\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): Yuan Qi, yuanqi • Partner 2 (name and cnet ID): Aurora Zhang, zhanght
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: **_AZ, YQ_**
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: **_2_** Late coins left after submission: **_2_**
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

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

```python
# SEtup
import pandas as pd
import altair as alt
import geopandas as gpd
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from shapely.geometry import Point
import time
```

1.

The variables we pulled: PRVDR_CTGRY_SBTYP_CD, PRVDR_CTGRY_CD, FAC_NAME, PRVDR_NUM, PGM_TRMNTN_CD, ZIP_CD

2.

a.

```python
# Load the re-uploaded CSV file to inspect and filter based on the specified criteria
pos2016 = pd.read_csv(r'C:\Users\Aurora\Desktop\pos2016.csv')
```

```python
# Display the first few rows to understand its structure and confirm the presence of relevant columns
pos2016.head()

# Filter the shortterm hospitals
short_term_2016 = pos2016[
    (pos2016['PRVDR_CTGRY_CD'] == 1) & (pos2016['PRVDR_CTGRY_SBTYP_CD'] == 1)
]

# Calculte the number of unique hospitals
unique_2016 = short_term_2016['PRVDR_NUM'].nunique()
print("Unique short-term hospitals in 2016:", unique_2016)
```

```
Unique short-term hospitals in 2016: 7245
```

For the whole country, thousands of short term hospitals sounds reasonable. But when I compared with other data source, seems this data is too high. The comparison with other data sources related to hospitals are described in question b as below.

b.

Although I didn’t find other data sources about the number of short term hospitals, based on data from the American Hospital Association (AHA), there were approximately 5,157 community hospitals in the United States in 2021(which may include short-term general hospitals, specialty hospitals, and teaching hospitals, which represent the majority of facilities accessible to the public). The number of 7,245 short-term hospitals calculated here is much higher than AHA’s figure.

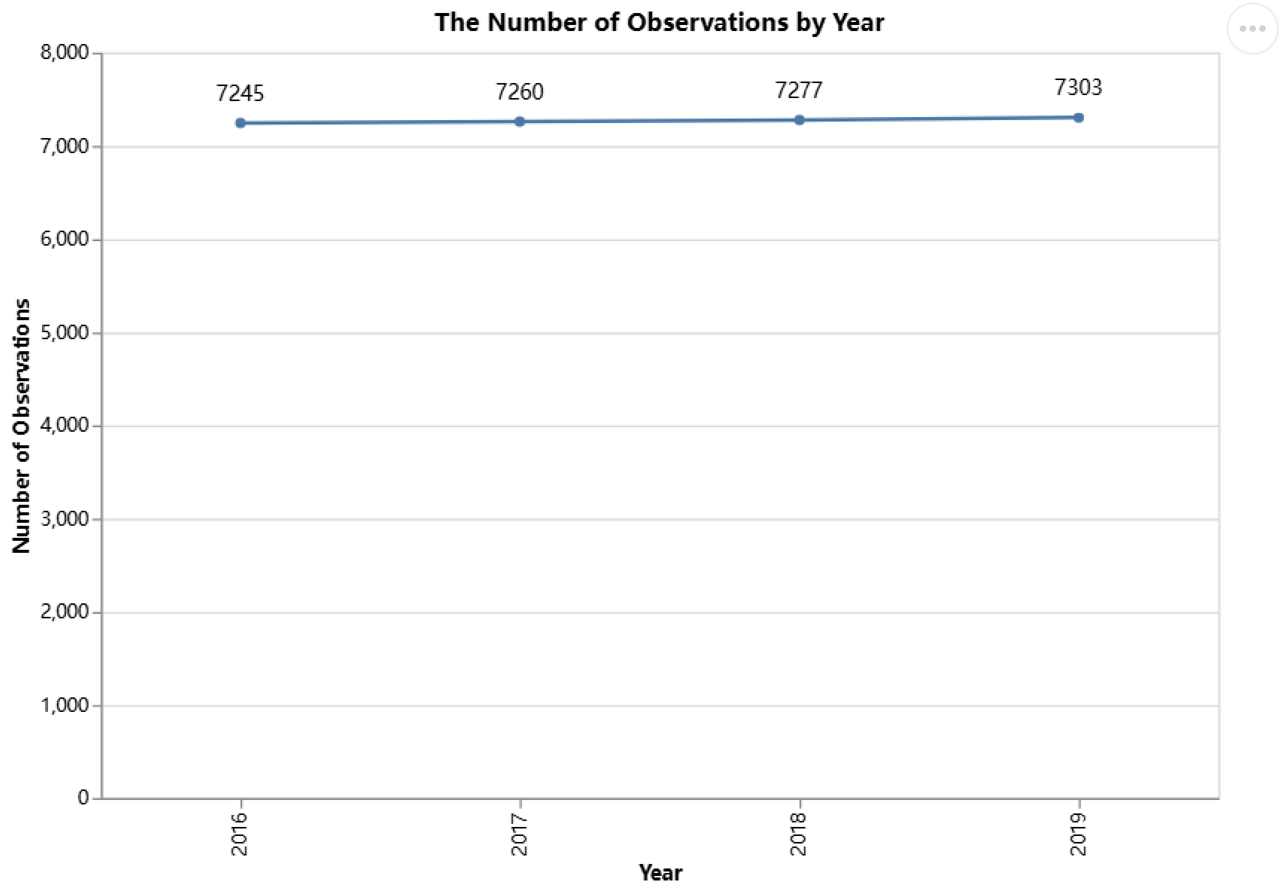
Some reasons of the difference are as below: 1) The difference may because the definition difference, the short term hospital here may contain more kinds of specialized facilities under the short-term designatio that AHA doesn’t include. 2) Different data sources may use varying methodologies to record and classify hospital data, leading to inconsistent numbers.

3.

```python
# Import the dataset of 2016-2019
pos2016 = pd.read_csv(r'C:\Users\Aurora\Desktop\pos2016.csv', encoding='latin1')
pos2017 = pd.read_csv(r'C:\Users\Aurora\Desktop\pos2017.csv', encoding='latin1')
pos2018 = pd.read_csv(r'C:\Users\Aurora\Desktop\pos2018.csv', encoding='latin1')
pos2019 = pd.read_csv(r'C:\Users\Aurora\Desktop\pos2019.csv', encoding='latin1')

# Add the column Year
pos2016['Year'] = 2016
pos2017['Year'] = 2017
pos2018['Year'] = 2018
pos2019['Year'] = 2019

pos2016['Year'] = pos2016['Year'].astype(int)
pos2017['Year'] = pos2017['Year'].astype(int)
pos2018['Year'] = pos2018['Year'].astype(int)
pos2019['Year'] = pos2019['Year'].astype(int)

# Set a function to repeat the calculation for dataset pos2016.csv
def count_hospitals(data, year):
    # Filter the short-term hospitals
    short_term_hospitals = data[
        (data['PRVDR_CTGRY_CD'] == 1) & (data['PRVDR_CTGRY_SBTYP_CD'] == 1)
    ]
    # Calculate the count of obsevations
    hospital_count = short_term_hospitals['PRVDR_NUM']
    print(f"Unique short-term hospitals in {year}:", hospital_count)
    return hospital_count, short_term_hospitals

# Return he result of the number of obsevations
obs_2016, short_term_2016 = count_hospitals(pos2016, 2016)
obs_2017, short_term_2017 = count_hospitals(pos2017, 2017)
obs_2018, short_term_2018 = count_hospitals(pos2018, 2018)
obs_2019, short_term_2019 = count_hospitals(pos2019, 2019)

# Save all results in a distionary
obs_by_year = {
    2016: obs_2016,
    2017: obs_2017,
    2018: obs_2018,
    2019: obs_2019
}

# Append all 4 datasets
all_data = pd.concat([short_term_2016, short_term_2017, short_term_2018, short_term_2019])
all_data['Year'] = all_data['Year'].astype(int)

# Calculate the oberservations by year
yearly_counts = all_data.groupby('Year')['PRVDR_NUM'].size().reset_index(name='The_Number_of_Observati

# Plot the result using altair
chart1 = alt.Chart(yearly_counts).mark_line(point=True).encode(
    x=alt.X('Year:O', title='Year'),
    y=alt.Y('The_Number_of_Observations:Q', title='Number of Observations'),
    tooltip=['Year', 'The_Number_of_Observations']
).properties(
    title='The Number of Observations by Year',
    width=600,
    height=400
)

# Adding labels
text = chart1.mark_text(
    align='center',
    baseline='bottom',
    dy=-10
).encode(
    text='The_Number_of_Observations:Q'
)

# Combine line chart and data labels
chart1_with_label = (chart1 + text)
chart1_with_label.show()
```

```
Unique short-term hospitals in 2016: 0         010001
1          010004
2          010005
3          010006
4          010007
            ...
133526     670114
```

```
133527    670115
133528    670116
133529    670117
133530    670118
Name: PRVDR_NUM, Length: 7245, dtype: object
Unique short-term hospitals in 2017: 0         010001
1         010004
2         010005
3         010006
4         010007
           ...
135471    670118
135472    670119
135473    670120
135474    670121
135475    670122
Name: PRVDR_NUM, Length: 7260, dtype: object
Unique short-term hospitals in 2018: 0         010001
1         010004
2         010005
3         010006
4         010007
           ...
137567    670121
137568    670122
137569    670124
137570    670125
137571    670126
Name: PRVDR_NUM, Length: 7277, dtype: object
Unique short-term hospitals in 2019: 0         010001
1         010004
2         010005
3         010006
4         010007
           ...
139514    670126
139515    670127
139516    670128
139517    670129
139518    670130
Name: PRVDR_NUM, Length: 7303, dtype: object
```



