Problem-set-4

PS4: Due Sat Nov 2 at 5:00PM Central. Worth 100 points. We use (*) to indicate a problem that we think might be time consuming.

Style Points (10 pts)

Please refer to the minilesson on code style **here**.

Submission Steps (10 pts)

1. This problem set is a paired problem set.
2. Play paper, scissors, rock to determine who goes first. Call that person *Partner 1*.
    - Partner 1 (name and cnet ID): Xinyi Zhou (xzhou66)
    - Partner 2 (name and cnet ID): Wuzhen Han (hanwuzhen)
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: xinyi wuzhen**___** **___**
5. "I have uploaded the names of anyone else other than my partner and I worked with on the problem set **here**" xyz wuzhen (1 point)
6. Late coins used this pset: xinyi wuzhen**___** Late coins left after submission: 3**___**
7. Knit your `ps4.qmd` to an PDF file to make `ps4.pdf`,
    - The PDF should not be more than 25 pages. Use `head()` and re-size figures when appropriate.
8. (Partner 1): push `ps4.qmd` and `ps4.pdf` to your github repo.
9. (Partner 1): submit `ps4.pdf` via Gradescope. Add your partner on Gradescope.
10. (Partner 1): tag your submission in Gradescope

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

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

1. PRVDR_CTGRY_SBTYP_CD (Provider Category Subtype Code) PRVDR_CTGRY_CD (Provider Category Code) PRVDR_NUM (CMS Certification Number) FAC_NAME (Facility Name) ZIP_CD (Address: Zip Code) PGM_TRMNTN_CD (Terpmination Code) STATE_CD (State Code)

2. a.

   ::: {.cell execution_count=1} “‘ {.python .cell-code} import pandas as pd

   file_path_2016 = ‘~/Desktop/problem-set-4-xy-wz/pos2016.csv’ pos2016 = pd.read_csv(file_path_2016)

   short_term_hospitals_2016 = pos2016[(pos2016[‘PRVDR_CTGRY_CD’] == 1) & (pos2016[‘PRVDR_CTGRY
   == 1)]

   num_hospitals_2016 = short_term_hospitals_2016.shape[0] print(f”Number of short-term hospitals in 2016: {num_hospitals_2016}“) “‘

   ::: {.cell-output .cell-output-stdout} `Number of short-term hospitals in 2016: 7245` ::: :::

   This number appears reasonable given that the CMS dataset is known to include all hospitals eligible to bill Medicare.

b.The number of short-term hospitals reported in the CMS POS dataset for 2016 is 7,245. However, the Agency for Healthcare Research and Quality (AHRQ) reported 4,661 short-term acute care hospitals for the same year. This difference might be because the CMS dataset includes all hospitals eligible to bill Medicare, while the AHRQ data only includes hospitals that meet specific criteria, like operating for at least 180 days.

Cite from: https://www.ahrq.gov/sites/default/files/wysiwyg/data/SyH-DR-stat-brief-3-financial-measures.pdf

3.

```
import matplotlib.pyplot as plt
import altair as alt

pos2017 = pd.read_csv(
    '~/Desktop/problem-set-4-xy-wz/pos2017.csv', encoding='latin1')
pos2018 = pd.read_csv(
    '~/Desktop/problem-set-4-xy-wz/pos2018.csv', encoding='latin1')
pos2019 = pd.read_csv(
    '~/Desktop/problem-set-4-xy-wz/pos2019.csv', encoding='latin1')

short_term_hospitals_2017 = pos2017[(pos2017['PRVDR_CTGRY_CD'] == 1) & (
    pos2017['PRVDR_CTGRY_SBTYP_CD'] == 1)]
short_term_hospitals_2018 = pos2018[(pos2018['PRVDR_CTGRY_CD'] == 1) & (
    pos2018['PRVDR_CTGRY_SBTYP_CD'] == 1)]
short_term_hospitals_2019 = pos2019[(pos2019['PRVDR_CTGRY_CD'] == 1) & (
    pos2019['PRVDR_CTGRY_SBTYP_CD'] == 1)]

all_years = pd.concat([short_term_hospitals_2016, short_term_hospitals_2017,
 ↪  short_term_hospitals_2018, short_term_hospitals_2019], keys=[
```

```
                      '2016', '2017', '2018',
                       ↪  '2019']).reset_index(level=0).rename(columns={'level_0':
                       ↪  'Year'})

observations_per_year = all_years.groupby(
    'Year').size().reset_index(name='Number_of_Hospitals')

chart_observations = alt.Chart(observations_per_year).mark_bar().encode(
    alt.X('Year:O', title='Year'),
    alt.Y('Number_of_Hospitals:Q', title='Number of Hospitals'),
    tooltip=['Year', 'Number_of_Hospitals']
).properties(
    title='Number of Short-Term Hospitals by Year (2016-2019)',
    width=400,
    height=250
)

chart_observations.display()

num_hospitals_2017 = short_term_hospitals_2017.shape[0]
num_hospitals_2018 = short_term_hospitals_2018.shape[0]
num_hospitals_2019 = short_term_hospitals_2019.shape[0]

print(f"Number of short-term hospitals in 2017: {num_hospitals_2017}")
print(f"Number of short-term hospitals in 2018: {num_hospitals_2018}")
print(f"Number of short-term hospitals in 2019: {num_hospitals_2019}")
```

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

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

4. a.

```
unique_hospitals_per_year =
 ↪  all_years.groupby('Year')['PRVDR_NUM'].nunique().reset_index()
unique_hospitals_per_year.columns = ['Year', 'Unique_Hospitals']

chart_unique_per_year = alt.Chart(unique_hospitals_per_year).mark_bar().encode(
    alt.X('Year:O', title='Year'),
    alt.Y('Unique_Hospitals:Q', title='Number of Unique Hospitals'),
    tooltip=['Year', 'Unique_Hospitals']
).properties(
    title='Number of Unique Short-Term Hospitals by Year (2016-2019)',
    width=400,
    height=250
)

chart_unique_per_year.display()
```

`alt.Chart(...)`

b.The comparison between the total number of hospitals and the number of unique hospitals reveals that the data is highly consistent across the years. In both plots, the number of hospitals remains almost unchanged from 2016 to 2019, with the total number of hospitals and unique hospitals being very close to each other. This suggests that most hospitals were consistently present throughout the time period, indicating minimal new entries or exits from the dataset. The stability in both total and unique counts highlights that the healthcare system had little fluctuation in terms of hospital availability, pointing to a continuous and stable system over these years.

