Python 2 Problem Set 4

Yufei Liu

PS4: Due Sat Nov 2 at 5:00PM Central. Worth 100 points.

Style Points (10 pts)

We use pep8 style.

Submission Steps (10 pts)

Partner 1: Yufei Liu (yufeil) Partner 2: Yufei Liu (yufeil)

This submission is our work alone and complies with the 30538 integrity policy. Add your initials to indicate your agreement: **Yufei Liu**

I have uploaded the names of anyone else other than my partner and I worked with on the problem set. (No other one)

Late coins used this pset: **0** Late coins left after submission: **3** of **Yufei Liu**

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

```
import pandas as pd
import altair as alt
import time
import os
from tabulate import tabulate
import geopandas as gpd
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from matplotlib.colors import ListedColormap
from shapely.geometry import Polygon

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

```
# local path set-up
base_path = "/Users/nancy/Documents/Document/U.S/class/fall
 ↪  2024/dap2/homework/problem-set-4-y/data"
```

1. **Selected Variables**:

- **FAC_NAME**: Facility Name
- **PRVDR_CTGRY_SBTYP_CD**: Identifies the subtype of the provider, within the primary category. Used in reporting to show the breakdown of provider categories, mainly for hospitals and SNFs.
- **PRVDR_CTGRY_CD**: Identifies the type of provider participating in the Medicare/Medicaid program.
- **PGM_TRMNTN_CD**: Termination Code (00=ACTIVE PROVIDER)
- **PRVDR_NUM**: provider. CMS Certification Number.
- **CITY_NAME**: City in which the provider is physically located.
- **SSA_CNTY_CD**: the county where the provider is located.
- **SSA_STATE_CD**: Social Security Administration geographic code indicating the state where the provider is located.
- **STATE_CD**: Two-character state abbreviation
- **TRMNTN_EXPRTN_DT**: Date the provider was terminated. For CLIA providers, date the laboratory's certificate was terminated or the expiration date of the current CLIA certificate
- **PSYCH_UNIT_TRMNTN_CD**: Indicates the reason that a psychiatric unit of a hospital is no longer exempt from Prospective Payment System (PPS).
- **ZIP_CD**: Five-digit ZIP code for a provider's physical address
- **GNRL_FAC_TYPE_CD**: Indicates the category-specific facility type code, for certain provider categories only.

2.

```
# import 2016 pos dataset
path_data_pos2016 = os.path.join(base_path, 'pos_1/pos2016.csv')
df_pos2016 = pd.read_csv(path_data_pos2016)
print(df_pos2016.shape)
# df_pos2016.head(3)
```

```
(141557, 13)
```

a. Short-term hospitals in dataset

Short-term facilities are hospitals with type code 01(PRVDR_CTGRY_CD = 01) and subtype code 01(PRVDR_CTGRY_SBTYP_CD = 01).

There are short-term 7245 hospitals according to the selected dataset.

```
# Short-term hospitals (type code 01 and subtype code 01)
df_short_pos2016 = df_pos2016[
    (df_pos2016["PRVDR_CTGRY_CD"] == 1.0) & (df_pos2016['PRVDR_CTGRY_SBTYP_CD'] == 1)
]
```

```
print(df_short_pos2016.shape)
print(df_short_pos2016['PRVDR_NUM'].nunique())
```

```
(7245, 13)
7245
```

```
df_short_pos2016_2 = df_pos2016[df_pos2016["GNRL_FAC_TYPE_CD"] == 1.0]
print(df_short_pos2016_2['PRVDR_NUM'].nunique())
```

```
7245
```

b. Cross-reference

However, after checking with the report from the Kaiser Family Foundation, we find only nearly 5,000 short-term, acute care hospitals in the United States.

It's different from what we got in the datasets, might because the 5000 shor-term hospitals are the one provide acute care while there are some counted hospital don't provide such service.

3.

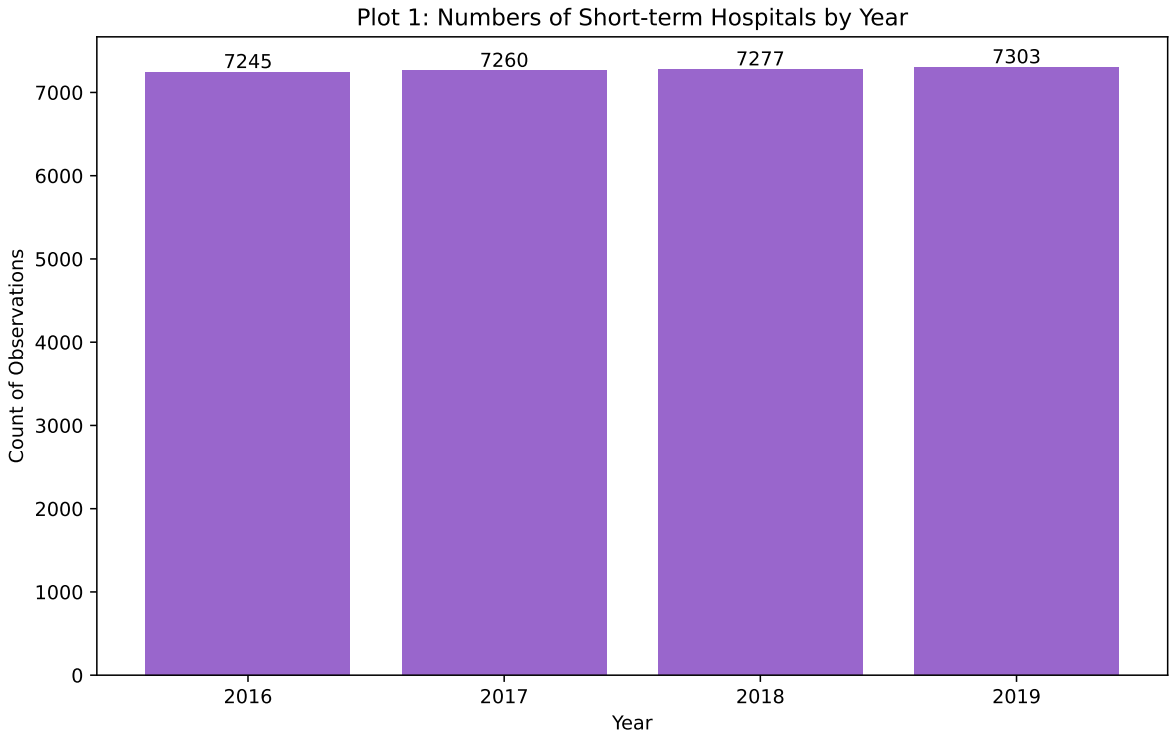
```
# Import datasets from 2017Q4 ~ 2019Q4
def load_yearly_data(base_path, start_year=2017, end_year=2019):
    # Dictionary to store DataFrames for each year
    data_frames = {}

