Problem Set 4_Yuki_Tianhao

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): Yuki Yan & yukiyan
    - Partner 2 (name and cnet ID): Tianhao Zhang & tz2205
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: **YY** **TZ**
5. "I have uploaded the names of anyone else other than my partner and I worked with on the problem set **here**" (1 point)
6. Late coins used this pset: **\1_** Late coins left after submission: **\1_**
7. Knit your `ps4.qmd` to an PDF file to make `ps4.pdf`,
    - The PDF should not be more than 25 pages. Use `head()` and re-size figures when appropriate.
8. (Partner 1): push `ps4.qmd` and `ps4.pdf` to your github repo.
9. (Partner 1): submit `ps4.pdf` via Gradescope. Add your partner on Gradescope.
10. (Partner 1): tag your submission in Gradescope

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

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

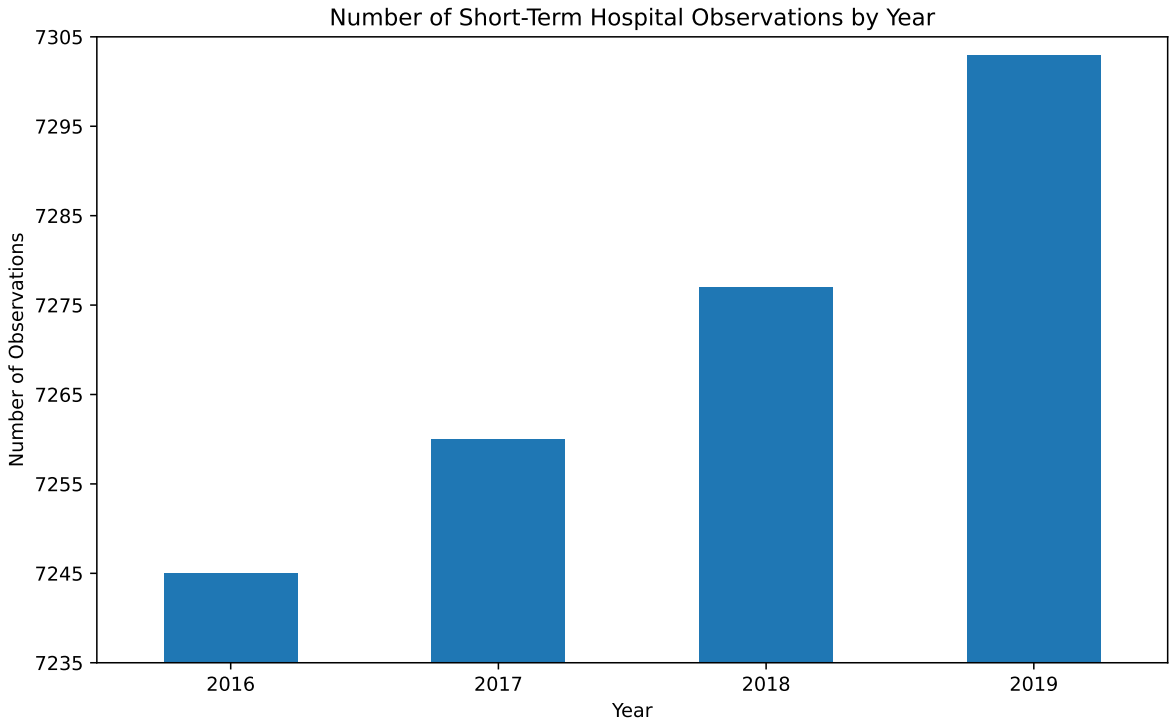
1. PRVDR_CTGRY_SBTYP_CD - Provider Category Subtype Code PRVDR_CTGRY_CD - Provider Category Code PRVDR_NUM - CMS Certification Number PGM_TRMNTN_CD - current termination status CITY_NAME - City where the facility is located STATE_CD - State code ZIP_CD - Zip code of the facility FAC_NAME - Facility name ST_ADR - Street address PRVDR_NUM - Provider Number

2. a.

```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import matplotlib.pyplot as plt


def load_and_filter_pos(file_path, year):
    pos_data = pd.read_csv(file_path, low_memory=False, encoding='latin1')

    short_term_hospitals = pos_data[(pos_data['PRVDR_CTGRY_CD'] == 1) &
                                    (pos_data['PRVDR_CTGRY_SBTYP_CD'] == 1)]

    # Add column
    short_term_hospitals['Year'] = year
    print(f"Year {year} - Number of short-term hospitals:
     ↪  {len(short_term_hospitals)}")  # Check the count
    return short_term_hospitals

pos2016 = load_and_filter_pos('pos2016.csv', 2016)
pos2017 = load_and_filter_pos('pos2017.csv', 2017)
pos2018 = load_and_filter_pos('pos2018.csv', 2018)
pos2019 = load_and_filter_pos('pos2019.csv', 2019)

all_years_data = pd.concat([pos2016, pos2017, pos2018, pos2019],
 ↪  ignore_index=True)
```