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

1.

```
file_paths = ["pos2016.csv", "pos2017.csv", "pos2018.csv", "pos2019.csv"]
dataframes = [pd.read_csv(file_path, encoding="latin1") for file_path in file_paths]

active_code = 0
short_term_category_code = 1
short_term_subtype_code = 1

for i in range(len(dataframes)):
    dataframes[i]["PRVDR_NUM"] = dataframes[i]["PRVDR_NUM"].astype(str)

active_hospitals_2016 = dataframes[0][
    (dataframes[0]["PGM_TRMNTN_CD"] == active_code)
    & (dataframes[0]["PRVDR_CTGRY_CD"] == short_term_category_code)
    & (dataframes[0]["PRVDR_CTGRY_SBTYP_CD"] == short_term_subtype_code)
]

merged_df = active_hospitals_2016.merge(
    dataframes[1][["PRVDR_NUM", "PGM_TRMNTN_CD"]].rename(columns={"PGM_TRMNTN_CD":
 ↪  "PGM_TRMNTN_CD_2017"}),
    on="PRVDR_NUM", how="left"
).merge(
    dataframes[2][["PRVDR_NUM", "PGM_TRMNTN_CD"]].rename(columns={"PGM_TRMNTN_CD":
 ↪  "PGM_TRMNTN_CD_2018"}),
    on="PRVDR_NUM", how="left"
).merge(
    dataframes[3][["PRVDR_NUM", "PGM_TRMNTN_CD"]].rename(columns={"PGM_TRMNTN_CD":
 ↪  "PGM_TRMNTN_CD_2019"}),
    on="PRVDR_NUM", how="left"
)

suspected_closures = []

for _, row in merged_df.iterrows():
    closure_year = None
    if pd.isna(row["PGM_TRMNTN_CD_2017"]) or row["PGM_TRMNTN_CD_2017"] != active_code:
```

```
        closure_year = 2017
    elif pd.isna(row["PGM_TRMNTN_CD_2018"]) or row["PGM_TRMNTN_CD_2018"] != active_code:
        closure_year = 2018
    elif pd.isna(row["PGM_TRMNTN_CD_2019"]) or row["PGM_TRMNTN_CD_2019"] != active_code:
        closure_year = 2019

    if closure_year:
        suspected_closures.append(
            {
                "Facility Name": row["FAC_NAME"] if pd.notna(row["FAC_NAME"]) else
 ↪  "Unknown",
                "ZIP Code": row["ZIP_CD"] if pd.notna(row["ZIP_CD"]) else "Unknown",
                "Year of Suspected Closure": closure_year,
                "Type of Reason": 1,
                "PRVDR_NUM": row["PRVDR_NUM"],
            }
        )

suspected_closures_df = pd.DataFrame(
    suspected_closures,
    columns=["Facility Name", "ZIP Code", "Year of Suspected Closure", "PRVDR_NUM", "Type
 ↪  of Reason"]
)

num_suspected_closures_2019 = suspected_closures_df.shape[0]
print(
    f"Number of short-term hospitals suspected to have closed by 2019:
 ↪  {num_suspected_closures_2019}"
)
```

```
Number of short-term hospitals suspected to have closed by 2019: 174
```

2.

```
sorted_suspected_closures_df = suspected_closures_df.sort_values(
    by="Facility Name"
).head(10)
print(
    sorted_suspected_closures_df[
        ["Facility Name", "Year of Suspected Closure", "ZIP Code", "PRVDR_NUM"]
    ]
)
```

```
                                  Facility Name  Year of Suspected Closure  \
4                          ABRAZO MARYVALE CAMPUS                       2017
10       ADVENTIST MEDICAL CENTER - CENTRAL VALLEY                       2017
97                         AFFINITY MEDICAL CENTER                       2018
80   ALBANY MEDICAL CENTER / SOUTH CLINICAL CAMPUS                       2017
140       ALLEGIANCE SPECIALTY HOSPITAL OF KILGORE                       2017
62                         ALLIANCE LAIRD HOSPITAL                       2019
```

```
101                        ALLIANCEHEALTH DEACONESS                        2019
26                      ANNE BATES LEACH EYE HOSPITAL                      2019
21           ARKANSAS VALLEY REGIONAL MEDICAL CENTER                       2017
69               BANNER CHURCHILL COMMUNITY HOSPITAL                       2017

     ZIP Code PRVDR_NUM
4     85031.0    030001
10    93230.0    050196
97    44646.0    360151
80    12208.0    330189
140   75662.0    450488
62    39365.0    250159
101   73112.0    370032
26    33136.0    100240
21    81050.0    060036
69    89406.0    290006
```

