Food Waste in NJ

New Jersey's Policy

P.L. 2020. CC. (NJSA 13-99.122)

Generator subjected to the law:

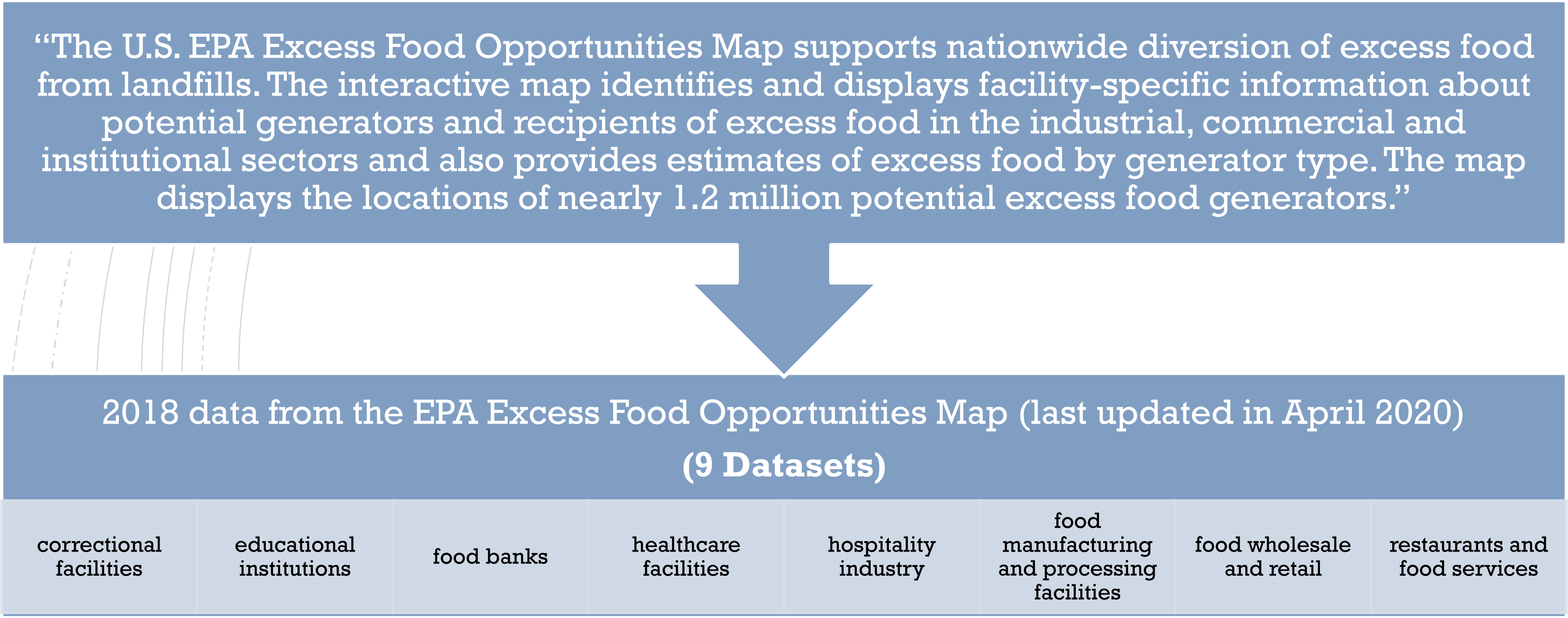
- Produces **52 tons** or more annually
- Within **25 road miles** of recycling facility

Goal:

- **Meet food waste reduction goal by 2030**
- **Free up space in landfills**
- **Repurpose emissions to energy**

Food Waste Recycling & Food Waste-to-Energy Production Law

Data Source: Environmental Protection Agency



"The U.S. EPA Excess Food Opportunities Map supports nationwide diversion of excess food from landfills. The interactive map identifies and displays facility-specific information about potential generators and recipients of excess food in the industrial, commercial and institutional sectors and also provides estimates of excess food by generator type. The map displays the locations of nearly 1.2 million potential excess food generators."

2018 data from the EPA Excess Food Opportunities Map (last updated in April 2020)

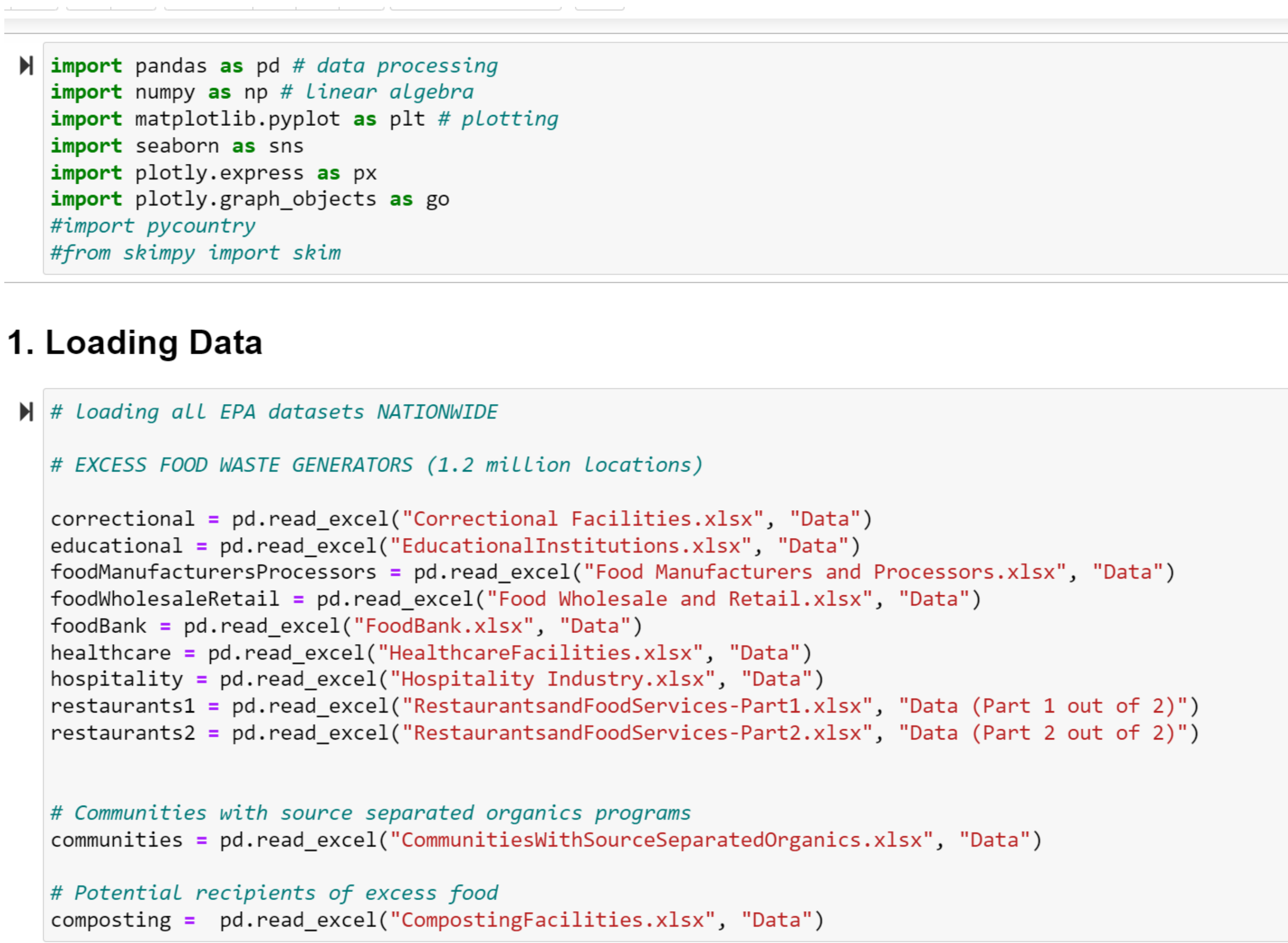
(9 Datasets)

- correctional facilities
- educational institutions
- food banks
- healthcare facilities
- hospitality industry
- food manufacturing and processing facilities
- food wholesale and retail
- restaurants and food services

What is Excess Food?

- "Food—whether processed, semi-processed, or raw—that is intended for human consumption but was removed from the supply chain and is managed in a variety of ways, such as donation to feed people, creation of animal feed, composting, anaerobic digestion, or sending to landfills or combustion facilities" (EPA530-R-20-001, Abstract).

Loading Data

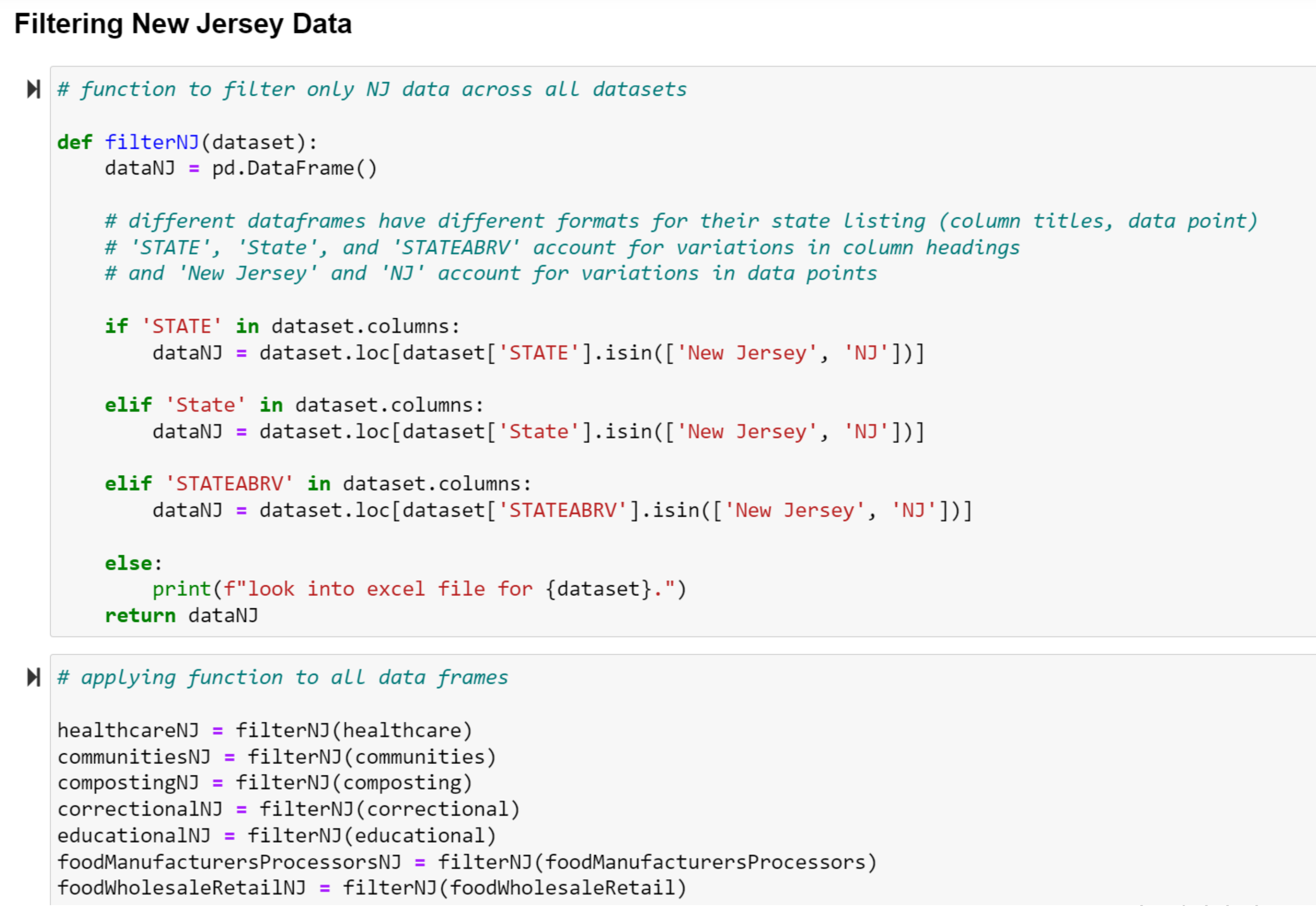


```
import pandas as pd # data processing
import numpy as np # linear algebra
import matplotlib.pyplot as plt # plotting
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
#import pycountry
#from skimpy import skim
```

