Problem Set 4 Submission

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):Pei-Chin Lu and peichin
    - Partner 2 (name and cnet ID): Sohyun Lim and shlim
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." **PL SL**
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: 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.

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

1. For this exercise, I reviewed the problem set and data dictionary to determine the essential variables needed from the `Provider of Services (POS)` dataset. Since we are focused on analyzing short-term hospitals, the selected variables include:

- **Provider Category Code (PRVDR_CTGRY_CD)**: Identifies the main category of the provider (e.g., hospital).

- **Provider Category Subtype Code (PRVDR_CTGRY_SBTYP_CD)**: Further categorizes providers, with code "01" used to filter short-term hospitals.
- **State Code (STATE_CD)**: Provides the state in which the provider is located, enabling regional analysis.
- **County Code (SSA_CNTY_CD)**: Indicates the county, for more detailed geographic filtering if needed.
- **Provider Number (PRVDR_NUM)**: A unique identification number for each provider, which allows for individual tracking.
- **Certification Date (CRTFCTN_DT)**: Records the date of certification, which could be used for analysis based on the certification timeline.
- **Termination Code (PGM_TRMNTN_CD)**: Shows whether the provider is still active, helping exclude terminated providers if necessary.
- **Facility Name (FAC_NAME)**: The name of the hospital, useful for identification.
- **City Name (CITY_NAME)**: The city where the provider is located, allowing for finer geographical analysis.
- **Zip code (ZIP_CD)**:ZIP Code (ZIP_CD): The five-digit ZIP code representing the provider's physical location, useful for precise regional analysis and mapping.

2.

```
import pandas as pd
file_path = r"/Users/sohyunlim/Desktop/python-ps4-shilm/pos2016.csv.csv"
data = pd.read_csv(file_path)
```

```
/var/folders/r1/ts7xr4hs55b3944wkk5s9_xr0000gn/T/ipykernel_24415/1683037145.py:3: DtypeWarning: Colum
  data = pd.read_csv(file_path)
```

a.

```
    # Filter for short-term hospitals with provider category code (PRVDR_CTGRY_CD) = 1 and
     ↪  subtype code (PRVDR_CTGRY_SBTYP_CD) = 1
short_term_hospitals = data[(data['PRVDR_CTGRY_CD'] == 1) & (data['PRVDR_CTGRY_SBTYP_CD'] ==
 ↪  1)]
    # Count the number of short-term hospitals
hospital_count = short_term_hospitals.shape[0]
print("Number of short-term hospitals:", hospital_count)
    # Check the number of short-term hospitals
hospital_count = short_term_hospitals.shape[0]
print("Number of short-term hospitals:", hospital_count)
```

```
Number of short-term hospitals: 7245
Number of short-term hospitals: 7245
```

b.

```
##Based on my analysis, I identified **7,245** short-term hospitals in the 2016 dataset, using `provi
##type code = 01` and `subtype code = 01`. This count is notably higher than external estimates,
##which report approximately **4,862** short-term acute care hospitals in 2016.

##The discrepancy may be due to differences in dataset definitions or the inclusion of additional fac
##types in the `pos2016.csv` file. Further examination is required to align the dataset criteria with
##standard definitions used in external sources.
```

##Reference:
##- Agency for Healthcare Research and Quality (AHRQ)

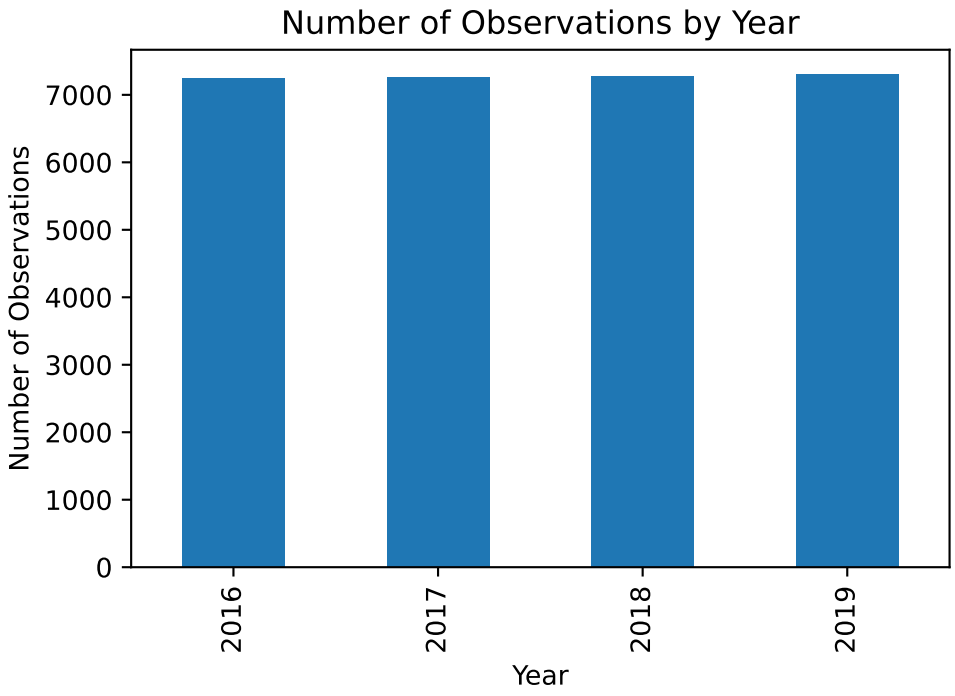
3.

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

# Define file paths for each year
file_paths = {
    '2016': r"/Users/sohyunlim/Desktop/python-ps4-shilm/pos2016.csv.csv",
    '2017': r"/Users/sohyunlim/Desktop/python-ps4-shilm/pos2017.csv.csv",
    '2018': r"/Users/sohyunlim/Desktop/python-ps4-shilm/pos2018.csv.csv",
    '2019': r"/Users/sohyunlim/Desktop/python-ps4-shilm/pos2019.csv.csv"
}

# Dictionary to store filtered data for each year
data = {}

# Loop through each file and check if it exists
for year, file_path in file_paths.items():
    if os.path.exists(file_path):
        try:
            # Load data with specified encoding
            df = pd.read_csv(file_path, encoding="ISO-8859-1", low_memory=False)

            # Filter for short-term hospitals: provider category code = 1 and subtype code = 1
            df_filtered = df[(df['PRVDR_CTGRY_CD'] == 1) & (df['PRVDR_CTGRY_SBTYP_CD'] ==
 ↪  1)].copy()
            df_filtered['Year'] = year  # Add year column

            # Store filtered data for each year
            data[year] = df_filtered
            print(f"Data for {year} loaded and filtered successfully.")

        except Exception as e:
            print(f"Error loading {file_path}: {e}")
    else:
        print(f"File for {year} does not exist: {file_path}")
```

