Your Title

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 :Sijie Wu, sijiewu
    - Partner 2 (Abu Bakar Siddique, abubakars):
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: **___** **___**
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: **___** Late coins left after submission: **___**
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` for provider subtype code; `PRVDR_CTGRY_CD` for provider type code; `PRVDR_NUM` for CMS Certification Number; `PGM_TRMNTN_CD` for Termination Code; `ZIP_CD` for zip code; `TRMNTN_EXPRTN_DT` for Termination Date; `FAC_NAME` for facility name.
2.

```
import pandas as pd
```

```
filepath1 = "/Users/naumanali/Desktop/4th Quarter/Python II/PS
 ↪  4/problem-set-4-sijie-abu-bakar-main/pos2016.csv"
pos2016 = pd.read_csv(filepath1)
pos2016.head()
pos2016["year"] = 2016
pos2016_short = pos2016[(pos2016["PRVDR_CTGRY_CD"] == 1) &
 ↪  (pos2016["PRVDR_CTGRY_SBTYP_CD"] == 1)]
```

```
# pos2016_short.shape
# print(f"Number of facility records: {pos2016_short.shape[0]}")

unique_count1 = pos2016_short['FAC_NAME'].nunique()
unique_count2 = pos2016_short['PRVDR_NUM'].nunique()
print(f"Number of unique PRVDR_NUM values: {unique_count2}")
print(f"Number of unique FAC_NAME values: {unique_count1}")
```

```
Number of unique PRVDR_NUM values: 7245
Number of unique FAC_NAME values: 6770
```

```
a. I use `FAC_NAME` and `PRVDR_NUM` respectively to calculate the unique
value of the short-term hospitals. Under CMS Certification Number, there are
7245 unique hospitals in Q4 2016 data set. Under facility name, there are
6770 unique hospitals in Q4 2016 data set. I believe this number is a bit
large for short-term hospitals in Q4 2016.
```

b. I searched for number of hospitals in 2020 to 2024 on [this website] (https://www.aha.org/statistics/fast-facts-us-hospitals), and discovers that the total number of hospitals in the United States is a little bit more than 6000, with 6039 in 2022 and 6120 in 2024. Also, [Kaiser Family Foundation] (https://www.kff.org/report-section/a-look-at-rural-hospital-closures-and-implications-for-a
states that, there are nearly 5000 short-term hospitals in the US.

I believe the reason behind the difference is that, some hospitals change names over the time, and even though using the unique value of `facility name`, the hospitals are still not unique.

3.

```
filepath2 = "/Users/naumanali/Desktop/4th Quarter/Python II/PS
 ↪  4/problem-set-4-sijie-abu-bakar-main/pos2017.csv"
pos2017 = pd.read_csv(filepath2)
pos2017["year"] = 2017
pos2017_short = pos2017[(pos2017["PRVDR_CTGRY_CD"] == 1) &
 ↪  (pos2017["PRVDR_CTGRY_SBTYP_CD"] == 1)]


filepath3 = "/Users/naumanali/Desktop/4th Quarter/Python II/PS
 ↪  4/problem-set-4-sijie-abu-bakar-main/pos2018.csv"
pos2018 = pd.read_csv(filepath3, encoding='ISO-8859-1')
pos2018["year"] = 2018
pos2018_short = pos2018[(pos2018["PRVDR_CTGRY_CD"] == 1) &
 ↪  (pos2018["PRVDR_CTGRY_SBTYP_CD"] == 1)]

filepath4 = "/Users/naumanali/Desktop/4th Quarter/Python II/PS
 ↪  4/problem-set-4-sijie-abu-bakar-main/pos2019.csv"
pos2019 = pd.read_csv(filepath4, encoding='ISO-8859-1')
pos2019["year"] = 2019
pos2019_short = pos2019[(pos2019["PRVDR_CTGRY_CD"] == 1) &
 ↪  (pos2019["PRVDR_CTGRY_SBTYP_CD"] == 1)]
```

```
print(pos2017.head())
print(pos2018.head())
print(pos2019.head())
```

```
   PRVDR_CTGRY_SBTYP_CD  PRVDR_CTGRY_CD                          FAC_NAME  \
0                   1.0               1  SOUTHEAST ALABAMA MEDICAL CENTER
```

```
1                   1.0               1            NORTH JACKSON HOSPITAL
2                   1.0               1     MARSHALL MEDICAL CENTER SOUTH
3                   1.0               1    ELIZA COFFEE MEMORIAL HOSPITAL
4                   1.0               1          MIZELL MEMORIAL HOSPITAL

  PRVDR_NUM  PGM_TRMNTN_CD  TRMNTN_EXPRTN_DT   ZIP_CD  year
0    010001              0               NaN  36301.0  2017
1    010004              1        20010831.0  35740.0  2017
2    010005              0               NaN  35957.0  2017
3    010006              0               NaN  35631.0  2017
4    010007              0               NaN  36467.0  2017
   PRVDR_CTGRY_SBTYP_CD  PRVDR_CTGRY_CD                          FAC_NAME  \
0                   1.0               1  SOUTHEAST ALABAMA MEDICAL CENTER
1                   1.0               1            NORTH JACKSON HOSPITAL
2                   1.0               1     MARSHALL MEDICAL CENTER SOUTH
3                   1.0               1    ELIZA COFFEE MEMORIAL HOSPITAL
4                   1.0               1          MIZELL MEMORIAL HOSPITAL

  PRVDR_NUM  PGM_TRMNTN_CD  TRMNTN_EXPRTN_DT   ZIP_CD  year
0    010001              0               NaN  36301.0  2018
1    010004              1        20010831.0  35740.0  2018
2    010005              0               NaN  35957.0  2018
3    010006              0               NaN  35631.0  2018
4    010007              0               NaN  36467.0  2018
   PRVDR_CTGRY_SBTYP_CD  PRVDR_CTGRY_CD  \
0                   1.0               1
1                   1.0               1
2                   1.0               1
3                   1.0               1
4                   1.0               1

                            FAC_NAME PRVDR_NUM  PGM_TRMNTN_CD  \
0       SOUTHEAST ALABAMA MEDICAL CENTER    010001              0
1                  NORTH JACKSON HOSPITAL    010004              1
2  MARSHALL MEDICAL CENTERS SOUTH CAMPUS    010005              0
3           NORTH ALABAMA MEDICAL CENTER    010006              0
4               MIZELL MEMORIAL HOSPITAL    010007              0

   TRMNTN_EXPRTN_DT   ZIP_CD  year
0               NaN  36301.0  2019
1       20010831.0  35740.0  2019
2               NaN  35957.0  2019
3               NaN  35630.0  2019
```

```
4                NaN  36467.0  2019
```

```
pos_merge = pd.concat([pos2016_short, pos2017_short, pos2018_short,
 ↪  pos2019_short], ignore_index=True)
pos_merge.shape
pos_merge.to_csv("pos_merge.csv", index = False)
```

The plot of number of observations by year is as followed.

```
import altair as alt

year_count = pos_merge["year"].value_counts().reset_index()
year_count.columns = ["year", "count"]

alt.Chart(year_count).mark_bar().encode(
    x = alt.X("count:Q", title = "Number of Observations"),
    y = alt.Y("year:N", title = "year")
).properties(
    title = "Number of Observations by Year"
)
```

```
alt.Chart(...)
```

4. a.

```
year_count_unique = pos_merge.groupby(
    "year")["PRVDR_NUM"].nunique().reset_index()
year_count_unique.columns = ["year", "count"]

alt.Chart(year_count_unique).mark_bar().encode(
    x=alt.X("count:Q", title="Number of Hospitals"),
    y=alt.Y("year:N", title="year")
).properties(
    title="Number of Unique Hospitals by Year"
)
```

```
alt.Chart(...)
```

b.

These two plots are the same. It means that the CMS certification number corresponds uniquely with the observations of hospitals.

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

1.

```
hospital_active2016 = pos2016_short[pos2016_short["PGM_TRMNTN_CD"] ==
↪  0]["PRVDR_NUM"].unique()
hospital_close_suspect = []
for hospital_id in hospital_active2016:
    hospital_data = pos_merge[pos_merge["PRVDR_NUM"] ==
↪  hospital_id].sort_values(by="year")
    closed = False
    for year in range(2017, 2020):  # 2017 to 2019
        year_data = hospital_data[hospital_data["year"] == year]
        if year_data.empty or (year_data["PGM_TRMNTN_CD"].iloc[0] != 0):
            # If hospital data is missing or not active in this year, record
            ↪  closure details
            facility_name = hospital_data["FAC_NAME"].iloc[0]
            zip_code = hospital_data["ZIP_CD"].iloc[0]
            hospital_close_suspect.append({
                "PRVDR_NUM": hospital_id,
                "FAC_NAME": facility_name,
                "ZIP_CD": zip_code,
                "year": year
            })
            closed = True
            break

hospital_close_suspect = pd.DataFrame(hospital_close_suspect)
print(hospital_close_suspect)
```

```
    PRVDR_NUM                                      FAC_NAME   ZIP_CD  year
0      030001                        ABRAZO MARYVALE CAMPUS  85031.0  2017
1      050196     ADVENTIST MEDICAL CENTER - CENTRAL VALLEY  93230.0  2017
2      360151                       AFFINITY MEDICAL CENTER  44646.0  2018
3      330189 ALBANY MEDICAL CENTER / SOUTH CLINICAL CAMPUS  12208.0  2017
4      450488      ALLEGIANCE SPECIALTY HOSPITAL OF KILGORE  75662.0  2017
..        ...                                           ...      ...   ...
169    010032                           WEDOWEE HOSPITAL  36278.0  2019
170    100234                          WEST PALM HOSPITAL  33407.0  2018
171    520204        WHEATON FRANCISCAN HEALTHCARE FRANKLIN  53132.0  2018
172    190236         WILLIS KNIGHTON BOSSIER HEALTH CENTER  71111.0  2018
173    190205               WOMEN'S AND CHILDREN'S HOSPITAL  70508.0  2019
```

```
[174 rows x 4 columns]
```

There are 174 hospitals fit this definition.