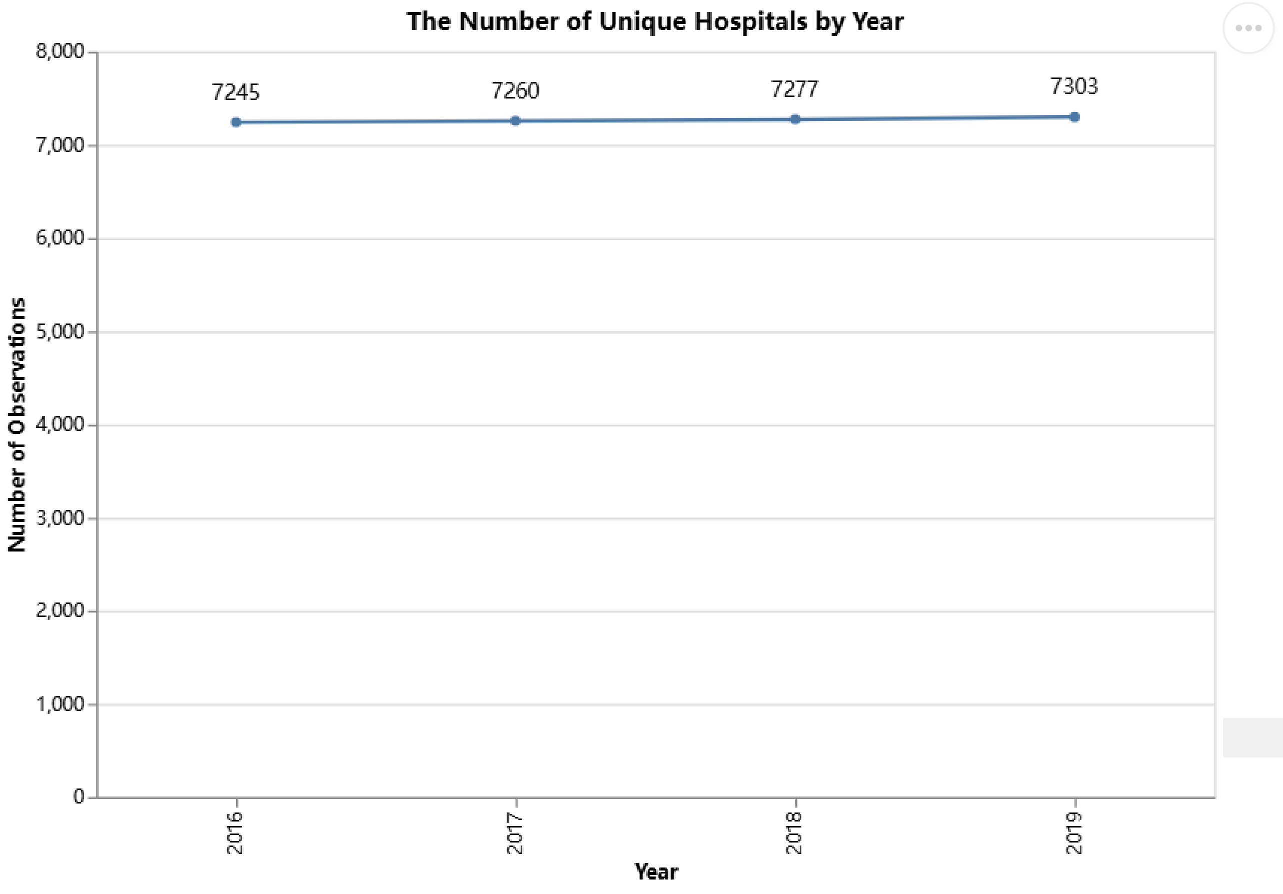
4.

a.

```
# Calculation the unique hospitals by year
yearly_unique_hospitals = all_data.groupby('Year')['PRVDR_NUM'].nunique().reset_index(name='The_Number
print(yearly_unique_hospitals)

# Plot the result using altair
chart2 = alt.Chart(yearly_unique_hospitals).mark_line(point=True).encode(
    x=alt.X('Year:O', title='Year'),
    y=alt.Y('The_Number_of_Unique_Hospitals:Q', title='Number of Observations'),
    tooltip=['Year', 'The_Number_of_Unique_Hospitals']
).properties(
    title='The Number of Unique Hospitals by Year',
    width=600,
    height=400
)

# Adding labels
text = chart2.mark_text(
    align='center',
    baseline='bottom',
    dy=-10
).encode(
    text='The_Number_of_Unique_Hospitals:Q'
)

# Combine line chart and data labels
```

```
chart2_with_label = (chart2 + text)
chart2_with_label
```

```
   Year  The_Number_of_Unique_Hospitals
0  2016                            7245
1  2017                            7260
2  2018                            7277
3  2019                            7303
```



b.

According to the output, we can see that the two plots show no difference for the years 2016, 2017, 2018, and 2019. In each of these years, the count of unique hospitals matches the total hospital counts calculated in the previous question.

