

PS 4

PS4: Due Sat Nov 2 at 5:00PM Central. Worth 100 points. We use (*) to indicate a problem that we think might be time consuming.

Style Points (10 pts)

Please refer to the minilesson on code style [here](#).

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): Alejandra Silva - aosilva
 - Partner 2 (name and cnet ID): Guillermina Marto - gmarto
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: **AS GM**
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: **\1** Late coins left after submission: **\3**
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

Important: Repositories are for tracking code. **Do not commit the data or shapefiles to your repo.** The best way to do this is with `.gitignore`, which we have covered in class. If you do accidentally commit the data, Github has a [guide](#). The best course of action depends on whether you have pushed yet. This also means that both partners will have to download the initial raw data and any data cleaning code will need to be re-run on both partners' computers.

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

1.

For `pos2016.csv`, We selected the following variables to streamline the dataset for analysis: `PRVDR_CTGRY_CD` (Provider Category Code) and `PRVDR_CTGRY_SBTYP_CD` (Provider Subtype Code) to categorize providers; `PRVDR_NUM` (CMS Certification Number) to uniquely identify each hospital; `PGM_TRMNTN_CD` (Termination Code) to flag closures; `FAC_NAME` (Facility Name) for context on specific hospitals; `ZIP_CD` (ZIP Code) for mapping closures at the ZIP level; and `STATE_CD` (State Abbreviation) to filter for Texas and neighboring states. These fields allow for targeted analysis of hospital closures and their geographic impact, removing unnecessary data.

```
import requests
import pandas as pd
import altair as alt
import numpy as np

# Base URL for accessing the CMS dataset API
base_url = "https://data.cms.gov/data-api/v1/dataset/{uuid}/data"
uuid = "96ba2257-2080-49c1-9e5b-7726f9f83cad" # Unique ID for the specific
    ↴ dataset

# Columns to be retrieved from the dataset
columns = [
    "PRVDR_CTGRY_CD",          # Provider Category Code
    "PRVDR_CTGRY_SBTYP_CD",    # Provider Subtype Code
    "PRVDR_NUM",               # CMS Certification Number
    "PGM_TRMNTN_CD",           # Termination Code
    "FAC_NAME",                # Facility Name
    "ZIP_CD",                  # ZIP Code
    "STATE_CD"                 # State Abbreviation
]

# Prepare columns for API request
```

```

columns_param = ",".join(columns)

# Initialize offset and limit for paginated API requests
offset = 0
limit = 5000 # Maximum size per request set to 5000 as per API limit

all_data = [] # List to accumulate all retrieved data

while True:
    # Set parameters for API request
    params = {
        "column": columns_param,
        "size": limit,
        "offset": offset
    }

    # Format URL with the dataset UUID
    url = base_url.format(uuid=uuid)
    response = requests.get(url, params=params)

    # Check if the response is successful
    if response.status_code != 200:
        print(f"Error: {response.status_code}, {response.text}")
        break

    # Parse the response JSON data
    data = response.json()

    # Check if data is empty, indicating the end of available data
    if not data:
        print("No more data available.")
        break

    # Append retrieved data to the list
    all_data.extend(data)

    # Increment offset for the next batch of data
    offset += limit

# Create a DataFrame from the accumulated data
df = pd.DataFrame(all_data)

```

```
# Export the DataFrame to a CSV file without the index column
df.to_csv("pos2016.csv", index=False)
```

No more data available.

2.

```
# Read data from the CSV file into a DataFrame
df = pd.read_csv("pos2016.csv")

# Filter the DataFrame to include only short-term hospitals
# based on specific category and subtype codes
df_st_hospitals = df[
    # Provider Category Code for short-term hospitals
    (df["PRVDR_CTGRY_CD"] == 1) &
    # Provider Subtype Code for short-term hospitals
    (df["PRVDR_CTGRY_SBTYP_CD"] == 1)
]

# Count the number of short-term hospitals
num_hospitals = df_st_hospitals.shape[0]

# Display the number of short-term hospitals
display(
    f"Number of short-term hospitals reported in the data: {num_hospitals}")
```

'Number of short-term hospitals reported in the data: 7245'

According to the American Hospital Association (AHA) Annual Survey, the estimated number of short-term hospitals is 4,500–5,000. Similarly, the CMS Hospital Compare dataset indicates around 4,800 hospitals. The discrepancy could be due to the narrower definition used in our dataset and the timing of data collection, which only includes hospitals in Q4 2016. Additionally, the CMS dataset might not include hospitals that do not participate in Medicare or Medicaid, which could lead to lower numbers.

3.

```
uuid_dict = {
    "2016Q4": "96ba2257-2080-49c1-9e5b-7726f9f83cad",
    "2017Q4": "d338dc0d-641c-486a-b586-88a662f36963",
    "2018Q4": "4ff7fcfb-2a40-4f76-875d-a4ac2aec268e",
    "2019Q4": "03cca0cc-13a0-4b8d-82c4-57185b6bbfbdb"
```

```

}

columns = [
    "PRVDR_CTGRY_CD",           # Provider Category Code
    "PRVDR_CTGRY_SBTYP_CD",     # Provider Subtype Code
    "PRVDR_NUM",                # CMS Certification Number
    "PGM_TRMNTN_CD",            # Termination Code
    "FAC_NAME",                 # Facility Name
    "ZIP_CD",                   # ZIP Code
    "STATE_CD"                  # State Abbreviation
]

# Prepare columns for the API request
columns_param = ",".join(columns)

combined_data = [] # List to store data from all years

# Loop through each year and fetch data
for year_quarter, uuid in uuid_dict.items():
    offset = 0
    limit = 5000 # API request limit
    all_data = []

    print(f"Fetching data for {year_quarter}...")

    while True:
        # Set parameters for API request
        params = {
            "column": columns_param,
            "size": limit,
            "offset": offset
        }

        # Create the complete API URL
        url = f"https://data.cms.gov/data-api/v1/dataset/{uuid}/data"
        response = requests.get(url, params=params)

        if response.status_code != 200:
            print(f"Error: {response.status_code}, {response.text}")
            break

        data = response.json()

```

```

# Stop loop if no more data is returned
if not data:
    print("No more data available.")
    break

all_data.extend(data)
offset += limit

year_data = pd.DataFrame(all_data)
year_data["Year"] = year_quarter[:4] # Add year column for reference

# Filter DataFrame for short-term hospitals
year_data = year_data[
    (year_data["PRVDR_CTGRY_CD"] == "01") &
    (year_data["PRVDR_CTGRY_SBTYP_CD"] == "01")
]

# Append filtered data to the combined list
combined_data.append(year_data)

# Concatenate all the yearly DataFrames into one
combined_df = pd.concat(combined_data, axis=0)

# Save combined data to a CSV file
combined_df.to_csv("combined_data.csv", index=False)

# Print total number of records in the combined DataFrame
print(f"Total records retrieved across all years: {combined_df.shape[0]}")

