Electricity_Reliability_CLEANED

December 28, 2022

[1]:
```
import pandas as pd
import numpy as np
#from scipy.stats import iqr
```

1 Quantifying Disparities in Electricity Reliability Data Processing & Cleaning

Merged Weather Data: weather_data_station.csv

Merged Outage Data: outage_data_final_typescleaned2.csv

Census2020_block: town_blocks.csv

Census2020: census_race_profile_2020.xlsx

Merged Income data: 2013_2021_income_data_final.csv

GIS merged weather outage data: weather_outage_towns.csv

2 Cleaning Weather data

[2]:
```
# read in census data by census blocks 2020
df = pd.read_csv('weather_data_station.csv')

# drop rows with nan values in 2 min wind speed column
df = df.dropna(subset=[ 'WSF2'])
df_small = df.dropna(subset=[ 'WSF2', 'PRCP', 'TMIN', 'TMAX'])
# dropping columns that will not be used
df_small = df_small.drop(['TOBS', 'WDF2', 'WESD', 'TAVG'], axis=1)
# filling nan vals in dummy variables to 0
df_small['WT03'] = df_small['WT03'].fillna(0)
df_small['WT04'] = df_small['WT04'].fillna(0)
df_small['WT05'] = df_small['WT05'].fillna(0)
df_small['WT06'] = df_small['WT06'].fillna(0)
df_small['WT08'] = df_small['WT08'].fillna(0)
df_small['WT09'] = df_small['WT09'].fillna(0)
df_small['WT11'] = df_small['WT11'].fillna(0)
```

```
# filling in nan values for snow with the monthly average snow amount for that␣
↪year
tem = df_small.groupby(['YEAR', 'MONTH'])[['SNOW']].mean().reset_index()
tem.rename(columns={'SNOW': 'SNOW_mean'}, inplace=True)
df_snow = pd.merge(df_small, tem, how='left', on=['YEAR', 'MONTH'])
df_snow.SNOW.fillna(df_snow.SNOW_mean, inplace=True)
# filling in nan values for average wind speed with the monthly average for␣
↪that year
win = df_snow.groupby(['YEAR', 'MONTH'])[['AWND']].mean().reset_index()
win.rename(columns={'AWND': 'AWND_mean'}, inplace=True)
df_weather = pd.merge(df_snow, win, how='left', on=['YEAR', 'MONTH'])
df_weather.AWND.fillna(df_weather.AWND_mean, inplace=True)
# drop monthly mean by year cols generated by above
df_weather = df_weather.drop( [ 'SNOW_mean', 'AWND_mean'], axis=1)
# viewing
df_weather.head()
```

[2]:
```
       STATION                             NAME  LATITUDE  LONGITUDE  \
0  USW00054756  ORANGE MUNICIPAL AIRPORT, MA US  42.56998  -72.28696
1  USW00054756  ORANGE MUNICIPAL AIRPORT, MA US  42.56998  -72.28696
2  USW00054756  ORANGE MUNICIPAL AIRPORT, MA US  42.56998  -72.28696
3  USW00054756  ORANGE MUNICIPAL AIRPORT, MA US  42.56998  -72.28696
4  USW00054756  ORANGE MUNICIPAL AIRPORT, MA US  42.56998  -72.28696

         DATE  AWND  PRCP      SNOW  TMAX  TMIN  …  WT03  WT04  WT05  WT06  \
0  2013-01-01   4.7   0.0  5.043011   1.7  -9.9  …   0.0   0.0   0.0   0.0
1  2013-01-02   3.4   0.0  5.043011  -3.2 -14.9  …   0.0   0.0   0.0   0.0
2  2013-01-03   0.6   0.0  5.043011  -5.5 -23.2  …   0.0   0.0   0.0   0.0
3  2013-01-04   2.6   0.0  5.043011   2.2 -15.5  …   0.0   0.0   0.0   0.0
4  2013-01-05   3.4   0.0  5.043011   3.3  -9.3  …   0.0   0.0   0.0   0.0

   WT08  WT09  WT11  YEAR  MONTH  DAY
0   0.0   0.0   0.0  2013      1    1
1   0.0   0.0   0.0  2013      1    2
2   0.0   0.0   0.0  2013      1    3
3   0.0   0.0   0.0  2013      1    4
4   0.0   0.0   0.0  2013      1    5

[5 rows x 21 columns]
```

