PS4 v1.1: Spatial

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

Style Points (10 pts)

Please refer to the minilesson on code style **here**.

Submission Steps (10 pts)

1. This problem set is a paired problem set.
2. Play paper, scissors, rock to determine who goes first. Call that person *Partner 1*.
    - Partner 1 (name and cnet ID): Astari Raihanah (astari)
    - Partner 2 (name and cnet ID): Surya Hardiansyah (sur)
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: **AR** **SH**
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: **00** Late coins left after submission: **04**
7. Knit your `ps4.qmd` to an PDF file to make `ps4.pdf`,
    - The PDF should not be more than 25 pages. Use `head()` and re-size figures when appropriate.
8. (Partner 1): push `ps4.qmd` and `ps4.pdf` to your github repo.
9. (Partner 1): submit `ps4.pdf` via Gradescope. Add your partner on Gradescope.
10. (Partner 1): tag your submission in Gradescope

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

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

1. List of variables pulled from the dataset:

(1.) PRVDR_CTGRY_SBTYP_CD = Identifies the subtype of the provider, within the primary category. Used in reporting to show the breakdown of provider categories, mainly for hospitals and SNFs. Code 01 Indicates it is a short term hospital.

(2.) PRVDR_CTGRY_CD = Identifies the type of provider participating in the Medicare/Medicaid Program. Code 01 is for Hospital.

(3.) FAC_NAME = Name of the provider certified to participate in the Medicare and/or Medicaid programs.

(4.) PRVDR_NUM = Six or ten position identification number that is assigned to a certified provider. This is CMS Certification Number.

(5.) STATE_CD = Two-character state abbreviation.

(6.) PGM_TRMNTN_CD = Indicates the current termination status for the provider. Code 00 means that it is an active provider. Code 07 means other - provider status change, which indicates to a changes like a shift in provider ownership, significant operational restructuring, or a reclassification that does not imply the provider ceases to operate.

(7.) ZIP_CD = Five-digit ZIP code for a provider's physical address.

2.

a.

```
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

# Load the dataset
pos2016 = pd.read_csv('pos2016.csv', encoding='ISO-8859-1')
```

```
# Filter the dataset
filtered_2016 = pos2016[
    (pos2016['PRVDR_CTGRY_SBTYP_CD'] == 1) &
    (pos2016['PRVDR_CTGRY_CD'] == 1)
]

# Print the total number of records
print(f"\nTotal number of records: {len(filtered_2016)}")
```

```
Total number of records: 7245
```

Interpretation: Based on the CMS Dataset from Provider Category Subtype Code = 1 (short term hospital), Provider Category Code = 1 (hospital), and Provider Termination Code = 0 (Currently active), we found that there are 7,245 number of hospital in the US.

b. Based on the American Hospital Association (AHA) Hospital Annual Survey in 2016, the number of the US Community Hospitals are 4,840 hospitals likely stems from differences in definitions, scope, and data collection methods. The AHA report focuses on community hospitals—nonfederal, short-term general, and special hospitals—excluding federal facilities like VA hospitals and specialized ones such

as prison or military hospitals. In contrast, the CMS dataset (7,245 hospitals) includes all Medicare-certified hospitals, potentially covering a broader range of facilities, including those not counted as community hospitals. Additionally, the CMS data might include inactive or reclassified hospitals and uses certification data, while the AHA relies on survey responses, which could result in underreporting or different inclusion criteria.

3. 2017Q4 data

```
# Load the 2017Q4 data
pos2017 = pd.read_csv('pos2017.csv', encoding='ISO-8859-1')

# Filter the dataset
filtered_2017 = pos2017[
    (pos2017['PRVDR_CTGRY_SBTYP_CD'] == 1) &
    (pos2017['PRVDR_CTGRY_CD'] == 1)
]
# Print the total number of records
print(f"\nTotal number of records: {len(filtered_2017)}")
```

```
Total number of records: 7260
```

2018Q4 data

```
# Load the 2018 data with a specified encoding
pos2018 = pd.read_csv('pos2018.csv', encoding='ISO-8859-1')

# Filter the dataset
filtered_2018 = pos2018[
    (pos2018['PRVDR_CTGRY_SBTYP_CD'] == 1) &
    (pos2018['PRVDR_CTGRY_CD'] == 1)
]
# Print the total number of records
print(f"\nTotal number of records: {len(filtered_2018)}")
```

```
Total number of records: 7277
```

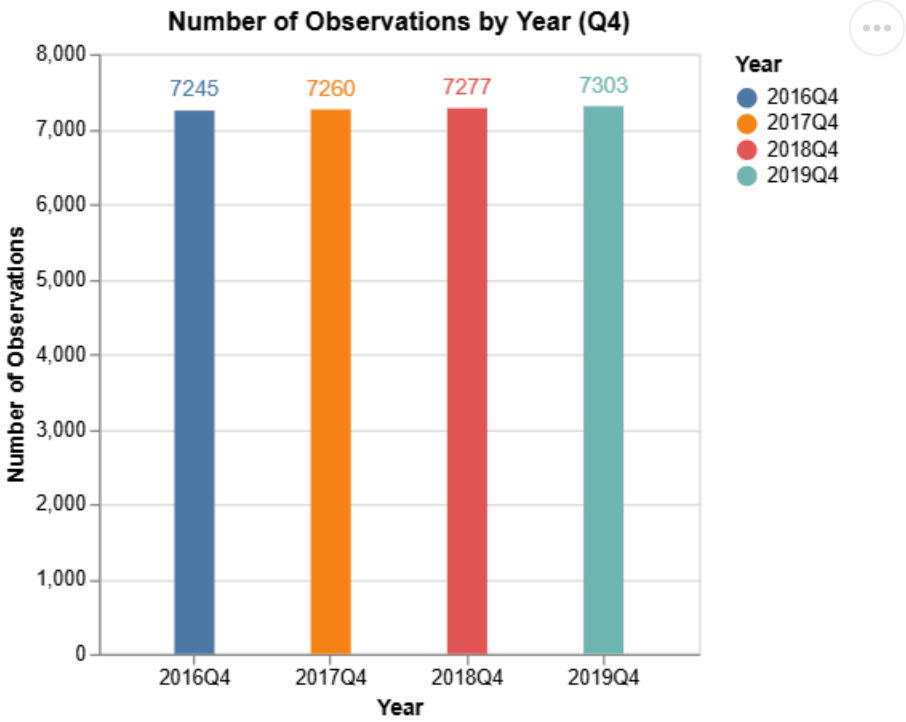
2019Q4 data

```
# Load the 2019 data
pos2019 = pd.read_csv('pos2019.csv', encoding='ISO-8859-1')

# Filter the dataset
filtered_2019 = pos2019[
    (pos2019['PRVDR_CTGRY_SBTYP_CD'] == 1) &
    (pos2019['PRVDR_CTGRY_CD'] == 1)
]
```

```
# Print the total number of records
print(f"\nTotal number of records: {len(filtered_2019)}")
```

```
Total number of records: 7303
```

```
import altair as alt

# Create a DataFrame with the number of records for each year
data = pd.DataFrame({
    'Year': ['2016Q4', '2017Q4', '2018Q4', '2019Q4'],
    'Number of Records': [7245, 7260, 7277, 7303]
})
# Create the Altair bar chart with increased spacing between the bars
chart = alt.Chart(data).mark_bar(size=20).encode(
    x=alt.X('Year', title='Year', axis=alt.Axis(
        labelAngle=0), scale=alt.Scale(padding=0.4)),
    y=alt.Y('Number of Records', title='Number of Observations'),
    color='Year'
).properties(
    title='Number of Observations by Year (Q4)',
    width=300
)
# Add text labels to the chart
text = chart.mark_text(
    align='center',
    baseline='bottom',
    dy=-5
).encode(
    text='Number of Records:Q'
)
# Combine the bar chart and the text labels
final_chart = chart + text
# Display the chart
final_chart.show()
```

