PS4

Partner1: Yue Wang Partner2: Zhuohao Yang

2024-11-02

PS4: Due Sat Nov 2 at 5:00PM Central. Worth 100 points. Submission Steps (10 pts) 1. This problem set is a paired problem set. 2. Play paper, scissors, rock to determine who goes first. Call that person Partner 1. • Partner 1 (name and cnet ID):**Yue Wang, yuew3** • Partner 2 (name and cnet ID):**Zhuohao Yang, zhuohao** 3. Partner 1 will accept the ps4 and then share the link it creates with their partner. You can only share it with one partner so you will not be able to change it after your partner has accepted. 4. "This submission is our work alone and complies with the 30538 integrity policy." Add your initials to indicate your agreement: **This submission is our work alone and complies with the 30538 integrity policy** 5. "I have uploaded the names of anyone else other than my partner and I worked with on the problem set here" (1 point) 6. Late coins used this pset: ***3*** Late coins left after submission: ***1*** 7. Knit your ps4.qmd to an PDF file to make ps4.pdf, • The PDF should not be more than 25 pages. Use head() and re-size figures when appropriate. 8. (Partner 1): push ps4.qmd and ps4.pdf to your github repo. 9. (Partner 1): submit ps4.pdf via Gradescope. Add your partner on Gradescope. 10. (Partner 1): tag your submission in Gradescope

Style Points (10 pts)

Submission Steps (10 pts)

Download and explore the Provider of Services (POS) file (10 pts)

1. PRVDR_CTGRY_SBTYP_CD, PGM_TRMNTN_CD, TRMNTN_EXPRTN_DT, ZIP_CD, FAC_NAME, PRVDR_NUM,PRVDR_NUM_STR, PRVDR_NUM_LEN, PRVDR_CTGRY_CD,
2. a.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


# Read the file firstly
df_2016 = pd.read_csv('/Users/yuewang1/Desktop/python
 ↪  2/hw4/data/POS_File_Hospital_Non_Hospital_Facilities_Q4_2016 3.csv',
 ↪  low_memory = False)
df_2016.head()

# Count the number of short-term hospitals
short_term_hos_2016 = df_2016[(df_2016['PRVDR_CTGRY_SBTYP_CD'] == 1) &
 ↪  (df_2016['PRVDR_CTGRY_CD'] == 1)]
print(len(short_term_hos_2016))
```

```
7245
```

From the result, we could see that the number of short-term hospitals from the dataset are 7245. This does not make sense.

From the outer sources, we could see that, the total number of All U.S. Registered Hospitals are 5534, which is far lower than the result from dataset.
(https://www.aha.org/system/files/2018-02/2018-aha-hospital-fast-facts.pdf)

Morevoer, we cannot find the exact term of 'short-term' hospital, but we did find a similary term, which is community hospital, the data of community hospitals in 2016 is 3385+1882=5267
(https://www.ahadata.com/hospitaltrendwatch/hospitalorganizationaltrends), which is very close to the number for the total number of registered hospitals in U.S. While the definition of community hospital is community hospitals included all non-federal, short-term general and specialty hospitals whose facilities and services are available to the public. Therefore, since the community hospital has included the number of short-term general and specialty hospitals. The number from the dataset is not making any sense since it far more exceeds the number for the other resouces.

b. From my perspective, the differences could be the following reasons:

1. The discrepancy may be due to variations in how short-term is defined across datasets. POS may include more specialized facilities under the short-term designatio taht AHA excludes.
2. The POS dataset might include affiliated or multilocation entries separately, whereas AHA may consolidate these under single institutional entries.
3. The POS dataset might have wrong data entry progress, making the dataset includes duplicate hospitals.

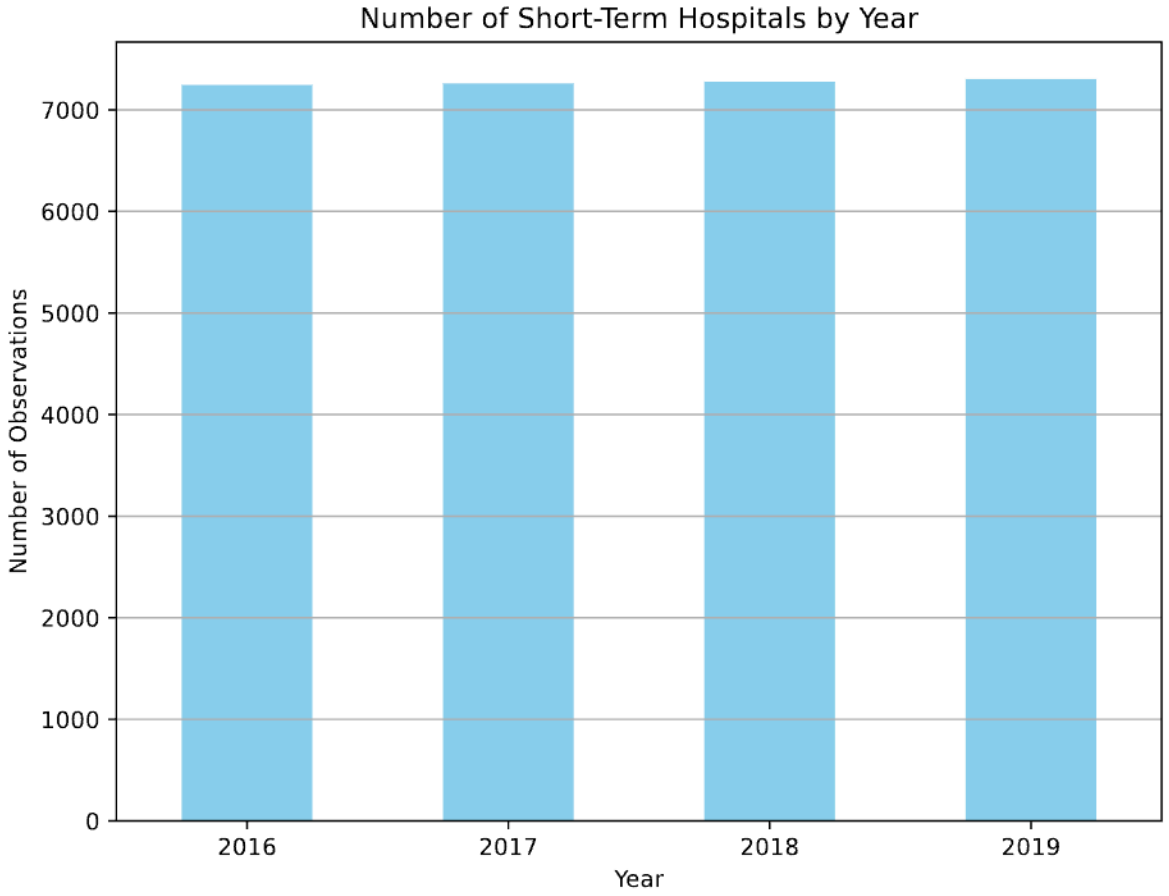
3.

```
# Load the datasets
df_2017 = pd.read_csv('/Users/yuewang1/Desktop/python
 ↪  2/hw4/data/POS_File_Hospital_Non_Hospital_Facilities_Q4_2017 2.csv',
 ↪  low_memory = False, encoding='ISO-8859-1')
df_2018 = pd.read_csv('/Users/yuewang1/Desktop/python
 ↪  2/hw4/data/POS_File_Hospital_Non_Hospital_Facilities_Q4_2018 2.csv',
 ↪  low_memory = False, encoding='ISO-8859-1')
df_2019 = pd.read_csv('/Users/yuewang1/Desktop/python
 ↪  2/hw4/data/POS_File_Hospital_Non_Hospital_Facilities_Q4_2019 2.csv',
 ↪  low_memory = False, encoding='ISO-8859-1')

# Define a function to subset short-term hospitals
def subset_short_term(df):
    return df[
        (df['PRVDR_CTGRY_CD'] == 1) &
        (df['PRVDR_CTGRY_SBTYP_CD'] == 1)
    ]

# Subset each year's data
short_term_2016 = subset_short_term(df_2016)
short_term_2017 = subset_short_term(df_2017)
short_term_2018 = subset_short_term(df_2018)
short_term_2019 = subset_short_term(df_2019)

# Add a 'Year' column to each dataset
short_term_2016['Year'] = 2016
short_term_2017['Year'] = 2017
short_term_2018['Year'] = 2018
short_term_2019['Year'] = 2019

# Combine all years into one DataFrame
all_years = pd.concat([
```

```
    short_term_2016,
    short_term_2017,
    short_term_2018,
    short_term_2019
], ignore_index=True)