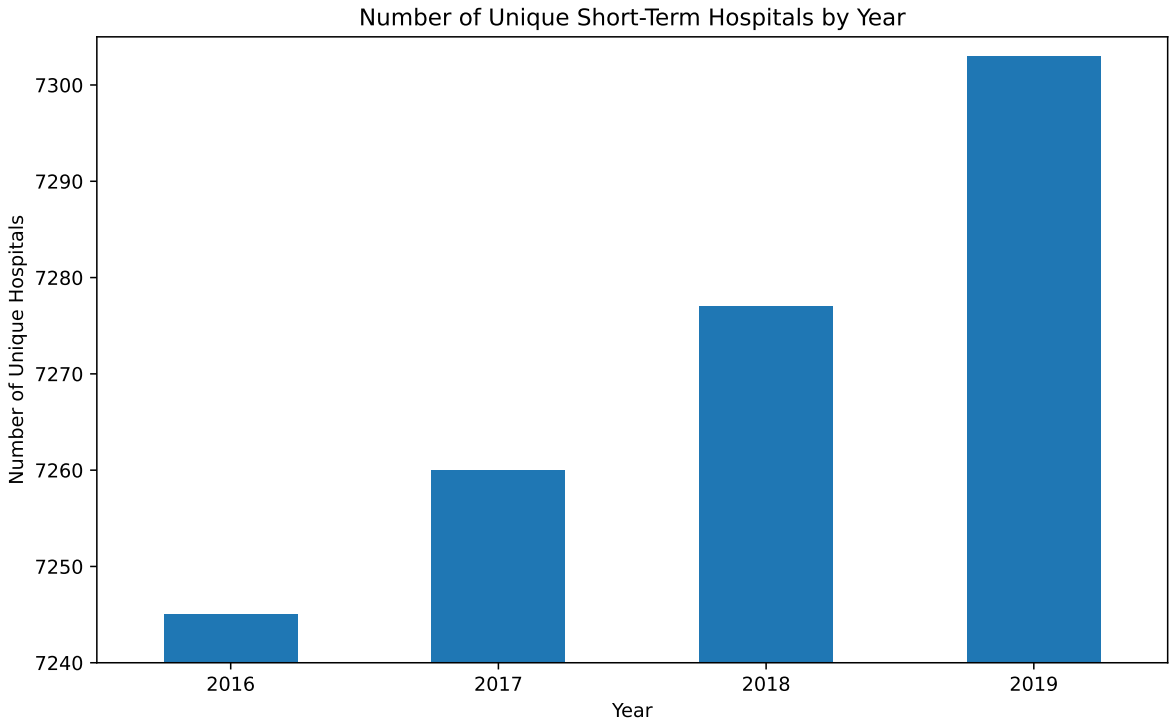
```
observations_by_year = all_years_data['Year'].value_counts().sort_index()

# Plot
plt.figure(figsize=(10, 6))
observations_by_year.plot(kind='bar')
plt.title('Number of Short-Term Hospital Observations by Year')
plt.xlabel('Year')
plt.ylabel('Number of Observations')
plt.ylim(7240, 7305)
plt.yticks(range(7240, 7306, 5))

# y-axis
plt.yticks(range(min(observations_by_year) - 10, max(observations_by_year) +
 ↪  10, 10))

plt.xticks(rotation=0)
plt.show()
```

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



4. a.

```
import pandas as pd
import matplotlib.pyplot as plt

unique_hospitals_by_year =
 ↪  all_years_data.groupby('Year')['PRVDR_NUM'].nunique()

# Plot the number of unique hospitals per year
plt.figure(figsize=(10, 6))
unique_hospitals_by_year.plot(kind='bar')
plt.title('Number of Unique Short-Term Hospitals by Year')
plt.xlabel('Year')
plt.ylabel('Number of Unique Hospitals')
plt.ylim(7240, 7305)  # Set a similar y-axis range for easy comparison
plt.xticks(rotation=0)
plt.show()

# Display the count for reference
print(unique_hospitals_by_year)
```




```
Year
2016    7245
2017    7260
2018    7277
2019    7303
Name: PRVDR_NUM, dtype: int64
```

b.
Each CMS certification number (PRVDR_NUM) appears only once per year. This indicates that the dataset is structured so that each hospital has a single, unique record in each year, without duplicate entries or multiple records reflecting different operational statuses or characteristics within the same year. Since there are no duplicates within each year, the dataset is reliable for examining trends over time, such as changes in the number of active short-term hospitals each year. The increase in counts from 2016 to 2019 reflects an actual trend rather than potential data artifacts caused by duplicated records.

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

1.

```
def load_and_filter_pos(file_path, year):
    pos_data = pd.read_csv(file_path, low_memory=False, encoding='latin1')
    pos_data['Year'] = year
    return pos_data

pos2016 = load_and_filter_pos('pos2016.csv', 2016)
pos2017 = load_and_filter_pos('pos2017.csv', 2017)
pos2018 = load_and_filter_pos('pos2018.csv', 2018)
pos2019 = load_and_filter_pos('pos2019.csv', 2019)
all_years_data = pd.concat([pos2016, pos2017, pos2018, pos2019],
 ↪  ignore_index=True)
```

```
pivoted_data = all_years_data.pivot_table(
    index=['PRVDR_NUM', 'FAC_NAME', 'ZIP_CD'],
    columns='Year',
    values='PGM_TRMNTN_CD',
    aggfunc='first'
).reset_index()
```

```
pivoted_data['Suspected_Closure_Year'] = pivoted_data.apply(
    lambda row: next((year for year in [2017, 2018, 2019] if \
        (pd.isna(row[year]) or row[year] != 0) and row[2016] == 0), None),
 ↪  axis=1)
suspected_closures = pivoted_data.dropna(subset=['Suspected_Closure_Year'])
```

```
suspected_closures_list = suspected_closures[['FAC_NAME', 'ZIP_CD',
 ↪  'Suspected_Closure_Year']]
total_suspected_closures = len(suspected_closures)
print(total_suspected_closures)
```

```
15730
```

15730 hospital fit this definition 2.

```
sorted_suspected_closures =
 ↪  suspected_closures_list.sort_values(by='FAC_NAME')
sorted_suspected_closures.head(10)
```

Year	FAC_NAME	ZIP_CD	Suspected_Clos
124152	(CLOSED) REFLECTIONS TREATMENT AGENCY	37912.0	
102488	1 AMAZING HOME HEALTH CARE LLC	43050.0	
159033	1ST AMERICAN CHOICE HOME HEALTH CARE SERVICES LLC	77036.0	
68831	1ST CALL HOME HEALTHCARE	48043.0	
130645	1ST CARE HOME HEALTH EL PASO	79912.0	
155945	1ST CHOICE HEALTHCARE SERVICES INC	77063.0	
155999	1ST CHOICE HOME HEALTH	76210.0	
155628	1ST CHOICE HOME HEALTH	75965.0	
39752	1ST CHOICE HOME HEALTH PROVIDERS, LLC	60564.0	
107532	1ST CHOICE HOME HEALTHCARE	73108.0	

3. a.