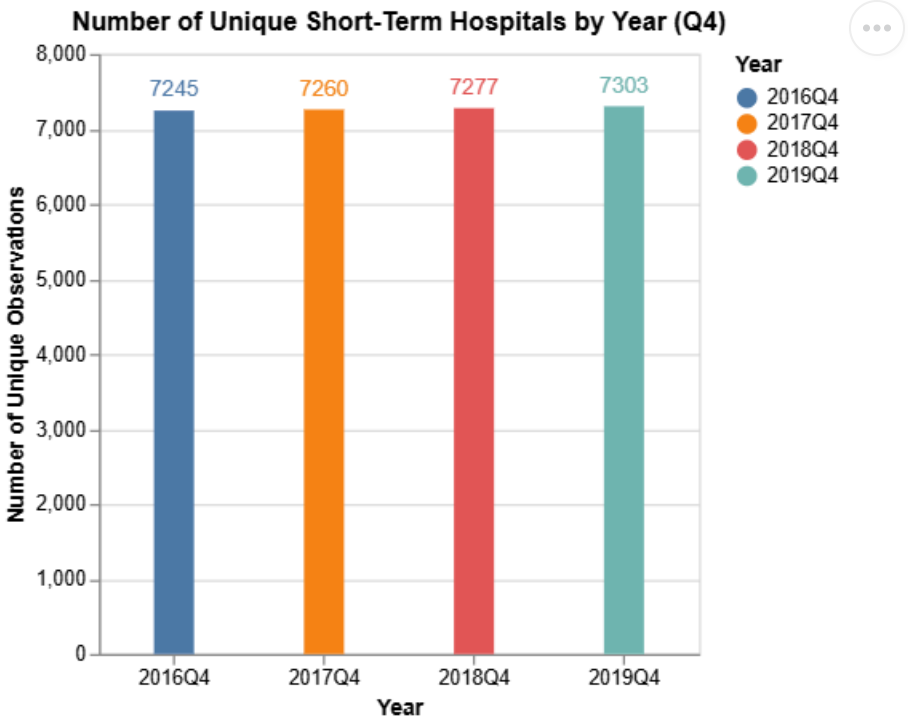


4.

a.

```
# Extract the number of unique hospitals (by CMS certification number)
unique_counts_filtered = {
    '2016Q4': filtered_2016['PRVDR_NUM'].nunique(),
    '2017Q4': filtered_2017['PRVDR_NUM'].nunique(),
    '2018Q4': filtered_2018['PRVDR_NUM'].nunique(),
    '2019Q4': filtered_2019['PRVDR_NUM'].nunique()
}
# Create a DataFrame for plotting
unique_counts_df_filtered = pd.DataFrame({
    'Year': list(unique_counts_filtered.keys()),
    'Number of Unique Hospitals': list(unique_counts_filtered.values())
})
# Plot the number of unique hospitals per year after filtering
chart_filtered = alt.Chart(unique_counts_df_filtered).mark_bar(size=20).encode(
    x=alt.X('Year', title='Year', axis=alt.Axis(labelAngle=0)),
    y=alt.Y('Number of Unique Hospitals', title='Number of Unique Observations'),
    color='Year'
).properties(
    title='Number of Unique Short-Term Hospitals by Year (Q4)',
    width = 300
)
# Add text labels to the chart
text_filtered = chart_filtered.mark_text(
    align='center',
    baseline='bottom',
    dy=-5
).encode(
```

```
    text='Number of Unique Hospitals:Q'
)
# Combine the bar chart and the text labels
unique_cms_chart = chart_filtered + text_filtered
# Display the chart
unique_cms_chart.show()
```



b. The fact that each hospital (PRVDR_NUM) appears only once per year in the dataset indicates that the data is well-structured, with no duplicate records for the same hospital. This consistency simplifies analyses, as there is no need to deduplicate when counting short-term hospitals. The identical plots for total observations and unique hospitals confirm that the filtered data accurately reflects the distinct count of short-term hospitals for each year. Overall, this means there are no overlaps or repeated entries in the dataset, ensuring reliable and straightforward comparisons year over year.

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

1. List of all active 2016 hospitals that were suspected to have closed by 2019: facility name, zip, and year of suspected closure

```
# Filtering active hospitals for each year
act_hos_2016 = pos2016[(pos2016["PGM_TRMNTN_CD"] == 0) & (pos2016["PRVDR_CTGRY_CD"] == 1)]
act_hos_2017 = pos2017[(pos2017["PGM_TRMNTN_CD"] == 0) & (pos2017["PRVDR_CTGRY_CD"] == 1)]
act_hos_2018 = pos2018[(pos2018["PGM_TRMNTN_CD"] == 0) & (pos2018["PRVDR_CTGRY_CD"] == 1)]
act_hos_2019 = pos2019[(pos2019["PGM_TRMNTN_CD"] == 0) & (pos2019["PRVDR_CTGRY_CD"] == 1)]
# Create a copy of the DataFrame to avoid SettingWithCopyWarning
act_hos_2016_copy = act_hos_2016.copy()
# Adding termination year by checking presence in subsequent years using if-else
termination_years = []
for _, row in act_hos_2016_copy.iterrows():
```

```
    if not (act_hos_2017["FAC_NAME"] == row["FAC_NAME"]).any():
        termination_years.append(2017)
    elif not (act_hos_2018["FAC_NAME"] == row["FAC_NAME"]).any():
        termination_years.append(2018)
    elif not (act_hos_2019["FAC_NAME"] == row["FAC_NAME"]).any():
        termination_years.append(2019)
    else:
        termination_years.append(None)
# Assign the termination year using .loc to avoid warnings
act_hos_2016_copy.loc[:, "TRMNTN_YR"] = termination_years
# Filtering only hospitals that have a termination year set (suspected closures)
suspected_closures = act_hos_2016_copy[act_hos_2016_copy["TRMNTN_YR"].notnull()]
# Selecting relevant columns for the result
result = suspected_closures[["FAC_NAME", "ZIP_CD", "TRMNTN_YR"]]
# Displaying the number of suspected closed hospitals
result.head()
print(f"Total 2016 hospitals suspected to have closed by 2019: {len(result)}")
```

```
Total 2016 hospitals suspected to have closed by 2019: 1078
```

2. Sort this list of hospitals by name and report the names and year of suspected closure for the first 10 rows

```
result.sort_values(by="FAC_NAME").head(10)[["FAC_NAME", "TRMNTN_YR"]]
```