# Count the number of observations by year
observations_by_year = all_years['Year'].value_counts().sort_index()

# Plot the number of observations by year
plt.figure(figsize=(8, 6))
observations_by_year.plot(kind='bar', color='skyblue')
plt.title('Number of Short-Term Hospitals by Year')
plt.xlabel('Year')
plt.ylabel('Number of Observations')
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.show()
```

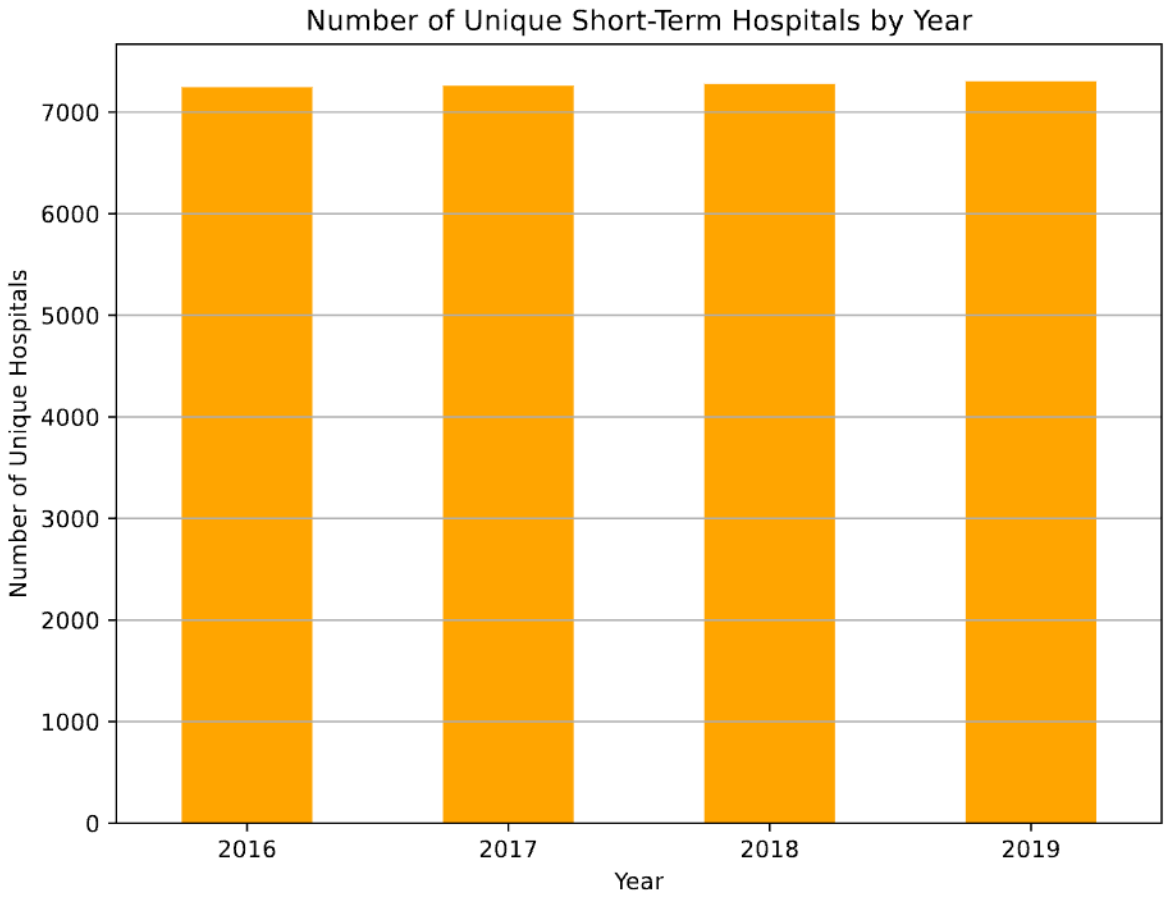
```
/var/folders/27/5vfjdszd7w16ks8_5dyn2nvr0000gn/T/ipykernel_21045/1932147756.py:20:
SettingWithCopyWarning:


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

/var/folders/27/5vfjdszd7w16ks8_5dyn2nvr0000gn/T/ipykernel_21045/1932147756.py:21:
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

/var/folders/27/5vfjdszd7w16ks8_5dyn2nvr0000gn/T/ipykernel_21045/1932147756.py: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

/var/folders/27/5vfjdszd7w16ks8_5dyn2nvr0000gn/T/ipykernel_21045/1932147756.py:23:
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
```



4. a.

```
# Define the function to filter unique hospitals with valid CMS Certification
 ↪ Numbers
def filter_valid_cms_numbers(df):
    df = df.copy()
    df['PRVDR_NUM_STR'] = df['PRVDR_NUM'].astype(str)
    df['PRVDR_NUM_LEN'] = df['PRVDR_NUM_STR'].str.len()
    valid_lengths = [6, 10]  # since CMS numbers can be 6 or 10 positions
    df_valid = df[df['PRVDR_NUM_LEN'].isin(valid_lengths)]
    return df_valid

all_years_valid = filter_valid_cms_numbers(all_years)

# Get the number of unique hospitals per year
unique_hospitals_per_year =
 ↪ all_years_valid.groupby('Year')['PRVDR_NUM'].nunique()

# Plot the number of unique hospitals per year
plt.figure(figsize=(8, 6))
unique_hospitals_per_year.plot(kind='bar', color='orange')
plt.title('Number of Unique Short-Term Hospitals by Year')
plt.xlabel('Year')
plt.ylabel('Number of Unique Hospitals')
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.show()
```



b. According to the output, we can see that these two plots have no different, for years of 2016, 2017, 2018 and 2019, they all have the same quantities of unique hospitals with the quantities of hospital we have calculated in the last question.

Identify hospital closures in POS file (15 pts) (*)

1.

```
# Define a function to subset short_term hospitals
def subset_short_term(df, year):
    df = df[
        (df['PRVDR_CTGRY_CD'] == 1) &
        (df['PRVDR_CTGRY_SBTYP_CD'] == 1)
    ].copy()
```

```
    df['Year'] = year
    return df

short_term_2016 = subset_short_term(df_2016, 2016)
short_term_2017 = subset_short_term(df_2017, 2017)
short_term_2018 = subset_short_term(df_2018, 2018)
short_term_2019 = subset_short_term(df_2019, 2019)

# combine them
all_years = pd.concat([
    short_term_2016,
    short_term_2017,
    short_term_2018,
    short_term_2019
], ignore_index=True)

# Ensure that col is string
all_years['PRVDR_NUM'] = all_years['PRVDR_NUM'].astype(str)

# Filter active hospitals in 2016
active_2016 = short_term_2016[
    (short_term_2016['PGM_TRMNTN_CD'] == 0)
].copy()

# Ensure 'PRVDR_NUM' is a string and has valid length
active_2016['PRVDR_NUM'] = active_2016['PRVDR_NUM'].astype(str)
active_2016 = active_2016[
    active_2016['PRVDR_NUM'].str.len().isin([6, 10])
]

# Get the list of unique provider numbers for active hospitals in 2016
active_hospitals_2016 = active_2016['PRVDR_NUM'].unique()

# Create a DataFrame to store closure information
closures = []

# Loop over each hospital
for prvd_num in active_hospitals_2016:
    # Initialize closure year as None
    closure_year = None
    # Initialize a flag to check if the hospital remains active
    hospital_active = True
```

```
    # Check each subsequent year
    for year, df_year in zip([2017, 2018, 2019], [short_term_2017,
     ↪  short_term_2018, short_term_2019]):
        # Filter the year's data for this provider
        df_provider = df_year[df_year['PRVDR_NUM'].astype(str) == prvd_num]

        if df_provider.empty:
            # Hospital does not appear in this year
            closure_year = year
            hospital_active = False
            break
        else:
            # Check if the hospital is still active
            if df_provider.iloc[0]['PGM_TRMNTN_CD'] != 0:
                closure_year = year
                hospital_active = False
                break

    if not hospital_active:
        # Get facility name and ZIP code from the 2016 data
        facility_name = active_2016[active_2016['PRVDR_NUM'] ==
↪   prvd_num]['FAC_NAME'].values[0]
        zip_code = active_2016[active_2016['PRVDR_NUM'] ==
↪   prvd_num]['ZIP_CD'].values[0]

        # Append to closures list
        closures.append({
            'PRVDR_NUM': prvd_num,
            'Facility_Name': facility_name,
            'ZIP_Code': zip_code,
            'Closure_Year': closure_year
        })

# Convert closures list to DataFrame
closures_df = pd.DataFrame(closures)

# Number of hospitals suspected to have closed
num_closures = closures_df.shape[0]
print(f"Number of hospitals suspected to have closed: {num_closures}")
```

```
Number of hospitals suspected to have closed: 174
```

2.

