Locating a New Grocery Store

Erika Fox and Raza Lamb

In this assignment, we will be identifying potential new locations for Whole Foods in New York state. To do this, we will be using a dataset of all retail food stores in New York provided by the NY Department of Agriculture and Markets as well as a dataset of demographic data from the American Community Survey (ACS).

Exercise 1

Importing data

First, we just imported the required packages, then read in the demographic and retail data. Subequently, we plotted both datasets below. As is visible, the demographic data represents the state of New York, and the retail data is represented as points.

In [ ]:
```
import pandas as pd
import geopandas as gpd
import matplotlib
```

In [ ]:
```
census = gpd.read_file("ny_census_blockgroups_2019/ny_census_blockgroups_2019.shp")
```

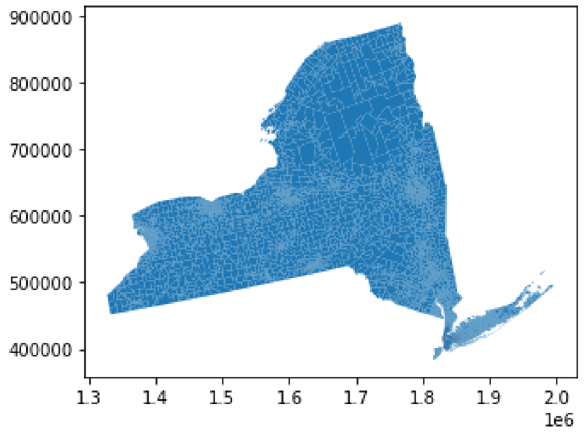
In [ ]:
```
food = gpd.read_file("ny_retail_food/ny_retail_food.shp")
```

In [ ]:
```
food.shape
```

Out[ ]:
```
(28472, 15)
```

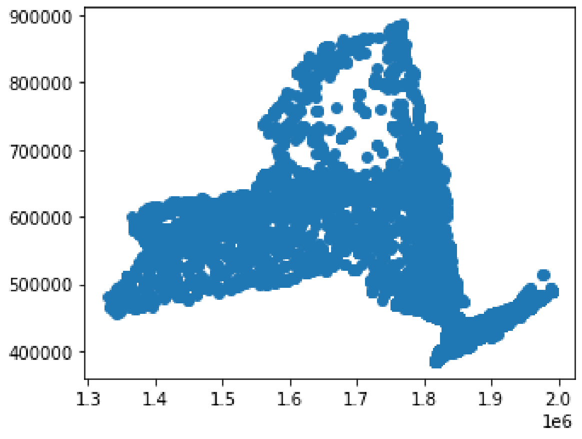
In [ ]:
```
census.plot()
```

Out[ ]:
```
<AxesSubplot:>
```



In [ ]:
```
food.plot()
```

Out[ ]:
```
<AxesSubplot:>
```



Exercise 2

Establishing Whole Foods, where are they located?

Next, we attempted to create a subset of the retail data containing only Whole Foods. After investigateion, it appeared that the Whole Foods are referred to with a few different names including:

- WHOLE FOODS MARKET GROUP INC
- WHOLE FOODS MARKET GROUP IN
- WHOLE FOODS MARKET GROUP, INC.

In order to capture all of these options, we utilized the regular expression: `WHOLE FOODS MARKET GROUP,? INC?` . We confirmed that this worked (all 25 Whole Foods were captured, including the 3 different names), and then plotted this data. Based on the plot, it appears that most Whole Foods are contained towards New York city. To confirm that, we simply calculated value counts for the city of this reduced data set. Visibily, more than half (14) of all Whole Foods in New York state are in New York City (including Brooklyn).