```
Data for 2016 loaded and filtered successfully.
Data for 2017 loaded and filtered successfully.
Data for 2018 loaded and filtered successfully.
Data for 2019 loaded and filtered successfully.
```

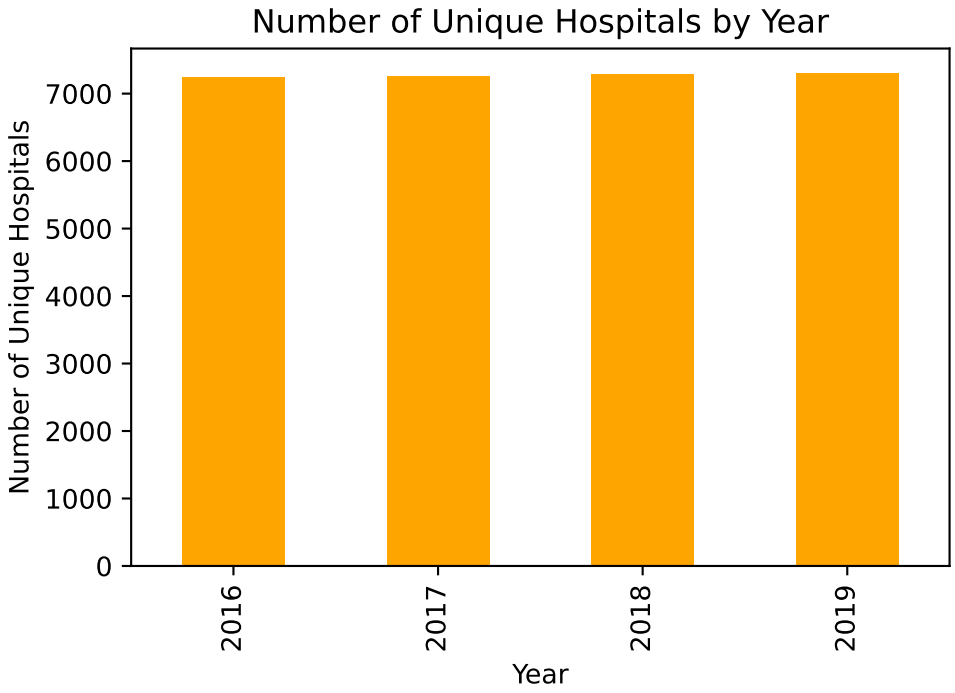
Check if any data was loaded

```
if data:
    # Combine all years into one DataFrame
    all_data = pd.concat(data.values(), ignore_index=True)

    # Question 3: Plot the number of observations by year
    observations_by_year = all_data['Year'].value_counts().sort_index()
    observations_by_year.plot(kind='bar', title="Number of Observations by Year")
    plt.xlabel('Year')
    plt.ylabel('Number of Observations')
    plt.show()
```



4

```
    # Question 4(a): Plot the number of unique hospitals by year (using 'PRVDR_NUM' as a
    ↪ unique identifier)
    unique_hospitals_by_year = all_data.groupby('Year')['PRVDR_NUM'].nunique()
    unique_hospitals_by_year.plot(kind='bar', color='orange', title="Number of Unique
↪   Hospitals by Year")
    plt.xlabel('Year')
    plt.ylabel('Number of Unique Hospitals')
    plt.show()
```




```
    # Question 4(b): Verification that each hospital appears only once per year
    # Count the occurrences of each hospital per year
    hospital_counts = all_data.groupby(['Year', 'PRVDR_NUM']).size()


    # Find cases where a hospital appears more than once in a given year
    duplicates = hospital_counts[hospital_counts > 1]

    # Check for duplicates
    if duplicates.empty:
        print("Each hospital appears only once per year. Data structure is valid.")
    else:
        print("Some hospitals appear multiple times per year. Below are the details:")
        print(duplicates)
```

```
Each hospital appears only once per year. Data structure is valid.
```

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

1. There are 174 hospitals that fit the definition of suspected closure.

```
# Convert the data type of Year from string to numeric
all_data['Year'] = pd.to_numeric(all_data['Year'], errors='coerce')

# Identify hospitals active in 2016
active_2016 = all_data[(all_data['Year'] == 2016) & (all_data['PGM_TRMNTN_CD'] == 0)]

# Initialize a list to store suspected closures
suspected_closures = []
```

```
# Check each hospital from 2016 for activity in subsequent years
for index, row in active_2016.iterrows():
    provider_id = row['PRVDR_NUM']
    facility_name = row['FAC_NAME']
    zip_code = row['ZIP_CD']

    # Get data for the provider in subsequent years
    provider_data = all_data[(all_data['PRVDR_NUM'] == provider_id) & (all_data['Year'] >
 ↪  2016)].sort_values('Year')

    # Initialize closure year as None
    closure_year = None

    # Check each year for termination
    for year, year_data in provider_data.groupby('Year'):
        if year_data['PGM_TRMNTN_CD'].iloc[0] != 0:  # If the provider is terminated
            closure_year = year
            break

    # If the provider has been identified as closed in subsequent years, add to the list
    if closure_year:
        suspected_closures.append([provider_id, facility_name, zip_code, closure_year])

# Convert results to DataFrame and display the number of closures
suspected_closures_df = pd.DataFrame(suspected_closures, columns=['provider_id',
 ↪  'facility_name', 'zip_code', 'suspected_closure_year'])
print("Number of suspected closures:", len(suspected_closures_df))
```

```
Number of suspected closures: 174
```

2.

```
# Sort suspected_closures_df by facility_name
closures_sorted = suspected_closures_df.sort_values(by='facility_name')

# Select only the facility_name and suspected_closure_year columns and display the top 10 rows
top_10_closures = closures_sorted[['facility_name', 'suspected_closure_year']].head(10)

# Display the result
print(top_10_closures)
```

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

3.

```
# Find the zip code whose number of hospitals has not decreased.
active_by_zip_year = all_data[all_data['PGM_TRMNTN_CD'] == 0].groupby(['ZIP_CD',
 ↪  'Year']).size().unstack(fill_value=0)

# Initialize lists to store confirmed closures and potential merger/acquisition lists
confirmed_closures_indexes = []

# Compare 2016 and 2018 to check if the number of hospitals has decreased due to the suspected
 ↪  closure in 2017 (A)
A = suspected_closures_df[
    (suspected_closures_df['suspected_closure_year'] == 2017) &
    (suspected_closures_df['zip_code'].map(lambda z: active_by_zip_year.loc[z, 2016] >
 ↪  active_by_zip_year.loc[z, 2018]))
].index.tolist()

# Compare 2017 and 2019 to check if the number of hospitals has decreased due to the suspected
 ↪  closure in 2018 (B)
B = suspected_closures_df[
    (suspected_closures_df['suspected_closure_year'] == 2018) &
    (suspected_closures_df['zip_code'].map(lambda z: active_by_zip_year.loc[z, 2017] >
 ↪  active_by_zip_year.loc[z, 2019]))
].index.tolist()

# Combine indices of hospitals confirmed to be closed in A and B, remove duplicates to create
 ↪  confirmed closures list
confirmed_closures_indexes = list(set(A + B))

# Convert confirmed closures and merger/acquisition lists to DataFrames
confirmed_closures_df = suspected_closures_df.loc[confirmed_closures_indexes]
merge_acquisition_df = suspected_closures_df.drop(index=confirmed_closures_indexes)

# Print the result
print("Number of confirmed closures after filtering:", len(confirmed_closures_df))
print("Number of hospitals identified as potential mergers/acquisitions:",
 ↪  len(merge_acquisition_df))
```

```
Number of confirmed closures after filtering: 91
Number of hospitals identified as potential mergers/acquisitions: 83
```

a.
  Among the suspected closures, 83 hospitals fit the definition of potentially being a merger/acquisi
b.
  After correcting this, 91 hospitals have left.
c.