3. a.

```
zip_counts = pd.DataFrame()
for year, df in enumerate(dataframes, start=2016):
    active_counts = (
        df[
            (df["PRVDR_CTGRY_CD"] == short_term_category_code)
            & (df["PRVDR_CTGRY_SBTYP_CD"] == short_term_subtype_code)
            & (df["PGM_TRMNTN_CD"] == active_code)
        ]
        .groupby("ZIP_CD")["PRVDR_NUM"]
        .count()
        .reset_index()
    )
    active_counts.columns = ["ZIP_CD", f"Active_Count_{year}"]

    if zip_counts.empty:
        zip_counts = active_counts
    else:
        zip_counts = zip_counts.merge(active_counts, on="ZIP_CD", how="outer").fillna(0)

filtered_suspected_closures = []
merger_candidates = []

for _, row in suspected_closures_df.iterrows():
    zip_code = row["ZIP Code"]
    closure_year = row["Year of Suspected Closure"]

    if closure_year < 2019:
        current_count = zip_counts.loc[
            zip_counts["ZIP_CD"] == zip_code, f"Active_Count_{closure_year}"
        ].values[0]
        next_year_count = zip_counts.loc[
            zip_counts["ZIP_CD"] == zip_code, f"Active_Count_{closure_year + 1}"
```

```
        ].values[0]

        if next_year_count >= current_count:
            merger_candidates.append(row)
        else:
            filtered_suspected_closures.append(row)
    else:
        filtered_suspected_closures.append(row)

filtered_suspected_closures_df = pd.DataFrame(filtered_suspected_closures)
merger_candidates_df = pd.DataFrame(merger_candidates)

num_merger_candidates = merger_candidates_df.shape[0]
print(
    f"Number of hospitals potentially involved in a merger/acquisition:
     ↪ {num_merger_candidates}"
)
```

```
Number of hospitals potentially involved in a merger/acquisition: 97
```

b.

```
num_filtered_suspected_closures = filtered_suspected_closures_df.shape[0]
print(
    f"Number of hospitals left after correcting for potential mergers/acquisitions:
     ↪ {num_filtered_suspected_closures}"
)
```

```
Number of hospitals left after correcting for potential mergers/acquisitions: 77
```

c.

```
sorted_filtered_closures = filtered_suspected_closures_df.sort_values(
    by="Facility Name"
).head(10)

print(
    sorted_filtered_closures[["Facility Name", "ZIP Code", "Year of Suspected Closure"]]
)
```

```
                                      Facility Name  ZIP Code  \
62                            ALLIANCE LAIRD HOSPITAL   39365.0
101                          ALLIANCEHEALTH DEACONESS   73112.0
26                     ANNE BATES LEACH EYE HOSPITAL   33136.0
115                    BARIX CLINICS OF PENNSYLVANIA   19047.0
171                  BAYLOR EMERGENCY MEDICAL CENTER   75087.0
166  BAYLOR SCOTT & WHITE EMERGENCY MEDICAL CENTER ...   78613.0
98                         BELMONT COMMUNITY HOSPITAL   43906.0
67                             BIG SKY MEDICAL CENTER   59716.0
65              BLACK RIVER COMMUNITY MEDICAL CENTER   63901.0
```

```
142                      CARE REGIONAL MEDICAL CENTER   78336.0

     Year of Suspected Closure
62                        2019
101                       2019
26                        2019
115                       2019
171                       2019
166                       2019
98                        2019
67                        2019
65                        2019
142                       2019
```

Download Census zip code shapefile (10 pt)

1. ## a. The .dbf file is a database file that stores attribute data for each geographical feature, similar to a spreadsheet, which allows you to access information like names, zip codes, or other attributes associated with each shape. The .prj file contains the projection information, which defines the coordinate system used by the shapefile. This is crucial for aligning the data with other geographic datasets properly. The .shp file is the main shape file and stores the actual geometry, including the coordinates that define the spatial features like points, lines, and polygons. The .shx file is an index file that provides a positional index of the geometries in the .shp file, enabling faster data access and locating features within the dataset. Lastly, the .xml file contains metadata, which offers descriptions and additional information about the shapefile, such as field definitions and projection details, helping to understand the dataset more effectively.

   b.

   ::: {.cell execution_count=9} “‘ {.python .cell-code} import os

   base_path = ‘/Users/cynthia/Desktop/gz_2010_us_860_00_500k’

   file_names = [‘gz_2010_us_860_00_500k.dbf’, ‘gz_2010_us_860_00_500k.prj’, ‘gz_2010_us_860_00_500k.s
   ‘gz_2010_us_860_00_500k.shx’, ‘gz_2010_us_860_00_500k.xml’]

   for file in file_names: file_path = os.path.join(base_path, file)

   ```
   if os.path.exists(file_path):
       size_bytes = os.path.getsize(file_path)
       size_mb = size_bytes / (1024 * 1024)
       print(f"{file}: {size_mb:.2f} MB")
   else:
       print(f"File not found: {file_path}")
   ```

   ::: {.cell-output .cell-output-stdout}

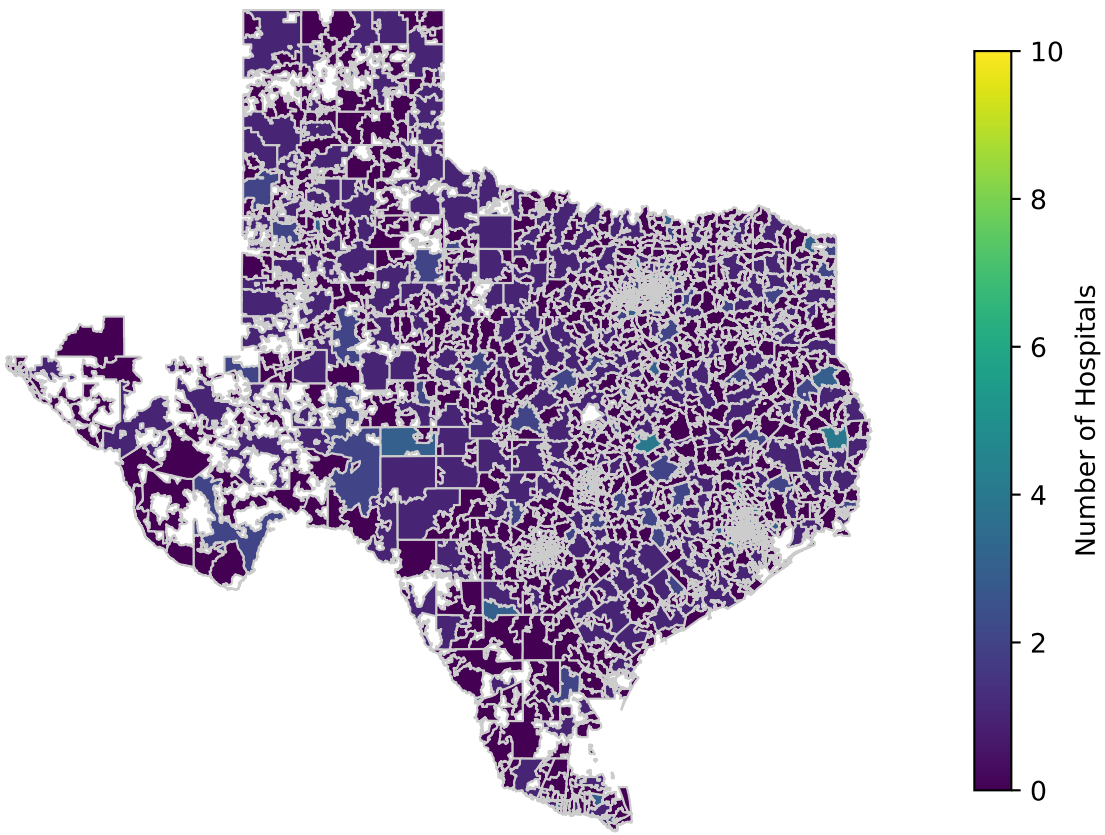
   gz_2010_us_860_00_500k.dbf: 6.13 MB gz_2010_us_860_00_500k.prj: 0.00 MB gz_2010_us_860_00_500k.
   798.74 MB gz_2010_us_860_00_500k.shx: 0.25 MB gz_2010_us_860_00_500k.xml: 0.01 MB “‘ :::
   :::