	FAC_NAME	TRMNTN_YR
112137	ABILENE BEHAVIORAL HEALTH LLC	2019.0
2404	ABRAZO MARYVALE CAMPUS	2017.0
52883	ACADIA HOSPITAL, THE	2018.0
111024	ACUITY HOSPITAL OF SOUTH TEXAS	2018.0
87431	ACUTE CARE SPECIALTY HOSPITAL - AULTMAN	2018.0
78837	ADIRONDACK MEDICAL CENTER	2019.0
7237	ADVENTIST HEALTH MEDICAL CENTER TEHACHAPI VALLEY	2018.0
53866	ADVENTIST HEALTHCARE BEHAVIORAL HEALTH & WELLNESS	2019.0
53400	ADVENTIST HEALTHCARE WASHINGTON ADVENTIST HOSP...	2019.0
6498	ADVENTIST MEDICAL CENTER	2018.0

3. However, not all suspected hospital closures are true closures. For example, in the case of a merger, a CMS certification number will be appear to be “terminated,” but then the hospital re-appear under a similar name/address with a new CMS certification number in the next year. As a first pass to address this, remove any suspected hospital closures that are in zip codes where the number of active hospitals does not decrease in the year after the suspected closure.

```
# Count the number of active hospitals in each ZIP code for each year
zip_counts_2016 = act_hos_2016.groupby("ZIP_CD").size().reset_index(
```

```
    name="count_2016")
zip_counts_2017 = act_hos_2017.groupby("ZIP_CD").size().reset_index(
    name="count_2017")
zip_counts_2018 = act_hos_2018.groupby("ZIP_CD").size().reset_index(
    name="count_2018")
zip_counts_2019 = act_hos_2019.groupby("ZIP_CD").size().reset_index(
    name="count_2019")
# Merge the ZIP code counts to compare years
zip_counts = zip_counts_2016.merge(zip_counts_2017, on="ZIP_CD", how="left") \
                            .merge(zip_counts_2018, on="ZIP_CD", how="left") \
                            .merge(zip_counts_2019, on="ZIP_CD", how="left")
# Mark suspected closures that occur in ZIP codes where the number of
# active hospitals does not decrease
potential_mergers = []
for _, row in suspected_closures.iterrows():
    zip_code = row["ZIP_CD"]
    year = row["TRMNTN_YR"]

    # Check the number of active hospitals in the year of termination and
    # the following year
    if year == 2017:
        count_in_2017 = zip_counts.loc[
            zip_counts["ZIP_CD"] == zip_code, "count_2017"].values[0]
        count_in_2018 = zip_counts.loc[
            zip_counts["ZIP_CD"] == zip_code, "count_2018"].values[0]
        if count_in_2017 <= count_in_2018:
            potential_mergers.append(row["FAC_NAME"])
    elif year == 2018:
        count_in_2018 = zip_counts.loc[
            zip_counts["ZIP_CD"] == zip_code, "count_2018"].values[0]
        count_in_2019 = zip_counts.loc[
            zip_counts["ZIP_CD"] == zip_code, "count_2019"].values[0]
        if count_in_2018 <= count_in_2019:
            potential_mergers.append(row["FAC_NAME"])

# Filter out potential mergers from the suspected closures
corrected_closures = suspected_closures[~suspected_closures["FAC_NAME"].isin(
    potential_mergers)]
```

a. Among the suspected closures, how many hospitals fit this definition of potentially being a merger/acquisition?

```
# Answer
print(f"{len(potential_mergers)}")
```

```
552
```

Interpretation: There are 552 hospitals potentially merged or having an acquisition.

b. After correcting for this, how many hospitals do you have left?

```
# Answer
print(f"{len(corrected_closures)}")
```

```
525
```

Interpretation: We have 525 hospital remaining after correction

c. Sort this list of corrected hospital closures by name and report the first 10 rows.

```
# Sort and report the first 10 rows of the corrected list
corrected_closures.sort_values(
    by="FAC_NAME")[["FAC_NAME", "ZIP_CD", "TRMNTN_YR"]].head(10)
```

	FAC_NAME	ZIP_CD	TRMNTN_YR
112137	ABILENE BEHAVIORAL HEALTH LLC	79608.0	2019.0
2404	ABRAZO MARYVALE CAMPUS	85031.0	2017.0
111024	ACUITY HOSPITAL OF SOUTH TEXAS	78212.0	2018.0
78837	ADIRONDACK MEDICAL CENTER	12983.0	2019.0
53866	ADVENTIST HEALTHCARE BEHAVIORAL HEALTH & WELLNESS	20850.0	2019.0
53400	ADVENTIST HEALTHCARE WASHINGTON ADVENTIST HOSP...	20912.0	2019.0
53851	ADVENTIST REHABILITATION HOSPITAL OF MARYLAND	20850.0	2019.0
86758	AFFINITY MEDICAL CENTER	44646.0	2018.0
110054	ALLEGIANCE SPECIALTY HOSPITAL OF KILGORE	75662.0	2017.0
66094	ALLIANCE LAIRD HOSPITAL	39365.0	2019.0

III Download Census zip code shapefile (10 pt)

1.

a. The five file types in a shapefile are: shp (Shape Format): The main file that stores the geometry data of the features (e.g., points, lines, polygons).

shx (Shape Index Format): An index file that allows for quick access to the geometry data in the .shp file.

dbf (Attribute Format): A dBASE table that contains attribute data for each shape. This file links spatial data (in the .shp) to non-spatial attributes (e.g., ZIP code details).

prj (Projection Format): Contains the coordinate system and projection information, defining how the spatial data is represented on a flat surface.

xml (Metadata Format): Provides metadata about the shapefile, such as the content description, creation date, and other details.

b. The file sizes are typically shown when unzipping the archive. The file sizes are as follows:

