DAP II 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): Sara Van Valkenburgh, vanvals
    - Partner 2 (name and cnet ID): Jennifer Edouard, jkedouard
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: **sv** **je**
5. "I have uploaded the names of anyone else other than my partner and I worked with on the problem set **here**" (1 point)
6. Late coins used this pset: **\1\4** Late coins left after submission: **\2\4**
7. Knit your `ps4.qmd` to an PDF file to make `ps4.pdf`,
    - The PDF should not be more than 25 pages. Use `head()` and re-size figures when appropriate.
8. (Partner 1): push `ps4.qmd` and `ps4.pdf` to your github repo.
9. (Partner 1): submit `ps4.pdf` via Gradescope. Add your partner on Gradescope.
10. (Partner 1): tag your submission in Gradescope

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

```
import pandas as pd
import os
import altair as alt
import geopandas as gpd
import matplotlib.pyplot as plt
import time

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

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

1. Variables pulled:

- PRVDR_CTGRY_SBTYP_CD - Identifies the subtype of the provider, within the primary category
- PRVDR_CTGRY_CD - Identifies the type of provider participating in the Medicare/Medicaid program
- FAC_NAME - Facility name
- PRVDR_NUM - Unique CMS certification number
- PGM_TRMNTN_CD - Indicates the current termination status for the provider (termination code)
- ZIP_CD - Zip code

2.

```
# read in the data for Q4 2016
filepath = "data/pos2016.csv"
data_2016 = pd.read_csv(filepath)

# Adding the year earlier so that python recognizes it when we combine the dataframes later
data_2016['year'] = 2016
```

```
# Make sure the codes are in the right format -- 01
data_2016['PRVDR_CTGRY_CD'] = data_2016['PRVDR_CTGRY_CD'].apply(
    lambda x: f"{int(float(x)):02}" if pd.notna(x) else x)
data_2016['PRVDR_CTGRY_SBTYP_CD'] = data_2016['PRVDR_CTGRY_SBTYP_CD'].apply(
    lambda x: f"{int(float(x)):02}" if pd.notna(x) else x)

# subset to provider type code 01 and subtype code 01
short_term_hospitals = data_2016[(data_2016['PRVDR_CTGRY_CD'] == '01') & (
    data_2016['PRVDR_CTGRY_SBTYP_CD'] == '01')]

short_term_hospitals['PRVDR_NUM'].unique
```

```
<bound method Series.unique of 0         010001
1         010004
2         010005
3         010006
4         010007
           ...
133526    670114
133527    670115
133528    670116
133529    670117
133530    670118
Name: PRVDR_NUM, Length: 7245, dtype: object>
```

a. We know from the data dictionary that each hospital has a unique CMS certification number, which is listed under PRVDR_NUM. There are 7245 unique CMS numbers in this 2016 dataset, indicating that there are 7245 hospitals that fit these categories.

b. According to "Definitive Healthcare" https://www.definitivehc.com/blog/how-many-hospitals-are-in-the-us, there are 3,873 short-term accute hospitals in the United States as of April 2024, which is a much smaller number than the one in our data set. It could be that many hospitals have merged or closed since 2016. According to "Statista" https://www.statista.com/statistics/185843/number-of-all-hospitals-in-the-us/, there were 5534 hospitals in the US in 2016. This is also smaller than the number in our data set, even though the data was from the same year. It could be that hospitals are coded differently in different data sets, or this dataset might exclude or include additional facilities not in other counts.

3.

```
# read in other data sets
data_2017 = pd.read_csv('data/pos2017.csv')
data_2018 = pd.read_csv('data/pos2018.csv', encoding='latin1')
data_2019 = pd.read_csv('data/pos2019.csv', encoding='latin1')

# get rid of strange symbol in the column name
data_2018.rename(
    columns={'ï»¿PRVDR_CTGRY_SBTYP_CD': 'PRVDR_CTGRY_SBTYP_CD'}, inplace=True)
```

```
# Function to make sure that codes in each dataset are in 01 format
def format_provider_codes(data):

    for column in ['PRVDR_CTGRY_CD', 'PRVDR_CTGRY_SBTYP_CD']:
        data[column] = data[column].apply(
            lambda x: f"{int(float(x)):02}" if pd.notna(x) else x
        )
    return data

data_2017 = format_provider_codes(data_2017)
data_2018 = format_provider_codes(data_2018)
data_2019 = format_provider_codes(data_2019)

# add a year column to each df
data_2017['year'] = 2017
data_2018['year'] = 2018
data_2019['year'] = 2019
```

```
# subset data for each year
short_term_hospitals_2017 = data_2017[
    (data_2017['PRVDR_CTGRY_CD'] == '01') &
    (data_2017['PRVDR_CTGRY_SBTYP_CD'] == '01')
]

short_term_hospitals_2018 = data_2018[
    (data_2018['PRVDR_CTGRY_CD'] == '01') &
    (data_2018['PRVDR_CTGRY_SBTYP_CD'] == '01')
]

short_term_hospitals_2019 = data_2019[
    (data_2019['PRVDR_CTGRY_CD'] == '01') &
    (data_2019['PRVDR_CTGRY_SBTYP_CD'] == '01')
]

# combine all four datasets
data = pd.concat(
    [short_term_hospitals, short_term_hospitals_2017,
        short_term_hospitals_2018, short_term_hospitals_2019],
    ignore_index=True
)
```

```
# Count the observations by year
observations_by_year = data.groupby('year').size().reset_index(name='count')
```