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

1.

```
# Filter 2016 active providers (PGM_TRMNTN_CD == 0 means active provider)
active_2016 = short_term_2016[short_term_2016['PGM_TRMNTN_CD'] == 0]

# Create sets of PRVDR_NUM for each year
cms_2017 = set(short_term_2017['PRVDR_NUM'])
cms_2018 = set(short_term_2018['PRVDR_NUM'])
cms_2019 = set(short_term_2019['PRVDR_NUM'])

# Initialize suspected closures
suspected_closures = []
```

```
# Identify 2016 active hospitals that disappeared in 2017 and did not appear in 2018 or 2019
# or were terminated in 2017 (PGM_TRMNTN_CD != 0)
terminated_2017 = short_term_2017[(short_term_2017['PRVDR_NUM'].isin(active_2016['PRVDR_NUM'])) & (sho
suspected_closures_2017 = active_2016[(~active_2016['PRVDR_NUM'].isin(cms_2017) &
                                       ~active_2016['PRVDR_NUM'].isin(cms_2018) &
                                       ~active_2016['PRVDR_NUM'].isin(cms_2019)) |
                                      (active_2016['PRVDR_NUM'].isin(terminated_2017['PRVDR_NUM']))]
suspected_closures_2017['Suspected Closure Year'] = 2017

# Remove 2017 closures from active_2016
active_2016 = active_2016[~active_2016['PRVDR_NUM'].isin(suspected_closures_2017['PRVDR_NUM'])]
suspected_closures.append(suspected_closures_2017)

# Identify 2016 active hospitals that disappeared in 2018 and did not appear in 2019 or were terminate
terminated_2018 = short_term_2018[(short_term_2018['PRVDR_NUM'].isin(active_2016['PRVDR_NUM'])) & (sho
suspected_closures_2018 = active_2016[(active_2016['PRVDR_NUM'].isin(cms_2017)) &
                                      (~active_2016['PRVDR_NUM'].isin(cms_2018) &
                                       ~active_2016['PRVDR_NUM'].isin(cms_2019)) |
                                      (active_2016['PRVDR_NUM'].isin(terminated_2018['PRVDR_NUM']))]
suspected_closures_2018['Suspected Closure Year'] = 2018

# Remove 2018 closures from active_2016
active_2016 = active_2016[~active_2016['PRVDR_NUM'].isin(suspected_closures_2018['PRVDR_NUM'])]
suspected_closures.append(suspected_closures_2018)

# Identify 2016 active hospitals that disappeared in 2019
# or were terminated in 2019 (PGM_TRMNTN_CD != 0)
terminated_2019 = short_term_2019[(short_term_2019['PRVDR_NUM'].isin(active_2016['PRVDR_NUM'])) & (sho
suspected_closures_2019 = active_2016[(active_2016['PRVDR_NUM'].isin(cms_2017)) &
                                      (active_2016['PRVDR_NUM'].isin(cms_2018)) &
                                      (~active_2016['PRVDR_NUM'].isin(cms_2019)) |
                                      (active_2016['PRVDR_NUM'].isin(terminated_2019['PRVDR_NUM']))]
suspected_closures_2019['Suspected Closure Year'] = 2019

# Combine all suspected closures
suspected_closures.append(suspected_closures_2019)
suspected_closures_df = pd.concat(suspected_closures, ignore_index=True)

# Print result
print("Number of suspected closed hospitals:", suspected_closures_df['PRVDR_NUM'].nunique())
```

```
Number of suspected closed hospitals: 174

C:\Users\Aurora\AppData\Local\Temp\ipykernel_5708\2575845588.py:19: 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

C:\Users\Aurora\AppData\Local\Temp\ipykernel_5708\2575845588.py:31: 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

C:\Users\Aurora\AppData\Local\Temp\ipykernel_5708\2575845588.py:44: 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
```

2.

```python
# Sort the list of suspected closures by facility name
sorted_closures = suspected_closures_df.sort_values(by='FAC_NAME')

# Select the first 10 rows and display the names and year of suspected closure
first_10_closures = sorted_closures[['FAC_NAME', 'Suspected Closure Year']].head(10)

# Print the result
print("First 10 hospitals sorted by name with their year of suspected closure:")
print(first_10_closures)
```

```
First 10 hospitals sorted by name with their year of suspected closure:
                                        FAC_NAME  Suspected Closure Year
0                           ABRAZO MARYVALE CAMPUS                    2017
1          ADVENTIST MEDICAL CENTER - CENTRAL VALLEY                    2017
73                          AFFINITY MEDICAL CENTER                    2018
15    ALBANY MEDICAL CENTER / SOUTH CLINICAL CAMPUS                    2017
29         ALLEGIANCE SPECIALTY HOSPITAL OF KILGORE                    2017
132                         ALLIANCE LAIRD HOSPITAL                    2019
147                       ALLIANCEHEALTH DEACONESS                    2019
109                    ANNE BATES LEACH EYE HOSPITAL                    2019
3           ARKANSAS VALLEY REGIONAL MEDICAL CENTER                    2017
11              BANNER CHURCHILL COMMUNITY HOSPITAL                    2017
```

3.

a.

```python
# Count the number of active hospitals by ZIP code for each year
active_2016_count = short_term_2016[short_term_2016['PGM_TRMNTN_CD'] == 0].groupby('ZIP_CD').size()
active_2017_count = short_term_2017[short_term_2017['PGM_TRMNTN_CD'] == 0].groupby('ZIP_CD').size()
active_2018_count = short_term_2018[short_term_2018['PGM_TRMNTN_CD'] == 0].groupby('ZIP_CD').size()
active_2019_count = short_term_2019[short_term_2019['PGM_TRMNTN_CD'] == 0].groupby('ZIP_CD').size()

# Merge active counts into a single DataFrame for comparison
active_counts = pd.DataFrame({
    2016: active_2016_count,
    2017: active_2017_count,
    2018: active_2018_count,
    2019: active_2019_count
}).fillna(0)
```

```
# Function to check if the number of active hospitals did not decrease in the year after the suspected
def is_active_count_stable(row):
    zip_code = row['ZIP_CD']
    year = row['Suspected Closure Year']
    if year in active_counts.columns and (year + 1) in active_counts.columns:
        return active_counts.loc[zip_code, year + 1] >= active_counts.loc[zip_code, year]
    return False

# Apply the function to filter hospitals where the active count in ZIP code did not decrease
merger_closures = sorted_closures[sorted_closures.apply(is_active_count_stable, axis=1)]

print("b. Number of hospitals potentially due to merger/acquisition:", merger_closures.shape[0])
```

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

b.

```
# Exclude potential mergers from closures_df
corrected_closures = sorted_closures[~sorted_closures['PRVDR_NUM'].isin(merger_closures['PRVDR_NUM'])]

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

# Save the cleaned closure data to a CSV file
corrected_closures.to_csv(r"C:\Users\Aurora\Desktop\corrected_closures.csv", index=False)
```

```
Number of hospitals after correcting for potential mergers: 77
```

c.

```
# Sort the corrected list by facility name
sorted_corrected_closures = corrected_closures.sort_values(by='FAC_NAME')

# Print the first 10 rows for part (c)
print("c. First 10 rows of corrected hospital closures:")
print(sorted_corrected_closures[['FAC_NAME', 'ZIP_CD', 'Suspected Closure Year']].head(10))

# Save the corrected closures to a CSV file
sorted_corrected_closures[['FAC_NAME', 'ZIP_CD', 'Suspected Closure Year']].to_csv('corrected_closures
```

```
c. First 10 rows of corrected hospital closures:
                                             FAC_NAME   ZIP_CD  \
132                             ALLIANCE LAIRD HOSPITAL  39365.0
147                            ALLIANCEHEALTH DEACONESS  73112.0
109                       ANNE BATES LEACH EYE HOSPITAL  33136.0
153                       BARIX CLINICS OF PENNSYLVANIA  19047.0
172                     BAYLOR EMERGENCY MEDICAL CENTER  75087.0
170  BAYLOR SCOTT & WHITE EMERGENCY MEDICAL CENTER ...  78613.0
145                           BELMONT COMMUNITY HOSPITAL  43906.0
135                               BIG SKY MEDICAL CENTER  59716.0
134                  BLACK RIVER COMMUNITY MEDICAL CENTER  63901.0
```

```
160                      CARE REGIONAL MEDICAL CENTER  78336.0

     Suspected Closure Year
132                    2019
147                    2019
109                    2019
153                    2019
172                    2019
170                    2019
145                    2019
135                    2019
134                    2019
160                    2019
```

Download Census zip code shapefile (10 pt)

1.

a.

The five file types are .dbf, .prj, .shp, .shx, .xml. **.shp (Shapefile Format):** Stores the geometry of spatial features, such as points, lines, and polygons. This is the primary file for representing the spatial shape data. **.shx (Shape Index Format):** Contains an index of the feature geometry to allow for faster access. It acts as a companion to .shp by organizing and indexing the shapes for efficient retrieval. **.dbf (Database File):** Holds attribute data for each spatial feature in tabular form. This file is used to store non-spatial information (like names, IDs, and categories) associated with the features. **.prj (Projection File):** Defines the coordinate system and projection information for the spatial data, ensuring it is aligned and interpreted correctly on a map. **.xml (Metadata File):** Contains metadata about the dataset, which can include details on its source, creation, and content descriptions, often in a structured XML format.

For the size of files: The most useful files are .shp and .dbf, as .shp file contains the geometries, which is the largest, 817915KB; .dbf file is the second largest, 6275KB, which stores attribute data; Other three files are much less in size, .shx is 259KB; .xml is 16KB; .prj is only 1KB.