3 Cleaning Outage data

[3]:
```
# reading in the outage dataset that has years 2013-2021
outage = pd.read_csv('outage_data_final_typescleaned2.csv')
```

```
# removing unnecessary columns from the outage data - such as what streets were␣
↪affected and the voltage level
outage_clean = outage.drop(['date_in','time_in', 'street', 'company_name',
                            'voltage_levels',␣
↪'circuit_type','failed_or_damaged_equipment',
                            'weather_condition_type','load_type' ], axis=1)
# removing rows where the number of customers is less than zero as this is not␣
↪possible
outage_clean = outage_clean[outage_clean['number_of_customers_affected']>=0]
# removing rows where the outage duration is less than zero as this is not␣
↪possible
outage_clean = outage_clean[outage_clean['outage_duration']>=0]
# change column names to all CAPS
outage_clean.columns = outage_clean.columns.str.upper()
# capitalizing the first letter of every town name
outage_clean['CITY_TOWN'] = outage_clean['CITY_TOWN'].str.capitalize()
# re-naming the date col
outage_clean=outage_clean.rename(columns = {'DATE_OUT':'DATE'})
# dropping any completely duplicare rows from DF as there were some duplicates␣
↪entried with different capitalizations earlier
outage_clean = outage_clean.drop_duplicates()
# only keeping rows with a town name in them
outage_clean = outage_clean[outage_clean['CITY_TOWN'].notna()]

## changing town names to be consistent across all datasets used (i.e. 'E.␣
↪bridgewater' should be 'East bridgewater')
# boston downtown to boston
outage_clean['CITY_TOWN'] = outage_clean['CITY_TOWN'].replace({'Boston␣
↪downtown': 'Boston'})
# Bourne plymouth to Bourne
outage_clean['CITY_TOWN'] = outage_clean['CITY_TOWN'].replace({'Bourne␣
↪plymouth': 'Bourne'})
# E. bridgewater to east bridgewater
##outage_clean["CITY_TOWN"] = outage_clean["CITY_TOWN"].apply(lambda x: x.
↪replace("E. bridgewater", "East bridgewater"))
outage_clean['CITY_TOWN'] = outage_clean['CITY_TOWN'].replace({'E. bridgewater':
↪ 'East bridgewater'})
# East boston to boston
##outage_clean["CITY_TOWN"] = outage_clean["CITY_TOWN"].apply(lambda x: x.
↪replace("East boston", "Boston"))
outage_clean['CITY_TOWN'] = outage_clean['CITY_TOWN'].replace({'East boston':␣
↪'Boston'})
# Manchester-by-the-sea to  Manchester by the sea
##outage_clean["CITY_TOWN"] = outage_clean["CITY_TOWN"].apply(lambda x: x.
↪replace("Manchester-by-the-sea", "Manchester by the sea"))
```

```
outage_clean['CITY_TOWN'] = outage_clean['CITY_TOWN'].
 ↪replace({'Manchester-by-the-sea': 'Manchester by the sea'})
# Mt. washington to Mount washington
##outage_clean["CITY_TOWN"] = outage_clean["CITY_TOWN"].apply(lambda x: x.
 ↪replace("Mt. washington", "Mount washington"))
outage_clean['CITY_TOWN'] = outage_clean['CITY_TOWN'].replace({'Mt. washington':
 ↪ 'Mount washington'})
# Manchester to Manchester by the sea
outage_clean['CITY_TOWN'] = outage_clean['CITY_TOWN'].replace({'Manchester':␣
 ↪'Manchester by the sea'})
# New marlboro to New marlborough
outage_clean['CITY_TOWN'] = outage_clean['CITY_TOWN'].replace({'New marlboro':␣
 ↪'New marlborough'})
# Changing Brighton, Charlestown, Dorchester, Hyde park, Roxbury, South boston,␣
 ↪West roxbury to Boston as they are all part of Boston and only called boston␣
 ↪in other datasets
outage_clean['CITY_TOWN'] = outage_clean['CITY_TOWN'].replace({'Brighton':␣
 ↪'Boston'})
outage_clean['CITY_TOWN'] = outage_clean['CITY_TOWN'].replace({'Charlestown':␣
 ↪'Boston'})
outage_clean['CITY_TOWN'] = outage_clean['CITY_TOWN'].replace({'Dorchester':␣
 ↪'Boston'})
outage_clean['CITY_TOWN'] = outage_clean['CITY_TOWN'].replace({'Hyde park':␣
 ↪'Boston'})
outage_clean['CITY_TOWN'] = outage_clean['CITY_TOWN'].replace({'Roxbury':␣
 ↪'Boston'})
outage_clean['CITY_TOWN'] = outage_clean['CITY_TOWN'].replace({'South boston':␣
 ↪'Boston'})
outage_clean['CITY_TOWN'] = outage_clean['CITY_TOWN'].replace({'West roxbury':␣
 ↪'Boston'})
# making the col a srting
outage_clean['CITY_TOWN'] = outage_clean['CITY_TOWN'].astype(str)
# removing rows where the number of customers is less than zero as this is not␣
 ↪possible
outage_clean = outage_clean[outage_clean['NUMBER_OF_CUSTOMERS_AFFECTED']>=0]
# removing rows where the outage duration is less than zero as this is not␣
 ↪possible
outage_clean = outage_clean[outage_clean['OUTAGE_DURATION']>=0]
# viewing
outage_clean.head()
```

[3]:

	GEOID	DATE	TIME_OUT	DAY	MONTH	YEAR	OUTAGE_DURATION	\
0	2.501770e+09	3/25/2014	14:01:00	25	3	2014	1.400	
1	2.501770e+09	4/22/2014	14:23:39	22	4	2014	1.082	
2	2.502737e+09	5/15/2014	15:30:00	15	5	2014	4.033	
3	2.502724e+09	5/21/2014	15:07:19	21	5	2014	0.678	

```
4  2.502737e+09   6/7/2014  13:22:24    7      6  2014            1.877

   NUMBER_OF_CUSTOMERS_AFFECTED  CITY_TOWN REASON_FOR_OUTAGE  LATITUDE  \
0                             2   Townsend  Action By Others   42.6669
1                            72   Townsend  Action By Others   42.6669
2                             1  Lunenburg  Action By Others   42.5897
3                           216  Fitchburg  Action By Others   42.5912
4                          1206  Lunenburg  Action By Others   42.5897

   LONGITUDE
0   -71.7116
1   -71.7116
2   -71.7199
3   -71.8156
4   -71.7199
```

The outage dataset was run throu GIS to match counties to their nearest sensor. **Final_outage_town_list_station.csv** is the output of the GIS manipulation.