```
# Base bar chart
base_chart = alt.Chart(observations_by_year).mark_bar(color='hotpink').encode(
    x=alt.X('year:O', title='Year'),
    y=alt.Y('count:Q', title='Number of Observations')
).properties(
    title='Number of Observations by Year',
    width=400,
    height=300
)

# Text chart
text_chart = alt.Chart(observations_by_year).mark_text(
    align='center',
    baseline='bottom',
    dy=-5,
    color='black'
).encode(
    x=alt.X('year:O'),
    y=alt.Y('count:Q'),
    text='count:Q'
)

# Layer the bar and text charts
chart = base_chart + text_chart
chart
```

```
alt.LayerChart(...)
```

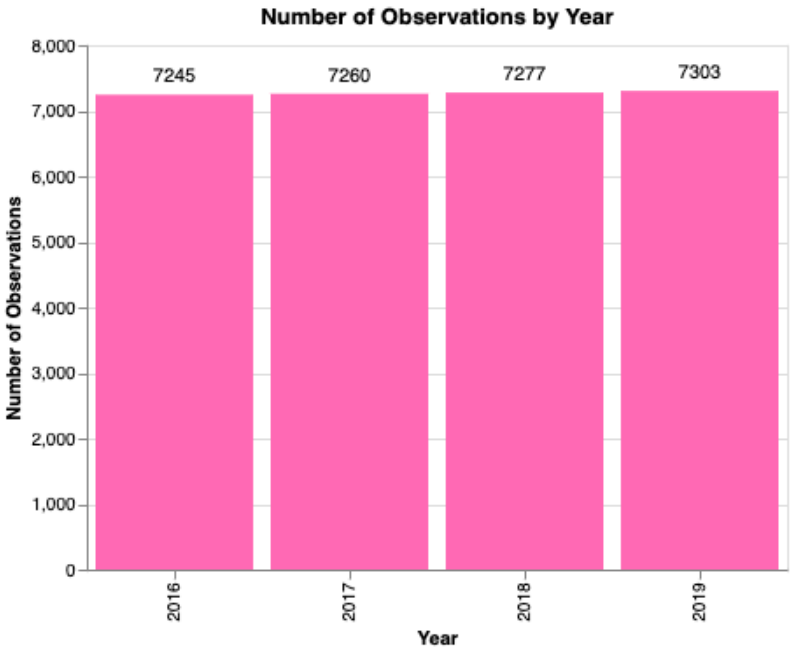


Figure 1: Chart 1

4.

a.

```
# Count the number of unique hospitals by year
unique_hospitals = data.groupby(
    'year')['PRVDR_NUM'].nunique().reset_index(name='unique_hospitals')

# Base bar chart
base_chart2 = alt.Chart(unique_hospitals).mark_bar(color='green').encode(
    x=alt.X('year:O', title='Year'),
    y=alt.Y('unique_hospitals:Q', title='Number of Unique Hospitals')
).properties(
    title='Number of Unique Hospitals by Year',
    width=400,
    height=300
)

# Text chart
text_chart2 = alt.Chart(unique_hospitals).mark_text(
    align='center',
    baseline='bottom',
    dy=-5,
    color='black'
).encode(
    x=alt.X('year:O'),
    y=alt.Y('unique_hospitals:Q'),
    text='unique_hospitals:Q'
)

# Layer the bar and text charts
chart2 = base_chart2 + text_chart2
```

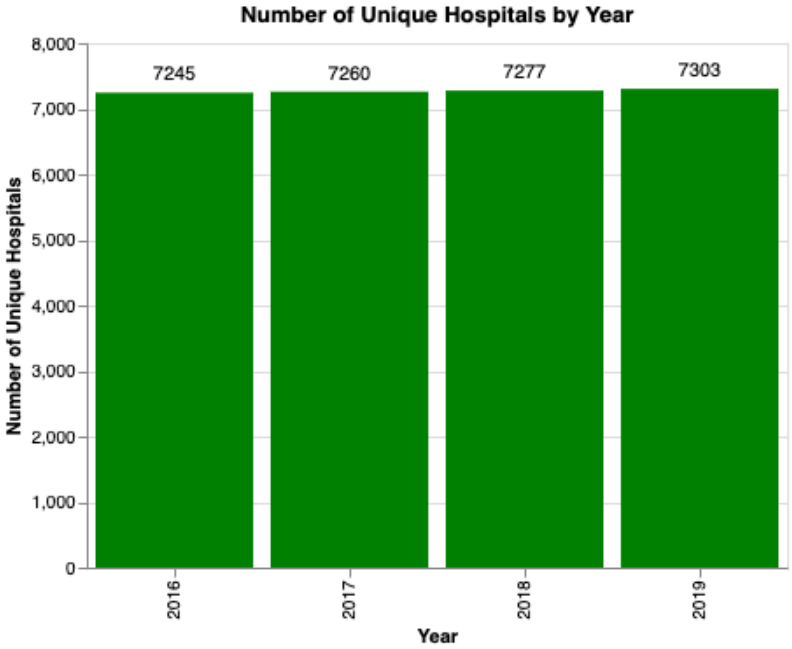


Figure 2: Chart 2

b. The two plots are exactly the same, which tells us that there is one hospital per row of our data set.

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

1.