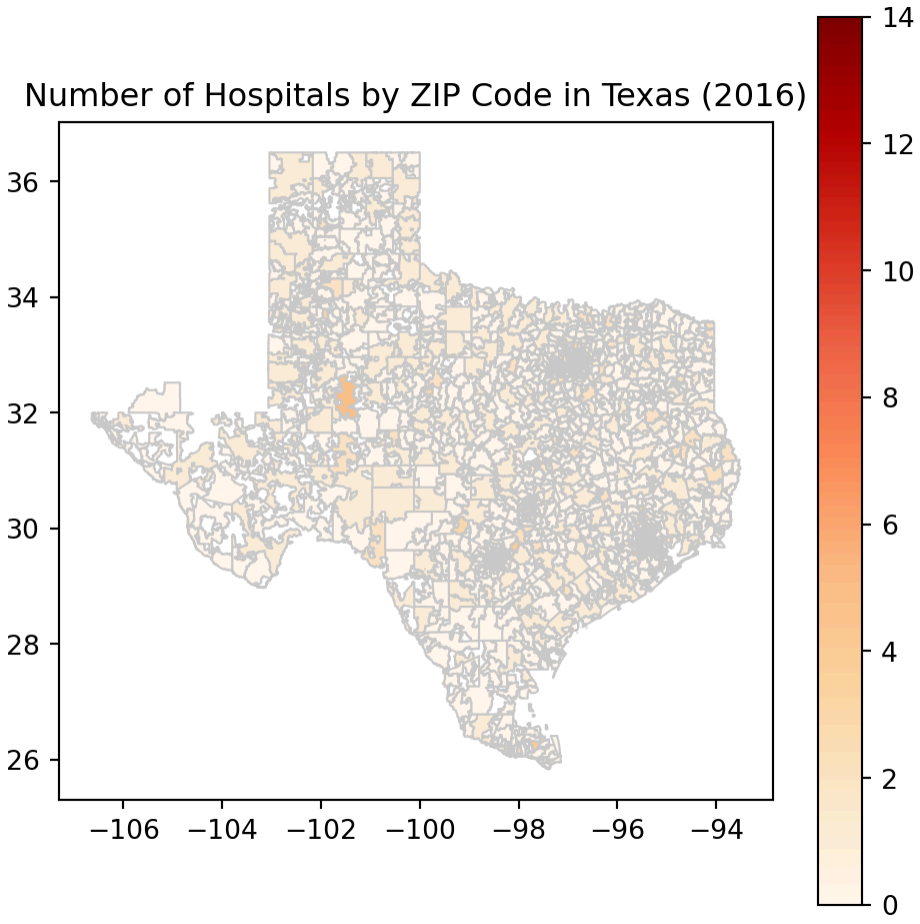
```
shp: 817,915 KB (the largest file containing geometric data)

shx: 259 KB (index file)

dbf: 6,275 KB (attribute data)

prj: 1 KB (projection information)

xml: 16 KB (metadata)
```

2.

```
# Install supporting packages
import geopandas as gpd
import matplotlib.pyplot as plt
```

```
# Load the shapefile
zip_shapefile = gpd.read_file("gz_2010_us_860_00_500k.shp")
```

```
# Filter for Texas ZIP codes (starting with 733 and 75-79)
texas_prefixes = ("733","75", "76", "77", "78", "79")
texas_zip_shapefile = zip_shapefile[zip_shapefile["ZCTA5"].str.startswith(
    texas_prefixes)]
# Group by ZIP code to count the number of hospitals per ZIP code
hospitals_per_zip = act_hos_2016.groupby("ZIP_CD").size().reset_index(
    name="hospital_count")
# Merge the Texas ZIP shapefile with the hospital counts
texas_zip_shapefile["ZIP_CD"] = texas_zip_shapefile["ZCTA5"].astype(int)
texas_hospitals_map = texas_zip_shapefile.merge(
    hospitals_per_zip, on="ZIP_CD", how="left").fillna(0)
# Plot the choropleth map
fig, ax = plt.subplots(1, 1, figsize=(6, 6))
texas_hospitals_map.plot(column="hospital_count",
cmap="OrRd", linewidth=0.8, ax=ax, edgecolor="0.8", legend=True)
plt.title("Number of Hospitals by ZIP Code in Texas (2016)")
plt.show()
```




```
# Show Texas zip codes by hospital count in desccending order
texas_hospitals_map.sort_values("hospital_count",
ascending=False).head()[["ZCTA5","hospital_count"]]
```

	ZCTA5	hospital_count
472	77030	14.0
1587	76104	13.0
903	75701	9.0
334	79902	9.0
1397	75235	7.0

Plot interpretation: The map shows that most Texas areas based on the ZIP codes have a low number of hospitals, which can be seen from the majority of lighter shades in the choropleth. Only a few areas are having a higher number of hospitals (darker shades), such as zip code 77030 (eastern Houston, Texas), and 76104 (Fort Worth, Texas). The concentration of hospitals in specific ZIP codes highlights potential disparities in healthcare access, with more rural areas potentially facing challenges in reaching medical facilities compared to urban centers.

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

1.

```
# Create centroid GeoDataFrame of zip_shapefile
zips_all_centroids=zip_shapefile[["ZCTA5"]].copy()
zips_all_centroids["centroid"]=zip_shapefile.geometry.centroid
zips_all_centroids = zips_all_centroids.set_geometry("centroid")

zips_all_centroids.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 33120 entries, 0 to 33119
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   ZCTA5     33120 non-null  object
 1   centroid  33120 non-null  geometry
dtypes: geometry(1), object(1)
memory usage: 517.6+ KB
```

Dimension: 33120 points, 3 columns

Each column's meaning:

1st column: the index of all points/object in that GeoDataFrame

2nd column: the zip code of each point

3rd column: the well known text that describe the object type (point, linestring, polygon, etc) and describe the coordinate/location of the object on the earth surface, e.g.: POINT (-72.64 42.21)

2. 2 subsets of nationwide zip code GeoDataFrame

```
# Texas zip code centroids
zips_texas_centroids = texas_zip_shapefile.copy()
zips_texas_centroids["centroid"] = zips_texas_centroids.centroid
zips_texas_centroids= zips_texas_centroids.set_geometry("centroid")
print(f"N of unique Texas zip codes: {len(
    zips_texas_centroids["ZCTA5"].unique())}")
```

```
N of unique Texas zip codes: 1935
```

N of unique Texas zip codes: 1935

```
# Texas's bordering states: OK (730, 731, 734-739, 74),
# AR (716-729), LA (700-714), NM (870-885)
# Texas or a bordering state zip code centroids
oklahoma_prefixes = ('730', '731', '734', '735', '736', '737', '738', '739', '74')
arkansas_prefixes = tuple(str(i) for i in range(716, 730))
louisiana_prefixes = tuple(str(i) for i in range(700, 715))
new_mexico_prefixes = tuple(str(i) for i in range(870, 886))
```

```
all_prefixes1 = texas_prefixes + oklahoma_prefixes + arkansas_prefixes
all_prefixes = all_prefixes1 + louisiana_prefixes + new_mexico_prefixes
# Filter the nationwide ZIP code shapefile for Texas and its bordering states
zips_texas_borderstates_centroids = zip_shapefile[
    zip_shapefile['ZCTA5'].astype(str).str.startswith(all_prefixes)
].copy()
zips_texas_borderstates_centroids[
    "centroid"] = zips_texas_borderstates_centroids.centroid
zips_texas_borderstates_centroids= zips_texas_borderstates_centroids.set_geometry(
    "centroid")
print("N of unique Texas+ zip codes:")
print({len(zips_texas_borderstates_centroids["ZCTA5"].unique())})
```

```
N of unique Texas+ zip codes:
{4057}
```

N of unique Texas and bordering states zip codes: 4057

3.

```
# Retain rows with null ZIP codes with fill NA
act_hos_2016["ZIP_CD"] = act_hos_2016["ZIP_CD"].fillna(0).astype(int)
# Ensure the ZIP code columns are of the same data type for merging
act_hos_2016["ZIP_CD"] = act_hos_2016["ZIP_CD"].astype(int)
zips_texas_borderstates_centroids["ZCTA5"] = zips_texas_borderstates_centroids[
    "ZCTA5"].astype(int)
# Inner merge
zips_withhospital_centroids = zips_texas_borderstates_centroids.merge(
    act_hos_2016["ZIP_CD"],
    left_on = "ZCTA5",
    right_on = "ZIP_CD",
    how = "inner").drop_duplicates(
        subset="ZIP_CD")
zips_withhospital_centroids.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
Index: 777 entries, 0 to 1234
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   GEO_ID      777 non-null    object
 1   ZCTA5       777 non-null    int64
 2   NAME        777 non-null    object
 3   LSAD        777 non-null    object
 4   CENSUSAREA  777 non-null    float64
 5   geometry    777 non-null    geometry
 6   centroid    777 non-null    geometry
 7   ZIP_CD      777 non-null    int64
dtypes: float64(1), geometry(2), int64(2), object(3)
memory usage: 54.6+ KB
```

Merge used: An **inner merge** was used to only keep matching rows between `zips_texas_borderstates_centroids` and `act_hos_2016`, ensuring that `zips_withhospital_centroids` contains only ZIP codes that have at least one hospital while maintaining the non-redundant state of `zips_texas_borderstates_centroids`.