[4]:
```
# reading in the outage dataset after GIS anaysis
df_out_we_towns = pd.read_csv('weather_outage_towns.csv')

df_out_we_towns.columns

## catching missed changes needed to town names to be consistent across all␣
↪datasets used (i.e. 'E. bridgewater' should be'East bridgewater')

# change column names to all CAPS
df_out_we_towns.columns = df_out_we_towns.columns.str.upper()
# capitalizing the first letter of every town name
df_out_we_towns['TOWN'] = df_out_we_towns['TOWN'].str.capitalize()
# re-naming the date col
df_out_we_towns=df_out_we_towns.rename(columns = {'DATE_OUT':'DATE'})
# dropping any completely duplicare rows from DF as there were some duplicates␣
↪entried with different capitalizations earlier
df_out_we_towns = df_out_we_towns.drop_duplicates()
# only keeping rows with a town name in them
df_out_we_towns = df_out_we_towns[df_out_we_towns['TOWN'].notna()]

## changing town names to be consistent across all datasets used (i.e. 'E.␣
↪bridgewater' should be 'East bridgewater')
# boston downtown to boston
df_out_we_towns['TOWN'] = df_out_we_towns['TOWN'].replace({'Boston downtown':␣
↪'Boston'})
# Bourne plymouth to Bourne
df_out_we_towns['TOWN'] = df_out_we_towns['TOWN'].replace({'Bourne plymouth':␣
↪'Bourne'})
```

```
# E. bridgewater to east bridgewater
##outage_clean["CITY_TOWN"] = outage_clean["CITY_TOWN"].apply(lambda x: x.
↪replace("E. bridgewater", "East bridgewater"))
df_out_we_towns['TOWN'] = df_out_we_towns['TOWN'].replace({'E. bridgewater':␣
↪'East bridgewater'})
# East boston to boston
##outage_clean["CITY_TOWN"] = outage_clean["CITY_TOWN"].apply(lambda x: x.
↪replace("East boston", "Boston"))
df_out_we_towns['TOWN'] = df_out_we_towns['TOWN'].replace({'East boston':␣
↪'Boston'})
# Manchester-by-the-sea to  Manchester by the sea
##outage_clean["CITY_TOWN"] = outage_clean["CITY_TOWN"].apply(lambda x: x.
↪replace("Manchester-by-the-sea", "Manchester by the sea"))
df_out_we_towns['TOWN'] = df_out_we_towns['TOWN'].
↪replace({'Manchester-by-the-sea': 'Manchester by the sea'})
# Mt. washington to Mount washington
##outage_clean["CITY_TOWN"] = outage_clean["CITY_TOWN"].apply(lambda x: x.
↪replace("Mt. washington", "Mount washington"))
df_out_we_towns['TOWN'] = df_out_we_towns['TOWN'].replace({'Mt. washington':␣
↪'Mount washington'})

# Manchester to Manchester by the sea
df_out_we_towns['TOWN'] = df_out_we_towns['TOWN'].replace({'Manchester':␣
↪'Manchester by the sea'})
# New marlboro to New marlborough
df_out_we_towns['TOWN'] = df_out_we_towns['TOWN'].replace({'New marlboro': 'New␣
↪marlborough'})
# remove Brighton, Charlestown, Dorchester, Hyde park, Roxbury, South boston,␣
↪West roxbury
df_out_we_towns['TOWN'] = df_out_we_towns['TOWN'].replace({'Brighton':␣
↪'Boston'})
df_out_we_towns['TOWN'] = df_out_we_towns['TOWN'].replace({'Charlestown':␣
↪'Boston'})
df_out_we_towns['TOWN'] = df_out_we_towns['TOWN'].replace({'Dorchester':␣
↪'Boston'})
df_out_we_towns['TOWN'] = df_out_we_towns['TOWN'].replace({'Hyde park':␣
↪'Boston'})
df_out_we_towns['TOWN'] = df_out_we_towns['TOWN'].replace({'Roxbury': 'Boston'})
df_out_we_towns['TOWN'] = df_out_we_towns['TOWN'].replace({'South boston':␣
↪'Boston'})
df_out_we_towns['TOWN'] = df_out_we_towns['TOWN'].replace({'West roxbury':␣
↪'Boston'})



# removing rows where the number of customers is less than zero as this is not␣
↪possible
```

```
df_out_we_towns =␣
 ↪df_out_we_towns[df_out_we_towns['NUMBER_OF_CUSTOMERS_AFFECTED']>=0]
# removing rows where the outage duration is less than zero as this is not␣
 ↪possible
df_out_we_towns = df_out_we_towns[df_out_we_towns['OUTAGE_DURATION']>=0]
# viewing
df_out_we_towns.head()
```

[4]:
```
   OID_  JOIN_COUNT  TARGET_FID  FIELD1  UNNAMED__0         GEOID  \
0     1           1           1       0           0  2.501770e+09
1     2           1           2       1           1  2.501770e+09
2     3           1           3       2          19  2.501770e+09
3     4           1           4       3          20  2.501770e+09
4     5           1           5       4          22  2.501770e+09

                DATE  TIME_OUT  DAY  MONTH  …  WT03  WT04  WT05 WT06 WT08  \
0  3/25/2014 0:00:00  14:01:00   25      3  …   0.0   0.0   0.0  0.0  0.0
1  4/22/2014 0:00:00  14:23:39   22      4  …   0.0   0.0   0.0  0.0  0.0
2   9/2/2016 0:00:00  11:12:40    2      9  …   0.0   0.0   0.0  0.0  0.0
3   9/2/2016 0:00:00  11:12:40    2      9  …   0.0   0.0   0.0  0.0  0.0
4   3/3/2017 0:00:00   6:01:00    3      3  …   0.0   0.0   0.0  0.0  0.0

   WT09  WT11  YEAR_1 MONTH_1 DAY_1
0   0.0   0.0    2013       1     1
1   0.0   0.0    2013       1     1
2   0.0   0.0    2013       1     1
3   0.0   0.0    2013       1     1
4   0.0   0.0    2013       1     1

[5 rows x 39 columns]
```