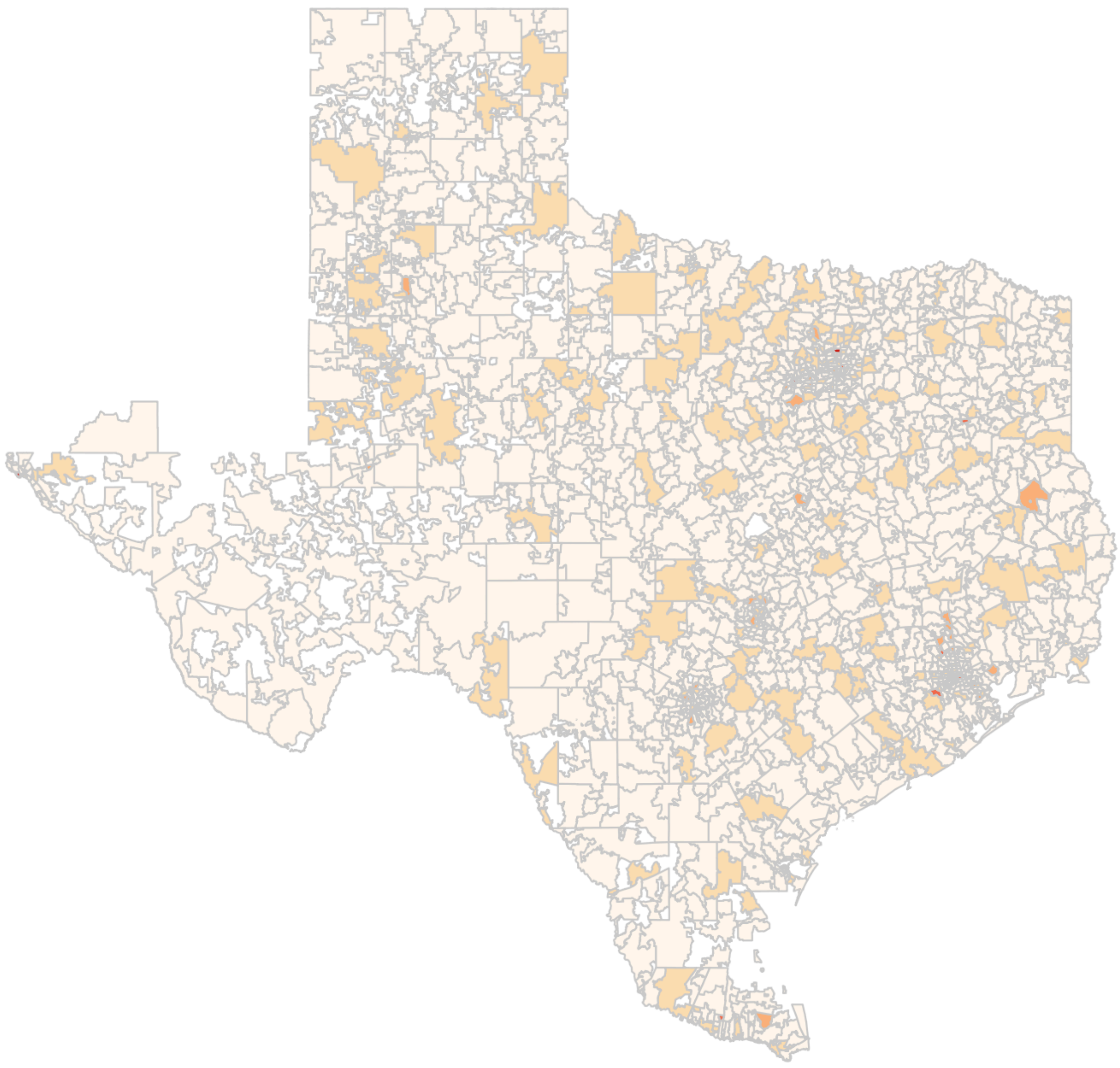
b.

```python
# Load the zipcode shapefile
zip_codes = gpd.read_file(r"C:\Users\Aurora\Desktop\gz_2010_us_860_00_500k\gz_2010_us_860_00_500k.shp"

print(zip_codes.columns)

# Restrict to Texas ZIP codes, starting with '75', '76', '77', '78' and '79'
texas_zip_codes = zip_codes[zip_codes['ZCTA5'].str.startswith(('75', '76', '77', '78','79'))]

# Calculate the number of hospitals per ZIP code
hospital_counts = active_2016.groupby('ZIP_CD')['PRVDR_NUM'].size().reset_index()
hospital_counts.columns = ['ZIP_CD', 'Number_of_Hospitals']

# Merge the Texas zip codes data and hospital counts data
texas_zip_codes = texas_zip_codes.rename(columns={'ZCTA5': 'ZIP_CD'})
texas_zip_codes['ZIP_CD'] = texas_zip_codes['ZIP_CD'].astype(str)
hospital_counts['ZIP_CD'] = hospital_counts['ZIP_CD'].apply(lambda x: str(int(float(x))))
hospital_counts['ZIP_CD'] = hospital_counts['ZIP_CD'].astype(str)
```

```
texas_hospitals_map = texas_zip_codes.merge(hospital_counts, on='ZIP_CD', how='left')
texas_hospitals_map['Number_of_Hospitals'] = texas_hospitals_map['Number_of_Hospitals'].fillna(0)

# Plot the choropleth
fig, ax = plt.subplots(1, 1, figsize=(12, 10))
texas_hospitals_map.plot(column='Number_of_Hospitals', cmap='OrRd', linewidth=0.8, ax=ax, edgecolor='0
ax.set_title("Number of Hospitals by ZIP Code in Texas (2016)")
ax.set_axis_off()
plt.show()
```

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

Number of Hospitals by ZIP Code in Texas (2016)



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

1.

```
# Create a new column for the centroid of each ZIP code area
zip_codes['centroid'] = zip_codes.geometry.centroid

# Create a GeoDataFrame for the centroids with relevant columns
zips_all_centroids = gpd.GeoDataFrame(zip_codes, geometry='centroid')

print("Dimensions of the GeoDataFrame:", zips_all_centroids.shape)
print(zips_all_centroids.head())
```

```
C:\Users\Aurora\AppData\Local\Temp\ipykernel_5708\3551865163.py:2: UserWarning:

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



Dimensions of the GeoDataFrame: (33120, 7)
           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  \
0  POLYGON ((-72.62734 42.16203, -72.62764 42.162...
1  POLYGON ((-72.95393 42.34379, -72.95385 42.343...
2  POLYGON ((-72.68286 42.37002, -72.68287 42.369...
3  POLYGON ((-72.39529 42.18476, -72.39653 42.183...
4  MULTIPOLYGON (((-72.39191 42.08066, -72.39077 ...

                    centroid
0  POINT (-72.64107 42.21257)
1  POINT (-72.86985 42.28786)
2  POINT (-72.71162 42.35349)
3  POINT (-72.45805 42.19215)
4   POINT (-72.3243 42.09165)
```

The GeoDataFrame has 33,120 rows and 7 columns:

'GEO_ID': A unique identifier for each ZIP Code Tabulation Area (ZCTA) in the dataset, used for cross-referencing with census and geographic data.

'ZCTA5': Represents the ZIP Code Tabulation Area, which approximates U.S. Postal Service ZIP codes for analysis.

'NAME': The name of the ZCTA, often matching 'ZCTA5', providing a clear reference to the specific area.

'LSAD': Stands for "Legal/Statistical Area Description," indicating the type of area (e.g., ZCTA).

'CENSUSAREA': Shows the area of the ZCTA as measured by the Census Bureau, giving insight into its physical size.

'geometry': Contains the polygon shapes representing the boundaries of each ZCTA, used for mapping and spatial analysis.

'centroid': This shows the calculated central point of the 'geometry' polygon, used for quick spatial referencing of the ZIP code area's central location.