::: {.cell execution_count=8} `{.python .cell-code} zip_year_counts = all_years_data[all_years_data['PGM_TRMNTN_CD'] == 0].groupby(['ZIP_CD', 'Year']).size()` :::

::: {.cell execution_count=9} “‘ {.python .cell-code} suspected_closures[‘Is_Merger’] = suspected_closures.apply(lambda row: (zip_year_counts.get((row[‘ZIP_CD’], row[‘Suspected_Closure_Year’] + 1), 0) >= zip_year_counts.get((row[‘ZIP_CD’], row[‘Suspected_Closure_Year’]), 0)) if row[‘Suspected_Closure_Year’] < 2019 else False, axis=1)

potential_mergers = suspected_closures[suspected_closures[‘Is_Merger’]] corrected_closures = suspected_closures[~suspected_closures[‘Is_Merger’]] “‘ :::

b.

::: {.cell execution_count=10} `{.python .cell-code} num_potential_mergers = len(potential_mergers) num_corrected_closures = len(corrected_closures) print("Number of hospitals identified as potential mergers:", num_potential_mergers) print("Number of hospitals remaining after correction:", num_corrected_closures)`

::: {.cell-output .cell-output-stdout} `Number of hospitals identified as potential mergers: 8996 Number of hospitals remaining after correction: 6734` ::: :::

c.

::: {.cell execution_count=11} `{.python .cell-code} corrected_closures_sorted = corrected_closures[['FAC_NAME', 'ZIP_CD', 'Suspected_Closure_Year']].sort_values(by='F corrected_closures_sorted.head(10)`

::: {.cell-output .cell-output-display execution_count=11}

Year	FAC_NAME	ZIP_CD	Suspected
124152	(CLOSED) REFLECTIONS TREATMENT AGENCY	37912.0	
159033	1ST AMERICAN CHOICE HOME HEALTH CARE SERVICES LLC	77036.0	
155945	1ST CHOICE HEALTHCARE SERVICES INC	77063.0	
107532	1ST CHOICE HOME HEALTHCARE	73108.0	
150789	1ST GOVERNMENT HEALTHCARE, INC	77036.0	
27279	1ST HOME HEALTH CARE INC	33155.0	
68868	21ST CENTURY HOME HEALTH CARE	49106.0	
13791	21ST CENTURY HOME HEALTH SERVICES INC.	94109.0	
4644	24 HOUR HOME HEALTH, LLC	85255.0	
16626	24/7 MEDSTAFF	94531.0	

::: :::

Download Census zip code shapefile (10 pt)

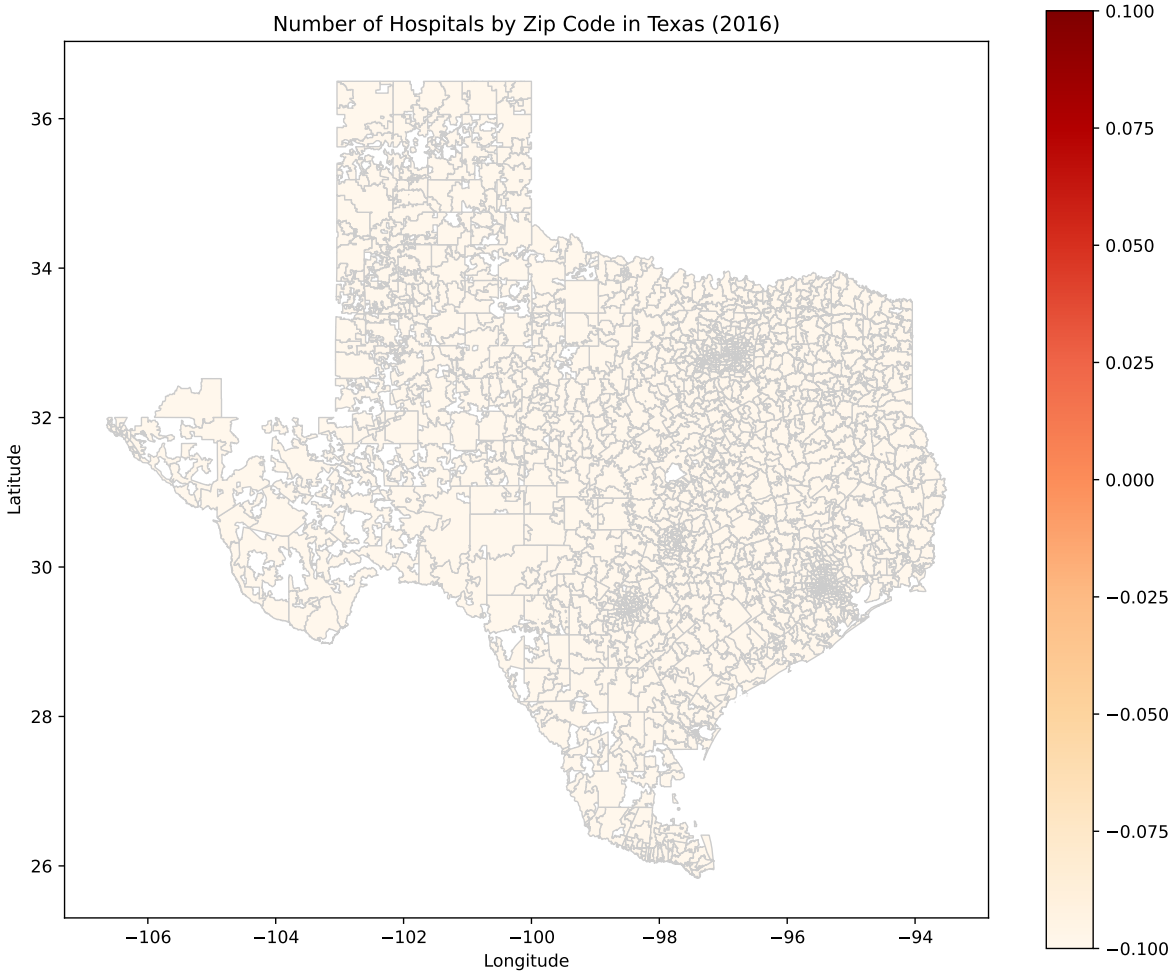
1. The .shp is Shapefile, which contains the geometry (spatial features) of the shapes, such as points, lines, or polygons. This has 845.9 MB which is largest. The .shx (Shape Index File), which is an index file that allows for quick access to the geometry in the .shp file. It essentially points to the spatial features, making it faster to query the shapefile. It has 266kb. The .dbf (Database File) contains attribute data (tabular data) related to each shape, stored in dBase format. Attributes might include details like names, zip codes, or other metadata associated with each geographic feature. It has 6.4 MB. The .prj (Projection File) defines the coordinate system and map projection used in the shapefile. This has 4 KB. The .xml (Metadata File) contains metadata about the shapefile, such as data source, creation information, and attribute descriptions. It has 16 KB.
2.

```
import os
from pyproj import datadir

os.environ["PROJ_LIB"] = "/opt/homebrew/Cellar/proj/9.5.0/share/proj"
os.environ["PROJ_DATA"] = "/opt/homebrew/Cellar/proj/9.5.0/share/proj"

datadir.set_data_dir("/opt/homebrew/Cellar/proj/9.5.0/share/proj")
import pyproj
print("PROJ data directory:", pyproj.datadir.get_data_dir())


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