```
# Report the first 10 rows of the result by name.
confirmed_closures_sorted_df = confirmed_closures_df.sort_values(by='facility_name',
 ↪  ascending=True)
print(confirmed_closures_sorted_df[['facility_name', 'zip_code',
 ↪  'suspected_closure_year']].head(10))
```

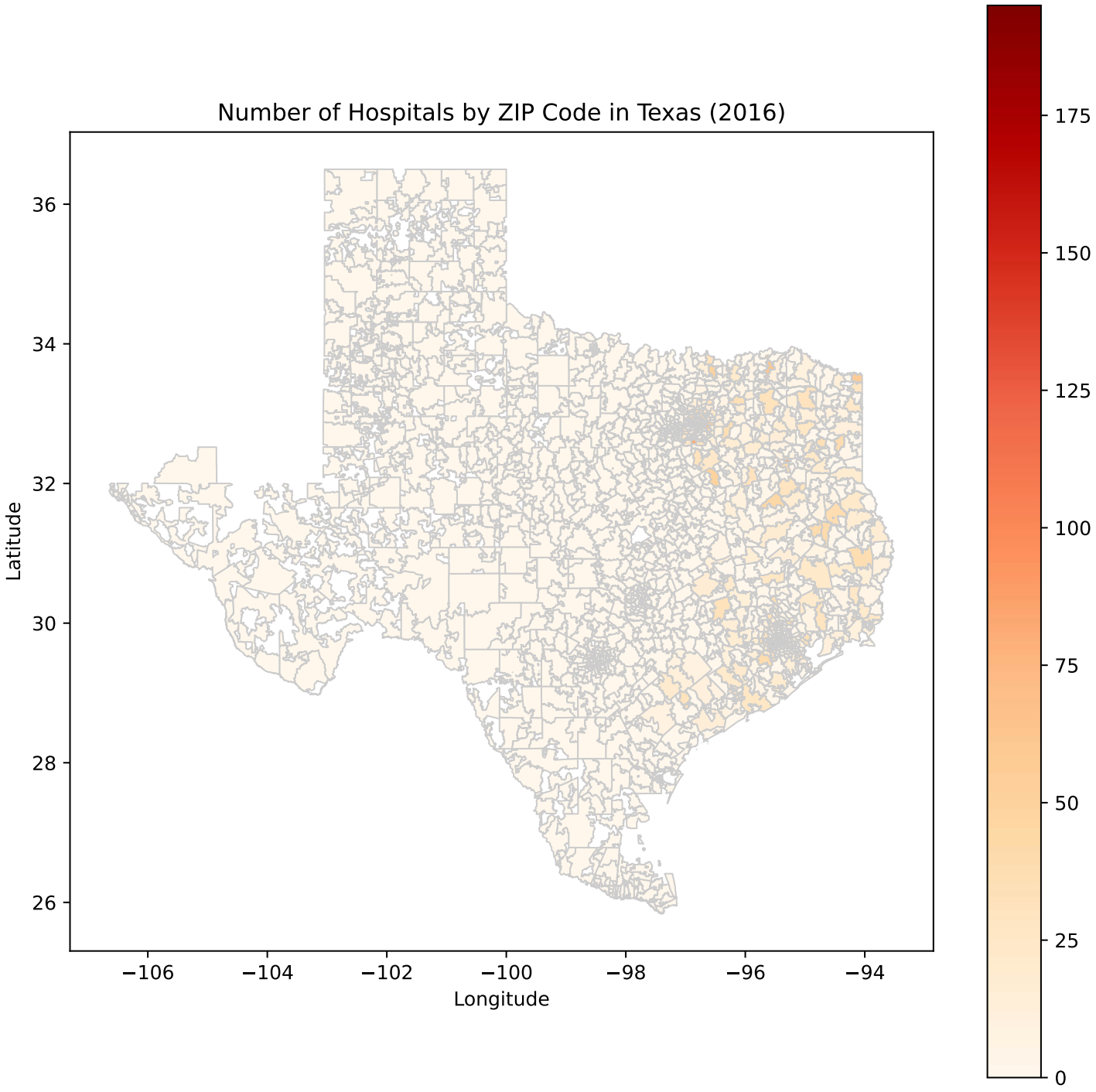
```
                                    facility_name  zip_code  \
4                           ABRAZO MARYVALE CAMPUS   85031.0
10       ADVENTIST MEDICAL CENTER - CENTRAL VALLEY   93230.0
97                         AFFINITY MEDICAL CENTER   44646.0
80   ALBANY MEDICAL CENTER / SOUTH CLINICAL CAMPUS   12208.0
140       ALLEGIANCE SPECIALTY HOSPITAL OF KILGORE   75662.0
21         ARKANSAS VALLEY REGIONAL MEDICAL CENTER   81050.0
69             BANNER CHURCHILL COMMUNITY HOSPITAL   89406.0
5                    BANNER PAYSON MEDICAL CENTER   85541.0
170          BAY AREA REGIONAL MEDICAL CENTER, LLC   77598.0
137    BAYLOR SCOTT & WHITE MEDICAL CENTER GARLAND   75042.0

     suspected_closure_year
4                      2017
10                     2017
97                     2018
80                     2017
140                    2017
21                     2017
69                     2017
5                      2018
170                    2018
137                    2018
```

Download Census zip code shapefile (10 pt)

1. a. The five file types found in the downloaded dataset are .shp, .shx, .dbf, .prj, and .xml. Each type serves a specific purpose: .shp stores the geometrical shapes of geographical features, .shx acts as an index file for faster access, .dbf contains attribute data related to each feature, .prj defines the projection and coordinate system, and .xml holds metadata describing the dataset.
   b. After unzipping the file, the sizes of each file type are as follows: the .shp file is approximately 514 MB, the .dbf file is about 6 MB, the .shx file is 259 KB, the .prj file is 1 KB, and the .xml file is 16 KB. This indicates that the .shp file holds the most data as it stores the detailed geometrical information of geographical features.
2.

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

# Load the ZIP code shapefile
shapefile_path =
 ↪  r"/Users/sohyunlim/Desktop/python-ps4-shilm/gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.shp"
zip_gdf = gpd.read_file(shapefile_path)

# Filter for Texas ZIP codes
texas_zip_gdf = zip_gdf[
    (zip_gdf['ZCTA5'].astype(str).str[:3].astype(int).between(750, 799)) |
    (zip_gdf['ZCTA5'].astype(str).str.startswith("733"))
]
```

```
# Load the cleaned POS hospital data for 2016
pos_data_path = r"/Users/sohyunlim/Desktop/python-ps4-shilm/pos2016.csv.csv"
pos_df = pd.read_csv(pos_data_path, dtype={'ZIP_CD': str}, low_memory=False)

# Filter POS data for Texas ZIP codes (using ZIP_CD)
texas_pos_df = pos_df[pos_df['ZIP_CD'].astype(str).str.startswith(('75', '77'))]

# Calculate the number of hospitals per ZIP code in Texas
hospital_counts = texas_pos_df['ZIP_CD'].value_counts().reset_index()
hospital_counts.columns = ['ZIP_CD', 'Hospital_Count']

# Merge with Texas ZIP shapefile data
texas_zip_gdf = texas_zip_gdf.merge(hospital_counts, left_on='ZCTA5', right_on='ZIP_CD',
 ↪  how='left').fillna(0)

# Plot choropleth of the number of hospitals by ZIP code in Texas
fig, ax = plt.subplots(1, 1, figsize=(10, 10))
texas_zip_gdf.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()
```