    # Loop through each year in the range
    for year in range(start_year, end_year + 1):
        # Construct the file path for the specific year
        file_path = os.path.join(base_path, f'pos_1/pos{year}.csv')
        df = pd.read_csv(file_path)

        # Filter for short-term facilities
        df = df[(df["PRVDR_CTGRY_CD"] == 1) & (df["PRVDR_CTGRY_SBTYP_CD"] == 1.0)]

        # Add a 'Year' column
        df["YEAR"] = year

        # Store the filtered DataFrame with the year column in the dictionary
        data_frames[year] = df

    return data_frames


data_frames = load_yearly_data(base_path)

df_short_pos2017 = data_frames[2017]
df_short_pos2018 = data_frames[2018]
df_short_pos2019 = data_frames[2019]
```

```
# append datset
df_short_pos2016['YEAR'] = 2016
df_short = pd.concat([df_short_pos2016] + list(data_frames.values()),
 ↪  ignore_index=True)

df_short['ZIP_CD'] =
 ↪  df_short['ZIP_CD'].fillna(0).astype(int).astype(str).str.zfill(5)
df_short['PRVDR_NUM'] = df_short['PRVDR_NUM'].fillna(0).astype(int).astype(str)

# df_short.head(3)
df_short.shape
```

```
(29085, 14)
```

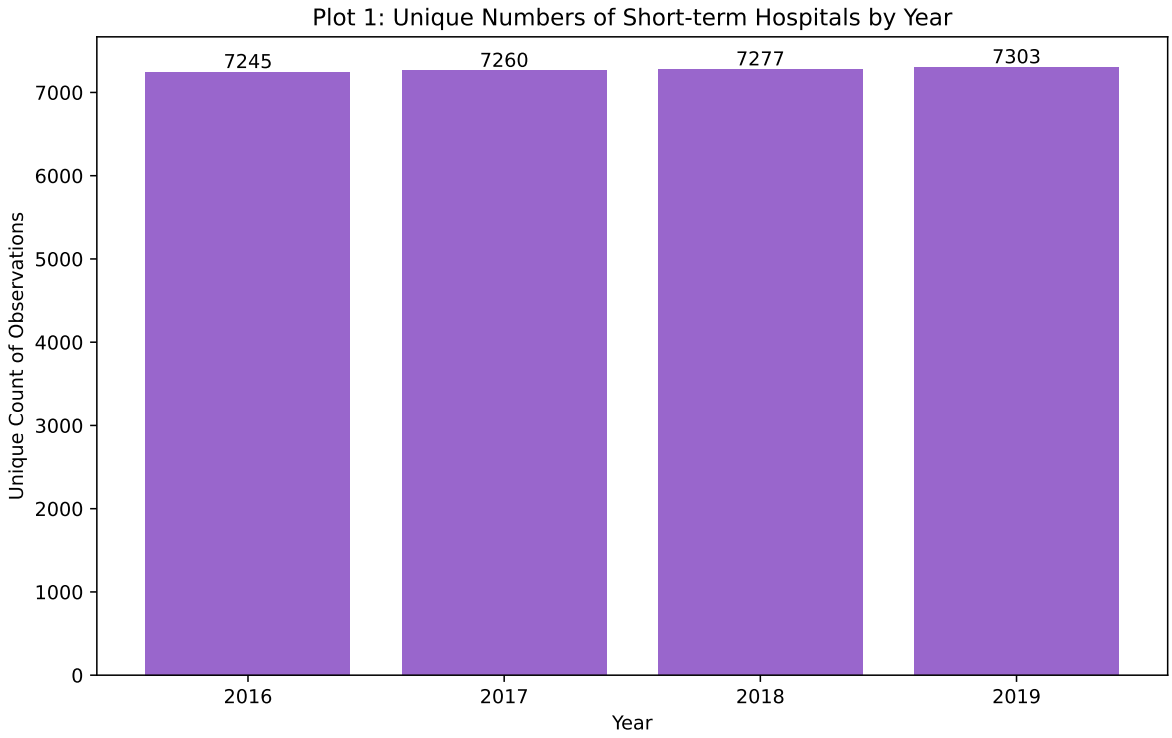
The Plot for observations from 2016 to 2019

```
obs_year = df_short.groupby('YEAR')['PRVDR_NUM'].count().reset_index()
obs_year.rename(columns={'PRVDR_NUM': 'NUM_HOS'}, inplace=True)

plt.figure(figsize=(10, 6))
plt.bar(obs_year['YEAR'], obs_year['NUM_HOS'], color='#9966CC')

for i, value in enumerate(obs_year['NUM_HOS']):
    plt.text(obs_year['YEAR'][i], value, str(value), ha='center', va='bottom')

plt.title('Plot 1: Numbers of Short-term Hospitals by Year')
plt.xlabel('Year')
plt.ylabel('Count of Observations')

plt.xticks(obs_year['YEAR'])

plt.show()
```



4. a.

```
obs_year_unique = df_short.groupby('YEAR')['PRVDR_NUM'].nunique().reset_index()
obs_year_unique.rename(columns={'PRVDR_NUM': 'NUM_HOS_UQ'}, inplace=True)

# Create bar plot using matplotlib
plt.figure(figsize=(10, 6))
plt.bar(obs_year_unique['YEAR'], obs_year_unique['NUM_HOS_UQ'], color='#9966CC')

# Add text labels on top of each bar
for i, value in enumerate(obs_year_unique['NUM_HOS_UQ']):
    plt.text(obs_year_unique['YEAR'][i], value, str(value), ha='center', va='bottom')

# Set title and labels
plt.title('Plot 1: Unique Numbers of Short-term Hospitals by Year')
plt.xlabel('Year')
plt.ylabel('Unique Count of Observations')

# Set x-axis to display integer years only
plt.xticks(obs_year_unique['YEAR'])

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



b.

The numbers of observations in each year are the same as the unique counts. Therefore, we can find each row of data in each year is one unique hospital. The dataset likely has a panel structure, where each PRVDR_NUM is tracked over multiple years but appears only once in each year.

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

1.

