Probelm Set 4: Ben and Helen

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): Ben Zhang bzhang369
    - Partner 2 (name and cnet ID): Helen Chen chen61
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: XC, BZ
5. "I have uploaded the names of anyone else other than my partner and I worked with on the problem set **here**" : XC,BZ (1 point)
6. Late coins used this pset: 0 Late coins left after submission: 4
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.

```
import altair as alt
alt.renderers.enable("png")

import warnings
warnings.filterwarnings('ignore')
```

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

1.

```
import pandas as pd

data_2016 = pd.read_csv('pos2016.csv')

print(data_2016.columns)
```

```
Index(['PRVDR_CTGRY_SBTYP_CD', 'PRVDR_CTGRY_CD', 'FAC_NAME', 'PRVDR_NUM',
       'PGM_TRMNTN_CD', 'TRMNTN_EXPRTN_DT', 'ZIP_CD'],
      dtype='object')
```

We selected 7 variables: provider type code 'PRVDR_CTGRY_SBTYP_CD', provider subtype code 'PRVDR_CTGRY_CD', CMS certification number 'PRVDR_NUM', termination code 'PGM_TRMNTN_CD', facility name 'FAC_NAME' , zip 'ZIP_CD', years of suspected closure 'TRMNTN_EXPRTN_DT'.

2.

a.

```
short_term_2016 = data_2016 [(data_2016['PRVDR_CTGRY_SBTYP_CD'] == 1) &
                             (data_2016['PRVDR_CTGRY_CD'] == 1)]

count_short_2016 = len(short_term_2016)
print(count_short_2016)
```

7245

There are 7,245 short-term hospitals in this dataset. I believe this number makes sense, as we filtered the data for 2016 using conditions for both "hospital" and "short-term."

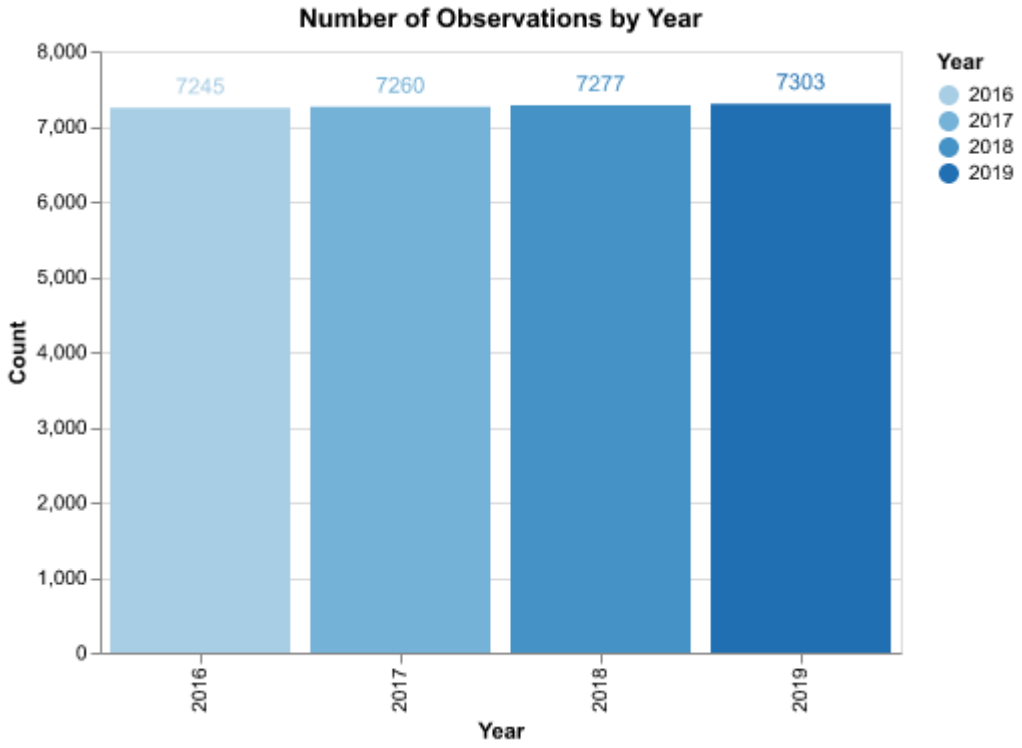
b. According to the KFF, there are approximately 5,000 short-term acute care hospitals in the U.S. In our data, we identified 7,245 short-term hospitals for 2016, which is higher than the KFF reference. This difference could be due to KFF using a different reference year than we did here, or due to variations in how hospitals are categorized across databases. Additionally, our data may include hospitals that were recently closed or merged but are still coded as active.