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

1. 1-1. The dimensions of the dimensions of the resulting GeoDataFrame is 33,120 rows and 6 columns. 1-2. The meaning of each columns is as follow:

- GEO_ID : A unique identifier for each ZIP code area
- ZCTA5 : ZIP Code Tabulation Area (ZCTA), a 5-digit code corresponding to the ZIP code
- NAME : Name of the ZCTA (usually the same as ZCTA5)
- LSAD : Legal/Statistical Area Description, indicating the legal or statistical description of the area
- CENSUSAREA : The size of the ZIP code area, often used in census data
- geometry : The centroid coordinates (latitude and longitude) of the ZIP code area

```
import geopandas as gpd
from shapely.geometry import Point

# Load ZIP code shapefile
shapefile_path =
 ↪  r"/Users/sohyunlim/Desktop/python-ps4-shilm/gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.shp"
gdf = gpd.read_file(shapefile_path)

# Calculate centroids for each ZIP code
zips_all_centroids = gdf.copy()
zips_all_centroids['geometry'] = zips_all_centroids.geometry.centroid

# Display dimensions and column explanations
print("Dimensions of zips_all_centroids:", zips_all_centroids.shape)
print("Columns in zips_all_centroids:", zips_all_centroids.columns)
```

```
/var/folders/r1/ts7xr4hs55b3944wkk5s9_xr0000gn/T/ipykernel_24415/3694902334.py:10: UserWarning: Geome

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

Dimensions of zips_all_centroids: (33120, 6)
Columns in zips_all_centroids: Index(['GEO_ID', 'ZCTA5', 'NAME', 'LSAD', 'CENSUSAREA', 'geometry'], d
```

2.

- Unique zip codes in zips_texas_centroids are 1,935.
- Unique zip codes in zips_texas_borderstates_centroids are 4,057.

```
# Filter ZIP codes for Texas (prefix: 750-799, including 733)
zips_texas_centroids = zips_all_centroids[
    zips_all_centroids['ZCTA5'].astype(str).str[:3].astype(int).between(750, 799) |
    (zips_all_centroids['ZCTA5'].astype(str).str[:3] == '733')
]

# Filter ZIP codes for Texas and bordering states (NM: 870-885, OK: 730-749 (except 733), AR:
 ↪  716-729, LA: 700-714)
zips_texas_borderstates_centroids = zips_all_centroids[
    zips_all_centroids['ZCTA5'].astype(str).str[:3].astype(int).between(700, 714) |  # LA
    zips_all_centroids['ZCTA5'].astype(str).str[:3].astype(int).between(716, 729) |  # AR
    (zips_all_centroids['ZCTA5'].astype(str).str[:3].astype(int).between(730, 749) &
     (zips_all_centroids['ZCTA5'].astype(str).str[:3] != '733')) |                   # OK
 ↪  excluding 733
    zips_all_centroids['ZCTA5'].astype(str).str[:3].astype(int).between(870, 885) |  # NM
    zips_all_centroids['ZCTA5'].astype(str).str[:3].astype(int).between(750, 799)    # TX
 ↪  including 750-799
]

# Print unique ZIP code counts
print("Unique ZIP codes in Texas:", zips_texas_centroids['ZCTA5'].nunique())
print("Unique ZIP codes in Texas and bordering states:",
 ↪  zips_texas_borderstates_centroids['ZCTA5'].nunique())
```

```
Unique ZIP codes in Texas: 1935
Unique ZIP codes in Texas and bordering states: 4057
```

3.

- I used inner merge to ensure that zips_withhospital_centroids contains only the ZIP codes in Texas and bordering states that had at least one hospital in 2016.
- Variable for Merging: I used ZIP_CD hospitals_2016 and ZCTA5 in zips_texas_borderstates_centroids, because they are representing zip codes that is useful to merge.

```
## Filter hospital data for active hospitals in 2016
hospitals_2016 = all_data[(all_data['Year'] == 2016) & (all_data['PGM_TRMNTN_CD'] == 0)]

# Extract unique ZIP codes where there is a hospital in 2016
zip_codes_with_hospitals_2016 = hospitals_2016['ZIP_CD'].unique()

## Ensure data format consistency - Convert ZIP codes to string and remove decimals
# Convert ZIP_CD in hospitals_2016 to string format and remove decimals
hospitals_2016.loc[:, 'ZIP_CD'] = hospitals_2016['ZIP_CD'].astype(int).astype(str)

# Convert ZCTA5 in zips_texas_borderstates_centroids to string format for consistency
zips_texas_borderstates_centroids.loc[:, 'ZCTA5'] =
 ↪  zips_texas_borderstates_centroids['ZCTA5'].astype(str)

# Convert zip_codes_with_hospitals_2016 to string format
zip_codes_with_hospitals_2016 = [str(int(zip_code)) for zip_code in
 ↪  zip_codes_with_hospitals_2016]

## Filter and verify results
# Check if there are any matching ZIP codes between zip_codes_with_hospitals_2016 and
 ↪  zips_texas_borderstates_centroids['ZCTA5']

common_zip_codes =
 ↪  set(zip_codes_with_hospitals_2016).intersection(set(zips_texas_borderstates_centroids['ZCTA5']))
print("Number of common ZIP codes:", len(common_zip_codes))
print("Sample common ZIP codes:", list(common_zip_codes)[:10])

# If matching ZIP codes are found, perform filtering
if common_zip_codes:
    zips_withhospital_centroids = zips_texas_borderstates_centroids[
        zips_texas_borderstates_centroids['ZCTA5'].isin(common_zip_codes)
    ]
else:
    print("No matching ZIP codes found between hospitals_2016 and
     ↪  zips_texas_borderstates_centroids.")

# Print the result
print("Dimensions of zips_withhospital_centroids:", zips_withhospital_centroids.shape)
print(zips_withhospital_centroids.head())
```

```
Number of common ZIP codes: 448
Sample common ZIP codes: ['75662', '88201', '76092', '72653', '70403', '76255', '71301', '75142', '73
Dimensions of zips_withhospital_centroids: (448, 6)
```

```
              GEO_ID  ZCTA5   NAME   LSAD  CENSUSAREA  \
8878  8600000US70043  70043  70043  ZCTA5       7.775
8897  8600000US70127  70127  70127  ZCTA5       7.095
8901  8600000US70301  70301  70301  ZCTA5     288.050
8908  8600000US70360  70360  70360  ZCTA5      64.325
8915  8600000US70403  70403  70403  ZCTA5      42.960

                       geometry
8878  POINT (-89.96276 29.94804)
8897  POINT (-89.97675 30.02501)
8901   POINT (-90.74089 29.8141)
8908  POINT (-90.81028 29.58819)
8915  POINT (-90.48388 30.48002)
```

```
/var/folders/r1/ts7xr4hs55b3944wkk5s9_xr0000gn/T/ipykernel_24415/2879250119.py:9: FutureWarning: Sett
  hospitals_2016.loc[:, 'ZIP_CD'] = hospitals_2016['ZIP_CD'].astype(int).astype(str)
```