A hospital is suspected to have closed if its Termination Code('PGM_TRMNTN_CD') in the POS file lists them as an "Active Provider" ('PGM_TRMNTN_CD' = 0) in 2016 and then they are either not active or do not appear in the data at all in a subsequent year. We will also record suspected closed hospital's zip code ('ZIP_CD'), facility name ('FAC_NAME') and year of closure.

```
active_short2016 = df_short[
    (df_short['PGM_TRMNTN_CD'] == 0) & (df_short['YEAR'] == 2016)
]['PRVDR_NUM'].unique()

def find_closure_csm(active_short2016, df_short):
    closure_csm = []

    for csm in active_short2016:
        # Flag to stop searching once added to closure_csm
```

```
        added_to_closure = False

        # Retrieve 2016 ZIP code and facility name for fallback use
        info_2016 = df_short[
            (df_short['PRVDR_NUM'] == csm) & (df_short['YEAR'] == 2016)
        ]
        zip_2016 = info_2016['ZIP_CD'].iloc[0] if not info_2016.empty else None
        fac_name_2016 = info_2016['FAC_NAME'].iloc[0] if not info_2016.empty else
↪   None

        for year in [2017, 2018, 2019]:
            year_check = df_short[
                (df_short['YEAR'] == year) & (df_short['PRVDR_NUM'] == csm)
            ]

            # Check conditions:
            # 1. If the provider exists and PGM_TRMNTN_CD is not 0, add to
             ↪  closure_csm
            # 2. If the provider does not exist for this year, also add to
             ↪  closure_csm
            if not year_check.empty and (year_check['PGM_TRMNTN_CD'].iloc[0] != 0):
                closure_csm.append({
                    'PRVDR_NUM': csm,
                    'YEAR_CLO': year,
                    'ZIP_CD': year_check['ZIP_CD'].values[0],
                    'FAC_NAME': year_check['FAC_NAME'].values[0]
                })
                added_to_closure = False
                break

            elif year_check.empty:
                closure_csm.append({
                    'PRVDR_NUM': csm,
                    'YEAR_CLO': year,
                    'ZIP_CD': zip_2016,
                    'FAC_NAME': fac_name_2016
                })
                added_to_closure = True
                break

            if added_to_closure:
                continue

    return closure_csm

closure_csm = find_closure_csm(active_short2016, df_short)
print(f"There are {len(closure_csm)} hospitals suspected to close.")
```

```
There are 174 hospitals suspected to close.
```

2.

```
# Sort the list by 'FAC_NAME'
closure_csm.sort(key=lambda x: x['FAC_NAME'])

# Print the first 10 items by 'FAC_NAME' and 'YEAR_CLO'
for entry in closure_csm[:10]:
    print(f"The facility named '{entry['FAC_NAME']}' was closed in
     ↪  {entry['YEAR_CLO']}.")
```

```
The facility named 'ABRAZO MARYVALE CAMPUS' was closed in 2017.
The facility named 'ADVENTIST MEDICAL CENTER - CENTRAL VALLEY' was closed in 2017.
The facility named 'AFFINITY MEDICAL CENTER' was closed in 2018.
The facility named 'ALBANY MEDICAL CENTER / SOUTH CLINICAL CAMPUS' was closed in
2017.
The facility named 'ALLEGIANCE SPECIALTY HOSPITAL OF KILGORE' was closed in 2017.
The facility named 'ALLIANCE LAIRD HOSPITAL' was closed in 2019.
The facility named 'ALLIANCEHEALTH DEACONESS' was closed in 2019.
The facility named 'ARKANSAS VALLEY REGIONAL MEDICAL CENTER' was closed in 2017.
The facility named 'ASCENSION NE WISCONSIN MERCY CAMPUS' was closed in 2018.
The facility named 'ATRIUM HEALTH KINGS MOUNTAIN' was closed in 2019.
```

3. a.

To find true closure, we need to firstly remove firms potentially being merger/acquisition.

We remove any suspected hospital closures that are in zip codes (ZIP_CD) where the number of active hospitals does not decrease in the year after the suspected closure. To count number of hospitals, we group by CMS number('PGM_TRMNTN_CD').

```
# Convert closure data into a DataFrame
closure_df = pd.DataFrame(closure_csm)
# Filter out active short-term hospitals with zip codes in closure list
# df_active = df_short[(df_short['PGM_TRMNTN_CD'] == 0) &
 ↪  df_short['ZIP_CD'].isin(closure_df['ZIP_CD'])]
df_active = df_short[df_short['PGM_TRMNTN_CD'] == 0]
# Get the counts of active hospitals in each zip code and year
zip_year1 = df_active.groupby(['ZIP_CD', 'YEAR']).size().reset_index(name='active')
zip_year1 = zip_year1[['ZIP_CD', 'YEAR', 'active']]
zip_year1['active_next_y'] = zip_year1.groupby('ZIP_CD')['active'].shift(-1)

# Prepare closure_counts with unique counts of PRVDR_NUM per ZIP_CD and YEAR,
 ↪  retaining PRVDR_NUM column
closure_counts = closure_df[['ZIP_CD', 'YEAR_CLO', 'PRVDR_NUM']]
closure_counts = (
    closure_counts.groupby(['ZIP_CD', 'YEAR_CLO'])
    .agg(num_closure=('PRVDR_NUM', 'nunique'), PRVDR_NUM_list=('PRVDR_NUM',
 ↪  'unique'))
```

```
    .reset_index()
)

# Rename 'YEAR_CLO' to 'YEAR' for consistency with zip_year
closure_counts = closure_counts.rename(columns={'YEAR_CLO': 'YEAR'})
# Merge DataFrames and check row count
zip_year = closure_counts.merge(zip_year1, on=['ZIP_CD', 'YEAR'], how='left')
# zip_year = zip_year.merge(closure_counts, on=['ZIP_CD', 'YEAR'], how='left')
# Fill NaN values in the 'active' column and check row count again
zip_year['active'] = zip_year['active'].fillna(0)

zip_year['active_next_y'] = zip_year['active_next_y'].fillna(0)

zip_false_clo = zip_year[
    (zip_year['num_closure'] != 0) &
    (zip_year['active'] <= zip_year['active_next_y'])
]

print(zip_false_clo.shape)
unique_prvdr_num_count = set([prvdr_num for sublist in
 ↪  zip_false_clo['PRVDR_NUM_list'].dropna() for prvdr_num in sublist]).__len__()
print(f"Unique PRVDR_NUM count in zip_false_clo: {unique_prvdr_num_count}")
```

```
(148, 6)
Unique PRVDR_NUM count in zip_false_clo: 148
```

```
# Create a set of (YEAR, PRVDR_NUM) pairs from zip_false_clo to remove
remove_pairs = set(
    (year, prvdr_num)
    for year, prvdr_num_list in zip(zip_false_clo['YEAR'],
     ↪  zip_false_clo['PRVDR_NUM_list'])
    if prvdr_num_list is not None
    for prvdr_num in prvdr_num_list
)

# Filter closure_csm to exclude entries with matching (YEAR_CLO, PRVDR_NUM) in
 ↪  remove_pairs
closure_csm_true = [
    entry for entry in closure_csm
    if (entry['YEAR_CLO'], entry['PRVDR_NUM']) not in remove_pairs
]

# closure_csm_true now contains only the entries that don't match the specified
 ↪  (YEAR, PRVDR_NUM) pairs
print(f"There are {len(closure_csm_true)} hospitals left.")
```

```
There are 26 hospitals left.
```