1. Loading Data

```
# loading all EPA datasets NATIONWIDE

# EXCESS FOOD WASTE GENERATORS (1.2 million locations)

correctional = pd.read_excel("Correctional Facilities.xlsx", "Data")
educational = pd.read_excel("EducationalInstitutions.xlsx", "Data")
foodManufacturersProcessors = pd.read_excel("Food Manufacturers and Processors.xlsx", "Data")
foodWholesaleRetail = pd.read_excel("Food Wholesale and Retail.xlsx", "Data")
foodBank = pd.read_excel("FoodBank.xlsx", "Data")
healthcare = pd.read_excel("HealthcareFacilities.xlsx", "Data")
hospitality = pd.read_excel("Hospitality Industry.xlsx", "Data")
restaurants1 = pd.read_excel("RestaurantsandFoodServices-Part1.xlsx", "Data (Part 1 out of 2)")
restaurants2 = pd.read_excel("RestaurantsandFoodServices-Part2.xlsx", "Data (Part 2 out of 2)")


# Communities with source separated organics programs
communities = pd.read_excel("CommunitiesWithSourceSeparatedOrganics.xlsx", "Data")

# Potential recipients of excess food
composting =  pd.read_excel("CompostingFacilities.xlsx", "Data")
```

Data Cleaning

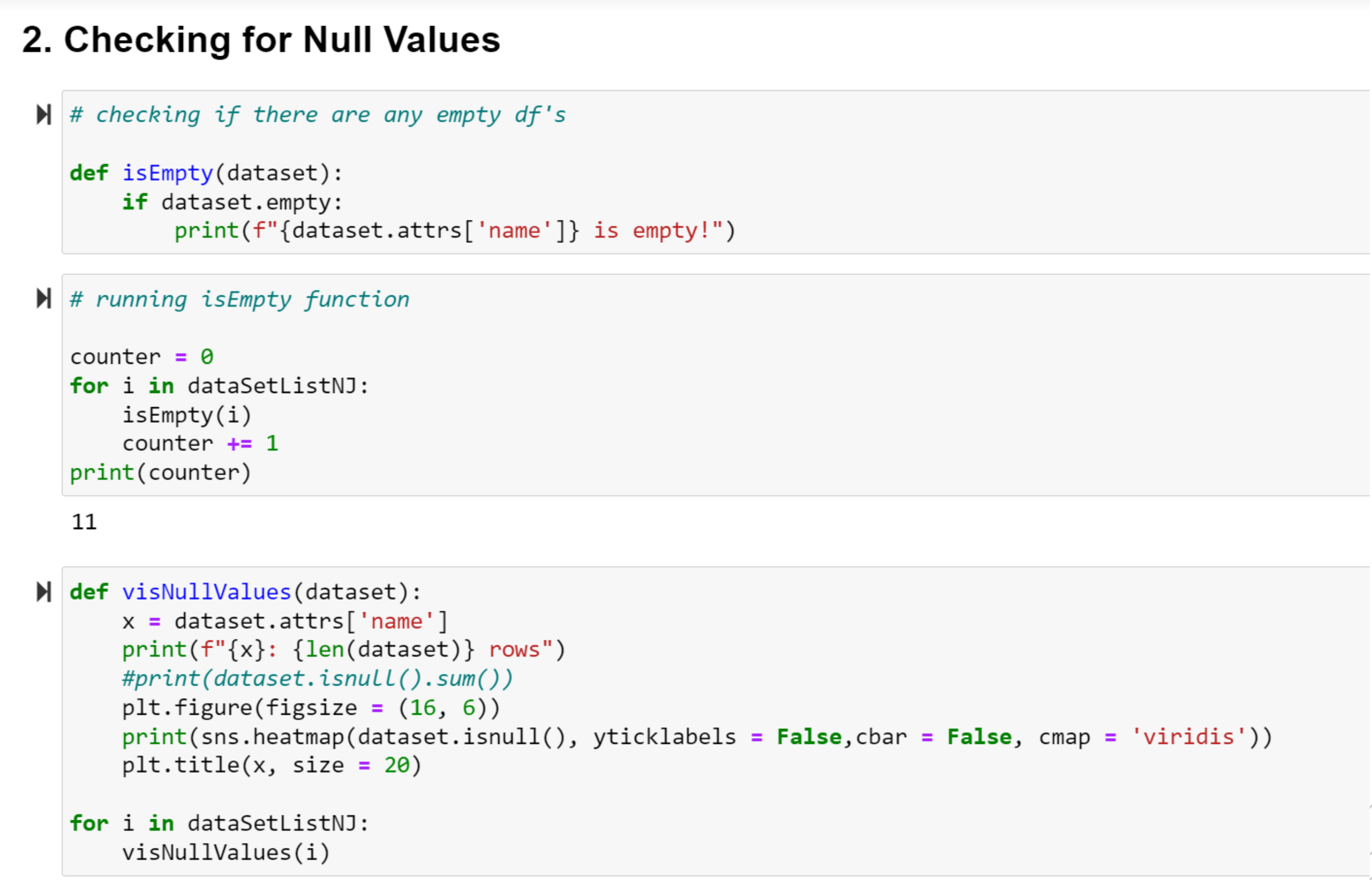


Filtering New Jersey Data

```
# function to filter only NJ data across all datasets

def filterNJ(dataset):
    dataNJ = pd.DataFrame()

    # different dataframes have different formats for their state listing (column titles, data point)
    # 'STATE', 'State', and 'STATEABRV' account for variations in column headings
    # and 'New Jersey' and 'NJ' account for variations in data points

    if 'STATE' in dataset.columns:
        dataNJ = dataset.loc[dataset['STATE'].isin(['New Jersey', 'NJ'])]

    elif 'State' in dataset.columns:
        dataNJ = dataset.loc[dataset['State'].isin(['New Jersey', 'NJ'])]

    elif 'STATEABRV' in dataset.columns:
        dataNJ = dataset.loc[dataset['STATEABRV'].isin(['New Jersey', 'NJ'])]

    else:
        print(f"look into excel file for {dataset}.")
    return dataNJ
```

```
# applying function to all data frames

healthcareNJ = filterNJ(healthcare)
communitiesNJ = filterNJ(communities)
compostingNJ = filterNJ(composting)
correctionalNJ = filterNJ(correctional)
educationalNJ = filterNJ(educational)
foodManufacturersProcessorsNJ = filterNJ(foodManufacturersProcessors)
foodWholesaleRetailNJ = filterNJ(foodWholesaleRetail)
```

Data Cleaning



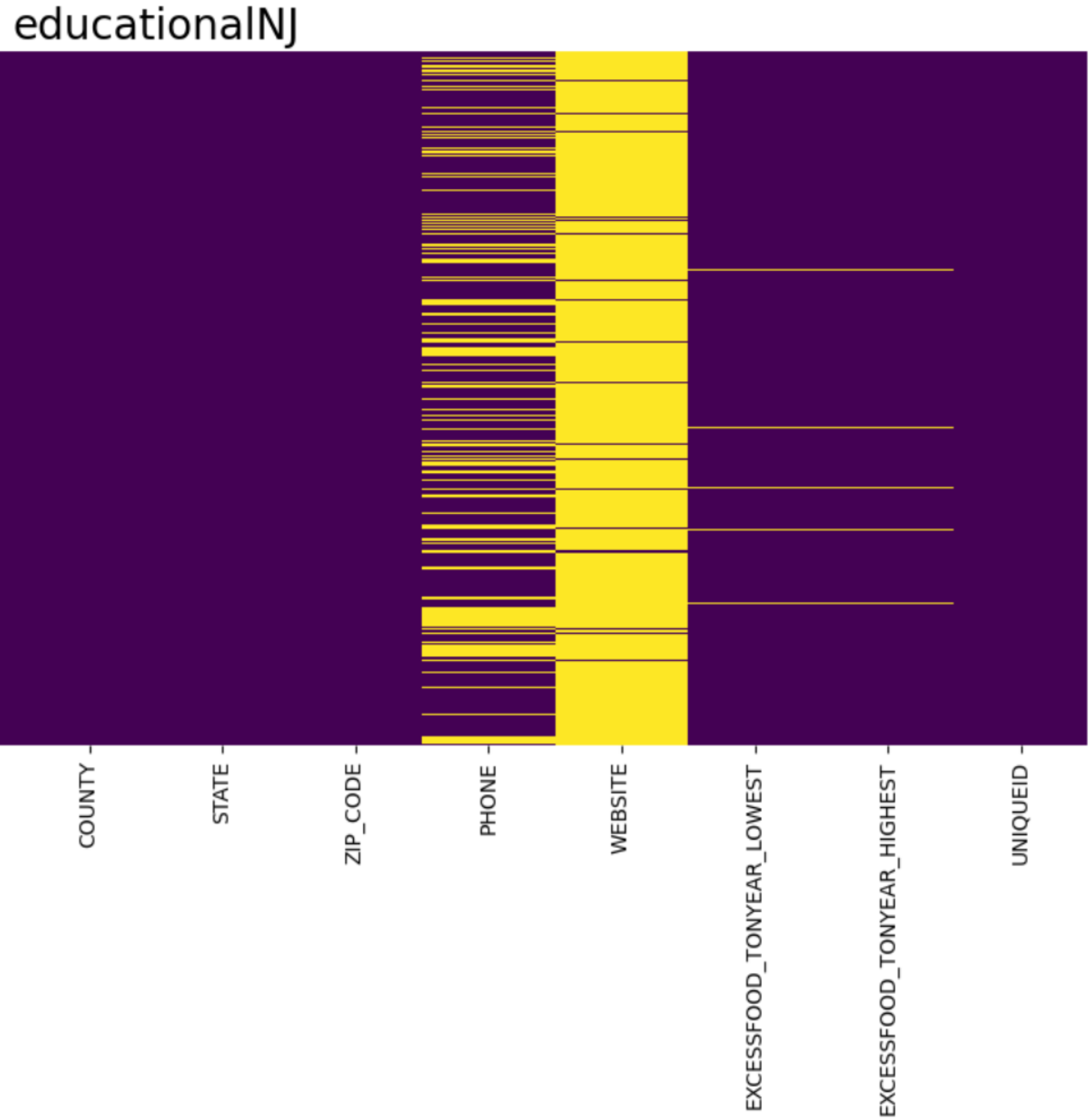
2. Checking for Null Values

```
# checking if there are any empty df's

def isEmpty(dataset):
    if dataset.empty:
        print(f"{dataset.attrs['name']} is empty!")
```