4. a. It took approximately 0.04 seconds.

```python
import geopandas as gpd
import time

# Assuming zips_texas_centroids is already loaded
# Select a subset of 10 ZIP codes for testing
subset_texas_zips = zips_texas_centroids.head(10)

# Start timing
start_time = time.time()

# Calculate distance for each ZIP in the subset to the nearest ZIP with a hospital
nearest_hospital_test = gpd.sjoin_nearest(
    subset_texas_zips,
    zips_withhospital_centroids,
    how="left",
    distance_col="distance"
)

# Check end time and calculate time taken
end_time = time.time()
test_time = end_time - start_time

# Print the result
print(f"Time taken for 10 ZIP code test: {test_time:.2f} seconds")
```

```
Time taken for 10 ZIP code test: 0.02 seconds
```

```
/Users/sohyunlim/Desktop/Python 2 - inclass practice/myenv/lib/python3.12/site-packages/geopandas/arr
  warnings.warn(
```

b.

Since the full dataset (448 entries) is 4.5 times larger than the sample dataset (10), I expected it would take approximately 0.2 seconds (0.04 * 4.5). However, it actually only took 0.07 seconds. This is likely because the dataset size is relatively small, and the distance calculations are efficiently vectorized.

```
import geopandas as gpd
import time

# Start timing for the full dataset
start_time_full = time.time()

# Calculate distance for each ZIP in Texas to the nearest ZIP with a hospital
nearest_hospital_full = gpd.sjoin_nearest(
    zips_texas_centroids,
    zips_withhospital_centroids,
    how="left",
    distance_col="distance"
)

# Check end time and calculate time taken
end_time_full = time.time()
total_time_full = end_time_full - start_time_full

# Print the result
print(f"Total time taken for full calculation: {total_time_full:.2f} seconds")
print(nearest_hospital_full[['ZCTA5_left', 'ZCTA5_right', 'distance']].head())
```

```
Total time taken for full calculation: 0.02 seconds
     ZCTA5_left ZCTA5_right  distance
9207      78624       78624  0.000000
9208      78626       78664  0.167651
9209      78628       78681  0.110844
9210      78631       78028  0.337251
9211      78632       78629  0.219909

/Users/sohyunlim/Desktop/Python 2 - inclass practice/myenv/lib/python3.12/site-packages/geopandas/arr

  warnings.warn(
```

c.

Unit: degree

```
# Check .prj file to find the unit : degree
with
 ↪  open("/Users/sohyunlim/Desktop/python-ps4-shilm/gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.prj
 ↪  "r") as prj_file:
    prj_content = prj_file.read()
    print(prj_content)
```

```
GEOGCS["GCS_North_American_1983",DATUM["D_North_American_1983",SPHEROID["GRS_1980",6378137,298.257222
```

```
import numpy as np

# Calculate average latitude
average_latitude = zips_texas_centroids.geometry.y.mean()

# Conversion factors for latitude and longitude differences in miles
miles_per_degree_latitude = 69.0
miles_per_degree_longitude = 69.0 * np.cos(np.radians(average_latitude))

# Convert distances to miles based on latitude and longitude
nearest_hospital_full['distance_miles'] = nearest_hospital_full['distance'] * (
    miles_per_degree_latitude if 'latitude' in 'distance' else miles_per_degree_longitude
)

# Print a sample of the converted distances to verify
print("Average latitude for conversion:", average_latitude)
print(nearest_hospital_full[['ZCTA5_left', 'ZCTA5_right', 'distance',
 ↪  'distance_miles']].head())
```

```
Average latitude for conversion: 31.26293914563278
     ZCTA5_left ZCTA5_right  distance  distance_miles
9207      78624       78624  0.000000        0.000000
9208      78626       78664  0.167651        9.888185
9209      78628       78681  0.110844        6.537668
9210      78631       78028  0.337251       19.891345
9211      78632       78629  0.219909       12.970414
```

5. a. The unit is "mile" because we already converted the unit of the dataset from "degree" to "mile" in Q4.

```
import numpy as np

# Check if the 'distance_miles' column already exists, if not, calculate it
if 'distance_miles' not in nearest_hospital_full.columns:
    # Calculate average latitude for conversion
    average_latitude = nearest_hospital_full.geometry.y.mean()
    miles_per_degree_latitude = 69.0
    miles_per_degree_longitude = 69.0 * np.cos(np.radians(average_latitude))

    # Convert distances in degrees to miles if needed
    nearest_hospital_full['distance_miles'] = nearest_hospital_full.apply(
        lambda row: row['distance'] * miles_per_degree_latitude
        if row['ZCTA5_left'] == row['ZCTA5_right'] else row['distance'] *
         ↪  miles_per_degree_longitude,
        axis=1
    )

# Calculate the average distance to the nearest hospital for Texas ZIP codes
average_distance_miles = nearest_hospital_full['distance_miles'].mean()
print(f"Average distance to the nearest hospital in Texas (in miles):
 ↪  {average_distance_miles:.2f} miles")
```