4 Cleaning Census block data to get population density

[5]:
```
# read in census data by census blocks 2020
town = pd.read_csv('Town_blocks.csv')

# drop nan values rows
town = town.dropna(subset=['TOWN'])
# making eveything one uppercase letter
town['TOWN'] = town['TOWN'].str.capitalize()
# changing manchester by the sea
town['TOWN'] = town['TOWN'].replace({'Manchester': 'Manchester by the sea'})
# viewing
town.head()
```

[5]:
```
   OID_  Join_Count  TARGET_FID  JOIN_FID         GEOID        NAME  \
0     1           1           1       176  2.500390e+14  BLOCK 2058
1     2           1           2       176  2.500390e+14  BLOCK 2048
2     3           1           3       176  2.500390e+14  BLOCK 2049
3     4           1           4       244  2.500390e+14  BLOCK 3035
4     5           1           5       244  2.500390e+14  BLOCK 3036

        County_Name    C_NAMES     State_Name  F_Population18Years_Over  …  \
0  Berkshire County  Berkshire  Massachusetts                     100.0  …
1  Berkshire County  Berkshire  Massachusetts                     100.0  …
2  Berkshire County  Berkshire  Massachusetts                       NaN  …
3  Berkshire County  Berkshire  Massachusetts                     100.0  …
4  Berkshire County  Berkshire  Massachusetts                     100.0  …

   POP2000  POP2010  POPCH80_90  POPCH90_00  POPCH00_10  HU2010  FOURCOLOR  \
0    130.0    167.0        35.0        -5.0        37.0   148.0        1.0
1    130.0    167.0        35.0        -5.0        37.0   148.0        1.0
2    130.0    167.0        35.0        -5.0        37.0   148.0        1.0
3   3335.0   3257.0       158.0       425.0       -78.0  1751.0        2.0
4   3335.0   3257.0       158.0       425.0       -78.0  1751.0        2.0

   TYPE  AREA_ACRES   SQ_MILES
0     T  14315.1223  22.367379
1     T  14315.1223  22.367379
2     T  14315.1223  22.367379
3     T  31081.9779  48.565591
4     T  31081.9779  48.565591

[5 rows x 45 columns]
```

[6]:
```
# Get data
data =  town #pd.read_csv('TOWNS_BLOCKS.csv')
# Retrieve unique list of towns
towns = data.TOWN.unique()

# Running the conversion factor on land area for blocks from square meters to␣
↪square miles
data['block_sq_miles'] = np.array(data['AREALAND'])*0.00000038610

# Calculating population densities by block with units in people per square mile
population = np.array(data['POP100'])
area = np.array(data['block_sq_miles'])

population_density = []
for i in range(len(population)):
    if area[i] == 0:
        population_density.append(0)
```

```
    else:
        population_density.append(population[i]/area[i])

data['population_density'] = population_density

# Aggregating the densities by weight/percent of population for the town within␣
↪each block
town_weighted_densities = []
for town in towns:
    town_data = data.loc[data['TOWN']==town]
    town_pop = town_data['POP100'].sum()
    sum_list = []
    for i in town_data.index:
        density = town_data['population_density'][i]
        block_pop = town_data['POP100'][i]
        weight = block_pop/town_pop
        weighted_item = weight * density
        sum_list.append(weighted_item)
    weighted_density = sum(sum_list)
    town_weighted_densities.append((town, weighted_density))

# Turn above array into dataframe
density_df = pd.DataFrame(town_weighted_densities, columns=['town',␣
↪'weighted_density'])
# viewing
density_df.head()
```

[6]:

	town	weighted_density
0	Mount washington	69.485570
1	Sheffield	277.009963
2	Egremont	252.534570
3	Great barrington	1761.960157
4	Alford	91.574317

[7]:
```
# read population density calculated from the 2020 census
den = density_df #pd.read_csv('town_weighted_density.csv')

# rename cols
den.rename(columns = {'town':'TOWN', 'weighted_density':'WEIGHTED_DENSITY'},␣
↪inplace = True)
# making density levels high, average, and low
den_data = den['WEIGHTED_DENSITY'].astype(np.float64)
data_array = np.array(den_data)
middle_bottom, middle, middle_top = np.percentile(data_array, [25,50,75])
minimum = np.min(data_array)
maximum = np.max(data_array)
#
```

```python
den_group = []
for density in den_data:
    if density >= minimum and density < middle_bottom:
        den_group.append('low density')
    elif density >= middle_bottom and density < middle:
        den_group.append('low middle density')
    elif density >= middle and density < middle_top:
        den_group.append('high middle density')
    elif density >= middle_top and density <= maximum:
        den_group.append('high density')
den['DENSITY_GROUP'] = den_group
# viewing
den.head()
```

[7]:

	TOWN	WEIGHTED_DENSITY	DENSITY_GROUP
0	Mount washington	69.485570	low density
1	Sheffield	277.009963	low density
2	Egremont	252.534570	low density
3	Great barrington	1761.960157	low middle density
4	Alford	91.574317	low density