3.

```
import pandas as pd

file_paths = {
    2016: 'pos2016.csv',
    2017: 'pos2017.csv',
    2018: 'pos2018.csv',
    2019: 'pos2019.csv'
}

all_short_term_data = []

# Reference: chatGPT told me to use .copy() to avoid the
 ↪  SettingWithCopyWarning
for year, path in file_paths.items():
    data = pd.read_csv(path, encoding='ISO-8859-1')

    short_term_data = data[(data['PRVDR_CTGRY_SBTYP_CD'] == 1) &
                           (data['PRVDR_CTGRY_CD'] == 1)].copy()

    short_term_data.loc[:, 'Year'] = year

    all_short_term_data.append(short_term_data)

    count_short_term = len(short_term_data)
    print(f"Number of short-term hospitals in {year}: {count_short_term}")

combined_short_term = pd.concat(all_short_term_data, ignore_index=True)

import altair as alt
```

```
year_counts = combined_short_term['Year'].value_counts().reset_index()
year_counts.columns = ['Year', 'Count']

bar_chart = alt.Chart(year_counts).mark_bar().encode(
    alt.X('Year:O'),
    alt.Y('Count:Q', scale=alt.Scale(zero=False)),
    alt.Color('Year:O')
).properties(
    title='Number of Observations by Year',
    width=400,
    height=300
)

# Chatgpt, 'How can I add a text to my graph to make it more readable'
text = bar_chart.mark_text(
    align='center',
    baseline='bottom',
    dy=-5
).encode(
    text='Count:Q'
)

final_chart = bar_chart + text
display(final_chart)
```

```
Number of short-term hospitals in 2016: 7245
Number of short-term hospitals in 2017: 7260
Number of short-term hospitals in 2018: 7277
Number of short-term hospitals in 2019: 7303
```

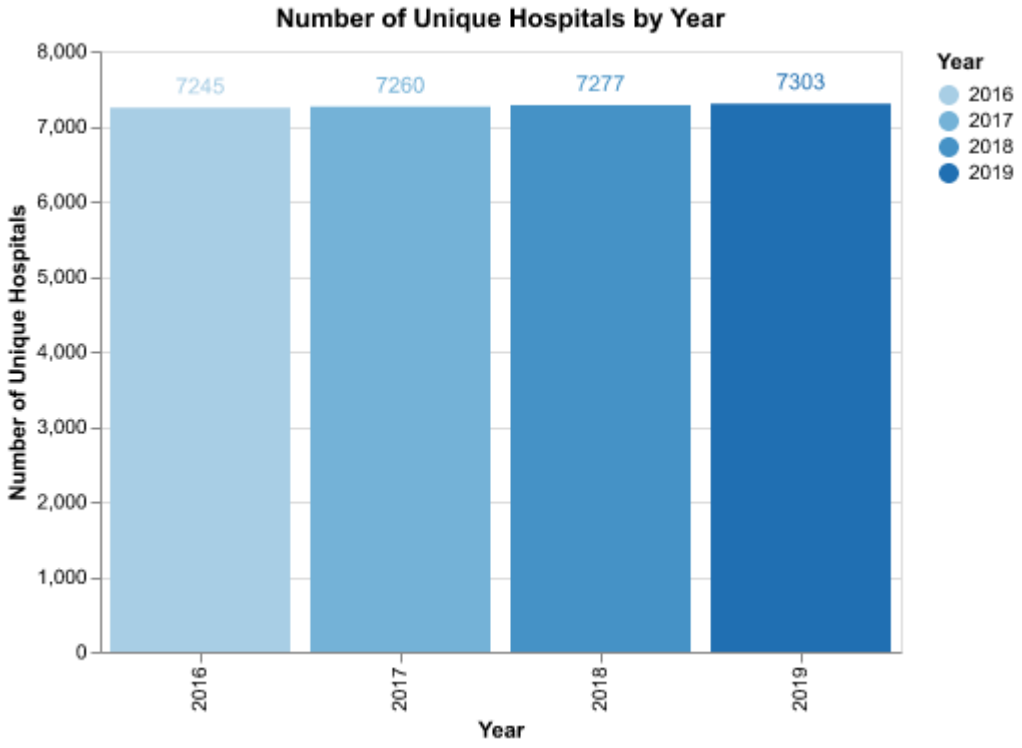


4.

a.

```
unique_hospital_counts = combined_short_term.groupby(
    'Year')['PRVDR_NUM'].nunique().reset_index()
unique_hospital_counts.columns = ['Year', 'Unique_Hospitals']

unique_hospitals_chart =
 ↪  alt.Chart(unique_hospital_counts).mark_bar(point=True).encode(
    x=alt.X('Year:O', title='Year'),
    y=alt.Y('Unique_Hospitals:Q', title='Number of Unique Hospitals'),
    color=alt.Color('Year:O', title='Year')
).properties(
    title='Number of Unique Hospitals by Year',
    width=400,
    height=300
)

text = unique_hospitals_chart.mark_text(
    align='center',
```

```
    baseline='bottom',
    dy=-5
).encode(
    text='Unique_Hospitals:Q'
)

unique_hospitals = unique_hospitals_chart + text
display(unique_hospitals)
```



b.

The two graphs report same number of short term hospitals each year, whether unique or not by CMS ID. The counts of observations and unique hospitals for each year are identical, suggesting that each observation in the dataset likely corresponds to a unique hospital for that year. This pattern implies that there is a one-to-one relationship between observations and unique hospitals annually. Or this might mean that the dataset does not contain multiple entries for the same hospital within a single year, or if it does, these duplicates have been aggregated or removed.