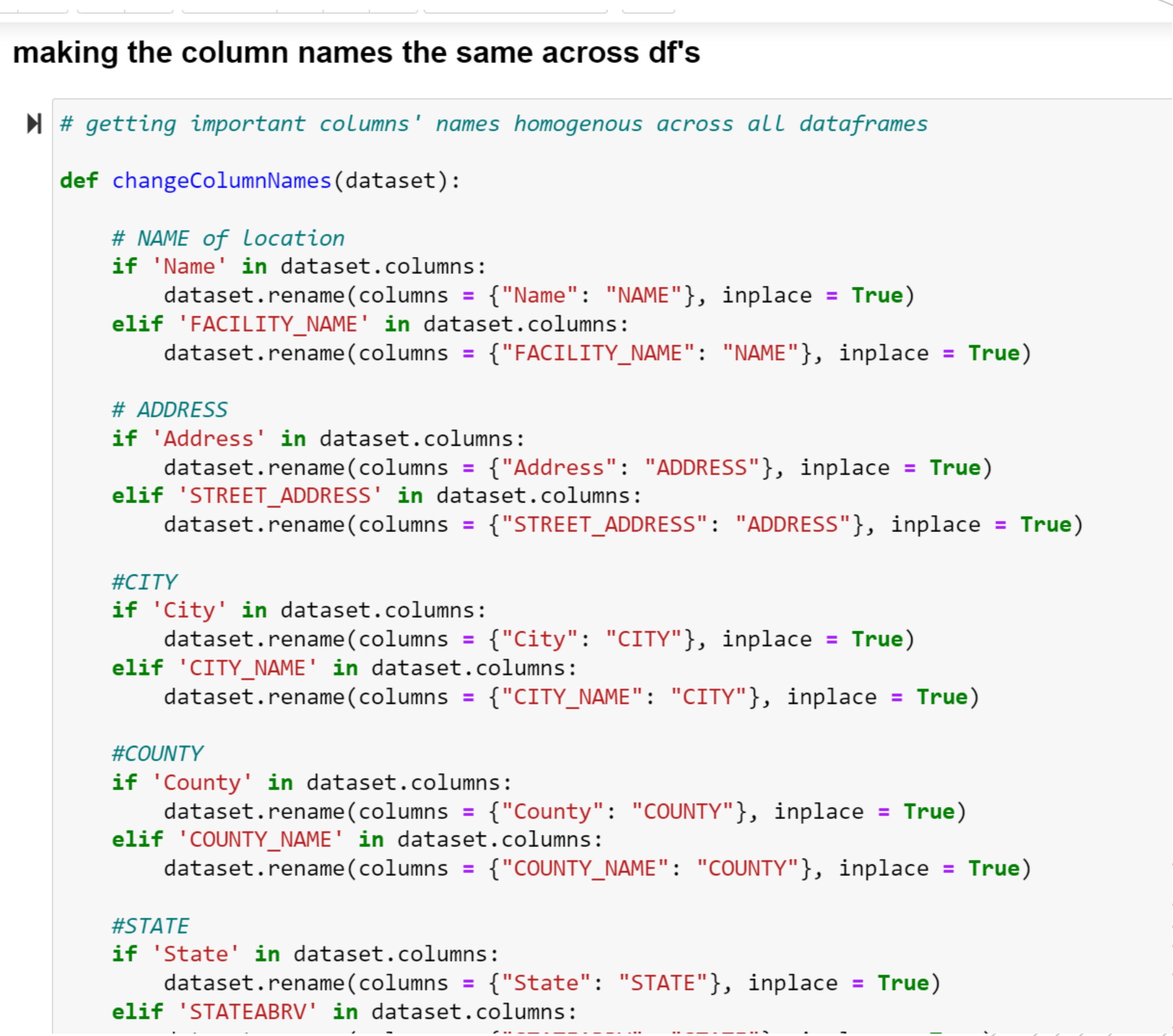
```
# running isEmpty function

counter = 0
for i in dataSetListNJ:
    isEmpty(i)
    counter += 1
print(counter)
```

```
11
```

```
def visNullValues(dataset):
    x = dataset.attrs['name']
    print(f"{x}: {len(dataset)} rows")
    #print(dataset.isnull().sum())
    plt.figure(figsize = (16, 6))
    print(sns.heatmap(dataset.isnull(), yticklabels = False,cbar = False, cmap = 'viridis'))
    plt.title(x, size = 20)

for i in dataSetListNJ:
    visNullValues(i)
```

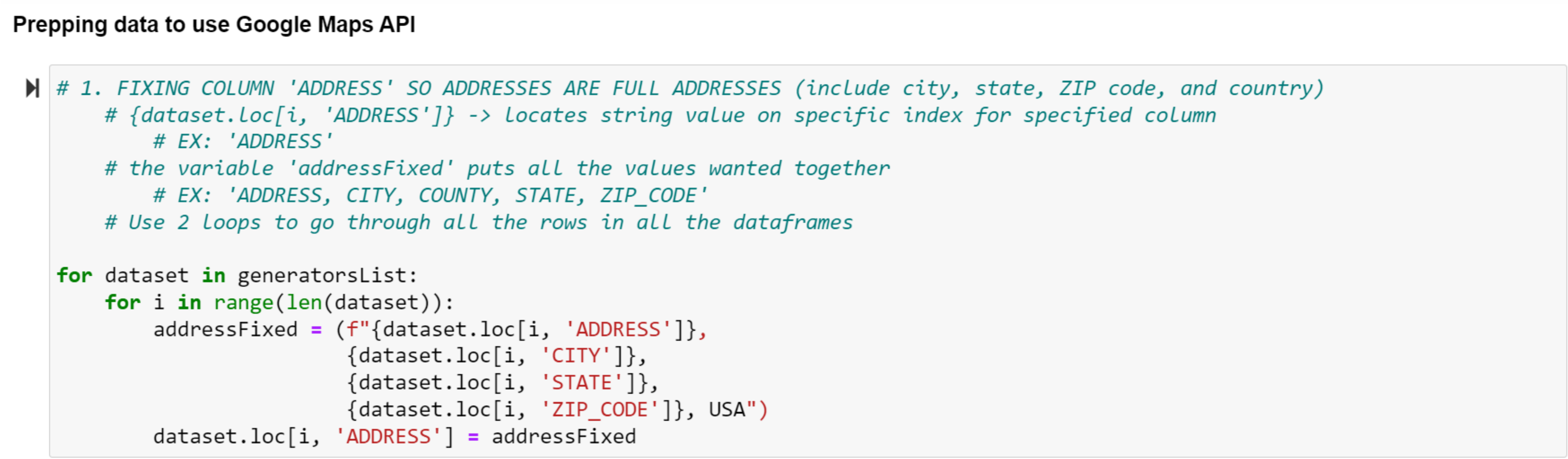
educationalNJ
XCoord
YCoord
NAME
SCHOOL_TYPE
ADDRESS
CITY
COUNTY
STATE
ZIP_CODE
PHONE
WEBSITE
EXCESSFOOD_TONYEAR_LOWEST
EXCESSFOOD_TONYEAR_HIGHEST
UNIQUEID

Data Cleaning



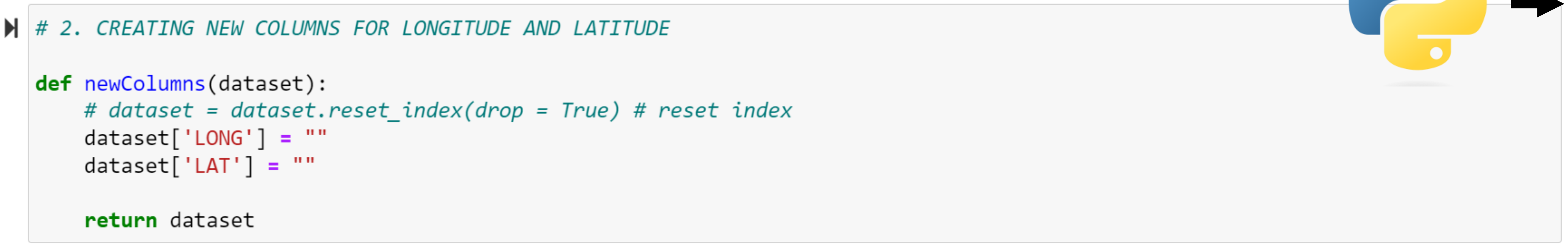
making the column names the same across df's

```
# getting important columns' names homogenous across all dataframes

def changeColumnNames(dataset):

    # NAME of Location
    if 'Name' in dataset.columns:
        dataset.rename(columns = {"Name": "NAME"}, inplace = True)
    elif 'FACILITY_NAME' in dataset.columns:
        dataset.rename(columns = {"FACILITY_NAME": "NAME"}, inplace = True)

    # ADDRESS
    if 'Address' in dataset.columns:
        dataset.rename(columns = {"Address": "ADDRESS"}, inplace = True)
    elif 'STREET_ADDRESS' in dataset.columns:
        dataset.rename(columns = {"STREET_ADDRESS": "ADDRESS"}, inplace = True)

    #CITY
    if 'City' in dataset.columns:
        dataset.rename(columns = {"City": "CITY"}, inplace = True)
    elif 'CITY_NAME' in dataset.columns:
        dataset.rename(columns = {"CITY_NAME": "CITY"}, inplace = True)

    #COUNTY
    if 'County' in dataset.columns:
        dataset.rename(columns = {"County": "COUNTY"}, inplace = True)
    elif 'COUNTY_NAME' in dataset.columns:
        dataset.rename(columns = {"COUNTY_NAME": "COUNTY"}, inplace = True)

    #STATE
    if 'State' in dataset.columns:
        dataset.rename(columns = {"State": "STATE"}, inplace = True)
    elif 'STATEABRV' in dataset.columns:
```