[8]:

```python
print('Lower Population Density Score:', round(minimum,2), '- 793.81')
print('Lower_Middle Population Density Score:', round(middle_bottom,2), '- 2115.
↪4')
print('Upper_Middle Population Density Score:', round(middle,2), '4601.17')
print('Upper Population Density Score:',␣
↪round(middle_top,2),'-',round(maximum,2))
```

```
Lower Population Density Score: 55.92 - 793.81
Lower_Middle Population Density Score: 793.82 - 2115.4
Upper_Middle Population Density Score: 2115.5 4601.17
Upper Population Density Score: 4601.18 - 57121.5
```

5 Census 2020 data with race percentages

[9]:

```python
# read in race percentage data by town from the 2020 census
race = pd.read_excel('census_race_profile_2020.xlsx', header=1)

# rename col
race.rename(columns = {'MCD':'TOWN'}, inplace = True)
# capitalize the first letter of town names
race['TOWN'] = race['TOWN'].str.capitalize()
# changing manchester by the sea
race['TOWN'] = race['TOWN'].replace({'Manchester': 'Manchester by the sea'})
# view
race.head()
```

[9]:
```
       TOWN  White Alone (Non-Hispanic)  \
0  Abington                    0.866493
1     Acton                    0.690369
2  Acushnet                    0.940322
3     Adams                    0.932016
4    Agawam                    0.890145

   Black or African American Alone (Non-Hispanic)  Asian Alone (Non-Hispanic)  \
0                                        0.029473                    0.021394
1                                        0.017499                    0.219523
2                                        0.004650                    0.004410
3                                        0.009249                    0.004865
4                                        0.015964                    0.021407

   Hispanic or Latino (of Any Race)  \
0                          0.029352
1                          0.032212
2                          0.017975
3                          0.019458
4                          0.048433

   American Indian and Alaska Native Alone (Non-Hispanic)  \
0                                           0.002027
1                                           0.000675
2                                           0.001821
3                                           0.000901
4                                           0.000998

   Hawaiian Native or Pacific Islander Alone (Non-Hispanic)  \
0                                           0.000333
1                                           0.000152
2                                           0.000000
3                                           0.000120
4                                           0.000053

   Some Other Race Alone (Non-Hispanic)  \
0                              0.018428
1                              0.008510
2                              0.007334
3                              0.001321
4                              0.001960

   Two or More Races Alone (Non-Hispanic)
0                                0.032499
1                                0.031059
2                                0.023488
3                                0.032070
```

```
4                                  0.021040
```

[10]:
```python
# making income levels of lower, middle, and upper
diversity_data = race['White Alone (Non-Hispanic)'].astype(np.float64)
data_array = np.array(diversity_data)
middle_bottom, middle, middle_top = np.percentile(data_array, [25,50,75])
minimum = np.min(data_array)
maximum = np.max(data_array)
#
diversity_group = []
for diversity in diversity_data:
    if diversity >= minimum and diversity < middle_bottom:
        diversity_group.append('high diversity')
    elif diversity >= middle_bottom and diversity < middle:
        diversity_group.append('high middle diversity')
    elif diversity >= middle and diversity < middle_top:
        diversity_group.append('low middle diversity')
    elif diversity >= middle_top and diversity <= maximum:
        diversity_group.append('low diversity')
race['DIVERSITY_GROUP'] = diversity_group

# viewing
race.head()
```

[10]:
```
       TOWN  White Alone (Non-Hispanic)  \
0  Abington                    0.866493
1     Acton                    0.690369
2  Acushnet                    0.940322
3     Adams                    0.932016
4    Agawam                    0.890145

   Black or African American Alone (Non-Hispanic)  Asian Alone (Non-Hispanic)  \
0                                        0.029473                    0.021394
1                                        0.017499                    0.219523
2                                        0.004650                    0.004410
3                                        0.009249                    0.004865
4                                        0.015964                    0.021407

   Hispanic or Latino (of Any Race)  \
0                          0.029352
1                          0.032212
2                          0.017975
3                          0.019458
4                          0.048433

   American Indian and Alaska Native Alone (Non-Hispanic)  \
0                                           0.002027
```

```
1                                             0.000675
2                                             0.001821
3                                             0.000901
4                                             0.000998

   Hawaiian Native or Pacific Islander Alone (Non-Hispanic)  \
0                                             0.000333
1                                             0.000152
2                                             0.000000
3                                             0.000120
4                                             0.000053

   Some Other Race Alone (Non-Hispanic)  \
0                              0.018428
1                              0.008510
2                              0.007334
3                              0.001321
4                              0.001960

   Two or More Races Alone (Non-Hispanic)        DIVERSITY_GROUP
0                                0.032499  high middle diversity
1                                0.031059         high diversity
2                                0.023488          low diversity
3                                0.032070   low middle diversity
4                                0.021040  high middle diversity
```

[11]:

```
print('Lower Diversity Score:', round(minimum,2), '- .82')
print('Lower_Middle Diversity Score:', round(middle_bottom,2), '- .89')
print('Upper_Middle Diversity Score:', round(middle,2), '.92')
print('Upper Diversity Score:', round(middle_top,2),'-',round(maximum,2))
```

```
Lower Diversity Score: 0.16 - .82
Lower_Middle Diversity Score: 0.83 - .89
Upper_Middle Diversity Score: 0.9 .92
Upper Diversity Score: 0.93 - 0.97
```

6 Cleaning Income data

[12]:

```
# read in income data fro years 2013-2021
income = pd.read_csv('2013_2021_income_data_final.csv')

# dropping columns that will not be used
income = income.drop(columns=['NameLSAD', 'DOR Code', 'LEA Code'])
# capitalizing the first letter of each town
income['Municipality'] = income['Municipality'].str.capitalize()
# re- naming columns
```

```
income.columns = ['NAME', 'COUNTY', 'YEAR', 'POP', 'INCOME', 'INC_PER_CAPITA',␣
 ↪'LAT', 'LONG' ]
# viewing
income.head()
```