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

1.

```
active_2016 = combined_short_term[combined_short_term['Year'] == 2016]
active_2016 = active_2016[active_2016['PGM_TRMNTN_CD'] == 0]


def check_status_in_year(hospital_id, year):
    hospital = combined_short_term[(combined_short_term['PRVDR_NUM'] ==
 ↪  hospital_id) &
                                   (combined_short_term['Year'] == year)]
    if hospital.empty:
        return 'Not Found'
    elif hospital.iloc[0]['PGM_TRMNTN_CD'] == 0:
        return 'Active'
    else:
        return 'Terminated'


closed_hospitals = []

# Chatgpt, "help me combine two loop together and please make sure the loop
 ↪  not run forever"
# Reference:
 ↪  https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iterrows.html

for _, row in active_2016.iterrows():
    hospital_id = row['PRVDR_NUM']
    hospital_name = row['FAC_NAME']
    zip_code = row['ZIP_CD']

    for year in range(2017, 2020):
        status = check_status_in_year(hospital_id, year)
        if status == 'Not Found' or status == 'Terminated':
            closed_hospitals.append((hospital_name, zip_code, year))
            break


closed_hospitals_df = pd.DataFrame(closed_hospitals, columns=[
                                   'Facility Name', 'ZIP Code', 'Year of
                                    ↪  Suspected Closure'])
```

```
num_closed_hospitals = len(closed_hospitals_df)
print(num_closed_hospitals)
```

```
174
```

The number of hospitals suspected to have closed by 2019 is 174.

2.

```
sorted_closed_hospitals_df = closed_hospitals_df.sort_values(
    by='Facility Name')

first_10_closed_hospitals = sorted_closed_hospitals_df.head(10)

print(first_10_closed_hospitals[[
      'Facility Name', 'Year of Suspected Closure']])
```

```
                                    Facility Name  Year of Suspected Closure
4                            ABRAZO MARYVALE CAMPUS                       2017
10       ADVENTIST MEDICAL CENTER - CENTRAL VALLEY                       2017
97                          AFFINITY MEDICAL CENTER                       2018
80   ALBANY MEDICAL CENTER / SOUTH CLINICAL CAMPUS                       2017
140       ALLEGIANCE SPECIALTY HOSPITAL OF KILGORE                       2017
62                           ALLIANCE LAIRD HOSPITAL                       2019
101                        ALLIANCEHEALTH DEACONESS                       2019
26                   ANNE BATES LEACH EYE HOSPITAL                       2019
21          ARKANSAS VALLEY REGIONAL MEDICAL CENTER                      2017
69              BANNER CHURCHILL COMMUNITY HOSPITAL                      2017
```

3.

a.

```
active_hospitals = combined_short_term[combined_short_term['PGM_TRMNTN_CD']
 ↪  == 0]

active_by_zip_df = active_hospitals.groupby(
    ['ZIP_CD', 'Year']).size().unstack()

# Chatgpt, 'How to solve this arrtibuteerror of series object have no
 ↪  attribute colums', ask for debug help and added .unstack()
```

```
# reference of unstack:
 ↪  https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.unstack.html

potential_mergers = []

for _, row in closed_hospitals_df.iterrows():
    zip_code = row['ZIP Code']
    year_closed = row['Year of Suspected Closure']

    if year_closed in [2016, 2017, 2018]:
        next_year = year_closed + 1

        if zip_code in active_by_zip_df.index and next_year in
         ↪  active_by_zip_df.columns:
            if active_by_zip_df.loc[zip_code, year_closed] ==
             ↪  active_by_zip_df.loc[zip_code, next_year]:
                potential_mergers.append(row)

# ChatGpt, 'how to write if loop if I want to compare one year and a year
 ↪  after to see if there is any change in number of some column?'

potential_mergers_df = pd.DataFrame(potential_mergers)

num_potential_mergers = len(potential_mergers_df)

print(num_potential_mergers)
```

27

After filtering for suspected hospital closures in ZIP codes where the number of active hospitals does not decrease in the year following the suspected closure, we identified 27 hospitals that fit the definition of potential mergers.

b.

```
adjusted_closures_df = closed_hospitals_df[
    (~closed_hospitals_df.index.isin(potential_mergers_df.index))]

num_adjusted_closures = len(adjusted_closures_df)

print(num_adjusted_closures)
```

```
147
```

After accounting for possible merged hospitals, there are 147 hospitals remaining that are potential true closures. However, it's important to note that, due to the absence of data for 2020, these 147 hospitals may still include some that were merged in 2019, which we cannot currently distinguish.

c.

```
adjusted_closures_df.head(10)
```

	Facility Name	ZIP Code	Year of Suspected Closure
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
6	GILBERT HOSPITAL	85295.0	2018
7	FLORENCE HOSPITAL AT ANTHEM, LLC	85132.0	2018
8	PACIFIC ALLIANCE MEDICAL CENTER	90012.0	2019
9	O'CONNOR HOSPITAL	95128.0	2019
11	TULARE REGIONAL MEDICAL CENTER	93274.0	2018

Download Census zip code shapefile (10 pt)

1.

a.

Database File: .dbf 6.4 mb This file contains attribute data in a tabular format associated with the shapes in the .shp file.

Projection File: .prj 165 bytes This file defines the coordinate system and projection information for the shapefile.