2.

```
texas_prefixes = ('75', '76', '77', '78', '79')
zips_texas_centroids = zips_all_centroids[zips_all_centroids['ZCTA5'].str.startswith(texas_prefixes)]

border_states_prefixes = (
    '75', '76', '77', '78', '79',
    '73', '71',
    '68', '69',
    '72',
    '88', '86', '87'
)
zips_texas_borderstates_centroids = zips_all_centroids[zips_all_centroids['ZCTA5'].str.startswith(bord

print("Number of unique ZIP codes in Texas:", zips_texas_centroids['ZCTA5'].nunique())
print("Number of unique ZIP codes in Texas and bordering states:", zips_texas_borderstates_centroids['
```

```
Number of unique ZIP codes in Texas: 1935
Number of unique ZIP codes in Texas and bordering states: 4099
```

3.

```
# Rename and convert columns for consistent merging
zips_texas_borderstates_centroids = zips_texas_borderstates_centroids.rename(columns={'ZCTA5': 'ZIP_CD
zips_texas_borderstates_centroids['ZIP_CD'] = zips_texas_borderstates_centroids['ZIP_CD'].astype(str)

# Expand hospital_counts to include Texas and bordering states
# Ensure ZIP_CD is a string type for the .str operation
hospital_counts_border_states = all_data[(all_data['Year'] == 2016) &
                                         (all_data['ZIP_CD'].astype(str).str.startswith(border_states_

# Calculate the number of hospitals per ZIP code for Texas and bordering states
hospital_counts_border_states = hospital_counts_border_states.groupby('ZIP_CD')['PRVDR_NUM'].size().re
hospital_counts_border_states.columns = ['ZIP_CD', 'Number_of_Hospitals']

# Convert ZIP_CD in hospital_counts_border_states to a consistent format
hospital_counts_border_states['ZIP_CD'] = hospital_counts_border_states['ZIP_CD'].apply(lambda x: str(
hospital_counts_border_states['ZIP_CD'] = hospital_counts_border_states['ZIP_CD'].astype(str)

# Perform an inner merge to keep only ZIP codes with at least 1 hospital in 2016
zips_withhospital_centroids = zips_texas_borderstates_centroids.merge(
    hospital_counts_border_states[hospital_counts_border_states['Number_of_Hospitals'] > 0],
    on='ZIP_CD',
    how='inner'
)

print("Dimensions of zips_withhospital_centroids:", zips_withhospital_centroids.shape)
print(zips_withhospital_centroids.head())
```

```
Dimensions of zips_withhospital_centroids: (850, 8)
           GEO_ID ZIP_CD   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  \
0  POLYGON ((-96.77784 42.01708, -96.7818 42.0170...
1  POLYGON ((-96.55082 42.27773, -96.54103 42.277...
2  POLYGON ((-96.08154 41.32116, -96.08142 41.322...
3  POLYGON ((-96.08154 41.22154, -96.08152 41.223...
4  POLYGON ((-96.15786 41.24854, -96.15758 41.248...

                     centroid  Number_of_Hospitals
0  POINT (-96.74046 42.10671)                    1
1  POINT (-96.47116 42.23563)                    1
2  POINT (-96.05106 41.36825)                    2
3   POINT (-96.05159 41.2353)                    1
4  POINT (-96.19589 41.23423)                    1
```

I used an inner merge to combine the data. I changed the ZCTA5 column from the zips_texas_borderstates_centroids GeoDataFrame to ZIP_CD to ensure consistency for merging. The merge was then performed on the ZIP_CD variable to align it with the hospital_counts_border_states DataFrame, which also uses ZIP_CD for ZIP codes.

4.

a.

```python
subset_zips_texas_centroids = zips_texas_centroids.rename(columns={'ZCTA5': 'ZIP_CD'})
subset_zips_texas_centroids['ZIP_CD'] = subset_zips_texas_centroids['ZIP_CD'].astype(str)
zips_withhospital_centroids = zips_withhospital_centroids.rename(columns={'ZCTA5': 'ZIP_CD'})
zips_withhospital_centroids['ZIP_CD'] = zips_withhospital_centroids['ZIP_CD'].astype(str)

# Subset to 10 ZIP codes from zips_texas_centroids
subset_zips_texas_centroids = subset_zips_texas_centroids.head(10)

# Function to calculate the minimum distance from each ZIP code to ZIPs with hospitals
def calculate_min_distance(centroid, hospital_centroids):
    return hospital_centroids.distance(centroid).min()

# Start timing the operation
start_time = time.time()

# Calculate the minimum distance for each ZIP code in the subset
subset_zips_texas_centroids['Min_Distance'] = subset_zips_texas_centroids['centroid'].apply(
    lambda x: calculate_min_distance(x, zips_withhospital_centroids['centroid'])
)

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

print("Time taken for 10 ZIP codes:", time_taken, "seconds")
print("Estimated time for the entire dataset:", time_taken * (len(zips_texas_centroids) / 10), "second
print(subset_zips_texas_centroids[['ZIP_CD', 'Min_Distance']])
```

```
C:\Users\Aurora\AppData\Local\Temp\ipykernel_5708\945730530.py:11: 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.


Time taken for 10 ZIP codes: 0.40346360206604004 seconds
Estimated time for the entire dataset: 78.07020699977875 seconds
     ZIP_CD  Min_Distance
9207  78624      0.000000
9208  78626      0.000000
9209  78628      0.110844
9210  78631      0.337251
9211  78632      0.164115
9212  78633      0.174639
9213  78634      0.114657
9214  78635      0.169356
9215  78636      0.000000
9216  78638      0.164065
```

b.

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

# Calculate the minimum distance for each ZIP code in the full dataset
zips_texas_centroids['Min_Distance'] = zips_texas_centroids['centroid'].apply(
    lambda x: calculate_min_distance(x, zips_withhospital_centroids['centroid'])
)

# End timing
end_time_full = time.time()
time_taken_full = end_time_full - start_time_full

# Print results for the full dataset
print("Time taken for the full calculation:", time_taken_full, "seconds")

# Compare with the estimation
estimated_time = time_taken * (len(zips_texas_centroids) / 10)

print("Estimated time for the entire dataset:", estimated_time, "seconds")
print("Difference between actual and estimated time:", abs(time_taken_full - estimated_time), "seconds
```

```
C:\Users\Aurora\AppData\Local\Temp\ipykernel_5708\945730530.py:11: 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.


Time taken for the full calculation: 74.28131604194641 seconds
Estimated time for the entire dataset: 78.07020699977875 seconds
Difference between actual and estimated time: 3.7888909578323364 seconds
```

```
C:\Users\Aurora\Anaconda3\Lib\site-packages\geopandas\geodataframe.py:1819: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

c.

```python
# Convert the minimum distance from degrees to miles
subset_zips_texas_centroids['Min_Distance_Miles'] = subset_zips_texas_centroids['Min_Distance'] * 69

# Print the updated DataFrame
print(subset_zips_texas_centroids[['ZIP_CD', 'Min_Distance', 'Min_Distance_Miles']])
```

```
      ZIP_CD  Min_Distance  Min_Distance_Miles
9207   78624      0.000000            0.000000
9208   78626      0.000000            0.000000
9209   78628      0.110844            7.648232
9210   78631      0.337251           23.270319
9211   78632      0.164115           11.323948
9212   78633      0.174639           12.050063
9213   78634      0.114657            7.911356
9214   78635      0.169356           11.685542
9215   78636      0.000000            0.000000
9216   78638      0.164065           11.320466
```

The .prj file indicates that the unit used is “Degree”. 1 degree is approximately equal to 69 miles.

5.

a.

The unit of the original distance is in degrees, and I converted it into miles by multiplying by 69.

b.

```python
# Convert the distance from degrees to miles
zips_texas_centroids['Min_Distance_Miles'] = zips_texas_centroids['Min_Distance'] * 69

# Calculate the average distance in miles
average_distance_miles = zips_texas_centroids['Min_Distance_Miles'].mean()

# Print the average distance
print("b. The average distance to the nearest hospital for each ZIP code in Texas is approximately:",
      round(average_distance_miles, 2), "miles.")
```

```
b. The average distance to the nearest hospital for each ZIP code in Texas is approximately: 8.86
miles.

C:\Users\Aurora\Anaconda3\Lib\site-packages\geopandas\geodataframe.py:1819: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
```

