\usepackage{fvextra} \DefineVerbatimEnvironment{Highlighting}{Verbatim}{breaklines,commandchars=\\\{\}}

\RecustomVerbatimEnvironment{verbatim}{Verbatim}{ showspaces = false, showtabs = false, breaksymbolleft={}, breaklines }

PS4

PS4: Due Sat Nov 2 at 5:00PM Central. Worth 100 points.

Style Points (10 pts)

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): Zhengye Chen ,allenzhengyechen • Partner 2 (name and cnet ID): Liling Shen, liling
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: **ZC LS**
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: **0** Late coins left after submission: **3**

"This submission is our work alone and complies with the 30538 integrity policy." **ZC LS** "I have uploaded the names of anyone else other than my partner and I worked with on the problem set here" ## Download and explore the Provider of Services (POS) file (10 pts)

Set up

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import numpy as np
import altair as alt
from shapely.geometry import Polygon
from shapely.geometry import MultiPoint
from shapely.ops import unary_union
from shapely.ops import nearest_points
import time
from pyproj import CRS
```

1. The variables I pulled: FAC_NAME #FacilityName CRTFCTN_ACTN_TYPE_CD #termination PRVDR_CTGRY_SBTYP_CD #short-term PRVDR_CTGRY_CD #hospital PRVDR_NUM #CMS PGM_TRMNTN_CD #termination ZIP_CD #zip code

2. a.

```
pos2016_df = pd.read_csv\
    ('POS_File_Hospital_Non_Hospital_Facilities_Q4_2016.csv')

short_hos_2016_df = pos2016_df[(pos2016_df['PRVDR_CTGRY_CD']\
    == 1) & (pos2016_df['PRVDR_CTGRY_SBTYP_CD'] == 1)]

number_of_hospitals_2016 = short_hos_2016_df.shape[0]
print(number_of_hospitals_2016)
```

```
7245
```

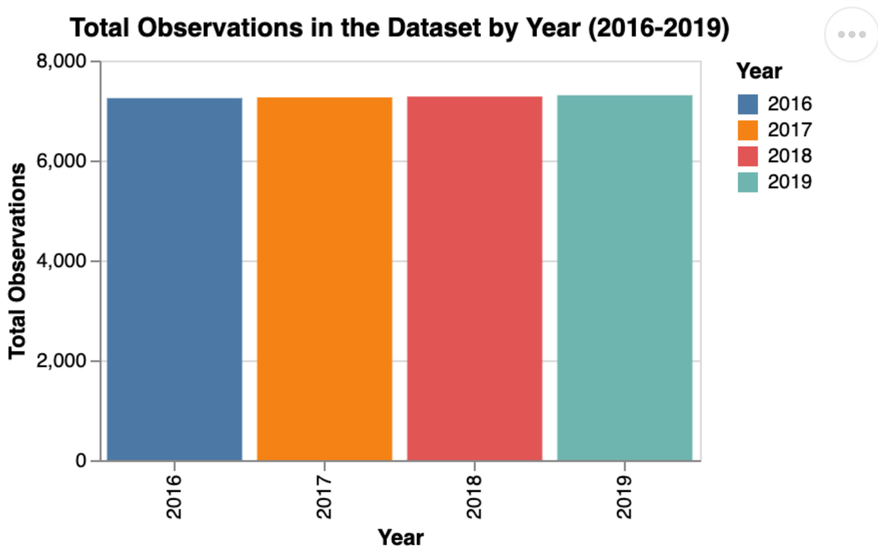
There are 7,245 short-term hospitals. This makes sense if the definition of a short-term hospital is broad and encompasses the entire United States.

b.
According to the American Hospital Association (AHA), there are 5,534 registered hospitals in the United States. The reported number of 7,245 short-term hospitals is much higher than the 5,534 registered by the AHA for 2016. This difference may be due to how data is collected. For example, hospitals might be classified in different ways based on various criteria or definitions.

3.

```
pos2017_df = pd.read_csv\
  ('POS_File_Hospital_Non_Hospital_Facilities_Q4_2017.csv')
pos2018_df = pd.read_csv\
  ('POS_File_Hospital_Non_Hospital_Facilities_Q4_2018.csv',\
    encoding='ISO-8859-1')
pos2019_df = pd.read_csv\
  ('POS_File_Hospital_Non_Hospital_Facilities_Q4_2019.csv',\
    encoding='ISO-8859-1')


short_hos_2017_df = pos2017_df[(pos2017_df['PRVDR_CTGRY_CD'] \
  == 1) & (pos2017_df['PRVDR_CTGRY_SBTYP_CD'] == 1)]
short_hos_2018_df = pos2018_df[(pos2018_df['PRVDR_CTGRY_CD'] \
  == 1) & (pos2018_df['PRVDR_CTGRY_SBTYP_CD'] == 1)]
short_hos_2019_df = pos2019_df[(pos2019_df['PRVDR_CTGRY_CD'] \
  == 1) & (pos2019_df['PRVDR_CTGRY_SBTYP_CD'] == 1)]

number_of_hospitals_2017 = short_hos_2017_df.shape[0]
number_of_hospitals_2018 = short_hos_2018_df.shape[0]
number_of_hospitals_2019 = short_hos_2019_df.shape[0]

short_hos_df = pd.DataFrame({
    'Year': ['2016', '2017', '2018', '2019'],
    'Number of Short-Term Hospitals': \
      [number_of_hospitals_2016,number_of_hospitals_2017,\
```

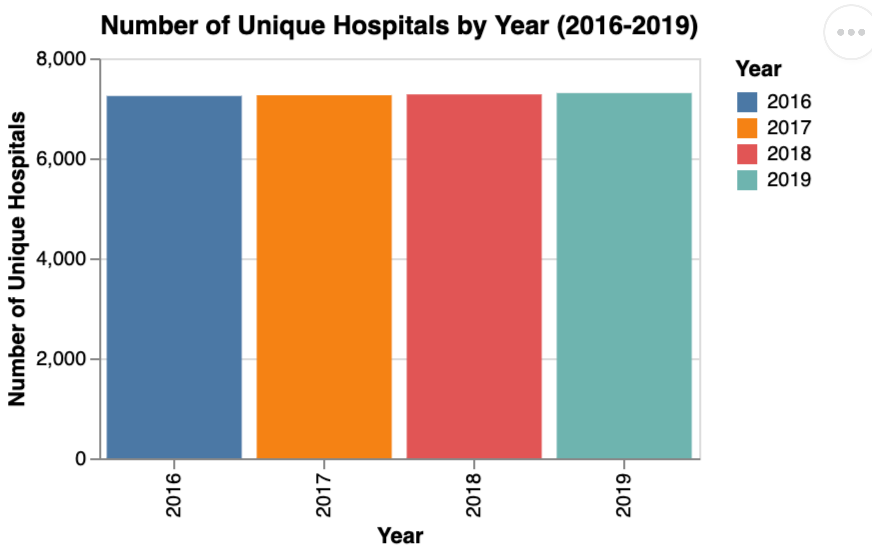
```
        number_of_hospitals_2018,number_of_hospitals_2019]
})

short_hos_plot = alt.Chart(short_hos_df).mark_bar().encode(
    x=alt.X('Year:O', title='Year'),
    y=alt.Y('Number of Short-Term Hospitals:Q', title='Total Observations'),
    color='Year:N',
).properties(
    title='Total Observations in the Dataset by Year (2016-2019)',
    width=300,
    height=200
)

short_hos_plot.show()
```



4. a.