```

```

Fetching data for 2016Q4...
No more data available.
Fetching data for 2017Q4...
No more data available.
Fetching data for 2018Q4...
No more data available.
Fetching data for 2019Q4...
No more data available.
Total records retrieved across all years: 29085

```

```
# Plotting Number of Observations Per Year
```

```

combined_year_df = combined_df.groupby(
    "Year").size().reset_index(name="Number of Observations")

obs_chart = alt.Chart(combined_year_df).mark_bar().encode(
    x=alt.X("Year:O", title="Year"),
    y=alt.Y("Number of Observations:Q", title="Number of Observations"),
    tooltip=["Year", "Number of Observations"]
).properties(
    title="Number of Observations Per Year"
)

obs_chart.show()

```

alt.Chart(...)

4. a.

```

# Plotting Number of Unique Hospitals Per Year
unique_hospitals = combined_df.groupby("Year")["PRVDR_NUM"].nunique(
).reset_index(name="Number of Unique Hospitals")

unique_hospitals_chart = alt.Chart(unique_hospitals).mark_bar().encode(
    x=alt.X("Year:O", title="Year"),
    y=alt.Y("Number of Unique Hospitals:Q",
           title="Number of Unique Hospitals"),
    tooltip=["Year", "Number of Unique Hospitals"]
).properties(
    title="Number of Unique Hospitals Per Year"
)

unique_hospitals_chart.show()

print("Observations Per Year:")
print(combined_year_df)
print("\nUnique Hospitals Per Year:")
print(unique_hospitals)

```

alt.Chart(...)

	Year	Number of Observations
0	2016	7245

1	2017	7260
2	2018	7277
3	2019	7303

Unique Hospitals Per Year:

	Year	Number of Unique Hospitals
0	2016	7245
1	2017	7260
2	2018	7277
3	2019	7303

b.

Since the number of observations equals the number of unique hospitals, it suggests that the data is clean and does not contain repeated entries for hospitals within the same year. Each hospital appears only once per year, simplifying analysis.

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

- Termination code equal to 00=ACTIVE PROVIDER. The data contain only up to the code 07. The other codes apply to CLIA.

```
# Ensure the program termination code is treated as a string for comparison
combined_df["PGM_TRMNTN_CD"] = combined_df["PGM_TRMNTN_CD"].astype(str)

# List of codes indicating inactive or terminated facilities
inactive_codes = ["01", "02", "03", "04", "05", "06", "07"]

# Filter for active hospitals in 2016
active_2016 = combined_df[(combined_df["Year"] == "2016") & (
    combined_df["PGM_TRMNTN_CD"] == "00")]

# List to collect suspected hospital closures
suspected_closures = []

# Iterate through each active hospital in 2016
for idx, hospital in active_2016.iterrows():
    provider_num = hospital["PRVDR_NUM"]
    facility_name = hospital["FAC_NAME"]
    zip_code = hospital["ZIP_CD"]

    # Check subsequent years to see if the hospital has closed
```

```

for year in ["2017", "2018", "2019"]:
    yearly_data = combined_df[(combined_df["PRVDR_NUM"] == provider_num)
    & (
        combined_df["Year"] == year)]

    # If the hospital record is missing or marked as inactive, add to
    # suspected closures
    if yearly_data.empty or
       yearly_data["PGM_TRMNTN_CD"].isin(inactive_codes).any():
        suspected_closures.append({
            "Provider Number": provider_num,
            "Facility Name": facility_name,
            "ZIP Code": zip_code,
            "Year Closed": year # Record the year in which closure is
            # suspected
        })
    break # Exit the loop once a closure is detected

# Create a DataFrame for the suspected closures
suspected_closures_df = pd.DataFrame(suspected_closures)

# Count the number of suspected closures
num_closures = len(suspected_closures_df)

# Display the total number of suspected hospital closures
display(f"Total suspected hospital closures: {num_closures}")

```

'Total suspected hospital closures: 174'

2.

```

# Sort the suspected closures DataFrame by the "Facility Name" column
sorted_closures = suspected_closures_df.sort_values(by="Facility Name")

# Select the top 10 records and display only "Facility Name" and "Year
# Closed" columns
top_10_closures = sorted_closures[["Facility Name", "Year Closed"]].head(10)

# Display the top 10 sorted closures
display(top_10_closures)

```

	Facility Name	Year Closed
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.

```
# Step 1: Count the number of active hospitals (PGM_TRMNTN_CD == "00") by
# ZIP_CD for each year
active_hospitals_by_year = (
    combined_df[combined_df['PGM_TRMNTN_CD'] == "00"]
    .groupby(['ZIP_CD', 'Year'])
    .size()
    .unstack(fill_value=0)
)

# Step 2: Create columns with differences between years for each ZIP_CD
# Calculate the differences between consecutive years
active_hospitals_by_year['diff_2017_2016'] = active_hospitals_by_year["2017"]
    - \
    active_hospitals_by_year["2016"]
active_hospitals_by_year['diff_2018_2017'] = active_hospitals_by_year["2018"]
    - \
    active_hospitals_by_year["2017"]
active_hospitals_by_year['diff_2019_2018'] = active_hospitals_by_year["2019"]
    - \
    active_hospitals_by_year["2018"]

# Reset index to prepare for merging
active_hospitals_by_year = active_hospitals_by_year.reset_index()

# Step 3: Merge the active_hospitals_by_year dataframe with
# suspected_closures by ZIP Code
merged_df = pd.merge(suspected_closures_df, active_hospitals_by_year,
```

```

        left_on='ZIP Code', right_on='ZIP_CD', how='left')

# Initialize the closure_difference column with NaN
merged_df['closure_difference'] = np.nan

# Loop through each row and set closure_difference based on Year Closed
for index, row in merged_df.iterrows():
    if row['Year Closed'] == "2017":
        merged_df.at[index, 'closure_difference'] = row['diff_2018_2017']
    elif row['Year Closed'] == "2018":
        merged_df.at[index, 'closure_difference'] = row['diff_2019_2018']
    else:
        # Optionally set it to a specific value if Year Closed doesn't match
        # any condition
        merged_df.at[index, 'closure_difference'] = None

# Step 5: Create a dummy variable that is 1 if the closure_difference is 0 or
# greater, else 0
merged_df['closure_dummy'] = merged_df['closure_difference'].apply(
    lambda x: 1 if pd.notna(x) and x > 0 else (0 if pd.notna(x) else np.nan)
)

# Count the number of rows where closure_dummy is 1
count_of_ones = merged_df['closure_dummy'].sum()
display(f"Number of rows with closure_dummy = 1: {count_of_ones}")

# Count the number of rows where closure_dummy is 0
count_of_zeros = (merged_df['closure_dummy'] == 0).sum()
display(f"Number of rows with closure_dummy = 0: {count_of_zeros}")

'Number of rows with closure_dummy = 1: 4.0'

'Number of rows with closure_dummy = 0: 94'

# Question a: Count hospitals that fit the definition of potentially being a
# merger/acquisition
count_of_ones = merged_df['closure_dummy'].sum()
display(f"Number of hospitals potentially being a merger/acquisition:
        {count_of_ones}")

# Question b: Count hospitals left after correcting for potential mergers
corrected_closures = merged_df[merged_df['closure_dummy'] == 0]