```
# Step 1: Sort the closures_df by Facility_Name
closures_sorted = closures_df.sort_values(by='Facility_Name')

# Step 2: Get the first 10 rows
first_10_closures = closures_sorted.head(10)

# Step 3: Extract Facility_Name and Closure_Year
first_10_closures_info = first_10_closures[['Facility_Name', 'Closure_Year']]

# Step 4: Display the results
print("First 10 Hospitals Suspected to Have Closed:")
print(first_10_closures_info.to_string(index=False))
```

```
First 10 Hospitals Suspected to Have Closed:
                              Facility_Name  Closure_Year
                      ABRAZO MARYVALE CAMPUS          2017
    ADVENTIST MEDICAL CENTER - CENTRAL VALLEY          2017
                     AFFINITY MEDICAL CENTER          2018
ALBANY MEDICAL CENTER / SOUTH CLINICAL CAMPUS          2017
     ALLEGIANCE SPECIALTY HOSPITAL OF KILGORE          2017
                    ALLIANCE LAIRD HOSPITAL          2019
                  ALLIANCEHEALTH DEACONESS           2019
               ANNE BATES LEACH EYE HOSPITAL          2019
      ARKANSAS VALLEY REGIONAL MEDICAL CENTER          2017
          BANNER CHURCHILL COMMUNITY HOSPITAL          2017
```

3. a.

```
# Ensure 'PRVDR_NUM' is a string
all_years['PRVDR_NUM'] = all_years['PRVDR_NUM'].astype(str)

# Filter active hospitals (PGM_TRMNTN_CD == 0)
active_hospitals = all_years[all_years['PGM_TRMNTN_CD'] == 0].copy()

# Ensure 'ZIP_Code' is available
active_hospitals['ZIP_Code'] = active_hospitals['ZIP_CD']

# Get counts of active hospitals per ZIP code per year
active_counts = (
    active_hospitals.groupby(['Year', 'ZIP_Code'])['PRVDR_NUM']
```

```
    .nunique()
    .reset_index(name='Active_Hospitals')
)

# Filter closures where we can check the next year's data (Closure_Year less
 ↪  than 2019)
closures_df_to_check = closures_df[closures_df['Closure_Year'] < 2019].copy()

# Create a column for the year after the closure
closures_df_to_check['Closure_Year_Plus1'] =
 ↪  closures_df_to_check['Closure_Year'] + 1

# Merge to get active hospitals in the closure year
closures_df_with_counts = closures_df_to_check.merge(
    active_counts,
    left_on=['Closure_Year', 'ZIP_Code'],
    right_on=['Year', 'ZIP_Code'],
    how='left'
).rename(columns={'Active_Hospitals': 'Active_Hospitals_N'})

closures_df_with_counts.drop(columns=['Year'], inplace=True)

# Merge to get active hospitals
closures_df_with_counts = closures_df_with_counts.merge(
    active_counts,
    left_on=['Closure_Year_Plus1', 'ZIP_Code'],
    right_on=['Year', 'ZIP_Code'],
    how='left'
).rename(columns={'Active_Hospitals': 'Active_Hospitals_N_plus1'})

closures_df_with_counts.drop(columns=['Year'], inplace=True)

# Fill NaN values
closures_df_with_counts['Active_Hospitals_N'] =
 ↪  closures_df_with_counts['Active_Hospitals_N'].fillna(0)
closures_df_with_counts['Active_Hospitals_N_plus1'] =
 ↪  closures_df_with_counts['Active_Hospitals_N_plus1'].fillna(0)

# Identify potential mergers/acquisitions
closures_df_with_counts['Potential_Merger'] = (
    closures_df_with_counts['Active_Hospitals_N_plus1'] >=
 ↪  closures_df_with_counts['Active_Hospitals_N']
```

```
)

# Identify potential mergers/acquisitions
closures_df_with_counts['Potential_Merger'] = (
    closures_df_with_counts['Active_Hospitals_N_plus1'] >=
 ↪  closures_df_with_counts['Active_Hospitals_N']
)

# Hospitals that fit the definition of potential mergers/acquisitions
potential_mergers_df =
 ↪  closures_df_with_counts[closures_df_with_counts['Potential_Merger']]

num_potential_mergers = potential_mergers_df.shape[0]
print(f"Number of hospitals potentially due to merger/acquisition:
 ↪  {num_potential_mergers}")
```

```
Number of hospitals potentially due to merger/acquisition: 97
```

b.

```
# Exclude potential mergers from closures_df
corrected_closures_df =
 ↪  closures_df[~closures_df['PRVDR_NUM'].isin(potential_mergers_df['PRVDR_NUM'])].copy()

# Number of hospitals after correction
num_corrected_closures = corrected_closures_df.shape[0]
print(f"Number of hospitals after correcting for potential mergers:
 ↪  {num_corrected_closures}")

# Save the cleaned data to a CSV file
output_path =
 ↪  "/Users/yuewang1/Desktop/corrected_hospital_closures_dataset.csv"
corrected_closures_df.to_csv(output_path, index=False)
print(f"Cleaned data has been saved to {output_path}")
```

```
Number of hospitals after correcting for potential mergers: 77
Cleaned data has been saved to
/Users/yuewang1/Desktop/corrected_hospital_closures_dataset.csv
```

c.

```
corrected_closures_sorted =
 ↪  corrected_closures_df.sort_values(by='Facility_Name')
result = corrected_closures_sorted.head(10)
result_selected = result[['Facility_Name', 'Closure_Year']]
# Display the first 10 facilities
print("First 10 Corrected Hospital Closures:")
print(result_selected.to_string(index=False))
```

```
First 10 Corrected Hospital Closures:
                                    Facility_Name  Closure_Year
                          ALLIANCE LAIRD HOSPITAL          2019
                         ALLIANCEHEALTH DEACONESS          2019
                    ANNE BATES LEACH EYE HOSPITAL          2019
                    BARIX CLINICS OF PENNSYLVANIA          2019
                  BAYLOR EMERGENCY MEDICAL CENTER          2019
BAYLOR SCOTT & WHITE EMERGENCY MEDICAL CENTER AT C          2019
                       BELMONT COMMUNITY HOSPITAL          2019
                          BIG SKY MEDICAL CENTER          2019
              BLACK RIVER COMMUNITY MEDICAL CENTER          2019
                     CARE REGIONAL MEDICAL CENTER          2019
```

Download Census zip code shapefile (10 pt)

1. a. ## File types and the file information ### .shp (Shape File): Contains the geometry of the features, such as points, lines, and polygons. It is the core spatial data file that defines the location, shape, and boundaries of geographical features. size: 837,544,580 bytes (837.5 MB on disk) ### .shx (Shape Index File): An index for the .shp file, which helps in efficiently locating records. It stores the offsets of the geometries in the .shp file. size: 265,060 bytes (266 KB on disk) ### .dbf (Database File): A dBASE file that contains attribute data associated with each shape. This file includes tabular information, such as names, population counts, or other relevant attributes of the geometries. size: 6,425,474 bytes (6.4 MB on disk) ### .prj (Projection File): Describes the coordinate system and projection information used for the shapefile. It ensures that spatial data from different sources can align properly. size: 165 bytes (4 KB on disk) ### .xml (Metadata File): Contains metadata, such as descriptive information about the data content, creation process, and purpose of the dataset. size: 15,639 bytes (16 KB on disk)

   b. It will be useful going forward to having .shp and .dbf files larger. The .shp file contains the geometries, which can be bulky, while the .dbf stores attribute data. However, .shx, .prj, and .xml files are expected to be relatively smaller since they

only store index, projection, and metadata information, respectively. After unzipping, the size of those files shows in a.

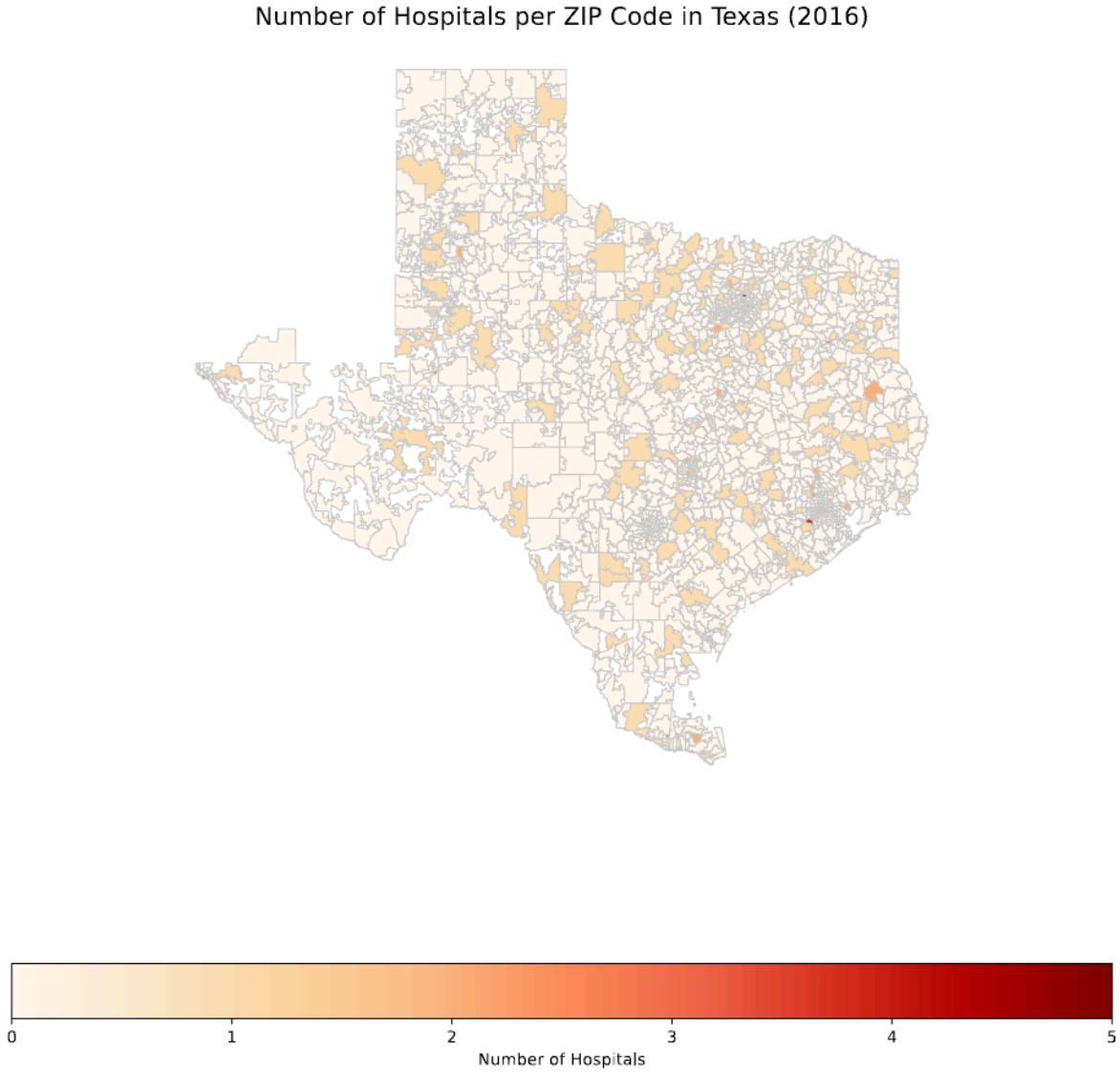
2.