```
unique_hospitals_count = {
    '2016': short_hos_2016_df['PRVDR_NUM'].nunique(),
    '2017': short_hos_2017_df['PRVDR_NUM'].nunique(),
    '2018': short_hos_2018_df['PRVDR_NUM'].nunique(),
    '2019': short_hos_2019_df['PRVDR_NUM'].nunique()
}

unique_hospitals_df = pd.DataFrame({
    'Year': ['2016', '2017', '2018', '2019'],
    'Unique Hospitals': [unique_hospitals_count['2016'],
                         unique_hospitals_count['2017'],
                         unique_hospitals_count['2018'],
                         unique_hospitals_count['2019']]
})

unique_hospitals_plot = alt.Chart(unique_hospitals_df).mark_bar().encode(
    x=alt.X('Year:O', title='Year'),
    y=alt.Y('Unique Hospitals:Q', title='Number of Unique Hospitals'),
```

```
    color='Year:N',
).properties(
    title='Number of Unique Hospitals by Year (2016-2019)',
    width=300,
    height=200
)

unique_hospitals_plot.show()
```



b.
The number of unique hospitals matches the total observations for each year, which indicates that there are no duplicate records in the dataset. This suggests a well-structured dataset where each hospital entry is distinct.

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

```
# Read csv files and specify encoding as ISO-8859-1 to avoid error
df_2016 = pd.read_csv("POS_File_Hospital_Non_Hospital_Facilities_Q4_2016.csv", encoding="
df_2017 = pd.read_csv("POS_File_Hospital_Non_Hospital_Facilities_Q4_2017.csv", encoding="
df_2018 = pd.read_csv("POS_File_Hospital_Non_Hospital_Facilities_Q4_2018.csv", encoding="
df_2019 = pd.read_csv("POS_File_Hospital_Non_Hospital_Facilities_Q4_2019.csv", encoding="
# Add column 'year' for filtering
df_2016['Year'] = 2016
df_2017['Year'] = 2017
df_2018['Year'] = 2018
df_2019['Year'] = 2019
# Merge data
all_years = pd.concat([df_2016, df_2017, df_2018, df_2019], ignore_index=True)
# Check data types and size
all_years.dtypes
all_years.shape
```

```
(581569, 8)
```

```python
# Filter datasets for short-term hospitals in each year
df_2016_filtered = df_2016[(df_2016['PRVDR_CTGRY_SBTYP_CD'] == 1) & (df_2016['PRVDR_CTGRY
df_2017_filtered = df_2017[(df_2017['PRVDR_CTGRY_SBTYP_CD'] == 1) & (df_2017['PRVDR_CTGRY
df_2018_filtered = df_2018[(df_2018['PRVDR_CTGRY_SBTYP_CD'] == 1) & (df_2018['PRVDR_CTGRY
df_2019_filtered = df_2019[(df_2019['PRVDR_CTGRY_SBTYP_CD'] == 1) & (df_2019['PRVDR_CTGRY
all_years_filtered = all_years[(all_years['PRVDR_CTGRY_SBTYP_CD'] == 1) & (all_years['PRV

# Combine 2017-2019 data and filter
df_2017_2019 = pd.concat([df_2017, df_2018, df_2019], ignore_index=True)
df_2017_2019_filtered = df_2017_2019[(df_2017_2019['PRVDR_CTGRY_SBTYP_CD'] == 1) & (df_20

# Output row counts only
print(df_2016_filtered.shape[0])
print(df_2017_filtered.shape[0])
print(df_2018_filtered.shape[0])
print(df_2019_filtered.shape[0])
print(all_years_filtered.shape[0])
print(df_2017_2019_filtered.shape[0])
```

```
7245
7260
7277
7303
29085
21840
```

1.

```python
# Filter hospitals active in 2016 & non-active in 2017-2019
active_2016 = df_2016_filtered[df_2016_filtered["PGM_TRMNTN_CD"] == 0]
non_active_2017_2019 = df_2017_2019_filtered[df_2017_2019_filtered["PGM_TRMNTN_CD"] != 0]

# Merge active 2016 hospitals with 2017-2019 non-active records
merged_hospitals = active_2016.merge(
    non_active_2017_2019[["PRVDR_NUM", "PGM_TRMNTN_CD", "Year"]],
    on="PRVDR_NUM",
    how="left",
    indicator=True
)

# Assign non-active year to Closure_Year in merged dataset using Year_y
merged_hospitals["Closure_Year"] = merged_hospitals["Year_y"]

# Keep only hospitals present in both datasets (active in 2016 and non-active after)
closed_hospitals = merged_hospitals[merged_hospitals["_merge"] == "both"]

# Find hospitals active in 2016 but disappeared in 2017-2019
disappeared_hospitals = active_2016[~active_2016["PRVDR_NUM"].isin(df_2017_2019_filtered[
disappeared_hospitals = disappeared_hospitals.assign(Closure_Year=2017)
```

```
# Combine non-active hospitals and disappeared hospitals
all_closed_hospitals = pd.concat([closed_hospitals, disappeared_hospitals], ignore_index=

# Sort by unique CMS number and closure year
all_closed_hospitals.sort_values(by=["PRVDR_NUM", "Closure_Year"], ascending=[False, True

# Keep the first year each hospital became non-active after 2016
final_closed_hospitals = all_closed_hospitals.groupby("PRVDR_NUM").aggregate(
    FAC_NAME=("FAC_NAME", "first"),
    ZIP_CD=("ZIP_CD", "first"),
    Closure_Year=("Closure_Year", "first")
).reset_index()

# Convert ZIP_CD and Closure_Year to integer type
final_closed_hospitals["ZIP_CD"] = final_closed_hospitals["ZIP_CD"].astype(int)
final_closed_hospitals["Closure_Year"] = final_closed_hospitals["Closure_Year"].astype(in

# Display the number of hospitals suspected to have closed by 2019
num_closed_hospitals = final_closed_hospitals.shape[0]
print(num_closed_hospitals)

# Just to check
print(final_closed_hospitals.head())
```

```
174
  PRVDR_NUM                           FAC_NAME  ZIP_CD  Closure_Year
0    010032                   WEDOWEE HOSPITAL   36278          2019
1    010047           GEORGIANA MEDICAL CENTER   36033          2019
2    010146                   RMC JACKSONVILLE   36265          2018
3    010172  NORTH ALABAMA SPECIALITY HOSPITAL   35611          2018
4    030001               ABRAZO MARYVALE CAMPUS   85031          2017
```

2.

