PS4 v1.1: Spatial

Boya Lin and Zidan Kong

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): Boya Lin, boya1
    - Partner 2 (name and cnet ID): Zidan Kong, zidank
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: **BL** **ZK**
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** Late coins left after submission: **2**
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 aguide. 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 altair as alt
alt.renderers.enable('png')
import geopandas as gpd
import matplotlib.pyplot as plt
```

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

1.

Vairables we pulled in pos2016.csv include PRVDR_CTGRY_SBTYP_CD, PRVDR_CTGRY_CD, CITY_NAME, FAC_NAME(Facility Name), ORGNL_PRTCPTN_DT, PRVDR_NUM(CMS Certification Number), PGM_TRMNTN_CD(Termination Code), TRMNTN_EXPRTN_DT, and ZIP_CD.

2. a.

```
# read csv file and filter to short-term hospitals in 2016
pos_2016 = pd.read_csv('pos2016.csv')
short_term_hospitals_2016 = pos_2016[(pos_2016['PRVDR_CTGRY_CD'] == 1) & (
    pos_2016['PRVDR_CTGRY_SBTYP_CD'] == 1)].copy()
# ChatGDP reference for SettingWithCopyWarning, the usage of .copy

count_2016 = short_term_hospitals_2016.shape[0]
print(f'Number of short-term hospitals in 2016: {count_2016}')
```

```
Number of short-term hospitals in 2016: 7245
```

This number appears slightly high, since the American Hospital Association(AHA) reported a total of 6,120 hospitals in the U.S. in 2024. In consideration of the growing hospital industry, we would generally expect the number of hospitals in 2016 to be lower than that in 2024.

Data Source(AHA): https://www.aha.org/statistics/fast-facts-us-hospitals

b.

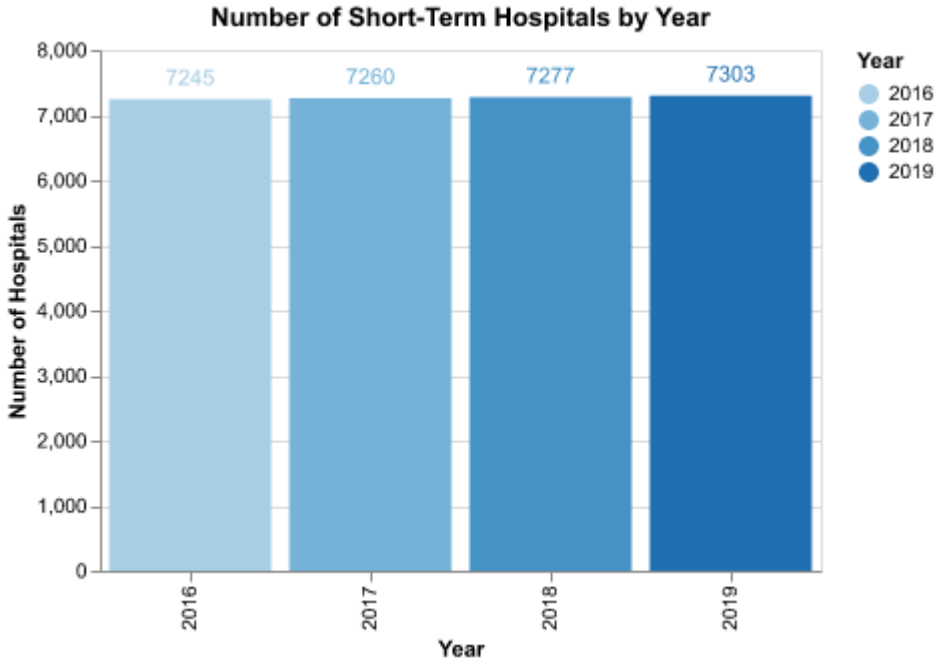
Unlike 7,245 hospitals reported in our data, the American Hospital Association(AHA) reports a total number of 5,534 U.S. hospitals for 2016. The difference may arise from factors such as hospitals name changes, acquisitions, or the presence of branches or devisions, which can lead to multiple entries for the same facility in our data. As a result, when using .nunique(), these hospitals or facilitates might be counted separately, leading to an inflated total.

Data Source(AHA): https://www.aha.org/system/files/2018-01/Fast%20Facts%202018%20pie%20charts.pdf

3.