In [ ]:
```
food[food["entity_nam"].str.contains("WHOLE.*FOODS")].entity_nam.value_counts()
```

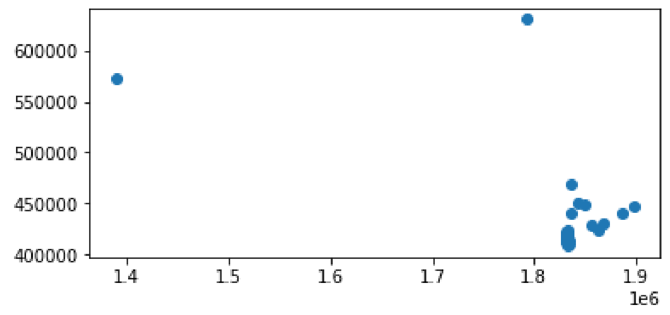
Out[ ]:
```
WHOLE FOODS MARKET GROUP INC         23
WHOLE FOODS MARKET GROUP IN           1
ES WHOLESOME FOODS OF BROOKLYN        1
WHOLE FOODS MARKET GROUP, INC.        1
WHOLESOME FOODS INC                   1
GARY NULLS UPTOWN WHOLE FOODS INC     1
HAMILTON WHOLE FOODS LTD              1
BUFFALO WHOLESALE FOODS INC           1
DOWN TO EARTH WHOLE FOODS INC         1
Name: entity_nam, dtype: int64
```

In [ ]:
```
wholefoods = food[food["entity_nam"].str.contains("WHOLE FOODS MARKET GROUP,? INC?")]
wholefoods.entity_nam.value_counts()
```

Out[ ]:
```
WHOLE FOODS MARKET GROUP INC      23
WHOLE FOODS MARKET GROUP IN        1
WHOLE FOODS MARKET GROUP, INC.     1
Name: entity_nam, dtype: int64
```

In [ ]:
```
wholefoods.plot()
```

Out[ ]:
```
<AxesSubplot:>
```



In [ ]:
```
wholefoods.city.value_counts()
```

Out[ ]:
```
NEW YORK         11
BROOKLYN          3
GARDEN CITY       1
COMMACK           1
CHAPPAQUA         1
BUFFALO           1
ALBANY            1
PORT CHESTER      1
YONKERS           1
WHITE PLAINS      1
LAKE GROVE        1
JERICHO           1
MUNSEY PARK       1
Name: city, dtype: int64
```

Exercise 3

Creating GeoDataFrame with Competing Grocery Stores

Here, we now created another subset of the retail GeoDataFrame with potential Whole Foods competitors (i.e. other supermarkets). Here we included Walmart, ACME Markets (Albertson's), Trader Joe's, ALDI, and any stores identifying themselves as supermarkets or super markets.

Similar to how we treated Whole Foods, first we had to examine the different names that each of these competitors was referrred to as. This code is omitted for brevity, but in short, the regular expressions that match each competitor are listed below. Then, we concatenated each so that we had one GeoDataFrame with all potential Whole Foods competitors—this included 697 supermarkets.

In [ ]:
```
walmart = food[(food["entity_nam"].str.contains("WAL-?MART"))]
acme = food[food["entity_nam"].str.contains("ACME MARKETS INC")]
trader = food[food["entity_nam"].str.contains("TRADER JOE")]
aldi = food[food["entity_nam"].str.contains("ALDI INC")]
superm = food[food["entity_nam"].str.contains("SUPER ?MARKET")]
```

In [ ]:
```
supermarkets = pd.concat([walmart, acme, trader, aldi, superm])
supermarkets.shape
```

Out[ ]:
```
(697, 15)
```

Exercise 4

Joining Census and Whole Foods Data

Now, we wanted to join the two dataframes, finding the Whole Foods nearest to each census block. To do this, we used the method `.sjoin_nearest` with the census dataframe as the "left" dataframe with a left join, so that each census block would have one Whole Foods. We also used the `distance_col` keyword to cread a distance column.

To confirm that we had obtained distance for every census block, we asserted that the number of rows in the new dataframe were equal to the number of rows in the original census dataframe.

In [ ]:
```
wholefoodscensus = census.sjoin_nearest(
    wholefoods, how="left", distance_col="wf_distance"
)
```

In [ ]:
```
assert census.shape[0] == wholefoodscensus.shape[0]
```

Exercise 5

Drop Non-Urban Census Blocks

Now, the goal was to try and start identifying possible locations. Here, we wanted to drop non-urban census blocks, because Whole Foods is mainly an urban company.