There are 148 hospital fit this definition of potentially being a merger/acquisition. After correcting them, 26 hospitals have left as of true closure.

b.

```
sorted_closure_csm_true = sorted(closure_csm_true, key=lambda x: x['FAC_NAME'])[:10]
sorted_closure_df = pd.DataFrame(sorted_closure_csm_true)
sorted_closure_df.head(10)
```

	PRVDR_NUM	YEAR_CLO	ZIP_CD	FAC_NAME
0	370032	2019	73112	ALLIANCEHEALTH DEACONESS
1	390302	2019	19047	BARIX CLINICS OF PENNSYLVANIA
2	670087	2019	78613	BAYLOR SCOTT & WHITE EMERGENCY MEDICAL CENTER ...
3	260227	2019	63901	BLACK RIVER COMMUNITY MEDICAL CENTER
4	110187	2019	30533	CHESTATEE REGIONAL HOSPITAL
5	190297	2019	71446	DOCTORS HOSPITAL AT DEER CREEK L L C
6	450845	2019	79902	EL PASO SPECIALTY HOSPITAL
7	150182	2019	46032	FRANCISCAN HEALTH CARMEL
8	110004	2019	30742	HUTCHESON MEDICAL CENTER
9	130067	2019	83221	IDAHO DOCTORS HOSPITAL

c.

The list of corrected hospital closures by name and the first 10 rows:

```
# Sort the list by 'FAC_NAME'
closure_csm_true.sort(key=lambda x: x['FAC_NAME'])

# Print the first 10 items by 'FAC_NAME' and 'YEAR_CLO'
for entry in closure_csm_true[:10]:
    print(f"The facility named '{entry['FAC_NAME']}' was truely closed in
     ↪ {entry['YEAR_CLO']}.")
```

```
The facility named 'ALLIANCEHEALTH DEACONESS' was truely closed in 2019.
The facility named 'BARIX CLINICS OF PENNSYLVANIA' was truely closed in 2019.
The facility named 'BAYLOR SCOTT & WHITE EMERGENCY MEDICAL CENTER AT C' was truely
closed in 2019.
The facility named 'BLACK RIVER COMMUNITY MEDICAL CENTER' was truely closed in 2019.
The facility named 'CHESTATEE REGIONAL HOSPITAL' was truely closed in 2019.
The facility named 'DOCTORS HOSPITAL AT DEER CREEK L L C' was truely closed in 2019.
The facility named 'EL PASO SPECIALTY HOSPITAL' was truely closed in 2019.
The facility named 'FRANCISCAN HEALTH CARMEL' was truely closed in 2019.
The facility named 'HUTCHESON MEDICAL CENTER' was truely closed in 2019.
The facility named 'IDAHO DOCTORS HOSPITAL' was truely closed in 2019.
```

Download Census zip code shapefile (10 pt)

```
# import census data
census_path = os.path.join(base_path,
 ↪  'gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.shp')
# print(census_path)
census_data = gpd.read_file(census_path)
# census_data.head(3)
```

1. a.

There are five types of files: '.dbf', 'prj', '.shp', 'shx' and 'xml'.

The five file types in the shapefile set you have are:

.shp (Shapefile) - This is the core file that stores the actual geometric shapes of the spatial features, such as points, lines, or polygons, which represent geographic entities (like boundaries of ZIP codes).

.shx (Shape Index File) - This file is an index of the geometric data in the `.shp` file. It allows for quick access to the spatial data by providing offsets to locate shapes within the `.shp` file.

.dbf (Database File) - This file stores attribute data for each spatial feature in a tabular format. It includes columns of information related to each geographic entity, such as names, ZIP codes, or other associated data fields. This data is non-spatial but is linked to the spatial shapes in the `.shp` file.

.prj (Projection File) - This file contains projection information for the spatial data, describing how the shapes in the `.shp` file align with the Earth's surface. It specifies the coordinate system and map projection, which are essential for accurate spatial analysis.

.xml (Metadata File) - This file holds metadata about the dataset, including information on the data's source, creation date, and a description of its contents. Metadata helps users understand the context, quality, and intended use of the dataset.

`b.`

The sizes of each dataset: **.dbf** : 6.4 MB (less bigger) **.prj** : 165 bytes (smallest) **.shp** : 837.5 MB (biggest) **.shx** : 265 KB (smaller) **.xml** : 16KB (smaller)

2.