```
# Sort by facility name
closed_hospitals_sorted = final_closed_hospitals.sort_values(by="FAC_NAME")

# Select and print the first 10 rows and display the names and closure year
first_10_closed_hospitals = closed_hospitals_sorted[["FAC_NAME", "Closure_Year"]].head(10
print(first_10_closed_hospitals)
```

```
                                       FAC_NAME  Closure_Year
4                        ABRAZO MARYVALE CAMPUS          2017
10       ADVENTIST MEDICAL CENTER - CENTRAL VALLEY          2017
97                        AFFINITY MEDICAL CENTER          2018
80   ALBANY MEDICAL CENTER / SOUTH CLINICAL CAMPUS          2017
140       ALLEGIANCE SPECIALTY HOSPITAL OF KILGORE          2017
62                          ALLIANCE LAIRD HOSPITAL          2019
101                        ALLIANCEHEALTH DEACONESS          2019
26                     ANNE BATES LEACH EYE HOSPITAL          2019
```

```
21         ARKANSAS VALLEY REGIONAL MEDICAL CENTER         2017
69             BANNER CHURCHILL COMMUNITY HOSPITAL         2017
```

3. a.

```
# Count active hospitals per zip codes per year
active_hospitals_by_zip = all_years_filtered.groupby(["ZIP_CD", "Year"]).size().reset_ind

# Identify zip codes with no decrease in active hospitals after suspected closures
closures_with_merger_check = final_closed_hospitals.merge(
    active_hospitals_by_zip,
    left_on=["ZIP_CD", "Closure_Year"],
    right_on=["ZIP_CD", "Year"],
    how="left"
)

closures_with_merger_check["Next_Year"] = closures_with_merger_check["Closure_Year"] + 1

# Get active hospital counts for the year following each suspected closure
closures_with_merger_check = closures_with_merger_check.merge(
    active_hospitals_by_zip,
    left_on=["ZIP_CD", "Next_Year"],
    right_on=["ZIP_CD", "Year"],
    how="left",
    suffixes=('', '_NextYear')
)

# Mark closures that are suspected of being mergers
closures_with_merger_check["Potential_Merger"] = (
    closures_with_merger_check["Active_Hospitals_NextYear"] >= closures_with_merger_check
)

# Filter out suspected mergers
final_closed_hospitals_corrected = closures_with_merger_check[~closures_with_merger_check

# Count the number of hospitals fitting the potential merger/acquisition definition
num_potential_mergers = closures_with_merger_check["Potential_Merger"].sum()

# Calculate the remaining number of hospitals after removing potential mergers
num_hospitals_after_correction = final_closed_hospitals_corrected.shape[0]

# Output results
print("Number of potentially being a merger/acquisition:", num_potential_mergers)
print("Number of hospitals left after correction:", num_hospitals_after_correction)
```

```
Number of potentially being a merger/acquisition: 96
Number of hospitals left after correction: 78
```

Number of potentially being a merger/acquisition: 96

b.

Number of hospitals left after correction: 78 c.

```
# Sort by facility name
corrected_closed_hospitals_sorted = final_closed_hospitals_corrected.sort_values(by="FAC_

# Select and display the first 10 in this list of corrected hospital closures
first_10_corrected_hospitals = corrected_closed_hospitals_sorted[["FAC_NAME", "ZIP_CD", "
print(first_10_corrected_hospitals)
```

```
                                            FAC_NAME  ZIP_CD  Closure_Year
62                             ALLIANCE LAIRD HOSPITAL   39365          2019
101                             ALLIANCEHEALTH DEACONESS   73112          2019
26                         ANNE BATES LEACH EYE HOSPITAL   33136          2019
115                        BARIX CLINICS OF PENNSYLVANIA   19047          2019
171                      BAYLOR EMERGENCY MEDICAL CENTER   75087          2019
166  BAYLOR SCOTT & WHITE EMERGENCY MEDICAL CENTER ...   78613          2019
98                          BELMONT COMMUNITY HOSPITAL   43906          2019
67                              BIG SKY MEDICAL CENTER   59716          2019
65                BLACK RIVER COMMUNITY MEDICAL CENTER   63901          2019
142                         CARE REGIONAL MEDICAL CENTER   78336          2019
```

Download Census zip code shapefile (10 pt)

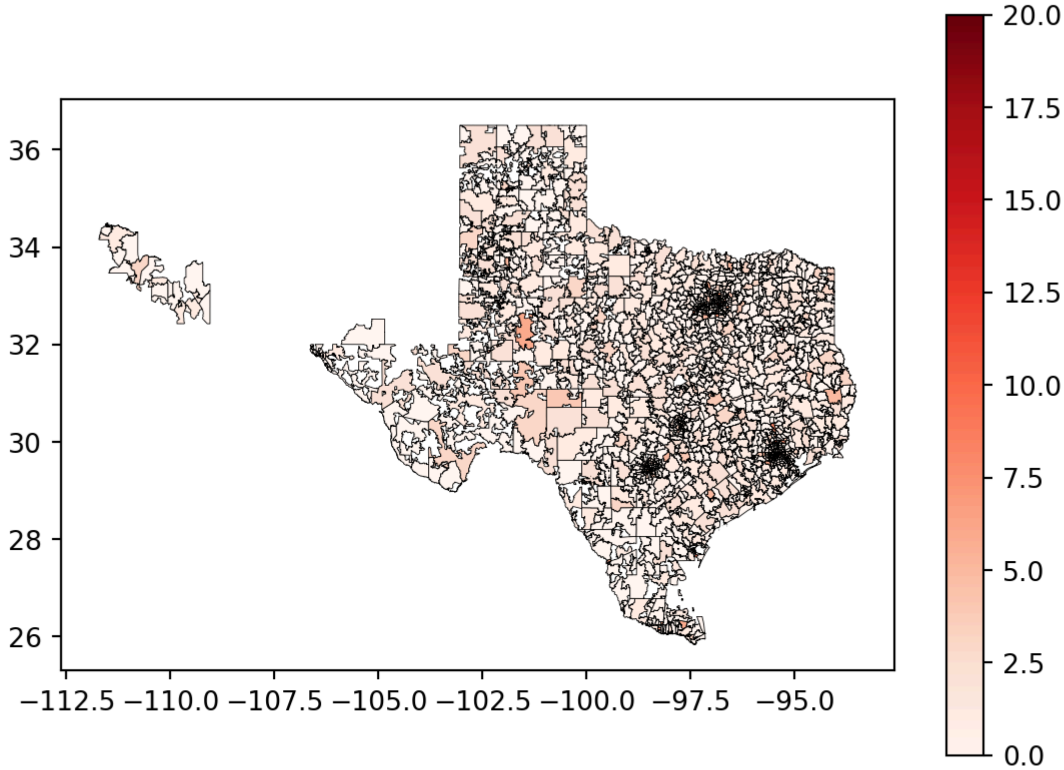
1. a. The five file types used in GIS data are .dbf, .prj, .shp, .shx, and .xml. The .dbf file holds information about geographic features in a table format. The .prj file specifies the coordinate system and mapping details for the spatial data. The .shp file contains the shapes and locations of the geographic features. The .shx file is an index that helps access the geometric data in the .shp file more quickly. Lastly, the .xml file includes metadata about the shapefile, such as the data source and attribute descriptions. Together, these files help manage and analyze spatial data.

   b. The .dbf file is 6.4MB. The .prj is 165 bytes. The .shp is 837.5MB. The .shx file is 265KB. The .xml file is 16KB.

2.