```
# read csv file and filter to short-term hospitals for 2017-2019
pos_2017 = pd.read_csv('pos2017.csv')
short_term_hospitals_2017 = pos_2017[(pos_2017['PRVDR_CTGRY_CD'] == 1) & (
    pos_2017['PRVDR_CTGRY_SBTYP_CD'] == 1)].copy()

pos_2018 = pd.read_csv('pos2018.csv', encoding='ISO-8859-1')
short_term_hospitals_2018 = pos_2018[(pos_2018['PRVDR_CTGRY_CD'] == 1) & (
    pos_2018['PRVDR_CTGRY_SBTYP_CD'] == 1)].copy()

pos_2019 = pd.read_csv('pos2019.csv', encoding='ISO-8859-1')
short_term_hospitals_2019 = pos_2019[(pos_2019['PRVDR_CTGRY_CD'] == 1) & (
    pos_2019['PRVDR_CTGRY_SBTYP_CD'] == 1)].copy()
# ChatGDP reference for "UnicodeDecodeError: 'utf-8' codec can't decode
# byte 0xa0 in position 119424", solution: add encoding='ISO-8859-1'

# add column indicating corresponding year
short_term_hospitals_2016.loc[:, 'Year'] = 2016
short_term_hospitals_2017.loc[:, 'Year'] = 2017
short_term_hospitals_2018.loc[:, 'Year'] = 2018
short_term_hospitals_2019.loc[:, 'Year'] = 2019
all_hospitals = pd.concat([short_term_hospitals_2016,
                           short_term_hospitals_2017,
                           short_term_hospitals_2018,
                           short_term_hospitals_2019], ignore_index=True)
# Pandas official website reference for pd.concat:
# https://pandas.pydata.org/docs/reference/api/pandas.concat.html

# group by year and make a barchart
hospitals_by_year = all_hospitals.groupby(
    'Year').size().reset_index(name='Number of Hospitals')

bar_chart = alt.Chart(hospitals_by_year).mark_bar().encode(
    x='Year:O', y='Number of Hospitals:Q', color='Year:O'
```

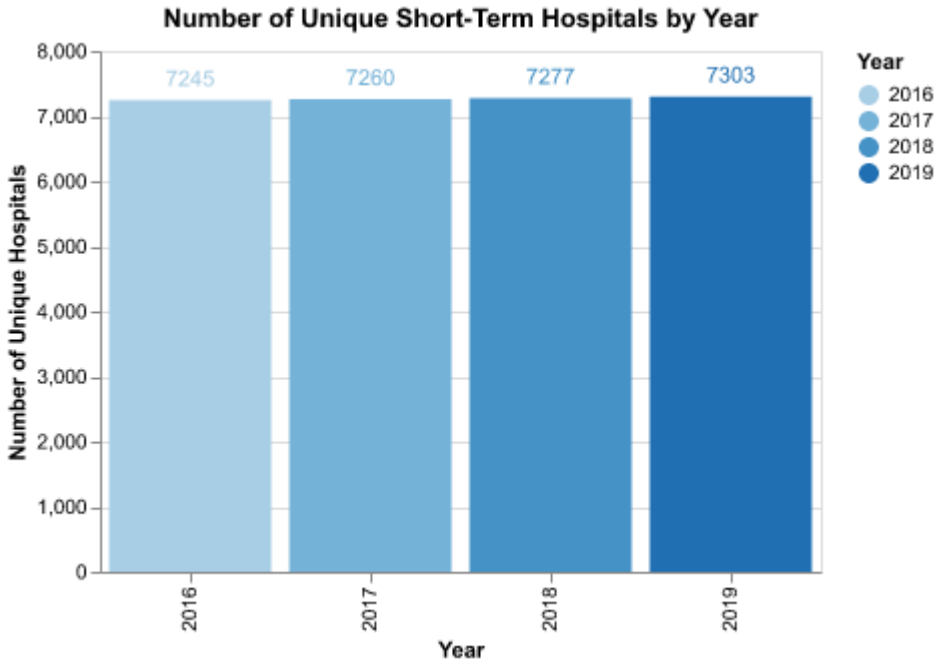
```
).properties(
    title='Number of Short-Term Hospitals by Year', width=360, height=260
)

text_labels = bar_chart.mark_text(
    align='center', baseline='bottom', dy=-5).encode(text='Number of Hospitals:Q')
hospital_chart = bar_chart + text_labels
hospital_chart
# ChatGDP reference for adding exact values of data above bar chart
```



4. a.