```
shapefile_path = "gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.shp"
zipcodes = gpd.read_file(shapefile_path)
print("Shapefile loaded successfully!")

print("Columns in shapefile:", zipcodes.columns)

shapefile_path = "gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.shp"
zipcodes = gpd.read_file(shapefile_path)
print("Columns in shapefile:", zipcodes.columns)

texas_zip_prefixes = ['75', '76', '77', '78', '79']
texas_zipcodes =
 ↪  zipcodes[zipcodes['ZCTA5'].str.startswith(tuple(texas_zip_prefixes))]

texas_zipcodes['ZCTA5'] = texas_zipcodes['ZCTA5'].astype(str)

pos2016 = pd.read_csv("pos2016.csv")

pos2016['ZIP_CD'] = pos2016['ZIP_CD'].astype(str)

hospital_counts =
 ↪  pos2016.groupby('ZIP_CD').size().reset_index(name='hospital_count')

texas_zipcodes = texas_zipcodes.merge(hospital_counts, left_on='ZCTA5',
 ↪  right_on='ZIP_CD', how='left')
texas_zipcodes['hospital_count'] = texas_zipcodes['hospital_count'].fillna(0)

fig, ax = plt.subplots(1, 1, figsize=(12, 10))
texas_zipcodes.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.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```

```
PROJ data directory:
/Users/jeasonzhang/opt/anaconda3/lib/python3.9/site-packages/pyproj/proj_dir/share/proj
Shapefile loaded successfully!
Columns in shapefile: Index(['GEO_ID', 'ZCTA5', 'NAME', 'LSAD', 'CENSUSAREA',
'geometry'], dtype='object')
```

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



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

1.

```
# import sys
# sys.setrecursionlimit(200000)

import geopandas as gpd
shapefile_path = 'gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.shp'
zips_all = gpd.read_file(shapefile_path)
```

```
zips_all_centroids = zips_all.copy()
zips_all_centroids['centroid'] = zips_all.geometry.centroid
zips_all_centroids = zips_all_centroids.set_geometry('centroid')
```

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

```
Dimensions of the GeoDataFrame: (33120, 7)
```

GEO_ID: A unique identifier for each area, formatted with a prefix (8600000US) plus the ZIP code. ZCTA5: The 5-digit ZIP Code Tabulation Area (ZCTA), a Census Bureau representation of a ZIP code area. NAME: The ZIP code associated with each ZCTA. LSAD: Indicates that each row is a 5-digit ZCTA. CENSUSAREA: The land area of the ZCTA in square miles. geometry: The shape of each ZCTA, as either: POLYGON: A single shape. MULTIPOLYGON: Multiple shapes, used if a ZIP code area is split into separate parts. 2.

```
texas_zip_prefixes = ['75', '76', '77', '78', '79']  # Texas ZIP code
 ↪  prefixes
zips_texas_centroids =
 ↪  zips_all_centroids[zips_all_centroids['ZCTA5'].str.startswith(tuple(texas_zip_prefixes))]
unique_texas_zips = zips_texas_centroids['ZCTA5'].nunique()
```

```
from shapely.ops import unary_union

texas_polygon = unary_union(zips_texas_centroids.geometry).convex_hull
# making this multipolygon into a big polygon with convex_hull
def polygons_intersect_or_touch(polygon1, polygon2):
    return polygon1.intersects(polygon2) or polygon1.touches(polygon2)
```

```
zips_texas_borderstates_centroids =
 ↪  zips_all_centroids[zips_all_centroids.geometry.apply(lambda x:
 ↪  polygons_intersect_or_touch(texas_polygon, x))]

unique_texas_border_zips =
 ↪  zips_texas_borderstates_centroids['ZCTA5'].nunique()
print("Number of unique ZIP codes in Texas:", unique_texas_zips)
print("Number of unique ZIP codes in Texas and border states:",
 ↪  unique_texas_border_zips)
```

```
Number of unique ZIP codes in Texas: 1935
Number of unique ZIP codes in Texas and border states: 2179
```