2.

```
hospital_close_suspect = hospital_close_suspect.sort_values(by = "FAC_NAME")

hospital_close_top_10 = hospital_close_suspect[['FAC_NAME', 'year']].head(10)
print(hospital_close_top_10)
```

```
                                      FAC_NAME  year
0                         ABRAZO MARYVALE CAMPUS  2017
1      ADVENTIST MEDICAL CENTER - CENTRAL VALLEY  2017
2                         AFFINITY MEDICAL CENTER  2018
3  ALBANY MEDICAL CENTER / SOUTH CLINICAL CAMPUS  2017
4       ALLEGIANCE SPECIALTY HOSPITAL OF KILGORE  2017
5                         ALLIANCE LAIRD HOSPITAL  2019
6                        ALLIANCEHEALTH DEACONESS  2019
7                   ANNE BATES LEACH EYE HOSPITAL  2019
8         ARKANSAS VALLEY REGIONAL MEDICAL CENTER  2017
9             BANNER CHURCHILL COMMUNITY HOSPITAL  2017
```

3. I devide the question into two steps, and use GPT by asking 1) how to count the active hospitals by ZIP code for each year; and 2) how to identity the zip codes that the number of hospitals do not decrease in the next year.

```
# Step 1: Count active hospitals by ZIP code for each year (0 in
 ↪  PGM_TRMNTN_CD indicates "Active Provider")
active_hospitals_by_zip = pos_merge[pos_merge["PGM_TRMNTN_CD"] == 0].groupby(
    ["ZIP_CD", "year"])["PRVDR_NUM"].nunique().reset_index()
active_hospitals_by_zip.columns = ["ZIP_CD", "year", "active_hospitals"]

# Step 2: Identify ZIP codes to filter out
zip_codes_to_filter = []
for _, row in active_hospitals_by_zip.iterrows():
    zip_code = row["ZIP_CD"]
    current_year = row["year"]
    current_count = row["active_hospitals"]

    # Get count for the following year
    next_year_data =
 ↪  active_hospitals_by_zip[(active_hospitals_by_zip["ZIP_CD"] == zip_code) &
```

```
                                                (active_hospitals_by_zip["year"]
    == current_year + 1)]

    if not next_year_data.empty:
        next_year_count = next_year_data["active_hospitals"].values[0]

        # Check if the count does not decrease
        if next_year_count >= current_count:
            zip_codes_to_filter.append(zip_code)

# Step 3: Filter out suspected closures in these ZIP codes
corrected_closures =
    hospital_close_suspect[~hospital_close_suspect["ZIP_CD"].isin(zip_codes_to_filter)].reset

# Display the remaining suspected closures
print(corrected_closures)
```

```
    index PRVDR_NUM                                        FAC_NAME  \
0       0    030001                              ABRAZO MARYVALE CAMPUS
1       4    450488           ALLEGIANCE SPECIALTY HOSPITAL OF KILGORE
2       8    060036            ARKANSAS VALLEY REGIONAL MEDICAL CENTER
3       9    290006                BANNER CHURCHILL COMMUNITY HOSPITAL
4      16    220050                           BAYSTATE MARY LANE HOSPITAL
5      23    340189     CAROLINAS  CONTINUECARE HOSPITAL AT UNIVERSITY
6      29    450008      CHILDREN'S HOSPITAL -SCOTT & WHITE HEALTHCARE
7      41    450620                              DIMMIT REGIONAL HOSPITAL
8      44    450749                  EAST TEXAS MEDICAL CENTER TRINITY
9      48    050435                           FALLBROOK HOSPITAL DISTRICT
10     53    450293                                FRIO REGIONAL HOSPITAL
11     54    460033                            GARFIELD MEMORIAL HOSPITAL
12     64    320013  HOLY CROSS HOSPITAL A DIV OF TAOS HEALTH SYSTEMS
13     65    670072                                HOPEBRIDGE HOSPITAL
14     68    450580                      HOUSTON COUNTY MEDICAL CENTER
15     72    060043                             KEEFE MEMORIAL HOSPITAL
16     76    400026                                  LAFAYETTE HOSPITAL
17     80    670079               LAKEWAY REGIONAL MEDICAL CENTER, LLC
18     82    100327             LANDMARK HOSPITAL OF SOUTHWEST FLORIDA
19     89    190250                              LOUISIANA HEART HOSPITAL
20     96    490079  MEMORIAL HOSPITAL OF MARTINSVILLE & HENRY COUNTY
21     98    370138       MEMORIAL HOSPITAL OF TEXAS COUNTY AUTHORITY
22    110    670089                       NIX COMMUNITY GENERAL HOSPITAL
23    121    450178                       PECOS COUNTY MEMORIAL HOSPITAL
```

```
24    123    430081                PHS INDIAN HOSPITAL AT PINE RIDGE
25    134    370103                     SAYRE MEMORIAL HOSPITAL, INC
26    143    330066                              ST CLARE'S HOSPITAL
27    162    240132                                   UNITY HOSPITAL

     ZIP_CD  year
0   85031.0  2017
1   75662.0  2017
2   81050.0  2017
3   89406.0  2017
4    1082.0  2017
5   28217.0  2017
6   76502.0  2017
7   78834.0  2017
8   75862.0  2017
9   92028.0  2017
10  78061.0  2017
11  84759.0  2017
12  87571.0  2017
13  77035.0  2017
14  75835.0  2017
15  80810.0  2017
16    714.0  2017
17  78734.0  2017
18  34108.0  2017
19  70445.0  2017
20  24115.0  2018
21  73942.0  2017
22  78017.0  2017
23  79735.0  2017
24  57770.0  2017
25  73662.0  2017
26  12304.0  2017
27  55432.0  2017
```

a.

```
num_mergers = len(hospital_close_suspect) - len(corrected_closures)
print(f"{num_mergers} suspected hospital closures that fit the definition of
 ↪  a merger/acquisition.")
```

```
146 suspected hospital closures that fit the definition of a
merger/acquisition.
```

b.

```
num_corrected = len(corrected_closures)
print(f"{num_corrected} corrected hospital closures is left after the
 ↪  filter.")
```

```
28 corrected hospital closures is left after the filter.
```

c.

```
corrected_closures = corrected_closures.sort_values(by = "FAC_NAME")

corrected_closures_top_10 = corrected_closures[['FAC_NAME', 'year']].head(10)
print(corrected_closures_top_10)
```

```
                                      FAC_NAME  year
0                         ABRAZO MARYVALE CAMPUS  2017
1        ALLEGIANCE SPECIALTY HOSPITAL OF KILGORE  2017
2         ARKANSAS VALLEY REGIONAL MEDICAL CENTER  2017
3             BANNER CHURCHILL COMMUNITY HOSPITAL  2017
4                     BAYSTATE MARY LANE HOSPITAL  2017
5  CAROLINAS  CONTINUECARE HOSPITAL AT UNIVERSITY  2017
6   CHILDREN'S HOSPITAL -SCOTT & WHITE HEALTHCARE  2017
7                        DIMMIT REGIONAL HOSPITAL  2017
8               EAST TEXAS MEDICAL CENTER TRINITY  2017
9                     FALLBROOK HOSPITAL DISTRICT  2017
```

Download Census zip code shapefile (10 pt)

1.

```
import geopandas as gpd
```

```
filepath1 = "/Users/naumanali/Desktop/4th Quarter/Python II/PS
 ↪  4/problem-set-4-sijie-abu-bakar-main/gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.dbf"
data1 = gpd.read_file(filepath1)
data1.head()
```

	GEO_ID	ZCTA5	NAME	LSAD	CENSUSAREA	geometry
0	8600000US01040	01040	01040	ZCTA5	21.281	POLYGON ((-72.62734 42.16203, -72.62
1	8600000US01050	01050	01050	ZCTA5	38.329	POLYGON ((-72.95393 42.34379, -72.95
2	8600000US01053	01053	01053	ZCTA5	5.131	POLYGON ((-72.68286 42.37002, -72.68
3	8600000US01056	01056	01056	ZCTA5	27.205	POLYGON ((-72.39529 42.18476, -72.39
4	8600000US01057	01057	01057	ZCTA5	44.907	MULTIPOLYGON (((-72.39191 42.0806

I don't know how to open and look up a .prj file, so I ask GPT "how to open the .prj file", and it offers me the `with open` code.

```
filepath2 = "/Users/naumanali/Desktop/4th Quarter/Python II/PS
 ↪  4/problem-set-4-sijie-abu-bakar-main/gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.prj"

with open(filepath2, 'r') as file:
    prj_content = file.read()
    print(prj_content)
```

```
GEOGCS["GCS_North_American_1983",DATUM["D_North_American_1983",SPHEROID["GRS_1980",6378137,29
```

```
filepath3 = "/Users/naumanali/Desktop/4th Quarter/Python II/PS
 ↪  4/problem-set-4-sijie-abu-bakar-main/gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.shp"
data3 = gpd.read_file(filepath3)
data3.head()
```

	GEO_ID	ZCTA5	NAME	LSAD	CENSUSAREA	geometry
0	8600000US01040	01040	01040	ZCTA5	21.281	POLYGON ((-72.62734 42.16203, -72.62
1	8600000US01050	01050	01050	ZCTA5	38.329	POLYGON ((-72.95393 42.34379, -72.95
2	8600000US01053	01053	01053	ZCTA5	5.131	POLYGON ((-72.68286 42.37002, -72.68
3	8600000US01056	01056	01056	ZCTA5	27.205	POLYGON ((-72.39529 42.18476, -72.39
4	8600000US01057	01057	01057	ZCTA5	44.907	MULTIPOLYGON (((-72.39191 42.0806

```
filepath4 = "/Users/naumanali/Desktop/4th Quarter/Python II/PS
 ↪  4/problem-set-4-sijie-abu-bakar-main/gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.shx"
data4 = gpd.read_file(filepath4)
data4.head()
```