```
# group by year and count unique hospitals based on PRVDR_NUM
unique_hospitals_by_year = all_hospitals.groupby(
    'Year')['PRVDR_NUM'].nunique().reset_index(name='Number of Unique Hospitals')

# make a barchart
bar_chart = alt.Chart(unique_hospitals_by_year).mark_bar().encode(
    x='Year:O', y='Number of Unique Hospitals:Q', color='Year:O'
).properties(
    title='Number of Unique Short-Term Hospitals by Year',
    width=360, height=260
```

```
)

text_labels = bar_chart.mark_text(
    align='center', baseline='bottom', dy=-5).encode(text='Number of Unique Hospitals:Q')
unique_hospitals_chart = bar_chart + text_labels
unique_hospitals_chart
```



b.

Both plots present the same counts of observations for each year. It suggests that each observation in our dataset corresponds to a unique hospital(identified by its CMS certification number). In other words, there are no duplicate reocrds for nay hospital in any year.

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

1.

```
# filter active hospitals in 2016
active_2016 = short_term_hospitals_2016[short_term_hospitals_2016['PGM_TRMNTN_CD'] == 0]
active_2016 = active_2016[['PRVDR_NUM', 'FAC_NAME', 'ZIP_CD']]
```

```
# merge with 2017 data
merged_2017 = active_2016.merge(
    short_term_hospitals_2017[['PRVDR_NUM', 'FAC_NAME',
                               'ZIP_CD', 'PGM_TRMNTN_CD']],
    on='PRVDR_NUM',
    how='left',
    indicator="_merge_2017",
    suffixes=('', '_2017')
)

closed_2017 = merged_2017[(merged_2017['_merge_2017'] == 'left_only') | (
    merged_2017['PGM_TRMNTN_CD'] != 0)].copy()

# add the suspected closure year
closed_2017.loc[:, 'Year of Suspected Closure'] = 2017
closed_2017 = closed_2017[['FAC_NAME_2017', 'ZIP_CD_2017',
                           'Year of Suspected Closure', 'PRVDR_NUM']]
```

```
# merge with 2018 data
merged_2018 = active_2016.merge(
    short_term_hospitals_2018[['PRVDR_NUM', 'FAC_NAME',
                               'ZIP_CD', 'PGM_TRMNTN_CD']],
    on='PRVDR_NUM',
    how='left',
    indicator="_merge_2018",
    suffixes=('', '_2018')
)

closed_2018 = merged_2018[(merged_2018['_merge_2018'] == 'left_only') | (
    merged_2018['PGM_TRMNTN_CD'] != 0)].copy()

# add the suspected closure year
closed_2018.loc[:, 'Year of Suspected Closure'] = 2018
closed_2018 = closed_2018[['FAC_NAME_2018', 'ZIP_CD_2018',
                           'Year of Suspected Closure', 'PRVDR_NUM']]
```

```
# merge with 2019 data
merged_2019 = active_2016.merge(
    short_term_hospitals_2019[['PRVDR_NUM', 'FAC_NAME',
                               'ZIP_CD', 'PGM_TRMNTN_CD']],
    on='PRVDR_NUM',
    how='left',
```

```
    indicator="_merge_2019",
    suffixes=('', '_2019')
)

closed_2019 = merged_2019[(merged_2019['_merge_2019'] == 'left_only') | (
    merged_2019['PGM_TRMNTN_CD'] != 0)].copy()

# add the suspected closure year
closed_2019.loc[:, 'Year of Suspected Closure'] = 2019
closed_2019 = closed_2019[['FAC_NAME_2019', 'ZIP_CD_2019',
                           'Year of Suspected Closure', 'PRVDR_NUM']]
```