```
# Which hospitals were active in 2016?
active_2016 = data[(data["year"] == 2016) & (data["PGM_TRMNTN_CD"] == 0)]

closure_info = []

for _, row in active_2016.iterrows():
    facility_name = row["FAC_NAME"]
    zip_code = row["ZIP_CD"]

    for check_year in range(2017, 2020):
        hospital_status = data[(data["FAC_NAME"] == facility_name) & (
            data["year"] == check_year)]

        if hospital_status.empty:
            closure_info.append({
                "FAC_NAME": facility_name,
                "ZIP_CD": zip_code,
                "Year_Closed": check_year
            })
            break
        elif hospital_status["PGM_TRMNTN_CD"].values[0] == 1:
            closure_info.append({
                "FAC_NAME": facility_name,
                "ZIP_CD": zip_code,
                "Year_Closed": check_year
            })
            break

closure_df = pd.DataFrame(closure_info)
```

```
print(f"There are {closure_df.shape[0]} hospitals that were active in 2016 but are suspected to
 ↪  have closed by 2019")
```

```
There are 646 hospitals that were active in 2016 but are suspected to have closed by 2019
```

2.

```
# Sort them by name
closure_df_sorted = closure_df.sort_values(by="FAC_NAME")

# Report the first 10 rows
print(closure_df_sorted.head(10))
```

```
                                       FAC_NAME   ZIP_CD  Year_Closed
14                          ABRAZO MARYVALE CAMPUS  85031.0         2017
327                      ADIRONDACK MEDICAL CENTER  12983.0         2019
226  ADVENTIST HEALTHCARE WASHINGTON ADVENTIST HOSP...  20912.0         2019
431                        ADVENTIST MEDICAL CENTER  97216.0         2018
32                         ADVENTIST MEDICAL CENTER  93230.0         2018
37       ADVENTIST MEDICAL CENTER - CENTRAL VALLEY  93230.0         2017
36             ADVENTIST MEDICAL CENTER - REEDLEY  93654.0         2017
397                          AFFINITY MEDICAL CENTER  44646.0         2018
```

```
339      ALBANY MEDICAL CENTER / SOUTH CLINICAL CAMPUS  12208.0         2017
280                            ALLIANCE LAIRD HOSPITAL  39365.0         2019
```

3.

```
# Creating a function to filter suspected closures
def count_active_hospitals(zip_code, year):
    return data[(data["ZIP_CD"] == zip_code) & (data["year"] == year) & (data["PGM_TRMNTN_CD"]
     ↪  == 0)].shape[0]


# Using the function
valid_closures = []

for index, closure in closure_df.iterrows():
    zip_code = closure["ZIP_CD"]
    year_closed = closure["Year_Closed"]
    active_before = count_active_hospitals(zip_code, year_closed)
    active_after = count_active_hospitals(zip_code, year_closed + 1)

    if active_after < active_before:
        valid_closures.append(closure)

valid_closure_df = pd.DataFrame(valid_closures)
```

a.

```
potential_mergers = []

for index, closure in valid_closure_df.iterrows():
    zip_code = closure["ZIP_CD"]
    year_closed = closure["Year_Closed"]

    active_after = count_active_hospitals(zip_code, year_closed + 1)

    if active_after >= 1:
        potential_mergers.append(closure)

potential_merger_df = pd.DataFrame(potential_mergers)
print(
    f"There are {potential_merger_df.shape[0]} hospitals that fit the definition of potentially
     ↪  being a merger/acquisition")
```

There are 8 hospitals that fit the definition of potentially being a merger/acquisition

b.

```
remaining_hospitals = []

for index, closure in valid_closure_df.iterrows():
    zip_code = closure["ZIP_CD"]
    year_closed = closure["Year_Closed"]
    active_after = count_active_hospitals(zip_code, year_closed + 1)
    if active_after >= 1:
        remaining_hospitals.append(index)

remaining_hospitals_df = valid_closure_df.loc[remaining_hospitals]
```

```
filtered_valid_closure_df = valid_closure_df[~valid_closure_df.index.isin(
    remaining_hospitals_df.index)]

print(
    f"There are {filtered_valid_closure_df.shape[0]} hospitals left after correcting for
     ↪  potential mergers/acquisitions")
```

```
There are 208 hospitals left after correcting for potential mergers/acquisitions
```

c.

```
# Filtering
filtered_valid_closure_df_sorted = filtered_valid_closure_df.sort_values(
    by="FAC_NAME")