2.

```
import geopandas as gpd

pathshp = os.path.join(base_path, "gz_2010_us_860_00_500k.shp")
pathshp = '~/Desktop/gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.shp'
df_shp = gpd.read_file(pathshp)

df_texas = df_shp[df_shp["ZCTA5"].str.startswith(("75", "76", "77", "78", "79"))]
df_texas["ZCTA5"] = df_texas["ZCTA5"].astype(str)

hospital_2016 = pos2016[
    (   pos2016["PRVDR_CTGRY_CD"] == 1) & (pos2016["PRVDR_CTGRY_SBTYP_CD"] == 1)
]

hospital_2016["ZIP_CD"] = (
    hospital_2016["ZIP_CD"]
    .astype(str)
    .str.replace(r"\.0$", "", regex=True)
    .str.zfill(5)
)

hospital_zip = hospital_2016.groupby("ZIP_CD").size().reset_index(name="hospital_count")

map_texas = df_texas.merge(hospital_zip, left_on="ZCTA5", right_on="ZIP_CD", how="left")
map_texas["hospital_count"] = map_texas["hospital_count"].fillna(0)

fig, ax = plt.subplots(1, 1, figsize=(8, 10))
map_texas.plot(
    column="hospital_count",
    cmap="viridis",
    linewidth=0.8,
    ax=ax,
    edgecolor="0.8",
    legend=True,
    legend_kwds={
        'shrink': 0.5,
        'label': "Number of Hospitals"
    },
)
ax.set_title("Number of Hospitals per ZIP Code in Texas (2016)", fontsize=10)
plt.axis("off")
plt.show()
```

```
/opt/anaconda3/lib/python3.11/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-a-co

/var/folders/6t/pjbppdks0vb8q5hf3h8p6mx00000gn/T/ipykernel_27552/1430434340.py:14:
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-a-co
```



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

1.

```
base_path = '~/Desktop/gz_2010_us_860_00_500k'
shapefile_path = os.path.join(base_path, "gz_2010_us_860_00_500k.shp")

gdf = gpd.read_file(shapefile_path)
```

```
zips_all_centroids = gdf.copy()
zips_all_centroids["geometry"] = gdf.centroid

print("GeoDataFrame dimensionality:", zips_all_centroids.shape)
print("GeoDataFrame Column information:")
print(zips_all_centroids.columns)

print(zips_all_centroids.head())
```

```
/var/folders/6t/pjbppdks0vb8q5hf3h8p6mx00000gn/T/ipykernel_27552/4045363704.py:7:
UserWarning:

Geometry is in a geographic CRS. Results from 'centroid' are likely incorrect. Use
'GeoSeries.to_crs()' to re-project geometries to a projected CRS before this operation.


GeoDataFrame dimensionality: (33120, 6)
GeoDataFrame Column information:
Index(['GEO_ID', 'ZCTA5', 'NAME', 'LSAD', 'CENSUSAREA', 'geometry'], dtype='object')
           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.

```
texas_prefixes = ("75", "76", "77", "78", "79")

zips_texas_centroids = zips_all_centroids[
    zips_all_centroids["ZCTA5"].str.startswith(texas_prefixes)
]
num_unique_texas_zips = zips_texas_centroids["ZCTA5"].nunique()
print(f"Number of unique zip codes in Texas: {num_unique_texas_zips}")

bordering_state_prefixes = (
    "70",
    "71",
    "72",
    "73",
    "74",
    "75",
    "76",
    "77",
    "78",
    "79",
    "87",
    "88",
)
```

```
zips_texas_borderstates_centroids = zips_all_centroids[
    zips_all_centroids["ZCTA5"].str.startswith(bordering_state_prefixes)
]
num_unique_borderstate_zips = zips_texas_borderstates_centroids["ZCTA5"].nunique()
print(
    f"Number of unique zip codes in Texas and bordering states:
     ↪  {num_unique_borderstate_zips}"
)
```

```
Number of unique zip codes in Texas: 1935
Number of unique zip codes in Texas and bordering states: 4057
```

3.