Shape File: .shp 837.5 mb This is the primary file that contains the geometric data for the shapes (points, lines, polygons) in a vector format.

Shape Index File: .shx 265 KB This file contains an index linking records in the .dbf attribute table to geometric shapes, optimizing spatial data handling and rendering.

Metadata File: .xml 16 KB This file contains metadata about the shapefile.

b.

.shp: 837.5 MB (largest, storing geographic shapes)

.dbf: 6.13 MB (contains attribute data)

.shx: 265 KB (shape index)

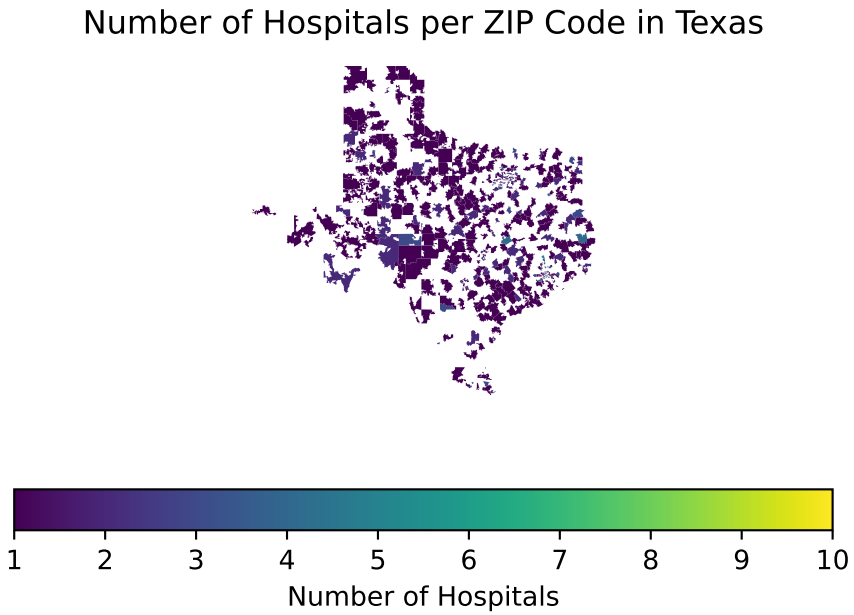
.xml: 16 KB (metadata)

.prj: 165 bytes=(projection)

The .shp file is by far the largest, as expected for spatial data, followed by the .dbf file containing attribute information. The other files are relatively small in comparison.

2.

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

zip_codes = gpd.read_file("gz_2010_us_860_00_500k.shp")

texas_zip_codes = zip_codes[zip_codes['ZCTA5'].str.startswith(('75', '76',
 ↪  '77', '78', '79'))]

hospital_counts_2016 =
 ↪  short_term_2016.groupby('ZIP_CD').size().reset_index(name='Hospital_Count')

hospital_counts_2016['ZIP_CD'] =
 ↪  hospital_counts_2016['ZIP_CD'].astype(int).astype(str)

texas_hospitals = texas_zip_codes.merge(hospital_counts_2016,
 ↪  left_on='ZCTA5', right_on='ZIP_CD', how='left')

# reference:
 ↪  https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html


fig = texas_hospitals.plot(column='Hospital_Count', legend=True,
                           legend_kwds={'label': "Number of Hospitals",
                                        'orientation': "horizontal"})

plt.title('Number of Hospitals per ZIP Code in Texas')
plt.axis("off")
plt.show()
```



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

1.

```
zip_codes['centroid'] = zip_codes.geometry.centroid

zips_all_centroids = gpd.GeoDataFrame(zip_codes, geometry='centroid')

dimensions = zips_all_centroids.shape
columns = zips_all_centroids.columns

print("Dimensions of zips_all_centroids:", dimensions)
print("Columns in zips_all_centroids:", columns)
```

```
Dimensions of zips_all_centroids: (33120, 7)
Columns in zips_all_centroids: Index(['GEO_ID', 'ZCTA5', 'NAME', 'LSAD',
'CENSUSAREA', 'geometry',
       'centroid'],
      dtype='object')
```

Dimensions of zips_all_centroids: (33120, 7)

33120: This number indicates that there are 33,120 rows in the GeoDataFrame, meaning there are 33,120 unique ZIP code areas represented. 7: This number indicates that there are 7 columns in the GeoDataFrame.

Columns: GEO_ID: A unique identifier for each geographic feature, specifically for ZIP code areas. It is often used for linking to other datasets or for geographic referencing.

ZCTA5: The 5-digit ZIP Code Tabulation Area code. This represents the specific ZIP code and is used for statistical purposes, particularly in census data.

NAME: The name of the ZIP code area. This typically matches the ZIP code designation and may also provide a description of the area.

LSAD: Legal/Statistical Area Description. This field indicates whether the area is a standard ZIP code or another type of designation, such as a non-standard area for statistical purposes.

CENSUSAREA: The area size of the ZIP code in square meters (or another unit). This value can be useful for spatial analysis, such as calculating population density.

geometry: The original geometric representation of the ZIP code boundaries. This column contains the shapes of the ZIP code areas, stored as geometric objects.

centroid: The calculated centroid of each ZIP code area. This point represents the geographic center of the ZIP code boundaries and can be useful for mapping and spatial analysis.

2.

ZIP code prefixes for Texas: 75, 76, 77, 78, 79

Bordering states: New Mexico: 87, 88 Oklahoma: 73, 74 Arkansas: 71, 72 Louisiana: 70 71

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

border_states_prefixes = ['75', '76', '77', '78', '79',  # Texas
                          '87', '88',  # New Mexico
                          '73', '74',  # Oklahoma
                          '71', '72',  # Arkansas
                          '70']  # Louisiana

zips_texas_borderstates_centroids =
 ↪  zips_all_centroids[zips_all_centroids['ZCTA5']

 ↪  .str.startswith(tuple(border_states_prefixes))]

unique_texas_zips = zips_texas_centroids['ZCTA5'].nunique()
```

```
unique_border_states_zips =
 ↪  zips_texas_borderstates_centroids['ZCTA5'].nunique(
)

print(unique_texas_zips)
print(unique_border_states_zips)
```

```
1935
4057
```