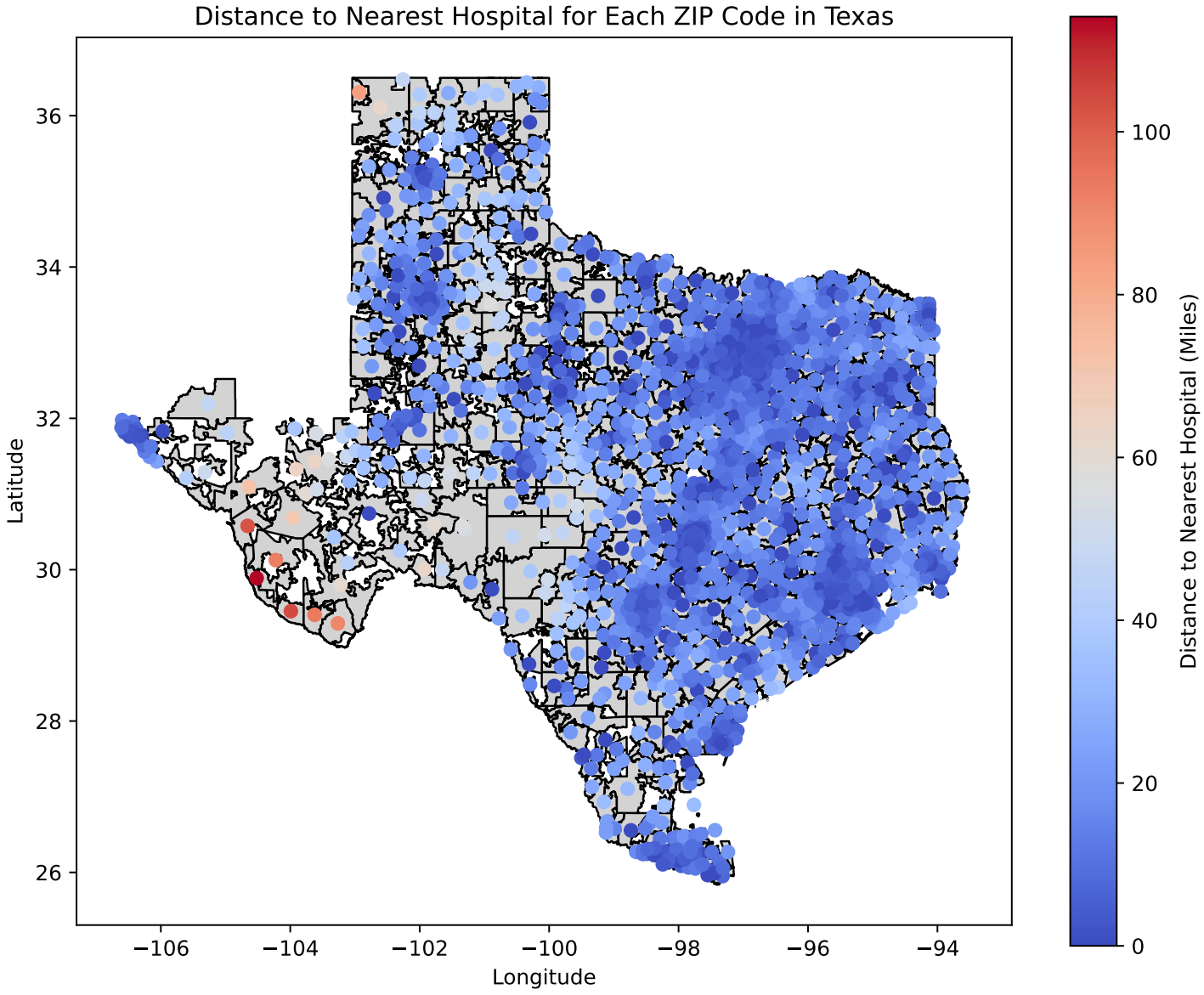
```
Average distance to the nearest hospital in Texas (in miles): 12.45 miles
```

b.

The average distance to the nearest hospital for each ZIP code in Texas is 12.45 miles. This is reasonable compared to the U.S. national average of around 10 miles, given Texas's vast rural areas and lower population density outside major cities.

c.

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

# Filter for Texas ZIP code range
texas_zip_codes = gdf[
    (gdf['ZCTA5'].astype(str).str[:3].astype(int).between(750, 799)) |
    (gdf['ZCTA5'].astype(str).str.startswith("733"))
]

# Check if texas_zip_codes and nearest_hospital_full are in EPSG:4269
if texas_zip_codes.crs != 'EPSG:4269':
    texas_zip_codes = texas_zip_codes.to_crs(epsg=4269)
if nearest_hospital_full.crs != 'EPSG:4269':
    nearest_hospital_full = nearest_hospital_full.to_crs(epsg=4269)

# Plot the map
fig, ax = plt.subplots(figsize=(10, 8))

# Plot Texas ZIP code boundaries as the background
texas_zip_codes.plot(ax=ax, color='lightgrey', edgecolor='black')

# Plot hospital distance data for each Texas ZIP code with color mapping in miles
nearest_hospital_full.plot(
    column='distance_miles',
    cmap='coolwarm',
    legend=True,
    ax=ax,
    legend_kwds={'label': "Distance to Nearest Hospital (Miles)"}
)

# Add title and axis labels
plt.title("Distance to Nearest Hospital for Each ZIP Code in Texas")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.show()
```



Effects of closures on access in Texas (15 pts)

1.

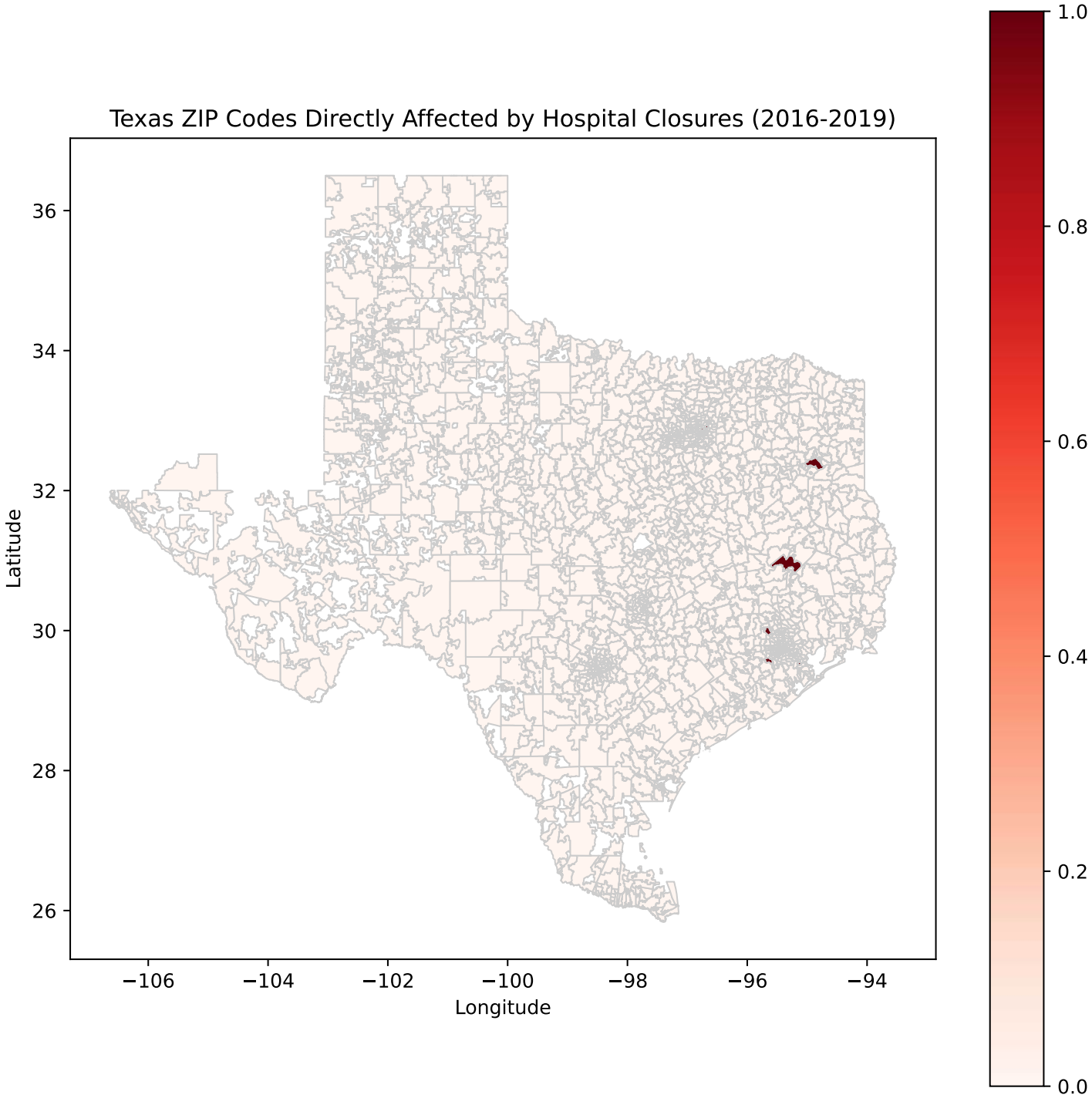
```
import pandas as pd