	GEO_ID	ZCTA5	NAME	LSAD	CENSUSAREA	geometry
0	8600000US01040	01040	01040	ZCTA5	21.281	POLYGON ((-72.62734 42.16203, -72.62
1	8600000US01050	01050	01050	ZCTA5	38.329	POLYGON ((-72.95393 42.34379, -72.95
2	8600000US01053	01053	01053	ZCTA5	5.131	POLYGON ((-72.68286 42.37002, -72.68
3	8600000US01056	01056	01056	ZCTA5	27.205	POLYGON ((-72.39529 42.18476, -72.39
4	8600000US01057	01057	01057	ZCTA5	44.907	MULTIPOLYGON (((-72.39191 42.0806

I don't know how to read a .xml file either, so I asked GPT "how to open the .xml file", and it offers me the `tree` and `root` code.

```
import xml.etree.ElementTree as ET

# Replace with the path to your .xml file
filepath5 = "/Users/naumanali/Desktop/4th Quarter/Python II/PS
 ↪  4/problem-set-4-sijie-abu-bakar-main/gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.xml"

# Parse the XML file
tree = ET.parse(filepath5)
root = tree.getroot()

# Example: Print out the tags and text in the XML
for elem in root.iter():
    print(elem.tag, elem.text)
```

```
metadata

idinfo

citation

citeinfo

origin U.S. Department of Commerce, U.S. Census Bureau, Geography
Division/Cartographic Products Branch
pubdate 2010
title 2010 Cartographic Boundary File, 5-Digit ZIP code Tabulation Area for
United States, 1:500,000
geoform vector digital data
serinfo

sername 2010 Census
```

issue Cartographic Boundary File
onlink http://www2.census.gov/geo/tiger/GENZ2010/
descript

abstract The 2010 cartographic boundary shapefiles are simplified representations of selected geographic areas from the Census Bureau's MAF/TIGER geographic database. These boundary files are specifically designed for small scale thematic mapping. When possible generalization is performed with intent to maintain the hierarchical relationships among geographies and to maintain the alignment of geographies within a file set for a given year. Geographic areas may not align with the same areas from another year. Some geographies are available as nation-based shapefiles while others are available only as state-based files.
purpose To improve the appearance of shapes at small scales, areas are represented with fewer vertices than detailed TIGER/Line equivalents. These boundary files take up less disk space than their ungeneralized counterparts. Cartographic boundary files take less time to render on screen. These files were specifically created to support small-scale thematic mapping.
timeperd

timeinfo

rngdates

begdate 201109
enddate 201109
current publication date
status

progress Complete
update None planned. New cartographic boundary files will be produced on an annual release schedule. Types of geography released may vary from year to year.
spdom

bounding

westbc -167.650000
eastbc -65.221527
northbc 71.342941
southbc -14.605210

keywords

theme

themekt None
themekey 2010 Census
themekey Cartographic Boundary
themekey Generalized
themekey Shapefile
themekey ZCTA
themekey ZIP code Tabulation Area
theme

themekt ISO 19115 Topic Categories
themekey Boundaries
place

placekt None

placekey
United States

placekey
US

accconst None
useconst These products are free to use in a product or publication, however acknowledgement must be given to the U.S. Census Bureau as the source. The boundary information is for visual display at appropriate small scales only. Cartographic boundary files should not be used for geographic analysis including area or perimeter calculation. Files should not be used for geocoding addresses. Files should not be used for determining precise geographic area relationships.
ptcontac

cntinfo

cntorgp

cntorg U.S. Department of Commerce, U.S. Census Bureau, Geography Division
cntaddr

addrtype mailing
address 4600 Silver Hill Road

city Washington
state DC
postal 20233-7400
country United States
cntvoice 301.763.1128
cntfax 301.763.4710
cntemail geo.geography@census.gov
native Files were generated by U.S. Census Bureau Linux-based batch generalization software. FME (by Safe Software) was used to convert polygons from Oracle Spatial format into ESRI shapefile format.
dataqual

attracc

attraccr Accurate against American National Standards Institute (ANSI) Publication INCITS 446-2008 (Geographic Names Information System (GNIS)) at the 100% level for the codes and base names present in the file. Some entities may have NULL name and/or Legal/Statistical Area Description (LSAD) attribution and this does not imply, in all cases, that the entities do not have names and/or LSADs. The remaining attribute information has been examined but has not been fully tested for accuracy.
logic The Census Bureau performed automated tests to ensure logical consistency of the source database. Segments making up the outer and inner boundaries of a polygon tie end-to-end to completely enclose the area. All polygons were tested for closure. The Census Bureau uses its internally developed geographic update system to enhance and modify spatial and attribute data in the Census MAF/TIGER database. Standard geographic codes, such as FIPS codes for states, counties, municipalities, county subdivisions, places, American Indian/Alaska Native/Native Hawaiian areas, and congressional districts are used when encoding spatial entities. The Census Bureau performed spatial data tests for logical consistency of the codes during the compilation of the original Census MAF/TIGER database files. Feature attribute information has been examined but has not been fully tested for consistency.
complete The Cartographic Boundary Files are generalized representations of extracts taken from the MAF/TIGER Database. Generalized boundary files are clipped to a simplified version of the U.S. outline. As a result, some off-shore areas may be excluded from the generalized files. Some small holes or discontiguous parts of areas are not included in generalized files.
posacc

horizpa

horizpar Data are not accurate. Data are generalized representations of 2010 Census geographic boundaries at 1:500,000.
lineage

srcinfo

srccite

citeinfo

origin U.S. Department of Commerce, U.S. Census Bureau, Geography Division
pubdate unpublished material
title Master Address File/Topologically Integrated Geographic Encoding and Referencing Database: Final 2010 Tabulation Benchmark (TAB10)
typesrc Geo-spatial Relational Database
srctime

timeinfo

rngdates

begdate 20100101
enddate 20100101
srccurr The dates describe the effective date of 2010 Census boundaries.
srccitea MAF/TIGER
srccontr All spatial and feature data
procstep

procdesc Spatial data were extracted from the MAF/TIGER database and processed through a U.S. Census Bureau batch generalization system.
srcused MAF/TIGER
procdate 201109
spdoinfo

indspref INCITS (formerly FIPS) codes.
direct Vector
ptvctinf

sdtsterm

sdtstype G-polygon
ptvctcnt 33120
spref

horizsys

geograph

latres 0.000458
longres 0.000458
geogunit Decimal degrees
geodetic

horizdn North American Datum of 1983
ellips Geodetic Reference System 80
semiaxis 6378137.000000
denflat 298.257222
eainfo

overview

eaover The 2010 cartographic boundary polygon shapefiles contain boundary data for a specific type of Census geography. See the U.S. Census Bureau Geographic Terms and Concepts documentation for further explanation of Census geography (http://www.census.gov/geo/www/2010census/GTC_10.pdf).

Field names, descriptions and data types are listed in this format: FIELD_NAME Description [data value or type].

At a miniumum, the following four fields will appear in every file: GEO_ID Unique identifier for a geographic entity [alphanumeric]; LSAD Standard abbreviation translation of Legal/Statistical Area Description (LSAD) code as used on census maps [alpha]; NAME Name without translated Legal/Statistical Area Description (LSAD) [alphanumeric]; CENSUSAREA Area of entity before generalization in square miles [numeric]. The CENSUSAREA field is the calculated area derived from the ungeneralized area of each entity. This field can be used in density calculations. Users should not calculate the area of the generalized polygons for use in any analysis. Use the GEO_ID field to join these spatial tables to 2010 Census data tables. NAME and/or LSAD may be NULL. This does not imply the entities do not have names and/or LSADs but rather the names and/or LSADs may not have been populated for a given file.

Additional attribution for a shapefile will vary depending on the fields appropriate for a given type of geography. These fields may include: AIANHH American Indian Area/Alaska Native Area/Hawaiian Home Land [4-digit Census code]; AIHHTLI American Indian Trust Land/Hawaiian Home Land Indicator ['R' for reservation, 'T' for off-reservation trust land], AITSCE American Indian Tribal Subdivision [3-digit Census code]; ANRC Alaska Native Regional Corporation [5-digit FIPS code]; BLKGRP Block Group [1-digit Census code]; CBSA Metropolitan Statistical Area/Micropolitan Statistical Area [5-digit FIPS code]; CD Congressional District [2-digit FIPS code]; CNECTA Combined New England City and Town Area [3-digit FIPS code]; CONCIT Consolidated City [5-digit FIPS code]; COUNTY County [3-digit FIPS code]; COUSUB County Subdivision [5-digit FIPS code]; CSA Combined Statistical Area [3-digit Census code]; DIVISION Census Division [1-digit Census code]; METDIV Metropolitan Division [5-digit FIPS code]; NECTA New England City and Town Area [5-digit FIPS code]; NECTADIV New England City and Town Area Division [5-digit FIPS code]; PLACE Place [5-digit FIPS code]; REGION Census Region [1-digit Census code]; SDELM Elementary School District [5-digit Local Education Agency code]; SDSEC Secondary School District [5-digit Local Education Agency code]; SDUNI Unified School District [5-digit Local Education Agency code]; SLDL State Legislative District (Lower Chamber) [alphanumeric]; SLDU State Legislative District (Upper Chamber) [alphanumeric]; STATE State [2-digit FIPS code]; SUBMCD Subminor Civil Division [5-digit FIPS code] TBLKGRP Tribal Block Group [1-digit alphanumeric]; TRACT Tract [6-digit Census code]; TTRACT Tribal Census Tract [6-digit Census code]; VTD Voting District [alphanumeric]; ZCTA5 ZIP Code Tabulation Area [5-digit Census code].