```
zip_codes_shapefile = gpd.read_file('gz_2010_us_860_00_500k.shp')
zip_codes_shapefile_tx = zip_codes_shapefile[zip_codes_shapefile['NAME'].str.startswith((


hospitals_per_zip_2016 = pos2016_df[pos2016_df['PRVDR_CTGRY_CD'] == 1].groupby('ZIP_CD')[

hospitals_per_zip_2016['ZIP_CD'] = hospitals_per_zip_2016['ZIP_CD'].astype(str).str.split

hospitals_per_zip_2016['ZIP_CD'] = hospitals_per_zip_2016['ZIP_CD'].astype(str)

zip_codes_shapefile_tx['NAME'] = zip_codes_shapefile_tx['NAME'].astype(str)
```

```
texas_hospitals = zip_codes_shapefile_tx.merge(hospitals_per_zip_2016, left_on='NAME', ri

texas_hospitals['num_hospital'] = texas_hospitals['num_hospital'].fillna(0)

texas_hospitals.plot(
    column="num_hospital",
    cmap="Reds",
    linewidth=0.3,
    edgecolor="black",
    legend=True
)
```



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

1.

```
# Read Shapefile
file_path_shp = "gz_2010_us_860_00_500k.shp"
zips_all = gpd.read_file(file_path_shp)
```

```
# Copy the GeoDataFrame and calculate the centroid of each ZIP code area
zips_all_centroids = zips_all.copy()
zips_all_centroids['geometry'] = zips_all_centroids.geometry.centroid # Create a new colu
```

```
# Print the dimensions of the resulting GeoDataFrame (number of rows and columns)
print("Dimensions of the resulting GeoDataFrame:", zips_all_centroids.shape)

# Display the first few rows to check
print(zips_all_centroids.head(10))
```

```
/var/folders/4h/s9_1g5zn2_qc7cdrphmtvzzm0000gn/T/ipykernel_34937/2526898498.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.

  zips_all_centroids['geometry'] = zips_all_centroids.geometry.centroid # Create a new
column

Dimensions of the resulting GeoDataFrame: (33120, 6)
           GEO_ID  ZCTA5   NAME   LSAD  CENSUSAREA                    geometry
0  8600000US01040  01040  01040  ZCTA5      21.281  POINT (-72.64107 42.21257)
1  8600000US01050  01050  01050  ZCTA5      38.329  POINT (-72.86985 42.28786)
2  8600000US01053  01053  01053  ZCTA5       5.131  POINT (-72.71162 42.35349)
3  8600000US01056  01056  01056  ZCTA5      27.205  POINT (-72.45805 42.19215)
4  8600000US01057  01057  01057  ZCTA5      44.907   POINT (-72.3243 42.09165)
5  8600000US01060  01060  01060  ZCTA5      10.918   POINT (-72.6313 42.32225)
6  8600000US01062  01062  01062  ZCTA5      18.154  POINT (-72.69279 42.32189)
7  8600000US01066  01066  01066  ZCTA5       0.710  POINT (-72.65482 42.40761)
8  8600000US01069  01069  01069  ZCTA5      28.065  POINT (-72.30543 42.18807)
9  8600000US01070  01070  01070  ZCTA5      20.541  POINT (-72.92387 42.51847)
```

Meaning of each Columns: * GEO_ID: A unique geographic identifier for each ZIP code area. It includes a prefix (8600000US) with the actual ZIP code. * ZCTA5: ZIP Code Tabulation Area (ZCTA) code, created by the U.S. Census Bureau and are very similar to postal ZIP code areas. * NAME: Repeats the ZIP code or ZCTA5 code. * LSAD: Specifies the Legal/Statistical Area Description. Here, it's set to "ZCTA5". * CENSUSAREA: The area of each ZIP code region. * geometry: The geometric data, now it contains the centroid points of each ZIP code area, i.e., the arithmetic mean position of all the points in a polygon. Previously, it contained Polygon data, representing the boundary of each ZIP code area.

2.

For zip codes (or ZCTA5), I refer to [website1](), [website2](), [website3]() and set the criterion that starting with 75-79 and '833'.

