ps4

PS4: Due Sat Nov 2 at 5:00PM Central. Worth 100 points. ### Academic integrity statement We checked Googled commands/ways to go about getting the output needed. I have included the links where I learned from. Additionally, I used ChatGPT for help with debugging double checking the flow of my code. I also referred to code from last quarter's python class for grouping, datetime, dictionary, length, float, round, print,a nd reset index functinos. ALso checked https://stackoverflow.com/questions/tagged/geospatial for specific codign questions.

1. "This submission is my work alone and complies with the 30538 integrity policy." **CT EA**
2. "I have uploaded the names of anyone I worked with on the problem set **here**" **CT EA** (1 point)
3. Late coins used this pset: **3** Late coins left after submission: **1**

- Partner 1 (cmtee):
  - Partner 2 (eandujar): ## Style Points (10 pts) ## Submission Steps (10 pts)

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

```
import pandas as pd
import os
import matplotlib.pyplot as plt
import geopandas as gpd
import time
```

1. I pulled

FAC_NAME STATE_CD PRVDR_CTGRY_CD PRVDR_CTGRY_SBTYP_CD ZIP-CD PGM_TRMNTN_CD PGM_TRMNTN_CD 2.

```
# Load the 2016 dataset
df = pd.read_csv(
    'C:/Users/clari/OneDrive/Documents/Python II/pos_files/pos2016.csv')
# change the path as needed
```

```
# Filter for short-term hospitals (provider type code 01 and subtype code 01)
st_hospitals_2016 = df[(df['PRVDR_CTGRY_CD'] == 1) &
                       (df['PRVDR_CTGRY_SBTYP_CD'] == 1)]
print(len(st_hospitals_2016))
```

7245

a.There are 7,245 reported in this data.This number doesn't make sense because there seem to be too many short-term hospitals.
b. According to AHA, there were only 4,840 in 2016, so this number doesn't make sense. This could be due to differences in
definitions, inclusion criteria, and scope. Maybe this raw dataset has a broader range of facilities certified for Medicare/Medicaid billing, which increases the cout. Maybe the includsion of subtypes increased our count too.

https://www.aha.org/system/files/2018-02/2018-aha-hospital-fast-facts_0.pdf

3.

```
# Define the path to your folder containing the POS files
folder_path = r'C:/Users/clari/OneDrive/Documents/Python II/pos_files'
# path for eddie: r'C:\Users\eddie\Downloads\pos2016.csv'

# List all files in the directory to check if they exist
print("Files in directory:")
for file in os.listdir(folder_path):
    print(file)

# Define a function to load CSV files with different encodings and filter for
 ↪ short-term hospitals
def load_and_filter_csv(file_name, encodings, year):
    for encoding in encodings:
        try:
            # Try reading the file with a specified encoding
            df = pd.read_csv(os.path.join(folder_path, file_name),
                             encoding=encoding, engine='python')
```

```
            print(f'Successfully loaded {file_name} with encoding
             ↪ {encoding}.')

            # Filter for short-term hospitals (provider type code 01 and
             ↪ subtype code 01)
            st_hospitals = df[(df['PRVDR_CTGRY_CD'] == 1) &
                              (df['PRVDR_CTGRY_SBTYP_CD'] == 1)]
            # Add a 'Year' column to the DataFrame
            st_hospitals['Year'] = year

            return st_hospitals  # Return the filtered DataFrame if loaded
             ↪ successfully
        except Exception as e:
            print(f'Error loading {file_name} with encoding {encoding}: {e}')
    return None  # Return None if all attempts fail


# List of encodings to try
encodings = ['utf-8', 'ISO-8859-1', 'latin1', 'utf-8-sig', 'cp1252']

# List of file names and corresponding years
files_with_years = [('pos2016.csv', 2016), ('pos2017.csv', 2017),
                    ('pos2018.csv', 2018), ('pos2019.csv', 2019)]

# Load datasets for all years and filter for short-term hospitals
appended_dfs = []
for file_name, year in files_with_years:
    df = load_and_filter_csv(file_name, encodings, year)
    if df is not None:
        appended_dfs.append(df)

# Check if we successfully loaded and filtered data for all years
if len(appended_dfs) == len(files_with_years):
    # Append all filtered DataFrames together into one DataFrame
    combined_df = pd.concat(appended_dfs, ignore_index=True)
    print("Successfully appended all datasets.")

    # Save the combined DataFrame to a CSV file named 'pos_6789.csv'
    output_file = os.path.join(folder_path, 'pos_6789.csv')
    combined_df.to_csv(output_file, index=False)
    print(f"Combined dataset saved as {output_file}")
else:
```

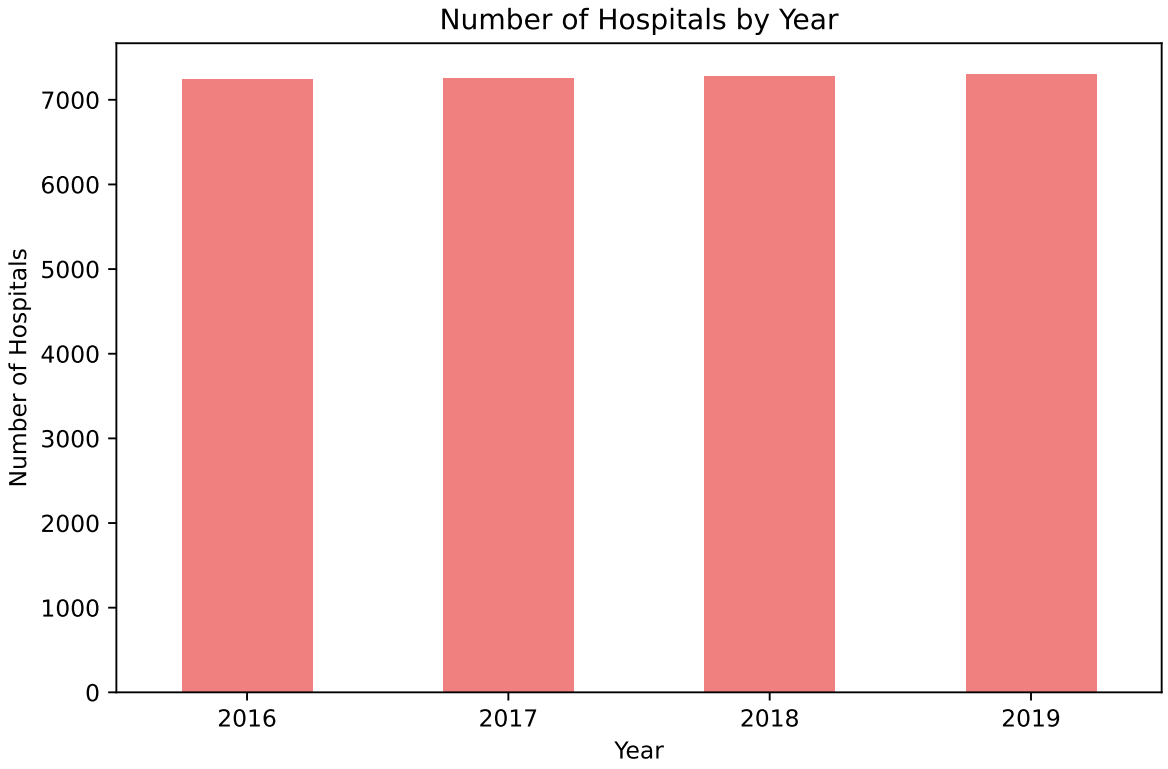
```
    print("Some files could not be loaded or filtered.")

# Optionally, print the first few rows of the combined DataFrame to verify
if len(appended_dfs) > 0:
    print(combined_df.head())
```

```
Files in directory:
corrected_suspected_hospital_closures_2016_2018.csv
corrected_suspected_hospital_closures_2016_2019.csv
pos2016.csv
pos2017.csv
pos2018.csv
pos2019.csv
pos_6789.csv
suspected_hospital_closures_2016_2019_sorted.csv
suspected_short_term_hospital_closures_2016_2019.csv
suspected_short_term_hospital_closures_2016_2019_sorted.csv
tx_closures_by_zip.csv
Successfully loaded pos2016.csv with encoding utf-8.

C:\Users\clari\AppData\Local\Temp\ipykernel_10652\3845987685.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

Successfully loaded pos2017.csv with encoding utf-8.
Error loading pos2018.csv with encoding utf-8: 'utf-8' codec can't decode
byte 0xa0 in position 7482: invalid start byte

C:\Users\clari\AppData\Local\Temp\ipykernel_10652\3845987685.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
```

```
Successfully loaded pos2018.csv with encoding ISO-8859-1.
Error loading pos2019.csv with encoding utf-8: 'utf-8' codec can't decode
byte 0x98 in position 4114: invalid start byte

C:\Users\clari\AppData\Local\Temp\ipykernel_10652\3845987685.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

Successfully loaded pos2019.csv with encoding ISO-8859-1.
Successfully appended all datasets.
Combined dataset saved as C:/Users/clari/OneDrive/Documents/Python
II/pos_files\pos_6789.csv
   PRVDR_CTGRY_SBTYP_CD  PRVDR_CTGRY_CD                          FAC_NAME  \
0                   1.0               1  SOUTHEAST ALABAMA MEDICAL CENTER
1                   1.0               1            NORTH JACKSON HOSPITAL
2                   1.0               1     MARSHALL MEDICAL CENTER SOUTH
3                   1.0               1    ELIZA COFFEE MEMORIAL HOSPITAL
4                   1.0               1          MIZELL MEMORIAL HOSPITAL

  PRVDR_NUM STATE_CD   ZIP_CD  GNRL_FAC_TYPE_CD  PGM_TRMNTN_CD  Year
0    010001       AL  36301.0               1.0              0  2016
1    010004       AL  35740.0               1.0              1  2016
2    010005       AL  35957.0               1.0              0  2016
3    010006       AL  35631.0               1.0              0  2016
4    010007       AL  36467.0               1.0              0  2016

C:\Users\clari\AppData\Local\Temp\ipykernel_10652\3845987685.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
```