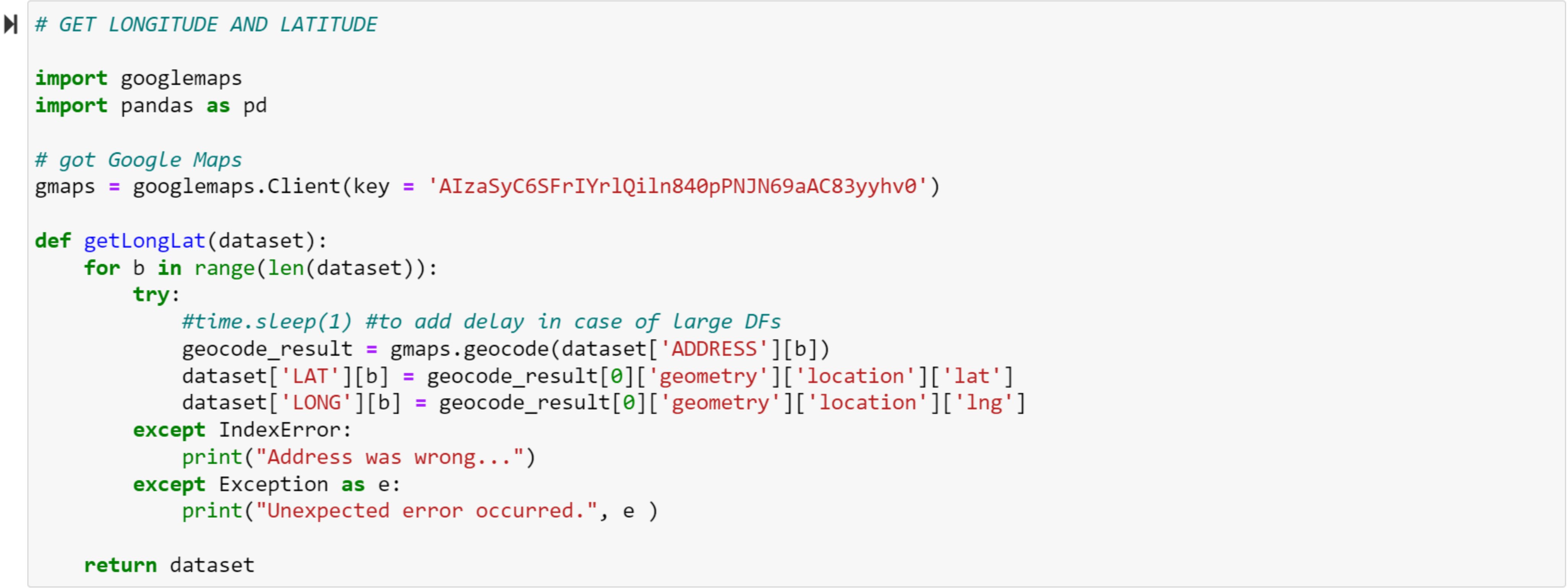


Prepping data to use Google Maps API

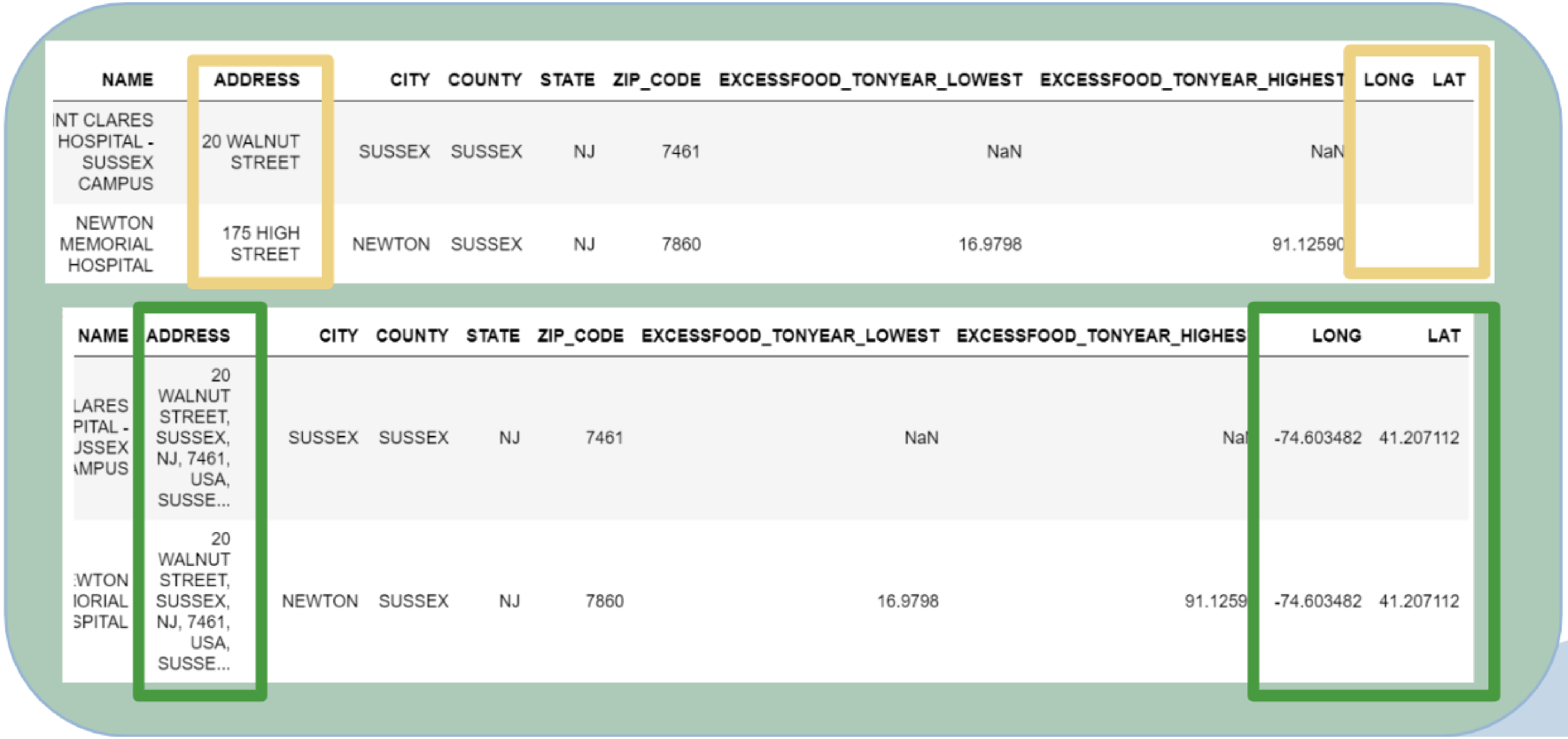
```python
# 1. FIXING COLUMN 'ADDRESS' SO ADDRESSES ARE FULL ADDRESSES (include city, state, ZIP code, and country)
    # {dataset.loc[i, 'ADDRESS']} -> locates string value on specific index for specified column
        # EX: 'ADDRESS'
    # the variable 'addressFixed' puts all the values wanted together
        # EX: 'ADDRESS, CITY, COUNTY, STATE, ZIP_CODE'
    # Use 2 loops to go through all the rows in all the dataframes

for dataset in generatorsList:
    for i in range(len(dataset)):
        addressFixed = (f"{dataset.loc[i, 'ADDRESS']},
                        {dataset.loc[i, 'CITY']},
                        {dataset.loc[i, 'STATE']},
                        {dataset.loc[i, 'ZIP_CODE']}, USA")
        dataset.loc[i, 'ADDRESS'] = addressFixed
```



```python
# 2. CREATING NEW COLUMNS FOR LONGITUDE AND LATITUDE

def newColumns(dataset):
    # dataset = dataset.reset_index(drop = True) # reset index
    dataset['LONG'] = ""
    dataset['LAT'] = ""

    return dataset
```

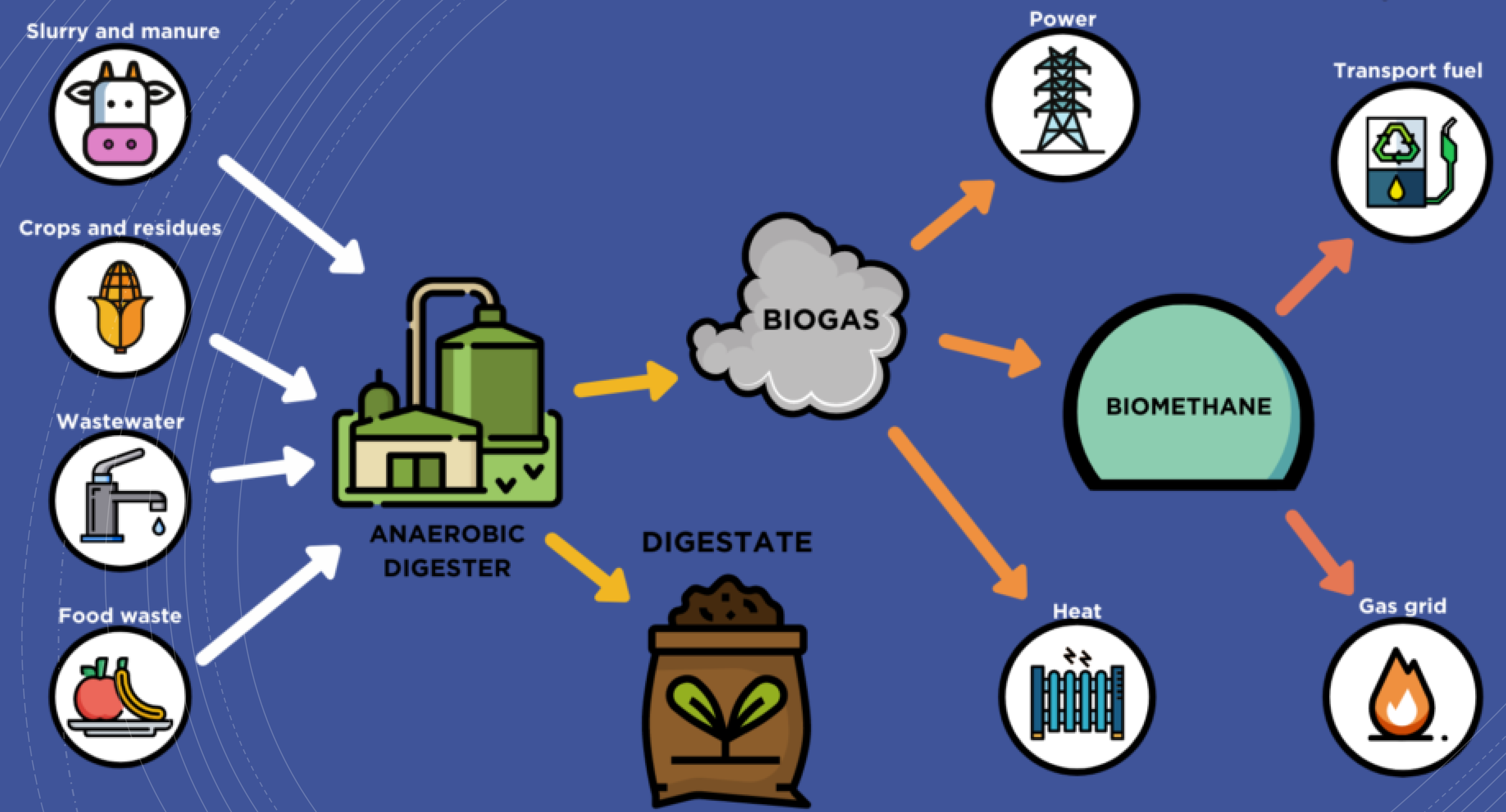


```python
# GET LONGITUDE AND LATITUDE

import googlemaps
import pandas as pd

# got Google Maps
gmaps = googlemaps.Client(key = 'AIzaSyC6SFrIYrlQiln840pPNJN69aAC83yyhv0')

def getLongLat(dataset):
    for b in range(len(dataset)):
        try:
            #time.sleep(1) #to add delay in case of large DFs
            geocode_result = gmaps.geocode(dataset['ADDRESS'][b])
            dataset['LAT'][b] = geocode_result[0]['geometry']['location']['lat']
            dataset['LONG'][b] = geocode_result[0]['geometry']['location']['lng']
        except IndexError:
            print("Address was wrong...")
        except Exception as e:
            print("Unexpected error occurred.", e )

    return dataset
```

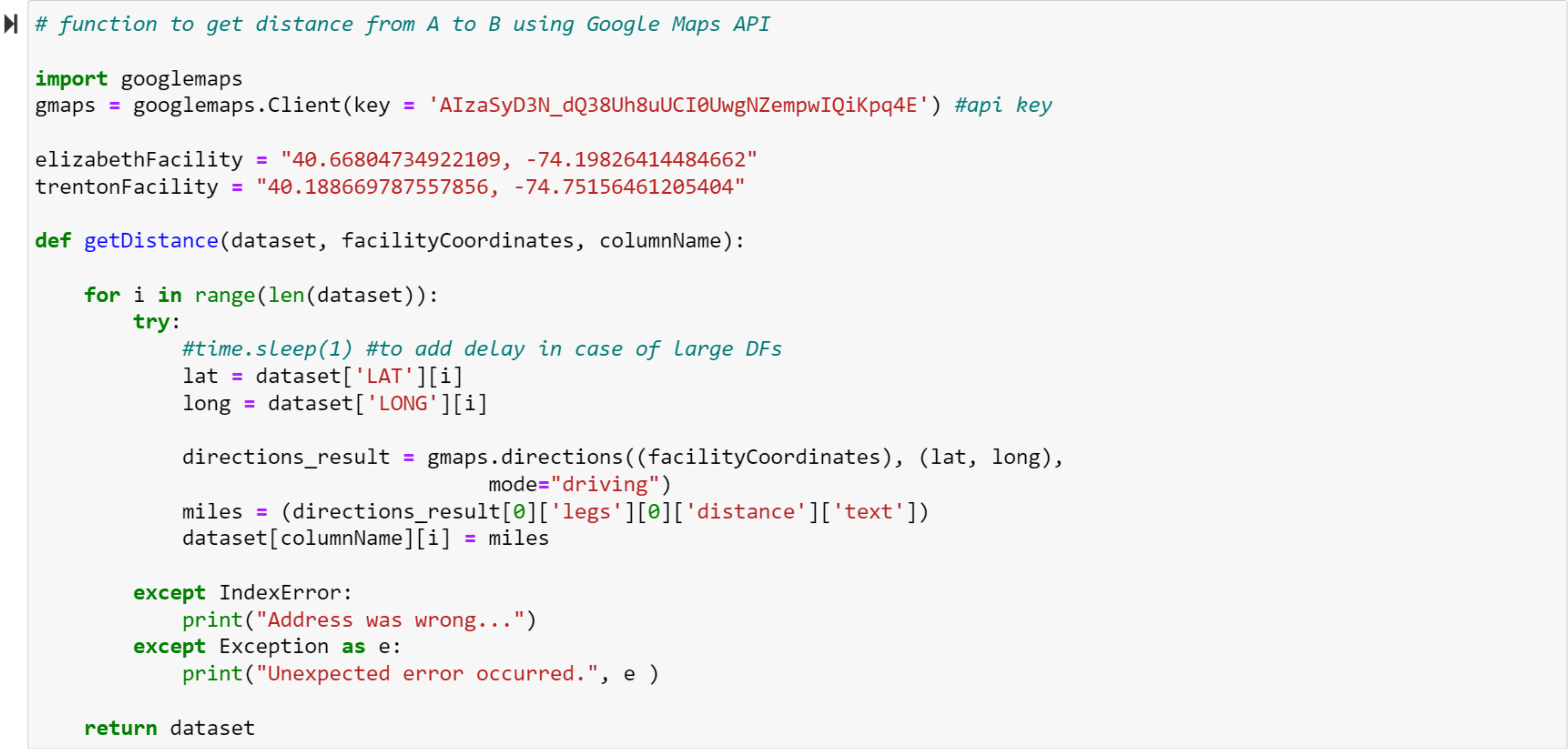


NAME	ADDRESS	CITY	COUNTY	STATE	ZIP_CODE	EXCESSFOOD_TONYEAR_LOWEST	EXCESSFOOD_TONYEAR_HIGHEST	LONG	LAT
INT CLARES HOSPITAL - SUSSEX CAMPUS	20 WALNUT STREET	SUSSEX	SUSSEX	NJ	7461	NaN	NaN		
NEWTON MEMORIAL HOSPITAL	175 HIGH STREET	NEWTON	SUSSEX	NJ	7860	16.9798	91.12590		