[12]:

```
       NAME     COUNTY  YEAR     POP         INCOME INC_PER_CAPITA        LAT  \
0  Abington   PLYMOUTH  2013  15,985    453,616,000         28,378  42.119964
1     Acton  MIDDLESEX  2013  21,924  1,085,528,000         49,513  42.483953
2  Acushnet    BRISTOL  2013  10,303    250,518,000         24,315  41.718218
3     Adams  BERKSHIRE  2013   8,485    154,561,000         18,216  42.625560
4    Agawam    HAMPDEN  2013  28,438    695,498,000         24,457  42.064731

        LONG
0 -70.957216
1 -71.438495
2 -70.901151
3 -73.119828
4 -72.653477
```

[13]:

```
# read in cleaned income data fro years 2013-2021
inc = income #pd.read_csv('2013_2021_INCOME_DATA.csv')
# rename col
inc.rename(columns = {'NAME':'TOWN'}, inplace = True)

# making income levels of lower, middle, and upper
income_data = inc['INC_PER_CAPITA'].map(lambda x: x.replace(',', '')).astype(np.
 ↪float64)
data_array = np.array(income_data)
middle_bottom, middle, middle_top = np.percentile(data_array, [25,50,75])
minimum = np.min(data_array)
maximum = np.max(data_array)
#
income_group = []
for income in income_data:
    if income >= minimum and income < middle_bottom:
        income_group.append('lower income')
    elif income >= middle_bottom and income < middle:
        income_group.append('lower middle income')
    elif income >= middle and income < middle_top:
        income_group.append('upper middle income')
    elif income >= middle_top and income <= maximum:
        income_group.append('upper income')
inc['INCOME_GROUP'] = income_group

# viewing
inc.head()
```

[13]:
```
        TOWN     COUNTY  YEAR     POP         INCOME INC_PER_CAPITA        LAT  \
0  Abington   PLYMOUTH  2013  15,985    453,616,000         28,378  42.119964
1     Acton  MIDDLESEX  2013  21,924  1,085,528,000         49,513  42.483953
2  Acushnet    BRISTOL  2013  10,303    250,518,000         24,315  41.718218
3     Adams  BERKSHIRE  2013   8,485    154,561,000         18,216  42.625560
4    Agawam    HAMPDEN  2013  28,438    695,498,000         24,457  42.064731

        LONG         INCOME_GROUP
0 -70.957216  lower middle income
1 -71.438495         upper income
2 -70.901151         lower income
3 -73.119828         lower income
4 -72.653477         lower income
```

[14]:
```python
print('Lower Income Score:', round(minimum), '- 25457')
print('Lower_Middle Income Score:', round(middle_bottom), '- 32107')
print('Upper_Middle Income Score:', round(middle), '- 43399')
print('Upper Income Score:', round(middle_top),'-',round(maximum))
```

```
Lower Income Score: 5545 - 25457
Lower_Middle Income Score: 25458 - 32107
Upper_Middle Income Score: 32108 - 43399
Upper Income Score: 43399 - 386499
```

make a town long lat file

[15]:
```python
town_long_lat = inc.drop_duplicates('TOWN', keep='first')
town_long_lat = town_long_lat[['TOWN','LAT','LONG']]
town_long_lat.head()
```

[15]:
```
        TOWN        LAT       LONG
0  Abington  42.119964 -70.957216
1     Acton  42.483953 -71.438495
2  Acushnet  41.718218 -70.901151
3     Adams  42.625560 -73.119828
4    Agawam  42.064731 -72.653477
```

7 Merging Datasets and addin Y_vars

[16]:
```python
# use cleaed outage data
df_outage = outage_clean #pd.read_csv('OUTAGE_CLEAN_FINAL.csv')
# use town lat long data
df_town = town_long_lat #pd.read_csv('Town_lat_long.csv')

# dropping cols that are not needed
df_outage.drop(['LATITUDE','LONGITUDE'],axis=1, inplace=True)
# re-naming cols
df_outage.rename(columns = {'CITY_TOWN':'TOWN'}, inplace = True)
```

```python
# merging town and outage data on town name
merged_outage_town = pd.merge(df_outage, df_town, on ='TOWN')

# only keeping important columns from the cleaned post GIS outage data
df_final_outage = df_out_we_towns[[ u'DATE', u'TIME_OUT', u'DAY', u'MONTH',
       u'YEAR', u'OUTAGE_DURATION', u'NUMBER_OF_CUSTOMERS_AFFECTED', u'TOWN',
       u'REASON_FOR_OUTAGE', u'STATION',
       u'NAME', u'LATITUDE', u'LONGITUDE']]

# make column date-time type
df_weather["DATE"] = pd.to_datetime(df_weather["DATE"])
# make column date-time type
df_final_outage["DATE"] = pd.to_datetime(df_final_outage["DATE"])
# merge outsge and weather data on date and name
df2 = pd.merge(df_final_outage,␣
 ↪df_weather,how='left',left_on=['DATE','NAME'],right_on =['DATE','NAME'])
# viewing
df2.head()
```

```
<ipython-input-16-2ff81ea0c1a7>:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_final_outage["DATE"] = pd.to_datetime(df_final_outage["DATE"])
```