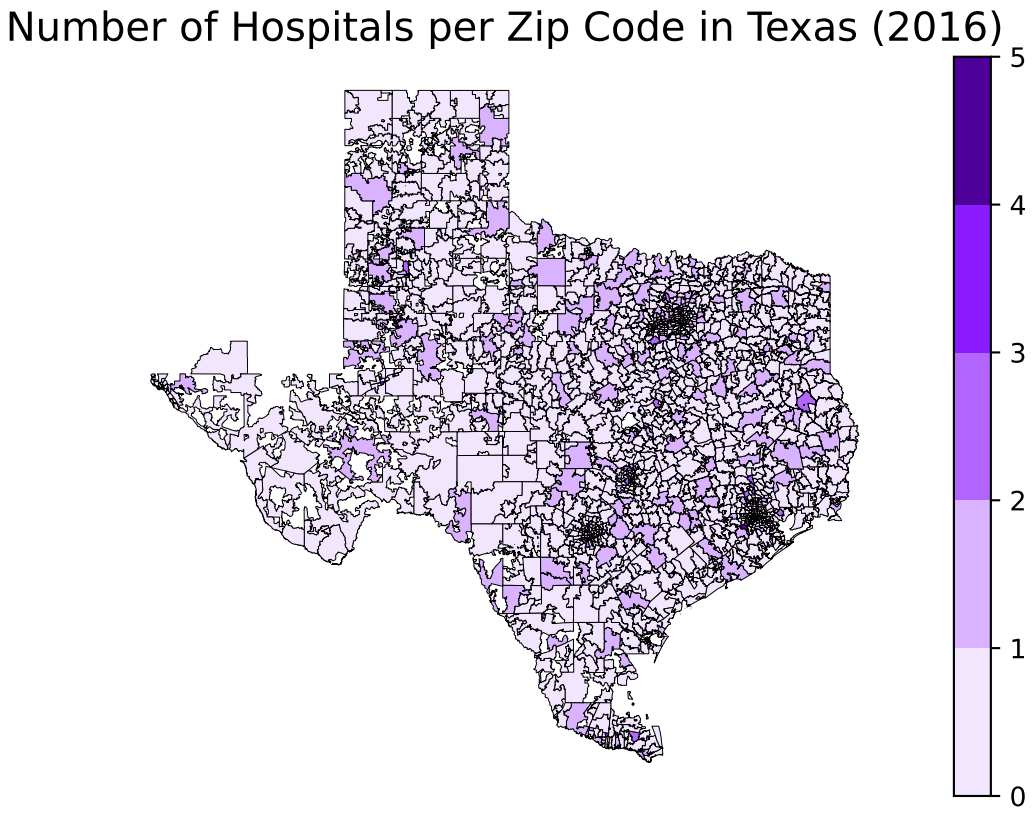
To focus on Texas, we will restrict to Texas zipcodes('ZCTA5'). According to Wikipedia tables, the first three digits of Texas (TX) zip codes range from 733 and 750 to 799, as well as 885.(733, 750~799, and 885)

```
# extract Texas cencus data
texas_zip_prefixes = [str(i) for i in range(750,800)] + ['733', '885']

census_tx = census_data[census_data['ZCTA5'].str[:3].isin(texas_zip_prefixes)]

# census_tx.head(3)
```

```
# Filter the dataframe for active hospitals in 2016
active_short2016_df = df_short[(df_short['PGM_TRMNTN_CD'] == 0) & (df_short['YEAR']
 ↪  == 2016)]

texas_hospitals_2016 = active_short2016_df[
    active_short2016_df['ZIP_CD'].astype(str).str[:3].isin(texas_zip_prefixes)
]

# Count the number of hospitals per zip code in Texas for 2016
hospital_counts =
 ↪  texas_hospitals_2016.groupby('ZIP_CD').size().reset_index(name='num_hospitals')

# Merge the hospital counts with the Texas zip code shapefile data in census_tx
texas_zip_hospitals = census_tx.merge(hospital_counts, left_on='ZCTA5',
 ↪  right_on='ZIP_CD', how='left')
texas_zip_hospitals['num_hospitals'] = texas_zip_hospitals['num_hospitals'].fillna(0)
```

```
# Define a discrete purple color map with specific colors for each integer value
discrete_colors = ListedColormap(['#f2e6ff', '#d9b3ff', '#b266ff', '#8c1aff',
 ↪  '#4d0099'])
# '#f2e6ff' is very light, moving to darker purple shades

fig, ax = plt.subplots(1, 1, figsize=(8, 5))

# Use the discrete color map
choropleth = texas_zip_hospitals.plot(
    column='num_hospitals',
    cmap=discrete_colors,   # Use the discrete color map
    linewidth=0.2,
    ax=ax,
    edgecolor='black',
    legend=True,
    vmin=0,  # Set minimum value for color scaling
    vmax=5   # Set the maximum value to align with the number of color steps
)

# Customize the legend to have integer values only
legend = choropleth.get_legend()
if legend:
    legend.set_bbox_to_anchor((1, 0.5))
    legend.set_title("Number of Hospitals")
    legend.ax.yaxis.set_major_locator(ticker.MultipleLocator(1))  # Set integer ticks
 ↪  with interval 1
    legend.ax.yaxis.set_major_formatter(ticker.FormatStrFormatter('%d'))  # Remove
 ↪  decimal places

ax.set_title('Number of Hospitals per Zip Code in Texas (2016)', fontsize=15)
```

```
plt.axis('off')
plt.show()
```



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

1.

```
# Create a GeoDataFrame with centroids
zips_all_centroids = census_data.copy()
zips_all_centroids['geometry'] = census_data['geometry'].centroid

# Display the dimensions of the GeoDataFrame
dimensions = zips_all_centroids.shape
dimensions

# Display columns and first few rows to understand what each column represents
zips_all_centroids.head()
```

	GEO_ID	ZCTA5	NAME	LSAD	CENSUSAREA	geometry
0	8600000US01040	01040	01040	ZCTA5	21.281	POINT (-72.64107 42.21257)
1	8600000US01050	01050	01050	ZCTA5	38.329	POINT (-72.86985 42.28786)
2	8600000US01053	01053	01053	ZCTA5	5.131	POINT (-72.71162 42.35349)
3	8600000US01056	01056	01056	ZCTA5	27.205	POINT (-72.45805 42.19215)
4	8600000US01057	01057	01057	ZCTA5	44.907	POINT (-72.3243 42.09165)

2.

```
# Subset of ZIP code centroids for Texas
zips_texas_centroids =
 ↪  zips_all_centroids[zips_all_centroids['ZCTA5'].str[:3].isin(texas_zip_prefixes)]

# Define ZIP code prefixes for Texas and bordering states
bordering_state_prefixes = texas_zip_prefixes + [str(i) for i in range(700, 740)] +
 ↪  [str(i) for i in range(870, 880)]
zips_texas_borderstates_centroids =
 ↪  zips_all_centroids[zips_all_centroids['ZCTA5'].str[:3].isin(bordering_state_prefixes)]

# Count unique ZIP codes in each subset
num_unique_texas_zips = zips_texas_centroids['ZCTA5'].nunique()
num_unique_borderstate_zips = zips_texas_borderstates_centroids['ZCTA5'].nunique()

# Display results
print("Unique ZIP codes in Texas:", num_unique_texas_zips)
print("Unique ZIP codes in Texas and bordering states:", num_unique_borderstate_zips)
```

```
Unique ZIP codes in Texas: 1935
Unique ZIP codes in Texas and bordering states: 3585
```

```
# Combine Texas ZIP code polygons into one polygon
texas_polygon = zips_texas_centroids.unary_union  # Creates a single combined polygon
 ↪  for all Texas ZIP codes

def polygons_intersect(poly1: Polygon, poly2: Polygon) -> bool:
    """Returns True if two polygons intersect, False otherwise."""
    return poly1.intersects(poly2)

bordering_polygons = zips_texas_borderstates_centroids.unary_union  # Combines all
 ↪  ZIP codes in bordering states

# Check if the bordering state polygons intersect with the Texas polygon
is_bordering = polygons_intersect(texas_polygon, bordering_polygons)
print("Do Texas ZIP codes intersect with bordering states' ZIP codes?", is_bordering)
```

```
Do Texas ZIP codes intersect with bordering states' ZIP codes? True
```

3.