# Printing the first 10 rows
print(filtered_valid_closure_df_sorted.head(10))
```

```
                                      FAC_NAME   ZIP_CD  Year_Closed
327                     ADIRONDACK MEDICAL CENTER  12983.0         2019
413                      ALLIANCEHEALTH DEACONESS  73112.0         2019
277          BAPTIST MEM HOSP/ GOLDEN TRIANGLE INC  39701.0         2019
458                  BARIX CLINICS OF PENNSYLVANIA  19047.0         2019
618                       BAY AREA MEDICAL CENTER  54143.0         2019
93   BAY MEDICAL CENTER SACRED HEART HEALTH SYSTEM  32401.0         2019
87           BAYHEALTH - MILFORD MEMORIAL HOSPITAL  19963.0         2019
642                BAYLOR EMERGENCY MEDICAL CENTER  76028.0         2019
629                BAYLOR EMERGENCY MEDICAL CENTER  76227.0         2019
537                BAYLOR MEDICAL CENTER AT UPTOWN  75204.0         2019
```

3. Download Census zip code shapefile (10 pt)

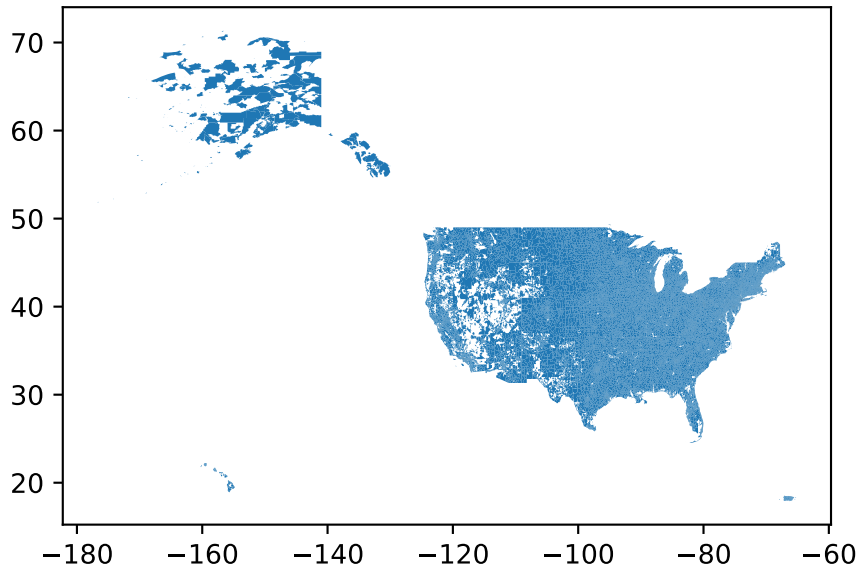
1. a. The five file types are: 1. .dbf (Database File) Stores attribute data associated with spatial features in the shapefile. Includes attributes such as names, values, categories, etc. 2. .prj (Projection File) Defines the coordinate reference system and projection information for the shapefile 3. .shp (Shapefile) Contains the geometry of spatial features like points, lines, and polygons 4. .shx (Shape Index File) Shape index position; stores offsets for each shape 5. .xml (Metadata File) Provides metadata about the dataset (who created it, data descriptions, etc.)

   b. The largest file by far is the .shp file, followed by the .dbf file. The rest are fairly small. 1. .dbf – 6,425,474 bytes (6.4 MB) 2. .prj – 165 bytes (0.000165 MB) 3. .shp – 837,544,580 bytes (847.3 MB) 4. .shx – 265,060 bytes (0.26506 MB) 5. .xml – 15,639 bytes (0.015639 MB)

Source: https://gisgeography.com/gis-formats/

2.

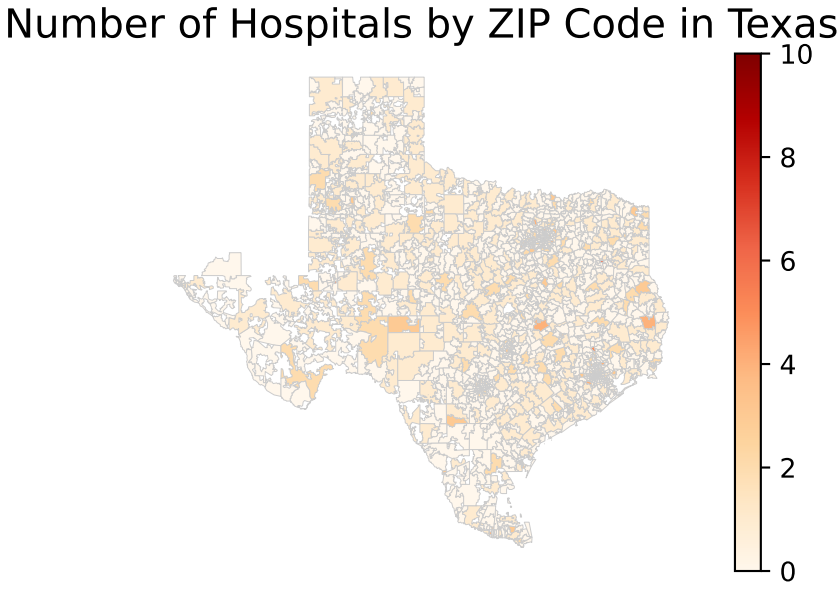
```
# Load the zip code .shp file
zip_shp = gpd.read_file("data/gz_2010_us_860_00_500k.shp")

# reduce the complexity of shape to help with memory
zip_shp['geometry'] = zip_shp.simplify(0.01)

# Plot to see what the data looks like
zip_shp.plot()
```




```
# Filter Texas zip codes
texas_zips = zip_shp[zip_shp['ZCTA5'].str.startswith(
    ('75', '76', '77', '78', '79'))]

# Calculate the number of hospitals per zip code and filter for Texas zip codes, making sure zip
 ↪  codes are the correct length
counts_by_zip = (short_term_hospitals
                 .groupby('ZIP_CD')
                 .size()
                 .reset_index(name='hospital_count'))
counts_by_zip = counts_by_zip[
    (counts_by_zip['ZIP_CD'].astype(str).str.startswith(
        ('75', '76', '77', '78', '79')))
]
```

```
texas_zips["ZIP_CD"] = texas_zips.ZCTA5.astype(float)

# merge data frames together
merged_texas_map = texas_zips.merge( counts_by_zip, how='left', on='ZIP_CD' )
merged_texas_map.hospital_count.fillna(0, inplace=True)

# create the choropleth
merged_texas_map.plot(column='hospital_count', cmap='OrRd',
                linewidth=0.1, edgecolor='0.8', legend=True)
plt.title("Number of Hospitals by ZIP Code in Texas", fontsize=15)
plt.axis("off")
```