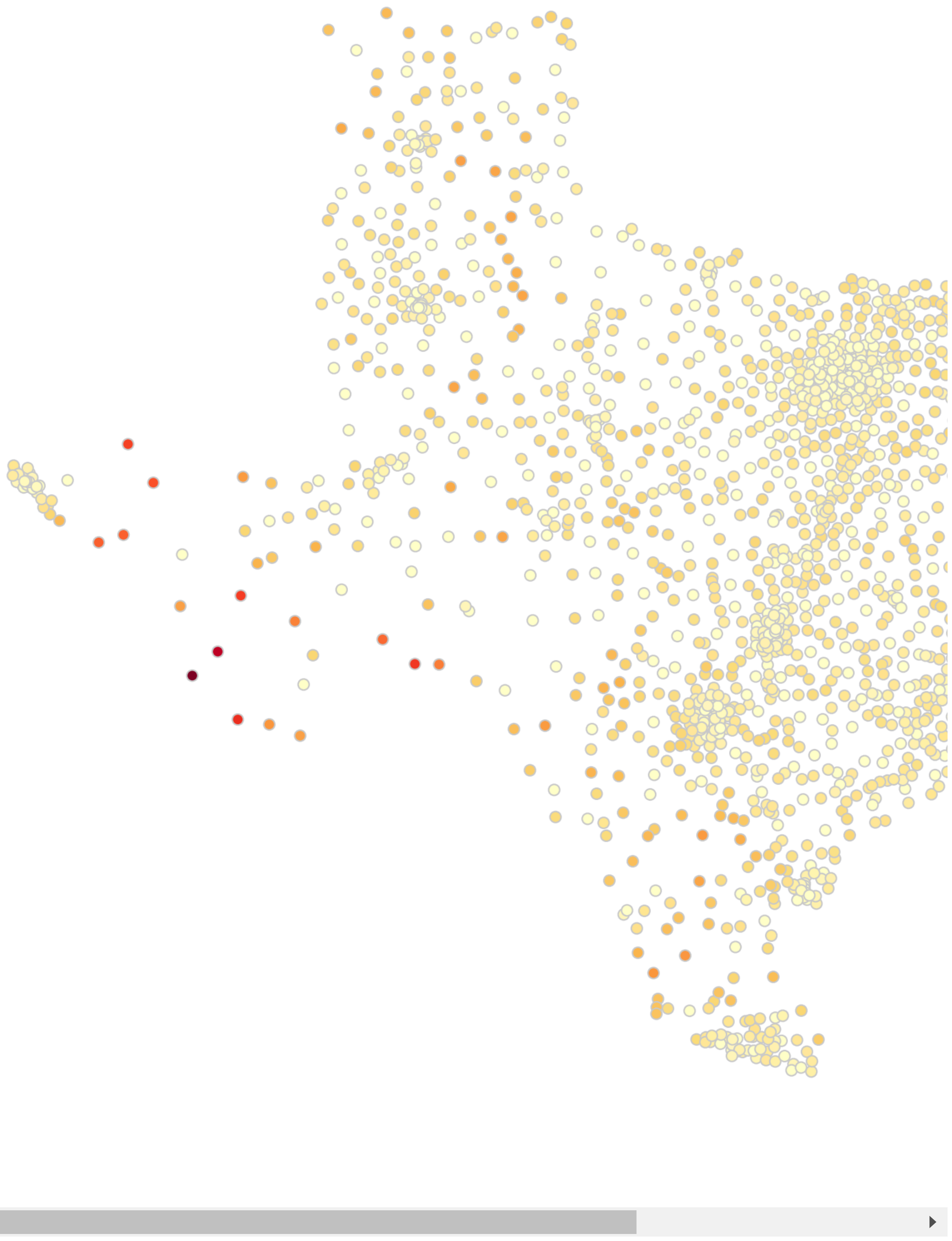
```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

The average distance of approximately 8.86 miles to the nearest hospital for each ZIP code in Texas makes sense. Urban ZIP codes would likely have shorter distances to hospitals, while rural ZIP codes would have longer distances, leading to an overall average that reflects both types of areas. The 8.86-mile distance seems to appropriately balance these variations and represents a realistic average for a state as large and diverse as Texas.

c.

```
fig, ax = plt.subplots(1, 1, figsize=(15, 12))
zips_texas_centroids.plot(column='Min_Distance_Miles', cmap='YlOrRd', linewidth=0.8, ax=ax, edgecolor=
ax.set_title("Average Distance to the Nearest Hospital for Each ZIP Code in Texas (in Miles)")
ax.set_axis_off()
plt.show()
```

Average Distance to the Nearest Hospital for Each ZIP Code in Texas (in M



Effects of closures on access in Texas (15 pts)

1.

```python
# Load the closure dataset
closures = pd.read_csv(r'C:\Users\Aurora\Desktop\corrected_closures.csv')

# Filter the data for Texas
closures_tx = closures[closures['ZIP_CD'].astype(str).str.startswith(('75', '76', '77', '78','79'))]

# Calculate the closure time for each zip code
closures_by_zip = closures_tx.groupby('ZIP_CD').size().reset_index(name='Number_of_Closures')
closures_by_zip['ZIP_CD'] = closures_by_zip['ZIP_CD'].apply(lambda x: str(int(float(x))))
closures_by_zip['ZIP_CD'] = closures_by_zip['ZIP_CD'].astype(str)
print(closures_by_zip)
```

```
   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
```

2.

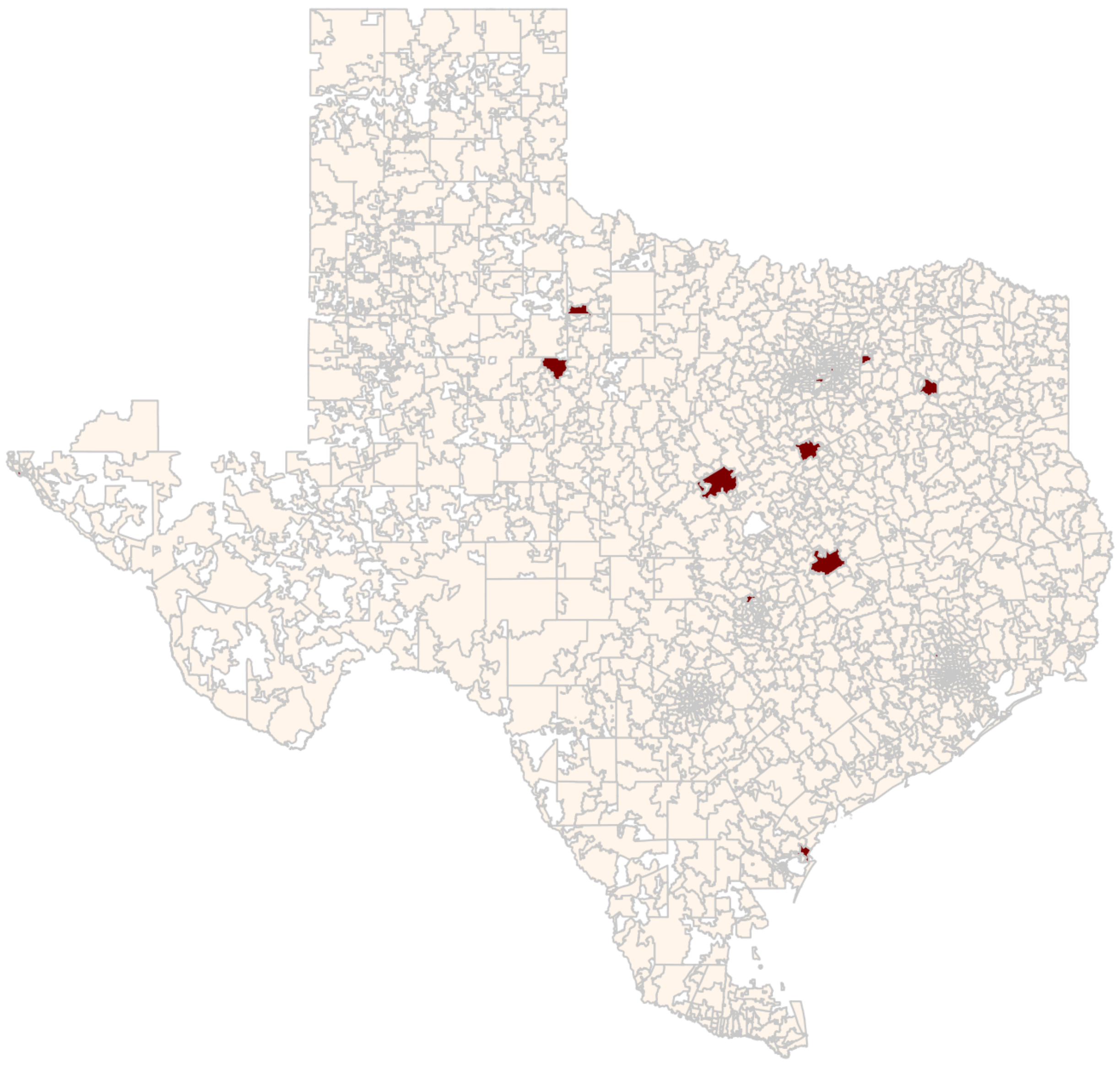
```python
# Merge the texas zip code data and closure data, and filter the data affected by closure
texas_closures_map = texas_zip_codes.merge(closures_by_zip, on='ZIP_CD', how='left')
texas_closures_map['Number_of_Closures'] = texas_closures_map['Number_of_Closures'].fillna(0)