Unique ZIP codes in Texas: 1935

Unique ZIP codes in Texas and bordering states: 4057

3.

```
zips_withhospital_centroids = zips_texas_borderstates_centroids.merge(
    hospital_counts_2016, how='inner', left_on='ZCTA5', right_on='ZIP_CD')
```

We used an inner join here, merging by the ZCTA5 and ZIP_CD columns, as it directly provides the subset of zips_texas_borderstates_centroids that matches ZIP codes in hospital_counts_2016. The purpose of this inner join is to retrieve only those ZIP codes within Texas and its bordering states that had at least one hospital in 2016. By joining only matching ZIP codes, we focus on relevant areas without processing additional records or applying further filters, making the workflow cleaner and more efficient.

4. a.

```
from shapely import Point
import time

zips_texas_sample = zips_texas_centroids.head(10)

start_time = time.time()

distances = []

# reference: Fast minimum distance between two sets of polygons
# https://gis.stackexchange.com/questions/226085/
# fast-minimum-distance-between-two-sets-of-polygons

for idx, ZCTA5 in zips_texas_sample.iterrows():
```

```
    min_distance = zips_withhospital_centroids.distance(ZCTA5.geometry).min()
    distances.append(min_distance)

end_time = time.time()
during_time = end_time - start_time

subset_size = len(zips_texas_sample)
full_size = len(zips_texas_centroids)
estimated_time = (during_time / subset_size) * full_size

print("Time taken for 10 ZIP codes:", during_time, "seconds")
print("Estimated time for full dataset:", estimated_time, "seconds")
```

```
Time taken for 10 ZIP codes: 0.23838281631469727 seconds
Estimated time for full dataset: 46.12707495689392 seconds
```

b.

```
start_time = time.time()

distances_full = []
for idx, ZCTA5 in zips_texas_centroids.iterrows():
    min_distance = zips_withhospital_centroids.distance(ZCTA5.geometry).min()
    distances_full.append(min_distance)

# reference: Fast minimum distance between two sets of polygons
# https://gis.stackexchange.com/questions/226085/
# fast-minimum-distance-between-two-sets-of-polygons

end_time = time.time()
full_time = end_time - start_time

print("Time for full dataset:", full_time, "seconds")
```

```
Time for full dataset: 37.93156027793884 seconds
```

After running the full model, the execution time was 36.57 seconds, while the predicted time from the previous step was approximately 46.22 seconds. This is close, but the actual time is shorter than the estimated time.

c.

```
# Read the .prj file
with open("gz_2010_us_860_00_500k.prj", "r") as file:
    prj_content = file.read()
# ChatGpt, 'help me read prj file', I ask help for how can I load .prj

print("Contents of .prj file:", prj_content)
```

```
Contents of .prj file:
GEOGCS["GCS_North_American_1983",DATUM["D_North_American_1983",SPHEROID["GRS_1980",6378137,29
```

The .prj file indicates the coordinate system as "GCS_North_American_1983" with the unit "Degree" of 0.017453292519943295.This means the distances are measured in degrees of latitude/longitude, not in a linear unit like meters.

```
converst_to_miles = 0.017453292519943295*69
print("Distance in miles:", converst_to_miles)
```

```
Distance in miles: 1.2042771838760873
```

After convert to miles, the distance is about 1.2043 miles.

5. a.

The unit is measure in degree.

b.