In this part, we would find zip code with at least 1 active short-term hospital.

```
# Group by ZIP_CD and count unique provider numbers
zip_with_hospitals_2016 =
 ↪  active_short2016_df.groupby('ZIP_CD')['PRVDR_NUM'].nunique()

# Filter ZIP codes with at least one unique hospital provider
zip_with_hospitals_2016 = zip_with_hospitals_2016[zip_with_hospitals_2016 >= 1]

# Display the resulting ZIP codes and the number of unique hospitals in each
zip_with_hospitals_2016 = zip_with_hospitals_2016.reset_index()
zip_with_hospitals_2016.columns = ['ZIP_CD', 'Unique_Hospitals']
zip_with_hospitals_2016.shape
```

```
(3067, 2)
```

```
zips_texas_borderstates_centroids =
 ↪  zips_texas_borderstates_centroids.rename(columns={'ZCTA5': 'ZIP_CD'})

# Merge the centroids GeoDataFrame with the DataFrame of ZIP codes with hospitals
zips_withhospital_centroids = zips_texas_borderstates_centroids.merge(
    zip_with_hospitals_2016,
    on='ZIP_CD',
    how='inner'
)
```

Merge Type: An inner merge was chosen because we want only the ZIP codes that are in both datasets (i.e., ZIP codes in Texas and bordering states that had at least one hospital in 2016).

Merge Key: The merge is done on the ZIP_CD column, which represents the ZIP codes.

4. a.

```
sample_zips = zips_texas_centroids.sample(10)
start_time = time.time()
distances = []

for idx, row in sample_zips.iterrows():
    # Find the nearest point in zips_withhospital_centroids for each zip in
     ↪  sample_zips
    nearest_geom = zips_withhospital_centroids.distance(row['geometry']).idxmin()
    nearest_distance =
 ↪  row['geometry'].distance(zips_withhospital_centroids.loc[nearest_geom,
 ↪  'geometry'])
    distances.append(nearest_distance)

end_time = time.time()
sample_duration = end_time - start_time
```

```
print("Time taken for sample of 10 ZIP codes:", sample_duration, "seconds")

total_rows = len(zips_texas_centroids)
estimated_time = (sample_duration / 10) * total_rows
print("Estimated time for full calculation:", estimated_time, "seconds")
```

```
Time taken for sample of 10 ZIP codes: 0.15216517448425293 seconds
Estimated time for full calculation: 29.443961262702942 seconds
```

b.

```
start_time_full = time.time()
full_distances = []

for idx, row in zips_texas_centroids.iterrows():
    nearest_geom = zips_withhospital_centroids.distance(row['geometry']).idxmin()
    nearest_distance =
 ↪  row['geometry'].distance(zips_withhospital_centroids.loc[nearest_geom,
 ↪  'geometry'])
    full_distances.append(nearest_distance)

end_time_full = time.time()
actual_time = end_time_full - start_time_full
print("Actual time for full calculation:", actual_time, "seconds")
print("Estimated vs. Actual:", estimated_time, "vs.", actual_time)
```

```
Actual time for full calculation: 24.098822116851807 seconds
Estimated vs. Actual: 29.443961262702942 vs. 24.098822116851807
```

c.

```
distances_in_miles = [dist / 1609.34 for dist in full_distances]
```

5. a.

```
distances = []
for idx, row in zips_texas_centroids.iterrows():
    # Find the nearest hospital ZIP code's geometry
    nearest_geom = zips_withhospital_centroids.distance(row['geometry']).idxmin()
    # Calculate the distance
    nearest_distance =
 ↪  row['geometry'].distance(zips_withhospital_centroids.loc[nearest_geom,
 ↪  'geometry'])
    distances.append(nearest_distance)

# Add the distances as a new column in the GeoDataFrame
zips_texas_centroids['distance_to_hospital'] = distances
```

```
# Convert the distances to miles
zips_texas_centroids['distance_to_hospital_miles'] =
 ↪  zips_texas_centroids['distance_to_hospital'] / 1609.34
```

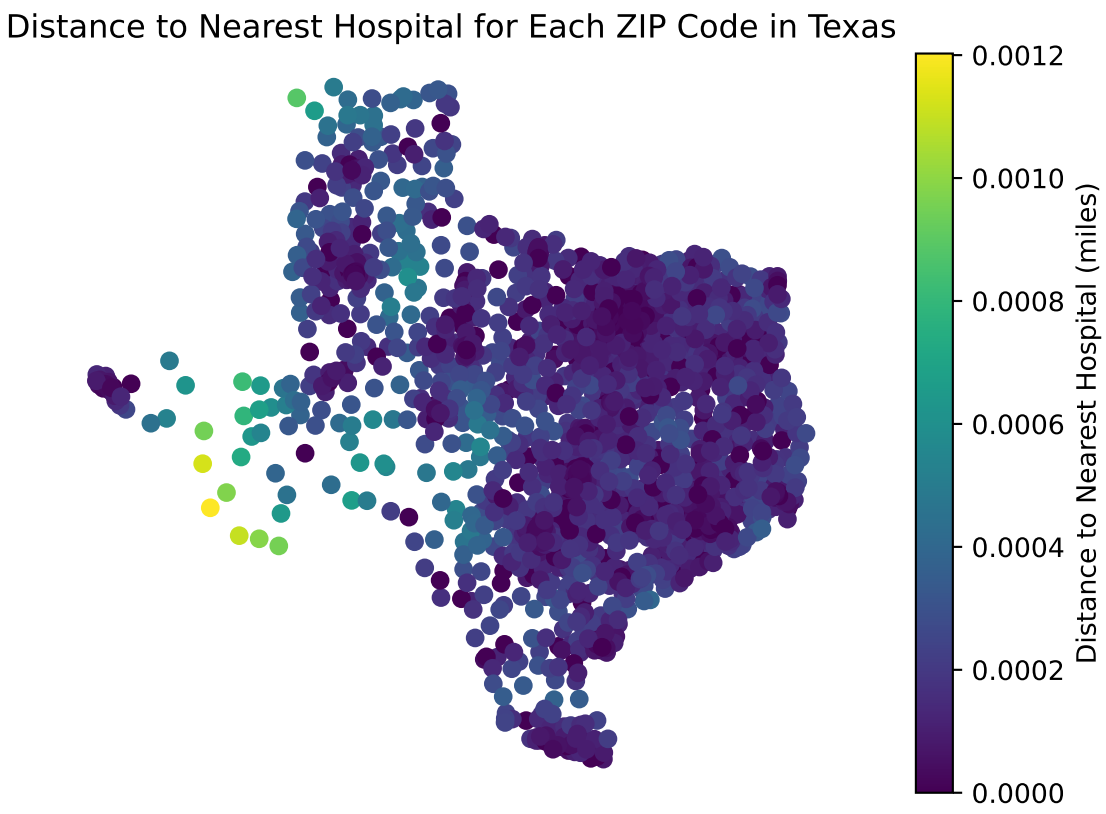
b.

```
average_distance_miles = zips_texas_centroids['distance_to_hospital_miles'].mean()
print("Average distance to the nearest hospital in miles:", average_distance_miles)
```

```
Average distance to the nearest hospital in miles: 0.00013192145269356578
```

c.

```
# Plot the Texas ZIP codes with color representing distance to the nearest hospital
fig, ax = plt.subplots(1, 1, figsize=(10, 5))
zips_texas_centroids.plot(column='distance_to_hospital_miles', cmap='viridis',
 ↪  legend=True,
                          legend_kwds={'label': "Distance to Nearest Hospital
 ↪  (miles)"},
                          ax=ax)
ax.set_title("Distance to Nearest Hospital for Each ZIP Code in Texas")
plt.axis('off')
plt.show()
```



Effects of closures on access in Texas (15 pts)

1.