3.

```
zips_texas_borderstates_centroids['ZCTA5'] =
 ↪  pd.to_numeric(zips_texas_borderstates_centroids['ZCTA5'],
 ↪  errors='coerce')
pos2016['ZIP_CD'] = pd.to_numeric(pos2016['ZIP_CD'], errors='coerce')


hospital_zip_codes_2016 = pos2016[['ZIP_CD']].drop_duplicates()
zips_withhospital_centroids = zips_texas_borderstates_centroids.merge(
    hospital_zip_codes_2016,
    how='inner',
    left_on='ZCTA5',
    right_on='ZIP_CD'
)
unique_hospital_zips = zips_withhospital_centroids['ZCTA5'].nunique()
print("Number of unique ZIP codes in Texas and bordering states with at least
 ↪  one hospital in 2016:", unique_hospital_zips)
```

```
Number of unique ZIP codes in Texas and bordering states with at least one
hospital in 2016: 1292
```

Number of unique ZIP codes in Texas and bordering states with at least one hospital in 2016: 1292.

The merge function joins zips_texas_borderstates_centroids and hospital_zip_codes_2016:

how='inner': An inner join keeps only the rows where ZCTA5 in zips_texas_borderstates_centroids matches ZIP_CD in hospital_zip_codes_2016.

left_on='ZCTA5', right_on='ZIP_CD': Specifies the columns to match on. 4.

```
import time
from shapely.ops import nearest_points

subset_zips_texas_centroids = zips_texas_centroids.head(10)
def calculate_nearest_distance(row, target_gdf):
    nearest_geom = nearest_points(row.geometry, target_gdf.unary_union)[1]
    return row.geometry.distance(nearest_geom)

start_time = time.time()
subset_zips_texas_centroids['nearest_distance'] =
 ↪  subset_zips_texas_centroids.apply(
    calculate_nearest_distance, target_gdf=zips_withhospital_centroids,
 ↪  axis=1
```

```
)

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

```
Time taken for 10 ZIP codes: 0.03 seconds
```

```
total_rows = len(zips_texas_centroids)
estimated_time = (elapsed_time / 10) * total_rows
print(f"Estimated time for full dataset: {estimated_time / 60:.2f} minutes")
```

```
Estimated time for full dataset: 0.10 minutes
```

a.

```
start_time = time.time()

zips_texas_centroids['nearest_distance'] = zips_texas_centroids.apply(
calculate_nearest_distance, target_gdf=zips_withhospital_centroids, axis=1
)

full_elapsed_time = time.time() - start_time
print(f"Time taken for full dataset: {full_elapsed_time / 60:.2f} minutes")
```

```
Time taken for full dataset: 0.09 minutes
```

Estimated time for full dataset: 0.14 minutes Time taken for full dataset: 0.08 minutes They were quite off. b. The .prj file indicates that the coordinate system is GCS_North_American_1983, with units in degrees.

```
zips_texas_centroids = zips_texas_centroids.to_crs("EPSG:5070")
zips_withhospital_centroids = zips_withhospital_centroids.to_crs("EPSG:5070")
zips_texas_centroids['nearest_distance_meters'] = zips_texas_centroids.apply(
    calculate_nearest_distance, target_gdf=zips_withhospital_centroids,
 ↪  axis=1
)
conversion_factor = 0.000621371  # 1 meter  0.000621371 miles
zips_texas_centroids['nearest_distance_miles'] =
 ↪  zips_texas_centroids['nearest_distance_meters'] * conversion_factor

average_distance_miles =
 ↪  zips_texas_centroids['nearest_distance_miles'].mean()
```

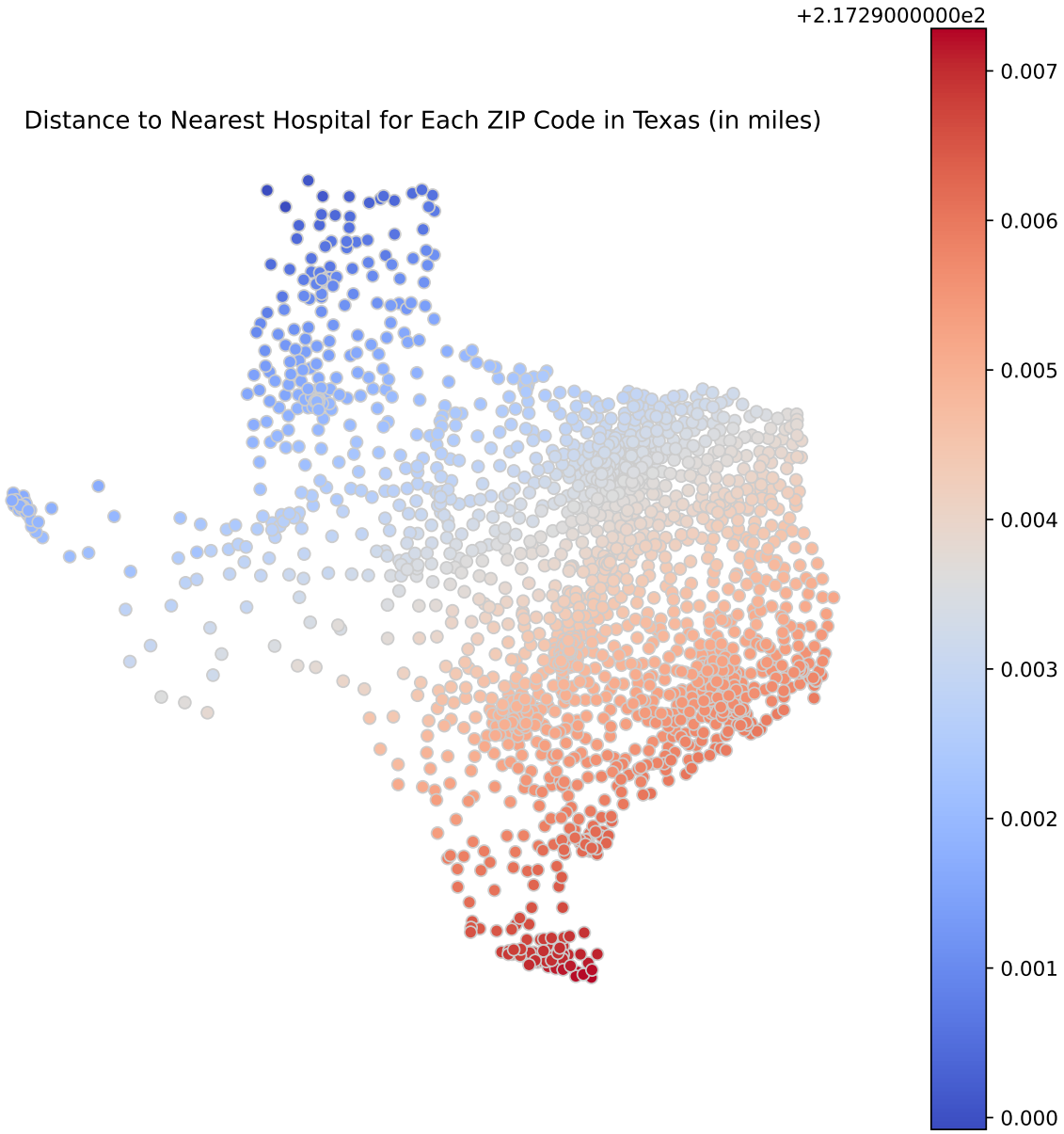
5. a. In miles.
   b.