NAME	ADDRESS	CITY	COUNTY	STATE	ZIP_CODE	EXCESSFOOD_TONYEAR_LOWEST	EXCESSFOOD_TONYEAR_HIGHES	LONG	LAT
LARES PITAL - JSSEX MPUS	20 WALNUT STREET, SUSSEX, NJ, 7461, USA, SUSSE...	SUSSEX	SUSSEX	NJ	7461	NaN	Na	-74.603482	41.207112
WTON IORIAL SPITAL	20 WALNUT STREET, SUSSEX, NJ, 7461, USA, SUSSE...	NEWTON	SUSSEX	NJ	7860	16.9798	91.1259	-74.603482	41.207112

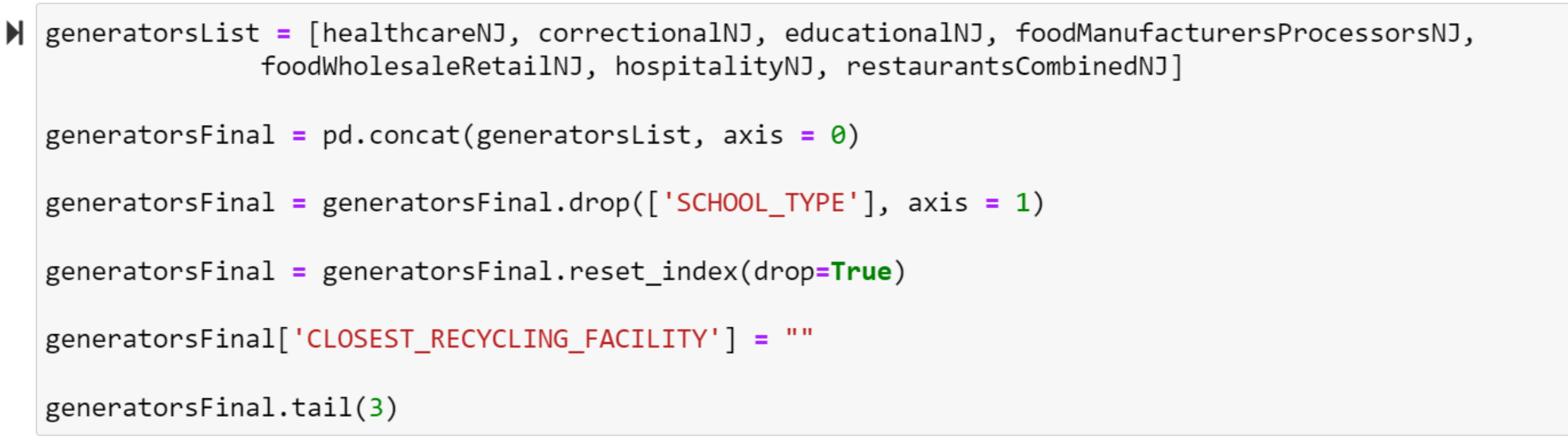

Slurry and manure
Crops and residues
Wastewater
Food waste
ANAEROBIC DIGESTER
BIOGAS
DIGESTATE
Power
Transport fuel
BIOMETHANE
Heat
Gas grid

2. GETTING DISTANCES FROM FOOD RECYCLING FACILITIES

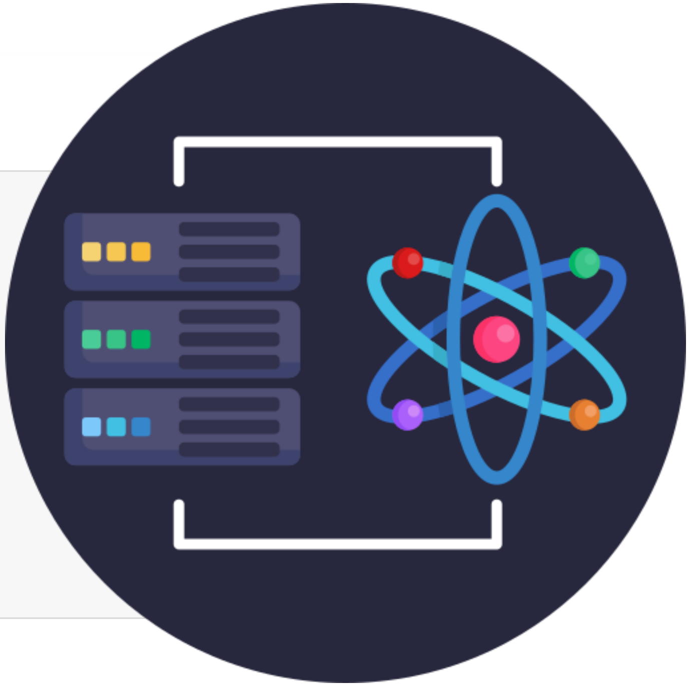


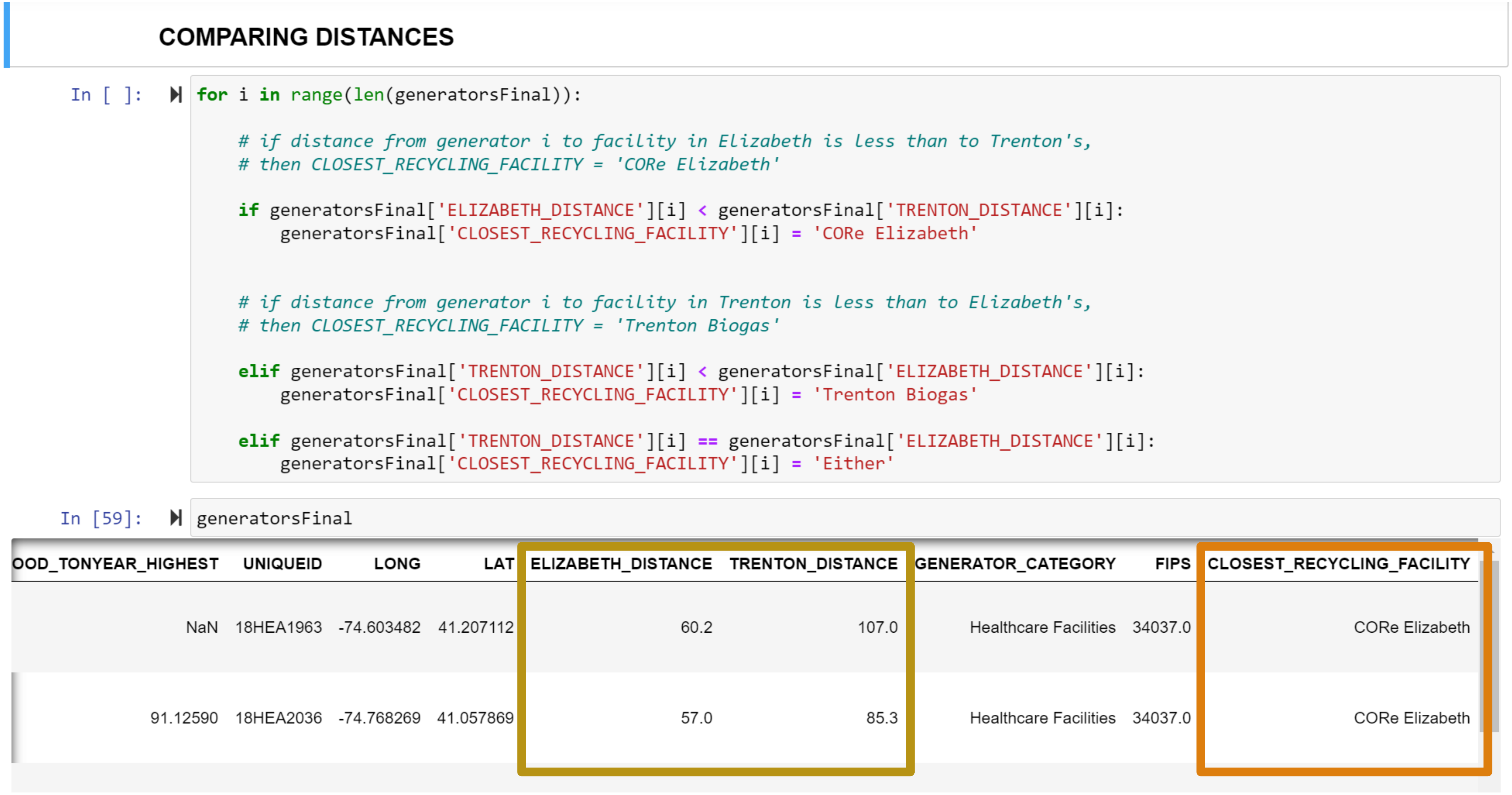
```python
# function to get distance from A to B using Google Maps API

import googlemaps
gmaps = googlemaps.Client(key = 'AIzaSyD3N_dQ38Uh8uUCI0UwgNZempwIQiKpq4E') #api key

elizabethFacility = "40.66804734922109, -74.19826414484662"
trentonFacility = "40.188669787557856, -74.75156461205404"

def getDistance(dataset, facilityCoordinates, columnName):

    for i in range(len(dataset)):
        try:
            #time.sleep(1) #to add delay in case of large DFs
            lat = dataset['LAT'][i]
            long = dataset['LONG'][i]

            directions_result = gmaps.directions((facilityCoordinates), (lat, long),
                                                 mode="driving")
            miles = (directions_result[0]['legs'][0]['distance']['text'])
            dataset[columnName][i] = miles

        except IndexError:
            print("Address was wrong...")
        except Exception as e:
            print("Unexpected error occurred.", e )

    return dataset
```

3. COMBINING ALL SETS INTO FINAL SET



```
generatorsList = [healthcareNJ, correctionalNJ, educationalNJ, foodManufacturersProcessorsNJ,
              foodWholesaleRetailNJ, hospitalityNJ, restaurantsCombinedNJ]

generatorsFinal = pd.concat(generatorsList, axis = 0)

generatorsFinal = generatorsFinal.drop(['SCHOOL_TYPE'], axis = 1)

generatorsFinal = generatorsFinal.reset_index(drop=True)

generatorsFinal['CLOSEST_RECYCLING_FACILITY'] = ""

generatorsFinal.tail(3)
```