```
# Define Texas zip codes prefixes and specific full code
texas_prefixes = ('75', '76', '77', '78', '79')
specific_texas_code = '833'

# Define 4 bordering state zip codesprefixes
border_states_prefixes = {
    'New Mexico': ('87', '88'),
    'Oklahoma': ('73', '74'),
    'Arkansas': ('71', '72'),
    'Louisiana': ('70', '71')
```

```
}

# Combine texas and all bordering state zip codes prefixes into a single tuple
all_border_state_prefixes = texas_prefixes + sum(border_states_prefixes.values(), ())

# Filter Texas zip codes
zips_texas_centroids = zips_all_centroids[
    zips_all_centroids['ZCTA5'].str.startswith(texas_prefixes) |
    (zips_all_centroids['ZCTA5'] == specific_texas_code)
]

# Filter Texas and bordering states zip codes
zips_texas_borderstates_centroids = zips_all_centroids[
    zips_all_centroids['ZCTA5'].str.startswith(all_border_state_prefixes)
]

# Count unique zip codes in each subset
num_texas_zip_codes = zips_texas_centroids['ZCTA5'].nunique()
num_borderstates_zip_codes = zips_texas_borderstates_centroids['ZCTA5'].nunique()

print(f"Unique Texas zip codes: {num_texas_zip_codes}")
print(f"Unique Texas and bordering states zip codes: {num_borderstates_zip_codes}")

# Function to check if two polygons intersect
def polygons_intersect(poly1, poly2):
    """Return True if two polygons intersect, False otherwise."""
    return poly1.intersects(poly2)

# Combine all Texas zip codes centroids into a single MultiPoint object and create a conv
texas_centroids = zips_texas_centroids['geometry']
texas_centroid_union = MultiPoint(list(texas_centroids)).convex_hull

# Identify bordering states by checking intersection with Texas's centroid polygon
bordering_states = []
for state, prefixes in border_states_prefixes.items():
    # Filter zip codes for the current state using centroids
    state_zip_centroids = zips_all_centroids[zips_all_centroids['ZCTA5'].str.\
        startswith(prefixes)]['geometry']

    # Combine the state's zip code centroids into a single MultiPoint and create a convex
    state_centroid_union = MultiPoint(list(state_zip_centroids)).convex_hull

    # Check if the state's centroid polygon intersects with Texas's centroid polygon
    if polygons_intersect(texas_centroid_union, state_centroid_union):
        bordering_states.append(state)

print(f"Bordering states: {bordering_states}")
```

```
Unique Texas zip codes: 1935
Unique Texas and bordering states zip codes: 4057
```

```
Bordering states: ['New Mexico', 'Oklahoma', 'Arkansas', 'Louisiana']
```

3.

```
# Pre-check data types for better merge
print(zips_all_centroids.dtypes)
print(df_2016_filtered['ZIP_CD'].unique())
```

```
GEO_ID          object
ZCTA5           object
NAME            object
LSAD            object
CENSUSAREA     float64
geometry      geometry
dtype: object
[36301. 35740. 35957. ... 77584. 78654. 78249.]
```

```
# Convert ZIP_CD to string
df_2016_filtered['ZIP_CD'] = df_2016_filtered['ZIP_CD'].fillna(0).apply(lambda x: str(int

# Filter zip codes with at least one active hospital in 2016
# Only include unique ZIP codes where there is at least one active hospital
hospitals_2016_zips = df_2016_filtered[df_2016_filtered['PGM_TRMNTN_CD'] == 0][['ZIP_CD']

# Merge to create zips_withhospital_centroids
zips_withhospital_centroids = zips_texas_borderstates_centroids.merge(
    hospitals_2016_zips, left_on='ZCTA5', right_on='ZIP_CD', how='inner'
)

# Check the resulting GeoDataFrame
print(zips_withhospital_centroids.head())
```

```
           GEO_ID  ZCTA5   NAME   LSAD  CENSUSAREA  \
0  8600000US70043  70043  70043  ZCTA5       7.775
1  8600000US70127  70127  70127  ZCTA5       7.095
2  8600000US70301  70301  70301  ZCTA5     288.050
3  8600000US70360  70360  70360  ZCTA5      64.325
4  8600000US70403  70403  70403  ZCTA5      42.960

                     geometry ZIP_CD
0  POINT (-89.96276 29.94804)  70043
1  POINT (-89.97675 30.02501)  70127
2   POINT (-90.74089 29.8141)  70301
3  POINT (-90.81028 29.58819)  70360
4  POINT (-90.48388 30.48002)  70403

/var/folders/4h/s9_1g5zn2_qc7cdrphmtvzzm0000gn/T/ipykernel_34937/514204334.py:2:
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_2016_filtered['ZIP_CD'] = df_2016_filtered['ZIP_CD'].fillna(0).apply(lambda x: str(int(float(x))) if x != '' else '')
```

Answer: * I did a inner merge. * I merged on 'ZCTA5' and 'ZIP_CD'.

4. a.

```
print(len(zips_texas_centroids))  # The answer is 1935
```

```
1935
```

```
# Subset 10 zip codes from zips_texas_centroids
sample_zips_texas = zips_texas_centroids.sample(10)

# Record start time
start_time = time.time()

# Initialize a list to store distances to the nearest hospital zip codes
distances = []

# Calculate the distance from each sample zip codes to the nearest zip codes with a hospi
for zip_code_geom in sample_zips_texas.geometry:
    # Find the nearest geometry in zips_withhospital_centroids
    nearest_geom = nearest_points(zip_code_geom, zips_withhospital_centroids.unary_union)
    # Calculate distance to the nearest geometry
    distance = zip_code_geom.distance(nearest_geom)
    distances.append(distance)

# Record end time
end_time = time.time()

# Calculate time taken for the sample subset
sample_duration = end_time - start_time
print(f"Time taken for 10 zip codes: {sample_duration:.2f} seconds")

# Estimate the total time for the full dataset
estimated_time_full = (sample_duration / 10) * 1935
print(f"Estimated time: {estimated_time_full:.2f} seconds")
```

```
Time taken for 10 zip codes: 0.01 seconds
Estimated time: 1.09 seconds

/var/folders/4h/s9_1g5zn2_qc7cdrphmtvzzm0000gn/T/ipykernel_34937/2458686753.py:13: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  nearest_geom = nearest_points(zip_code_geom, zips_withhospital_centroids.unary_union)
[1]
```

b.

```
# Record start time
start_time_full = time.time()

# Initialize a list to store distances
all_distances = []

# Calculate the distance from each sample zip codes to the nearest zip codes with a hospi
for zip_code_geom in zips_texas_centroids.geometry:
    # Find the nearest geometry in zips_withhospital_centroids
    nearest_geom = nearest_points(zip_code_geom, zips_withhospital_centroids.unary_union)
    # Calculate distance to the nearest geometry
    distance = zip_code_geom.distance(nearest_geom)
    all_distances.append(distance)

# Record end time
end_time_full = time.time()

# Calculate time taken for the full dataset
actual_duration_full = end_time_full - start_time_full
print(f"Actual time: {actual_duration_full:.2f} seconds")
```

```
/var/folders/4h/s9_1g5zn2_qc7cdrphmtvzzm0000gn/T/ipykernel_34937/500455622.py:10:
DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()'
method instead.
  nearest_geom = nearest_points(zip_code_geom, zips_withhospital_centroids.unary_union)
[1]

Actual time: 0.70 seconds
```

Answer: * In my latest attempt (I try to run them simultaneously), the estimated time was 0.97 seconds, while the actual time was 0.66 seconds. * The estimated time is slightly higher than the actual time, about 32% faster than the estimate. It could be due to factors like caching, resource usage or variability in processing time per zip codes.

c.