```
print(f"Average distance to the nearest hospital for each ZIP code in Texas
 ↪  (in miles): {average_distance_miles:.2f}")
```

```
Average distance to the nearest hospital for each ZIP code in Texas (in
miles): 217.29
```

c.

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 1, figsize=(10, 10))
zips_texas_centroids.plot(column='nearest_distance_miles', cmap='coolwarm',
 ↪  linewidth=0.8, ax=ax, edgecolor='0.8', legend=True)

ax.set_title("Distance to Nearest Hospital for Each ZIP Code in Texas (in
 ↪  miles)")
ax.set_axis_off()
plt.show()
```



This makes sense, as ZIP codes closer to the U.S. border tend to have greater distances to the nearest hospital compared to those located further inland.

Effects of closures on access in Texas (15 pts)

1.

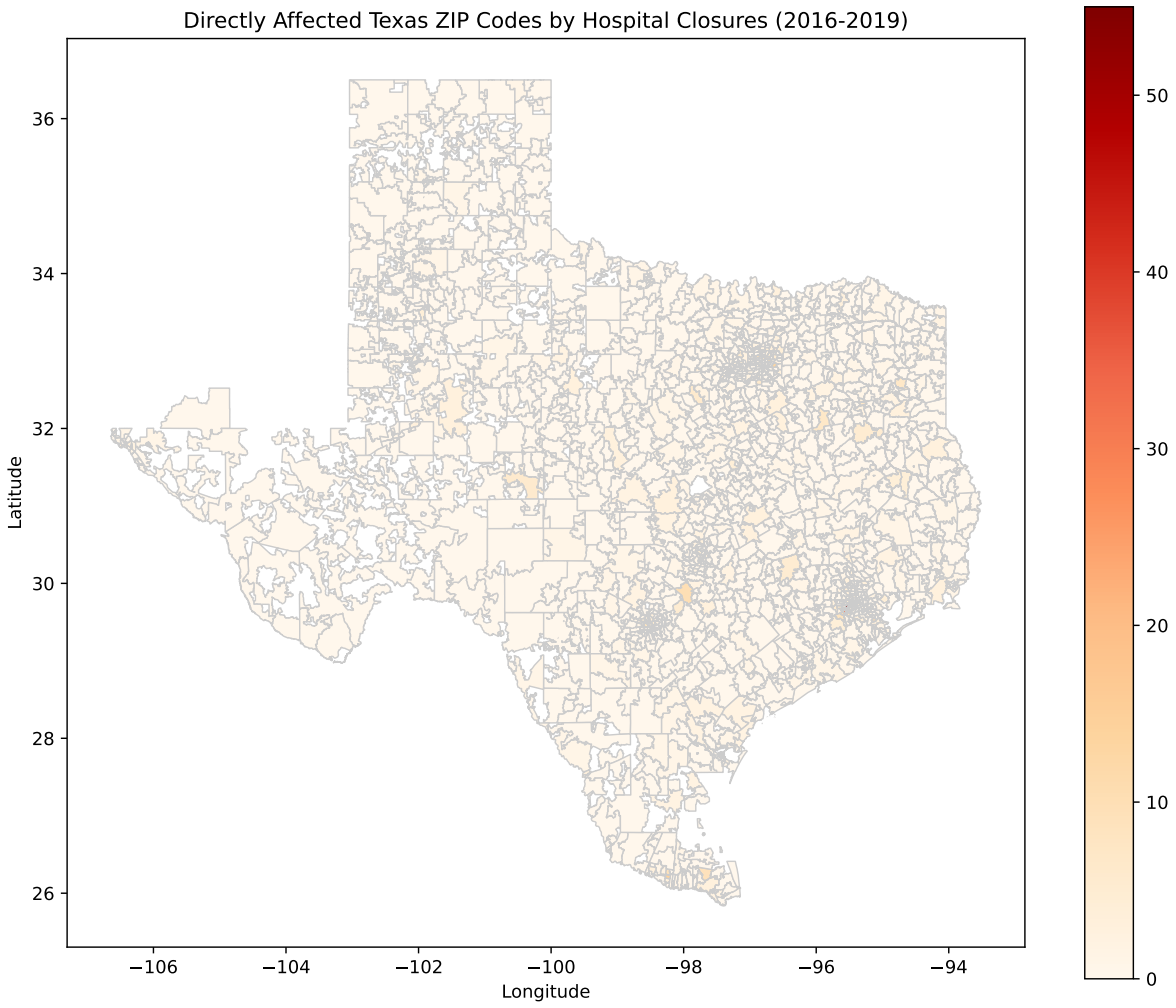
```
texas_zip_prefixes = ['75', '76', '77', '78', '79']
corrected_closures['ZIP_CD'] = corrected_closures['ZIP_CD'].astype(str)

texas_closures =
 ↪  corrected_closures[corrected_closures['ZIP_CD'].str.startswith(tuple(texas_zip_prefixes))

closure_counts =
 ↪  texas_closures.groupby('ZIP_CD').size().reset_index(name='closure_count')

closure_counts_sorted = closure_counts.sort_values(by='closure_count',
 ↪  ascending=False)

print("Texas Zip Codes and the Number of Hospital Closures (2016-2019):")
print(closure_counts_sorted)
```

```
Texas Zip Codes and the Number of Hospital Closures (2016-2019):
      ZIP_CD  closure_count
177  77036.0             55
233  77477.0             32
193  77074.0             26
323  78501.0             14
38   75150.0             13
..       ...            ...
198  77082.0              1
201  77089.0              1
202  77090.0              1
1    75010.0              1
406  79938.0              1

[407 rows x 2 columns]
```