Variables used for merging: The merge was performed using the `ZCTA5` column (ZIP code in the `zips_texas_borderstates_centroids` GeoDataFrame) and the `ZIP_CD` column (ZIP code in the `act_hos_2016` DataFrame).

4. For each zip code in `zips_texas_centroids`, calculate the distance to the nearest zip code with at least one hospital in `zips_withhospital_centroids`

   a.

```
# Ensure the GeoDataFrames are using a projected CRS for
# accurate distance calculation
zips_texas_centroids = zips_texas_centroids.to_crs(epsg=3081)
zips_withhospital_centroids = zips_withhospital_centroids.to_crs(epsg=3081)
# Subset 10 rows of zips_texas_centroids
zips_texas_centroids_10 = zips_texas_centroids.head(10)
# Measure time for the subset distance calculation
import time
start_time_10 = time.time()
# Perform a spatial join using sjoin_nearest with the distance calculated
zips_texas_centroids_10_with_distance = gpd.sjoin_nearest(
    zips_texas_centroids_10,
    zips_withhospital_centroids,
    how="left",
    distance_col="nearest_zipcode_withhospital_distance"
)
end_time_10 = time.time()
time_taken_10 = end_time_10 - start_time_10
print(f"Time taken for calculating distances for 10 ZIP codes:")
print(f"{time_taken_10:.2f} seconds")
```

```
Time taken for calculating distances for 10 ZIP codes:
0.01 seconds
```

Time taken for calculating distances for 10 ZIP codes: 0.01 seconds

```
# Estimate the time for the full dataset based on the subset time
n_total_zips = len(zips_texas_centroids)
estimated_full_time = (
    time_taken_10 / len(zips_texas_centroids_10)) * n_total_zips
print(f"Estimated time for the full dataset: {estimated_full_time:.2f} seconds")
```

```
Estimated time for the full dataset: 1.33 seconds
```

Estimated time for the full dataset: 1.57 seconds

b.

```
# Measure time for the full dataset
start_time_full = time.time()
# Perform the spatial join for the entire dataset
zips_texas_centroids_with_distance = gpd.sjoin_nearest(
    zips_texas_centroids,
    zips_withhospital_centroids,
    how="left",
    distance_col="nearest_zipcode_withhospital_distance"
)
end_time_full = time.time()
full_time_taken = end_time_full - start_time_full
# Display the time taken for the full calculation
print(f"Time taken for the full calculation: {full_time_taken:.2f} seconds")
```

```
Time taken for the full calculation: 0.04 seconds
```

Time taken for the full calculation: 0.02 seconds When we run the code, the actual time taken for the full calculation was 0.02 seconds, much faster than the estimated 1.57 seconds. This indicates that `gpd.sjoin_nearest()` is highly efficient for spatial join operations. The results show that the method handles large spatial datasets quickly and effectively.

c. Inside the `.prj` file:
`GEOGCS["GCS_North_American_1983",DATUM["D_North_American_1983",SPHEROID["GRS_1980",6378137,298.257222101]],PRIMEM["Greenwich",0],UNIT["Degree",0.0174532925199433295]]`

The `.prj` file content shows that the coordinate reference system (CRS) used is `GEOGCS["GCS_North_American_1983"]`, which means it is based on the North American Datum 1983 (NAD83). The unit specified in the CRS is `UNIT["Degree",0.0174532925199433295]`, indicating that the units for measurement are in degrees. Degrees are angular units used in geographic coordinate systems, representing distances on a spherical surface (e.g., latitude and longitude).

One degree of latitude is approximately equivalent to 69 miles, as this distance remains consistent across the globe. However, the distance between degrees of longitude varies with latitude, decreasing toward the poles, which can introduce significant measurement errors when converting directly to miles.

For more accurate distance calculations, it's best to reproject the data into a CRS designed for distance measurements in meters, such as EPSG:3081 (NAD83 / Texas State Mapping System) for Texas. Once reprojected,**distances measured in meters can be converted to miles by dividing by 1,609.34**. This method ensures accurate spatial measurements tailored to specific locations.

5.

```
# Ensure the GeoDataFrames are using a projected CRS for
# accurate distance calculation
zips_texas_centroids = zips_texas_centroids.to_crs(epsg=3081)
zips_withhospital_centroids = zips_withhospital_centroids.to_crs(epsg=3081)
```

```
# Perform a spatial join to find the nearest hospital ZIP code
# and calculate distances
zips_texas_centroids_with_distance = gpd.sjoin_nearest(
    zips_texas_centroids,
    zips_withhospital_centroids,
    how="left",
    distance_col="nearest_hospital_distance_meters"
)
```

a.

```
zips_texas_centroids_with_distance[
    ["ZCTA5_left", "nearest_hospital_distance_meters"]].head()
```

	ZCTA5_left	nearest_hospital_distance_meters
9207	78624	0.000000
9208	78626	0.000000
9209	78628	12175.241300
9210	78631	36431.369241
9211	78632	15857.343050

The distance is still in meter after we change the projection to EPSG:3081. Previously, the distance was in degree before we change the projection.

b.