```
import matplotlib.pyplot as plt
import geopandas as gpd

# Load the shapefile
shapefile_path = '/Users/yuewang1/Desktop/python
↪  2/hw4/gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.shp'
gdf = gpd.read_file(shapefile_path)

print(gdf.columns)
```

```
Index(['GEO_ID', 'ZCTA5', 'NAME', 'LSAD', 'CENSUSAREA', 'geometry'],
dtype='object')
```

```
# Assuming 'ZCTA5' is the correct column for ZIP codes in your shapefile
gdf['ZCTA5'] = gdf['ZCTA5'].astype(str).str.zfill(5)

# Filter for active short-term hospitals
active_2016 = df_2016[
    (df_2016['PGM_TRMNTN_CD'] == 0) &
    (df_2016['PRVDR_CTGRY_SBTYP_CD'] == 1) &
    (df_2016['PRVDR_CTGRY_CD'] == 1)
].copy()

# Ensure ZIP codes are strings and have leading zeros
active_2016['ZIP_CD'] = active_2016['ZIP_CD'].astype(str).str[:5]
active_2016['ZIP_CD'] = active_2016['ZIP_CD'].str.zfill(5)

# Filter to Texas ZIP codes (prefixes '75', '76', '77', '78', '79')
texas_zip_prefixes = ('75', '76', '77', '78', '79')
gdf_zipcodes = gdf[gdf['ZCTA5'].str.startswith(texas_zip_prefixes)].copy()
pos_hospitals =
↪  active_2016[active_2016['ZIP_CD'].str.startswith(texas_zip_prefixes)].copy()

# Count number of unique hospitals per ZIP code
hospital_counts =
↪  pos_hospitals.groupby('ZIP_CD')['PRVDR_NUM'].nunique().reset_index()
hospital_counts.columns = ['ZIP_CD', 'hospital_count']
```

```
# Merge hospital counts with the ZIP code shapefile
gdf_zipcodes = gdf_zipcodes.merge(hospital_counts, left_on='ZCTA5',
 ↪  right_on='ZIP_CD', how='left')

# Replace NaN values with zero (no hospitals)
gdf_zipcodes['hospital_count'] = gdf_zipcodes['hospital_count'].fillna(0)

# Plot choropleth
fig, ax = plt.subplots(figsize=(12, 12))
gdf_zipcodes.plot(
    column='hospital_count',
    cmap='OrRd',
    linewidth=0.8,
    ax=ax,
    edgecolor='0.8',
    legend=True,
    legend_kwds={'label': "Number of Hospitals", 'orientation': "horizontal"}
)
ax.set_title('Number of Hospitals per ZIP Code in Texas (2016)', fontsize=15)
ax.set_axis_off()
plt.show()
```



Calculate zip code's distance to the nearest hospital (20 pts) (*)

1.

```
# Create the GeoDataFrame
zip_all_centroids = gdf.copy()
zip_all_centroids['geometry'] = gdf['geometry'].centroid
```

```
# Check the dimentsions
print("Dimensions of zip_all_centroids: ", zip_all_centroids.shape)
```

```
/var/folders/27/5vfjdszd7w16ks8_5dyn2nvr0000gn/T/ipykernel_21045/3156651450.py:3:
UserWarning:

Geometry is in a geographic CRS. Results from 'centroid' are likely
incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected
CRS before this operation.


Dimensions of zip_all_centroids:  (33120, 6)
```

```
# Show the cols and explain the meanings
print("Columns in the GeoDataFrame:", zip_all_centroids.columns)
```

```
Columns in the GeoDataFrame: Index(['GEO_ID', 'ZCTA5', 'NAME', 'LSAD',
'CENSUSAREA', 'geometry'], dtype='object')
```

'GEO_ID': A unique identifier for each geographic entity in the dataset. It identifies each ZIP Code Tabulation Area according to census data standards. This ID helps in referencing specific areas and aligning them with other geographic or census datasets.

'ZCTA5': As mentioned above, this is zip code tabulation area standing for the approximations of U.S Postal Service ZIP codes used by the Census Bureau for geographic and demographic analysis

'NAME': The name of the area. Providing clarity on the location within specific regions or neighborhoods in some cases. It has the same functionality as ZCTA5

'LSAD': This stands for "Legal/Statistical Area Description." It provides a label indicating the type of geographic area (for example, whether it's a city, town, or other type of statistical area). For ZCTAs, this is usually set to standard codes to indicate a general area type, such as "ZCTA.

'CENSUSAREA': Census area represents the area of the ZCTA as measured by the Census Bureau. This term helps in understanding the physical size of each ZIP code area, which can vary significant across different regions.

'geometry': This column contains the geometric data for each ZCTA. In the zips_all_centroids GeoDataFrame, the geometry has been converted from polygons (the boundaries of each ZIP code area) to centroids (points representing the center of each ZIP code area). This column allows for spatial analysis, such as mapping and calculating distances.

2.

```
# Define the prefixes first and filter the texas zip code centroids
texas_prefixes = ('75', '76', '77', '78', '79')
zips_texas_centroids =
 ↪  zip_all_centroids[zip_all_centroids['ZCTA5'].str.startswith(texas_prefixes)]

# Filter for Texas and other boarder state
# Oklahoma to the north, Arkansas and Louisiana to the east, and New Mexico
 ↪  to the west
boarder_state_prefixes = texas_prefixes + ('70', '71', '72', '73', '74',
 ↪  '87', '88')
zips_texas_borderstates_centroids =
 ↪  zip_all_centroids[zip_all_centroids['ZCTA5'].str.startswith(boarder_state_prefixes)]

# Define a function
def polygons_intersect(p1, p2):
  return p1.intersect(p2)

# Combine them into one
texas_polygon = zips_texas_centroids.unary_union

# Count and text the unique zip codes in each subset
unique_texas_zip_cd = zips_texas_centroids['ZCTA5'].nunique()
unique_boarderstates_zip_cd =
 ↪  zips_texas_borderstates_centroids['ZCTA5'].nunique()

print("Number of unique zip codes in Texas:", unique_texas_zip_cd)
print("Number of unique zip codes in Boarder States:",
 ↪  unique_boarderstates_zip_cd)
```

```
Number of unique zip codes in Texas: 1935
Number of unique zip codes in Boarder States: 4057
```

```
/var/folders/27/5vfjdszd7w16ks8_5dyn2nvr0000gn/T/ipykernel_21045/3488886593.py:15:
DeprecationWarning:

The 'unary_union' attribute is deprecated, use the 'union_all()' method
instead.
```

3.