```
short_term_category_code = 1
short_term_subtype_code = 1
active_code = 0

active_hospitals_2016 = pos2016[
    (pos2016['PRVDR_CTGRY_CD'] == short_term_category_code) &
    (pos2016['PRVDR_CTGRY_SBTYP_CD'] == short_term_subtype_code) &
    (pos2016['PGM_TRMNTN_CD'] == active_code)
]

active_hospitals_2016 = active_hospitals_2016.copy()

active_hospitals_2016['ZIP_CD'] =
 ↪  active_hospitals_2016['ZIP_CD'].fillna(0).astype(int).astype(str).str.zfill(5)

hospital_counts = active_hospitals_2016['ZIP_CD'].value_counts().reset_index()
hospital_counts.columns = ['ZCTA5', 'hospital_count']

zips_with_hospitals = hospital_counts[hospital_counts['hospital_count'] >= 1]

zips_withhospital_centroids = zips_texas_borderstates_centroids.merge(
    zips_with_hospitals, on='ZCTA5', how='inner'
)

print(zips_withhospital_centroids.head())
print(f"Number of zip codes with at least 1 hospital in 2016:
 ↪  {zips_withhospital_centroids.shape[0]}")
```

```
           GEO_ID  ZCTA5   NAME   LSAD  CENSUSAREA  \
0  8600000US70043  70043  70043  ZCTA5       7.775
1  8600000US70127  70127  70127  ZCTA5       7.095
2  8600000US70301  70301  70301  ZCTA5     288.050
3  8600000US70360  70360  70360  ZCTA5      64.325
4  8600000US70403  70403  70403  ZCTA5      42.960
```

```
                     geometry  hospital_count
0  POINT (-89.96276 29.94804)               1
1  POINT (-89.97675 30.02501)               1
2   POINT (-90.74089 29.8141)               1
3  POINT (-90.81028 29.58819)               2
4  POINT (-90.48388 30.48002)               2
Number of zip codes with at least 1 hospital in 2016: 448
```

4. a.

```python
import time

zips_texas_centroids = zips_texas_centroids.to_crs(epsg=3857)
zips_withhospital_centroids = zips_withhospital_centroids.to_crs(epsg=3857)

test_zips = zips_texas_centroids.head(10)

start_time = time.time()

distances = []

for _, test_zip in test_zips.iterrows():
    test_geometry = test_zip["geometry"]

    min_distance = zips_withhospital_centroids["geometry"].distance(test_geometry).min()

    distances.append(
        {"ZCTA5": test_zip["ZCTA5"], "Nearest_Hospital_Distance": min_distance}
    )

end_time = time.time()

execution_time = end_time - start_time
print(f"Time taken for 10 zip codes: {execution_time:.2f} seconds")

distance_results = pd.DataFrame(distances)
print(distance_results)

total_zip_codes = len(zips_texas_centroids)
estimated_time = (execution_time / 10) * total_zip_codes
print(f"Estimated time for all zip codes in Texas: {estimated_time / 60:.2f} minutes")
```

```
Time taken for 10 zip codes: 0.00 seconds
   ZCTA5  Nearest_Hospital_Distance
0  78624                  0.000000
1  78626              21443.710461
2  78628              14228.011802
3  78631              42374.908087
4  78632              28125.820177
5  78633              27182.105800
6  78634              13251.123905
```

```
7  78635               36496.178533
8  78636               41783.476410
9  78638               22509.393995
Estimated time for all zip codes in Texas: 0.01 minutes
```

b.

```python
start_time = time.time()

all_distances = []

for _, texas_zip in zips_texas_centroids.iterrows():
    texas_geometry = texas_zip["geometry"]

    min_distance = (
        zips_withhospital_centroids["geometry"].distance(texas_geometry).min()
    )

    all_distances.append(
        {"ZCTA5": texas_zip["ZCTA5"], "Nearest_Hospital_Distance": min_distance}
    )

end_time = time.time()

execution_time = end_time - start_time
print(f"Time taken for all zip codes in Texas: {execution_time:.2f} seconds")

all_distance_results = pd.DataFrame(all_distances)
print(all_distance_results.head())

all_distance_results.to_csv("texas_zipcode_hospital_distances.csv", index=False)

estimated_time = (0.08 / 10) * len(zips_texas_centroids)
print(f"Estimated time for all zip codes in Texas: {estimated_time:.2f} seconds")

if execution_time <= estimated_time:
    print("Actual execution time is less than or close to the estimated time.")
else:
    print("Actual execution time is greater than the estimated time.")
```

```
Time taken for all zip codes in Texas: 0.31 seconds
   ZCTA5  Nearest_Hospital_Distance
0  78624                  0.000000
1  78626              21443.710461
2  78628              14228.011802
3  78631              42374.908087
4  78632              28125.820177
Estimated time for all zip codes in Texas: 15.48 seconds
Actual execution time is less than or close to the estimated time.
```

5. a. ### Ans: The unit in the .prj file is Degree.

b.

```
conversion_factor = 0.000621371

all_distance_results["Nearest_Hospital_Distance_Miles"] = (
    all_distance_results["Nearest_Hospital_Distance"] * conversion_factor
)

average_distance_miles = all_distance_results["Nearest_Hospital_Distance_Miles"].mean()
print(
    f"Average distance to the nearest hospital in Texas (in miles):
     ↪  {average_distance_miles:.2f} miles"
)