```

```

count_of_zeros = (merged_df['closure_dummy'] == 0).sum()
display(f"Number of hospitals left after correcting for potential mergers:
↪ {count_of_zeros}")

# Question c: Sort the corrected list of closures by hospital name and
↪ display the first 10 rows
sorted_corrected_closures = corrected_closures.sort_values(by='Facility
↪ Name').head(10)
display(sorted_corrected_closures[['Facility Name', 'ZIP Code', 'Year
↪ Closed', 'closure_difference']])

```

'Number of hospitals potentially being a merger/acquisition: 4.0'

'Number of hospitals left after correcting for potential mergers: 94'

	Facility Name	ZIP Code	Year Closed	closure_d
4	ABRAZO MARYVALE CAMPUS	85031	2017	0.0
10	ADVENTIST MEDICAL CENTER - CENTRAL VALLEY	93230	2017	0.0
97	AFFINITY MEDICAL CENTER	44646	2018	0.0
80	ALBANY MEDICAL CENTER / SOUTH CLINICAL CAMPUS	12208	2017	0.0
140	ALLEGIANCE SPECIALTY HOSPITAL OF KILGORE	75662	2017	0.0
21	ARKANSAS VALLEY REGIONAL MEDICAL CENTER	81050	2017	0.0
69	BANNER CHURCHILL COMMUNITY HOSPITAL	89406	2017	0.0
5	BANNER PAYSON MEDICAL CENTER	85541	2018	0.0
170	BAY AREA REGIONAL MEDICAL CENTER, LLC	77598	2018	0.0
137	BAYLOR SCOTT & WHITE MEDICAL CENTER GARLAND	75042	2018	0.0

Download Census zip code shapefile (10 pt)

1. a. Five File Types and Their Contents
2. **.shp (Shapefile)**: Stores the geometry of the feature, such as points, lines, or polygons representing geographic boundaries (e.g., ZIP code tabulation areas) (6.4 MB).
3. **.shx (Shape Index)**: Contains an index of the feature geometry, providing quick access to spatial data in the .shp file (165 bytes).
4. **.dbf (Database File)**: Stores attribute data associated with each shape, such as names, identifiers, and other relevant information (837.5 MB).
5. **.prj (Projection File)**: Contains projection information, specifying the coordinate system and projection details for the spatial data (265 KB).

6. **.xml (Metadata File)**: Provides metadata, including details about the data's origin, purpose, spatial extent, and additional descriptive information (16 KB).

b. File Sizes After Unzipping

The sizes of the files were included above. However, to determine the size of each file after unzipping, use the following command in the terminal of the folder:

```
total 1648944 -rw-r-r-@ 1 aosilva staff 6.1M Sep 14 2011 gz_2010_us_860_00_500k.dbf
-rw-r-r-@ 1 aosilva staff 165B Sep 14 2011 gz_2010_us_860_00_500k.prj
-rw-r-r-@ 1 aosilva staff 799M Sep 14 2011 gz_2010_us_860_00_500k.shp
-rw-r-r-@ 1 aosilva staff 259K Sep 14 2011 gz_2010_us_860_00_500k.shx
-rw-rw-rw-@ 1 aosilva staff 15K Dec 9 2011 gz_2010_us_860_00_500k
```