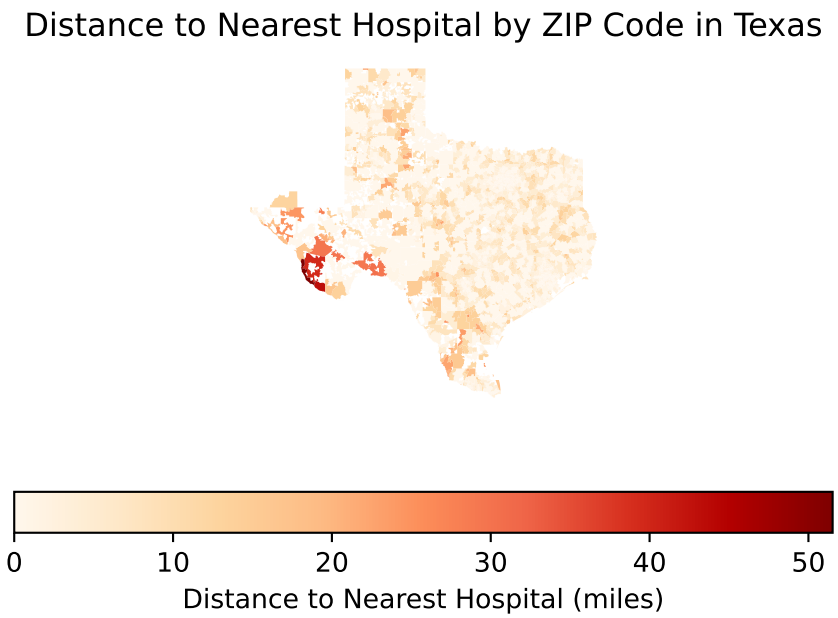
```
average_distance = sum(distances_full) / len(distances_full)
average_distance_miles = float(average_distance)*69

print("Average distance to nearest hospital (in miles):",
 ↪  average_distance_miles)
```

```
Average distance to nearest hospital (in miles): 4.810748479555737
```

I think the value make sense, 4.81 miles sounds like a real distance to a hospital.

c.

```
distances_miles = [d * 69 for d in distances_full]

zips_texas_centroids['Distance_to_Nearest_Hospital_miles'] = distances_miles

texas_distances = texas_zip_codes.merge(zips_texas_centroids[['ZCTA5',
 ↪  'Distance_to_Nearest_Hospital_miles']],
                                        left_on='ZCTA5', right_on='ZCTA5',
 ↪  how='left')

texas_distances.plot(column='Distance_to_Nearest_Hospital_miles',
                     legend=True,
                     legend_kwds={
                         'label': "Distance to Nearest Hospital (miles)",
                         'orientation': "horizontal"
                     },
                     cmap='OrRd')

plt.title('Distance to Nearest Hospital by ZIP Code in Texas')
plt.axis("off")
plt.show()
```



Effects of closures on access in Texas (15 pts)

1.

```
adjusted_closures_df['ZIP Code'] = adjusted_closures_df['ZIP
 ↪  Code'].astype(int).astype(str)

texas_closures = adjusted_closures_df[
    adjusted_closures_df['ZIP Code'].str.startswith(('75', '76', '77', '78',
 ↪  '79'))
]

texas_closures_2016_2019 = texas_closures[
    texas_closures['Year of Suspected Closure'].between(2016, 2019)
]

closures_per_zip = texas_closures_2016_2019.groupby('ZIP
 ↪  Code').size().reset_index(name='Closure_Count')

print(closures_per_zip)
```

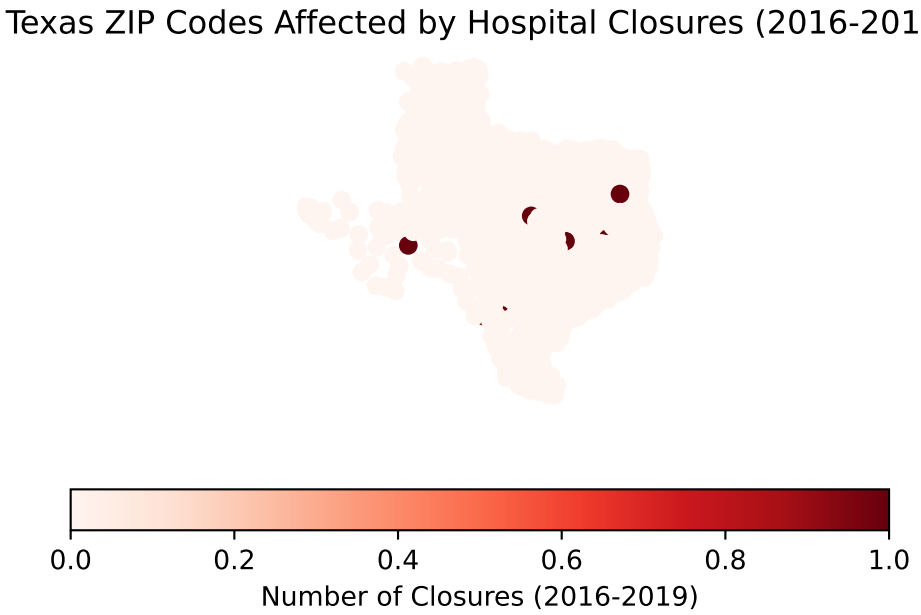
```
   ZIP Code  Closure_Count
0     75042              1
1     75051              1
2     75087              1
3     75140              1
4     75235              1
5     75390              1
6     75601              1
7     75662              1
8     75835              1
9     75862              1
10    76502              1
11    76520              1
12    76531              1
13    76645              1
14    77035              1
15    77065              1
16    77429              1
17    78017              1
18    78061              1
19    78336              1
```

```
20    78613              1
21    78734              1
22    78834              1
23    79520              1
24    79529              1
25    79553              1
26    79735              1
27    79902              1
```