```
(np.float64(-107.30234994999999),
 np.float64(-92.85116705),
 np.float64(25.304789749999998),
 np.float64(37.033545249999996))
```



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

1.

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

print(f"The dimensions of this GeoDataFrame are {zips_all_centroids.shape}")
```

```
The dimensions of this GeoDataFrame are (33120, 6)
```

```
print(zips_all_centroids.head(1))
```

```
           GEO_ID  ZCTA5   NAME   LSAD  CENSUSAREA                    geometry
0  8600000US01040  01040  01040  ZCTA5      21.281  POINT (-72.64235 42.20922)
```

GEO_ID: unique code for each area ZCTA5: the actual five digit zip code NAME: duplicative information; renames the actual five digit zip code LSAD: legal/statistical area description; each entry represents a five digit zip code CENSUSAREA: area of the zip code geometry: centroid (latitude and longitude) of the zipcode

2.

```
zips_texas_centroids = zips_all_centroids[
    zips_all_centroids["ZCTA5"].str.startswith(("75", "76", "77", "78", "79"))
]

zips_texas_centroids_grouped = zips_texas_centroids.sort_values("ZCTA5")
print(f"There are {zips_texas_centroids_grouped.shape[0]} unique zip codes in the subset of only
 ↪  Texas zip codes")
```

```
There are 1935 unique zip codes in the subset of only Texas zip codes
```

```
zips_texas_borderstates_centroids = zips_all_centroids[
    zips_all_centroids["ZCTA5"].str.startswith(("75", "76", "77", "78", "79", "870", "871",
 ↪  "872", "873", "874", "875", "876", "877", "878", "879", "880", "881", "882", "883", "884",
 ↪  "73", "74", "716", "717", "718", "719", "720", "721", "722", "723", "724", "725", "726",
 ↪  "727", "728", "729", "700", "701", "702", "703", "704", "705", "706", "707", "708", "709",
 ↪  "710", "711", "712", "713", "714", "715"))
]

zips_texas_borderstates_centroids_grouped =
 ↪  zips_texas_borderstates_centroids.sort_values("ZCTA5")
print(f"There are {zips_texas_borderstates_centroids_grouped.shape[0]} unique zip codes in the
 ↪  subset of Texas and its bordering states' zip codes")
```

```
There are 4057 unique zip codes in the subset of Texas and its bordering states' zip codes
```

3.

```
# filter the 2016 data down to just open hospitals
open_2016 = data_2016[data_2016["PGM_TRMNTN_CD"] == 0]
hospitals_per_zip_2016 = open_2016.groupby("ZIP_CD").size().reset_index(name="HOSPITAL_COUNT")
hospitals_per_zip_2016['ZIP_CD'] =
 ↪  hospitals_per_zip_2016['ZIP_CD'].astype(str).str.split('.').str[0]
hospitals_per_zip_2016['ZIP_CD'] = hospitals_per_zip_2016['ZIP_CD'].astype(str)

zips_withhospital_centroids = zips_texas_borderstates_centroids.merge(
    hospitals_per_zip_2016,
    left_on = 'ZCTA5',
    right_on ='ZIP_CD',
)

zips_withhospital_centroids.head()
```

	GEO_ID	ZCTA5	NAME	LSAD	CENSUSAREA	geometry	ZIP_CD	HOSPITAL_
0	8600000US70003	70003	70003	ZCTA5	7.019	POINT (-90.21273 29.9994)	70003	5
1	8600000US70039	70039	70039	ZCTA5	12.126	POINT (-90.38809 29.87832)	70039	1
2	8600000US70043	70043	70043	ZCTA5	7.775	POINT (-89.96387 29.94707)	70043	3
3	8600000US70047	70047	70047	ZCTA5	10.756	POINT (-90.36794 29.97183)	70047	2
4	8600000US70049	70049	70049	ZCTA5	18.121	POINT (-90.56763 30.02785)	70049	2

I did the default merge in Pandas on the zip code variable after creating a dataframe that grouped the 2016 open hospitals by zip code.

4.

```
# subsetting to 10 zip codes in zips_texas_centroids
zips_texas_centroids_subset = zips_texas_centroids.head(10)

from shapely.ops import nearest_points
from shapely.geometry import Point

zips_texas_centroids_subset = zips_texas_centroids_subset.to_crs(epsg=4269)
zips_withhospital_centroids = zips_withhospital_centroids.to_crs(epsg=4269)
```

a.

```
start_time1 = time.time()

zips_texas_centroids_subset_data = zips_texas_centroids_subset[["ZCTA5", "geometry"]]
zips_withhospital_centroids_data = zips_withhospital_centroids[["ZCTA5", "geometry"]]

nearest_hospital_texas_subset = gpd.sjoin_nearest(zips_texas_centroids_subset_data,
 ↪  zips_withhospital_centroids_data, how = "inner", distance_col = "distance")

end_time1 = time.time()

total_time1 = end_time1 - start_time1

print(f"It took {total_time1:.2f} seconds to join the subset of 10. I estimate it will then tak
 ↪  {((1935/10) * total_time1):.2f} seconds to join all the zipcodes in Texas.")
```

```
It took 0.01 seconds to join the subset of 10. I estimate it will then tak 2.17 seconds to join
all the zipcodes in Texas.
```

b.

```
zips_texas_centroids = zips_texas_centroids.to_crs(epsg=4269)

start_time2 = time.time()

zips_texas_centroids_data = zips_texas_centroids[["ZCTA5", "geometry"]]
zips_withhospital_centroids_data = zips_withhospital_centroids[["ZCTA5", "geometry"]]

nearest_hospital_texas = gpd.sjoin_nearest(zips_texas_centroids_data,
 ↪  zips_withhospital_centroids_data, how = "inner", distance_col = "distance")

end_time2 = time.time()

total_time2 = end_time2 - start_time2
print(f"The join for all the texas zip codes took {total_time2:.2f} seconds, even though I
 ↪  originally estimated it'd take {((1935/10) * total_time1):.2f} seconds. I was off by
 ↪  {((1935/10) * total_time1) - total_time2:.2f} seconds.")
nearest_hospital_texas.head()
```

```
The join for all the texas zip codes took 0.02 seconds, even though I originally estimated it'd
take 2.17 seconds. I was off by 2.15 seconds.
```