2.

```
import geopandas as gpd

# Load the shapefile (replace 'path_to_shapefile' with the actual path)
zip_shapefile =
    ↵ gpd.read_file('gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.shp')

# Filter for Texas ZIP codes (750-799)
texas_zip_shapefile =
    ↵ zip_shapefile[zip_shapefile['ZCTA5'].str.startswith(tuple(map(str,
    ↵ range(750, 799))))]
    ↵

# Check the result
texas_zip_shapefile.head()

# Filter combined_df for 2016 and Texas ZIP codes
hospitals_2016 = combined_df[(combined_df['Year'] == "2016") &
    ↵ (combined_df['ZIP_CD'].str.startswith(tuple(map(str, range(750, 799)))))

# Count hospitals per ZIP code
hospitals_per_zip =
    ↵ hospitals_2016.groupby('ZIP_CD').size().reset_index(name='hospital_count')

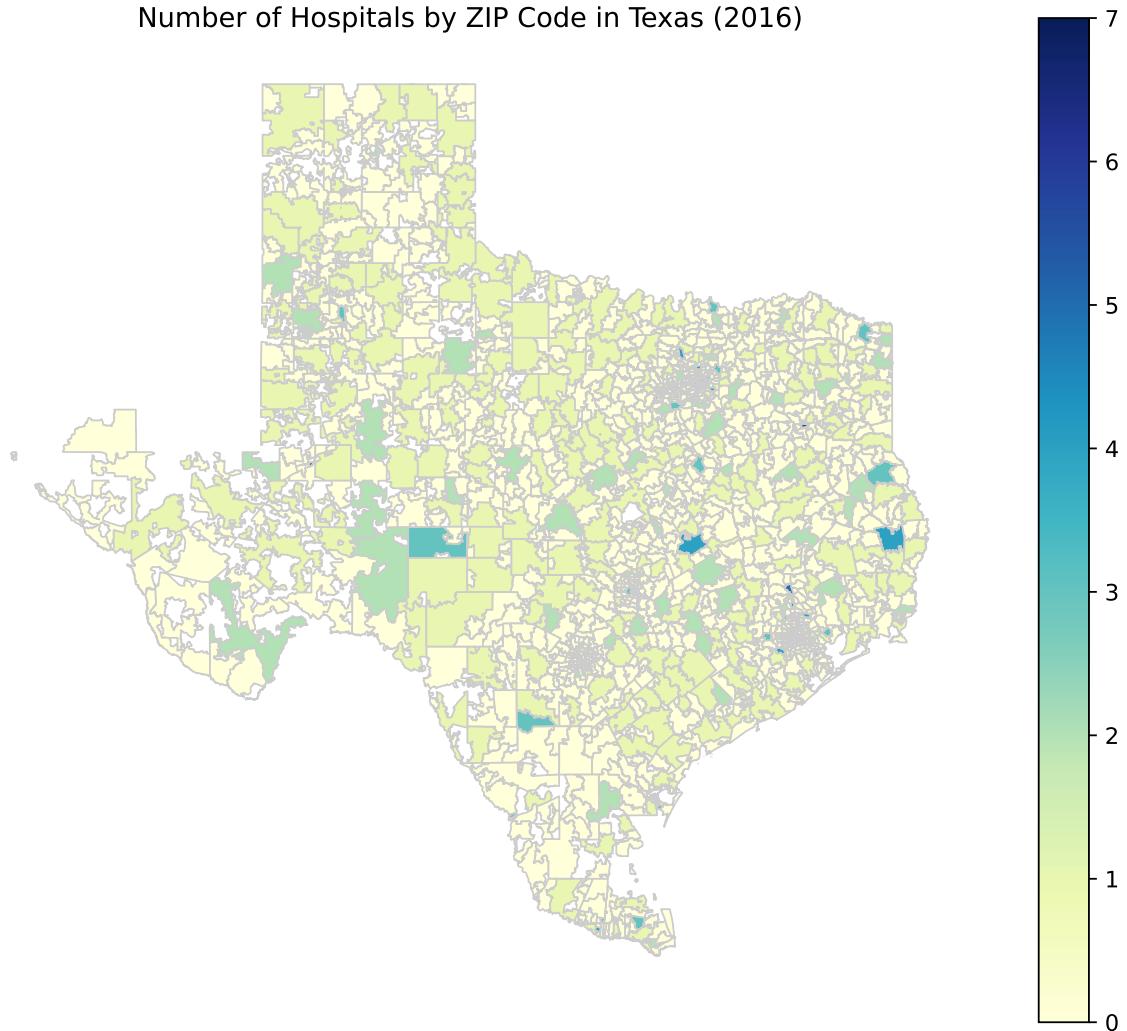
# Merge shapefile with hospital counts
texas_zip_map = texas_zip_shapefile.merge(hospitals_per_zip, left_on='ZCTA5',
    ↵ right_on='ZIP_CD', how='left')
```

```
# Fill NaN values with 0 for ZIP codes with no hospitals
texas_zip_map['hospital_count'] = texas_zip_map['hospital_count'].fillna(0)

import matplotlib.pyplot as plt

# Plot the choropleth
fig, ax = plt.subplots(1, 1, figsize=(10, 8))
texas_zip_map.plot(column='hospital_count', cmap='YlGnBu', linewidth=0.8,
                    ax=ax, edgecolor='0.8', legend=True)
ax.set_title('Number of Hospitals by ZIP Code in Texas (2016)')
ax.axis('off')
plt.show()
```

Number of Hospitals by ZIP Code in Texas (2016)



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

1.

The GeoDataFrame, `zips_all_centroids`, has dimensions that correspond to the number of unique ZIP code areas, with two main columns: `ZIP Code` and `geometry`. The `ZIP Code` column holds the specific ZIP Code Tabulation Area (ZCTA) for each entry, representing each distinct postal region. The `geometry` column contains the centroid point of each ZIP code area, with each point indicating the latitude and longitude coordinates for the center of the ZIP code region. This structure allows for further geographic analysis, such as calculating distances to the nearest hospital.

```

import geopandas as gpd

# Calculate centroids for each ZIP code and create a new GeoDataFrame with
# these centroids
zip_shapefile['centroid'] = zip_shapefile.geometry.centroid

# Convert this information into a new GeoDataFrame, focusing only on the
# centroids and ZIP code info
zips_all_centroids = gpd.GeoDataFrame(zip_shapefile[['ZCTA5', 'centroid']],
                                       geometry='centroid')

# Rename columns for clarity
zips_all_centroids.columns = ['ZIP Code', 'geometry']

# Display the dimensions and column descriptions
print("Dimensions of zips_all_centroids:", zips_all_centroids.shape)
print("Columns in zips_all_centroids:", zips_all_centroids.columns)

# Display the first few rows to verify the output
zips_all_centroids.head()

```

C:\Users\aosil\AppData\Local\Temp\ipykernel_48896\998579674.py:4:
 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.

```

zip_shapefile['centroid'] = zip_shapefile.geometry.centroid

Dimensions of zips_all_centroids: (33120, 2)
Columns in zips_all_centroids: Index(['ZIP Code', 'geometry'],
                                     dtype='object')

```

	ZIP Code	geometry
0	01040	POINT (-72.64107 42.21257)
1	01050	POINT (-72.86985 42.28786)
2	01053	POINT (-72.71162 42.35349)
3	01056	POINT (-72.45805 42.19215)
4	01057	POINT (-72.3243 42.09165)

2.

```

# Define ZIP code prefixes for Texas and its neighboring states
texas_prefixes = list(range(750, 799))
bordering_prefixes = texas_prefixes + [70, 71, 72, 73, 74, 87, 88]

# Convert prefixes to strings for matching with the 'ZIP Code' column
texas_prefixes_str = tuple(map(str, texas_prefixes))
bordering_prefixes_str = tuple(map(str, bordering_prefixes))

# Filter the zips_all_centroids GeoDataFrame for Texas ZIP codes
zips_texas_centroids = zips_all_centroids[
    zips_all_centroids['ZIP Code'].str.startswith(texas_prefixes_str)
]

# Filter the zips_all_centroids GeoDataFrame for Texas and bordering states
# ZIP codes
zips_texas_borderstates_centroids = zips_all_centroids[
    zips_all_centroids['ZIP Code'].str.startswith(bordering_prefixes_str)
]

# Print out the number of unique ZIP codes in each subset
num_texas_zips = zips_texas_centroids['ZIP Code'].nunique()
num_bordering_zips = zips_texas_borderstates_centroids['ZIP Code'].nunique()

print("Unique ZIP codes in Texas subset:", num_texas_zips)
print("Unique ZIP codes in Texas and bordering states subset:",
      num_bordering_zips)

```

Unique ZIP codes in Texas subset: 1910
 Unique ZIP codes in Texas and bordering states subset: 4032

3.