```
# Since we have defined active_2016
# Merge the active_2016 (which is containing the active hospitals in 2016)
 ↪  with boarder states centroids
```

```
pos_hospital_boarderstates =
 ↪  active_2016[active_2016['ZIP_CD'].str.startswith(boarder_state_prefixes)]
hospital_counts =
 ↪  pos_hospital_boarderstates['ZIP_CD'].value_counts().reset_index()
hospital_counts.columns = ['ZIP_CD', 'hospital_count']

# Adjust the format for ZIP_CD and ZCTA5
hospital_counts['ZIP_CD'] =
 ↪  hospital_counts['ZIP_CD'].astype(str).str.replace('.0', '', regex=False)
zips_texas_borderstates_centroids['ZCTA5'] =
 ↪  zips_texas_borderstates_centroids['ZCTA5'].astype(str).str.replace('.0',
 ↪  '', regex=False)


zips_withhospital_centroids =
 ↪  zips_texas_borderstates_centroids.merge(hospital_counts, left_on =
 ↪  'ZCTA5', right_on = 'ZIP_CD', how = 'left').fillna(0)

# Filter to keep only the ZIP codes with at least 1 hospital
zips_withhospital_centroids =
 ↪  zips_withhospital_centroids[zips_withhospital_centroids['hospital_count']
 ↪  >= 1]

print(zips_withhospital_centroids.head())
print(len(zips_withhospital_centroids))
```

```
            GEO_ID  ZCTA5   NAME   LSAD  CENSUSAREA  \
8   8600000US70043  70043  70043  ZCTA5       7.775
27  8600000US70127  70127  70127  ZCTA5       7.095
31  8600000US70301  70301  70301  ZCTA5     288.050
38  8600000US70360  70360  70360  ZCTA5      64.325
45  8600000US70403  70403  70403  ZCTA5      42.960

                     geometry ZIP_CD  hospital_count
8   POINT (-89.96276 29.94804)  70043            1.0
27  POINT (-89.97675 30.02501)  70127            1.0
31   POINT (-90.74089 29.8141)  70301            1.0
38  POINT (-90.81028 29.58819)  70360            2.0
45  POINT (-90.48388 30.48002)  70403            2.0
448
```

```
/Users/yuewang1/opt/anaconda3/lib/python3.9/site-packages/geopandas/geodataframe.py:1819:
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
```

I have decided to merge the 'zips_texas_borderstates_centroids' into 'hospital_counts', which is a data frame that store the boarder states of short-term hospitals that are active in 2016. The 'hospital_counts' contains a lot other information we might need to use in the future analysis. I decided to merge on the zip codes cols, which are 'ZCTA5' for the 'zips_texas_borderstates_centroids' and 'ZIP_CD' for 'hospital_counts' respectively, since these two cols have two different formats, therefore I have compiled them into the same format before merge them.

4. a.

```
from shapely.ops import nearest_points
import time
import random

# Subset the 10 zip codes in zip_texas...
zips_10_subsets = zips_texas_centroids.sample(n = 10, random_state = 10)

# Def function
def calculate_nearest_distance(row, another_gdf):
  nearest_geom = nearest_points(row.geometry, another_gdf.union_all())[1]
  return row.geometry.distance(nearest_geom)

# Set up the timer
start_time = time.time()

# Calculate the distance for those 10 subsets
zips_10_subsets['nearest_disance'] =
 ↪  zips_10_subsets.apply(calculate_nearest_distance, another_gdf =
 ↪  zips_withhospital_centroids, axis = 1)

end_time = time.time()
time_calculations = end_time - start_time
print(f"Time taken to calculate 10 ZIP codes is: {time_calculations}
 ↪  seconds")
```

```
# Time to calculate the entire dataset
time_calculations_entire = time_calculations * (len(zips_texas_centroids) /
 ↪  10)
print(f"Time taken to calculate entire dataset ZIP_CD is:
 ↪  {time_calculations_entire} seconds")
```

```
Time taken to calculate 10 ZIP codes is: 0.005197048187255859 seconds
Time taken to calculate entire dataset ZIP_CD is: 1.0056288242340088 seconds
```

!!!

```
# Ensure the CRS matches
zips_texas_centroids =
 ↪  zips_texas_centroids.to_crs(zips_withhospital_centroids.crs)

# Select the first 10 ZIP codes
subset_zips = zips_texas_centroids.head(10).copy()

# Start timing
start_time = time.time()

# Function to compute the nearest distance
def compute_nearest(row, destination_gdf, val_col, geom_col='geometry'):
    # Calculate the distance between the point and all destination points
    distances = destination_gdf[geom_col].distance(row[geom_col])
    # Find the minimum distance and corresponding value
    min_idx = distances.idxmin()
    min_distance = distances[min_idx]
    return min_distance

# Apply the function to compute the nearest distance
subset_zips['nearest_distance'] = subset_zips.apply(
    compute_nearest, axis=1, destination_gdf=zips_withhospital_centroids,
 ↪  val_col='ZCTA5'
)

# End timing
end_time = time.time()
elapsed_time = end_time - start_time

print(f"Time taken to compute nearest distances for 10 ZIP codes:
 ↪  {elapsed_time:.2f} seconds")
```

```python
total_zips = len(zips_texas_centroids)
print(f"Total number of ZIP codes in Texas: {total_zips}")

time_per_zip = elapsed_time / 10  # Average time per ZIP code
estimated_total_time = time_per_zip * total_zips
print(f"Estimated time to compute for all ZIP codes: {estimated_total_time /
↪ 60:.2f} minutes")
```

```
Time taken to compute nearest distances for 10 ZIP codes: 0.07 seconds
Total number of ZIP codes in Texas: 1935
Estimated time to compute for all ZIP codes: 0.22 minutes
```

```
/var/folders/27/5vfjdszd7w16ks8_5dyn2nvr0000gn/T/ipykernel_21045/3151206505.py:13:
UserWarning:

Geometry is in a geographic CRS. Results from 'distance' are likely
incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected
CRS before this operation.


/var/folders/27/5vfjdszd7w16ks8_5dyn2nvr0000gn/T/ipykernel_21045/3151206505.py:13:
UserWarning:

Geometry is in a geographic CRS. Results from 'distance' are likely
incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected
CRS before this operation.


/var/folders/27/5vfjdszd7w16ks8_5dyn2nvr0000gn/T/ipykernel_21045/3151206505.py:13:
UserWarning:

Geometry is in a geographic CRS. Results from 'distance' are likely
incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected
CRS before this operation.


/var/folders/27/5vfjdszd7w16ks8_5dyn2nvr0000gn/T/ipykernel_21045/3151206505.py:13:
UserWarning:
```

```
Geometry is in a geographic CRS. Results from 'distance' are likely
incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected
CRS before this operation.


/var/folders/27/5vfjdszd7w16ks8_5dyn2nvr0000gn/T/ipykernel_21045/3151206505.py:13:
UserWarning:

Geometry is in a geographic CRS. Results from 'distance' are likely
incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected
CRS before this operation.


/var/folders/27/5vfjdszd7w16ks8_5dyn2nvr0000gn/T/ipykernel_21045/3151206505.py:13:
UserWarning:

Geometry is in a geographic CRS. Results from 'distance' are likely
incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected
CRS before this operation.


/var/folders/27/5vfjdszd7w16ks8_5dyn2nvr0000gn/T/ipykernel_21045/3151206505.py:13:
UserWarning:

Geometry is in a geographic CRS. Results from 'distance' are likely
incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected
CRS before this operation.


/var/folders/27/5vfjdszd7w16ks8_5dyn2nvr0000gn/T/ipykernel_21045/3151206505.py:13:
UserWarning:

Geometry is in a geographic CRS. Results from 'distance' are likely
incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected
CRS before this operation.


/var/folders/27/5vfjdszd7w16ks8_5dyn2nvr0000gn/T/ipykernel_21045/3151206505.py:13:
UserWarning:

Geometry is in a geographic CRS. Results from 'distance' are likely
incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected
CRS before this operation.
```

```
/var/folders/27/5vfjdszd7w16ks8_5dyn2nvr0000gn/T/ipykernel_21045/3151206505.py:13:
UserWarning:

Geometry is in a geographic CRS. Results from 'distance' are likely
incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected
CRS before this operation.
```

b.

```
# Set up the timer
start_time_b = time.time()

# Calculate the distance for the entire dataset
zips_texas_centroids['nearest_distance'] =
 ↪  zips_texas_centroids.apply(calculate_nearest_distance, another_gdf =
 ↪  zips_withhospital_centroids, axis = 1)

# Time
end_time_b = time.time()
time_calculations_b = end_time_b - start_time_b

# Display
print(f"Time taken to calculate all distance for ZIP codes in texas centroids
 ↪  is: {time_calculations_b} seconds")
```

```
Time taken to calculate all distance for ZIP codes in texas centroids is:
0.7362430095672607 seconds
```

c.

The unit in the prj. file is 'Degree', therefore, we can conver the degree into mile by converting the distance into UTM Zone 14N for texas (meters) first, and then calculate the distance in meters and convert to miles.