2.

```
affected_zip_counts = texas_closures_2016_2019['ZIP
↪  Code'].value_counts().reset_index()
affected_zip_counts.columns = ['ZIP Code', 'closure_count']

num_affected_zip_codes = affected_zip_counts['ZIP Code'].nunique()
print(f"Number of directly affected ZIP codes in Texas:
↪  {num_affected_zip_codes}")

texas_zip_affected = zips_texas_centroids.merge(affected_zip_counts,
↪  left_on='ZCTA5', right_on='ZIP Code', how='left')
texas_zip_affected['closure_count'] =
↪  texas_zip_affected['closure_count'].fillna(0)

fig = texas_zip_affected.plot(column='closure_count', cmap='Reds',
↪  legend=True,
                              legend_kwds={'label': "Number of Closures
↪  (2016-2019)",
                                           'orientation': "horizontal"})

plt.title('Texas ZIP Codes Affected by Hospital Closures (2016-2019)')
plt.axis("off")
plt.show()
```

```
Number of directly affected ZIP codes in Texas: 28
```



3.

```
import geopandas as gpd

directly_affected_zips =
 ↪  texas_zip_affected[texas_zip_affected['closure_count'] > 0]

# The orginal data measured in degree, now we change to meters to match with
 ↪  buffer. Convert the CRS of both GeoDataFrames to EPSG:3857 for
 ↪  meter-based operations
texas_zip_affected = texas_zip_affected.to_crs(epsg=3857)
directly_affected_zips = directly_affected_zips.to_crs(epsg=3857)

directly_affected_zips['buffer'] = directly_affected_zips.geometry.buffer(10
 ↪  * 1609.34)

buffered_affected_zips = gpd.GeoDataFrame(directly_affected_zips,
 ↪  geometry='buffer')

indirectly_affected_zips = gpd.sjoin(texas_zip_affected,
 ↪  buffered_affected_zips[['ZCTA5', 'buffer']],
                                      how='inner', predicate='intersects')

indirectly_affected_zips.rename(columns={'ZCTA5_left': 'ZCTA5',
 ↪  'ZCTA5_right': 'buffered_ZCTA5'}, inplace=True)
```

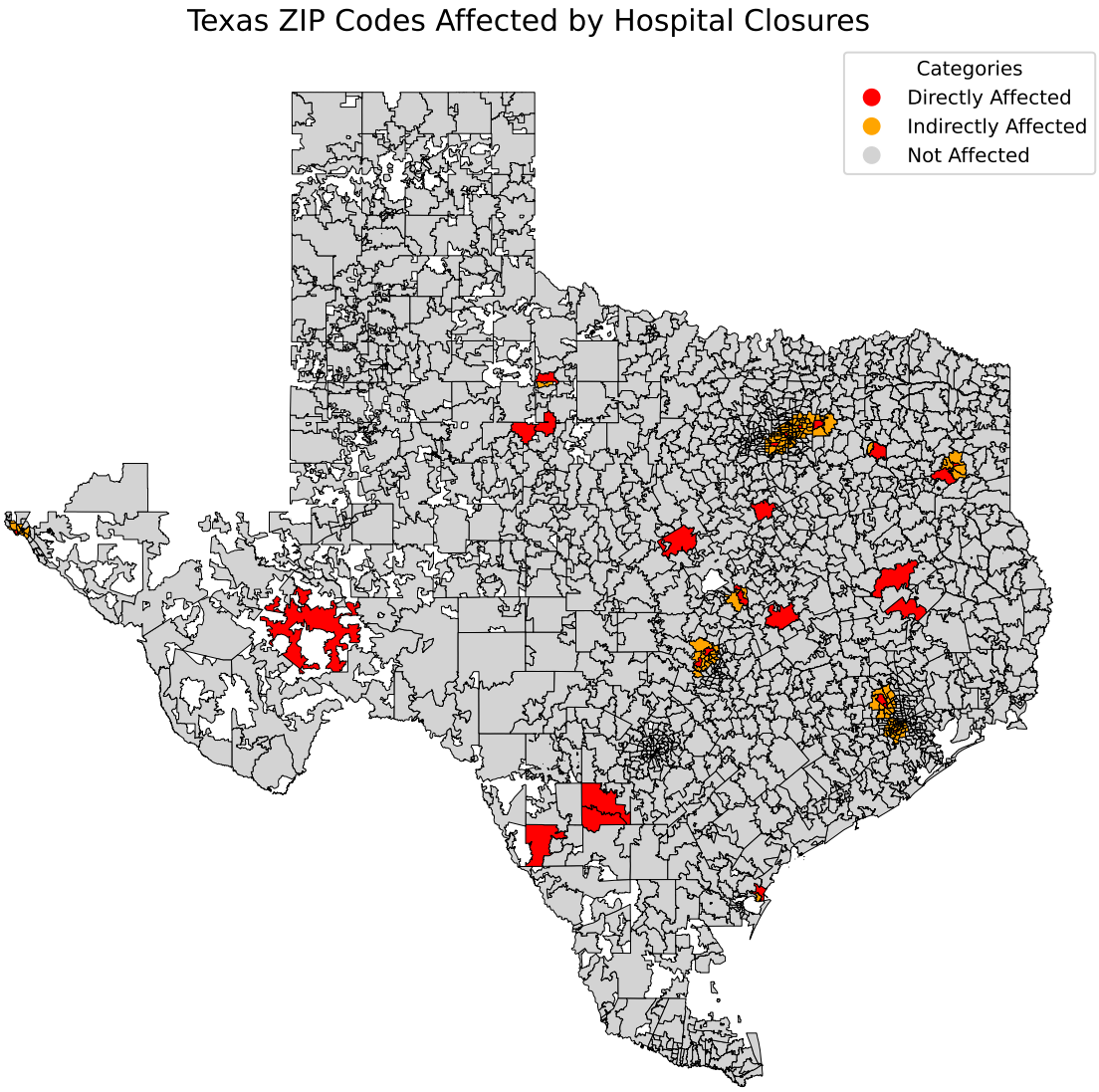
```
num_indirectly_affected = indirectly_affected_zips['ZCTA5'].nunique()

print("Number of indirectly affected ZIP codes in Texas:",
 ↪  num_indirectly_affected)
```

```
Number of indirectly affected ZIP codes in Texas: 192
```