There was a utf-8 error, so I asked chatGPT how to fix it

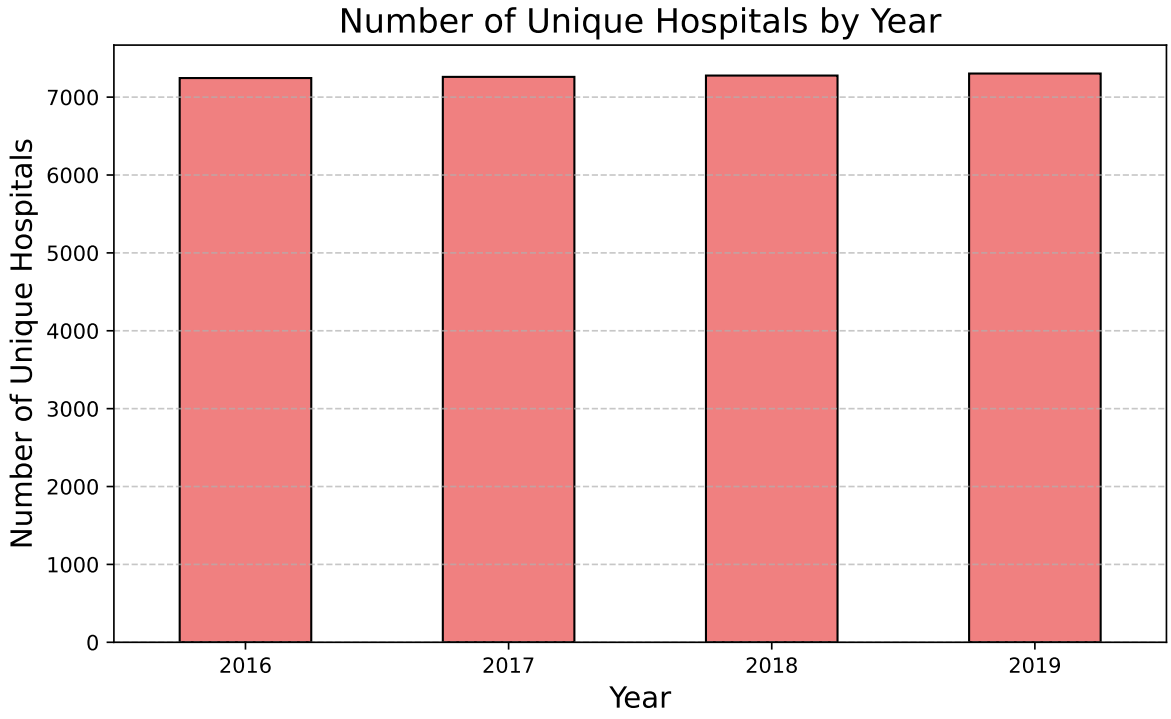
```
combined_df = pd.read_csv(os.path.join(folder_path, 'pos_6789.csv'))

# Group by 'Year' and count the number of rows per year
hospitals_by_year = combined_df.groupby('Year').size()

# Plotting the number of hospitals by year
plt.figure(figsize=(8, 5))
hospitals_by_year.plot(kind='bar', color='lightcoral')
plt.title('Number of Hospitals by Year')
plt.xlabel('Year')
plt.ylabel('Number of Hospitals')
plt.xticks(rotation=0)
plt.show()
```



4. a.

```
unique_hospitals_by_year = combined_df.groupby('Year')['PRVDR_NUM'].nunique()

# Plotting the number of unique hospitals by year
```

```
plt.figure(figsize=(8, 5))
unique_hospitals_by_year.plot(kind='bar', color='lightcoral',
 ↪  edgecolor='black')
plt.title('Number of Unique Hospitals by Year', fontsize=16)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Number of Unique Hospitals', fontsize=14)
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



b.Both plots (total vs. unique hospitals) give the same result. This strongly suggests that the dataset is clean and well-structured, with no duplicate entries for hospitals based on their CMS certification numbers (PRVDR_NUM). Each hospital appears only once per year, which means I can confidently proceed with further analysis without worrying about duplicate data affecting your results.

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

```
import altair as alt
alt.renderers.enable("png")
from vega_datasets import data as vega_data
import warnings
warnings.filterwarnings('ignore')

folder_path = r'C:/Users/clari/OneDrive/Documents/Python
 ↪  II/pos_files/pos_6789.csv'
# path for eddie: 'C:\Users\eddie\Downloads\pos_6789 (2).csv'

df = pd.read_csv(folder_path)
```

2.1

Checking columns to verify

```
# Get a list of all column names
column_names = df.columns.tolist()

# Print the list of column names
print(column_names)
```

```
['PRVDR_CTGRY_SBTYP_CD', 'PRVDR_CTGRY_CD', 'FAC_NAME', 'PRVDR_NUM',
'STATE_CD', 'ZIP_CD', 'GNRL_FAC_TYPE_CD', 'PGM_TRMNTN_CD', 'Year']
```

```
# Step 1: Filter hospitals active in 2016
active_2016 = df[(df['Year'] == 2016) & (df['PGM_TRMNTN_CD'] == 0)]

# Step 2: Initialize a list to capture closures
closures = []

# Step 3: Loop through each hospital that was active in 2016
for _, row in active_2016.iterrows():
    prvd_num = row['PRVDR_NUM']
    facility_name = row['FAC_NAME']
    zip_code = row['ZIP_CD']
    suspected_closure_year = None

    # Check each year from 2017 to 2019
```

```
    for year in range(2017, 2020):
        record = df[(df['Year'] == year) & (df['PRVDR_NUM'] == prvd_num)]

        # Check if hospital is not active or missing
        if record.empty or record.iloc[0]['PGM_TRMNTN_CD'] != 0:
            suspected_closure_year = year
            break

    # If closure year was found, add it to the list
    if suspected_closure_year:
        closures.append({
            'Facility Name': facility_name,
            'ZIP_CD': zip_code,
            'Suspected Closure Year': suspected_closure_year
        })

# Step 4: Convert the list to a DataFrame and display the count of closures
closures_df = pd.DataFrame(closures)
closure_count = closures_df.shape[0]

print(f"Total suspected closures: {closure_count}")
closures_df.head()
```

```
Total suspected closures: 174
```

	Facility Name	ZIP_CD	Suspected Closure Year
0	WEDOWEE HOSPITAL	36278.0	2019
1	GEORGIANA MEDICAL CENTER	36033.0	2019
2	RMC JACKSONVILLE	36265.0	2018
3	NORTH ALABAMA SPECIALITY HOSPITAL	35611.0	2018
4	ABRAZO MARYVALE CAMPUS	85031.0	2017

According to this, there are 174 hospitals that were active and have since closed.

2.2

```
# Sort closures by Facility Name and select the first 10 rows
sorted_closures_df = closures_df.sort_values(by='Facility Name').head(10)

# Display the Facility Name and Suspected Closure Year for the first 10 rows
print(sorted_closures_df[['Facility Name', 'Suspected Closure Year']])
```

	Facility Name	Suspected 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

2.3 - i

(After running into errors setting up the for loop, ChatGPT was used for clean-up and debugging)

```
# Step 1: Calculate number of active hospitals per zip code and year
active_counts = df[df['PGM_TRMNTN_CD'] == 0].groupby(
    ['ZIP_CD', 'Year']).size().reset_index(name='Active Hospital Count')

# Step 2: Create empty lists for mergers and closures.
potential_mergers = []
confirmed_closures = []

# Step 3: Loop through each row of the closure df
for _, closure in closures_df.iterrows():
    zip_code = closure['ZIP_CD']
    closure_year = closure['Suspected Closure Year']

    # Get active counts for the zip code in the closure year and the
    ↪ following year
    active_current_year = active_counts[(active_counts['ZIP_CD'] == zip_code)
↪   & (
        active_counts['Year'] == closure_year)]
    active_next_year = active_counts[(active_counts['ZIP_CD'] == zip_code) &
↪   (
        active_counts['Year'] == closure_year + 1)]

    # Check if there is an increase in hospital counts from one year to next
    if not active_current_year.empty and not active_next_year.empty:
        if active_next_year.iloc[0]['Active Hospital Count'] >=
        ↪ active_current_year.iloc[0]['Active Hospital Count']:
```