	ZCTA5_left	geometry	index_right	ZCTA5_right	distance
9207	78624	POINT (-98.87742 30.2824)	169	78624	0.000000
9208	78626	POINT (-97.60061 30.66661)	170	78626	0.000000
9209	78628	POINT (-97.75319 30.64091)	171	78628	0.000000
9210	78631	POINT (-99.30565 30.33748)	1933	78028	0.337357
9211	78632	POINT (-97.47022 29.69642)	1001	78648	0.163200

c.

It is in degrees, and one degree is roughly 69 miles (USGS.gov)

```
# Degrees to miles is roughly * 69
nearest_hospital_texas["distance_miles"] = nearest_hospital_texas["distance"] * 69
```

5.

```
nearest_hospital_texas["distance"].sum() / nearest_hospital_texas.shape[0]
```

```
np.float64(0.07534847363433031)
```

a. This value is in degrees

```
(nearest_hospital_texas["distance"].sum() / nearest_hospital_texas.shape[0]) * 69
```

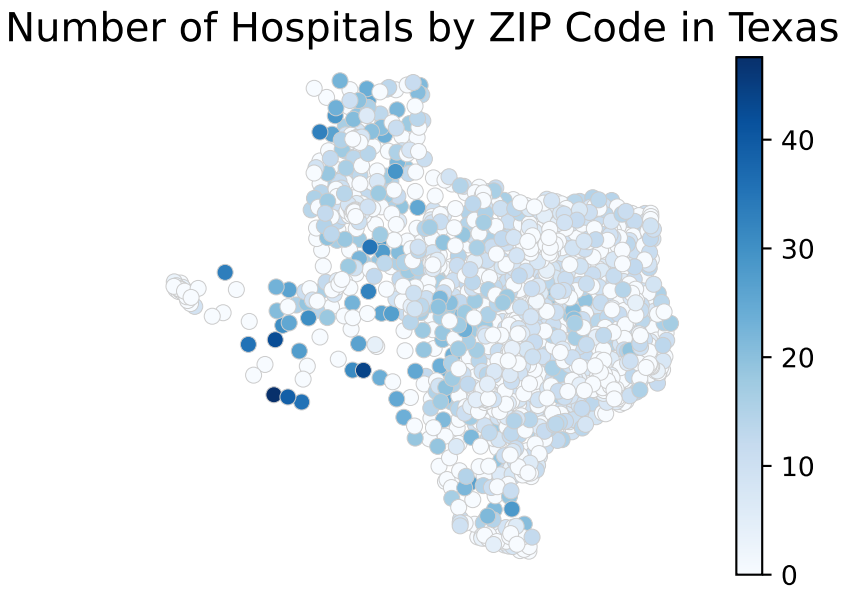
```
np.float64(5.199044680768791)
```

b. Now it's about 5.2 miles. This makes much more sense.

c.

```
# create the choropleth
nearest_hospital_texas.plot(column='distance_miles', cmap='Blues',
                linewidth=0.1, edgecolor='0.8', legend=True)
plt.title("Number of Hospitals by ZIP Code in Texas", fontsize=15)
plt.axis("off")
```

```
(np.float64(-107.2547319970452),
 np.float64(-92.96152391597771),
 np.float64(25.425492979153006),
 np.float64(37.00444329833162))
```



5. Effects of closures on access in Texas (15 pts)

1.

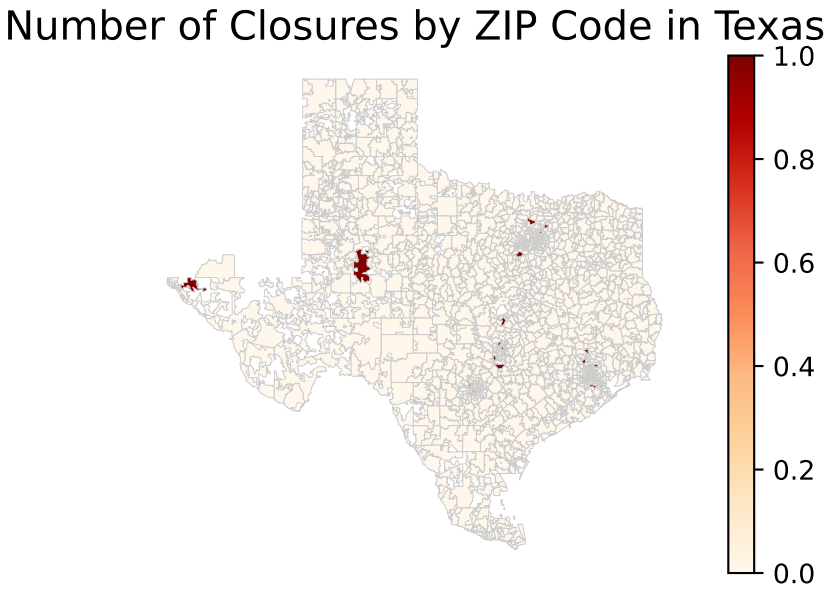
```
#filter closure df for Texas zip codes
filtered_valid_closure_df_TX = filtered_valid_closure_df[
    (filtered_valid_closure_df['ZIP_CD'].astype(str).str.startswith(
        ('75', '76', '77', '78', '79')))
]

# Count the number of closures for each zip code and put into a df
zip_code_counts = filtered_valid_closure_df['ZIP_CD'].value_counts()
closure_summary = zip_code_counts.reset_index()
closure_summary.columns = ['ZIP_CD', 'Number_of_Closures']

# Sort the DataFrame by the number of closures in descending order
closure_summary.sort_values(by='Number_of_Closures', ascending=False, inplace=True)
closure_summary.reset_index(drop=True, inplace=True)

closure_summary

total_closures_tx = closure_summary['Number_of_Closures'].sum()
print(f"There were {total_closures_tx} hospital closures in Texas between 2016 and 2019")
```

```
There were 208 hospital closures in Texas between 2016 and 2019
```