# Filter for zip codes with at least one closure
directly_affected = texas_closures_map[texas_closures_map['Number_of_Closures'] > 0]

directly_affected.head()
```

	GEO_ID	ZIP_CD	NAME	LSAD	CENSUSAREA	geometry	Number_of_Closures
290	8600000US79520	79520	79520	ZCTA5	194.748	MULTIPOLYGON (((-100.02004 32.96007, -100.0199...	1.0
293	8600000US79529	79529	79529	ZCTA5	89.177	POLYGON ((-99.73579 33.36257, -99.73584 33.363...	1.0
334	8600000US79902	79902	79902	ZCTA5	6.558	MULTIPOLYGON (((-106.5034 31.80737, -106.50344...	1.0
407	8600000US76645	76645	76645	ZCTA5	167.466	POLYGON ((-96.97886 32.08731, -96.97926 32.087...	1.0
488	8600000US77065	77065	77065	ZCTA5	8.213	POLYGON ((-95.60365 29.90352, -95.60384 29.903...	1.0

```python
# Plot the choropleth
fig, ax = plt.subplots(1, 1, figsize=(12, 10))
```

```
texas_closures_map.plot(
    column='Number_of_Closures',
    cmap='OrRd',
    linewidth=0.8,
    ax=ax,
    edgecolor='0.8',
    legend=True
)
ax.set_title("Texas Zip Codes Affected by Hospital Closures (2016-2019)")
ax.set_axis_off()
plt.show()

# Count the number of directly affected zip codes
directly_affected_count = directly_affected['ZIP_CD'].nunique()
print("Number of directly affected ZIP codes in Texas:", directly_affected_count)
```

Texas Zip Codes Affected by Hospital Closures (2016-2019)



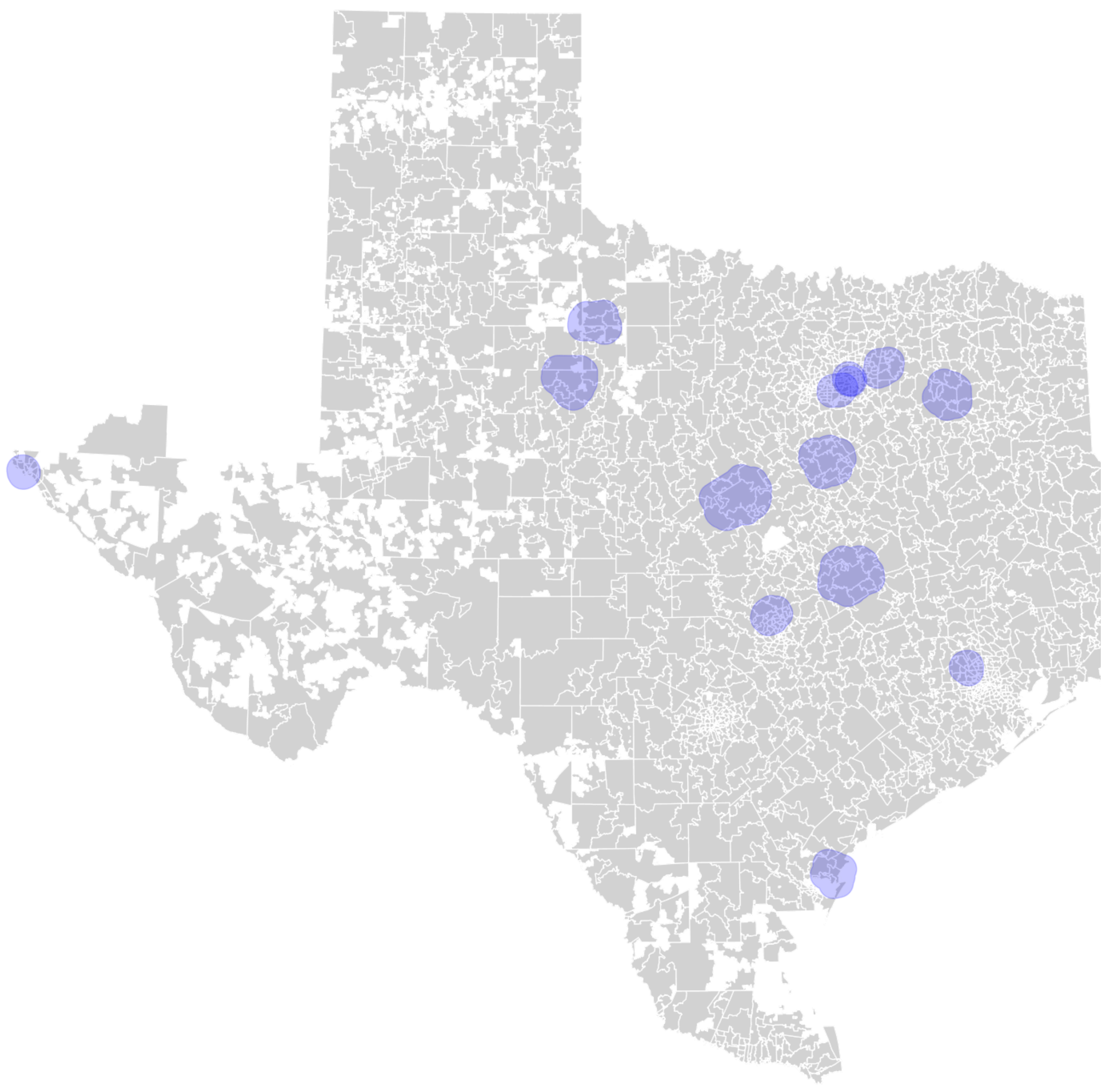
```
Number of directly affected ZIP codes in Texas: 14
```

3.

```
# Convert the projection to EPSG:3081, which uses meters as its unit
directly_affected = directly_affected.to_crs(epsg=3081)

# Create buffers with 10 miles (16093.4 meters)
buffer_gdf = directly_affected.copy()
buffer_gdf['geometry'] = directly_affected.buffer(16093.4)  # 10英里 = 16093.4米

# Make sure texas zip codes and buffer are in the same CRS
texas_zip_codes = texas_zip_codes.to_crs(buffer_gdf.crs)

# Create a plot
fig, ax = plt.subplots(1, 1, figsize=(12, 10))

# Plot texas zip codes
texas_zip_codes.plot(ax=ax, color="lightgrey", edgecolor="white", linewidth=0.5, label="Texas ZIP Code

# Plot the 10-miles buffer
buffer_gdf.plot(ax=ax, color="blue", alpha=0.2, edgecolor="blue", linewidth=0.5, label="10-Mile Buffer

# Ass title and legends to the plot
plt.title("10-Mile Buffer Around Directly Affected Texas ZIP Codes")
plt.axis("off")
plt.legend(loc="upper right")

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

```
C:\Users\Aurora\AppData\Local\Temp\ipykernel_5708\2640795348.py:23: UserWarning:

Legend does not support handles for PatchCollection instances.
See: https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html#implementing-a-custom-
legend-handler

No artists with labels found to put in legend.  Note that artists whose label start with an underscore
are ignored when legend() is called with no argument.
```