```
            # Potential merger if active hospital count is stable or
             ↪  increases
            potential_mergers.append(closure)
            continue

    # Otherwise, it's a confirmed closure
    confirmed_closures.append(closure)

# Step 4: Convert the lists to DataFrames
potential_mergers_df = pd.DataFrame(potential_mergers)
confirmed_closures_df = pd.DataFrame(confirmed_closures)

# Step 5: Output
print(f"Potential mergers: {len(potential_mergers_df)}")
print(f"Confirmed closures after correction: {len(confirmed_closures_df)}")

confirmed_closures_df.head()
```

```
Potential mergers: 27
Confirmed closures after correction: 147
```

	Facility Name	ZIP_CD	Suspected Closure Year
0	WEDOWEE HOSPITAL	36278.0	2019
1	GEORGIANA MEDICAL CENTER	36033.0	2019
2	RMC JACKSONVILLE	36265.0	2018
4	ABRAZO MARYVALE CAMPUS	85031.0	2017
5	BANNER PAYSON MEDICAL CENTER	85541.0	2018

We identify 27 potential mergers, based on an increase / stabilization of hospital amounts per zip code from one year to the next. By subtracting this from our previous closure total of 174, we get 174-27 = 147, our confirmed closures after accounting for these mergers.

2.3 - ii

```
# Sort confirmed closures by Facility Name and select the first 10 rows
sorted_confirmed_closures_df = confirmed_closures_df.sort_values(
    by='Facility Name').head(10)

# Display the Facility Name and Suspected Closure Year for the first 10 rows
print(sorted_confirmed_closures_df[[
      'Facility Name', 'ZIP_CD', 'Suspected Closure Year']])
```

	Facility Name	ZIP_CD	Suspected Closure Year
4	ABRAZO MARYVALE CAMPUS	85031.0	2017
97	AFFINITY MEDICAL CENTER	44646.0	2018
140	ALLEGIANCE SPECIALTY HOSPITAL OF KILGORE	75662.0	2017
62	ALLIANCE LAIRD HOSPITAL	39365.0	2019
101	ALLIANCEHEALTH DEACONESS	73112.0	2019
26	ANNE BATES LEACH EYE HOSPITAL	33136.0	2019
21	ARKANSAS VALLEY REGIONAL MEDICAL CENTER	81050.0	2017
69	BANNER CHURCHILL COMMUNITY HOSPITAL	89406.0	2017
5	BANNER PAYSON MEDICAL CENTER	85541.0	2018
115	BARIX CLINICS OF PENNSYLVANIA	19047.0	2019

Download Census zip code shapefile (10 pt)

1. a. .shp (Shapefile): This is the main file that contains the geometric data (points, lines, polygons) that represent the shapes of the geographic features. It is 817, 915 KB .shx (Shape Index File): This contains an index of the geometry in the .shp file, allowing for quick access to the geometric features. It is 259 KB. .dbf (Database File): This contains attribute data for each shape in the .shp file. It is stored in a tabular format (each row corresponds to a shape, and columns contain attributes like names, IDs, or other relevant information).It is 6,275 KB. .prj (Projection File): This file contains information about the coordinate system and map projection used by the shapefile. It ensures that the geographic data can be accurately placed on a map. It is 1 KB .xml(Extensible Markup Language) It's used to describe data and store data in a shareable manner. XML supports information exchange between computer systems such as websites, databases, and third-party applications.It is 16 KB.
2.

```
shapefile_path = r'C:\Users\clari\OneDrive\Documents\Python
 ↪  II\gz_2010_us_860_00_500k\gz_2010_us_860_00_500k.shp'
zip_restrict = gpd.read_file(shapefile_path)

# Filter for Texas ZIP codes (starting with 75, 76, 77, 78, 79, 733, or 885)
zip_texas_prefixes = ['75', '76', '77', '78', '79', '733', '885']
zip_restrict['ZIP'] = zip_restrict['ZCTA5'].astype(str)

# Filter for Texas ZIP codes based on their prefixes
zip_texas = zip_restrict[zip_restrict['ZIP'].str.startswith(
    tuple(zip_texas_prefixes))]

# Load the cleaned POS data for 2016
pos_2016_df = pd.read_csv(
    r'C:/Users/clari/OneDrive/Documents/Python II/pos_files/pos2016.csv')
pos_2016_df = pos_2016_df[(pos_2016_df['PRVDR_CTGRY_CD'] == 1) & (
    pos_2016_df['PRVDR_CTGRY_SBTYP_CD'] == 1)]
print(len(pos_2016_df))

# Convert floating-point ZIP codes to integers and then to strings (remove
 ↪  decimal points)
pos_2016_df['ZIP_CD'] =
 ↪  pos_2016_df['ZIP_CD'].fillna(0).astype(int).astype(str)

# Group by ZIP code and count the number of unique hospitals (unique CMS
 ↪  certification numbers) per ZIP code
hospitals_per_zip = pos_2016_df.groupby(
    'ZIP_CD')['PRVDR_NUM'].nunique().reset_index()
hospitals_per_zip.columns = ['ZIP_CD', 'num_hospitals']

# Ensure both DataFrames have consistent ZIP code formatting (string)
hospitals_per_zip['ZIP'] = hospitals_per_zip['ZIP_CD'].astype(str)
zip_texas['ZIP'] = zip_texas['ZIP'].astype(str)

# Debugging: Check if there are matching ZIP codes between the two DataFrames
matching_zips = set(zip_texas['ZIP']).intersection(
    set(hospitals_per_zip['ZIP']))
print(f"Number of matching ZIP codes: {len(matching_zips)}")
if len(matching_zips) == 0:
    print("No matching ZIP codes found between zip_texas and
     ↪  hospitals_per_zip.")
```

```
# Perform the merge without duplicating columns
zip_texas = zip_texas.merge(
    hospitals_per_zip[['ZIP', 'num_hospitals']], on='ZIP', how='left')

# Fill NaN values (for areas without hospitals) with 0
zip_texas['num_hospitals'] = zip_texas['num_hospitals'].fillna(
    0).astype(int).astype('category')
```

```
7245
Number of matching ZIP codes: 490
```

```
# Convert the 'num_hospitals' column to integer type
zip_texas_count = zip_texas.copy()
zip_texas_count['num_hospitals'] = zip_texas['num_hospitals'].astype(int)
# Sort the DataFrame by number of hospitals in descending order
zip_texas_sorted = zip_texas_count.sort_values(
    'num_hospitals', ascending=False)

# ZIP codes and number of hospitals
for _, row in zip_texas_count.iterrows():
    if row['num_hospitals'] > 0:
        print(f"{row['ZIP']:7} | {row['num_hospitals']}")

# Calculate and print total number of hospitals
total_hospitals = zip_texas_count['num_hospitals'].sum()
print(f"\nTotal number of hospitals in Texas: {total_hospitals}")
```

```
78624   | 1
78626   | 1
78636   | 1
78640   | 1
78643   | 1
78654   | 1
78681   | 1
78704   | 1
78734   | 1
78758   | 1
78759   | 1
78834   | 1
78840   | 1
78942   | 1
78945   | 2
```

```
79007   | 2
79015   | 1
75563   | 1
75568   | 1
75570   | 3
75652   | 1
75668   | 2
75670   | 1
75684   | 1
75783   | 1
75801   | 2
75831   | 1
75835   | 1
75901   | 2
75951   | 4
75956   | 1
75966   | 1
79022   | 1
79027   | 1
79029   | 1
79035   | 1
79041   | 1
79045   | 1
79070   | 1
79106   | 4
79124   | 1
79241   | 1
79248   | 1
79252   | 1
79316   | 1
79336   | 2
79347   | 2
79356   | 1
79360   | 1
79364   | 1
79373   | 1
75972   | 1
76012   | 2
76018   | 1
76028   | 2
76033   | 1
76043   | 1
76048   | 1
```

76092	1
76107	2
76110	2
76117	1
76201	3
76227	1
76230	1
76255	1
76301	2
76351	1
79415	3
79416	1
79501	1
79502	2
79512	1
79520	1
79521	1
79529	2
79546	1
79553	1
79601	1
79605	1
79701	2
79731	1
79744	1
79752	1
79782	1
79902	10
79904	1
79905	1
79925	1
79936	1
76360	1
76365	1
76374	1
76380	1
76384	1
76424	1
76426	1
76437	1
76448	1
76454	1
76457	1