if average_distance_miles < 30:
    print(
        "This value makes sense, considering the distribution of hospitals and rural
         ↪  areas in Texas."
    )
else:
    print(
        "This value seems unusually high, possibly indicating a lack of hospitals in some
         ↪  areas."
    )
```

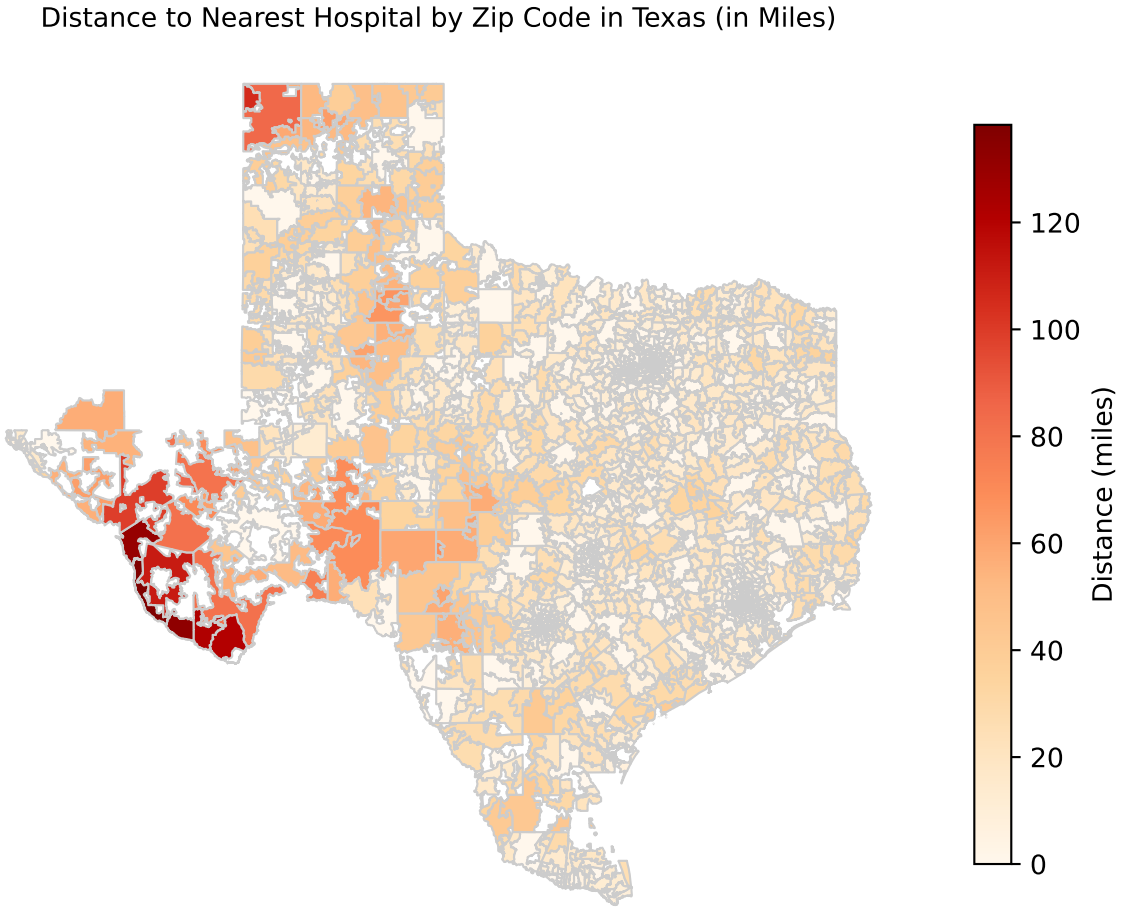
```
Average distance to the nearest hospital in Texas (in miles): 15.77 miles
This value makes sense, considering the distribution of hospitals and rural areas in
Texas.
```

c.

```
base_path = "/Users/cynthia/Desktop/gz_2010_us_860_00_500k"
pathshp = os.path.join(base_path, "gz_2010_us_860_00_500k.shp")
df_shp = gpd.read_file(pathshp)

df_texas = df_shp[df_shp["ZCTA5"].str.startswith(("75", "76", "77", "78", "79"))]
df_texas["ZCTA5"] = df_texas["ZCTA5"].astype(str)

df_texas_distance = df_texas.merge(
    all_distance_results[["ZCTA5", "Nearest_Hospital_Distance_Miles"]],
    on="ZCTA5",
    how="left"
)

df_texas_distance["Nearest_Hospital_Distance_Miles"] =
 ↪  df_texas_distance["Nearest_Hospital_Distance_Miles"].fillna(0)

fig, ax = plt.subplots(1, 1, figsize=(8, 10))
df_texas_distance.plot(
    column="Nearest_Hospital_Distance_Miles",
```

```
    cmap="OrRd",
    linewidth=0.8,
    ax=ax,
    edgecolor="0.8",
    legend=True,
    legend_kwds={
        'shrink': 0.5,
        'label': "Distance (miles)"
    },
)

ax.set_title("Distance to Nearest Hospital by Zip Code in Texas (in Miles)", fontsize=10)
plt.axis("off")

plt.show()
```

```
/opt/anaconda3/lib/python3.11/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-a-co
```



Effects of closures on access in Texas (15 pts)

1.

```
columns_to_drop = [
    col for col in filtered_suspected_closures_df.columns if 'STATE_CD' in col]
filtered_suspected_closures_df = filtered_suspected_closures_df.drop(
    columns=columns_to_drop, errors='ignore')

filtered_suspected_closures_df = filtered_suspected_closures_df.merge(
    dataframes[0][['PRVDR_NUM', 'STATE_CD']],
    on='PRVDR_NUM',
    how='left'
)

state_column = 'STATE_CD'
texas_closures_df =
 ↪  filtered_suspected_closures_df[filtered_suspected_closures_df[state_column] ==
 ↪  'TX'].copy(
)