COMPARING DISTANCES

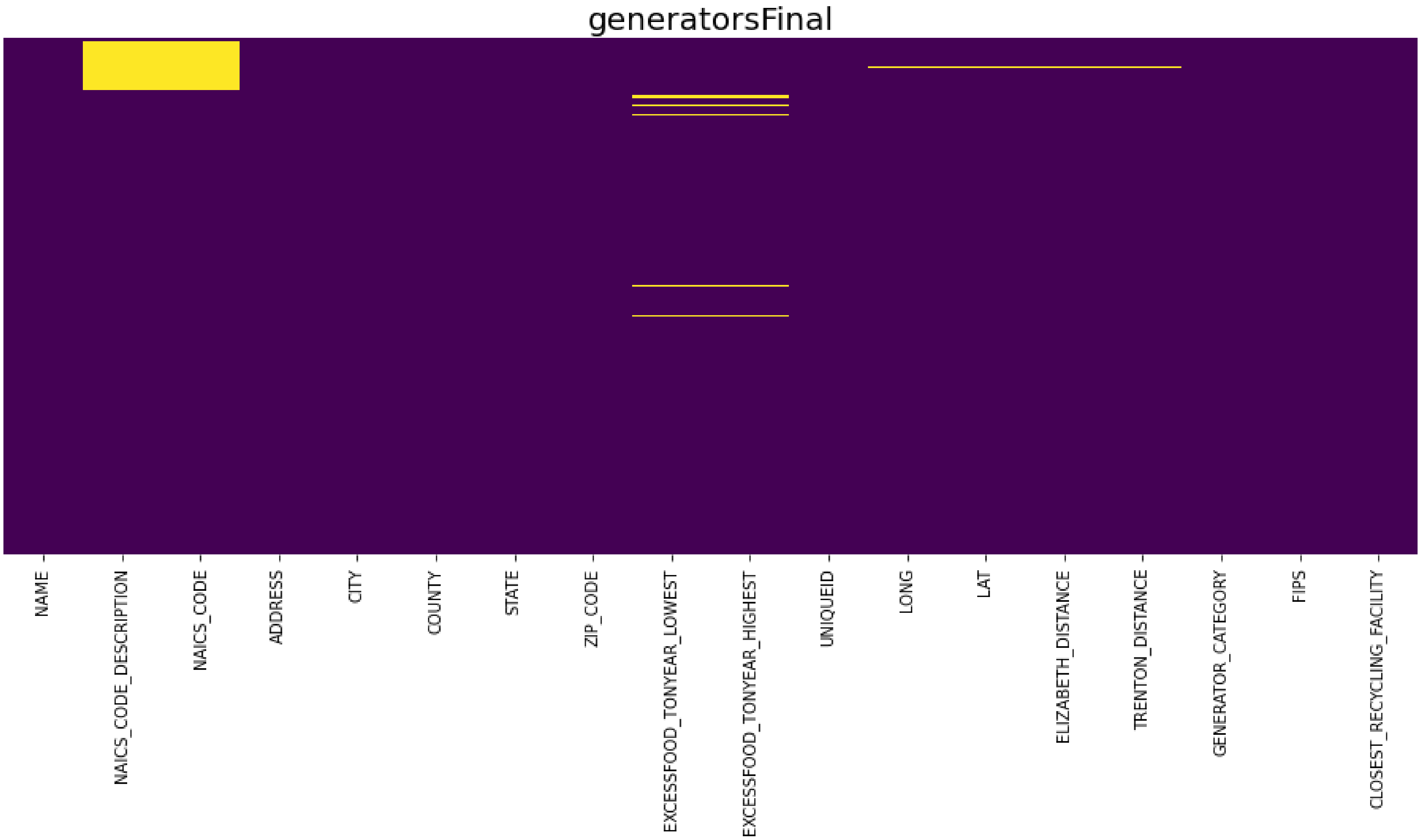
In [ ]:

```
for i in range(len(generatorsFinal)):

    # if distance from generator i to facility in Elizabeth is less than to Trenton's,
    # then CLOSEST_RECYCLING_FACILITY = 'CORe Elizabeth'

    if generatorsFinal['ELIZABETH_DISTANCE'][i] < generatorsFinal['TRENTON_DISTANCE'][i]:
        generatorsFinal['CLOSEST_RECYCLING_FACILITY'][i] = 'CORe Elizabeth'


    # if distance from generator i to facility in Trenton is less than to Elizabeth's,
    # then CLOSEST_RECYCLING_FACILITY = 'Trenton Biogas'

    elif generatorsFinal['TRENTON_DISTANCE'][i] < generatorsFinal['ELIZABETH_DISTANCE'][i]:
        generatorsFinal['CLOSEST_RECYCLING_FACILITY'][i] = 'Trenton Biogas'

    elif generatorsFinal['TRENTON_DISTANCE'][i] == generatorsFinal['ELIZABETH_DISTANCE'][i]:
        generatorsFinal['CLOSEST_RECYCLING_FACILITY'][i] = 'Either'
```

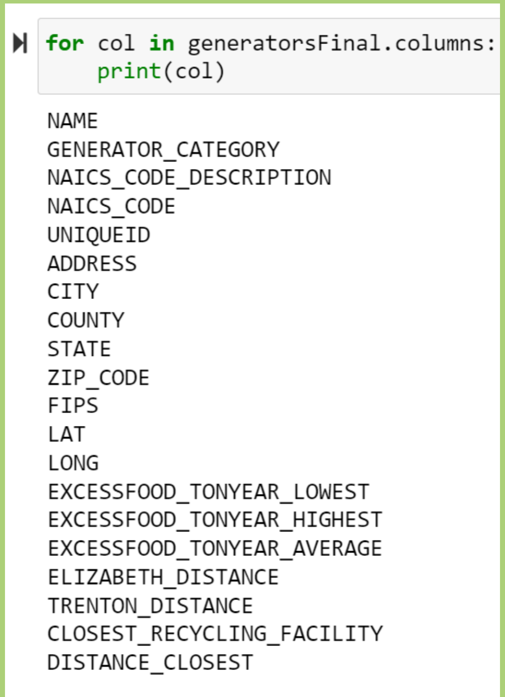
In [59]:

```
generatorsFinal
```

OOD_TONYEAR_HIGHEST	UNIQUEID	LONG	LAT	ELIZABETH_DISTANCE	TRENTON_DISTANCE	GENERATOR_CATEGORY	FIPS	CLOSEST_RECYCLING_FACILITY
NaN	18HEA1963	-74.603482	41.207112	60.2	107.0	Healthcare Facilities	34037.0	CORe Elizabeth
91.12590	18HEA2036	-74.768269	41.057869	57.0	85.3	Healthcare Facilities	34037.0	CORe Elizabeth

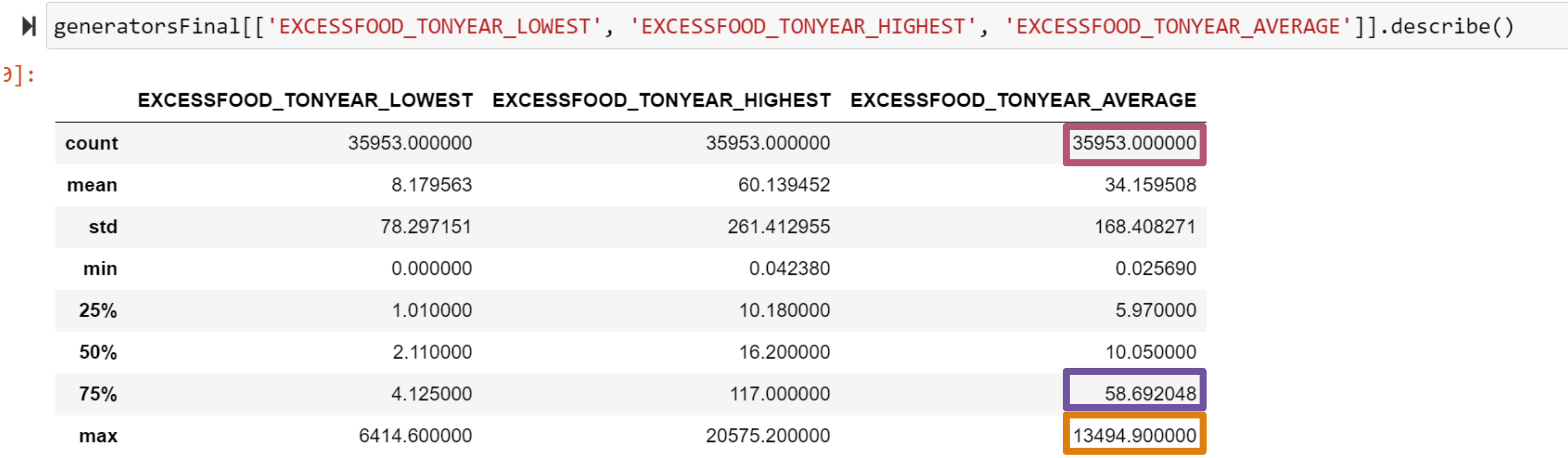

generatorsFinal
NAME
NAICS_CODE_DESCRIPTION
NAICS_CODE
ADDRESS
CITY
COUNTY
STATE
ZIP_CODE
EXCESSFOOD_TONYEAR_LOWEST
EXCESSFOOD_TONYEAR_HIGHEST
UNIQUEID
LONG
LAT
ELIZABETH_DISTANCE
TRENTON_DISTANCE
GENERATOR_CATEGORY
FIPS
CLOSEST_RECYCLING_FACILITY

Data Stats



```
for col in generatorsFinal.columns:
    print(col)
```

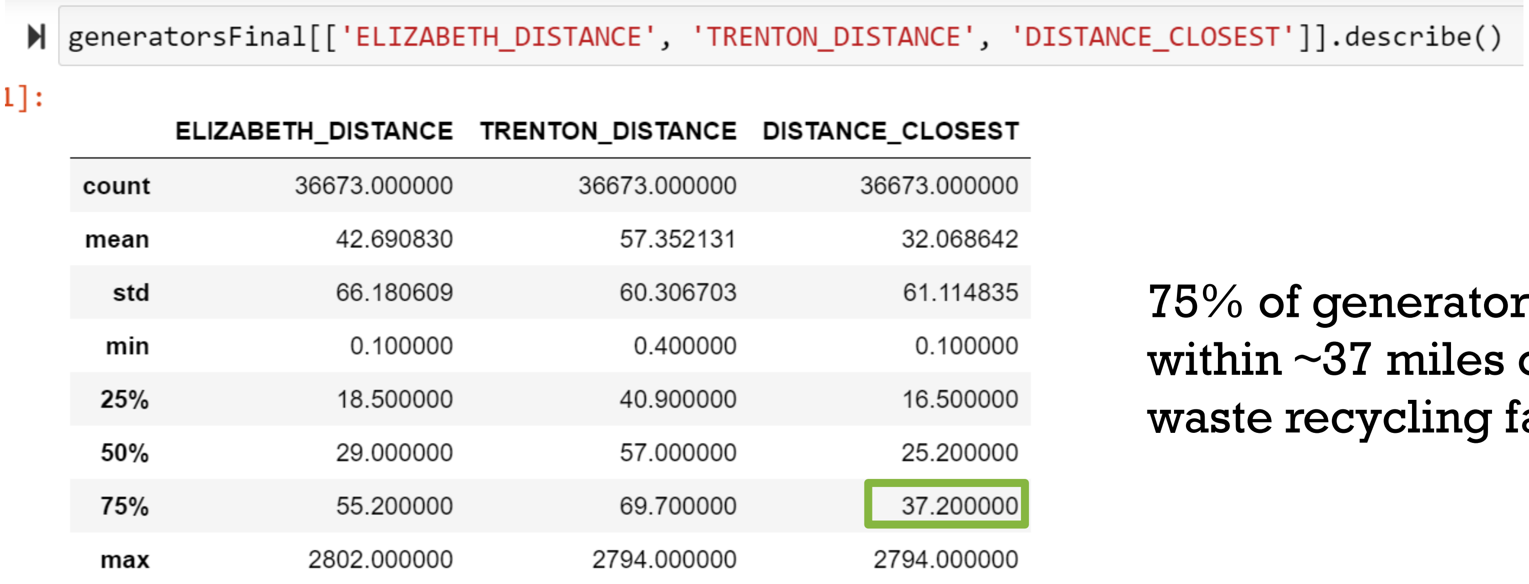
```
NAME
GENERATOR_CATEGORY
NAICS_CODE_DESCRIPTION
NAICS_CODE
UNIQUEID
ADDRESS
CITY
COUNTY
STATE
ZIP_CODE
FIPS
LAT
LONG
EXCESSFOOD_TONYEAR_LOWEST
EXCESSFOOD_TONYEAR_HIGHEST
EXCESSFOOD_TONYEAR_AVERAGE
ELIZABETH_DISTANCE
TRENTON_DISTANCE
CLOSEST_RECYCLING_FACILITY
DISTANCE_CLOSEST
```



```
generatorsFinal[['EXCESSFOOD_TONYEAR_LOWEST', 'EXCESSFOOD_TONYEAR_HIGHEST', 'EXCESSFOOD_TONYEAR_AVERAGE']].describe()
```