```
76470   | 1
76542   | 1
76550   | 1
76567   | 1
76570   | 1
76574   | 1
76642   | 1
76645   | 1
76691   | 1
76692   | 1
76712   | 2
76801   | 1
76837   | 1
76849   | 1
76859   | 1
76903   | 1
76905   | 2
76936   | 3
77024   | 1
77030   | 6
77035   | 2
77043   | 1
77054   | 7
77055   | 3
77063   | 1
77065   | 2
77070   | 3
77074   | 2
77077   | 1
77081   | 1
77090   | 2
77094   | 1
77304   | 5
77328   | 1
77340   | 1
77414   | 1
77418   | 1
77437   | 1
77450   | 1
77465   | 1
77469   | 1
77477   | 1
77480   | 1
```

77488 | 1
77494 | 3
77502 | 1
77503 | 1
77505 | 1
77511 | 1
77515 | 1
77521 | 3
77530 | 1
77550 | 1
77566 | 1
77575 | 1
77625 | 1
77640 | 1
77642 | 2
77707 | 1
77859 | 1
77868 | 1
77904 | 1
77954 | 1
77962 | 2
75013 | 1
75020 | 3
75032 | 1
75039 | 1
75041 | 1
75061 | 2
75063 | 1
75070 | 2
75082 | 1
75090 | 2
75093 | 4
75098 | 3
75110 | 1
75115 | 2
75119 | 2
75140 | 2
75142 | 1
77979 | 1
77984 | 1
77995 | 1
78013 | 2
78017 | 3

78026 | 1
78061 | 1
78118 | 1
78204 | 1
78205 | 3
78207 | 1
78224 | 2
78229 | 5
78233 | 1
78240 | 2
75149 | 4
75160 | 2
75165 | 1
75203 | 3
75206 | 1
75217 | 1
75227 | 1
75230 | 1
75234 | 1
75243 | 1
75401 | 1
75455 | 1
75503 | 1
75551 | 2
78249 | 1
78258 | 2
78332 | 2
78336 | 1
78355 | 1
78390 | 1
78414 | 3
78503 | 4
78539 | 3
78550 | 3
78582 | 1
78596 | 1
78613 | 2
75601 | 2
75605 | 1
75633 | 1
75644 | 2
75647 | 1
78644 | 2

78648 | 1
78664 | 2
78666 | 1
78701 | 1
75657 | 1
75686 | 1
75701 | 6
75702 | 2
75708 | 1
75751 | 1
75766 | 1
75773 | 1
75785 | 1
75840 | 1
75844 | 1
78702 | 1
78705 | 2
78731 | 1
78737 | 1
75860 | 1
75862 | 2
75904 | 1
75935 | 3
75961 | 2
78746 | 3
78756 | 1
79065 | 1
79072 | 1
79079 | 1
79081 | 1
79088 | 1
78801 | 1
78861 | 1
78880 | 1
78934 | 1
79096 | 1
78957 | 1
78962 | 2
79014 | 1
79109 | 2
79201 | 1
79225 | 1
79227 | 1

79245	1
79312	1
79322	1
79323	1
79331	1
79339	1
79346	1
79401	1
79549	1
79556	1
79567	1
79703	3
79706	1
79714	1
79720	2
79735	1
79745	2
79761	5
79772	1
79778	1
76634	1
77076	3
77080	1
77082	1
77087	1
77091	1
77093	2
77327	2
77338	4
77339	1
77351	1
77380	4
77384	2
77401	2
77429	1
78010	1
78041	3
78044	1
77445	1
77459	1
77506	2
77514	1
77520	1

77547 | 1
78102 | 1
78114 | 1
78119 | 1
78130 | 2
77474 | 1
77478 | 1
77479 | 4
77584 | 1
77591 | 1
77598 | 3
77612 | 1
77619 | 1
77627 | 1
77630 | 2
77656 | 1
78155 | 1
78164 | 1
78201 | 1
78212 | 1
77701 | 2
77702 | 2
77706 | 1
77802 | 3
77833 | 2
77836 | 2
77845 | 1
77864 | 1
77901 | 1
77957 | 1
78223 | 1
78230 | 1
78232 | 1
77963 | 1
77964 | 1
75226 | 1
75231 | 4
75235 | 6
75237 | 1
75246 | 3
78247 | 1
75182 | 1
75001 | 1

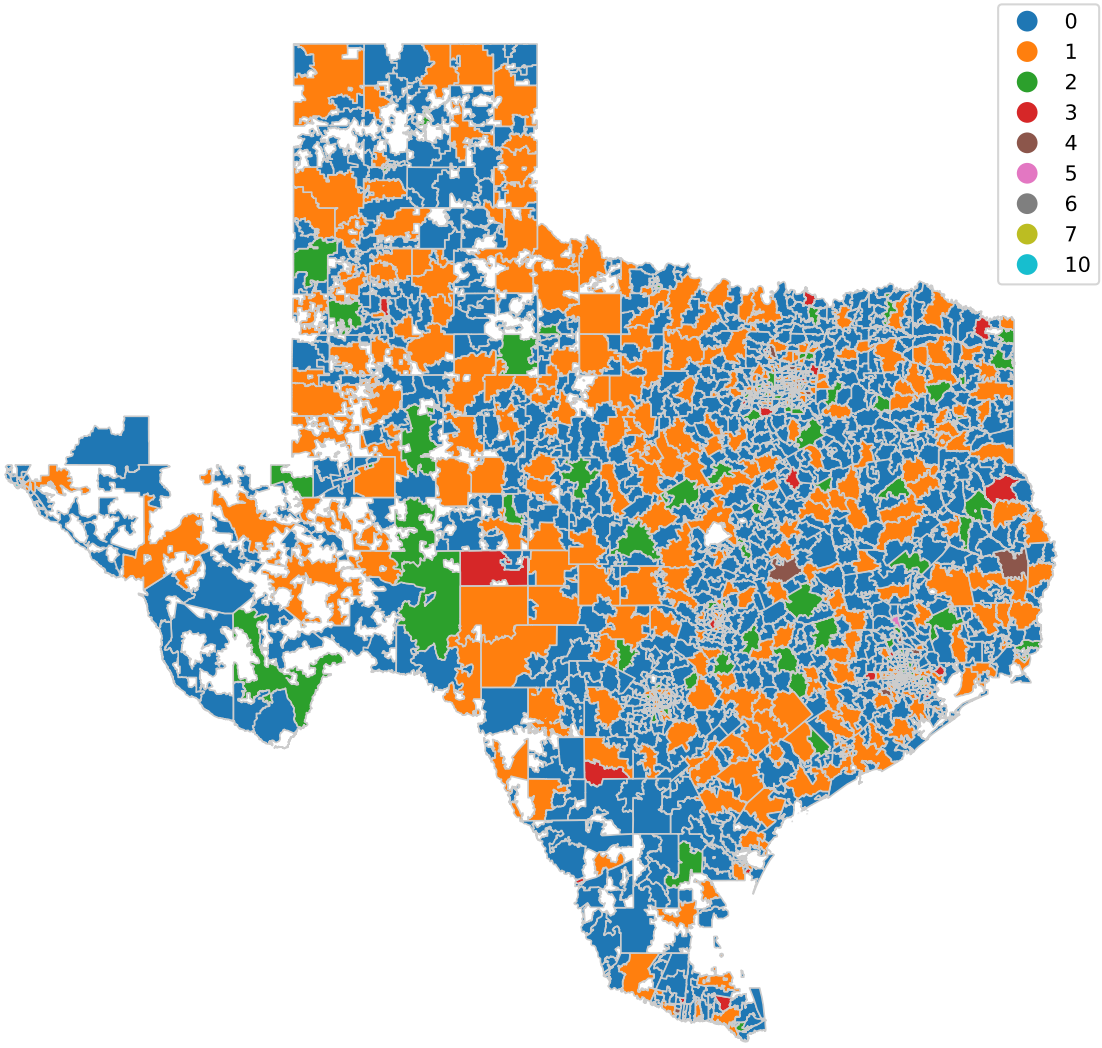
```
75010   | 2
75028   | 1
75034   | 1
75204   | 2
75208   | 1
75214   | 1
75218   | 2
75220   | 1
75224   | 2
75042   | 3
75051   | 2
75057   | 1
75069   | 2
75071   | 1
75075   | 1
75087   | 1
75146   | 2
79830   | 2
79855   | 1
79901   | 1
76648   | 3
76661   | 1
76665   | 2
76667   | 1
76693   | 2
76011   | 1
76015   | 2
76017   | 1
76020   | 1
76021   | 1
79915   | 1
79938   | 1
76821   | 1
76834   | 2
76844   | 1
76856   | 1
76877   | 2
76051   | 2
76054   | 2
76063   | 3
76067   | 1
76086   | 1
76904   | 1
```

76932	2
76943	2
76945	1
76950	1
77002	3
77004	6
76022	1
76104	6
76106	1
76108	1
76132	3
76177	2
76180	2
76208	4
76210	1
77006	1
77008	2
77015	1
76234	1
76244	1
76252	1
76262	1
76401	1
76430	1
76442	1
76444	1
76446	1
76450	1
76458	1
76483	1
77020	2
79410	3
79412	1
79413	2
76502	2
76508	1
76520	4
76528	1
76531	2
76548	1
77027	1
77028	1
75390	2

75418 | 1
75426 | 1
75428 | 1
75457 | 1
75460 | 2
75462 | 1
75482 | 1
75494 | 1
75501 | 2
78363 | 1
78377 | 1
78404 | 4
78411 | 2
78415 | 1
78520 | 1
78526 | 2
78572 | 1
78580 | 1
78602 | 2
78611 | 1
76310 | 1
77504 | 2
77665 | 1
75088 | 1
78852 | 1
78629 | 1
75662 | 1
75948 | 1
79095 | 1
79235 | 1
75979 | 1
76240 | 1
79606 | 1
79756 | 1
76825 | 1
76951 | 1
77029 | 1
77058 | 1
77375 | 2
77434 | 1
75035 | 1
78028 | 1
78222 | 2