```
# Set path for .prj file
file_path_prj = "gz_2010_us_860_00_500k.prj"

# Read the .prj file content as a WKT string
with open(file_path_prj, 'r') as f:
    prj_wkt = f.read()  # This line should be indented

# Create a CRS object from the WKT content
crs = CRS.from_wkt(prj_wkt)
```

```
# Print the CRS information
print("WKT Projection:", crs.to_wkt())
```

```
WKT Projection: GEOGCRS["NAD83",DATUM["North American Datum 1983",ELLIPSOID["GRS 1980",6378137,298.257222101,LENGTHUNIT["metre",1]],ID["EPSG",6269]],PRIMEM["Greenwich",0,ANGLEUNIT["Degree",0.0174532925199433]],CS[ellipsoidal,2],AXIS["longitude",east,ORDER[1],ANGLEUNIT["Degree",0.0174532925199433]],AXIS["latitude",north,ORDER[2],ANGLEUNIT["Degree",0.0174532925199433]]]
```

```
Click into the .prj file, I find the code:
```

```
GEOGCS["GCS_North_American_1983",DATUM["D_North_American_1983",\
SPHEROID["GRS_1980",6378137,298.257222101]],PRIMEM["Greenwich",0],UNIT["Degree",0.017453292519943295]]
```

```
Thus, the projection uses degrees as the unit of measurement.

The approximate conversion I find is:
* 1 degree ≈ 69 (or 69.4) miles
* To convert the given unit to miles, we can multiply the result by 69 (or 69.4).
([Reference](https://www.sco.wisc.edu/2022/01/21/how-big-is-a-degree/))
```

5. a.

It's in degrees as unit.

b.

```
# Calculate the average distance in degrees
average_distance_degrees = sum(all_distances) / len(all_distances)

# Convert the average distance from degrees to miles
average_distance_miles = average_distance_degrees * 69
print(f"Average distance in miles: {average_distance_miles:.2f} miles")
```

```
Average distance in miles: 14.56 miles
```

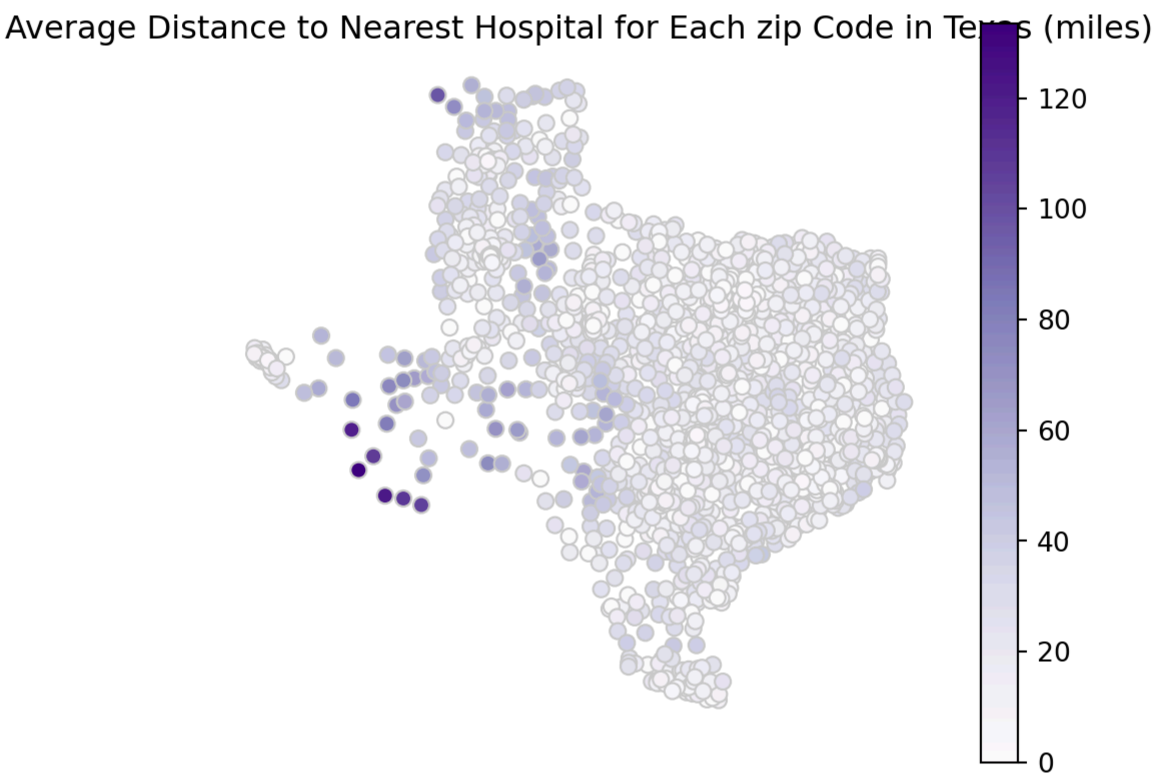
Answer: * Yes, it make sense. * Texas is the second-largest state in the U.S. by area, with many rural areas and a lower population density outside of major cities. This may lead to larger average distance. * Rural zip codes may be much farther from the nearest hospital.

c.

```
# Convert each distance from degrees to miles
all_distances_miles = [distance * 69 for distance in all_distances]

# Make a copy of zips_texas_centroids
zips_texas_centroids_copy = zips_texas_centroids.copy()
```

```
# Add a new column for distances in miles
zips_texas_centroids_copy['Distance_to_Nearest_Hospital'] = all_distances_miles
```

```
# Plot the map color-coded by the distance
fig, ax = plt.subplots(1, 1, figsize=(6, 5))
zips_texas_centroids_copy.plot(column="Distance_to_Nearest_Hospital",
                               cmap="Purples",
                               linewidth=0.8,
                               ax=ax,
                               edgecolor="0.8",
                               legend=True)

# Set title and remove coordinate axes
plt.title("Average Distance to Nearest Hospital for Each zip Code in Texas (miles)")
plt.axis("off")

# Show the plot
plt.show()
```



Effects of closures on access in Texas (15 pts)

1.

```
final_closed_hospitals_corrected['ZIP_CD'] = final_closed_hospitals_corrected['ZIP_CD'].a
```