Some nation-based files may include Puerto Rico and the Island Areas of the United States. Island areas include American Samoa, Guam, the Commonwealth of the Northern Mariana Islands (Northern Mariana Islands), and the United States Virgin Islands.
eadetcit http://www.census.gov/geo/www/2010census/GTC_10.pdf
distinfo

distrib

cntinfo

cntorgp

cntorg U.S. Department of Commerce, U.S. Census Bureau, Geography Division
cntaddr

addrtype mailing
address 4600 Silver Hill Road
city Washington
state DC
postal 20233-7400
country United States
cntvoice 301.763.1128
distliab No warranty, expressed or implied is made with regard to the accuracy of these data, and no liability is assumed by the U.S. Government in general or the U.S. Census Bureau in specific as to the spatial or attribute accuracy of the data. The act of distribution shall not constitute any such warranty and no responsibility is assumed by the U.S. government in the use of these files. The boundary information is for small-scale mapping purposes only; boundary depiction and designation for small-scale mapping purposes do not constitute a determination of jurisdictional authority or rights of ownership or entitlement and they are not legal land descriptions.
stdorder

digform

digtinfo

formname SHP (compressed)
filedec The files were compressed using Linux-based Info-ZIP Zip 2.32. Files can be decompressed with PK-ZIP, version 1.93A or higher, WinZip or other decompression software packages.
digtopt

onlinopt

computer

networka

networkr http://www.census.gov/geo/www/cob/index.html
fees The online cartographic boundary files may be downloaded without charge.
techpreq The cartographic boundary files contain geographic data only and do not include display or mapping software or statistical data. The files are delivered as zipped ESRI Shapefiles. Users are responsible for converting or translating the files into a format used by their specific software packages.
metainfo

metd 201109

```
metc

cntinfo

cntorgp

cntorg U.S. Department of Commerce, U.S. Census Bureau, Geography
Division/Cartographic Products Branch
cntaddr

addrtype mailing
address 4600 Silver Hill Road
city Washington
state DC
postal 20233-7400
country United States
cntvoice 301.763.1101
cntfax 301.763.4710
cntemail geo.geography@census.gov
metstdn Content Standard for Digital Geospatial Metadata
metstdv FGDC-STD-001-1998
```

a.

The five types are .dbf, .prj, .shp, .shx, .xml, respectively. .dbf, .shp, and .shx looks like a DataFrame and have the same content, including variables `CEO_ID`, `ZCTA5`, `NAME`, `LSAD`, `CENSUSAREA`, and `geometry`. .prj only contains a string, with values of `GEOGCS`, `PRIMEM`, `UNIT`. .xml is an Element type file, contains metadata and many texts.

b.

```python
import os

filepath_list = [
    filepath1,
    filepath2,
    filepath3,
    filepath4,
    filepath5
]

for filepath in filepath_list:
    file_name = os.path.basename(filepath)
```

```
    file_size = os.path.getsize(filepath)
    file_size_mb = file_size/(1024**2)
    print(f"{file_name} size: {file_size_mb:.2f} MB")
```

```
gz_2010_us_860_00_500k.dbf size: 6.13 MB
gz_2010_us_860_00_500k.prj size: 0.00 MB
gz_2010_us_860_00_500k.shp size: 798.74 MB
gz_2010_us_860_00_500k.shx size: 0.25 MB
gz_2010_us_860_00_500k.xml size: 0.01 MB
```

2.

```
zip = data3
zip.head()
zip.shape
```

```
(33120, 6)
```

```
zip_tx =
 ↪  zip[zip["NAME"].str[:3].isin(["733"])|zip["NAME"].str[:2].isin(["75",
 ↪  "76", "77", "78", "79"])]
zip_tx['ZIP_CD'] = zip_tx['ZCTA5']
zip_tx.shape
```

```
/Users/naumanali/Library/Python/3.9/lib/python/site-packages/geopandas/geodataframe.py:1819:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  super().__setitem__(key, value)
```

```
(1935, 7)
```

```
zipcode_tx = zip_tx["ZIP_CD"].unique()
zipcode_tx.shape
```

```
(1935,)
```

```
pos2016_short.head()
```

	PRVDR_CTGRY_SBTYP_CD	PRVDR_CTGRY_CD	FAC_NAME
16	1.0	1	(CLOSED) BAPTIST HICKMAN COMMU
20	1.0	1	(CLOSED) BAPTIST HOSPITAL OF ROA
21	1.0	1	(CLOSED) BAPTIST MEMORIAL HOSPIT
22	1.0	1	(CLOSED) BAPTIST MEMORIAL HOSPIT
26	1.0	1	(CLOSED) BEHAVIORAL HEALTHCARE

Below are all the hospitals counts in zip code nationwide.

```
count_zip2016 = pos2016_short.groupby("ZIP_CD").size().reset_index(name =
↪  "hospital_count")
count_zip2016
```

	ZIP_CD	hospital_count
0	603.0	1
1	613.0	1
2	614.0	1
3	618.0	1
4	625.0	1
...	...	...
5671	99801.0	1
5672	99833.0	1
5673	99835.0	2
5674	99901.0	1
5675	99929.0	1

```
pos2016_tx =
↪  pos2016_short[pos2016_short["ZIP_CD"].astype(str).str[:3].isin(["733"])|pos2016_short["Z
↪  "76", "77", "78", "79"])]

pos2016_tx_zipcount = pos2016_tx.groupby("ZIP_CD").size().reset_index(name =
↪  "hospital_count")
pos2016_tx_zipcount
```

	ZIP_CD	hospital_count
0	733.0	4
1	753.0	1
2	761.0	1
3	780.0	1
4	785.0	2
...	...	...
527	79905.0	1
528	79915.0	1
529	79925.0	1
530	79936.0	1
531	79938.0	1

```
pos2016_tx_zipcount = pd.DataFrame(pos2016_tx_zipcount)
pos2016_tx_zipcount['ZIP_CD'] = pos2016_tx_zipcount['ZIP_CD'].fillna(0)
pos2016_tx_zipcount.loc[:, 'ZIP_CD'] =
 ↪  pos2016_tx_zipcount['ZIP_CD'].astype(int).astype(str).str.zfill(5)
```

```
/var/folders/3r/jbvcnnqx3j54f6w2f5xynvgw0000gn/T/ipykernel_15810/3010439078.py:3:
FutureWarning: Setting an item of incompatible dtype is deprecated and will
raise in a future error of pandas. Value '['00733' '00753' '00761' '00780'
'00785' '00791' '00792' '07503' '07514'
 '07601' '07604' '07631' '07652' '07662' '07666' '07675' '07701' '07728'
 '07733' '07740' '07754' '07801' '07834' '07840' '07860' '07902' '07962'
 '75001' '75010' '75013' '75020' '75028' '75032' '75034' '75035' '75039'
 '75041' '75042' '75051' '75057' '75061' '75063' '75069' '75070' '75071'
 '75075' '75082' '75087' '75088' '75090' '75091' '75093' '75098' '75110'
 '75115' '75119' '75140' '75142' '75146' '75149' '75160' '75165' '75182'
 '75203' '75204' '75206' '75208' '75214' '75217' '75218' '75220' '75224'
 '75226' '75227' '75230' '75231' '75234' '75235' '75237' '75243' '75246'
 '75390' '75401' '75418' '75426' '75428' '75455' '75457' '75460' '75462'
 '75482' '75494' '75501' '75502' '75503' '75504' '75551' '75563' '75568'
 '75570' '75601' '75605' '75633' '75644' '75647' '75652' '75657' '75662'
 '75668' '75670' '75684' '75686' '75701' '75702' '75708' '75751' '75766'
 '75773' '75783' '75785' '75801' '75831' '75835' '75840' '75844' '75860'
 '75862' '75901' '75904' '75935' '75948' '75951' '75956' '75961' '75966'
 '75972' '75979' '76011' '76012' '76015' '76017' '76018' '76020' '76021'
 '76022' '76026' '76028' '76033' '76043' '76048' '76051' '76054' '76063'
 '76067' '76086' '76092' '76104' '76106' '76107' '76108' '76110' '76117'
 '76132' '76177' '76180' '76201' '76208' '76210' '76227' '76230' '76234'
 '76240' '76244' '76252' '76255' '76262' '76301' '76310' '76351' '76360'
```

```
'76365' '76374' '76380' '76384' '76401' '76424' '76426' '76430' '76437'
'76442' '76444' '76446' '76448' '76450' '76454' '76457' '76458' '76470'
'76483' '76502' '76508' '76520' '76528' '76531' '76542' '76548' '76550'
'76567' '76570' '76574' '76634' '76642' '76645' '76648' '76661' '76665'
'76667' '76691' '76692' '76693' '76712' '76773' '76801' '76821' '76825'
'76834' '76837' '76844' '76849' '76856' '76859' '76877' '76903' '76904'
'76905' '76932' '76936' '76943' '76945' '76950' '76951' '77002' '77004'
'77006' '77008' '77015' '77020' '77024' '77027' '77028' '77029' '77030'
'77035' '77043' '77054' '77055' '77058' '77063' '77065' '77070' '77074'
'77076' '77077' '77080' '77081' '77082' '77087' '77090' '77091' '77093'
'77094' '77304' '77325' '77327' '77328' '77338' '77339' '77340' '77351'
'77375' '77380' '77384' '77401' '77414' '77418' '77429' '77434' '77437'
'77445' '77450' '77459' '77465' '77469' '77474' '77477' '77478' '77479'
'77480' '77488' '77494' '77502' '77503' '77504' '77505' '77506' '77511'
'77514' '77515' '77520' '77521' '77530' '77547' '77550' '77555' '77566'
'77575' '77584' '77591' '77592' '77598' '77612' '77619' '77625' '77627'
'77630' '77640' '77642' '77656' '77665' '77701' '77702' '77706' '77707'
'77802' '77833' '77836' '77842' '77845' '77859' '77864' '77868' '77901'
'77902' '77904' '77954' '77957' '77962' '77963' '77964' '77979' '77984'
'77995' '78010' '78013' '78017' '78026' '78028' '78041' '78044' '78061'
'78102' '78114' '78118' '78119' '78130' '78155' '78164' '78201' '78204'
'78205' '78207' '78212' '78222' '78223' '78224' '78229' '78230' '78232'
'78233' '78240' '78247' '78249' '78258' '78285' '78332' '78336' '78355'
'78363' '78377' '78390' '78404' '78411' '78412' '78414' '78415' '78469'
'78503' '78520' '78526' '78539' '78550' '78572' '78580' '78582' '78586'
'78596' '78602' '78611' '78613' '78624' '78626' '78629' '78636' '78640'
'78643' '78644' '78648' '78654' '78664' '78666' '78681' '78701' '78702'
'78704' '78705' '78731' '78734' '78737' '78746' '78756' '78758' '78759'
'78763' '78801' '78834' '78840' '78852' '78861' '78880' '78934' '78942'
'78945' '78957' '78962' '79007' '79014' '79015' '79022' '79027' '79029'
'79035' '79041' '79045' '79065' '79070' '79072' '79079' '79081' '79088'
'79095' '79096' '79106' '79109' '79124' '79201' '79225' '79227' '79235'
'79241' '79245' '79248' '79252' '79312' '79316' '79322' '79323' '79331'
'79336' '79339' '79346' '79347' '79356' '79360' '79364' '79373' '79401'
'79410' '79412' '79413' '79415' '79416' '79501' '79502' '79512' '79520'
'79521' '79529' '79546' '79549' '79553' '79556' '79567' '79601' '79605'
'79606' '79701' '79703' '79704' '79706' '79714' '79720' '79731' '79735'
'79744' '79745' '79752' '79756' '79760' '79761' '79772' '79778' '79782'
'79830' '79855' '79901' '79902' '79904' '79905' '79915' '79925' '79936'
'79938']' has dtype incompatible with float64, please explicitly cast to a
compatible dtype first.
 pos2016_tx_zipcount.loc[:, 'ZIP_CD'] =
 pos2016_tx_zipcount['ZIP_CD'].astype(int).astype(str).str.zfill(5)
```

```
zip_tx = gpd.GeoDataFrame(zip_tx)
zip_tx.loc[:, 'NAME'] = zip_tx['NAME'].astype(int).astype(str)
```

```
print(zip_tx.dtypes)
print(pos2016_tx.dtypes)
```