```
78412   | 1
78586   | 1

Total number of hospitals in Texas: 731
```

```python
# Plot a choropleth map of the number of hospitals by ZIP code in Texas
fig, ax = plt.subplots(1, 1, figsize=(10, 10))
zip_texas.plot(column='num_hospitals', linewidth=0.8,
               ax=ax, edgecolor='0.8', legend=True)

plt.title('Number of Hospitals by ZIP Code in Texas (2016)', fontsize=15)
plt.axis('off')
plt.show()
```



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

4.1

```
import time

# Create a new GeoDataFrame with centroids
zips_all_centroids = zip_restrict.copy()
```

```
zips_all_centroids['geometry'] = zip_restrict.centroid

# Check the CRS and convert if necessary
print(zips_all_centroids.crs)  # Check current CRS
# Convert to projected CRS (e.g., EPSG:3857)
zips_all_centroids = zips_all_centroids.to_crs(epsg=3857)

# Reset the index to ensure it's unique
zips_all_centroids = zips_all_centroids.reset_index(drop=True)

# Print first few rows and dimensions
print(zips_all_centroids.head())
print(f'The GeoDataFrame dimensions are {zips_all_centroids.shape}')
```

```
EPSG:4269
           GEO_ID  ZCTA5   NAME   LSAD  CENSUSAREA  \
0  8600000US01040  01040  01040  ZCTA5      21.281
1  8600000US01050  01050  01050  ZCTA5      38.329
2  8600000US01053  01053  01053  ZCTA5       5.131
3  8600000US01056  01056  01056  ZCTA5      27.205
4  8600000US01057  01057  01057  ZCTA5      44.907

                           geometry    ZIP
0  POINT (-8086367.189 5192874.337)  01040
1  POINT (-8111834.501 5204197.981)  01050
2  POINT (-8094219.962 5214078.082)  01053
3  POINT (-8065992.709 5189806.911)  01056
4  POINT (-8051104.124 5174717.908)  01057
The GeoDataFrame dimensions are (33120, 7)
```

Based on the xml file, the columns mean this:

GEO_ID: This is a unique identifier for each geographic entity. It's an alphanumeric field that can be used to join these spatial tables to other 2010 Census data tables.

ZCTA5: This stands for ZIP Code Tabulation Area (5-digit). It's a 5-digit Census code representing the ZIP code area.

NAME: This field contains the name of the geographic entity without the translated Legal/Statistical Area Description (LSAD). It's an alphanumeric field.

LSAD: This stands for Legal/Statistical Area Description. It's a standard abbreviation used on census maps, represented as an alpha (text) field.

CENSUSAREA: This represents the area of the entity before generalization, measured in square miles. It's a numeric field derived from the ungeneralized area of each entity and can be used for density calculations.

geometry: This column contains the geometric information for each ZIP code area. In the original shapefile, this would be a polygon representing the boundaries of the ZIP code area.

ZIP: I have a duplicate of the ZCTA5 field, which I added for convenience or compatibility with other datasets.

centroid: This column contains the calculated centroid (center point) of each ZIP code area. It's a Point geometry representing the geographic center of the ZIP code polygon.

4.2

```
# Filter Texas zip codes
texas_zip_prefixes = ['75', '76', '77', '78', '79', '733', '885']
zips_texas_centroids = zips_all_centroids[zips_all_centroids['ZIP'].str
 ↪  .startswith(tuple(texas_zip_prefixes))]
print(f"Number of unique zip codes in Texas:
 ↪  {zips_texas_centroids['ZIP'].nunique()}")

# Define and filter for bordering states' ZIP codes
border_states_zip_prefixes = texas_zip_prefixes + ['73', '72', '71', '68',
 ↪  '69']
zips_texas_borderstates_centroids = zips_all_centroids[zips_all_centroids[
 ↪  'ZIP'].str.startswith(tuple(border_states_zip_prefixes))]
print(f"Number of unique zip codes in Texas and bordering states:
 ↪  {zips_texas_borderstates_centroids['ZIP'].nunique()}")
```

```
Number of unique zip codes in Texas: 1935
Number of unique zip codes in Texas and bordering states: 3628
```

4.3

```
# Prepare hospital data
pos_2016_df['ZIP_CD'] =
 ↪  pos_2016_df['ZIP_CD'].fillna(0).astype(int).astype(str)
hospitals_per_zip =
 ↪  pos_2016_df.groupby('ZIP_CD')['PRVDR_NUM'].nunique().reset_index()
hospitals_per_zip.columns = ['ZIP', 'num_hospitals']

# Ensure both DataFrames have consistent ZIP code formatting (string)
hospitals_per_zip['ZIP'] = hospitals_per_zip['ZIP'].astype(str)
```

```
zips_texas_borderstates_centroids['ZIP'] =
 ↪  zips_texas_borderstates_centroids['ZIP'].astype(str)

# Merge to find zip codes with hospitals
zips_withhospital_centroids = zips_texas_borderstates_centroids.merge(
    hospitals_per_zip[['ZIP', 'num_hospitals']], on='ZIP', how='inner'
)
print(f"Dimensions of zips_withhospital_centroids:
 ↪  {zips_withhospital_centroids.shape}")
print(zips_withhospital_centroids.head())
```

```
Dimensions of zips_withhospital_centroids: (790, 8)
           GEO_ID  ZCTA5   NAME   LSAD  CENSUSAREA  \
0  8600000US68047  68047  68047  ZCTA5     126.239
1  8600000US68071  68071  68071  ZCTA5      75.085
2  8600000US68122  68122  68122  ZCTA5      19.330
3  8600000US68124  68124  68124  ZCTA5       5.739
4  8600000US68130  68130  68130  ZCTA5       7.631

                            geometry    ZIP  num_hospitals
0  POINT (-10769098.803 5176977.986)  68047              1
1  POINT (-10739120.884 5196342.029)  68071              1
2   POINT (-10692355.438 5066810.57)  68122              2
3   POINT (-10692413.598 5047110.35)  68124              1
4  POINT (-10708477.136 5046952.188)  68130              1
```

We performed an inner merge on the ZIP column to create a subset of zips_texas_borderstates_centroids that contains only those ZIP codes with at least one hospital in 2016. The resulting GeoDataFrame (zips_withhospital_centroids) contains information about these ZIP codes, including their centroids and how many hospitals they had in 2016.

4.4 a

```
# Calculate distance to nearest hospital
zips_texas_subset = zips_texas_centroids.head(10)
start_time = time.time()
zips_texas_subset['nearest_hospital_dist'] =
 ↪  zips_texas_subset.geometry.apply(
    lambda x: zips_withhospital_centroids.distance(x).min()
)
end_time = time.time()
print(zips_texas_subset[['ZIP', 'nearest_hospital_dist']])
```

```
print(f"Time taken for 10 ZIP codes: {end_time - start_time:.2f} seconds")

# Check for NaN distances
print(zips_texas_subset['nearest_hospital_dist'].isna().sum())
```

```
        ZIP  nearest_hospital_dist
9207  78624               0.000000
9208  78626               0.000000
9209  78628           14228.011802
9210  78631           42374.908087
9211  78632           18272.293214
9212  78633           20089.785132
9213  78634           13251.123905
9214  78635           19819.708012
9215  78636               0.000000
9216  78638           18391.605097
Time taken for 10 ZIP codes: 0.01 seconds
0
```

The amount of time changes, but I get around .01 seconds to run 10 zip codes.

4.4 b

```
# Start timing the full computation
start_time_full = time.time()

# Calculate distances from each ZIP code in zips_texas_centroids to the
 ↪  nearest ZIP code with a hospital
zips_texas_centroids['nearest_hospital_dist'] =
 ↪  zips_texas_centroids.geometry.apply(
    lambda x: zips_withhospital_centroids.distance(x).min()
)

# End timing
end_time_full = time.time()

# Display results and time taken
print(zips_texas_centroids[['ZIP', 'nearest_hospital_dist']])
print(f"Time taken for full calculation: {end_time_full -
 ↪  start_time_full:.2f} seconds")

# Check for NaN distances
print(zips_texas_centroids['nearest_hospital_dist'].isna().sum())
```

```
         ZIP  nearest_hospital_dist
9207   78624               0.000000
9208   78626               0.000000
9209   78628           14228.011802
9210   78631           42374.908087
9211   78632           18272.293214
...      ...                    ...
32917  78261           12854.585280
32918  78368           39136.894333
32919  78412               0.000000
32920  78557            7258.416063
32921  78586               0.000000

[1935 rows x 2 columns]
Time taken for full calculation: 0.76 seconds
0
```

For me, the full calculation took .63-.68 seconds, much longer than just doing 10 zip codes. It's around 60 times longer.

c. This is what's in the prj file GEOGCS["GCS_North_American_1983", DATUM["D_North_American_1983", SPHEROID["GRS_1980",6378137,298.257222101]], PRIMEM["Greenwich",0], UNIT["Degree",0.017453292519943295]] One degree of latitude is roughly equivalent to 69 miles.

4.5

a. The distances are in degrees.

b.