We used an inner join to retain only the ZIP codes in Texas and neighboring states that have at least one hospital, ensuring that `zips_withhospital_centroids` includes only relevant ZIP codes. The join was conducted on the ZIP Code column from `zips_texas_borderstates_centroids` and `ZIP_CD` from `zips_with_hospital`, allowing us to filter specifically for ZIP codes that had at least one active hospital in 2016. The result is a GeoDataFrame, `zips_withhospital_centroids`, which contains only the ZIP codes in Texas or bordering states with at least one hospital in that year.

```
# Step 1: Filter hospitals active in 2016 and count by ZIP
```

```

hospitals_per_zip = combined_df[(combined_df['Year'] == "2016") & (
    combined_df['PGM_TRMNTN_CD'] ==
    "00")].groupby('ZIP_CD').size().reset_index(name='hospital_count')

# Filter to keep only ZIP codes with at least one hospital
zips_with_hospital = hospitals_per_zip[hospitals_per_zip['hospital_count'] >
    0]

# Step 2: Merge this count data with zips_texas_borderstates_centroids on ZIP
# code
zips_texas_borderstates_centroids['ZIP Code'] =
    zips_texas_borderstates_centroids['ZIP Code'].astype(
        str)
zips_with_hospital['ZIP_CD'] = zips_with_hospital['ZIP_CD'].astype(str)

# Inner join to keep only ZIP codes with hospitals in 2016
zips_withhospital_centroids = zips_texas_borderstates_centroids.merge(
    zips_with_hospital, left_on='ZIP Code', right_on='ZIP_CD', how='inner'
)

# Display
display(zips_withhospital_centroids.head())

```

C:\Users\aosil\AppData\Local\Programs\Python\Python312\Lib\site-packages\geopandas\geodataframe.py:103: 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-copy
super().__setitem__(key, value)

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

4. a.

```

from shapely.ops import nearest_points
import time

# The geometry column is set as active in both GeoDataFrames
zips_texas_centroids = zips_texas_centroids.set_geometry('geometry')
zips_withhospital_centroids =
    zips_withhospital_centroids.set_geometry('geometry')

# Subset to 10 ZIP codes from zips_texas_centroids for testing
test_zips_texas = zips_texas_centroids.head(10)

# Start timer
start_time = time.time()

# Calculate distances for each ZIP code in test subset
distances = []
for idx, row in test_zips_texas.iterrows():
    # Calculate the distance to the nearest hospital ZIP code
    nearest_hospital =
        zips_withhospital_centroids.distance(row.geometry).min()
    distances.append(nearest_hospital)

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

# Output the time taken and estimated total time for the full dataset
print("Time taken for 10 ZIP codes:", elapsed_time, "seconds")

# Estimate total time
total_rows = len(zips_texas_centroids)
estimated_total_time = (elapsed_time / 10) * total_rows
print("Estimated time for entire dataset:", estimated_total_time, "seconds")

```

Time taken for 10 ZIP codes: 0.11382412910461426 seconds
Estimated time for entire dataset: 21.740408658981323 seconds

C:\Users\aosil\AppData\Local\Temp\ipykernel_48896\1307856507.py:18:
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.

```
nearest_hospital = zips_withhospital_centroids.distance(row.geometry).min()
```

b.

The full dataset calculation completed significantly faster than initially estimated, suggesting that the small sample used for estimation might have over-predicted the time required. This difference could be due to optimizations in the spatial library that enhance performance with larger datasets, as well as variations in system performance between runs. Small sample estimates can sometimes lead to overestimations when the operation scales more efficiently across a larger volume of data.

```
import time

# Start timer for the full calculation
start_time_full = time.time()

# Calculate distances for each ZIP code in the entire Texas subset
full_distances = []
for idx, row in zips_texas_centroids.iterrows():
    # Calculate the distance to the nearest hospital ZIP code
    nearest_hospital_distance = zips_withhospital_centroids.distance(
        row.geometry).min()
    full_distances.append(nearest_hospital_distance)

# End timer
end_time_full = time.time()
elapsed_time_full = end_time_full - start_time_full

# Output the actual time taken
print("Actual time taken for full dataset:", elapsed_time_full, "seconds")

# Compare to estimated time from previous step
estimated_total_time = (elapsed_time / 10) * len(zips_texas_centroids)
print("Estimated time for entire dataset:", estimated_total_time, "seconds")
print("Difference between actual and estimated time:", abs(
    elapsed_time_full - estimated_total_time), "seconds")
```

```
C:\Users\aosil\AppData\Local\Temp\ipykernel_48896\714762939.py:10:
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.
```

```
nearest_hospital_distance = zips_withhospital_centroids.distance(
```

```

Actual time taken for full dataset: 22.815343141555786 seconds
Estimated time for entire dataset: 21.740408658981323 seconds
Difference between actual and estimated time: 1.074934482574463 seconds

```

c.

In the .prj file , the unit is degrees. This indicates that the geographic data is in a coordinate system based on latitude and longitude, with units in angular degrees rather than linear distance. To convert degrees to miles, we can use an approximate conversion. Each degree of latitude corresponds to about 69 miles, a consistent value because latitude lines are parallel. For longitude, the conversion varies depending on latitude due to the Earth's curvature; at the equator, 1 degree of longitude equals approximately 69 miles, but this distance decreases as you move toward the poles. A common formula to estimate the distance per degree of longitude at a given latitude L is $69 \times \cos(L)$.

Effects of closures on access in Texas (15 pts)

1.

```

# Filter for Texas ZIP codes in the range 75000 to 79999 and for closures
# between 2016 and 2019
texas_closures = corrected_closures[
    (corrected_closures['ZIP Code'].astype(str).str.startswith(tuple(map(str,
# range(750, 799)))))) &
    (corrected_closures['Year Closed'].between("2016", "2019")))
]

# Group by ZIP Code and count the number of closures for each ZIP code
closures_by_zipcode = texas_closures.groupby(
    'ZIP Code').size().reset_index(name='Number of Closures')

closures_by_zipcode

```

	ZIP Code	Number of Closures
0	75042	1
1	75231	1
2	75662	1
3	75862	1
4	76502	1
5	77035	1
6	77054	1

	ZIP Code	Number of Closures
7	77429	1
8	77479	1
9	77598	1
10	78017	1
11	78061	1
12	785	1
13	78734	1
14	78834	1
15	79553	1
16	79735	1
17	79761	1