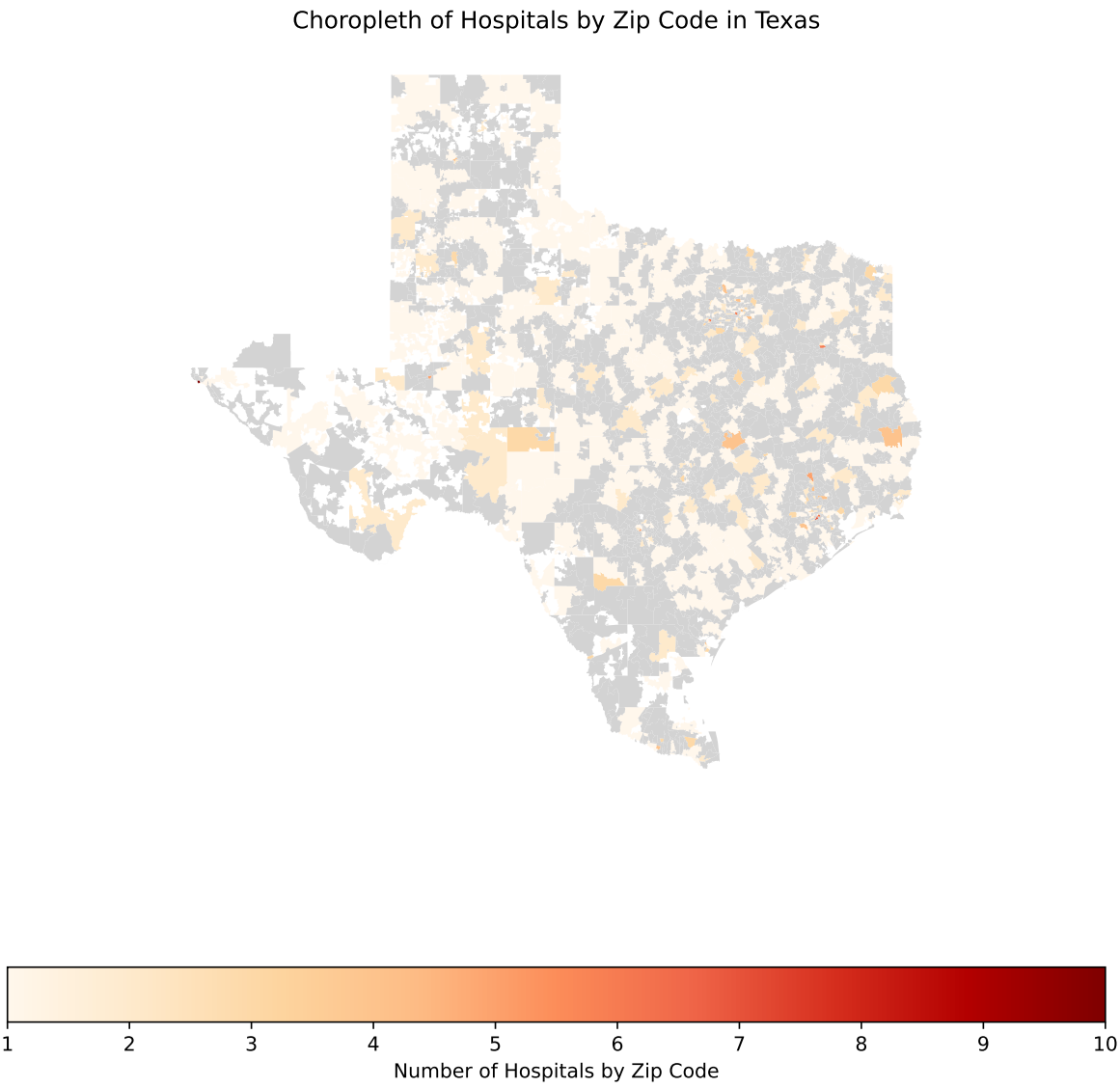
```
GEO_ID          object
ZCTA5           object
NAME            object
LSAD            object
CENSUSAREA     float64
geometry      geometry
ZIP_CD          object
dtype: object
PRVDR_CTGRY_SBTYP_CD    float64
PRVDR_CTGRY_CD            int64
FAC_NAME                 object
PRVDR_NUM                object
PGM_TRMNTN_CD             int64
TRMNTN_EXPRTN_DT        float64
ZIP_CD                  float64
year                      int64
dtype: object
```

```
import matplotlib.pyplot as plt
choropleth_data = zip_tx.merge(pos2016_tx_zipcount, on="ZIP_CD", how="left")
choropleth_data.head()
```

	GEO_ID	ZCTA5	NAME	LSAD	CENSUSAREA	geometry
0	8600000US78624	78624	78624	ZCTA5	708.041	POLYGON ((-98.96423 30.49848, -98.96
1	8600000US78626	78626	78626	ZCTA5	93.046	POLYGON ((-97.60944 30.57185, -97.61
2	8600000US78628	78628	78628	ZCTA5	73.382	POLYGON ((-97.69285 30.57122, -97.69
3	8600000US78631	78631	78631	ZCTA5	325.074	POLYGON ((-99.13053 30.36555, -99.13
4	8600000US78632	78632	78632	ZCTA5	96.278	POLYGON ((-97.40946 29.75929, -97.40

```
choropleth_data.to_csv("choropleth_data.csv", index = False)
```

```
fig, ax = plt.subplots(1, 1, figsize=(10, 10))
choropleth_data.plot(column='hospital_count', ax=ax, legend=True,
            legend_kwds={'label': "Number of Hospitals by Zip Code",
                         'orientation': "horizontal"},
            cmap='OrRd', missing_kwds={'color': 'lightgrey'})
ax.set_title('Choropleth of Hospitals by Zip Code in Texas')
ax.set_axis_off()

plt.show()
```



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

1.

```
zips_all_centroids = zip.copy().to_crs(epsg=32614)
zips_all_centroids["geometry"] = zips_all_centroids.centroid
dimensions = zips_all_centroids.shape
print(dimensions)
zips_all_centroids.head()
```

```
(33120, 6)
```

	GEO_ID	ZCTA5	NAME	LSAD	CENSUSAREA	geometry
0	8600000US01040	01040	01040	ZCTA5	21.281	POINT (2680901.614 5023480.354)
1	8600000US01050	01050	01050	ZCTA5	38.329	POINT (2659157.771 5025591.454)
2	8600000US01053	01053	01053	ZCTA5	5.131	POINT (2669825.294 5037254.622)
3	8600000US01056	01056	01056	ZCTA5	27.205	POINT (2696847.204 5026281.101)
4	8600000US01057	01057	01057	ZCTA5	44.907	POINT (2711728.101 5018798.677)

The dimension of this GeoDataFrame is (33120, 6). For each column, `GEO_ID` is the id, `ZCTA5` is the zip code, `NAME` is the zip code as well, `CENSUSAREA` is the area, `geometry` is the centroid of each zip code.