```
# Calculate distance to hospital in miles
zips_texas_centroids['distance_to_hospital_miles'] =
 ↪  zips_texas_centroids['nearest_hospital_dist'] * 69

# Check for NaN values in the distance column
nan_distances =
 ↪  zips_texas_centroids['distance_to_hospital_miles'].isna().sum()
if nan_distances > 0:
    print(f"Warning: There are {nan_distances} NaN values in
     ↪  'distance_to_hospital_miles'.")
```

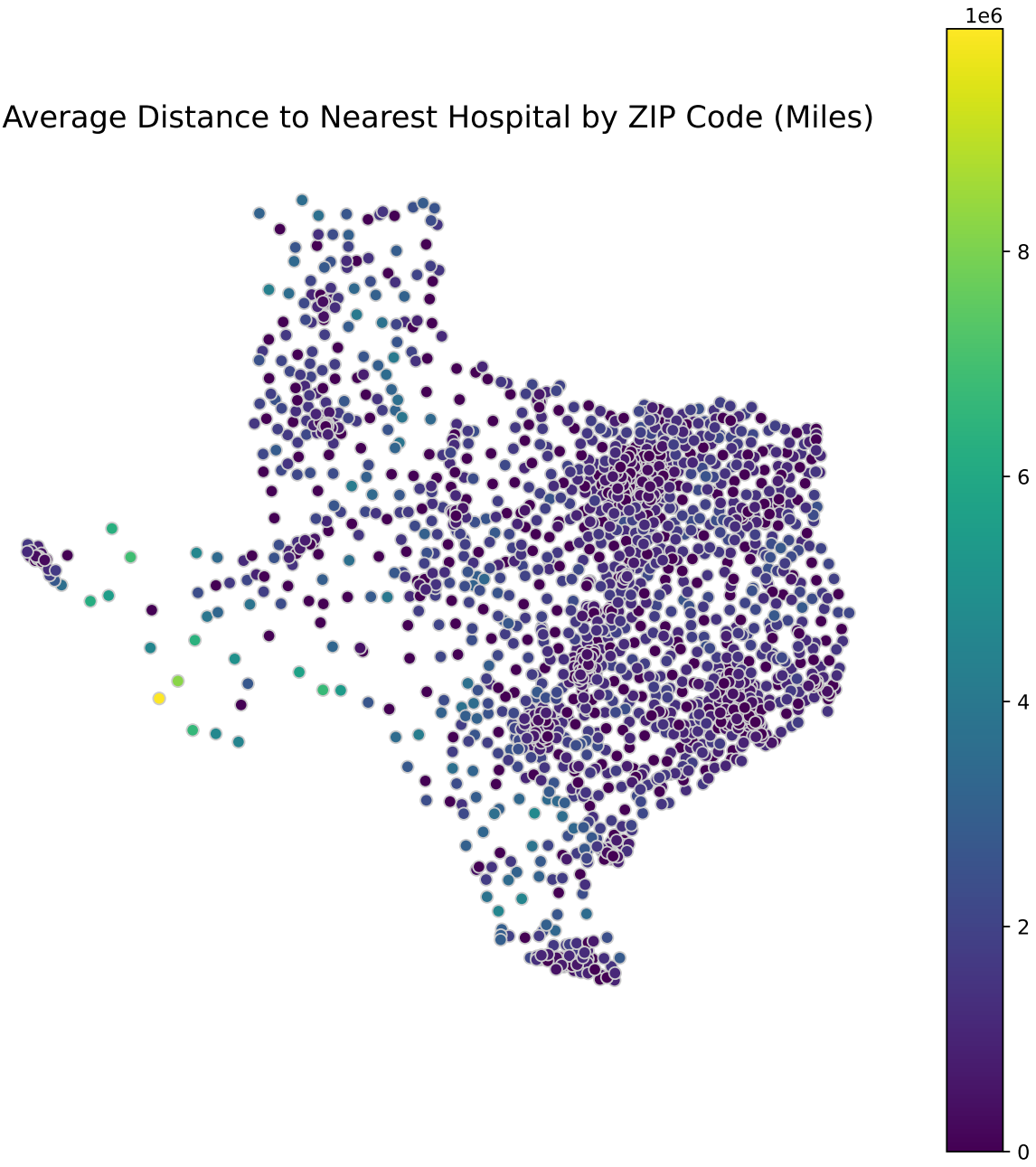
```
# Display the distances to the console
print(zips_texas_centroids[['ZIP', 'distance_to_hospital_miles']])
```

```
         ZIP  distance_to_hospital_miles
9207   78624                0.000000e+00
9208   78626                0.000000e+00
9209   78628                9.817328e+05
9210   78631                2.923869e+06
9211   78632                1.260788e+06
...      ...                         ...
32917  78261                8.869664e+05
32918  78368                2.700446e+06
32919  78412                0.000000e+00
32920  78557                5.008307e+05
32921  78586                0.000000e+00

[1935 rows x 2 columns]
```

This value makes sense. Although the values are by default in scientific notation, it is still a lot more intuitive to thing of distances in miles rather than 'degrees', a measurement rarely if ever used day-to-day when we think about moving from point A to B.

c

```
# Plotting the average distance to nearest hospital by ZIP code
fig, ax = plt.subplots(1, 1, figsize=(10, 10))
zips_texas_centroids.plot(column='distance_to_hospital_miles',
                          linewidth=0.8, ax=ax, edgecolor='0.8', legend=True)

plt.title('Average Distance to Nearest Hospital by ZIP Code (Miles)',
 ↪  fontsize=15)
plt.axis('off')  # Turn off axis
plt.show()
```



Effects of closures on access in Texas (15 pts)

1.

```
# Define Texas ZIP code prefixes
zip_texas_prefixes = ['75', '76', '77', '78', '79', '733', '885']

# Filter for closures in Texas between 2016-2019 based on ZIP code prefixes
texas_closures_2016_2019 = confirmed_closures_df[
    (confirmed_closures_df['ZIP_CD'].astype(str).str.startswith(tuple
 ↪  (zip_texas_prefixes))) &
    (confirmed_closures_df['Suspected Closure Year'] >= 2016) &
    (confirmed_closures_df['Suspected Closure Year'] <= 2019)
]

# Count closures by ZIP code
closures_by_zip = texas_closures_2016_2019['ZIP_CD'].value_counts(
).reset_index()
closures_by_zip.columns = ['ZIP_CD', 'Number of Closures']

# Display the table
print("Number of Hospital Closures by ZIP Code in Texas (2016-2019):")
print(closures_by_zip.to_string(index=False))

# Get the list of directly affected ZIP codes
affected_zip_codes = closures_by_zip['ZIP_CD'].tolist()

print(f"\nDirectly affected ZIP codes: {affected_zip_codes}")
print(f"Total number of affected ZIP codes: {len(affected_zip_codes)}")

# Calculate total number of closures
total_closures = closures_by_zip['Number of Closures'].sum()
print(f"Total number of hospital closures: {total_closures}")
```

```
Number of Hospital Closures by ZIP Code in Texas (2016-2019):
 ZIP_CD  Number of Closures
76502.0                   1
75601.0                   1
75087.0                   1
76520.0                   1
78017.0                   1
78613.0                   1
75051.0                   1
78734.0                   1
77035.0                   1
77429.0                   1
```

```
75235.0                   1
79902.0                   1
75390.0                   1
76531.0                   1
75862.0                   1
79529.0                   1
77065.0                   1
78834.0                   1
78336.0                   1
75835.0                   1
75662.0                   1
79553.0                   1
78061.0                   1
75042.0                   1
79520.0                   1
76645.0                   1
79735.0                   1
75140.0                   1