]:

	EXCESSFOOD_TONYEAR_LOWEST	EXCESSFOOD_TONYEAR_HIGHEST	EXCESSFOOD_TONYEAR_AVERAGE
count	35953.000000	35953.000000	35953.000000
mean	8.179563	60.139452	34.159508
std	78.297151	261.412955	168.408271
min	0.000000	0.042380	0.025690
25%	1.010000	10.180000	5.970000
50%	2.110000	16.200000	10.050000
75%	4.125000	117.000000	58.692048
max	6414.600000	20575.200000	13494.900000

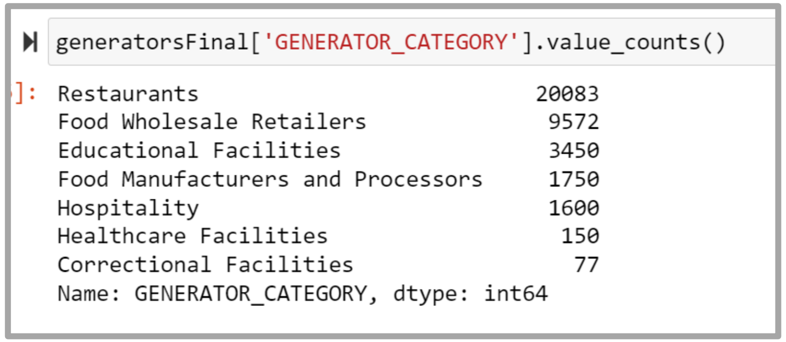


```
generatorsFinal[['ELIZABETH_DISTANCE', 'TRENTON_DISTANCE', 'DISTANCE_CLOSEST']].describe()
```

1]:

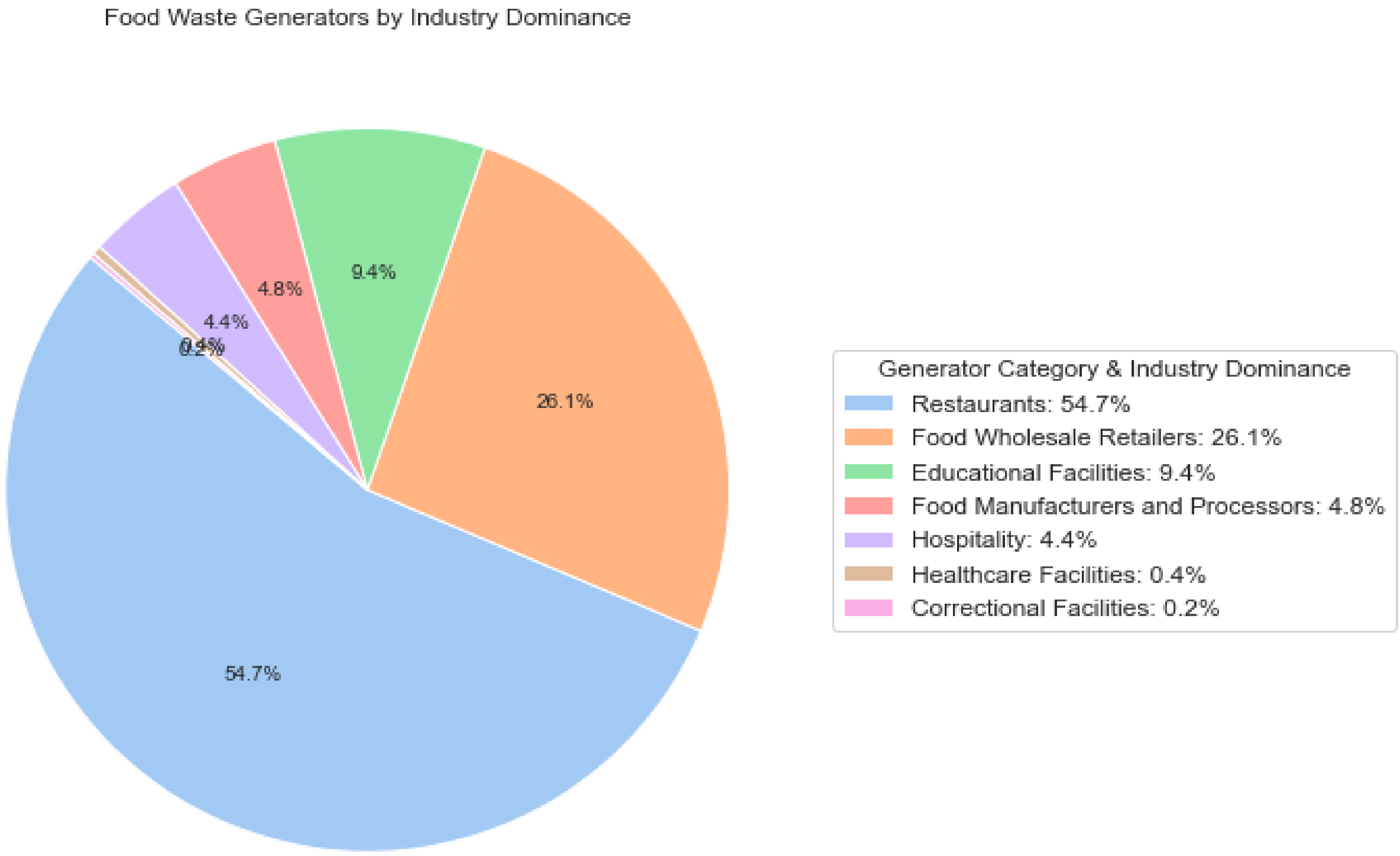
	ELIZABETH_DISTANCE	TRENTON_DISTANCE	DISTANCE_CLOSEST
count	36673.000000	36673.000000	36673.000000
mean	42.690830	57.352131	32.068642
std	66.180609	60.306703	61.114835
min	0.100000	0.400000	0.100000
25%	18.500000	40.900000	16.500000
50%	29.000000	57.000000	25.200000
75%	55.200000	69.700000	37.200000
max	2802.000000	2794.000000	2794.000000

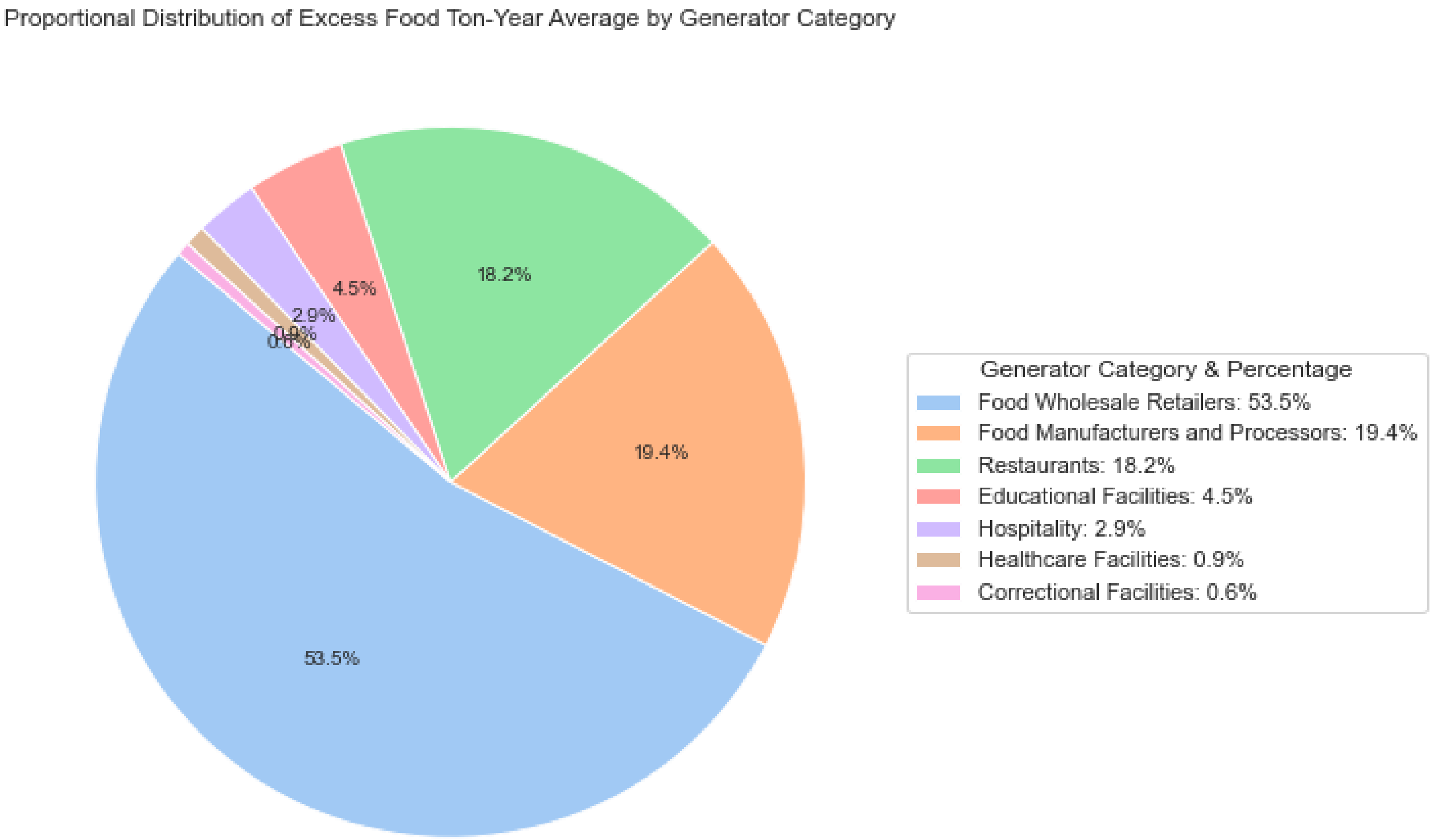
75% of generators are within ~37 miles of a food waste recycling facility



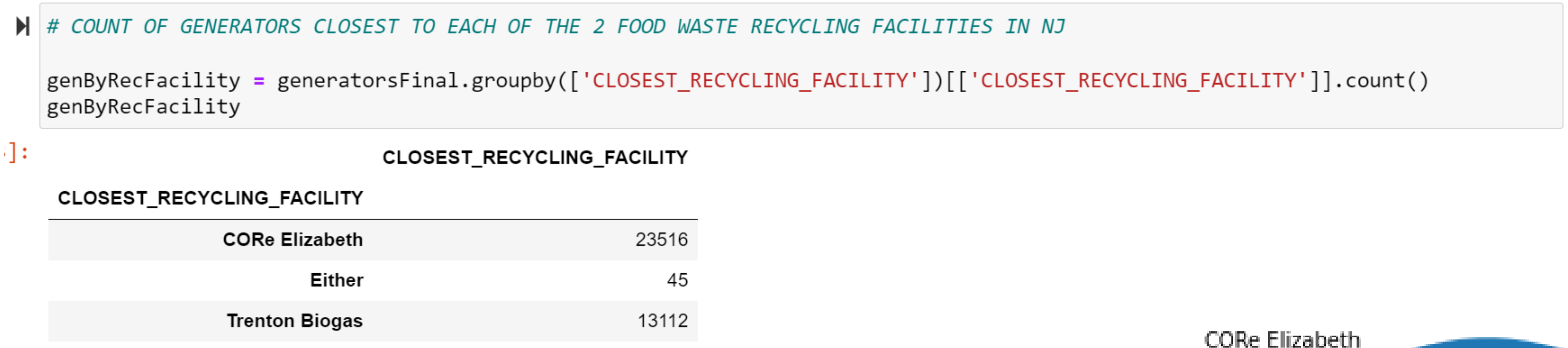
```
generatorsFinal['GENERATOR_CATEGORY'].value_counts()
```

```
]:  Restaurants                        20083
    Food Wholesale Retailers            9572
    Educational Facilities              3450
    Food Manufacturers and Processors   1750
    Hospitality                         1600
    Healthcare Facilities                150
    Correctional Facilities               77
    Name: GENERATOR_CATEGORY, dtype: int64
```