2.

```
texas_prefixes = ['75', '76', '77', '78', '79', '733']
bordering_states_prefixes = texas_prefixes + ['73', '74', '70', '71', '87',
↪  '88']

zips_texas_centroids =
↪  zips_all_centroids[zips_all_centroids['NAME'].str[:3].isin(['733']) |
↪  zips_all_centroids['NAME'].str[:2].isin(['75', '76', '77', '78',
↪  '79'])].copy()

zips_texas_centroids = zips_texas_centroids.to_crs(epsg=32614)

zips_texas_borderstates_centroids =
↪  zips_all_centroids[zips_all_centroids['NAME'].str[:2].isin(bordering_states_prefixes)].co

zips_texas_borderstates_centroids =
↪  zips_texas_borderstates_centroids.to_crs(epsg=32614)
```

```
texas_zip_count = zips_texas_centroids['NAME'].nunique()
texas_borderstates_zip_count =
 ↪  zips_texas_borderstates_centroids['NAME'].nunique()

print(f"There are {texas_zip_count} unique zip codes zips_texas_centroids
 ↪  subsets.")
print(f"There are {texas_borderstates_zip_count} unique zip codes
 ↪  zips_texas_borderstates_centroids subsets.")
```

```
There are 1935 unique zip codes zips_texas_centroids subsets.
There are 3596 unique zip codes zips_texas_borderstates_centroids subsets.
```

```
zips_texas_centroids.head()
```

	GEO_ID	ZCTA5	NAME	LSAD	CENSUSAREA	geometry
9207	8600000US78624	78624	78624	ZCTA5	708.041	POINT (511823.895 3349990.765)
9208	8600000US78626	78626	78626	ZCTA5	93.046	POINT (634378.798 3393352.86)
9209	8600000US78628	78628	78628	ZCTA5	73.382	POINT (619673.552 3390490.031)
9210	8600000US78631	78631	78631	ZCTA5	325.074	POINT (470662.193 3356245.449)
9211	8600000US78632	78632	78632	ZCTA5	96.278	POINT (647977.423 3286112.808)

3.

```
zips_texas_borderstates_centroids =
 ↪  zips_texas_borderstates_centroids.to_crs(epsg=32614)
count_zip2016 = pd.DataFrame(count_zip2016)
count_zip2016['ZIP_CD'] = count_zip2016['ZIP_CD'].fillna(0)
count_zip2016.loc[:, 'ZIP_CD'] =
 ↪  count_zip2016['ZIP_CD'].astype(int).astype(str).str.zfill(5)
zips_withhospital_centroids = zips_texas_borderstates_centroids.merge(
    count_zip2016,
    left_on='NAME',
    right_on='ZIP_CD',
    how='inner'
)
```

```
/var/folders/3r/jbvcnnqx3j54f6w2f5xynvgw0000gn/T/ipykernel_15810/2177933344.py:4:
FutureWarning: Setting an item of incompatible dtype is deprecated and will
raise in a future error of pandas. Value '['00603' '00613' '00614' ...
'99835' '99901' '99929']' has dtype incompatible with float64, please
explicitly cast to a compatible dtype first.
  count_zip2016.loc[:, 'ZIP_CD'] =
  count_zip2016['ZIP_CD'].astype(int).astype(str).str.zfill(5)
```

I choose inner merge, with `NAME` variable for `zips_texas_borderstates_centroids` and `ZIP_CD` variable for `count_zip2016`. Both `NAME` and `ZIP_CD` represent the zip code.

4. a.

```
zips_texas_centroids_10 = zips_texas_centroids.head(10)
zips_texas_centroids_10 = zips_texas_centroids_10.to_crs(epsg=4326)
zips_texas_centroids_10
zips_withhospital_centroids = zips_withhospital_centroids.to_crs(epsg=4326)
```

```
# Convert both GeoDataFrames to a suitable projected CRS (NAD83 / Texas
 ↪ Central, EPSG:2272)
zips_texas_centroids_10 = zips_texas_centroids_10.to_crs(epsg=2272)
zips_withhospital_centroids = zips_withhospital_centroids.to_crs(epsg=2272)

import time
start_time = time.time()

nearest_distances = []

# Calculate the nearest distance for each zip code
for _, zip_row in zips_texas_centroids_10.iterrows():
    # Calculate distance to all hospitals
    distances =
 ↪ zips_withhospital_centroids.geometry.distance(zip_row.geometry)
    # Get the minimum distance
    nearest_distance = distances.min()
    nearest_distances.append(nearest_distance)

# Add the nearest distances to the GeoDataFrame
zips_texas_centroids_10['nearest_hospital_distance'] = nearest_distances

# View the updated GeoDataFrame
print(zips_texas_centroids_10[['ZCTA5', 'nearest_hospital_distance']])
```

```
end_time = time.time()
print(f"Execution time: {end_time - start_time:.2f} seconds")
```

```
      ZCTA5  nearest_hospital_distance
9207  78624                  0.000000
9208  78626                  0.000000
9209  78628              40598.770716
9210  78631             121599.385074
9211  78632              52997.260441
9212  78633              57441.613822
9213  78634              37964.054159
9214  78635              56972.702620
9215  78636                  0.000000
9216  78638              53332.237074
Execution time: 0.02 seconds
```

For 10 records, it takes 0.02 seconds. For there are 1935 records, I think the total calculation time is 0.02 * 193 = 4 seconds.

b.

```
start_time = time.time()

nearest_distances = []

# Calculate the nearest distance for each zip code
for _, zip_row in zips_texas_centroids.iterrows():
    # Calculate distance to all hospitals
    distances =
 ↪  zips_withhospital_centroids.geometry.distance(zip_row.geometry)
    # Get the minimum distance
    nearest_distance = distances.min()
    nearest_distances.append(nearest_distance)

# Add the nearest distances to the GeoDataFrame
zips_texas_centroids['nearest_hospital_distance'] = nearest_distances

# View the updated GeoDataFrame
print(zips_texas_centroids[['ZCTA5', 'nearest_hospital_distance']])
```

```
end_time = time.time()
print(f"Execution time: {end_time - start_time:.2f} seconds")
```

```
       ZCTA5  nearest_hospital_distance
9207   78624               5.156244e+06
9208   78626               5.272121e+06
9209   78628               5.259872e+06
9210   78631               5.132721e+06
9211   78632               5.204933e+06
...      ...                        ...
32917  78261               5.140870e+06
32918  78368               5.058712e+06
32919  78412               5.062000e+06
32920  78557               4.876721e+06
32921  78586               4.922249e+06

[1935 rows x 2 columns]
Execution time: 2.01 seconds
```

The difference is close.

c.

```
with open(filepath2, 'r') as file:
    prj_content = file.read()
print(prj_content)
```

```
GEOGCS["GCS_North_American_1983",DATUM["D_North_American_1983",SPHEROID["GRS_1980",6378137,29
```

5. a. It is in feet now becasue I reprojected it to EPSG:2272

   b.

```
import pandas as pd
import geopandas as gpd

# Load zip and hospital data, assuming 'zips_texas_centroids' and
 ↪  'zips_withhospital_centroids' are prepared from previous steps

# Ensure both GeoDataFrames are in the same projected CRS for accurate
 ↪  distance measurement
# EPSG:2272 (NAD83 / Pennsylvania South (ftUS)) or another suitable Texas
 ↪  projection (feet)
```

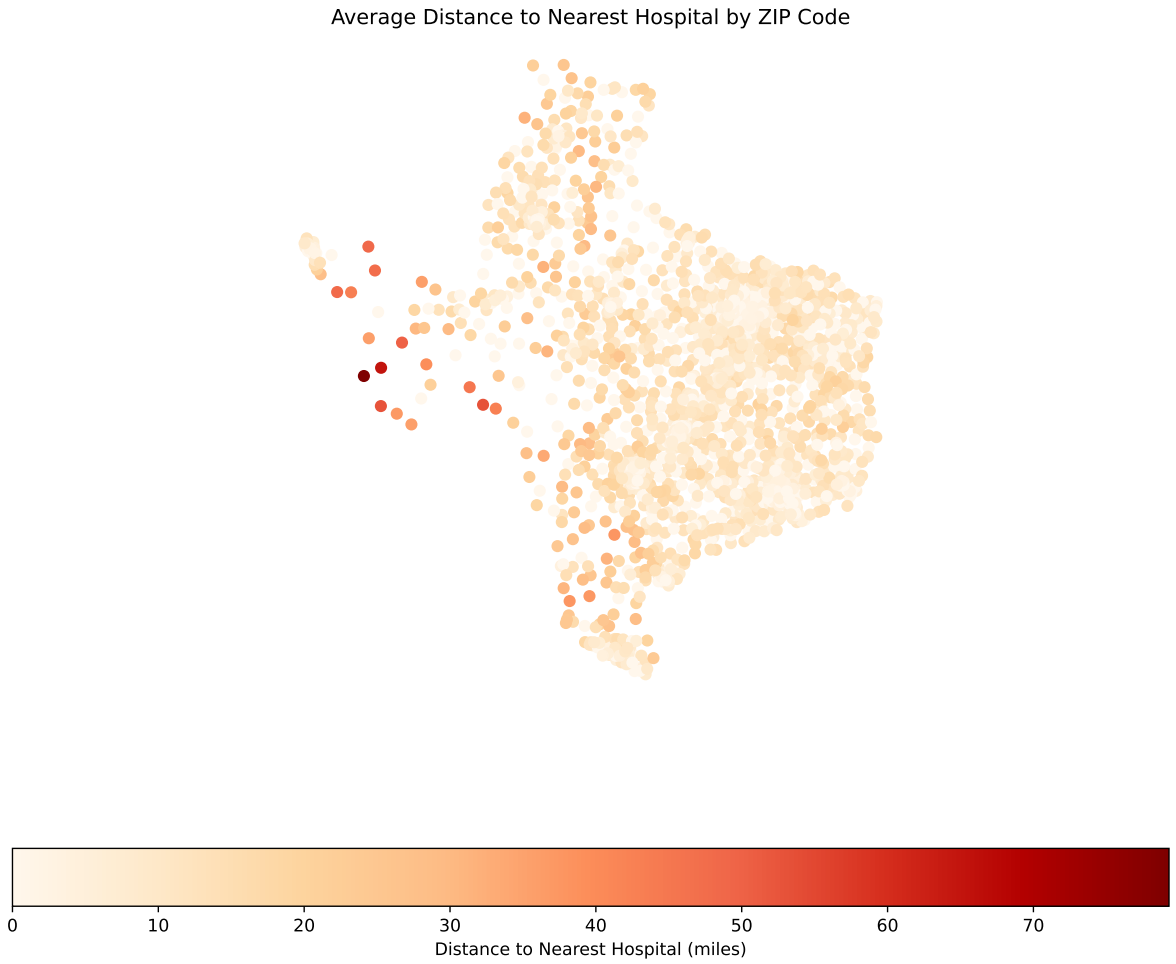
```
zips_texas_centroids = zips_texas_centroids.to_crs(epsg=2272)
zips_withhospital_centroids = zips_withhospital_centroids.to_crs(epsg=2272)

# Initialize a list to store nearest hospital distances for each ZIP centroid
nearest_distances = []

# Calculate the minimum distance for each ZIP centroid in Texas to the
 ↪  nearest hospital centroid
for _, zip_row in zips_texas_centroids.iterrows():
    distances =
 ↪  zips_withhospital_centroids.geometry.distance(zip_row.geometry)
    nearest_distance = distances.min()  # Minimum distance to the nearest
 ↪  hospital
    nearest_distances.append(nearest_distance)

# Add the calculated distances (in feet) to the GeoDataFrame and convert to
 ↪  miles
zips_texas_centroids['nearest_hospital_distance_feet'] = nearest_distances
zips_texas_centroids['nearest_hospital_distance_miles'] =
 ↪  zips_texas_centroids['nearest_hospital_distance_feet'] / 5280

# Calculate and display the average distance to the nearest hospital (in
 ↪  miles)
average_distance_miles =
 ↪  zips_texas_centroids['nearest_hospital_distance_miles'].mean()
print(f"Average distance to the nearest hospital in miles:
 ↪  {average_distance_miles:.2f}")
```

```
Average distance to the nearest hospital in miles: 8.29
```