10-Mile Buffer Around Directly Affected Texas ZIP Codes

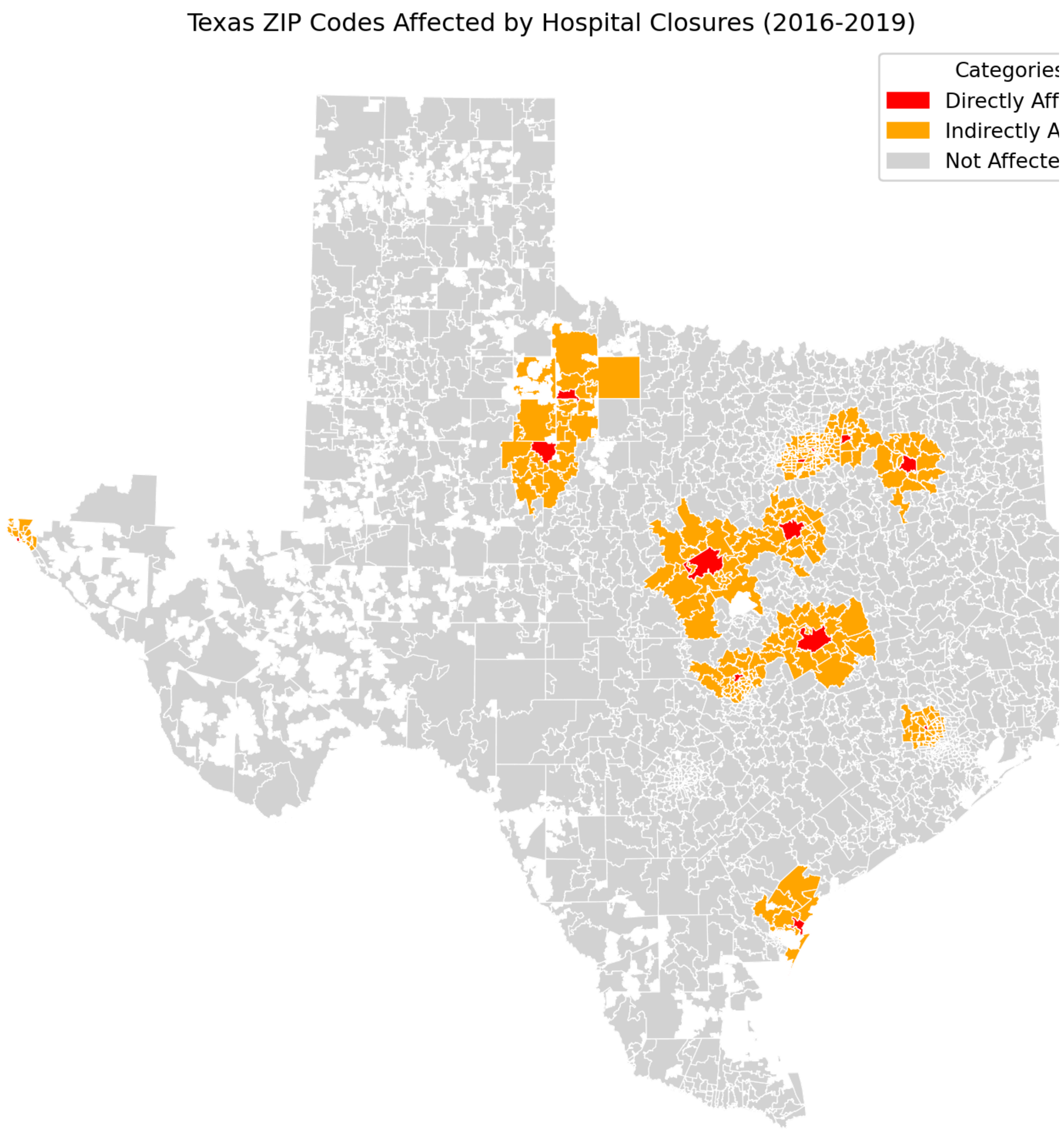


```
# Perform spatial join to find all Texas ZIP codes within the 10-mile buffer zone
indirectly_affected = gpd.sjoin(texas_zip_codes, buffer_gdf, how="inner", predicate='intersects')
indirectly_affected.head()

# Calculate the number of unique indirectly affected ZIP codes
indirectly_affected_unique = indirectly_affected[~indirectly_affected['ZIP_CD_left'].isin(directly_aff
indirectly_affected_count = indirectly_affected_unique['ZIP_CD_left'].nunique()

# Display the result
print("Number of indirectly affected ZIP codes in Texas:", indirectly_affected_count)
```

```
Number of indirectly affected ZIP codes in Texas: 343
```

4.

```python
# Categorize ZIP codes based on the three groups based on previous calculation
texas_zip_codes['Category'] = 'Not Affected'
texas_zip_codes.loc[texas_zip_codes['ZIP_CD'].isin(directly_affected['ZIP_CD']), 'Category'] = 'Direct
texas_zip_codes.loc[texas_zip_codes['ZIP_CD'].isin(indirectly_affected['ZIP_CD_left']) &
                    ~texas_zip_codes['ZIP_CD'].isin(directly_affected['ZIP_CD']), 'Category'] = 'Indir

# Create the plot
fig, ax = plt.subplots(1, 1, figsize=(14, 10))

# Plot each category with a different color
texas_zip_codes[texas_zip_codes['Category'] == 'Directly Affected'].plot(
    ax=ax, color="red", edgecolor="white", linewidth=0.5, label="Directly Affected"
)
texas_zip_codes[texas_zip_codes['Category'] == 'Indirectly Affected'].plot(
    ax=ax, color="orange", edgecolor="white", linewidth=0.5, label="Indirectly Affected"
)
texas_zip_codes[texas_zip_codes['Category'] == 'Not Affected'].plot(
    ax=ax, color="lightgrey", edgecolor="white", linewidth=0.5, label="Not Affected"
)

# Custom legend
import matplotlib.patches as mpatches
legend_labels = [
    mpatches.Patch(color='red', label='Directly Affected'),
    mpatches.Patch(color='orange', label='Indirectly Affected'),
    mpatches.Patch(color='lightgrey', label='Not Affected')
]
ax.legend(handles=legend_labels, title="Categories", loc="upper right")

# Add title
plt.title("Texas ZIP Codes Affected by Hospital Closures (2016-2019)")
plt.axis("off")

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



Reflecting on the exercise (10 pts)

1.

The potential problem: In some areas, new hospitals might open shortly after or around the same time an existing hospital closes, keeping the total number of hospitals in the ZIP code area stable. The "first-pass" method relies on a reduction in hospital count within the ZIP code to identify closures. This approach may mistakenly categorize these areas as unaffected by closures, overlooking actual closure events.

Additionally, the number of active hospitals in a ZIP code may not decrease even if a hospital closes because new hospitals could have opened within the same timeframe. This could lead to true closures being incorrectly excluded from the analysis.

Furthermore, if a hospital merges with another but relocates or changes its operations significantly, it may still be functionally closed in its original form, even if a similar entity exists under a different certification number or address.

Ways to better: Track New Hospital Openings: In addition to monitoring closures, track and flag newly opened hospitals to differentiate between actual growth in the number of hospitals within a ZIP code and the replacement of closed facilities. This adjustment would improve closure identification accuracy and reduce potential misclassification due to new hospital openings.

Match Hospital Attributes Beyond Certification Numbers: Use additional attributes such as the exact address, bed count, and ownership type to verify whether a newly listed hospital is genuinely a new establishment or the continuation of a previously closed one.

Use External Data Sources: Supplement the analysis with external data sources (e.g., state health department reports, industry publications) to confirm actual hospital closures and distinguish them from mergers or renaming events.

2.

Two potential issues: ***Distance and Travel Time to Nearest Hospital:*** The current approach overlooks whether alternative hospitals in nearby areas are accessible, failing to consider distance or travel time. Measuring the nearest hospital's distance or estimated travel time before and after closures would provide a clearer picture of access changes.

Hospital Capacity and Service Types: Not all hospitals offer the same services, so closures could lead to a reduction in specific healthcare options (e.g., emergency or specialized care). Categorizing closures by capacity and service type would better reflect the impact on healthcare availability.

To address these issues, we can

Incorporate Distance Metrics: Use GIS tools to measure and analyze changes in the distance to the nearest hospital for each ZIP code before and after a closure. This would give a clear understanding of whether closures result in a significant increase in travel distance or time.

Track Hospital Closures by Service Type and Capacity: Implement a system to identify and flag regions where closures lead to a significant reduction in available specialized healthcare services. This approach would help in understanding the broader impact of hospital closures on healthcare quality and accessibility.