2.

```

import geopandas as gpd
import matplotlib.pyplot as plt

# Load the ZIP code shapefile
zip_shapefile =
    ↪ gpd.read_file('gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.shp')

# Filter for Texas ZIP codes (ZIP codes starting with 750-799)
zip_shapefile =
    ↪ zip_shapefile[zip_shapefile['ZCTA5'].str.startswith(tuple(map(str,
    ↪ range(750, 799)))]]

# Ensure ZIP code columns are strings for matching
zip_shapefile['ZCTA5'] = zip_shapefile['ZCTA5'].astype(str)
closures_by_zipcode['ZIP Code'] = closures_by_zipcode['ZIP Code'].astype(str)

# Merge the shapefile with closures data on ZIP code
texas_zip_closures = zip_shapefile.merge(closures_by_zipcode,
    ↪ left_on='ZCTA5', right_on='ZIP Code', how='left')

# Replace NaN values in 'Number of Closures' with 0 for ZIP codes without
    ↪ closures
texas_zip_closures['Number of Closures'] = texas_zip_closures['Number of
    ↪ Closures'].fillna(0)

# Count the number of directly affected ZIP codes (where Number of Closures >
    ↪ 0)

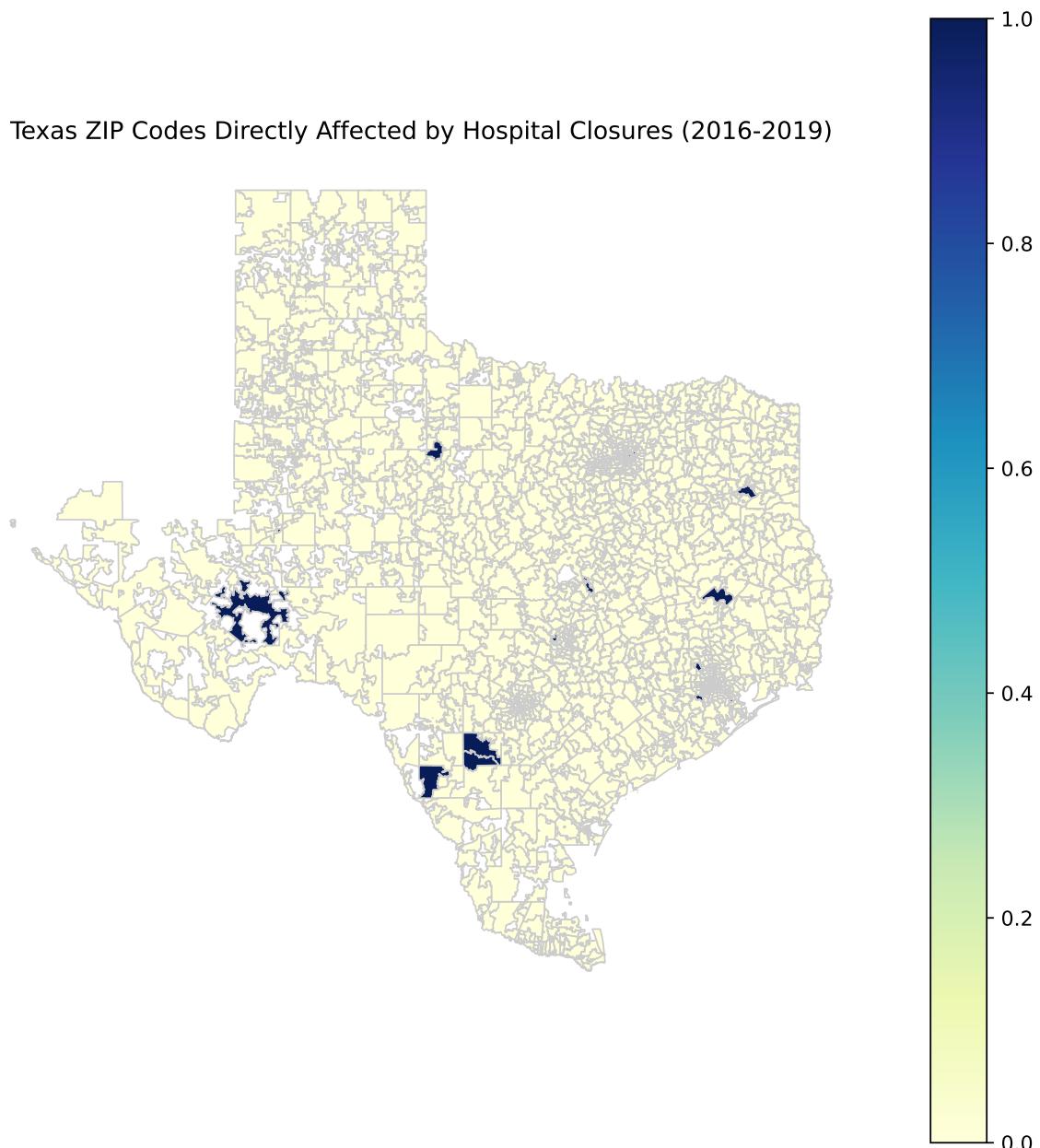
```

```
affected_zip_count = texas_zip_closures[texas_zip_closures['Number of
↪ Closures'] > 0].shape[0]
print(f"Number of directly affected ZIP codes in Texas:
↪ {affected_zip_count}")

# Plot the choropleth map
fig, ax = plt.subplots(1, 1, figsize=(10, 10))
texas_zip_closures.plot(column='Number of Closures', cmap='YlGnBu',
↪ linewidth=0.8, ax=ax, edgecolor='0.8', legend=True)
ax.set_title('Texas ZIP Codes Directly Affected by Hospital Closures
↪ (2016-2019)')
ax.set_axis_off()
plt.show()
```

Number of directly affected ZIP codes in Texas: 17

Texas ZIP Codes Directly Affected by Hospital Closures (2016-2019)



3.

```
import geopandas as gpd
from shapely.geometry import Point
import matplotlib.pyplot as plt
```

```

# Step 1: Filter for directly affected ZIP codes
directly_affected_zips = texas_zip_closures[texas_zip_closures['Number of
    ↵ Closures'] > 0]

# Convert Texas ZIP code coordinates to the appropriate projection for
    ↵ distance calculations .Project the GeoDataFrame to a projection that uses
    ↵ meters (such as EPSG: 3395 or EPSG: 3857)
zip_shapefile = zip_shapefile.to_crs(epsg=3395)
directly_affected_zips = directly_affected_zips.to_crs(epsg=3395)

# Step 2: Create a 10-mile (approx. 16093.4 m) buffer around each directly
    ↵ affected ZIP code
directly_affected_zips['buffer'] =
    ↵ directly_affected_zips.geometry.buffer(16093.4)

# Step 3: Create a GeoDataFrame containing only the buffers and set it to use
    ↵ the 'buffer' column as geometry
buffers_gdf = gpd.GeoDataFrame(directly_affected_zips, geometry='buffer')

# Step 4: Spatial join with the overall Texas ZIP code shapefile to find ZIP
    ↵ codes within the 10-mile buffer
indirectly_affected_zips = gpd.sjoin(zip_shapefile, buffers_gdf, how="inner",
    ↵ predicate="intersects")

# Step 5: Count the number of unique indirectly affected ZIP codes
indirectly_affected_zip_count =
    ↵ indirectly_affected_zips['ZCTA5_left'].nunique()

print(f"Number of indirectly affected ZIP codes in Texas:
    ↵ {indirectly_affected_zip_count}")