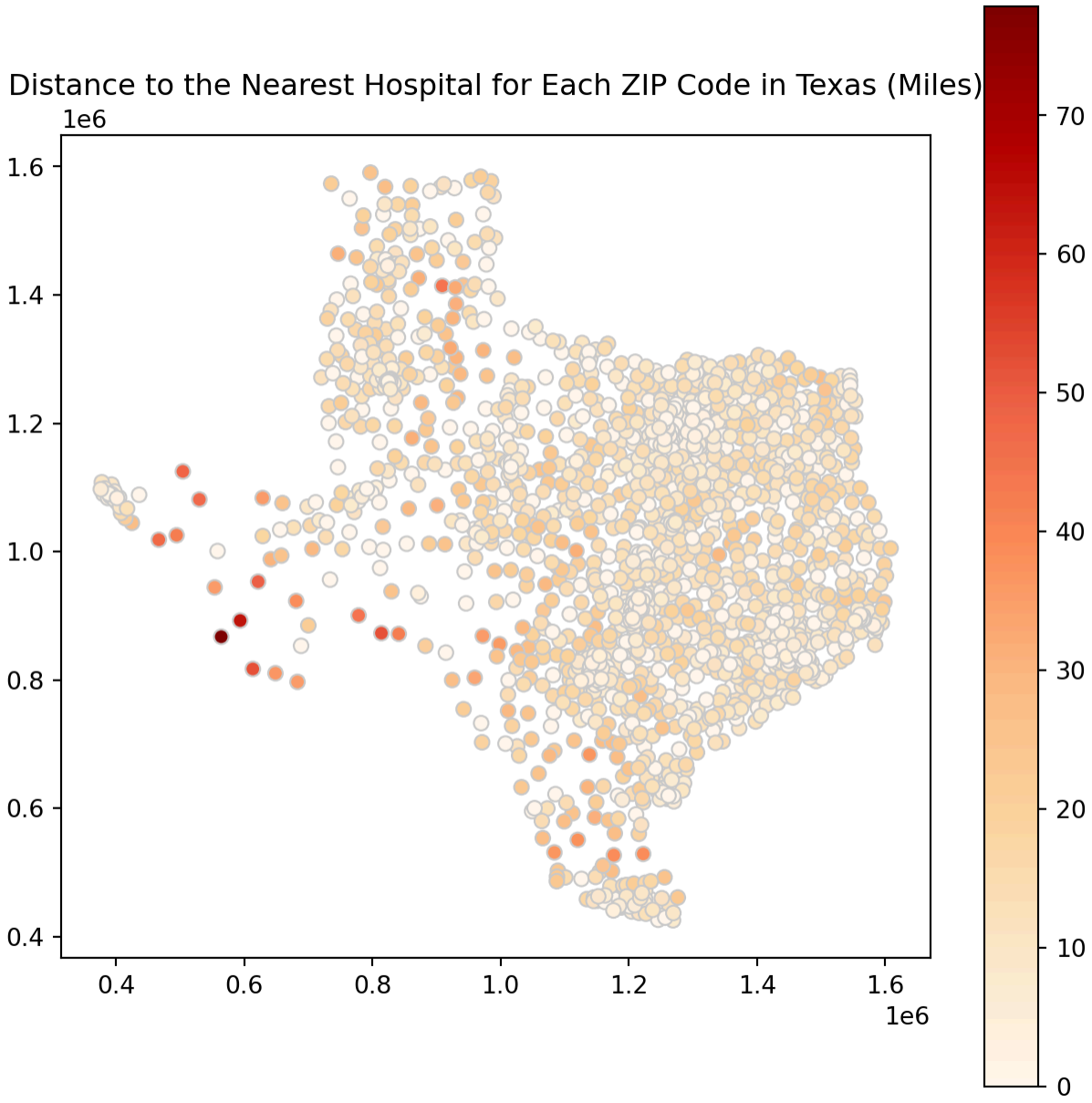
```
# Convert distance from meters to miles
zips_texas_centroids_with_distance["nearest_hospital_distance_miles"] = (
    zips_texas_centroids_with_distance[
        "nearest_hospital_distance_meters"] / 1609.34
)
# Show some rows of the distances in descending order
zips_texas_centroids_with_distance.sort_values(
    "nearest_hospital_distance_miles", ascending=False
)[["ZCTA5_left", "nearest_hospital_distance_miles"]].head()
```

	ZCTA5_left	nearest_hospital_distance_miles
32711	79845	77.859576
26872	79843	63.870396
25070	78851	52.183719
26873	79846	51.834331
9659	79734	49.185930

The furthest distance of a ZIP code to the nearest ZIP code with a hospital is 77.859 miles, highlighting the potential challenges faced by rural areas in Texas when accessing healthcare. However, since we calculate distances between centroids of ZIP codes, this method may overlook the actual distance from residential or inhabited areas within a ZIP code to a nearby hospital, which could be closer to the ZIP code's border.

c.

```
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
zips_texas_centroids_with_distance.plot(
    column="nearest_hospital_distance_miles",
    cmap="OrRd",
    linewidth=0.8,
    ax=ax,
    edgecolor="0.8",
    legend=True
)
plt.title("Distance to the Nearest Hospital for Each ZIP Code in Texas (Miles)")
plt.show()
```



V Effects of closures on access in Texas (15 pts)

1.

```
# Filter to include only Texas ZIP codes
# (assuming Texas prefixes are already defined)
texas_prefixes = ('733', '75', '76', '77', '78', '79')
corrected_closures['ZIP_CD'] = corrected_closures['ZIP_CD'].astype(str)
texas_closures = corrected_closures[
    corrected_closures['ZIP_CD'].str.startswith(texas_prefixes)]
# Create a table showing how many closures each
# ZIP code experienced from 2016 to 2019
zip_closure_counts = texas_closures.groupby(
    'ZIP_CD').size().reset_index(name='closure_count')
# Create a summary table showing the number of
```

```
# ZIP codes versus the number of closures
zip_closure_summary = zip_closure_counts.groupby(
    'closure_count').size().reset_index(name='num_zip_codes')
# Rename columns for clarity
zip_closure_summary.columns = ['Number of Closures', 'Number of ZIP Codes']
# Display the summary table
print("Table: Number of ZIP codes vs. Number of Closures")
zip_closure_summary
```

```
Table: Number of ZIP codes vs. Number of Closures
```

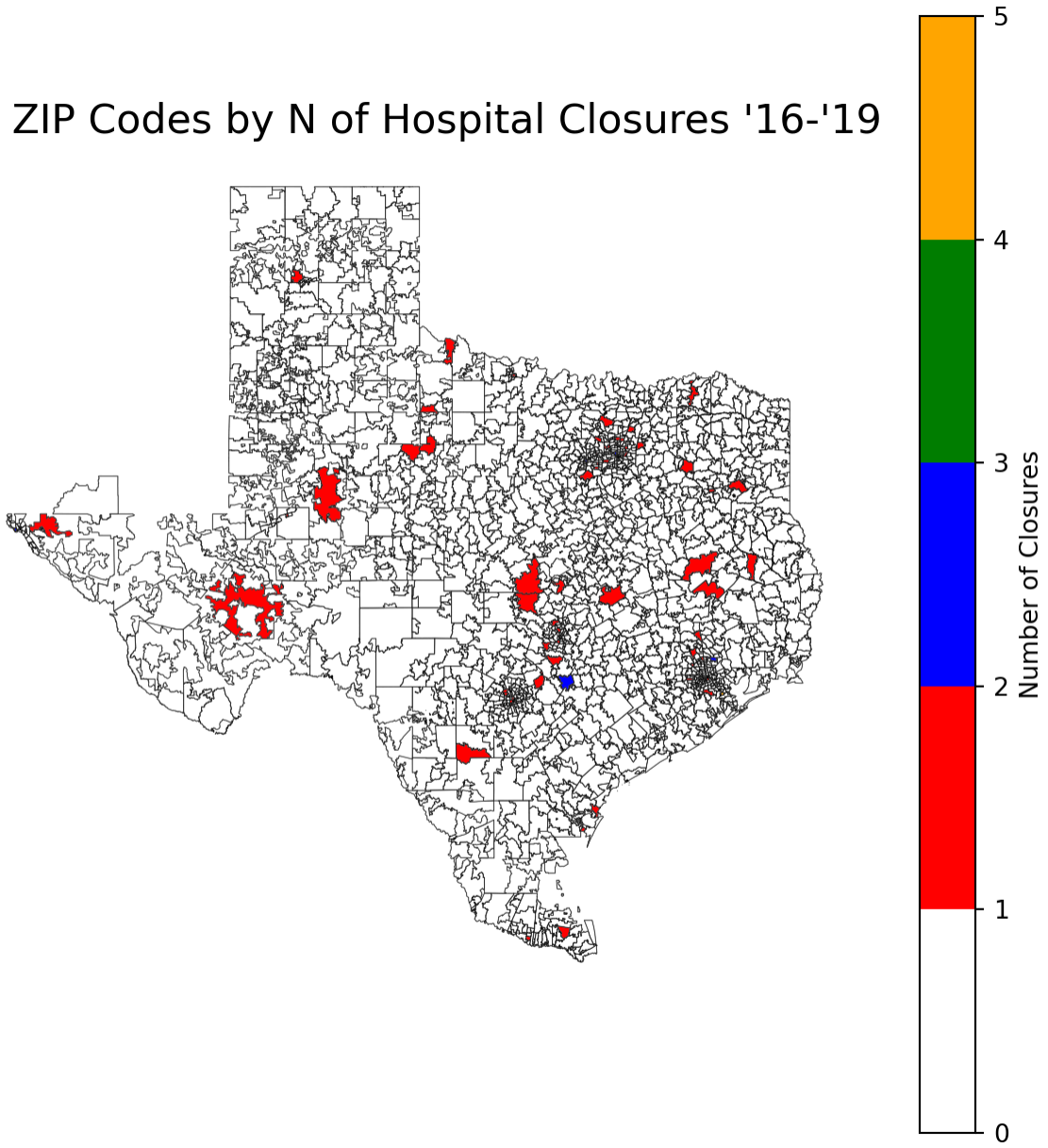
	Number of Closures	Number of ZIP Codes
0	1	67
1	2	5
2	5	1