[16]:

```
        DATE  TIME_OUT  DAY_x  MONTH_x  YEAR_x  OUTAGE_DURATION  \
0 2014-03-25  14:01:00     25        3    2014            1.400
1 2014-04-22  14:23:39     22        4    2014            1.082
2 2016-09-02  11:12:40      2        9    2016            3.789
3 2016-09-02  11:12:40      2        9    2016            3.456
4 2017-03-03   6:01:00      3        3    2017            1.900

   NUMBER_OF_CUSTOMERS_AFFECTED      TOWN REASON_FOR_OUTAGE    STATION_x  …  \
0                             2  Townsend   Action By Others  USW00004780  …
1                            72  Townsend   Action By Others  USW00004780  …
2                            10  Townsend   Action By Others  USW00004780  …
3                            35  Townsend   Action By Others  USW00004780  …
4                           177  Townsend   Action By Others  USW00004780  …

  WT03  WT04  WT05 WT06  WT08  WT09  WT11  YEAR_y  MONTH_y  DAY_y
0  0.0   0.0   0.0  0.0   0.0   0.0   0.0  2014.0      3.0   25.0
1  0.0   0.0   0.0  0.0   0.0   0.0   0.0  2014.0      4.0   22.0
2  0.0   0.0   0.0  0.0   0.0   0.0   0.0  2016.0      9.0    2.0
3  0.0   0.0   0.0  0.0   0.0   0.0   0.0  2016.0      9.0    2.0
4  0.0   0.0   0.0  0.0   0.0   0.0   0.0  2017.0      3.0    3.0
```

[5 rows x 32 columns]

[17]:
```
#dropping NA rows from outages that did not have weather data (7107 dropped and␣
↪204548 kept about 3% of rows dropped)
wo_clean = df2.dropna()
# list of cols that will be dropped bc they are duplicates from merge or will␣
↪not be used in analysis
drops = ['REASON_FOR_OUTAGE', 'NAME',
         'STATION_x', 'STATION_y', 'TIME_OUT', 'DATE',
         'LATITUDE_y', 'LONGITUDE_y', 'YEAR_y', 'MONTH_y', 'DAY_y']
# dropping cols from list above
wo_cleaned = wo_clean.drop(drops, axis=1)
# re naming columns
wo_cleaned.rename(columns = {'DAY_x':'DAY', 'MONTH_x':'MONTH', 'YEAR_x':'YEAR',
                             'LATITUDE_x':'LATITUDE', 'LONGITUDE_x':'LONGITUDE'
                             }, inplace = True)
# viewing
wo_cleaned.head()
```

[17]:
```
   DAY  MONTH  YEAR  OUTAGE_DURATION  NUMBER_OF_CUSTOMERS_AFFECTED      TOWN  \
0   25      3  2014            1.400                             2  Townsend
1   22      4  2014            1.082                            72  Townsend
2    2      9  2016            3.789                            10  Townsend
3    2      9  2016            3.456                            35  Townsend
4    3      3  2017            1.900                           177  Townsend

   LATITUDE  LONGITUDE  AWND  PRCP  …  TMAX  TMIN  WSF2  WT03  WT04  WT05  \
0  42.55495  -71.75699   2.2   0.0  …   2.8 -10.5   7.6   0.0   0.0   0.0
1  42.55495  -71.75699   3.1   0.3  …  25.6   8.9   8.9   0.0   0.0   0.0
2  42.55495  -71.75699   2.0   0.0  …  27.2  12.8   6.3   0.0   0.0   0.0
3  42.55495  -71.75699   2.0   0.0  …  27.2  12.8   6.3   0.0   0.0   0.0
4  42.55495  -71.75699   4.8   0.0  …   2.8  -7.1  11.2   0.0   0.0   0.0

   WT06  WT08  WT09  WT11
0   0.0   0.0   0.0   0.0
1   0.0   0.0   0.0   0.0
2   0.0   0.0   0.0   0.0
3   0.0   0.0   0.0   0.0
4   0.0   0.0   0.0   0.0

[5 rows x 21 columns]
```

[18]:
```
# merging weather-outage data with income data by town and year
wo_inc = wo_cleaned.merge(inc, on=['TOWN','YEAR'], how='left')

# merging weather-outage-income data with population density data by town
```

```
wo_inc_den = wo_inc.merge(den, on=['TOWN'], how='left')

# merging weather-outage-income-density data with race percentage data by town
wo_inc_den_race = wo_inc_den.merge(race, on=['TOWN'], how='left')

# cleaning dropping duplicate cols in df that are not needed
wo_inc_den_race_clean = wo_inc_den_race.drop(['LAT', 'LONG'], axis=1)

# Calculating the first Y_var
wo_inc_den_race_clean_y = wo_inc_den_race_clean
wo_inc_den_race_clean_y['Y_VAR_TIME_x_PEOPLE'] =␣
↪wo_inc_den_race_clean_y['NUMBER_OF_CUSTOMERS_AFFECTED'] *␣
↪wo_inc_den_race_clean_y['OUTAGE_DURATION']
```

[19]:
```
## Changing column names to be more readable
df = wo_inc_den_race_clean_y.rename(columns={'AWND':␣
↪'Average_daily_wind_speed', 'PRCP': 'Precipitation','TMAX':'Temp_max','TMIN':
↪'Temp_min',
                        'WSF2':'Fastest_Two_Min_Wind_Speed','WT03':
↪'Thunder','WT04':'Ice_pellets','WT05':'Hail',
                        'WT06':'Glaze','WT08':'Smoke','WT09':
↪'drifting_snow','WT11':'High_damaging_winds','POP':'Population',
                        'INC_PER_CAPITA':'Income_per_capita',
                        'WEIGHTED_DENSITY':'Pop_density','White Alone␣
↪(Non-Hispanic)':'White_percent',
                        'Black or African American Alone (Non-Hispanic)':
↪'Black_percent',
                        'Asian Alone (Non-Hispanic)':'Asian_percent',
                        'Hispanic or Latino (of Any Race)':'Hispanic_percent',
                        'American Indian and Alaska Native Alone (Non-Hispanic)':
↪'Alaskan_Native',
                        'Hawaiian Native or Pacific Islander Alone (Non-Hispanic)':
↪'Pacific_Native',
                        'Some Other Race Alone (Non-Hispanic)':'other_race',
                        'Two or More Races Alone (Non-Hispanic)':
↪'Two_or_more_races',
                        'Y_VAR_TIME_x_PEOPLE':'CUSTOMER_OUTAGE_HOURS'
                        })
```