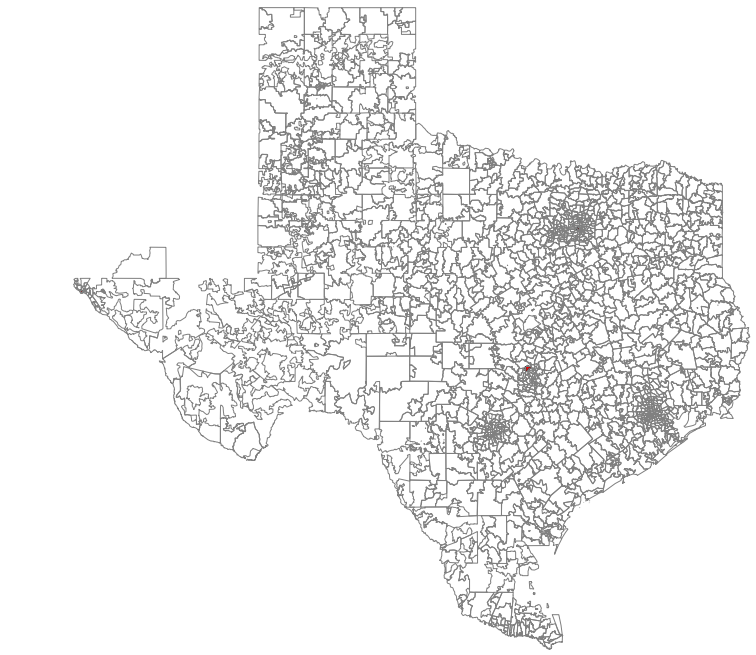
```
closure_true_tx = [entry for entry in closure_csm_true if str(entry['ZIP_CD'])[:3] in
 ↪  texas_zip_prefixes]
closure_true_tx_df = pd.DataFrame(closure_true_tx)

print(closure_true_tx_df.shape)

zip_code_closures =
 ↪  closure_true_tx_df.groupby('ZIP_CD').size().reset_index(name='number_of_closures')

print(tabulate(zip_code_closures, headers="keys", tablefmt="grid"))
```

```
(4, 4)
+----+----------+----------------------+
|    |   ZIP_CD |   number_of_closures |
+====+==========+======================+
|  0 |    75235 |                    1 |
```

```
+----+----------+----------------------+
|  1 |    75390 |                    1 |
+----+----------+----------------------+
|  2 |    78613 |                    1 |
+----+----------+----------------------+
|  3 |    79902 |                    1 |
+----+----------+----------------------+
```

2.

```
# Convert ZIP_CD in closure_true_tx_df to strings, removing any decimals
closure_true_tx_df['ZIP_CD'] = closure_true_tx_df['ZIP_CD'].astype(int).astype(str)

# Convert ZCTA5 to string in census_tx for consistent comparison
census_tx['ZCTA5'] = census_tx['ZCTA5'].astype(str)

# Create the 'affected' column: 1 if in closure_true_tx_df['ZIP_CD'], otherwise 0
census_tx['affected'] =
 ↪  census_tx['ZCTA5'].isin(closure_true_tx_df['ZIP_CD']).astype(int)

# Step 3: Plot the choropleth of Texas ZIP codes with color only for affected areas
fig, ax = plt.subplots(1, 1, figsize=(5, 5))

# Plot affected areas in purple with borders
census_tx[census_tx['affected'] == 1].plot(color='red', linewidth=0.3,
 ↪  edgecolor='black', ax=ax)

# Plot unaffected areas with no fill color, only borders
census_tx[census_tx['affected'] == 0].plot(color='none', edgecolor='grey',
 ↪  linewidth=0.2, ax=ax)

# Customize plot
ax.set_title("Texas ZIP Codes Affected by Closures (2016-2019)")
ax.set_axis_off()

plt.show()
```




```
census_tx[census_tx['affected'] == 1]
```

	GEO_ID	ZCTA5	NAME	LSAD	CENSUSAREA	geometry
9681	8600000US79902	79902	79902	ZCTA5	6.558	MULTIPOLYGON (((-106.5034 31.80737, -1
10867	8600000US78613	78613	78613	ZCTA5	28.088	MULTIPOLYGON (((-97.80206 30.47071, -9
26478	8600000US75235	75235	75235	ZCTA5	6.720	POLYGON ((-96.84953 32.81137, -96.84928
28307	8600000US75390	75390	75390	ZCTA5	0.005	POLYGON ((-96.8406 32.81349, -96.8406 32.

```
# Count the number of directly affected ZIP codes in Texas
num_directly_affected_zips = census_tx[census_tx['affected'] == 1]['ZCTA5'].nunique()
print("Number of directly affected ZIP codes in Texas:", num_directly_affected_zips)
```

```
Number of directly affected ZIP codes in Texas: 4
```

3.