texas_closures_df['ZIP Code'] = texas_closures_df['ZIP Code'].astype(
```

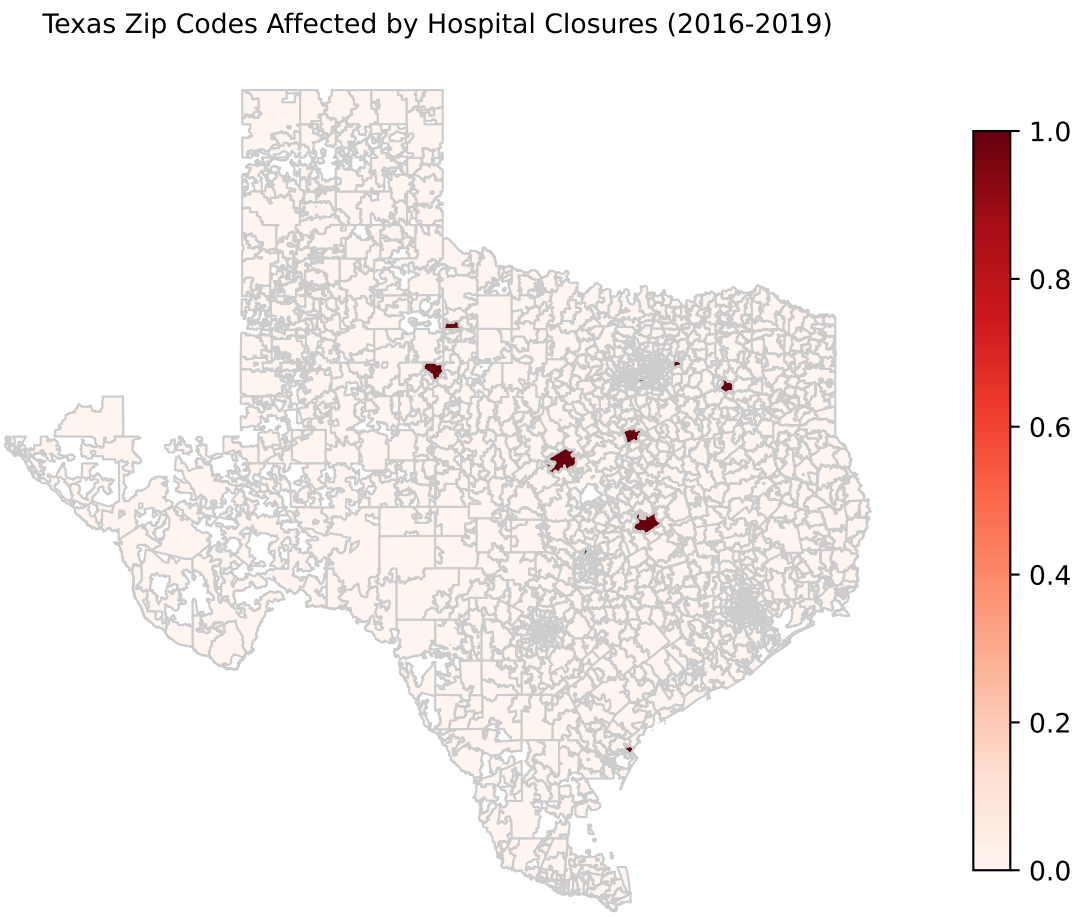
```
    str).str.zfill(5)

closures_per_zip = texas_closures_df.groupby(
    'ZIP Code').size().reset_index(name='closure_count')
closures_per_zip = closures_per_zip.rename(columns={"ZIP Code": "ZIP_CD"})

closures_per_zip['ZIP_CD'] = closures_per_zip['ZIP_CD'].astype(
    str).str.replace(r'\.0$', '', regex=True).str.zfill(5)

closures_per_zip = closures_per_zip.sort_values(by='ZIP_CD')

closures_per_zip.reset_index(inplace=True)
closures_per_zip.rename(columns={"index": " "}, inplace=True)

print(closures_per_zip.to_string(index=False))
```

	ZIP_CD	closure_count
0	75051	1
1	75087	1
2	75140	1
3	75235	1
4	75390	1
5	76520	1
6	76531	1
7	76645	1
8	77065	1
9	78336	1
10	78613	1
11	79520	1
12	79529	1
13	79902	1

2.

```
df_texas = df_shp[df_shp["ZCTA5"].str.startswith(
    ("75", "76", "77", "78", "79"))].copy()

df_texas['ZCTA5'] = df_texas['ZCTA5'].astype(str).str.zfill(5)

closures_per_zip['ZIP_CD'] = closures_per_zip['ZIP_CD'].astype(
    str).str.zfill(5)

map_closures_texas = df_texas.merge(
    closures_per_zip, left_on='ZCTA5', right_on='ZIP_CD', how='left'
)
map_closures_texas['closure_count'] = map_closures_texas['closure_count'].fillna(
    0)

fig, ax = plt.subplots(1, 1, figsize=(8, 10))
map_closures_texas.plot(
```

```
    column='closure_count',
    cmap='Reds',
    legend=True,
    ax=ax,
    linewidth=0.8,
    edgecolor='0.8',
    legend_kwds={
        'shrink': 0.5,
    }
)
ax.set_axis_off()
plt.title('Texas Zip Codes Affected by Hospital Closures (2016-2019)', fontsize=10)
plt.show()


directly_zip_count = closures_per_zip.shape[0]
print(
    f"The number of directly affected zip codes in Texas is {directly_zip_count}.")
```