Directly affected ZIP codes: [76502.0, 75601.0, 75087.0, 76520.0, 78017.0,
78613.0, 75051.0, 78734.0, 77035.0, 77429.0, 75235.0, 79902.0, 75390.0,
76531.0, 75862.0, 79529.0, 77065.0, 78834.0, 78336.0, 75835.0, 75662.0,
79553.0, 78061.0, 75042.0, 79520.0, 76645.0, 79735.0, 75140.0]
Total number of affected ZIP codes: 28
Total number of hospital closures: 28
```

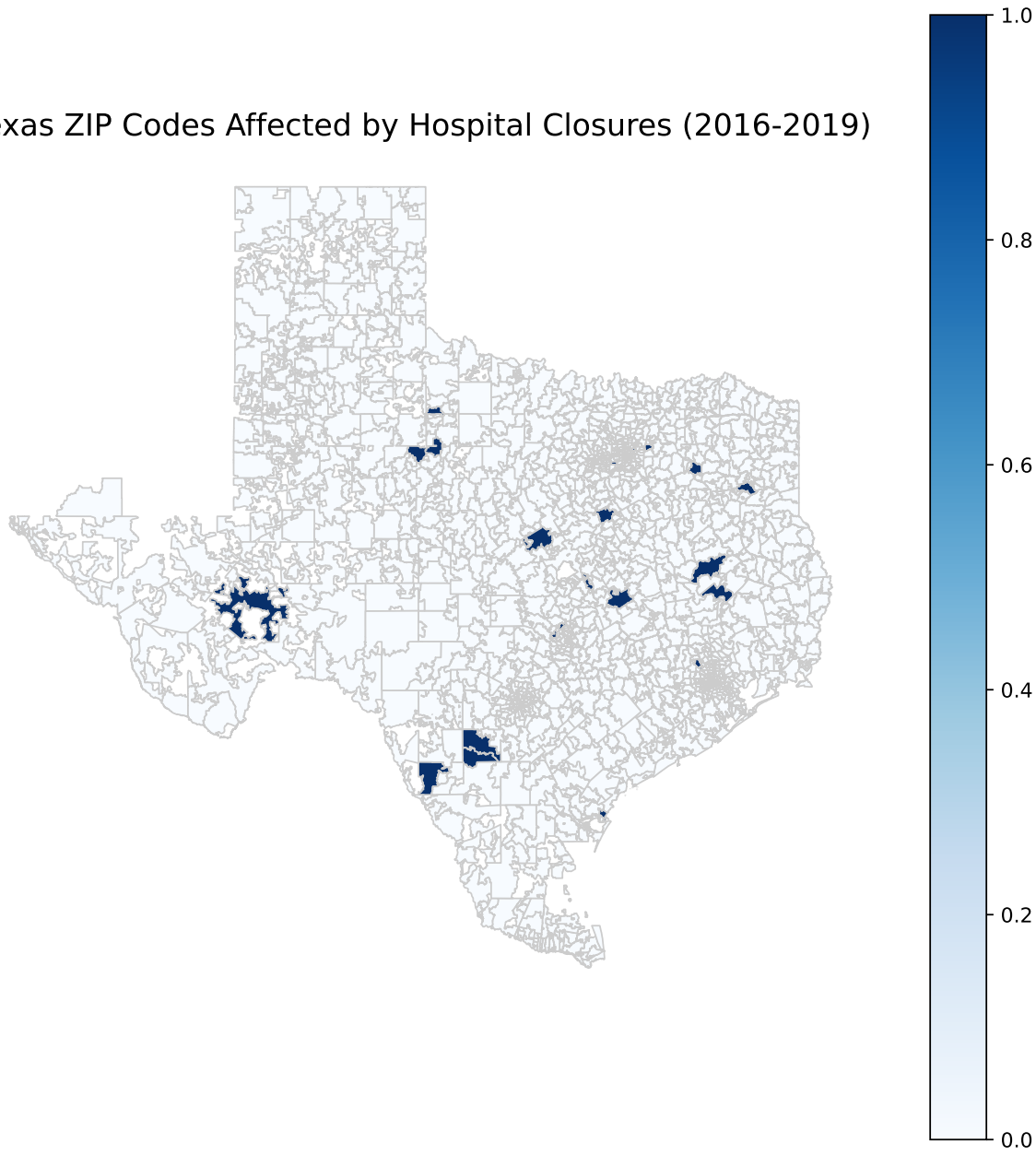
2.

```
# Define Texas ZIP code prefixes (already defined in your previous code)
zip_texas_prefixes = ['75', '76', '77', '78', '79', '733', '885']

# Filter for Texas ZIP codes based on their prefixes (reuse zip_restrict)
zip_restrict['ZIP'] = zip_restrict['ZCTA5'].astype(str)
zip_texas = zip_restrict[zip_restrict['ZIP'].str.startswith(
    tuple(zip_texas_prefixes))]

# Ensure closures_by_zip has consistent formatting (convert to string and
 ↪  remove decimal points)
closures_by_zip['ZIP_CD'] = closures_by_zip['ZIP_CD'].fillna(
    0).astype(int).astype(str)

# Debugging: Print sample data from both datasets to check for matching ZIP
 ↪  codes
```

```
print("Sample of closures_by_zip:")
print(closures_by_zip.head())

print("\nSample of zip_texas:")
print(zip_texas[['ZCTA5', 'ZIP']].head())

# Merge the closure data with the shapefile data for Texas ZIP codes
zip_texas_closure_map = zip_texas.merge(
    closures_by_zip, left_on='ZIP', right_on='ZIP_CD', how='left')

# Fill NaN values in 'Number of Closures' with 0 (for ZIPs with no closures)
zip_texas_closure_map['Number of Closures'] = zip_texas_closure_map['Number
 ↪  of Closures'].fillna(
    0)

# Debugging: Check if any closures were found after merging
print("\nSample of merged data (after filling NaNs):")
print(zip_texas_closure_map[['ZIP', 'Number of Closures']].head())
```

```
Sample of closures_by_zip:
  ZIP_CD  Number of Closures
0  76502                   1
1  75601                   1
2  75087                   1
3  76520                   1
4  78017                   1

Sample of zip_texas:
      ZCTA5    ZIP
9207  78624  78624
9208  78626  78626
9209  78628  78628
9210  78631  78631
9211  78632  78632

Sample of merged data (after filling NaNs):
     ZIP  Number of Closures
0  78624                 0.0
1  78626                 0.0
2  78628                 0.0
3  78631                 0.0
4  78632                 0.0
```

```
# Plot a choropleth map of the number of closures by ZIP code in Texas
fig, ax = plt.subplots(1, 1, figsize=(10, 10))
zip_texas_closure_map.plot(column='Number of Closures', cmap='Blues',
                           linewidth=0.8, ax=ax, edgecolor='0.8',
 ↪  legend=True)

plt.title('Texas ZIP Codes Affected by Hospital Closures (2016-2019)',
 ↪  fontsize=15)
plt.axis('off')  # Turn off axis

plt.show()

# Count how many unique ZIP codes were directly affected by at least one
 ↪  closure
affected_zip_codes_count =
 ↪  zip_texas_closure_map[zip_texas_closure_map['Number of Closures'] >
 ↪  0]['ZIP'].nunique(
)
print(
    f"Total number of directly affected ZIP codes in Texas:
     ↪  {affected_zip_codes_count}")
```




```
Total number of directly affected ZIP codes in Texas: 28
```

3.

```
directly_affected_zip_gdf =
↪  zip_texas_closure_map[zip_texas_closure_map['Number of Closures'] > 0]
```

```
# Adding uffer
buffered_zips = directly_affected_zip_gdf.copy()
buffered_zips['geometry'] = buffered_zips.geometry.buffer(16093.4)
```

```
# Merging
buffered_zips_minimal = buffered_zips[['ZIP', 'geometry']]
indirectly_affected_zip_gdf = gpd.sjoin(
    zip_texas,
    buffered_zips_minimal,
    how='inner',
    predicate='intersects'
)

# Remove ZIP codes that were directly affected
indirectly_affected_zip_gdf = indirectly_affected_zip_gdf[
    ~indirectly_affected_zip_gdf['ZCTA5'].isin(
        directly_affected_zip_gdf['ZIP'])
]

# Count the number of indirectly affected ZIP codes
num_indirectly_affected_zips = indirectly_affected_zip_gdf['ZCTA5'].nunique()
print(
    f"Number of indirectly affected ZIP codes:
     ↪ {num_indirectly_affected_zips}")

# Display the first few rows of the result to verify
print(indirectly_affected_zip_gdf[['ZCTA5']].head())

# Create a list of unique indirectly affected ZIP codes
indirectly_affected_zip_list =
 ↪ indirectly_affected_zip_gdf['ZCTA5'].unique().tolist()

# Limit the list to the first 20 ZIP codes (document crashes otherwise)
limited_zip_list = indirectly_affected_zip_list[:20]