[20]:
```
# removing comma from pop col
df['Population'] = df['Population'].map(lambda x: x.replace(',', '')).astype(np.
↪float64)
#  removing comma from inc per cap col
df['Income_per_capita'] = df['Income_per_capita'].map(lambda x: x.replace(',',␣
↪'')).astype(np.float64)
# removing comma from income col
```

```
df['INCOME'] = df['INCOME'].map(lambda x: x.replace(',', '')).astype(np.float64)
# calculating SAIDI
df['SAIDI']=df['CUSTOMER_OUTAGE_HOURS']/df['Population']
# making all cols uppercase letters
df.columns = df.columns.str.upper()
# viewing
df.tail()
```

[20]:

```
        DAY  MONTH  YEAR  OUTAGE_DURATION  NUMBER_OF_CUSTOMERS_AFFECTED  \
204535   20     11  2013            2.833                           141
204536   29     10  2018            1.337                             1
204537   24     11  2013            1.683                           424
204538   24     11  2013            1.683                             1
204539   30      9  2015            0.867                            37

             TOWN  LATITUDE  LONGITUDE  AVERAGE_DAILY_WIND_SPEED  \
204535   Sterling  42.55495  -71.75699                       2.9
204536    Holyoke  42.16005  -72.71246                       2.0
204537  Groveland  42.71249  -71.12558                       7.9
204538  Groveland  42.71249  -71.12558                       7.9
204539  Ashburnham  42.55495  -71.75699                      3.5

        PRECIPITATION  …  BLACK_PERCENT  ASIAN_PERCENT  HISPANIC_PERCENT  \
204535            0.0  …       0.008231       0.008738          0.028810
204536            8.1  …       0.026358       0.010139          0.498093
204537            0.0  …       0.004920       0.009386          0.020967
204538            0.0  …       0.004920       0.009386          0.020967
204539           65.5  …       0.007583       0.010165          0.028074

        ALASKAN_NATIVE  PACIFIC_NATIVE  OTHER_RACE  TWO_OR_MORE_RACES  \
204535        0.001013        0.000507    0.003229           0.020515
204536        0.001050        0.000397    0.002944           0.016373
204537        0.000681        0.000303    0.002195           0.018469
204538        0.000681        0.000303    0.002195           0.018469
204539        0.001613        0.000161    0.002259           0.027348

              DIVERSITY_GROUP  CUSTOMER_OUTAGE_HOURS     SAIDI
204535  low middle diversity                399.453  0.051159
204536        high diversity                  1.337  0.000033
204537         low diversity                713.592  0.110480
204538         low diversity                  1.683  0.000261
204539  low middle diversity                 32.079  0.005230

[5 rows x 39 columns]
```

[21]:

```
df.to_csv('Electricity_Reliability_Dataset.csv', index=False)
```

[22]:
```python
# read in the final dataset file to check it
FINAL = pd.read_csv('Electricity_Reliability_Dataset.csv')
FINAL.tail()
```

[22]:
```
        DAY  MONTH  YEAR  OUTAGE_DURATION  NUMBER_OF_CUSTOMERS_AFFECTED  \
204535   20     11  2013            2.833                           141
204536   29     10  2018            1.337                             1
204537   24     11  2013            1.683                           424
204538   24     11  2013            1.683                             1
204539   30      9  2015            0.867                            37

             TOWN  LATITUDE  LONGITUDE  AVERAGE_DAILY_WIND_SPEED  \
204535   Sterling  42.55495  -71.75699                       2.9
204536    Holyoke  42.16005  -72.71246                       2.0
204537  Groveland  42.71249  -71.12558                       7.9
204538  Groveland  42.71249  -71.12558                       7.9
204539  Ashburnham  42.55495  -71.75699                      3.5

        PRECIPITATION  …  BLACK_PERCENT  ASIAN_PERCENT  HISPANIC_PERCENT  \
204535            0.0  …       0.008231       0.008738          0.028810
204536            8.1  …       0.026358       0.010139          0.498093
204537            0.0  …       0.004920       0.009386          0.020967
204538            0.0  …       0.004920       0.009386          0.020967
204539           65.5  …       0.007583       0.010165          0.028074

        ALASKAN_NATIVE  PACIFIC_NATIVE  OTHER_RACE  TWO_OR_MORE_RACES  \
204535        0.001013        0.000507    0.003229           0.020515
204536        0.001050        0.000397    0.002944           0.016373
204537        0.000681        0.000303    0.002195           0.018469
204538        0.000681        0.000303    0.002195           0.018469
204539        0.001613        0.000161    0.002259           0.027348

             DIVERSITY_GROUP  CUSTOMER_OUTAGE_HOURS     SAIDI
204535  low middle diversity                399.453  0.051159
204536        high diversity                  1.337  0.000033
204537         low diversity                713.592  0.110480
204538         low diversity                  1.683  0.000261
204539  low middle diversity                 32.079  0.005230

[5 rows x 39 columns]
```

[ ]:
```python

```