```
The number of directly affected zip codes in Texas is 14.
```

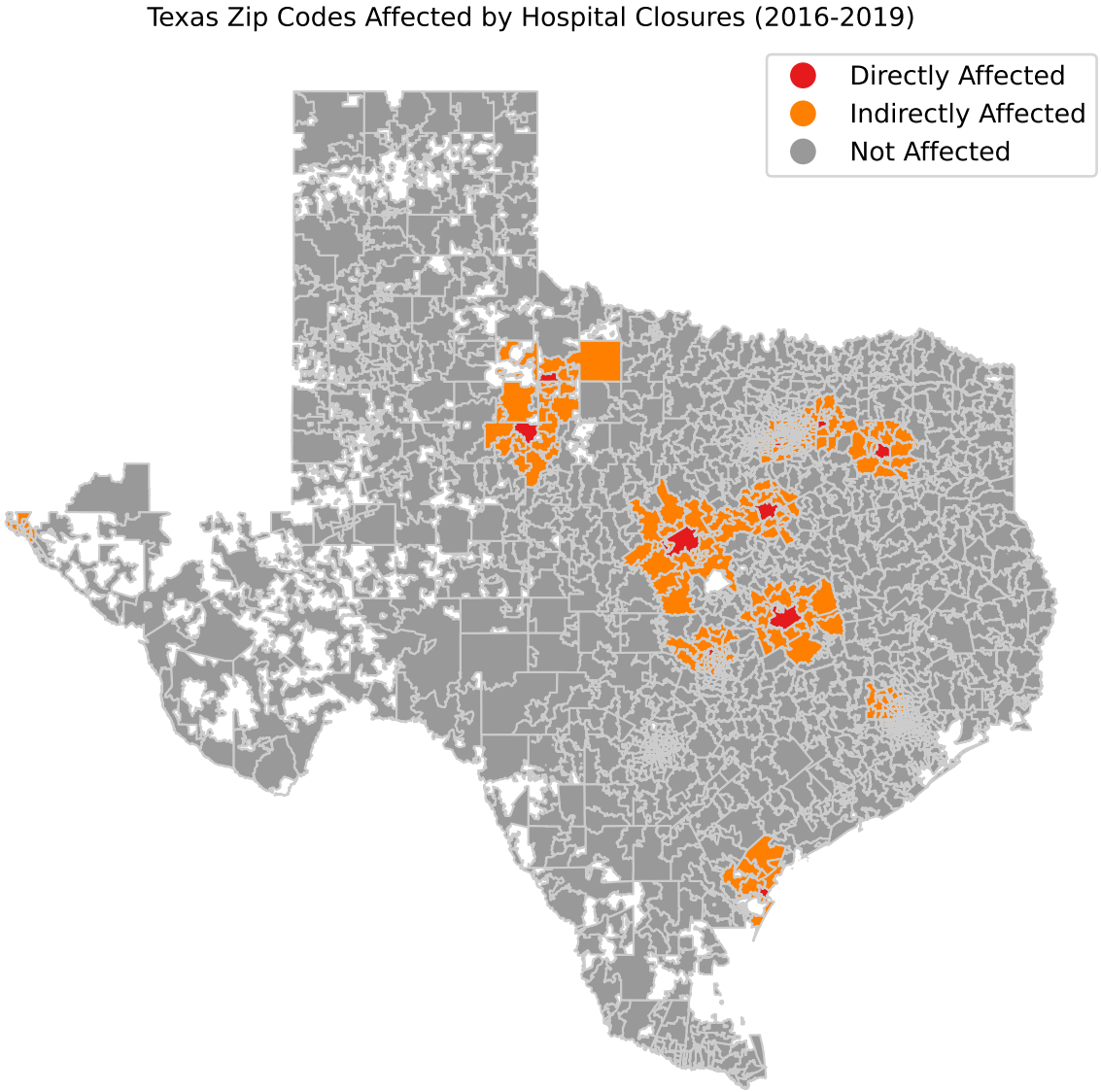
3.

```
direct_closures_texas_gdf = df_texas.merge(
    closures_per_zip, left_on='ZCTA5', right_on='ZIP_CD', how='inner'
)

direct_closures_texas_gdf = direct_closures_texas_gdf.to_crs(epsg=3395)

closures_buffer = direct_closures_texas_gdf.copy()
closures_buffer['geometry'] = closures_buffer.geometry.buffer(16093)

closures_buffer = closures_buffer.to_crs(df_texas.crs)

indirectly_zip = gpd.sjoin(df_texas, closures_buffer,
                           how="inner", predicate="intersects")

indirectly_zip = indirectly_zip[~indirectly_zip['ZCTA5_left'].isin(
    direct_closures_texas_gdf.to_crs(df_texas.crs)['ZCTA5'])]

indirectly_zip_count = indirectly_zip['ZCTA5_left'].nunique()
print(
    f"The number of indirectly affected zip codes in Texas is {indirectly_zip_count}.")
```

```
The number of indirectly affected zip codes in Texas is 311.
```

4.

```
df_texas['category'] = 'Not Affected'
df_texas.loc[df_texas['ZCTA5'].isin(direct_closures_texas_gdf.to_crs(
    df_texas.crs)['ZCTA5']), 'category'] = 'Directly Affected'
df_texas.loc[df_texas['ZCTA5'].isin(
    indirectly_zip['ZCTA5_left']), 'category'] = 'Indirectly Affected'

fig, ax = plt.subplots(1, 1, figsize=(8, 10))
df_texas.plot(column='category', ax=ax, legend=True,
              cmap='Set1', linewidth=0.8, edgecolor='0.8')
plt.title('Texas Zip Codes Affected by Hospital Closures (2016-2019)', fontsize=10)
plt.axis('off')
plt.show()
```


Texas Zip Codes Affected by Hospital Closures (2016-2019)
Directly Affected
Indirectly Affected
Not Affected

Reflecting on the exercise (10 pts)

Partner 1: The initial method for identifying hospital closures has several problems that could lead to mistakes. One issue is that mergers or acquisitions might be wrongly marked as closures. Hospitals that merge or are bought may get new certification numbers, which could make it seem like they closed. To improve accuracy, we could better match hospital names and addresses to identify these cases. Another issue is data delays or inconsistencies so hospitals might temporarily disappear from the data without actually closing. Using additional data sources, like state databases, could help verify if a hospital is really closed. Relying on just one field, such as the "Termination Code," can also be risky, because errors in that field could lead to wrong conclusions. It would be better to also consider other information, like staffing levels or available services. Some hospitals may also relocate rather than close, which could be misinterpreted as a closure. Using location data to track these changes could help. Finally, verifying closures using official sources would reduce false positives and make the analysis more reliable.

Partner 2: The current way of identifying zip codes affected by hospital closures has some limitations. Just using distance to decide if a zip code is directly or indirectly affected may not fully capture changes in access to healthcare. For example, two zip codes within 10 miles of a closed hospital might be affected differently depending on factors like transportation options, population density, and nearby healthcare facilities. Using only distance doesn't tell the whole story. There are some ways to improve this measure. First, include transportation factors like road access and public transit, because these influence how easily people can get to hospitals. Second, consider the capacity of nearby hospitals to handle more patients, as closures might lead to overcrowding. Third, include population density and demographics to understand which areas might be impacted the most. Also, consider travel time cause it can provide a better sense of accessibility, especially in rural areas. Finally, take into account other healthcare facilities like clinic centers to get a more complete picture of healthcare availability after a hospital closure.