# Ensure 'zip_code' column is in string format and remove decimal places
confirmed_closures_df['zip_code'] =
 ↪  confirmed_closures_df['zip_code'].astype(float).astype(int).astype(str)

# Filter Texas ZIP codes (starting with '75' or '77') with closures from 2016-2019
directly_affected_zip_codes = confirmed_closures_df[
    confirmed_closures_df['zip_code'].str.startswith(('75', '77')) &
    confirmed_closures_df['suspected_closure_year'].between(2016, 2019)
]

# Count closures by ZIP code
closures_by_zip =
 ↪  directly_affected_zip_codes.groupby('zip_code').size().reset_index(name='closure_count')
```

```
# Display the result
print("Directly affected ZIP codes:", closures_by_zip)
print("Number of directly affected ZIP codes:", len(closures_by_zip))
```

```
Directly affected ZIP codes:   zip_code  closure_count
0    75042              1
1    75231              1
2    75662              1
3    75862              1
4    77035              1
5    77054              1
6    77429              1
7    77479              1
8    77598              1
Number of directly affected ZIP codes: 9
```

2.

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

# Load the Texas ZIP codes shapefile
zip_shapefile_path =
 ↪  r"/Users/sohyunlim/Desktop/python-ps4-shilm/gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.shp"
zip_gdf = gpd.read_file(zip_shapefile_path)

# Filter for Texas ZIP codes (starting with '75' or '77')
texas_zip_gdf = zip_gdf[
    (zip_gdf['ZCTA5'].astype(str).str[:3].astype(int).between(750, 799)) |
    (zip_gdf['ZCTA5'].astype(str).str.startswith("733"))
]



# Merge Texas ZIP codes with the directly affected ZIP codes data
affected_zip_gdf = texas_zip_gdf.merge(closures_by_zip, left_on='ZCTA5', right_on='zip_code',
 ↪  how='left')

# Replace NaN values in 'closure_count' with 0 (indicating no closures in those ZIP codes)
affected_zip_gdf['closure_count'] = affected_zip_gdf['closure_count'].fillna(0)

# Plot the choropleth map
fig, ax = plt.subplots(figsize=(10, 10))
affected_zip_gdf.plot(column='closure_count', cmap='Reds', linewidth=0.8, ax=ax,
 ↪  edgecolor='0.8', legend=True)
plt.title("Texas ZIP Codes Directly Affected by Hospital Closures (2016-2019)")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.show()
```



3.

```
import numpy as np

# Check .prj file to find the unit: degree
prj_file_path =
 ↪  r"/Users/sohyunlim/Desktop/python-ps4-shilm/gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.prj"
with open(prj_file_path, "r") as prj_file:
    prj_content = prj_file.read()
    print("Projection file content:\n", prj_content)
```

```
# Calculate average latitude from the centroids DataFrame
# Make sure zips_texas_centroids exists and has a geometry column
average_latitude = zips_texas_centroids.geometry.y.mean()

# Conversion factors for latitude and longitude differences in miles
miles_per_degree_latitude = 69.0
miles_per_degree_longitude = 69.0 * np.cos(np.radians(average_latitude))

# Convert distances in degrees to miles in nearest_hospital_full DataFrame
# Ensure 'distance' column exists in nearest_hospital_full DataFrame
nearest_hospital_full['distance_miles'] = nearest_hospital_full.apply(
    lambda row: row['distance'] * miles_per_degree_latitude
    if row['ZCTA5_left'] == row['ZCTA5_right'] else row['distance'] *
     ↪  miles_per_degree_longitude,
    axis=1
)

# Print a sample of the converted distances to verify
print("Average latitude for conversion:", average_latitude)
print(nearest_hospital_full[['ZCTA5_left', 'ZCTA5_right', 'distance',
 ↪  'distance_miles']].head())
```

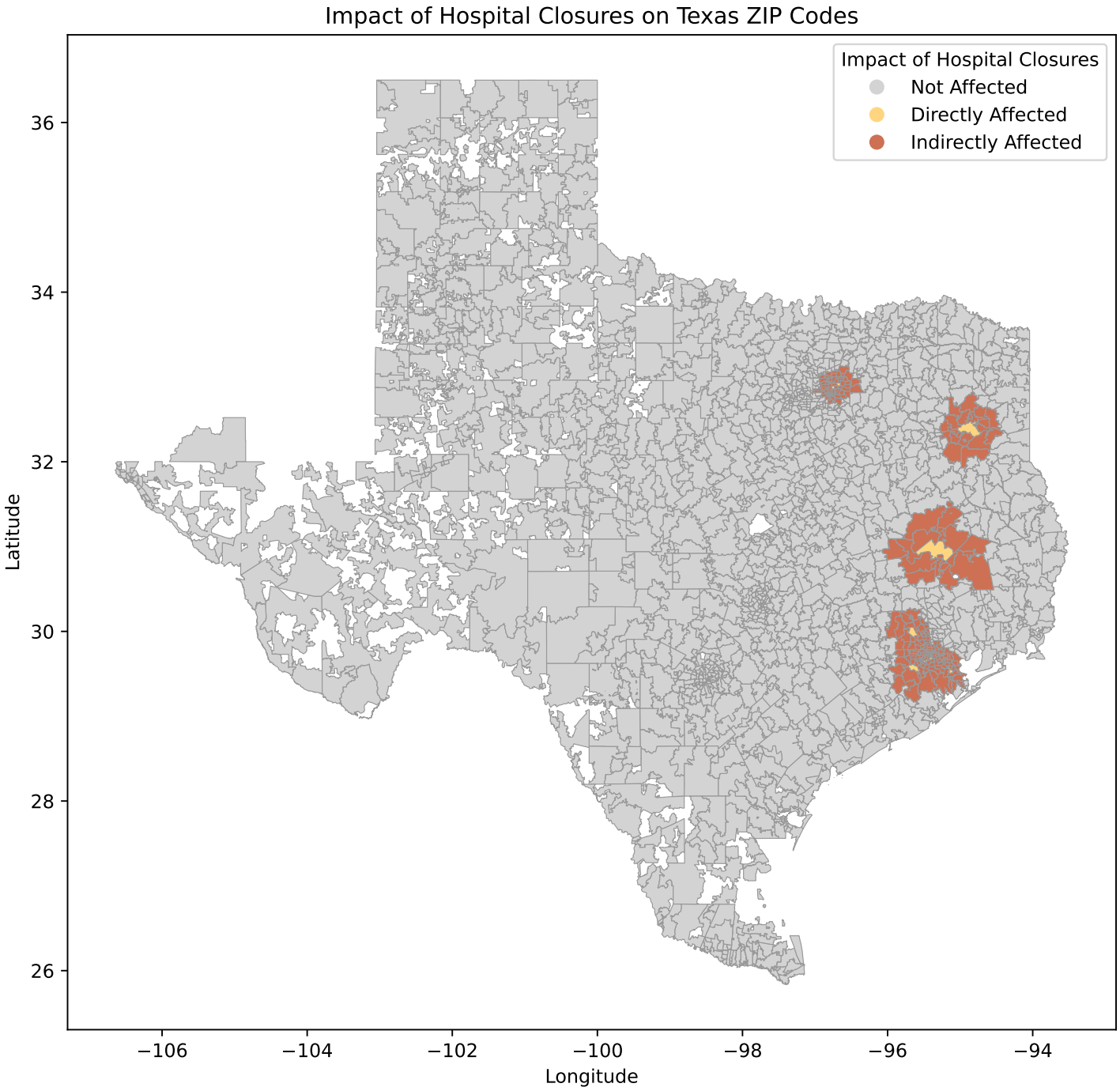
```
Projection file content:
 GEOGCS["GCS_North_American_1983",DATUM["D_North_American_1983",SPHEROID["GRS_1980",6378137,298.25722
Average latitude for conversion: 31.26293914563278
     ZCTA5_left ZCTA5_right  distance  distance_miles
9207      78624       78624  0.000000        0.000000
9208      78626       78664  0.167651        9.888185
9209      78628       78681  0.110844        6.537668
9210      78631       78028  0.337251       19.891345
9211      78632       78629  0.219909       12.970414
```