To do this, first we defined a new column representing the area of the census block, using the `area` method. Then we converted this area from meters squared to kilometers (km) squared. Finally, we divided population by the area to get population density, in persons per square km. Then, we filtered out data to included just census blocks with population density greater than or equaly to 100 persons per square km. This removed 2066 census blocks from the data.

In [ ]:
```
wholefoodscensus["area"] = wholefoodscensus.area
wholefoodscensus["area"] = wholefoodscensus["area"] / (1000 ** 2)
wholefoodscensus["pop_dens"] = wholefoodscensus["population"] / wholefoodscensus["area"]
```

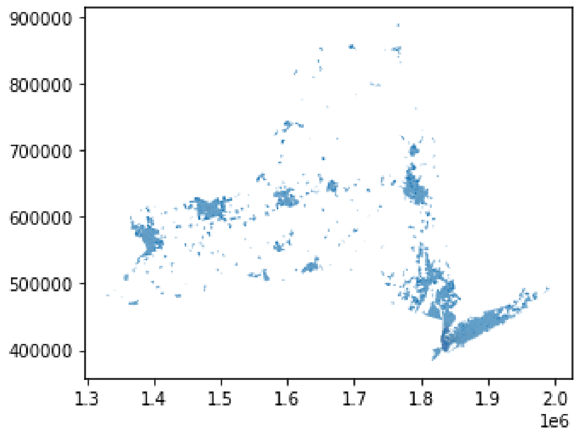
In [ ]:
```
wholefoodscensus = wholefoodscensus[wholefoodscensus["pop_dens"] >= 100]
```

In [ ]:
```
census.shape[0] - wholefoodscensus.shape[0]
```

Out[ ]:
```
2066
```

In [ ]:
```
wholefoodscensus.plot()
```

Out[ ]:
```
<AxesSubplot:>
```



Exercise 6

Remove Close Census Blocks

Next, we wanted to remove census blocks that were already close to a Whole Foods. In this case, we removed census blocks within 8 km (8000 meters) of an existing Whole Foods. This limited our data to 7,111 rows, less than half of the original.

In [ ]:
```
wholefoodscensus = wholefoodscensus[wholefoodscensus["wf_distance"] >= 8000]
```

In [ ]:
```
wholefoodscensus.shape
```

Out[ ]:
```
(7111, 83)
```

In [ ]:
```
census.shape
```

Out[ ]:
```
(15247, 65)
```

Exercise 7

Find Closest Whole-Foods Competitor

Now that we have limited the census blocks to those that are urban and more than 8 km away from an existing Whole Foods, we also wanted to find the closest other supermarket.

To do this, we left joined our existing data with our previously created supermarket GeoDataFrame, using the same strategy as with Whole Foods.

Importantly, this method could potentially create duplicates (i.e. multiple supermarkets with distance 0 from one census block). In order to fix this, we dropped duplicates, checking for unique values on `["STATEFP", "COUNTYFP", "TRACTCE", "BLKGRPCE"]` .

Then, we examined the distribution of the distance of closest supermarket by census block. The spread is fairly large, from 0 km to 57 km, with a median of 1.2 km an average of 2.2 km. A simple histogram plot shows that while the majority of census blocks have a close grocery store, the spread of possible distances is quite wide.

In [ ]:
```
wholefoodscensus = wholefoodscensus.drop("index_right", axis=1)
```

In [ ]:
```
grocerycensus = wholefoodscensus.sjoin_nearest(
    supermarkets, how="left", distance_col="other_distance"
)
```