c.

```
import matplotlib.pyplot as plt

# Plot the map
fig, ax = plt.subplots(1, 1, figsize=(12, 10))
zips_texas_centroids.plot(column='nearest_hospital_distance_miles',
                          ax=ax,
                          legend=True,
                          legend_kwds={'label': "Distance to Nearest Hospital
 ↪  (miles)", 'orientation': "horizontal"},
                          cmap='OrRd',  # Choose an appropriate color map
```

```
                          missing_kwds={'color': 'lightgrey'})  # Color for
↪  missing values

# Customize the plot
ax.set_title('Average Distance to Nearest Hospital by ZIP Code')
ax.set_axis_off()  # Hide the axis for a cleaner map

plt.show()
```



Effects of closures on access in Texas (15 pts)

1.

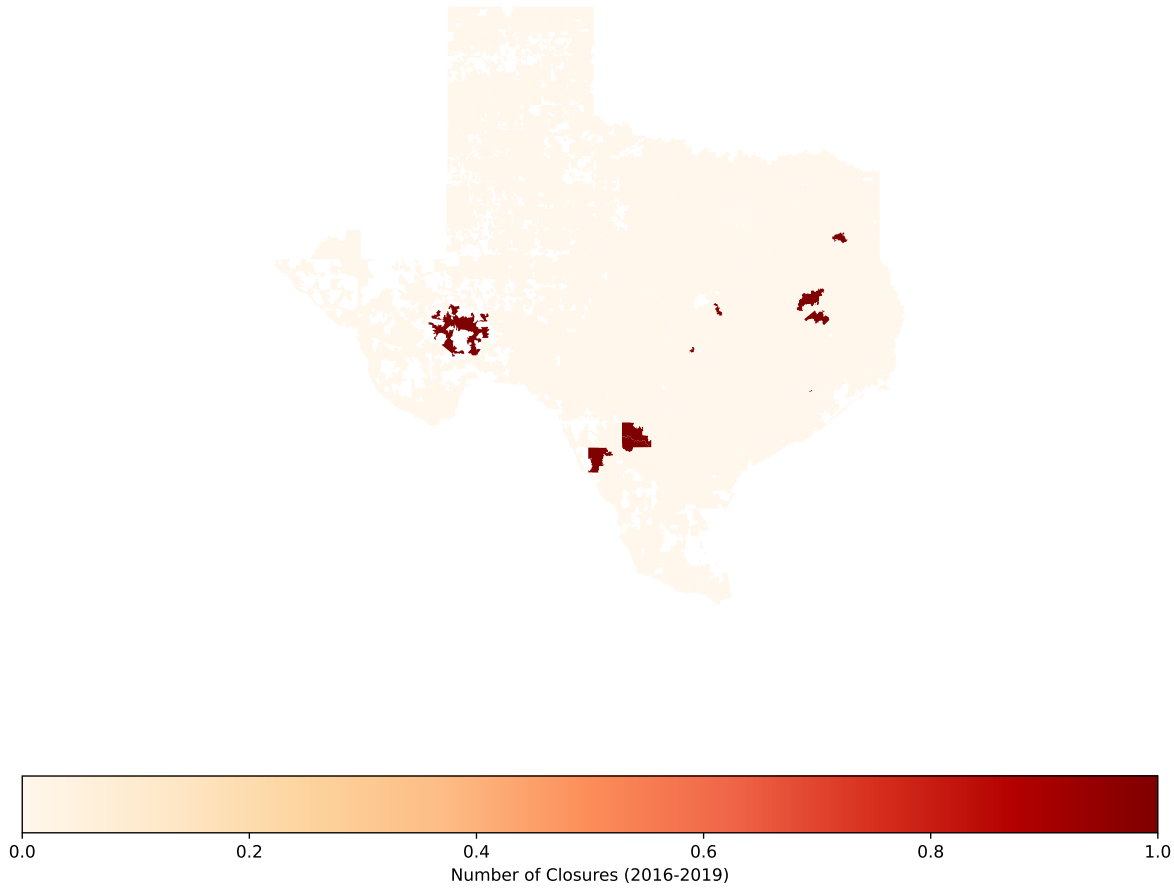
```
corrected_closures

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

# Filter the corrected closures dataset to include only ZIP codes in Texas
 ↪  and closures from 2016 to 2019
texas_closures = corrected_closures[
    (corrected_closures['ZIP_CD'].str.startswith('75') |  # Common Texas ZIP
 ↪  code prefixes
     corrected_closures['ZIP_CD'].str.startswith('76') |
     corrected_closures['ZIP_CD'].str.startswith('77') |
     corrected_closures['ZIP_CD'].str.startswith('78') |
     corrected_closures['ZIP_CD'].str.startswith('79') |
     corrected_closures['ZIP_CD'].str.startswith('733'))
]

# Count the number of closures by ZIP code
closures_by_zip =
 ↪  texas_closures.groupby('ZIP_CD').size().reset_index(name='closure_count')

import numpy as np
# Display the table of the number of closures by ZIP code
print("Table of the number of closures by ZIP code in Texas (2016-2019):")
closures_by_zip['ZIP_CD'] =
 ↪  closures_by_zip['ZIP_CD'].astype(float).astype(int).astype(str).str.zfill(5)
closures_by_zip
```

```
Table of the number of closures by ZIP code in Texas (2016-2019):
```

	ZIP_CD	closure_count
0	75662	1
1	75835	1
2	75862	1
3	76502	1
4	77035	1
5	78017	1
6	78061	1
7	78734	1
8	78834	1
9	79735	1

2.

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

# Ensure ZIP_CD in closures_by_zip and texas_zip_shapes are the same format
zip_tx['ZIP_CD'] = zip_tx['ZIP_CD'].astype(str).str.zfill(5)

# Merge the geographic data with the closure data
texas_closures_geo = zip_tx.merge(closures_by_zip, on='ZIP_CD',
 ↪  how='left').fillna(0)

# Plot the choropleth map
fig, ax = plt.subplots(1, 1, figsize=(12, 10))
texas_closures_geo.plot(column='closure_count', ax=ax, legend=True,
                        legend_kwds={'label': "Number of Closures
 ↪  (2016-2019)", 'orientation': "horizontal"},
                        cmap='OrRd',  # color map for the intensity of
 ↪  closures
                        missing_kwds={'color': 'lightgrey'})  # Color for
 ↪  areas with no data

# Customize the plot
ax.set_title('Hospital Closures in Texas by ZIP Code (2016-2019)')
ax.set_axis_off()  # Hide axis for a cleaner map

plt.show()

# Count the number of directly affected ZIP codes in Texas
num_affected_zip_codes = closures_by_zip['ZIP_CD'].nunique()
print(f"Number of directly affected ZIP codes in Texas:
 ↪  {num_affected_zip_codes}")
```




```
Number of directly affected ZIP codes in Texas: 10
```

3.