```

Number of indirectly affected ZIP codes in Texas: 359

Number of indirectly affected ZIP codes in Texas: 359

4.

```

import geopandas as gpd
import matplotlib.pyplot as plt

# Step 1: Start with all Texas ZIP codes
texas_zip_codes = zip_shapefile.to_crs(buffers_gdf.crs)

```

```

# Step 2: Create the categories
texas_zip_codes['Category'] = 'Not Affected'

texas_zip_codes.loc[texas_zip_codes['ZCTA5'].isin(
    directly_affected_zips['ZCTA5']), 'Category'] = 'Directly Affected'

texas_zip_codes.loc[
    (texas_zip_codes['ZCTA5'].isin(indirectly_affected_zips['ZCTA5_left'])) &
    (texas_zip_codes['Category'] != 'Directly Affected'), 'Category'
] = 'Indirectly Affected'

# Step 3: Plot the choropleth map
fig, ax = plt.subplots(1, 1, figsize=(10, 10))

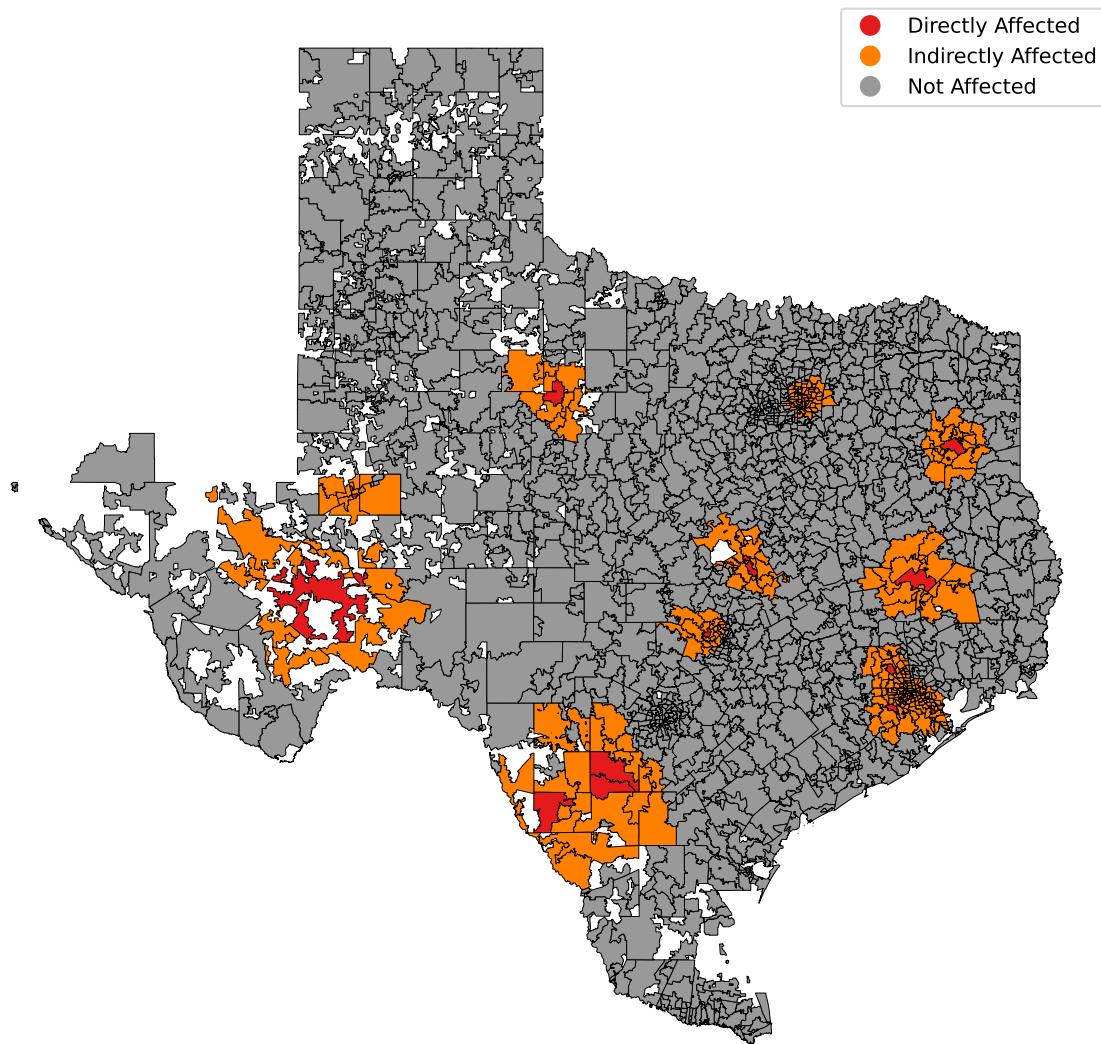
category_colors = {'Not Affected': 'lightgrey',
                   'Directly Affected': 'red', 'Indirectly Affected':
                       'orange'}

texas_zip_codes.plot(column='Category', cmap='Set1', legend=True, ax=ax,
                     edgecolor='black', linewidth=0.2)

ax.set_title('Impact of Hospital Closures on Texas ZIP Codes (2016-2019)')
ax.set_axis_off()
plt.show()

```

Impact of Hospital Closures on Texas ZIP Codes (2016-2019)



5. a. The distances are converted from degrees to miles by multiplying 69 times.

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

# Calculate the distance to the nearest hospital for each ZIP code in Texas
zips_texas_centroids['nearest_hospital_distance'] =
    zips_texas_centroids.geometry.apply(
        lambda x: zips_withhospital_centroids.distance(x).min())
```

```

)

# Step 2: Calculate the average distance (in degrees)
avg_dist_degrees = zips_texas_centroids['nearest_hospital_distance'].mean()

# Step 3: Convert average distance from degrees to miles
avg_dist_miles = avg_dist_degrees * 69 # 1 degree 69 miles

print("Average distance to the nearest hospital (in degrees):",
      avg_dist_degrees)
print("Average distance to the nearest hospital (in miles):", avg_dist_miles)

```

C:\Users\aosil\AppData\Local\Temp\ipykernel_48896\3339965745.py:6:
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.