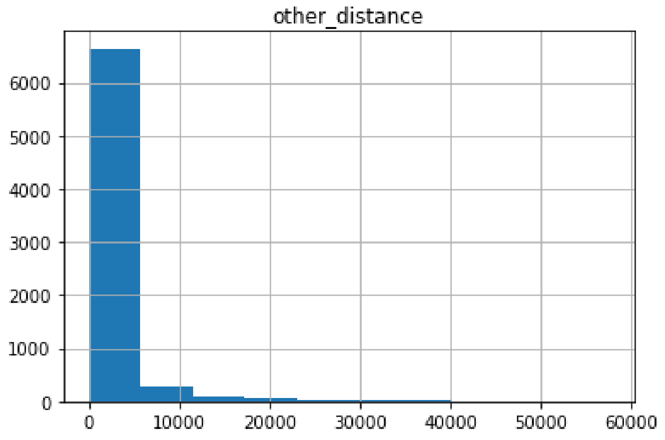
In [ ]:
```
grocerycensus = grocerycensus[
    ~grocerycensus.duplicated(["STATEFP", "COUNTYFP", "TRACTCE", "BLKGRPCE"])
]
```

In [ ]:
```
(grocerycensus["other_distance"] / 1000).describe()
```

Out[ ]:
```
count    7111.000000
mean        2.208458
std         3.768996
min         0.000000
25%         0.480705
50%         1.238037
75%         2.429651
max        57.377311
Name: other_distance, dtype: float64
```

In [ ]:
```
grocerycensus.hist(column="other_distance")
```

Out[ ]:
```
array([[<AxesSubplot:title={'center':'other_distance'}>]], dtype=object)
```



Exercise 8

Identifying Whole Foods Customers

For the point of this exercise, we will make the assumption that Whole Foods customers are mainly wealthy and highly educated. In our dataset, we already had a variable for median income. But, we also wanted to create a variable to construct the share of people of 25 with at least a bachelors degree.

In order to do this, we used the variables beginning with `ALWGE`, which represent the educational attainment for the population 25 years and over, according to the codebook. We used `ALWGE001` as the total, and included 4 educational categories: Bachelor's degree, Master's degree, Professional School degree, and Doctorate degree.

In [ ]:
```
grocerycensus["share_bachelors"] = (
    grocerycensus["ALWGE022"]
    + grocerycensus["ALWGE023"]
    + grocerycensus["ALWGE024"]
    + grocerycensus["ALWGE025"]
) / grocerycensus["ALWGE001"]
```

Exercise 9

Identifying Potential Counties

Now we have 4 variables to select on: share with college degree, median household income, distance to nearest competitor, and distance to nearest Whole Foods.

We already limited to census blocks more than 8 km away from a Whole Foods. Next, we subsetted for census blocks were the share with college degrees is > 50%, median household income is > $90,000 and the nearest competitor is more than 8 km away. After subsetting on these conditions, we had 26 census blocks.

Then, by conducting value counts on the county name, we can see that Westchester and Suffolk county are the best candidates for potential expansion.

In [ ]:
```
potential_candidates = grocerycensus[
    (grocerycensus["share_bachelors"] >= 0.50)
    & (grocerycensus["md_hh_inc"] >= 90000)
    & (grocerycensus["other_distance"] >= 8000)
]
```

In [ ]:
```
potential_candidates.shape
```

Out[ ]:
```
(26, 99)
```

In [ ]:
```
potential_candidates.COUNTY.value_counts()
```

Out[ ]:
```
Westchester County     9
Suffolk County         8
Putnam County          2
Chautauqua County      1
Erie County            1
Niagara County         1
Onondaga County        1
Oswego County          1
St. Lawrence County    1
Ulster County          1
Name: COUNTY, dtype: int64
```

Exercise 10

Changing Approach to Distance

Here, instead of simply finding the nearest grocery store, we want to find the number of grocery stores within 15 km of each census block. To do this, we needed to exapnd each census block polygon within our dataset by 15 km outwards, and then set that column as the GeoDataFrame's official geometry.

For clarity sake, we did all of the merging and subsetting again below.