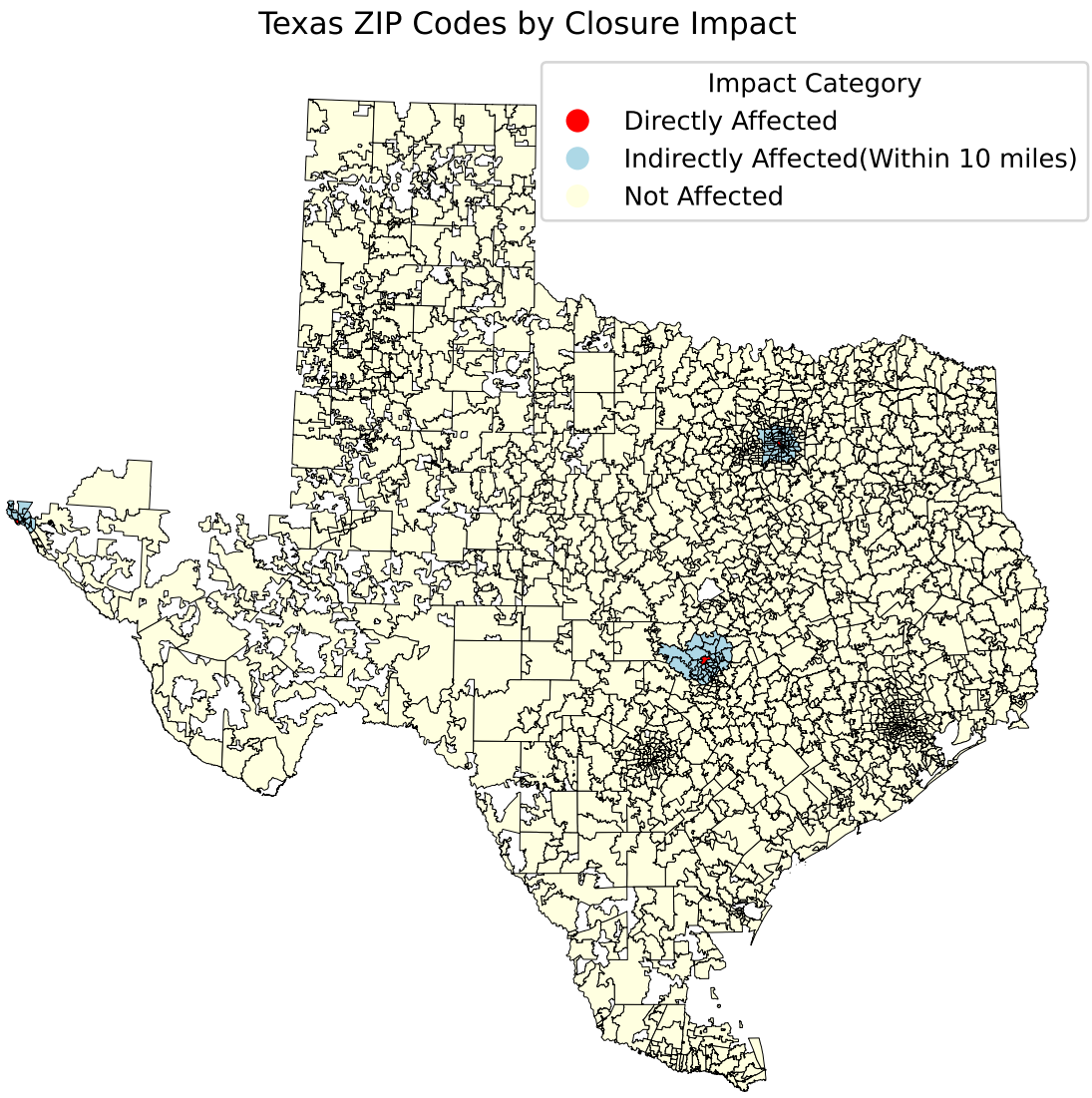
```
print(census_tx.crs)
```

```
EPSG:4269
```

```
census_tx = census_tx.to_crs(epsg=32614)
directly_affected_zips = census_tx[census_tx['affected'] == 1].copy()
directly_affected_zips['geometry'] =
 ↪  directly_affected_zips['geometry'].buffer(16093.4)
indirectly_affected_zips = gpd.sjoin(census_tx, directly_affected_zips[['geometry']],
 ↪  how='inner', predicate='intersects')
indirectly_affected_unique =
 ↪  indirectly_affected_zips[indirectly_affected_zips['affected'] ==
 ↪  0]['ZCTA5'].nunique()
print("Number of indirectly affected ZIP codes in Texas:",
 ↪  indirectly_affected_unique)
```

```
Number of indirectly affected ZIP codes in Texas: 124
```

```
# Mark directly affected ZIP codes
census_tx['category'] = 'Not Affected'

# Create 10-mile buffer and mark indirectly affected ZIP codes
census_tx.loc[indirectly_affected_zips.index, 'category'] = 'Indirectly Affected'

census_tx.loc[census_tx['affected'] == 1, 'category'] = 'Directly Affected'

# Define custom colors for the categories
category_colors = {'Directly Affected': 'red', 'Indirectly Affected': 'lightblue',
 ↪  'Not Affected': 'lightyellow'}
census_tx['color'] = census_tx['category'].map(category_colors)

# Plot the choropleth
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
census_tx.plot(color=census_tx['color'], linewidth=0.2, edgecolor='black', ax=ax)

# Add legend
handles = [
    plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='red', markersize=10,
 ↪  label='Directly Affected'),
    plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='lightblue',
 ↪  markersize=10, label='Indirectly Affected(Within 10 miles)'),
    plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='lightyellow',
 ↪  markersize=10, label='Not Affected')
]
ax.legend(handles=handles, title="Impact Category")

ax.set_title("Texas ZIP Codes by Closure Impact")
ax.set_axis_off()

plt.show()
```



4.

Reflecting on the exercise (10 pts)

a.

Some potential issues:

The "first-pass" method for identifying incorrectly classified hospital closures is limited in several ways. Here are some potential issues:

- Limitations with Recent Data: Since the first-pass method relies on comparing hospital counts year-over-year, it can only flag closures up until 2018 due to the lack of data beyond 2019. This means closures in 2019 might not be accurately verified, as there's no data for subsequent years to confirm the closure's impact.
- Merger and Acquisition Impact: The first-pass method doesn't account for closures due to mergers or acquisitions, which may lead to "false closures." When a hospital appears to close but has merged with or been acquired by another institution, the closure is not necessarily a true one.
- Influence of New Hospitals: New hospital openings are not considered in the first-pass method. If new hospitals open in the area, they can artificially inflate the count of active hospitals in the following year, giving the appearance of a false closure when, in fact, the facility has shut down.

Better Methods:

- A better approach might include obtaining more recent data or using other indicators that don't depend solely on a year-over-year comparison.
- A more robust approach would involve cross-referencing with external databases or public records on mergers and acquisitions to confirm if a hospital has genuinely ceased operations or simply changed ownership.
- Including variables such as new hospital openings or expansions in nearby regions could improve the accuracy of the closure verification.
- The variable TRMNTN_EXPRTN_DT (Date the provider was terminated) could provide an official source to verify closures. Incorporating this variable would allow us to directly confirm whether a facility has been terminated according to official records, significantly improving the accuracy of closure identification.

b.

How well does this reflect changes in zip-code-level access to hospitals?:

- This method assumes that residents in the area will seek medical care at the nearest hospital and that hospitals are generally similar. If these assumptions hold, this approach can reflect the impact accurately.
- However, different hospitals may offer different services, such as whether they provide emergency care, and each hospital may have historically served a different number of patients. The buffers could be different for each hospital.
- The 10-mile assumption requires stronger justification. For instance, if nearby residents have high vehicle ownership rates, the impact range of a hospital closure might extend beyond 10 miles.

Potential improvement: - It would be beneficial to gather and consider such data to better measure the impact. This could involve examining variables related to hospital functions, bed count, number of doctors and nurses, and service ratings for a comprehensive assessment.

- Different hospitals should consider use different mile-level buffers according to different historical data, like facilities level and vehicle ownership rates of residents.