4.

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

# Load the Texas ZIP codes shapefile
zip_shapefile_path =
 ↪  r"/Users/sohyunlim/Desktop/python-ps4-shilm/gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.shp"
zip_gdf = gpd.read_file(zip_shapefile_path)

# Filter for Texas ZIP codes (starting with '75' or '77')
texas_zip_gdf = zip_gdf[
    (zip_gdf['ZCTA5'].astype(str).str[:3].astype(int).between(750, 799)) |
    (zip_gdf['ZCTA5'].astype(str).str.startswith("733"))
].copy()


# Ensure 'zip_code' column is in string format
confirmed_closures_df['zip_code'] = confirmed_closures_df['zip_code'].astype(str)
```

```
# Filter Texas ZIP codes with closures from 2016-2019
directly_affected_zip_codes = confirmed_closures_df[
    confirmed_closures_df['zip_code'].str.startswith(('75', '77')) &
    confirmed_closures_df['suspected_closure_year'].between(2016, 2019)
].copy()

# Correct ZIP code format by converting to integer and then back to string
directly_affected_zip_codes['zip_code'] =
 ↪  directly_affected_zip_codes['zip_code'].astype(float).astype(int).astype(str)

# Count closures by ZIP code
closures_by_zip =
 ↪  directly_affected_zip_codes.groupby('zip_code').size().reset_index(name='closure_count')

# Merge Texas ZIP codes with the directly affected ZIP codes data
affected_zip_gdf = texas_zip_gdf.merge(
    closures_by_zip, left_on='ZCTA5', right_on='zip_code', how='left'
).fillna(0)

# Set default impact as 'Not Affected'
affected_zip_gdf['affect'] = 'Not Affected'
affected_zip_gdf.loc[affected_zip_gdf['closure_count'] > 0, 'affect'] = 'Directly Affected'

# Define directly affected areas and create a buffer
directly_affected_gdf = affected_zip_gdf[affected_zip_gdf['affect'] == 'Directly
 ↪  Affected'].copy()

# Convert CRS to one that supports distance measurement (EPSG:3857)
texas_zip_gdf = texas_zip_gdf.to_crs(epsg=3857)
directly_affected_gdf = directly_affected_gdf.to_crs(epsg=3857)

# Create a 10-mile buffer around directly affected ZIP codes
directly_affected_gdf['buffer'] = directly_affected_gdf.buffer(16093.4)  # 10 miles in meters

# Set the buffer as the active geometry for spatial join
directly_affected_gdf = directly_affected_gdf.set_geometry('buffer')

# Perform the spatial join to find indirectly affected ZIP codes within 10 miles of directly
 ↪  affected ones
indirectly_affected_gdf = gpd.sjoin(texas_zip_gdf, directly_affected_gdf[['buffer']],
 ↪  how='inner', predicate='intersects')
indirectly_affected_zip_codes = indirectly_affected_gdf['ZCTA5'].unique()


# Reset geometry back to the original for plotting purposes
affected_zip_gdf = affected_zip_gdf.set_geometry('geometry')

# Add 'affect' categories based on the impact (direct and indirect)
affected_zip_gdf['affect'] = 'Not Affected'  # Default to 'Not Affected'
affected_zip_gdf.loc[affected_zip_gdf['closure_count'] > 0, 'affect'] = 'Directly Affected'
affected_zip_gdf.loc[
```

```
    (affected_zip_gdf['ZCTA5'].isin(indirectly_affected_zip_codes)) &
    (affected_zip_gdf['affect'] != 'Directly Affected'),
    'affect'
] = 'Indirectly Affected'

# Define a softer color dictionary for each category
color_dict = {
    'Not Affected': '#D3D3D3',       # Light Gray
    'Directly Affected': '#FFD580',  # Softened Yellow
    'Indirectly Affected': '#CD7054' # Softened Red
}

# Map colors to each category
affected_zip_gdf['color'] = affected_zip_gdf['affect'].map(color_dict)

# Plot the map with custom colors and thinner boundaries
fig, ax = plt.subplots(figsize=(10, 10))
affected_zip_gdf.plot(
    color=affected_zip_gdf['color'], linewidth=0.5, edgecolor='0.6', ax=ax  # thinner boundary
 ↪  lines
)

# Add a custom legend for each category with thinner legend markers
for category, color in color_dict.items():
    ax.plot([], [], color=color, label=category, markersize=7, linestyle="None", marker="o")
ax.legend(title="Impact of Hospital Closures")

plt.title("Impact of Hospital Closures on Texas ZIP Codes")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.show()
```



Reflecting on the exercise (10 pts)

1. The "first-pass" method for identifying hospital closures has limitations, including potential misclassifications due to missing data, inconsistencies across sources, and possible confusion with mergers or name changes. To improve accuracy, we can cross-reference multiple data sources to verify closures, use a multi-year timeframe to confirm permanent closures, track facilities that might have merged or changed names, and employ more detailed data when available. Additionally, flagging and reviewing anomalous data entries can help mitigate errors. These steps can reduce misclassification and enhance the reliability of hospital closure identification.

2. Using provider_id instead of ZIP codes could provide a more accurate assessment of hospital accessibility. The ZIP code approach only considers hospital availability within boundaries, which may overlook accessibility in nearby ZIP codes. In contrast, tracking by provider_id allows us to accurately monitor each hospital's location and status, thus capturing changes in accessibility that extend beyond ZIP code boundaries.