In [ ]:
```
wholefoodscensus = census.sjoin_nearest(
    wholefoods, how="left", distance_col="wf_distance"
)
wholefoodscensus["area"] = wholefoodscensus.area
wholefoodscensus["area"] = wholefoodscensus["area"] / (1000 ** 2)
wholefoodscensus["pop_dens"] = wholefoodscensus["population"] / wholefoodscensus["area"]
wholefoodscensus = wholefoodscensus[wholefoodscensus["pop_dens"] >= 100]
wholefoodscensus = wholefoodscensus.drop("index_right", axis=1)
grocerycensus = wholefoodscensus.sjoin_nearest(
    supermarkets, how="left", distance_col="other_distance"
)
grocerycensus = grocerycensus[
    ~grocerycensus.duplicated(["STATEFP", "COUNTYFP", "TRACTCE", "BLKGRPCE"])
]
grocerycensus["share_bachelors"] = (
    grocerycensus["ALWGE022"]
    + grocerycensus["ALWGE023"]
    + grocerycensus["ALWGE024"]
    + grocerycensus["ALWGE025"]
) / grocerycensus["ALWGE001"]
```

In [ ]:
```
grocerycensus["buffered"] = grocerycensus.geometry.buffer(15000)
grocerycensus = grocerycensus.set_geometry("buffered")
```

Exercise 11

Merging Buffered Data with Competitors

After obtaining the new buffered data, we merged this again with the supermarket GeoDataFrame to obtain a GeoDataFrame with one row for each block group and grocer within 15 km.

In [ ]:
```
grocerycensus.drop("index_right", axis=1, inplace=True)
new_merge = grocerycensus.sjoin(supermarkets, how="left")
```

In [ ]:
```
new_merge.shape
```

Out[ ]:
```
(1050852, 114)
```

Exercise 12

Count Number of Gocers per Census Block

However, as before, now we have a separate row for each grocer within a census block. To rectify this, we used groupby with the unique identifies in order to count the number of grocers. Then we merged this dataset back into our whole foods dataset to get our final dataset containing the number of grocers (excluding whole foods) within 15 km of each census block.

To ensure this merge was done correctly, we validated that the merge was 1:1, and asserted that there were no rows that were only contained in one of the datasets.

In [ ]:
```
num_grocers = (
    new_merge.groupby(["STATEFP", "COUNTYFP", "TRACTCE", "BLKGRPCE"])[
        "entity_nam_right"
    ]
    .count()
    .reset_index()
    .rename({"entity_nam_right": "num_grocers"}, axis=1)
)
```

In [ ]:
```
final = grocerycensus.merge(
    num_grocers,
    how="outer",
    on=["STATEFP", "COUNTYFP", "TRACTCE", "BLKGRPCE"],
    indicator=True,
    validate="1:1",
)
```

In [ ]:
```
assert final[final["_merge"] != "both"].shape[0] == 0
```

In [ ]:
```
final["num_grocers"].describe()
```

Out[ ]:
```
count    13181.000000
mean        79.724755
std         67.839350
min          1.000000
25%         16.000000
50%         63.000000
75%        146.000000
max        216.000000
Name: num_grocers, dtype: float64
```

Exercise 13

Finding Potential Locations Pt. 2

Finally, with this new merged dataset, we subset based on the following 5 characteristics:

- college degrees is over 50%
- median household income is over $90,000,
- nearest competitor's grocery store is more than 5km away,
- the nearest Whole Foods is over 5km away,
- less than 3 competitors within 15km.

Again, there were 26 census blocks that fit these criteria. However, this time, there were 3 counties with more than one census block: Suffolk County, Saratoga County, and Westchester County.

In [ ]:
```
new_locations = final[
    (final["share_bachelors"] >= 0.50)
    & (final["md_hh_inc"] > 90000)
    & (final["other_distance"] >= 5)
    & (final["wf_distance"] >= 5)
    & (final["num_grocers"] < 3)
]
```

In [ ]:
```
new_locations.shape
```

Out[ ]:
```
(26, 101)
```

In [ ]:
```
new_locations.COUNTY.value_counts()
```

Out[ ]:
```
Suffolk County          7
Saratoga County         6
Westchester County      3
Cattaraugus County      1
Cayuga County           1
Chautauqua County       1
Erie County             1
Madison County          1
Oneida County           1
Orange County           1
Oswego County           1
St. Lawrence County     1
Steuben County          1
Name: COUNTY, dtype: int64
```