```

lambda x: zips_withhospital_centroids.distance(x).min()

Average distance to the nearest hospital (in degrees): 0.21460584693093776
Average distance to the nearest hospital (in miles): 14.807803438234705

```

An average distance of around 15 miles aligns with expectations, especially given Texas's mix of densely populated cities and sparsely populated rural regions.

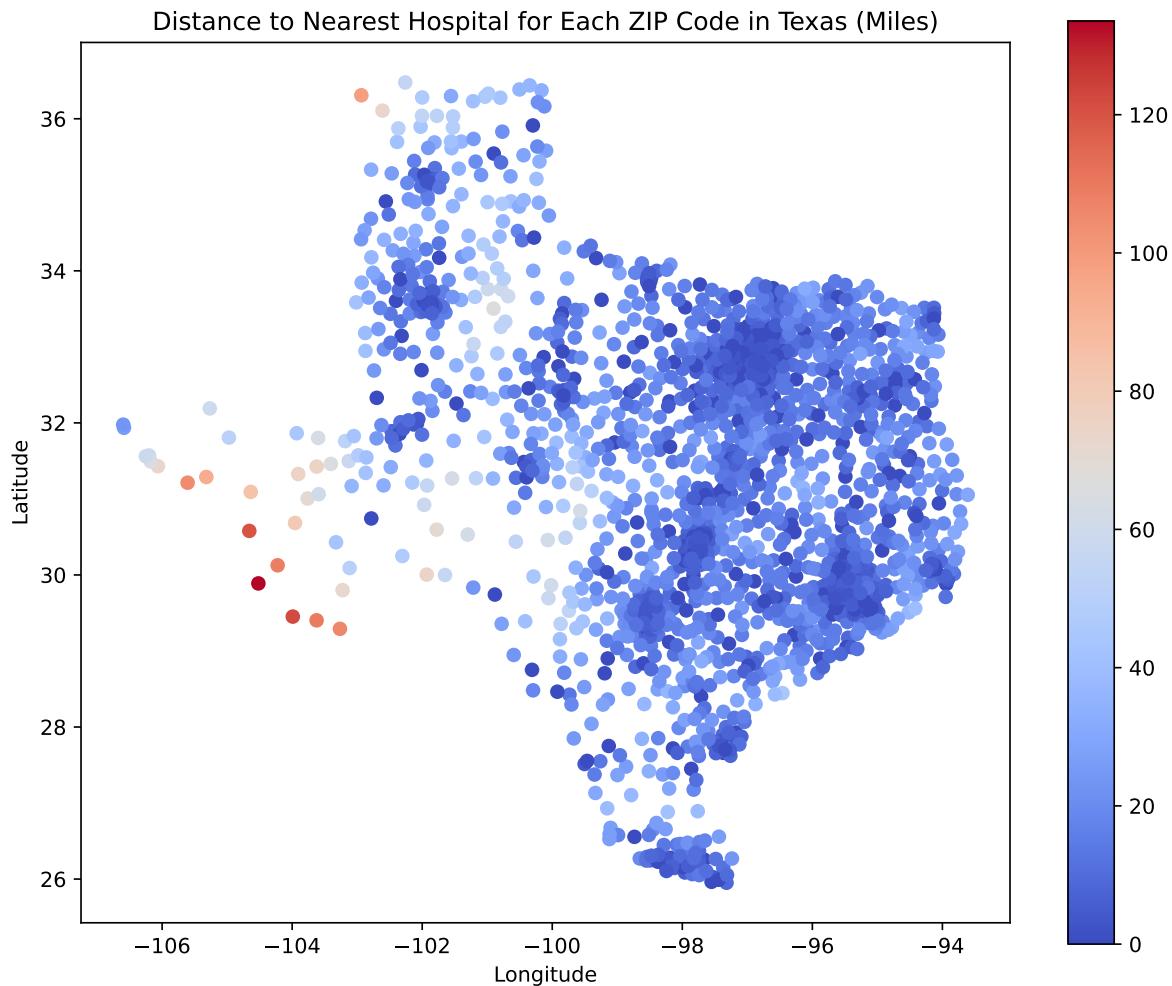
b.

```

# Step 4: Map the value for each ZIP code in Texas
zips_texas_centroids['nearest_hospital_distance_miles'] =
    zips_texas_centroids['nearest_hospital_distance'] * 69

fig, ax = plt.subplots(1, 1, figsize=(10, 8))
zips_texas_centroids.plot(column='nearest_hospital_distance_miles',
    cmap='coolwarm', legend=True, ax=ax)
ax.set_title("Distance to Nearest Hospital for Each ZIP Code in Texas
    (Miles)")
ax.set_xlabel("Longitude")
ax.set_ylabel("Latitude")
plt.show()

```



The choropleth map visually represents the distance to the nearest hospital for each ZIP code in Texas. This map highlights the healthcare access disparity between urban and rural regions in Texas. Rural areas, especially in western Texas, may face challenges in timely access to healthcare due to the greater distances to hospitals, which aligns with the calculated average distance.

Reflecting on the exercise (10 pts)

1.

The “first-pass” method we used to identify hospital closures based on termination codes and date ranges has several limitations that could lead to inaccuracies. One issue is that data reporting may be delayed or incomplete, meaning closures might not appear immediately or could be misrepresented if a hospital has only temporarily closed. Changes in ownership,

mergers, or rebranding may also trigger termination codes without representing an actual closure, which could result in false positives. Additionally, temporary closures, such as those for renovations or emergencies, may appear as permanent in the dataset if not properly labeled. Data inconsistencies, such as incomplete or varying codes, further complicate closure identification and could lead to missed closures or errors. Finally, using only ZIP codes to identify closures doesn't account for nearby hospitals that may still serve the affected area, thus misrepresenting the impact. To improve accuracy, we could cross-reference with external sources like health department records or news reports, validate closure status over multiple years, and incorporate additional details such as ownership changes, closure duration, and geographic proximity to other hospitals.

2.

The current approach of identifying ZIP codes affected by hospital closures, based on direct closures within a ZIP code or proximity within a 10-mile radius, provides a basic assessment of hospital access changes. However, this method has limitations. For example, it assumes that a 10-mile distance is equally impactful across all areas, which may not hold true. In rural regions, a 10-mile distance may indeed indicate a significant gap in access, but in urban areas, other hospitals within a short distance may still provide adequate coverage, making the closure less impactful. This method also doesn't consider population density or specific transportation factors that influence actual access to healthcare services.

To improve this measure, one approach could be to adjust the buffer distance based on urban or rural classification, using a smaller radius for urban areas and a larger one for rural ones. Additionally, measuring travel time or distance based on road networks rather than a simple radius would give a more accurate reflection of access. Another refinement could involve considering hospital capacity, as some hospitals may not be able to handle increased demand from nearby closures, further affecting access.