Interpretation: The table provided shows the distribution of ZIP codes in Texas based on the number of hospital closures they experienced between 2016 and 2019. Out of the total ZIP codes, 67 ZIP codes had no closures, 5 ZIP codes experienced 1 closure, and 1 ZIP code experienced 2 closures. This indicates that hospital closures during this period were relatively rare, affecting a small number of ZIP codes in the state, with most ZIP codes not experiencing any closures at all.

2.

```
import matplotlib.colors as mcolors
import matplotlib.patches as mpatches
# Ensure ZIP_CD in texas_closures is clean before converting to string
texas_closures['ZIP_CD'] = pd.to_numeric(texas_closures['ZIP_CD'], errors='coerce')
texas_closures = texas_closures.dropna(subset=['ZIP_CD'])
texas_closures['ZIP_CD'] = texas_closures['ZIP_CD'].astype(int).astype(str)
# Ensure ZCTA5 in texas_zip_shapefile is also a string for merging
texas_zip_shapefile['ZCTA5'] = texas_zip_shapefile['ZCTA5'].astype(str)
# Count the number of closures per ZIP code
closure_counts = texas_closures.groupby(
    'ZIP_CD').size().reset_index(name='Number of Closures')
# Merge the Texas ZIP code shapefile with the closure counts
texas_zip_closures_map = texas_zip_shapefile.merge(
    closure_counts,
    left_on='ZCTA5',
    right_on='ZIP_CD',
    how='left'
)
# Fill NaN values with 0 for ZIP codes with no closures
texas_zip_closures_map['Number of Closures'] = texas_zip_closures_map[
    'Number of Closures'].fillna(0)
# Define a colormap where 0 closures are white, and
# other numbers are mapped to distinct colors
cmap = mcolors.ListedColormap(['white', 'red', 'blue', 'green', 'orange'])
```

```
boundaries = [0, 1, 2, 3, 4, 5]
norm = mcolors.BoundaryNorm(boundaries, cmap.N)
# Plot the choropleth map showing all ZIP codes,
# including those with 0 closures in white
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
texas_zip_closures_map.boundary.plot(ax=ax, linewidth=0.1, color='black')
texas_zip_closures_map.plot(
    column='Number of Closures',
    cmap=cmap,
    norm=norm,
    linewidth=0.4,
    ax=ax,
    edgecolor='0.4',
    legend=True,
    legend_kwds={'label': "Number of Closures", 'orientation': "vertical"}
)
plt.title("TX ZIP Codes by N of Hospital Closures '16-'19",
fontsize=16)
ax.axis('off')
plt.show()
```



Interpretation: The choropleth map shows the distribution of Texas ZIP codes that were directly affected by hospital closures between 2016 and 2019. ZIP codes shaded in colors other than white indicate areas with closures, with the color intensity representing the number of closures (e.g., red for 1 closure, blue for 2 closures, etc.). According to the map, a total of 6 ZIP codes were directly affected: 5 ZIP codes experienced 1 closure, and 1 ZIP code experienced 2 closures. Most ZIP codes remain unaffected, indicated by the white color. This visual highlights that while some ZIP codes faced closures, the impact was limited to a relatively small portion of the state.

3.

```
# Ensure the CRS is set correctly for distance calculations in meters
texas_zip_shapefile = texas_zip_shapefile.to_crs(epsg=3081)
texas_zip_closures_map = texas_zip_closures_map.to_crs(epsg=3081)
# Convert distance from miles to meters
mile_to_meter = 1609.34
buffer_dist = 10 * mile_to_meter
```

```
# Step 1: Create a GeoDataFrame of directly affected ZIP codes
direct_affected = texas_zip_closures_map[
    texas_zip_closures_map['Number of Closures'] > 0
]
# Step 2: Create a 10-mile buffer around directly affected ZIP codes
direct_affected = direct_affected.set_geometry('geometry')
direct_affected_buf = direct_affected.copy()
direct_affected_buf['geometry'] = direct_affected_buf.geometry.buffer(
    buffer_dist)
# Step 3: Merge buffers into a single geometry to avoid overcounting
buffer_union = direct_affected_buf['geometry'].unary_union
# Create a GeoDataFrame with the buffer for visualization or further analysis
buffer_gdf = gpd.GeoDataFrame(geometry=[buffer_union],
crs=texas_zip_shapefile.crs)
# Step 4: Identify ZIP codes that intersect with
# the 10-mile buffer (indirectly affected)
indirect_affected = texas_zip_shapefile[
    texas_zip_shapefile.geometry.intersects(buffer_union)
]
# Step 5: Remove any directly affected ZIP codes
# from the indirectly affected ones
indirect_affected = indirect_affected[
    ~indirect_affected['ZCTA5'].isin(direct_affected['ZCTA5'])
]
# Count the unique number of indirectly affected ZIP codes
num_indirect_affected = indirect_affected['ZCTA5'].nunique()
print(f"Number of indirectly affected ZIP codes in Texas:")
print(f"{num_indirect_affected}")
```