```
# Define the projection
projected_crs = "EPSG:5070"

# Project both GeoDataFrames
zips_texas_centroids_projected = zips_texas_centroids.to_crs(projected_crs)
zips_withhospital_centroids_projected =
 ↪  zips_withhospital_centroids.to_crs(projected_crs)
```

```
# Start timing
start_time_projected = time.time()

# Compute nearest distances using projected data
zips_texas_centroids_projected['nearest_distance_meters'] =
 ↪  zips_texas_centroids_projected.apply(
    compute_nearest, axis=1,
 ↪  destination_gdf=zips_withhospital_centroids_projected, val_col='ZCTA5'
)

# End timing
end_time_projected = time.time()
elapsed_time_projected = end_time_projected - start_time_projected

print(f"Time taken with projected data: {elapsed_time_projected / 60:.2f}
 ↪  minutes")

# Convert distances from meters to miles
zips_texas_centroids_projected['nearest_distance_miles'] =
 ↪  zips_texas_centroids_projected['nearest_distance_meters'] * 0.000621371

# Display the first few records
print(zips_texas_centroids_projected[['ZCTA5',
 ↪  'nearest_distance_miles']].head())
```

```
Time taken with projected data: 0.01 minutes
      ZCTA5  nearest_distance_miles
9207  78624                0.000000
9208  78626               11.449751
9209  78628                7.596761
9210  78631               22.705377
9211  78632               15.136325
```

5. a. The unit in this dataset I put is meters since i have converted into the format of "EPSG:5070", however, for the average distance for the nearest hospital, i have converted into miles.

   b.

```
# Since in sub question 4, we have calculate the nearest_distance in mile
# Calculate the average distances in miles
```

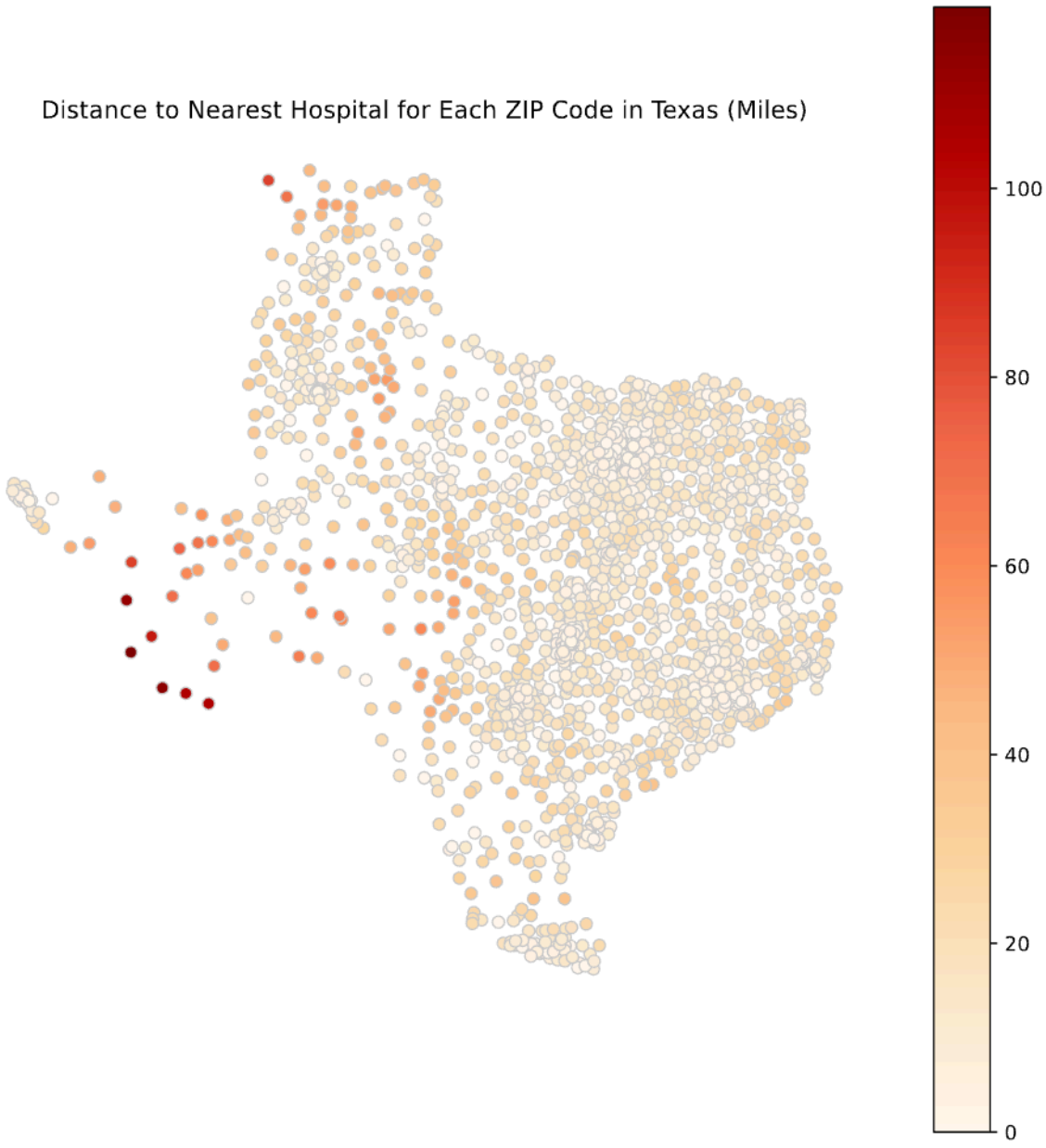
```
avg_distance_miles =
 ↪  zips_texas_centroids_projected['nearest_distance_miles'].mean()
print(f"Average distance to the nearest hospital for Texas ZIP codes:
 ↪  {avg_distance_miles:.6f} miles")
```

```
Average distance to the nearest hospital for Texas ZIP codes: 13.424206 miles
```

Obviously, the average distance to the nearest hospital for Texas is reasonable, almost 13 miles for each nearest hospital (13.424206). This is a fair distance for the two neareast hospital.

c.

```
# Plot plot plot
fig, ax = plt.subplots(figsize=(10, 10))
zips_texas_centroids_projected.plot(
    column='nearest_distance_miles',
    cmap='OrRd',
    linewidth=0.8,
    ax=ax,
    edgecolor='0.8',
    legend=True
)
ax.set_title('Distance to Nearest Hospital for Each ZIP Code in Texas
 ↪  (Miles)')
plt.axis('off')
plt.show()
```



Effects of closures on access in Texas (15 pts)

1.

```
# Load the dataset
file_path = "/Users/yuewang1/Desktop/corrected_hospital_closures_dataset.csv"
```

```
df = pd.read_csv(file_path)

# Filter for Texas hospitals and records from 2016-2019
df_filtered = df[df['Closure_Year'].between(2016, 2019)]
texas_prefixes = ('75', '76', '77', '78', '79')
df_filtered['ZIP_Code'] = df_filtered['ZIP_Code'].astype(str)
df_texas =
 ↪  df_filtered[df_filtered['ZIP_Code'].str.startswith(texas_prefixes)]

# show the number of colusure
closures_by_zip =
 ↪  df_texas.groupby('ZIP_Code').size().reset_index(name='Number_of_Closures')
```