2.

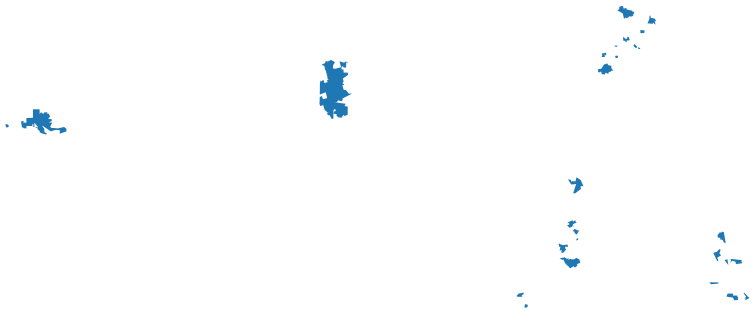
```
# merge texas zips and closure data frames together
merged_texas_zips = texas_zips.merge(closure_summary, how='left', on='ZIP_CD')
merged_texas_zips.Number_of_Closures.fillna(0, inplace=True)

# create the choropleth
merged_texas_zips.plot(column='Number_of_Closures', cmap='OrRd',
                linewidth=0.1, edgecolor='0.8', legend=True)
plt.title("Number of Closures by ZIP Code in Texas", fontsize=15)
plt.axis("off")
```

```
(np.float64(-107.30234994999999),
 np.float64(-92.85116705),
 np.float64(25.304789749999998),
 np.float64(37.033545249999996))
```

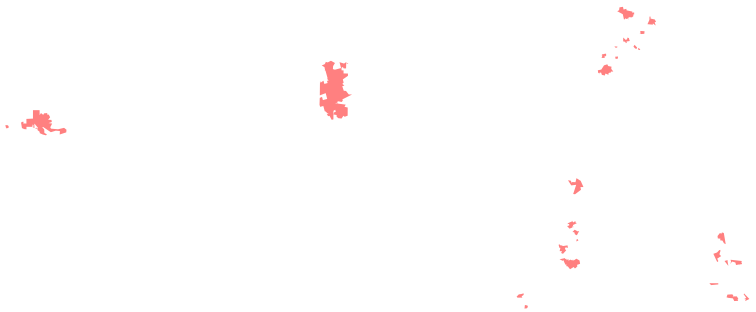


3.

```
# Geo Data Frame of the directly affected Zip Codes
directly_affected_zips = merged_texas_zips[merged_texas_zips['Number_of_Closures'] > 0]
directly_affected_zips.plot().set_axis_off()
```

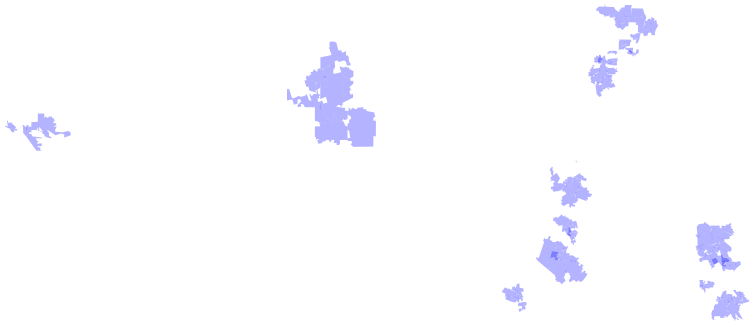


```
# Create the 10 mile buffer (converted miles to meters)
directly_affected_buffer = directly_affected_zips.copy()
directly_affected_buffer['ZIP_CD'] = directly_affected_zips.geometry.buffer(16093.4)
directly_affected_buffer.plot(color="red", alpha=0.5).set_axis_off()
```