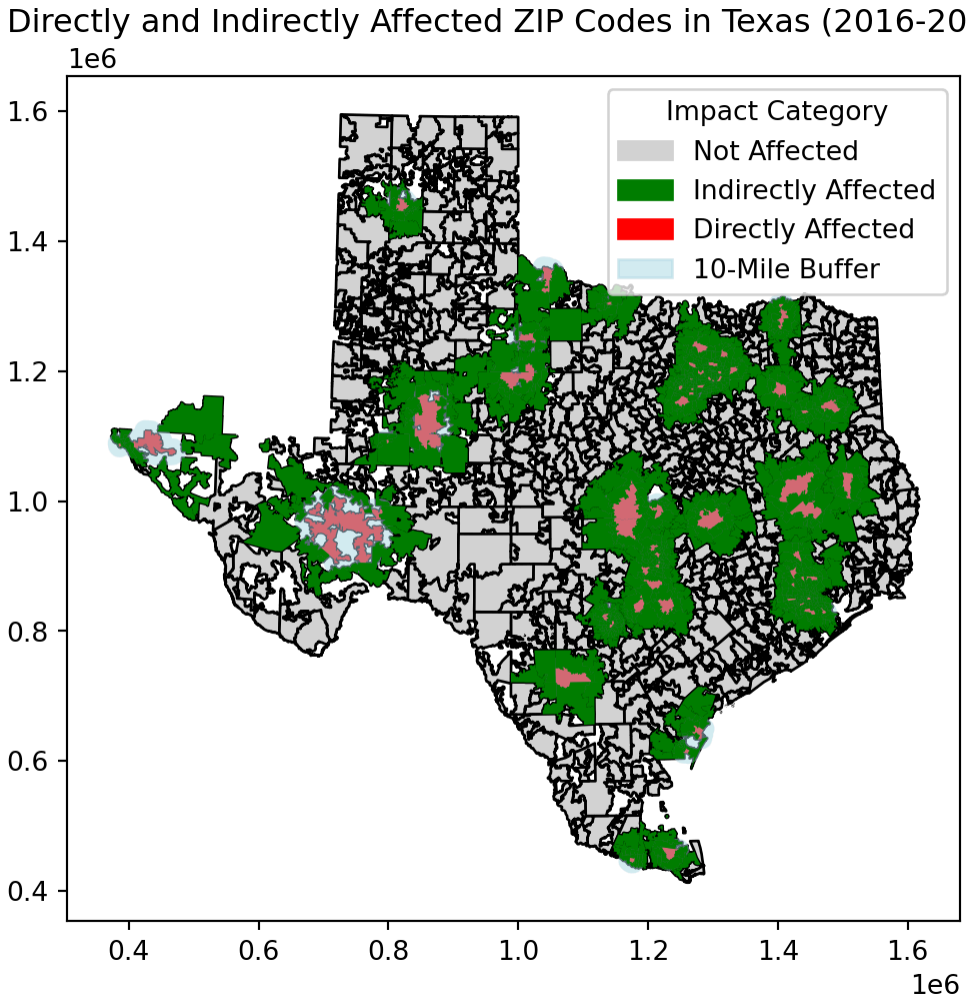
```
Number of indirectly affected ZIP codes in Texas:
882
```

Interpretation: The analysis identifies 882 ZIP codes in Texas as indirectly affected by hospital closures between 2016 and 2019, defined as being within a 10-mile radius of directly affected ZIP codes. This was determined by creating a buffer around directly affected ZIP codes (those with at least one hospital closure) and performing a spatial join with the entire Texas ZIP code dataset. The result indicates that while the number of directly impacted ZIP codes is small, the potential reach of their closures is substantial, affecting a large number of nearby areas and potentially impacting access to healthcare for residents in those ZIP codes.

4.

```
# Plot the results
fig, ax = plt.subplots(1, 1, figsize=(6, 6))
texas_zip_shapefile.plot(ax=ax, color='lightgray',
 edgecolor='black', label='Not Affected')
direct_affected.plot(ax=ax, color='red', label='Directly Affected')
buffer_gdf.plot(ax=ax, color='lightblue', alpha=0.5, label='10-Mile Buffer')
indirect_affected.plot(ax=ax, color='green', label='Indirectly Affected')
# Create a custom legend
```

```
legend_labels = [
    mpatches.Patch(color='lightgray', label='Not Affected'),
    mpatches.Patch(color='green', label='Indirectly Affected'),
    mpatches.Patch(color='red', label='Directly Affected'),
    mpatches.Patch(color='lightblue', alpha=0.5, label='10-Mile Buffer')
]
# Add the custom legend to the plot
plt.legend(handles=legend_labels, loc='upper right', title='Impact Category')
plt.title("Directly and Indirectly Affected ZIP Codes in Texas (2016-2019)")
plt.show()
```



Interpretation: The choropleth map provides a visual representation of Texas ZIP codes categorized by their impact from hospital closures between 2016 and 2019. ZIP codes highlighted in red indicate areas directly affected by at least one hospital closure, while green shows ZIP codes indirectly affected—those located within a 10-mile radius of directly affected areas but without closures themselves. The light blue areas represent the 10-mile buffer zones around directly affected ZIP codes, demonstrating the reach of their influence. Light gray represents ZIP codes that were not affected by closures or the 10-mile buffer zone. The map shows that while directly affected ZIP codes are fewer in number, the indirectly affected ZIP codes spread significantly, highlighting the potential ripple effect of hospital closures on surrounding communities.

VI Reflecting on the exercise (10 pts)

Partner 1 (Astari): The “first-pass” method for identifying hospital closures in a dataset can have significant limitations that lead to inaccuracies. Issues such as data completeness and accuracy can result in misidentified closures if the information is outdated or incomplete, and temporary closures or name changes may be missed. Ambiguities in defining what constitutes a closure, along with geocoding errors, can misclassify operational changes like mergers or relocations as closures. Additionally, using outdated records may misrepresent the current operational status of hospitals, and facilities offering limited services may not be correctly categorized.

To improve the accuracy of identifying hospital closures, several strategies can be employed. Cross-verifying the dataset with multiple reliable sources, such as state health department records and hospital associations, can confirm closure statuses. Real-time data integration ensures that information remains current, while advanced geospatial analysis enhances location accuracy. Automated systems can gather relevant news reports and public announcements for additional verification, and community feedback can provide firsthand insights into facility operations. Finally, analyzing patient admission and discharge trends can help indicate actual closures or reductions in service.

Partner 2 (Surya): The current method of identifying ZIP codes affected by hospital closures reflects changes in access to some extent but has limitations. The method accurately marks ZIP codes with closures, providing an overview of access shifts. However, it can misrepresent the true impact if hospitals are merged or relocated, as these are often mistakenly recorded as closures. Additionally, using ZIP code centroids for measuring distances can overlook actual residential areas and lead to misleading conclusions about accessibility, especially in larger or irregularly shaped ZIP codes.

To improve this measure, several strategies could be implemented. Cross-referencing data from multiple sources like state health department records, healthcare registries, and local news can validate true closures and minimize errors due to mergers or relocations. Using geocoordinates or exact hospital addresses instead of centroids would enhance the accuracy of distance calculations and better reflect access for residents. Incorporating historical trend analysis could differentiate between permanent closures and temporary changes, such as service suspensions or renovations. Additionally, gathering input from community surveys and government announcements could provide deeper insights into local healthcare accessibility and ensure that the analysis better reflects reality.