```
texas_prefixes = ('75', '76', '77', '78', '79', '855')
texas_closures = final_closed_hospitals_corrected[final_closed_hospitals_corrected['ZIP_C
    .str.startswith(texas_prefixes)]

closures_by_zipcode = texas_closures.groupby('ZIP_CD').size().reset_index(name='Number_of

closures_by_zipcode.sort_values(by='Number_of_Closures', ascending=False, inplace=True)

print(closures_by_zipcode)
```

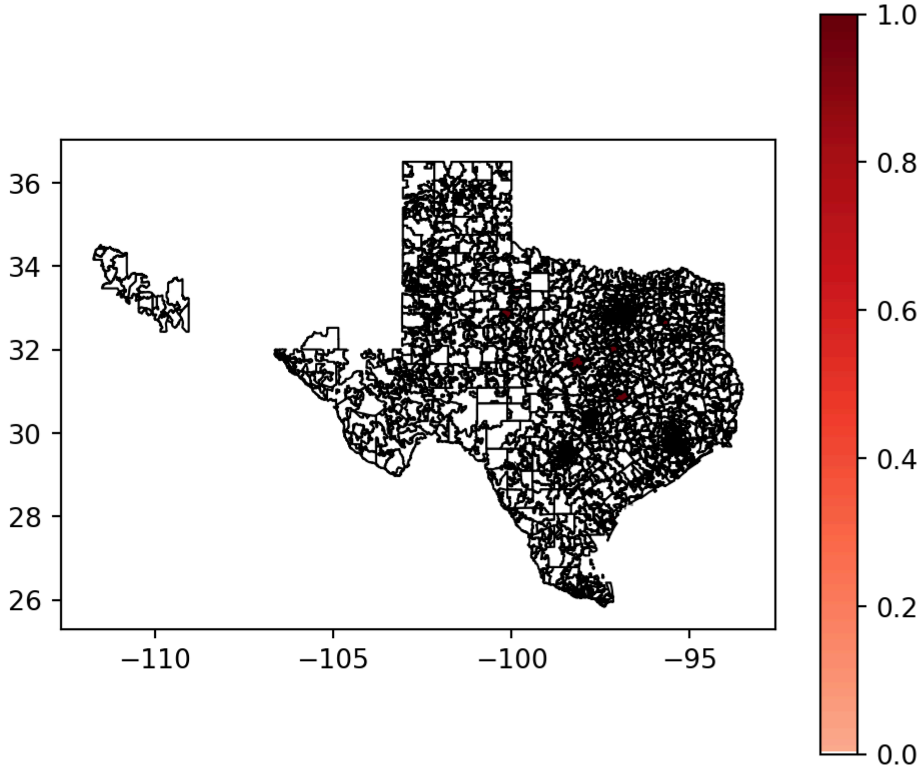
```
   ZIP_CD  Number_of_Closures
0   75051                   1
1   75087                   1
2   75140                   1
3   75235                   1
4   75390                   1
5   76520                   1
6   76531                   1
7   76645                   1
8   77065                   1
9   78336                   1
10  78613                   1
11  79520                   1
12  79529                   1
13  79902                   1

/var/folders/4h/s9_1g5zn2_qc7cdrphmtvzzm0000gn/T/ipykernel_34937/3239957475.py:1:
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
  final_closed_hospitals_corrected['ZIP_CD'] =
final_closed_hospitals_corrected['ZIP_CD'].astype(str)
```

2.

```
texas_closures_map = zip_codes_shapefile_tx.merge(
closures_by_zipcode, left_on='NAME', right_on='ZIP_CD', how='left'
)

texas_closures_map['Number_of_Closures'] = texas_closures_map['Number_of_Closures'].filln

cmap = mcolors.ListedColormap(['white'] + plt.cm.Reds(np.linspace(0.3, 1, 256)).tolist())

fig, ax = plt.subplots(1, 1, figsize=(6, 5))
texas_closures_map.plot(
    column='Number_of_Closures',
    cmap=cmap,
    linewidth=0.8,
```

```
    edgecolor='black',
    legend=True,
    ax=ax
)

directly_affected_zip_count = texas_closures_map[texas_closures_map['Number_of_Closures']
print(f'Number of directly affected ZIP codes in Texas: {directly_affected_zip_count}')
```

```
Number of directly affected ZIP codes in Texas: 14
```




```
#print(zip_codes_shapefile_tx)
#print(closures_by_zipcode)
print(texas_closures_map)
```

```
            GEO_ID  ZCTA5   NAME   LSAD  CENSUSAREA  \
0     8600000US78624  78624  78624  ZCTA5     708.041
1     8600000US78626  78626  78626  ZCTA5      93.046
2     8600000US78628  78628  78628  ZCTA5      73.382
3     8600000US78631  78631  78631  ZCTA5     325.074
4     8600000US78632  78632  78632  ZCTA5      96.278
...              ...    ...    ...    ...         ...
1950  8600000US78261  78261  78261  ZCTA5      29.865
1951  8600000US78368  78368  78368  ZCTA5     216.341
1952  8600000US78412  78412  78412  ZCTA5       8.798
1953  8600000US78557  78557  78557  ZCTA5      11.653
1954  8600000US78586  78586  78586  ZCTA5     176.313
```

```
                                          geometry ZIP_CD  \
0     POLYGON ((-98.96423 30.49848, -98.96416 30.498...    NaN
1     POLYGON ((-97.60944 30.57185, -97.61688 30.568...    NaN
2     POLYGON ((-97.69285 30.57122, -97.69286 30.571...    NaN
3     POLYGON ((-99.13053 30.36555, -99.13065 30.365...    NaN
4     POLYGON ((-97.40946 29.75929, -97.40947 29.758...    NaN
...                                                 ...    ...
1950  POLYGON ((-98.44369 29.71944, -98.44363 29.719...    NaN
1951  POLYGON ((-97.85308 28.25868, -97.8516 28.2561...    NaN
1952  POLYGON ((-97.30819 27.70988, -97.30819 27.709...    NaN
1953  POLYGON ((-98.20496 26.06642, -98.20503 26.066...    NaN
1954  POLYGON ((-97.59936 26.19655, -97.59524 26.195...    NaN

      Number_of_Closures
0                    0.0
1                    0.0
2                    0.0
3                    0.0
4                    0.0
...                  ...
1950                 0.0
1951                 0.0
1952                 0.0
1953                 0.0
1954                 0.0

[1955 rows x 8 columns]
```

Number of directly affected ZIP codes in Texas: 14

3.