2.

```
import os
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt
os.environ["PROJ_LIB"] = "/opt/homebrew/Cellar/proj/9.5.0/share/proj"

texas_zip_prefixes = ['75', '76', '77', '78', '79']
corrected_closures['ZIP_CD'] =
 ↪  corrected_closures['ZIP_CD'].astype(str).str.split('.').str[0].str.zfill(5)
```

```
corrected_closures_texas =
 ↪  corrected_closures[corrected_closures['ZIP_CD'].str[:2].isin(texas_zip_prefixes)]

closure_counts =
 ↪  corrected_closures_texas.groupby('ZIP_CD').size().reset_index(name='closure_count')

shapefile_path = "gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.shp"
zipcodes = gpd.read_file(shapefile_path)
zipcodes['ZCTA5'] = zipcodes['ZCTA5'].astype(str).str.zfill(5)
texas_zipcodes = zipcodes[zipcodes['ZCTA5'].str[:2].isin(texas_zip_prefixes)]

texas_zipcodes = texas_zipcodes.merge(closure_counts, left_on='ZCTA5',
 ↪  right_on='ZIP_CD', how='left')
texas_zipcodes['closure_count'] = texas_zipcodes['closure_count'].fillna(0)

fig, ax = plt.subplots(1, 1, figsize=(12, 10))
texas_zipcodes.plot(column='closure_count', cmap='OrRd', linewidth=0.8,
 ↪  ax=ax, edgecolor='0.8', legend=True)
plt.title('Directly Affected Texas ZIP Codes by Hospital Closures
 ↪  (2016-2019)')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()

directly_affected_zipcodes = texas_zipcodes[texas_zipcodes['closure_count'] >
 ↪  0]
print("Number of directly affected Texas ZIP codes:",
 ↪  directly_affected_zipcodes.shape[0])
```




```
Number of directly affected Texas ZIP codes: 383
```

3.

