new gym facility.

There are some drawbacks to this application. Primarily that the search for venues is conducted in a circular area of radius 1km from the coordinates pulled from the website https://www.zip-codes.com/state/fl.asp#zipcodes. These coordinates do not alwasy correspond to the geographic center of the area. If the coordinates map to a location within the zip code area that is in a remote section, there may not be many venues within 1km of the point. Also, some of the points may be less than 1km from the boundary. This may result in some venues from other zip codes being included with multiple zip codes.

However, the strength of this methodology is that it is dynamic. As more venue infromation is added or modified within the FourSquare platform, the results of this analysis will take those changes into account when rerun.

1.8 Conclusion

[42]:
```
from IPython.display import Markdown
zips = len(plt['features'])
#Markdown("# Title")
Markdown("""
At the time of this model run, there were {zips} zip codes that met the␣
 ↪criteria for the new location. The customer can now focus their location␣
 ↪serach to a few zip codes, saving time and money.
```

```
""".format(zips=zips))
```

[42]:

At the time of this model run, there were 7 zip codes that met the criteria for the new location. The customer can now focus their location serach to a few zip codes, saving time and money.