```python
texas_closures_map['ZIP_CD'] = texas_closures_map['NAME'].astype(str)

directly_affected_zips = texas_closures_map[texas_closures_map['Number_of_Closures'] > 0]

if directly_affected_zips.crs.is_geographic:
    directly_affected_zips = directly_affected_zips.to_crs(epsg=5070)

buffer_distance = 10 * 1609.34

directly_affected_zips['geometry'] = directly_affected_zips.buffer(buffer_distance)

texas_closures_map = texas_closures_map.to_crs(directly_affected_zips.crs)

indirectly_affected_zips = gpd.sjoin(texas_closures_map, directly_affected_zips, how='inn

indirectly_affected_zips = indirectly_affected_zips[
    ~indirectly_affected_zips['ZIP_CD_left'].isin(directly_affected_zips['ZIP_CD'])
]
```

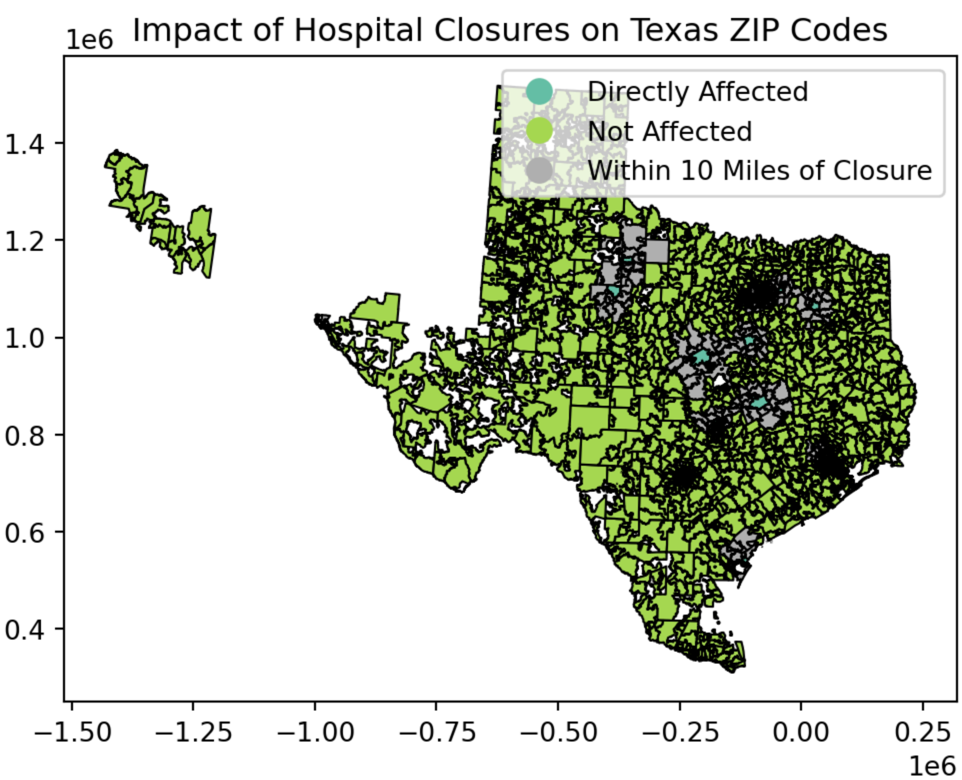
```
num_indirectly_affected_zips = indirectly_affected_zips['ZIP_CD_left'].nunique()
print(f'Number of indirectly affected ZIP codes in Texas: {num_indirectly_affected_zips}'
```

```
Number of indirectly affected ZIP codes in Texas: 342
```

There are 342 indirectly affected zip codes in Texas?

4.

```
texas_closures_map['Impact_Category'] = 'Not Affected'

texas_closures_map.loc[texas_closures_map\
    ['ZIP_CD'].isin(directly_affected_zips['ZIP_CD']), 'Impact_Category'] = 'Directly Aff

texas_closures_map.loc[texas_closures_map['ZIP_CD'].isin(indirectly_affected_zips\
    ['ZIP_CD_left']), 'Impact_Category'] = \
        'Within 10 Miles of Closure'

fig, ax = plt.subplots(1, 1, figsize=(6, 5))
texas_closures_map.plot(
    column='Impact_Category',
    cmap='Set2',
    linewidth=0.8,
    edgecolor='black',
    legend=True,
    ax=ax
)
ax.set_title('Impact of Hospital Closures on Texas ZIP Codes')
plt.show()
```



Reflecting on the exercise (10 pts)

1. The “first-pass” method of excluding suspected closures in ZIP codes where the number of active hospitals does not decrease in the following year is a useful initial filter, but it falls short of fully addressing the complexities of identifying true closures. This approach helps filter out cases where hospital services continue under a new CMS certification number due to mergers, acquisitions, or administrative changes, as these situations typically do not reduce the number of active hospitals in a ZIP code. However, this method has several limitations.

Temporary closures—such as those for renovations, natural disasters, or public health crises—could still be mistakenly flagged as permanent if a hospital doesn’t return to the dataset in the following year. Furthermore, data reporting inconsistencies or delays could misclassify active hospitals as closed if their records aren’t updated in a timely manner.

To build a more accurate closure identification method, a multi-year analysis would help confirm whether a hospital is truly out of service, as closures that persist for several years are more likely to be permanent. Cross-referencing suspected closures with external data sources, such as state health department records, hospital association databases, or news announcements, could confirm whether a closure is due to a permanent shutdown or an administrative change. Additionally, incorporating the distance to nearby hospitals would help account for closures in neighboring ZIP codes that affect patient access. While the first-pass method is a practical start, a more comprehensive approach that considers geographic context, multi-year trends, and external data verification would improve the reliability of closure identification.

2.

- a. In Section 2, We identify zip codes affected by closures by checking if the number of active hospitals decreases after a suspected closure. If the number of active hospitals remains the same or increases in the year following the closure, we flag the closure as a potential merger or acquisition, not an actual loss of access. After filtering out these suspected mergers, we are left with only closures likely to impact hospital access in those zip codes.
- b. This method of identifying closures by checking for “no decrease” in active hospitals has limitations and may not reliably distinguish between real closures and potential mergers or reclassifications: If the closure reflects only in the data for the same year, a real closure from 2017 to 2018 would show no change, while a false closure (like a merger or reclassification) might show an increase in 2018. This means that the “no decrease” criteria can be misleading, as it cannot accurately confirm the nature of the closure.
- c. Improvement: i. Track the number of active hospitals over multiple years, since a sustained decrease over several years would more likely indicate a true closure; ii. Consider the impact of hospitals in nearby zip codes, since residents may rely on hospitals in neighboring zip codes (spillover); iii. Cross-reference suspected closures with additional datasets or hospital registry information to identify cases of temporary closures or administrative changes, improving the accuracy.