```
print(closures_by_zip)
```

```
   ZIP_Code  Number_of_Closures
0   75051.0                   1
1   75087.0                   1
2   75140.0                   1
3   75235.0                   1
4   75390.0                   1
5   76520.0                   1
6   76531.0                   1
7   76645.0                   1
8   77065.0                   1
9   78336.0                   1
10  78613.0                   1
11  79520.0                   1
12  79529.0                   1
13  79902.0                   1
```

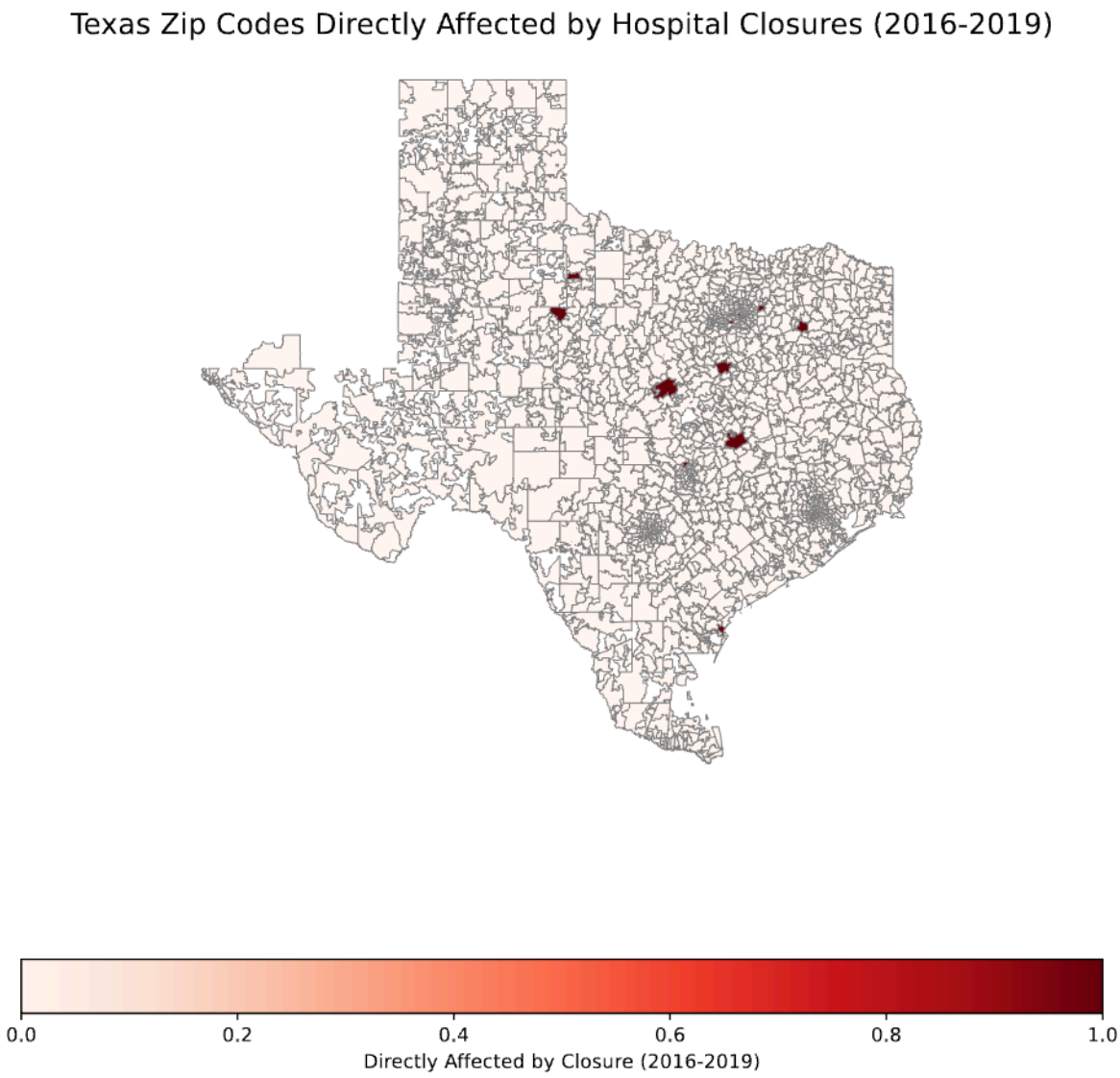
2.

```
# Standardize ZIP code format in df_texas (remove any decimals)
df_texas.loc[:, 'ZIP_Code'] = df_texas['ZIP_Code'].str.replace(r'\.0$', '',
 ↪  regex=True).str.strip()

# Count the number of unique directly affected ZIP codes in Texas
unique_texas_zips = df_texas['ZIP_Code'].unique()
num_directly_affected_zips = len(unique_texas_zips)
print(f"Number of directly affected Texas zip codes (2016-2019):
 ↪  {num_directly_affected_zips}")
```

```
# Prepare a DataFrame with directly affected ZIP codes
affected_zips_df = pd.DataFrame({
    'ZIP_Code': unique_texas_zips,
    'Affected': 1  # Indicator for affected zip codes
})

# Rename ZIP code column in gdf_zipcodes to match and standardize formatting
gdf_zipcodes = gdf_zipcodes.rename(columns={'ZCTA5': 'ZIP_Code'})
gdf_zipcodes['ZIP_Code'] =
 ↪  gdf_zipcodes['ZIP_Code'].astype(str).str.replace(r'\.0$', '',
 ↪  regex=True).str.strip()

# Merge directly affected ZIP codes with the shapefile GeoDataFrame
gdf_zipcodes = gdf_zipcodes.merge(affected_zips_df, on='ZIP_Code',
 ↪  how='left')

# Fill NaN values in 'Affected' column with 0 for unaffected ZIP codes
gdf_zipcodes['Affected'] = gdf_zipcodes['Affected'].fillna(0).astype(int)

# Display affected_zips_df for verification
print(affected_zips_df)
print(f"The number of directly affected ZIP codes in Texas:
 ↪  {len(affected_zips_df)}")

# Plot the data
fig, ax = plt.subplots(1, 1, figsize=(10, 10))
gdf_zipcodes.plot(column='Affected',
                  cmap='Reds',
                  linewidth=0.1,
                  ax=ax,
                  edgecolor='grey',
                  legend=True,
                  legend_kwds={'label': "Directly Affected by Closure
 ↪  (2016-2019)",
                               'orientation': "horizontal"})
ax.set_title('Texas Zip Codes Directly Affected by Hospital Closures
 ↪  (2016-2019)', fontsize=15)
ax.axis('off')
plt.show()
```

```
Number of directly affected Texas zip codes (2016-2019): 14
```

```
   ZIP_Code  Affected
0     76645         1
1     79520         1
2     78336         1
3     77065         1
4     79529         1
5     76531         1
6     75390         1
7     79902         1
8     75235         1
9     75051         1
10    78613         1
11    76520         1
12    75087         1
13    75140         1
The number of directly affected ZIP codes in Texas: 14
```



3.

```
# Define a projected CRS (NAD83 / Texas Centric)
projected_crs = "EPSG:3083"
# Reproject the entire ZIP codes GeoDataFrame to the projected CRS
gdf_zipcodes_projected = gdf_zipcodes.to_crs(projected_crs)

# Filter for directly affected zip codes in the projected GeoDataFrame
gdf_directly_affected =
 ↪  gdf_zipcodes_projected[gdf_zipcodes_projected['Affected'] == 1].copy()
```

```
# Compute centroids after reprojection to ensure accuracy
gdf_directly_affected['centroid'] = gdf_directly_affected.geometry.centroid

# Create a GeoDataFrame with centroids
gdf_directly_affected_centroids = gpd.GeoDataFrame(
    gdf_directly_affected,
    geometry='centroid',
    crs=projected_crs
)
```

```
# Define buffer distance in meters (10 miles)
miles = 10
meters_per_mile = 1609.34
buffer_distance = miles * meters_per_mile  # 16093.4 meters

# Create the buffer around each centroid
gdf_directly_affected_centroids['buffer'] =
 ↪  gdf_directly_affected_centroids.geometry.buffer(buffer_distance)

# Combine all individual buffers into a single unified buffer using union_all
buffer_union = gdf_directly_affected_centroids['buffer'].union_all()

# Create a GeoDataFrame for the unified buffer
gdf_buffer = gpd.GeoDataFrame(geometry=[buffer_union], crs=projected_crs)
```

```
# Compute centroids for all ZIP codes in the projected GeoDataFrame
gdf_zipcodes_projected['centroid'] = gdf_zipcodes_projected.geometry.centroid

# Create a GeoDataFrame of centroids for all ZIP codes
gdf_zip_centroids = gpd.GeoDataFrame(
    gdf_zipcodes_projected,
    geometry='centroid',
    crs=projected_crs
)
```

```
# Perform spatial join: find zip codes whose centroids are within the buffer
gdf_indirectly_affected = gpd.sjoin(
    gdf_zip_centroids,
    gdf_buffer,
    how='inner',
```

```
    predicate='within'
)
```

```
# Extract the list of directly affected zip codes
directly_affected_zips = set(gdf_directly_affected_centroids['ZIP_Code'])

# Filter out directly affected zip codes from indirectly affected
gdf_indirectly_affected = gdf_indirectly_affected[
    ~gdf_indirectly_affected['ZIP_Code'].isin(directly_affected_zips)
]

# Number of indirectly affected zip codes
num_indirectly_affected_zips = gdf_indirectly_affected['ZIP_Code'].nunique()

print(f"Number of indirectly affected zip codes in Texas:
 ↪  {num_indirectly_affected_zips}")
```