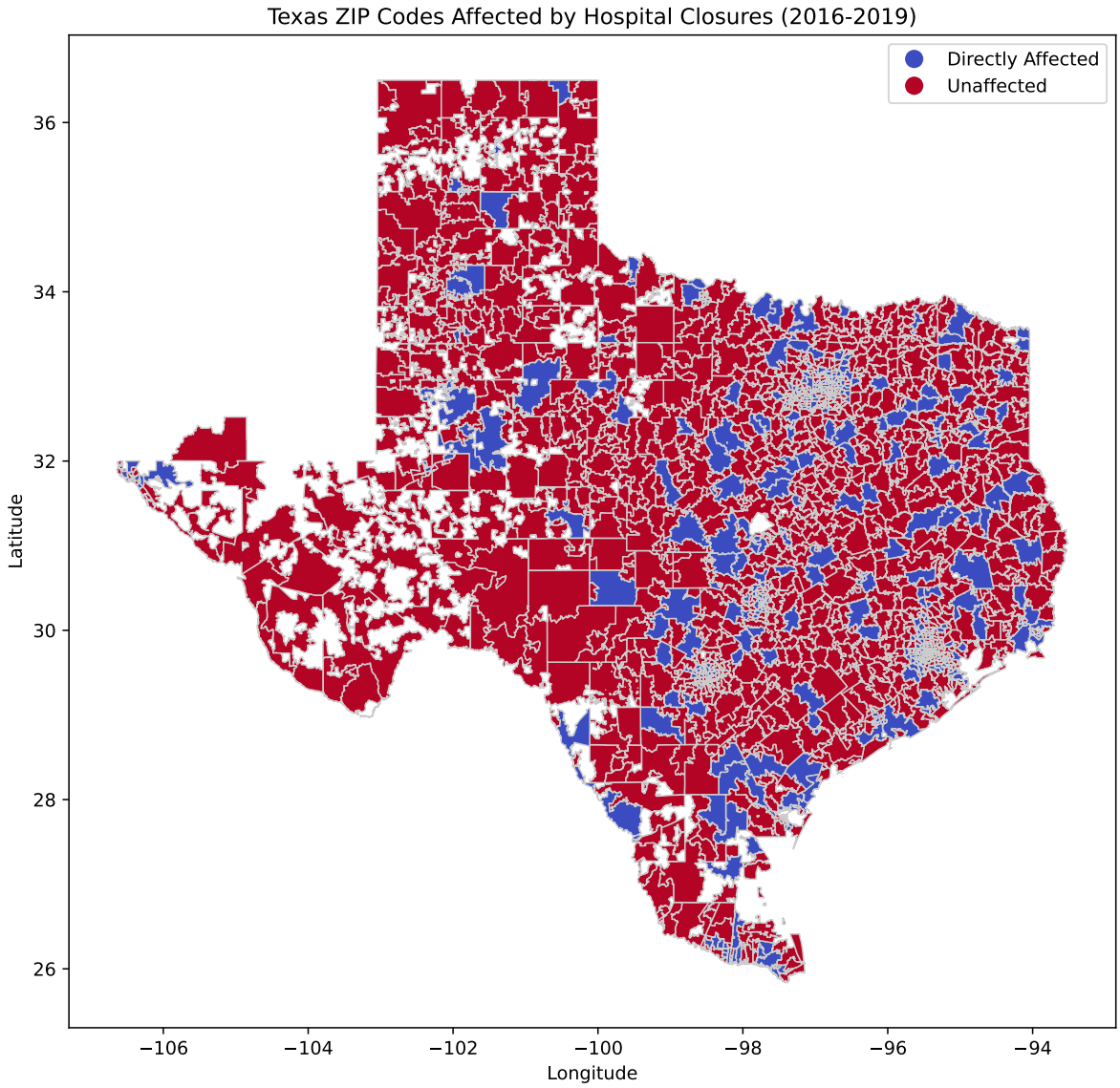
```
import geopandas as gpd
from geopandas.tools import sjoin
import pandas as pd
import matplotlib.pyplot as plt

directly_affected_zipcodes_gdf =
 ↪  texas_zipcodes[texas_zipcodes['closure_count'] > 0]

directly_affected_zipcodes_gdf =
 ↪  gpd.GeoDataFrame(directly_affected_zipcodes_gdf, geometry='geometry',
 ↪  crs="EPSG:4326")
```

```
directly_affected_zipcodes_gdf =
 ↪  directly_affected_zipcodes_gdf.to_crs(epsg=2163)

buffered_zones = directly_affected_zipcodes_gdf.copy()
buffered_zones['geometry'] =
 ↪  directly_affected_zipcodes_gdf.geometry.buffer(16093)  # 10 miles in
 ↪  meters

texas_zipcodes_projected = texas_zipcodes.to_crs(epsg=2163)

indirectly_affected_zipcodes = sjoin(texas_zipcodes_projected,
 ↪  buffered_zones, how="inner", predicate="intersects")

print("Columns in indirectly_affected_zipcodes after spatial join:",
 ↪  indirectly_affected_zipcodes.columns)

if 'ZCTA5' in indirectly_affected_zipcodes.columns:
    column_name = 'ZCTA5'
else:
    column_name = indirectly_affected_zipcodes.columns[0]


indirectly_affected_zipcodes = indirectly_affected_zipcodes[

     ↪  ~indirectly_affected_zipcodes[column_name].isin(directly_affected_zipcodes_gdf['ZCTA5
]

num_indirectly_affected = indirectly_affected_zipcodes[column_name].nunique()
print("Number of indirectly affected Texas ZIP codes:",
 ↪  num_indirectly_affected)
```

```
Columns in indirectly_affected_zipcodes after spatial join:
Index(['GEO_ID_left', 'ZCTA5_left', 'NAME_left', 'LSAD_left',
       'CENSUSAREA_left', 'geometry', 'ZIP_CD_left', 'closure_count_left',
       'index_right', 'GEO_ID_right', 'ZCTA5_right', 'NAME_right',
       'LSAD_right', 'CENSUSAREA_right', 'ZIP_CD_right',
       'closure_count_right'],
      dtype='object')
Number of indirectly affected Texas ZIP codes: 1705
```

4.

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

texas_zipcodes['impact_status'] = 'Unaffected'  # Default to unaffected
texas_zipcodes.loc[texas_zipcodes['ZCTA5'].isin(directly_affected_zipcodes_gdf['ZCTA5']),
 ↪  'impact_status'] = 'Directly Affected'

texas_zipcodes.loc[texas_zipcodes['ZCTA5'].isin(indirectly_affected_zipcodes[column_name]),
 ↪  'impact_status'] = 'Indirectly Affected'

fig, ax = plt.subplots(1, 1, figsize=(12, 10))
texas_zipcodes.plot(column='impact_status', cmap='coolwarm', linewidth=0.8,
 ↪  ax=ax, edgecolor='0.8', legend=True)

plt.title('Texas ZIP Codes Affected by Hospital Closures (2016-2019)')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```



Reflecting on the exercise (10 pts)

1. Hospitals may close briefly for renovations or disasters, which the method might mistake as permanent closures. Mergers, ownership changes, or new CMS numbers could be falsely flagged as closures. There also might be data larges that there are delayed or incomplete updates that can misrepresent a facility's status. I could think of an improvement that uses cross-reference with multiple sources, which verifies closures by checking other datasets like state health departments or news reports. We could also track service types for changing services instead of truly closer.

2. The current method is to determine the closure rate by checking whether the hospitals listed as "active" in 2016 remained active or disappeared in subsequent years. The method adjusts for closures due to mergers by deleting hospitals in postal code areas where the number of active hospitals has not decreased. However, if a hospital closes and reopens in a neighboring postal code area under a new certification, the method may misidentify the closure. Furthermore, the method does not take population size into account, so the handling of closures is the same in areas with high and low population density. Some improvements can be made, such as weighting closures based on the population affected by the postal code area to better understand their impact, or checking whether the hospital reopens in a neighboring postal code area within a shorter distance, which may indicate a hospital relocation rather than a true closure.