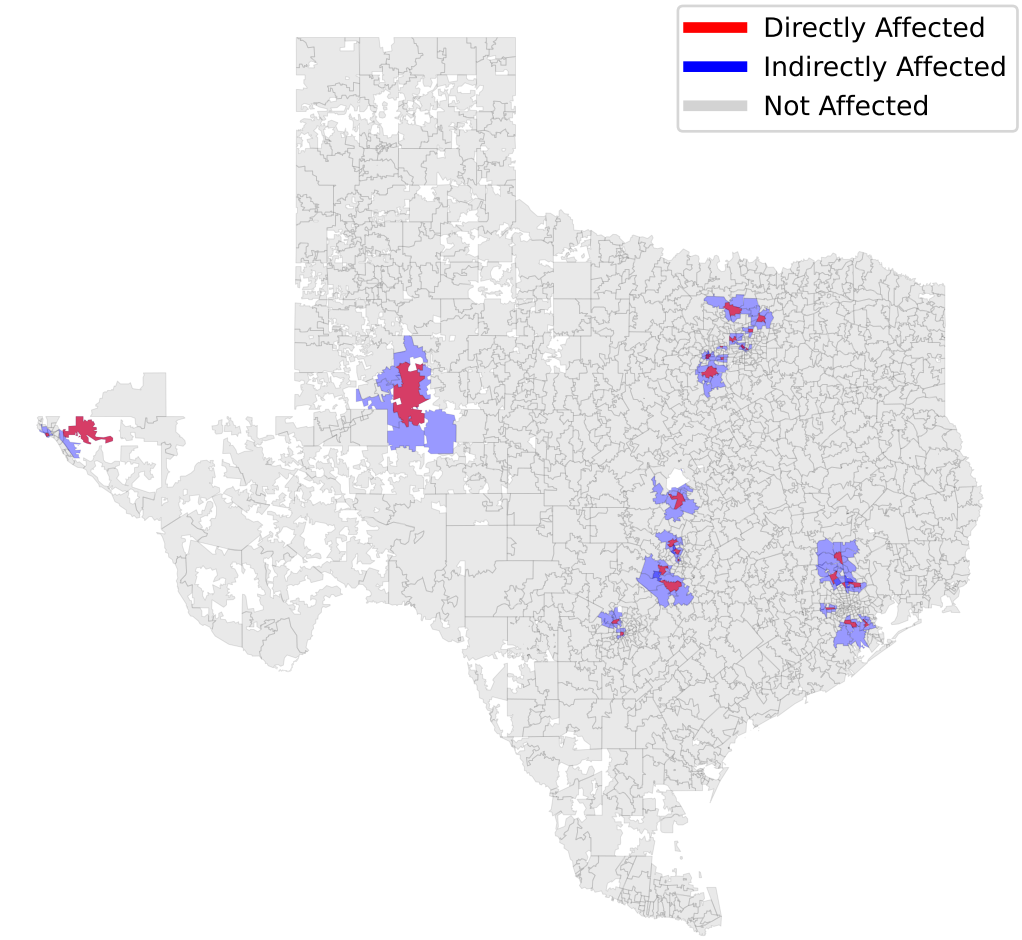
```
# Spatial Join
near_zip = gpd.sjoin(texas_zips, directly_affected_buffer, how="inner", predicate="intersects")
near_zip.plot(color="blue", alpha=0.3).set_axis_off()

num_indirectly_affected_zips = near_zip['ZIP_CD_left'].nunique()
print(f"Number of indirectly affected ZIP codes in Texas: {num_indirectly_affected_zips}")
```

```
Number of indirectly affected ZIP codes in Texas: 186
```



4.

```
# Create a new column for categories
merged_texas_zips['Category'] = 'Not Affected'
merged_texas_zips.loc[merged_texas_zips['Number_of_Closures'] > 0, 'Category'] = 'Directly
 ↪  Affected'
merged_texas_zips.loc[merged_texas_zips['ZIP_CD'].isin(near_zip['ZIP_CD_left']), 'Category'] =
 ↪  'Indirectly Affected'

#Create the plot
fig, ax = plt.subplots(figsize=(7, 7))
merged_texas_zips[merged_texas_zips['Category'] == 'Not Affected'].plot(ax=ax,
 ↪  color="lightgrey", alpha=0.5, label="Not Affected")
near_zip.plot(ax=ax, color="blue", alpha=0.4, label="Indirectly Affected")
directly_affected_zips.plot(ax=ax, color="red", alpha=0.6, label="Directly Affected")

# Add legend, boundaries, title, and adjust axes
texas_zips.boundary.plot(ax=ax, color='black', linewidth=0.1, alpha=0.1)
plt.title("Impact of Hospital Closures on Texas ZIP Codes", fontsize=16)
ax.axis('off')

from matplotlib.lines import Line2D

handles = [
    Line2D([0], [0], color='red', lw=4, label='Directly Affected'),
    Line2D([0], [0], color='blue', lw=4, label='Indirectly Affected'),
    Line2D([0], [0], color='lightgrey', lw=4, label='Not Affected'),
]
ax.legend(handles=handles, loc='upper right', fontsize=10)

plt.show()
```



6. Reflecting on the exercise (10 pts)

Partner 1 Reflection: The first-pass method is imperfect because we're looking at zip codes where the number of active hospitals does not decrease in the year after the suspected closure and assuming that when this number doesn't change; the hospital actually remained open. We are ignoring the possibility that this hospital actually did close down, but an entirely new hospital opened up in its place. A better way to address the hospital closures that might be illegitimate is to focus more on the repeated use of the facility name or CMS certification number, in order to actually zero in on the repeat players.

Partner 2 Reflection: The way in which we identified ZIP codes affected by hospital closures may not fully reflect the real-life complexities of accessing a hospital. Instead of solely focusing on closure status, it might be helpful to incorporate other metrics such as distance to the nearest hospital, travel time by different modes of transport (car, public transit), and population demographics. This method also overlooks factors like distance to the nearest healthcare facilities and the availability of alternative care options, such as urgent care centers or telehealth services.