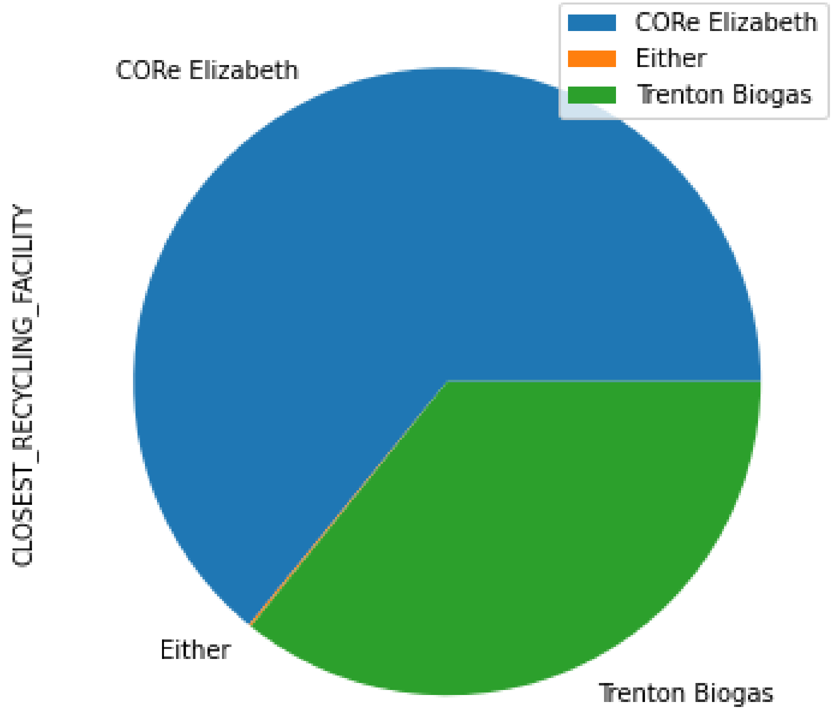


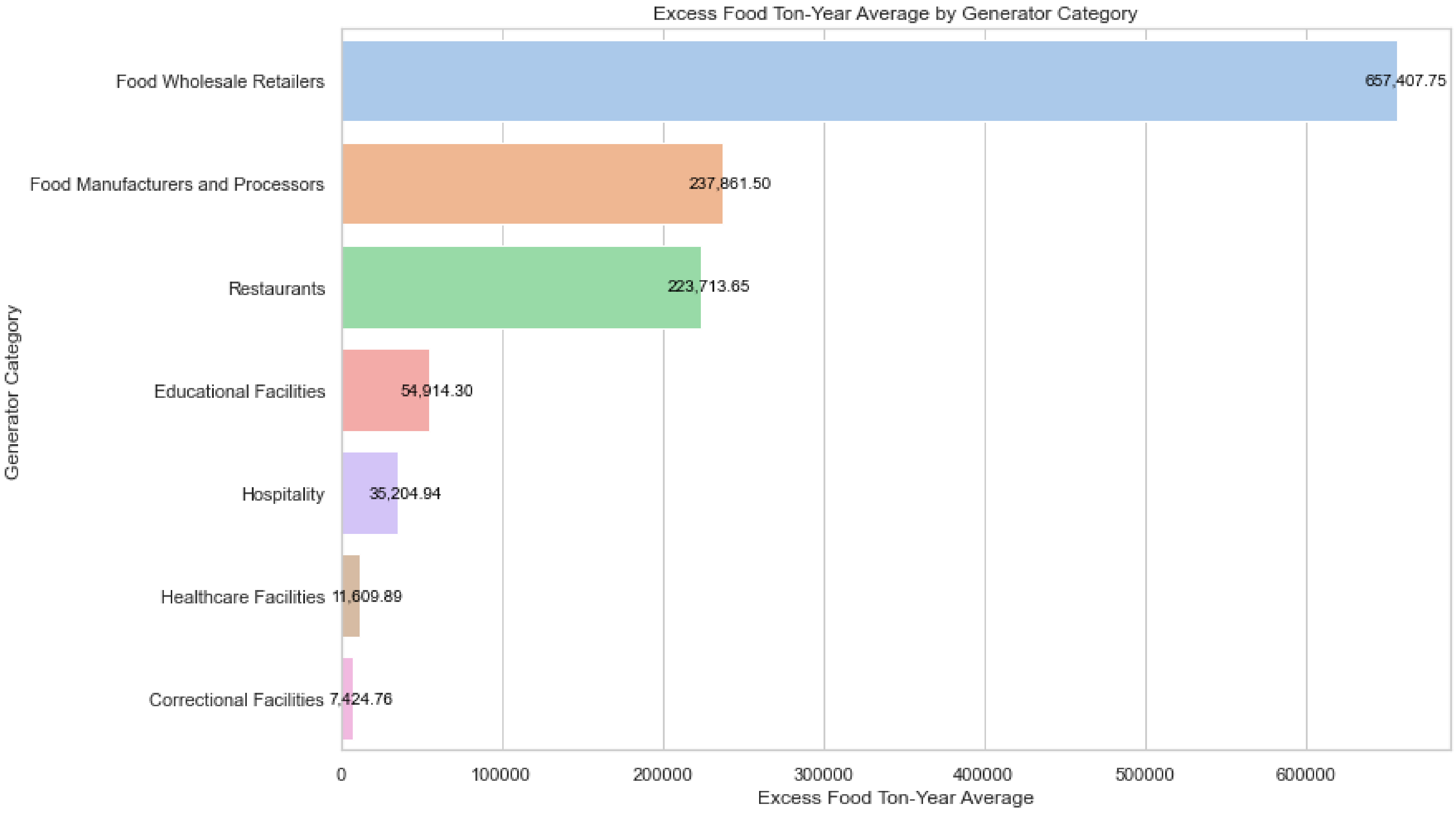
Proportional Distribution of Excess Food Ton-Year Average by Generator Category
53.5%
19.4%
18.2%
4.5%
2.9%
0.9%
0.6%
Generator Category & Percentage
🟦 Food Wholesale Retailers: 53.5%
🟧 Food Manufacturers and Processors: 19.4%
🟩 Restaurants: 18.2%
🟥 Educational Facilities: 4.5%
🟪 Hospitality: 2.9%
🟫 Healthcare Facilities: 0.9%
Correctional Facilities: 0.6%



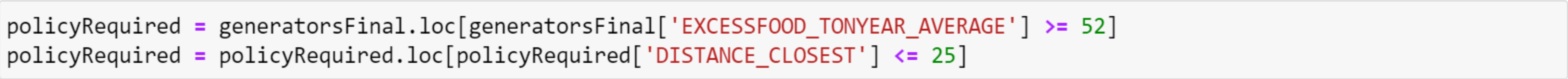
```
# COUNT OF GENERATORS CLOSEST TO EACH OF THE 2 FOOD WASTE RECYCLING FACILITIES IN NJ

genByRecFacility = generatorsFinal.groupby(['CLOSEST_RECYCLING_FACILITY'])[['CLOSEST_RECYCLING_FACILITY']].count()
genByRecFacility
```

	CLOSEST_RECYCLING_FACILITY
CLOSEST_RECYCLING_FACILITY	
CORe Elizabeth	23516
Either	45
Trenton Biogas	13112

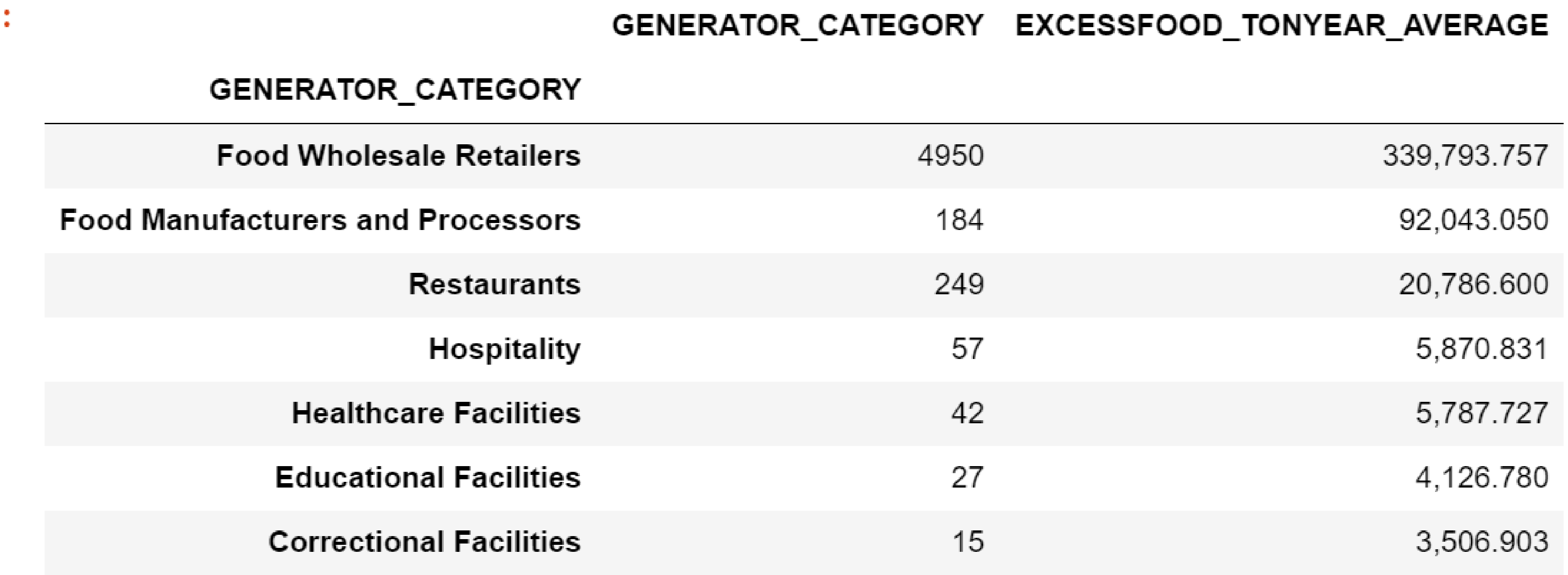



Excess Food Ton-Year Average by Generator Category
Generator Category
Food Wholesale Retailers
Food Manufacturers and Processors
Restaurants
Educational Facilities
Hospitality
Healthcare Facilities
Correctional Facilities
657,407.75
237,861.50
223,713.65
54,914.30
35,204.94
11,609.89
7,424.76
0
100000
200000
300000
400000
500000
600000
Excess Food Ton-Year Average

According to my analysis, about 5,497 generators must comply to the NJ policy



```
policyRequired = generatorsFinal.loc[generatorsFinal['EXCESSFOOD_TONYEAR_AVERAGE'] >= 52]
policyRequired = policyRequired.loc[policyRequired['DISTANCE_CLOSEST'] <= 25]
```



	GENERATOR_CATEGORY	EXCESSFOOD_TONYEAR_AVERAGE
GENERATOR_CATEGORY		
Food Wholesale Retailers	4950	339,793.757
Food Manufacturers and Processors	184	92,043.050
Restaurants	249	20,786.600
Hospitality	57	5,870.831
Healthcare Facilities	42	5,787.727
Educational Facilities	27	4,126.780
Correctional Facilities	15	3,506.903