```
# Ensure both the directly affected ZIP codes and Texas ZIP shapes
 ↪  GeoDataFrames are in the same coordinate system
# Assuming `closures_by_zip` contains directly affected ZIP codes, and
 ↪  `texas_zip_shapes` contains all Texas ZIP shapes

# Filter the directly affected ZIP codes from `closures_by_zip`
affected_zips = closures_by_zip[closures_by_zip['closure_count'] >
 ↪  0]['ZIP_CD'].unique()

# Convert `closures_by_zip` to a GeoDataFrame with directly affected ZIP
 ↪  codes only
```

```
directly_affected_geo = zip_tx[zip_tx['ZIP_CD'].isin(affected_zips)]

# Convert the GeoDataFrame to a projected coordinate system (e.g., EPSG:2272)
 ↪  for accurate distance-based operations
directly_affected_geo = directly_affected_geo.to_crs(epsg=2272)
texas_zip_shapes = zip_tx.to_crs(epsg=2272)

# Create a 10-mile buffer around each directly affected ZIP code
# 1 mile is approximately 1609.34 meters
buffer_distance = 10 * 1609.34  # 10 miles in meters
directly_affected_geo['geometry'] =
 ↪  directly_affected_geo.buffer(buffer_distance)

# Perform a spatial join to find ZIP codes within the 10-mile buffer of
 ↪  directly affected ZIP codes
indirectly_affected_geo = gpd.sjoin(texas_zip_shapes, directly_affected_geo,
 ↪  how='inner', predicate='intersects')

# Count unique indirectly affected ZIP codes
indirectly_affected_zip_count =
 ↪  indirectly_affected_geo['ZIP_CD_left'].nunique() - len(affected_zips)

# Display results
print(f"Number of indirectly affected ZIP codes in Texas:
 ↪  {indirectly_affected_zip_count}")
```

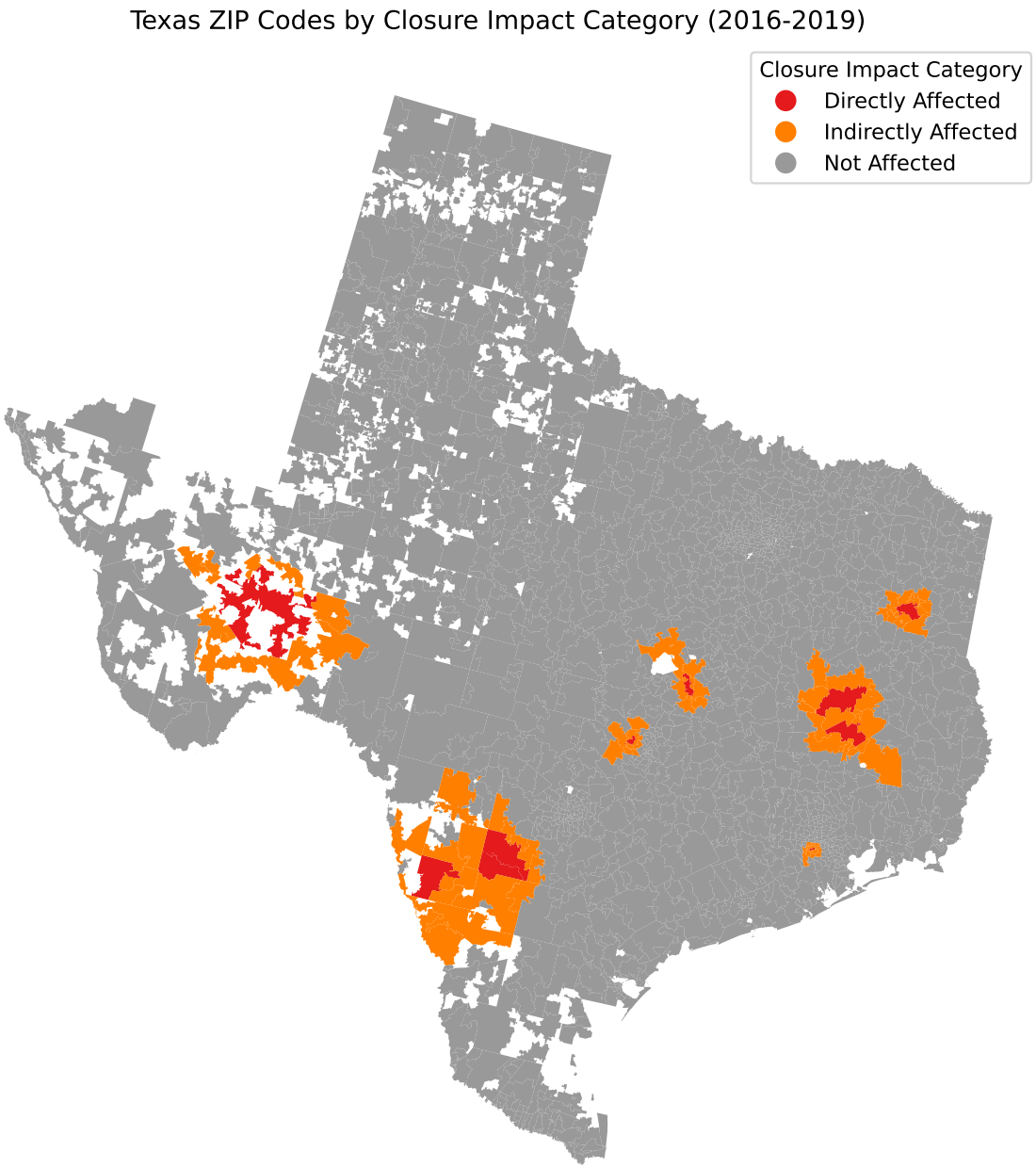
```
Number of indirectly affected ZIP codes in Texas: 95
```

4.

```
# Step 1: Prepare the Texas ZIP shapes and classify each ZIP code

# Step 1.1: Create a list of directly affected ZIP codes
directly_affected_zips = closures_by_zip[closures_by_zip['closure_count'] >
 ↪  0]['ZIP_CD'].unique()

# Step 1.2: Create the directly affected GeoDataFrame
directly_affected_geo = zip_tx[zip_tx['ZIP_CD'].isin(directly_affected_zips)]

# Convert to a projected CRS for distance calculation (e.g., EPSG:2272)
directly_affected_geo = directly_affected_geo.to_crs(epsg=2272)
texas_zip_shapes = texas_zip_shapes.to_crs(epsg=2272)
```

```
# Step 1.3: Create a 10-mile buffer around each directly affected ZIP code
buffer_distance = 10 * 1609.34  # 10 miles in meters
directly_affected_geo['geometry'] =
 ↪  directly_affected_geo.buffer(buffer_distance)

# Step 1.4: Perform a spatial join to identify indirectly affected ZIP codes
indirectly_affected_geo = gpd.sjoin(texas_zip_shapes, directly_affected_geo,
 ↪  how='inner', predicate='intersects')
indirectly_affected_zips = indirectly_affected_geo['ZIP_CD_left'].unique()

# Step 1.5: Classify ZIP codes
# Initialize a column to classify each ZIP code
texas_zip_shapes['closure_category'] = 'Not Affected'

# Mark directly affected ZIP codes
texas_zip_shapes.loc[texas_zip_shapes['ZIP_CD'].isin(directly_affected_zips),
 ↪  'closure_category'] = 'Directly Affected'

# Mark indirectly affected ZIP codes (exclude directly affected ones)
texas_zip_shapes.loc[
    (texas_zip_shapes['ZIP_CD'].isin(indirectly_affected_zips)) &
    (~texas_zip_shapes['ZIP_CD'].isin(directly_affected_zips)),
    'closure_category'
] = 'Indirectly Affected'

# Step 2: Plot the choropleth map

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

fig, ax = plt.subplots(1, 1, figsize=(12, 10))
texas_zip_shapes.plot(column='closure_category',
                      ax=ax,
                      legend=True,
                      categorical=True,
                      cmap='Set1',
                      legend_kwds={'title': "Closure Impact Category"})
```

```
# Customize the plot
ax.set_title('Texas ZIP Codes by Closure Impact Category (2016-2019)')
ax.set_axis_off()  # Hide axis for a cleaner map

plt.show()
```



Reflecting on the exercise (10 pts)

- **(Partner 1) The "first-pass" method we're using to address incorrectly identified closures in the data is imperfect. Can you think of some potential issues that could**

arise still and ways to do a better job at confirming hospital closures?

The limitations and issues about current approach are: Delayed reopenings. If a hospital closes but reopens later under a different name or management, it may still be incorrectly counted as a closure. This problem is common in facilities that undergo temporary closures for renovations or changes in ownership.

The closure detection focuses only on the termination status without considering geographical factors. In rural areas, where hospital coverage is sparse, a closure has a more significant impact on access than it might be in an urban setting with multiple nearby hospitals.

Data completeness and consistency. Hospital data may not be consistently reported or may have gaps due to issues in data collection or reporting delays. This could lead to misidentification if a hospital simply fails to report data in one year but resumes the next year.

Mergers and new names– Hospitals might seem closed due to changes in their certification number or mergers. They may still be active under a different identity and potentially leading to incorrect closure identification.

Potential Improvements: Data collection procedures can be improved to ensure that mergers, rebranding and any type of changes are properly reflected. Track name and address changes by developing a more robust matching system that accounts for slight changes in hospital names or addresses. Using similarity metrics can help identify hospitals with minor name changes that might indicate the same facility under new management.

Consider factors which can indicate the impact of closures

Geographical impact of closures is also important. For example, the closure of the only hospital within a 20-mile radius has a different implication than a closure in a well-served urban area. Therefore, it could be interesting to analyze the geographical impact on surrounding ZIP codes or regions.

- **(Partner 2) Consider the way we are identifying zip codes affected by closures. How well does this reflect changes in zip-code-level access to hospitals? Can you think of some ways to improve this measure?**

The method assumes that if a zip code is in proximity of a hospital has acces to the hospital without consodering accessibility and accessibility infrastructure. To access a facility, the ZIP codes within a 10-mile radius might be impacted by factors like road quality and transportation. Every ZIP code area does not have a similar impact by a closure. It could be interesting to conisder the demographic factors such as age, income, and health impacted by a hospital closures.

Some ZIP codes may have access to alternative healthcare facilities, such as urgent care centers and outpatient clinics, which can partially mitigate the impact of hospital closures. However, areas without these alternatives are more significantly affected.

Adding population and demographics data, weighting the impact of closures by the population density and demographics of each affected ZIP code could be more valuable. Identifying areas that lack alternative healthcare facilities within a reasonable distance could be important. Calculating averag, min and max travel times to the nearest operational hospital can also be important.