```
Number of indirectly affected zip codes in Texas: 136
```

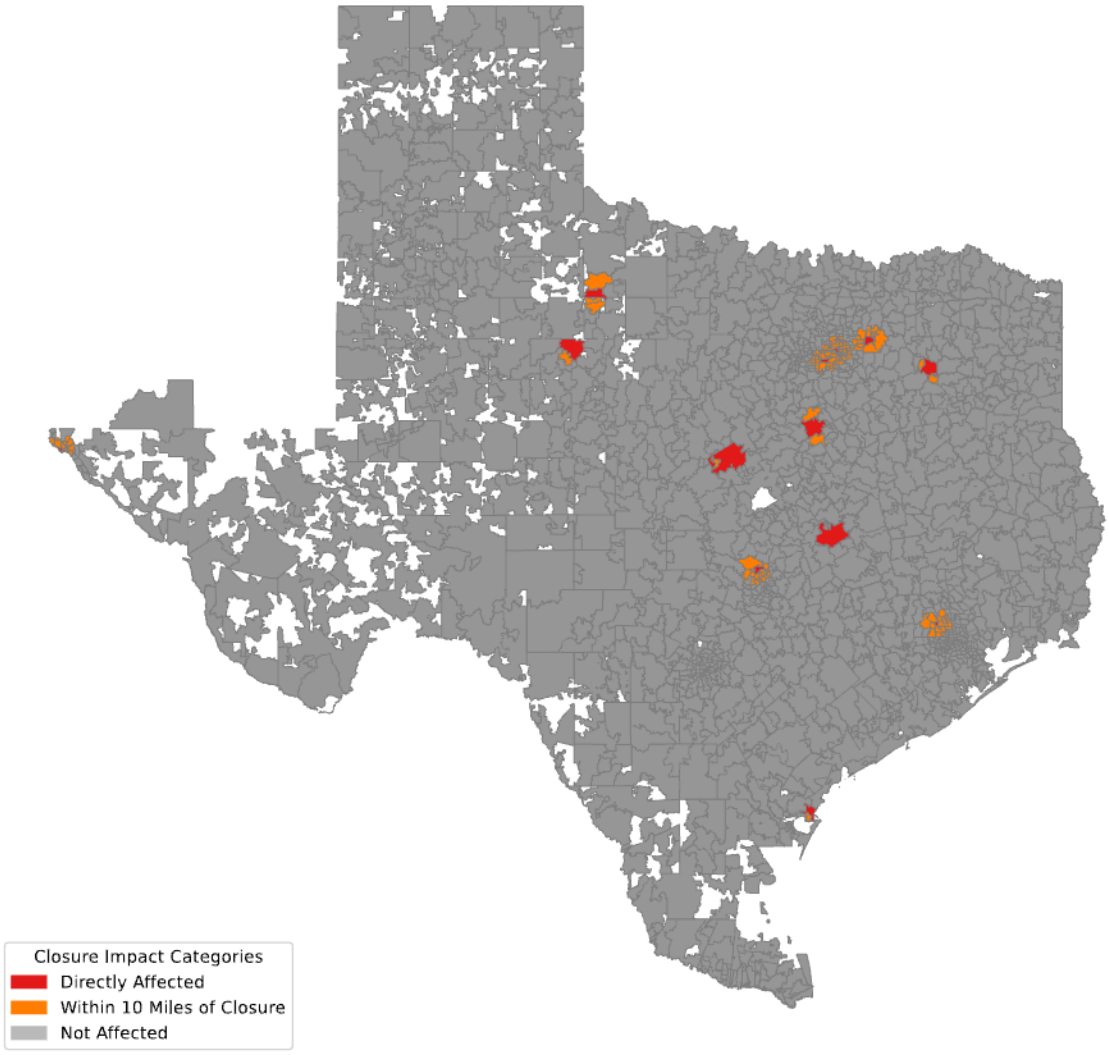
4.

```
# Define categories based on 'Affected' and indirectly affected ZIP codes
gdf_zipcodes['Category'] = 'Not Affected'  # Default category

# Assign 'Directly Affected' where 'Affected' == 1
gdf_zipcodes.loc[gdf_zipcodes['Affected'] == 1, 'Category'] = 'Directly
 ↪  Affected'

# Assign 'Indirectly Affected' where ZIP_Code is in gdf_indirectly_affected
indirectly_zips = set(gdf_indirectly_affected['ZIP_Code'])
gdf_zipcodes.loc[gdf_zipcodes['ZIP_Code'].isin(indirectly_zips), 'Category']
 ↪  = 'Indirectly Affected'
```

```
# Define color mapping for each category
category_colors = {
    'Directly Affected': '#e41a1c',
    'Indirectly Affected': '#ff7f00',
    'Not Affected': '#bdbdbd'
}

# Create a list of colors in the same order as the categories
```

```
colors = [category_colors[category] for category in gdf_zipcodes['Category']]

# Plot plot plot
import matplotlib.patches as mpatches

fig, ax = plt.subplots(1, 1, figsize=(12, 12))

gdf_zipcodes.plot(
    column='Category',
    categorical=True,
    cmap=plt.cm.get_cmap('Set1', 3),
    linewidth=0.1,
    ax=ax,
    edgecolor='grey',
    legend=False
)

# Create custom legend with defined category colors
legend_patches = [
    mpatches.Patch(color=category_colors['Directly Affected'],
 ↪  label='Directly Affected'),
    mpatches.Patch(color=category_colors['Indirectly Affected'],
 ↪  label='Within 10 Miles of Closure'),
    mpatches.Patch(color=category_colors['Not Affected'], label='Not
 ↪  Affected')
]

ax.legend(handles=legend_patches, loc='lower left', title='Closure Impact
 ↪  Categories')
ax.set_title('Texas ZIP Codes: Impact of Hospital Closures (2016-2019)',
 ↪  fontsize=16)
ax.axis('off')
plt.show()
```

```
/var/folders/27/5vfjdszd7w16ks8_5dyn2nvr0000gn/T/ipykernel_21045/1624408174.py:19:
MatplotlibDeprecationWarning:

The get_cmap function was deprecated in Matplotlib 3.7 and will be removed in
3.11. Use ``matplotlib.colormaps[name]`` or
``matplotlib.colormaps.get_cmap()`` or ``pyplot.get_cmap()`` instead.
```



Reflecting on the exercise (10 pts)

(Partner 1) The "first-pass" method for identifying hospital closures is a helpful starting point but has inherent limitations that can impact the accuracy of findings. One significant issue is data completeness and accuracy; relying solely on CMS data can lead to false positives or negatives due to incomplete or outdated information. Misclassification of provider type codes or subtypes may also result in facilities being wrongly identified as closed when they are not.

Another potential problem is the temporal lag in data reporting. Updates to CMS databases

may not reflect real-time changes, which means some closures or operational shifts might go unnoticed or be reported inaccurately. Additionally, changes in hospital CMS certification numbers due to reorganization or mergers can appear as closures when, in fact, the hospital remains operational under a different license.

To improve the accuracy of hospital closure identification, cross-referencing multiple data sources, such as state health department records or local news reports, is recommended. Verification through direct contact or site visits, when feasible, can also provide confirmation. Implementing time-series analyses to spot sudden drops in service or billing activity can help identify true closures.

(Partner 2) Current Approach: Directly Affected ZIP Codes: We identified ZIP codes that experienced at least one hospital closure between 2016 and 2019. These ZIP codes were considered "directly affected" by closures. Assumption: The presence of a closure within a ZIP code implies a reduction in hospital access for residents of that ZIP code. Limitations: (1)ZIP Code Boundaries vs. Real-World Behavior: Arbitrary Boundaries: ZIP codes are primarily used for mail delivery and may not correspond to how people access services like hospitals. Cross-ZIP Code Utilization: Residents often use hospitals located outside their own ZIP code, especially if they live near a boundary or in areas where hospitals are clustered. (2)Binary Measure of Impact: No Differentiation in Severity: The current method doesn't account for the size, capacity, or specialization of the closed hospital. Closing a major regional hospital has a different impact than closing a small clinic. Ignores Remaining Hospitals: ZIP codes with multiple hospitals might still have adequate access after a closure, while those with only one hospital might experience a significant loss. (3)Proximity to Hospitals in Neighboring ZIP Codes: Adjacent Access: A closure in one ZIP code can affect neighboring areas, especially if residents rely on that hospital due to proximity or lack of alternatives. Overlooking Nearby Facilities: Residents might have access to hospitals just across ZIP code boundaries, mitigating the impact of a closure within their own ZIP code. (4)Population Density and Demographics: Unequal Impact: The same number of closures in different ZIP codes can have varying effects depending on population size and demographics (e.g., age distribution, socioeconomic status). Vulnerable Populations: Some communities may be more adversely affected due to limited transportation options or higher healthcare needs.

Ways to Improve the Measure To more accurately reflect changes in ZIP-code-level access to hospitals, we can enhance our methodology by incorporating additional factors and employing more sophisticated spatial analyses. **1. Distance-Based Analysis: Calculate Changes in Distance to Nearest Hospital: (1) Before and After Analysis: For each ZIP code, compute the distance to the nearest hospital before and after the closures to assess the change in accessibility. (2) Use Network Distances: Utilize road networks to calculate actual travel distances or times rather than straight-line (Euclidean) distances. (3) Incorporate Public Transportation: Consider accessibility via public transit, which is critical for populations without private vehicles. Methodology: (1) Map All Hospitals: Create datasets of hospitals before closures (e.g., in 2015) and after closures (e.g., in 2019). (2) Compute Distances: Use spatial analysis**

tools to calculate the nearest hospital for each ZIP code centroid or population-weighted centroid. (3) Assess Impact: Identify ZIP codes where the nearest hospital is now significantly farther away. 2. Consider Hospital Capacity and Services: Hospital Size and Specialization: (1) Weighting: Assign weights to hospitals based on bed count, range of services, or patient volume to reflect their importance. (2) Service Gaps: Identify closures that eliminate critical services (e.g., maternity wards, emergency rooms). Methodology: (1) Collect Detailed Hospital Data: Gather information on hospital capacities and specialties. (2) Impact Scoring: Develop an impact score for each closure based on the significance of the lost services. **3. Population and Demographic Analysis: Assess Population Affected: (1) Population Density: Calculate the number of people affected by closures in each ZIP code. (2) Demographic Factors: Analyze the impact on vulnerable populations, such as the elderly, low-income families, or those with chronic illnesses.