# Print the limited list
print(f"List of indirectly affected ZIP codes (first 20):
 ↪ {limited_zip_list}")
```

```
Number of indirectly affected ZIP codes: 1907
      ZCTA5
```

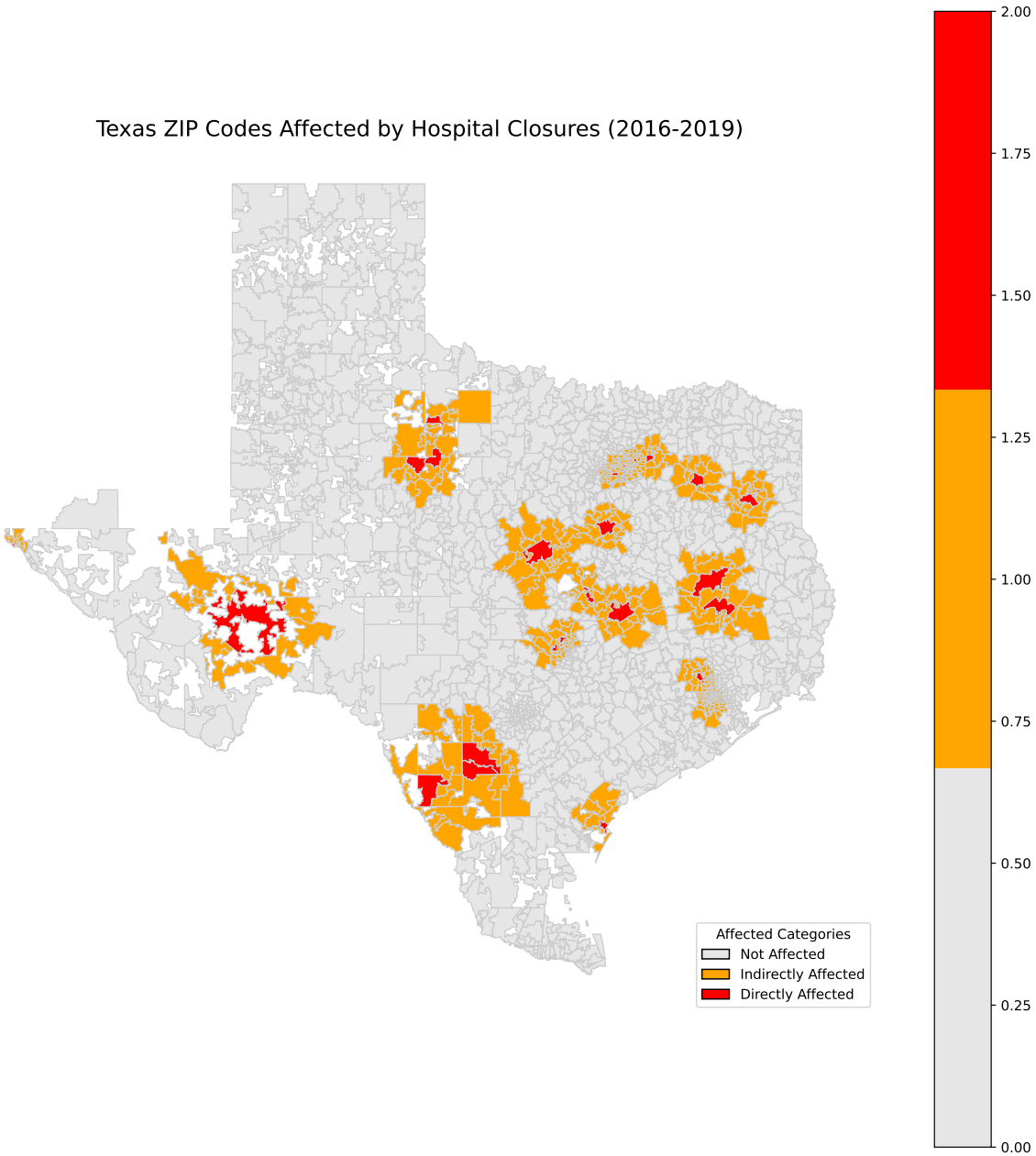
```
9207  78624
9207  78624
9207  78624
9207  78624
9207  78624
List of indirectly affected ZIP codes (first 20): ['78624', '78626', '78628',
'78631', '78632', '78633', '78634', '78635', '78636', '78638', '78639',
'78640', '78641', '78642', '78643', '78645', '78654', '78659', '78661',
'78663']
```

```
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from matplotlib.patches import Patch
```

```
# 1. Define Texas ZIP code prefixes
zip_texas_prefixes = ['75', '76', '77', '78', '79', '733', '885']

# 2. Filter for closures in Texas between 2016-2019 based on ZIP code
 ↪ prefixes
texas_closures_2016_2019 = confirmed_closures_df[
    (confirmed_closures_df['ZIP_CD'].astype(str).str.startswith(tuple
 ↪ (zip_texas_prefixes))) &
    (confirmed_closures_df['Suspected Closure Year'] >= 2016) &
    (confirmed_closures_df['Suspected Closure Year'] <= 2019)
]

# 3. Count closures by ZIP code
closures_by_zip = texas_closures_2016_2019['ZIP_CD'].value_counts(
).reset_index()
closures_by_zip.columns = ['ZIP_CD', 'Number of Closures']

# 4. Ensure closures_by_zip ZIP code column is string formatted for
 ↪ consistency
closures_by_zip['ZIP_CD'] = closures_by_zip['ZIP_CD'].fillna(
    0).astype(int).astype(str)

# 5. Filter Texas ZIP codes based on prefixes and ensure they are strings
zip_restrict['ZIP'] = zip_restrict['ZCTA5'].astype(str)
zip_texas = zip_restrict[zip_restrict['ZIP'].str.startswith(
    tuple(zip_texas_prefixes))]

# 6. Merge closure data with Texas ZIP codes
```

```
zip_texas_closure_map = zip_texas.merge(
    closures_by_zip, left_on='ZIP', right_on='ZIP_CD', how='left'
)

# 7. Fill NaN values in 'Number of Closures' with 0 for ZIPs with no closures
zip_texas_closure_map['Number of Closures'] = zip_texas_closure_map['Number
 ↪  of Closures'].fillna(
    0)

# 8. Convert to a projected CRS (meters) if necessary, and add a buffer
zip_texas_closure_map = zip_texas_closure_map.to_crs(epsg=3857)
buffered_zips = zip_texas_closure_map[zip_texas_closure_map['Number of
 ↪  Closures'] > 0].copy(
)
buffered_zips['geometry'] = buffered_zips.geometry.buffer(
    16093.4)  # 10-mile buffer

# 9. Keep only the ZIP and geometry columns for the spatial join
buffered_zips_minimal = buffered_zips[['ZIP', 'geometry']]

# 10. Reproject the original Texas ZIP data to match CRS
zip_texas = zip_texas.to_crs(epsg=3857)

# 11. Spatial join to find indirectly affected ZIP codes within the buffer
indirectly_affected_zip_gdf = gpd.sjoin(
    zip_texas,
    buffered_zips_minimal,
    how='inner',
    predicate='intersects'
)

# 12. Remove ZIP codes that were directly affected
indirectly_affected_zip_gdf = indirectly_affected_zip_gdf[
    ~indirectly_affected_zip_gdf['ZCTA5'].isin(buffered_zips['ZIP'])
]

# 13. Add category columns for directly and indirectly affected ZIPs
zip_texas_closure_map['Affected_Category'] = 'Not Affected'
zip_texas_closure_map.loc[zip_texas_closure_map['Number of Closures']
                          > 0, 'Affected_Category'] = 'Directly Affected'
indirectly_affected_zips = indirectly_affected_zip_gdf['ZCTA5'].unique()
zip_texas_closure_map.loc[
```

```
    zip_texas_closure_map['ZIP'].isin(indirectly_affected_zips),
    'Affected_Category'
] = 'Indirectly Affected'

# 14. Create a color map for the categories
# Not Affected, Indirectly Affected, Directly Affected
colors = ['#E6E6E6', '#FFA500', '#FF0000']
cmap = ListedColormap(colors)

# 15. Map categories to integer codes for plotting
zip_texas_closure_map['Category_Code'] =
 ↪  zip_texas_closure_map['Affected_Category'].map({
    'Not Affected': 0,
    'Indirectly Affected': 1,
    'Directly Affected': 2
})

# 16. Plot the data
fig, ax = plt.subplots(1, 1, figsize=(15, 15))
zip_texas_closure_map.plot(column='Category_Code', cmap=cmap, linewidth=0.8,
                           edgecolor='0.8', ax=ax, legend=True)

# Customize the legend
legend_elements = [
    Patch(facecolor='#E6E6E6', edgecolor='black', label='Not Affected'),
    Patch(facecolor='#FFA500', edgecolor='black', label='Indirectly
 ↪  Affected'),
    Patch(facecolor='#FF0000', edgecolor='black', label='Directly Affected')
]
ax.legend(handles=legend_elements,
          title='Affected Categories', loc='lower right')

# Add title and remove axis
plt.title('Texas ZIP Codes Affected by Hospital Closures (2016-2019)',
 ↪  fontsize=16)
plt.axis('off')
plt.show()

# Summary statistics
print("Summary of Affected ZIP Codes:")
print(zip_texas_closure_map['Affected_Category'].value_counts())
```




```
Summary of Affected ZIP Codes:
Affected_Category
Not Affected           1430
Indirectly Affected     477
Directly Affected        28
Name: count, dtype: int64
```

Reflecting on the exercise (10 pts)

(Partner 1) The method we use to identify closures is in fact imperfect. For cases where hospitals rebranded, for example, there isn't actually a merger or closure occurring, but the data will suggest that the old name for the hosptial closed, essentially marking down a closure when there was just a rebranding. Similarly, hospitals may delay in reporting information about their active status due to simple human error. Furthermore, there are hospitals that may only temporarily be closed, and so adding them to the closure list is innaccurate given enough time.

I think one of the best solutions we can employ is to analyze data over a longer period of time for more consistency. We may also consider writing contingencies that allow us to find hospitals that have rebranded. For example, if a hospital has location and provider information identical to a previously closed hospital, perhaps there are times where this is more accurate to flag as a rebrand rather than a merger. It may be useful to cross-reference our data with our databases that track whether a hospital can rebrand or whether their closure is temporary or permanent.

(Partner 2) What our zipcode analysis lacks is nuance. While we analyze which areas close and how this can affect other areas based on travel time, we don't know exactly what kinds of hospitals are closing. Let's say a relatively underutilized hospital closes but the most important facilites are still open. Let's take another case where one of the most visited facilities is the one to close. In our data, this difference isn't detected, but in real life, that difference is crucial. In the former case, relatively few individuals may be affected. In the latter, this may be disastrous for nearby citizens' access to healthcare. Perhaps some additional data analysis related to hospital type or average patients received can help us further detect how impactful closures within zipcodes will be.