4.

```
directly_affected_set = set(directly_affected_zips['ZCTA5'])
indirectly_affected_set = set(indirectly_affected_zips['ZCTA5'])

def categorize_zip(row):
    if row['ZCTA5'] in directly_affected_set:
        return 'Directly Affected'
    elif row['ZCTA5'] in indirectly_affected_set:
        return 'Indirectly Affected'
    else:
        return 'Not Affected'

texas_zip_codes_new = texas_zip_codes.copy()
texas_zip_codes_new['Category'] = texas_zip_codes_new.apply(categorize_zip,
 ↪  axis=1)
texas_zip_codes_new = texas_zip_codes_new.rename(columns={'Category':
 ↪  'Category_new'})

texas_categories = texas_zip_codes.merge(texas_zip_codes_new[['ZCTA5',
 ↪  'Category_new']], on='ZCTA5', how='left', suffixes=('', '_new'))

# Define color map for categories
color_map = {
    'Directly Affected': 'red',
    'Indirectly Affected': 'orange',
    'Not Affected': 'lightgrey'
}

fig, ax = plt.subplots(1, 1, figsize=(12, 10))
texas_categories.plot(
    column='Category_new',  # Use this column to define colors
    cmap=plt.cm.colors.ListedColormap(['red', 'orange', 'lightgrey']),  #
 ↪  Custom color map
```

```
    edgecolor='black',
    linewidth=0.5,
    ax=ax
)

handles = [plt.Line2D([0], [0], marker='o', color='w', label=key,
                      markerfacecolor=color_map[key], markersize=10) for key
 ↪  in color_map]
ax.legend(handles=handles, title='Categories')

plt.title("Texas ZIP Codes Affected by Hospital Closures", fontsize=15)
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.axis("off")
plt.show()
```



Reflecting on the exercise (10 pts)

a. We believe there are potential issues with excluding facilities that may have merged or changed names to address incorrectly identified closures. Firstly, some hospitals may have merged with others, which could reduce capacity or access in ways not fully captured by simply counting active hospitals. This method may still misclassify merged hospitals as "closed" if there's a lag in updating these changes in the dataset. Secondly, this type of merging method excludes certain years, meaning we might need additional data to ensure its accuracy. For example, we would need 2020 data to verify whether

closures identified in 2019 are accurate, but we currently lack that information. Additionally, a hospital may close its emergency (short-term) sector while remaining operational in other capacities. Simply counting active hospitals might not capture these nuances, potentially underestimating the true impact on healthcare access. We also have some ideas that might improve accuracy in confirming hospital closures. First, if we could access supplementary data, we could cross-check suspected closures with health regulatory bodies or databases that track hospital certifications, mergers, and closures to confirm whether a hospital actually ceased operations or was absorbed into another system. Secondly, rather than merely counting active hospitals, we could assess changes in capacity, especially in cases of mergers. For instance, tracking metrics such as the number of beds, emergency services, and key departments could provide insights into how healthcare capacity and accessibility have evolved.

b. Using a 10-mile buffer to define indirectly affected ZIP codes may not fully capture variations in healthcare access. This buffer is an arbitrary distance and doesn't account for factors like population density, travel time, or the presence of other healthcare facilities within the same radius. Additionally, geographic distance alone doesn't reflect real-world travel conditions, such as road infrastructure and travel time, which can significantly impact accessibility. Our method also doesn't account for hospital size and services; a ZIP code may be marked as "not affected" even if a smaller hospital in that area closed, potentially reducing access to specialized or high-demand services. There are several ways to address these limitations. Rather than a fixed buffer, we could use travel time, such as a 30-minute driving distance to the nearest hospital, derived from road network data, to provide a more realistic view of healthcare access—especially in rural areas. Incorporating population density data to calculate a Healthcare Accessibility Index, such as the number of hospitals per capita within a certain radius, could highlight areas with high populations but low hospital density, identifying those more severely impacted by closures. Additionally, distinguishing closures based on hospital capacity or type would be beneficial. For example, ZIP codes that lose a large hospital could be weighted more heavily than those losing smaller or less critical facilities.