```
# rename column for concat
closed_2017 = closed_2017.rename(columns={'ZIP_CD_2017': 'ZIP_CD'})
closed_2018 = closed_2018.rename(columns={'ZIP_CD_2018': 'ZIP_CD'})
closed_2019 = closed_2019.rename(columns={'ZIP_CD_2019': 'ZIP_CD'})

closed_2017 = closed_2017.rename(columns={'FAC_NAME_2017': 'FAC_NAME'})
closed_2018 = closed_2018.rename(columns={'FAC_NAME_2018': 'FAC_NAME'})
closed_2019 = closed_2019.rename(columns={'FAC_NAME_2019': 'FAC_NAME'})

# concat datasets
full_closures = pd.concat(
    [closed_2017, closed_2018, closed_2019], ignore_index=True)
full_closures = full_closures.drop_duplicates(
    subset=['FAC_NAME', 'ZIP_CD'], keep='first')
# Referencing ChatGDP with suffixes, asking how do i solve the problem when
# oit creates x and y column for coulumn appears in both dataset.

number_of_hospital = len(full_closures)
print(f'Number of hospitals facing suspected closures: {number_of_hospital}')
```

```
Number of hospitals facing suspected closures: 177
```

2.

```
# sorting by name
sorted_closed = full_closures.sort_values(by='FAC_NAME')
sorted_closed[['FAC_NAME', 'Year of Suspected Closure']].head(10)
```

	FAC_NAME	Year of Suspected Closure
265	(CLOSED) HEALTHSOUTH CHATTANOOGA REHAB HOSPITAL	2019
0	ABRAZO MARYVALE CAMPUS	2017
1	ADVENTIST MEDICAL CENTER - CENTRAL VALLEY	2017
90	AFFINITY MEDICAL CENTER	2018
15	ALBANY MEDICAL CENTER / SOUTH CLINICAL CAMPUS	2017
29	ALLEGIANCE SPECIALTY HOSPITAL OF KILGORE	2017
200	ALLIANCE LAIRD HOSPITAL	2019
239	ALLIANCEHEALTH DEACONESS	2019
3	ARKANSAS VALLEY REGIONAL MEDICAL CENTER	2017
125	ASCENSION NE WISCONSIN MERCY CAMPUS	2018

3.

```
# taking active hospitals datas and group by on zip codes
active_zip_2016 = short_term_hospitals_2016[short_term_hospitals_2016[
    'PGM_TRMNTN_CD'] == 0].groupby('ZIP_CD').size().reset_index(name='zip_count_2016')
active_zip_2017 = short_term_hospitals_2017[short_term_hospitals_2017[
    'PGM_TRMNTN_CD'] == 0].groupby('ZIP_CD').size().reset_index(name='zip_count_2017')
active_zip_2018 = short_term_hospitals_2018[short_term_hospitals_2018[
    'PGM_TRMNTN_CD'] == 0].groupby('ZIP_CD').size().reset_index(name='zip_count_2018')
active_zip_2019 = short_term_hospitals_2019[short_term_hospitals_2019[
    'PGM_TRMNTN_CD'] == 0].groupby('ZIP_CD').size().reset_index(name='zip_count_2019')
```

```
# merge number of active hospitals grouped by zip codes to one dataset
zip_20162017 = active_zip_2016.merge(active_zip_2017, on='ZIP_CD', how='outer')
zip_20162018 = zip_20162017.merge(active_zip_2018, on='ZIP_CD', how='outer')
zip_20162019 = zip_20162018.merge(active_zip_2019, on='ZIP_CD', how='outer')
```

```
# checking for definition for potential merges
closures_zip_checking = full_closures.merge(
    zip_20162019, on='ZIP_CD', how='left')
potential_mergers_2017 = closures_zip_checking[
    (closures_zip_checking['Year of Suspected Closure'] == 2017) &
    (closures_zip_checking['zip_count_2017'] <= closures_zip_checking['zip_count_2018'])]
potential_mergers_2018 = closures_zip_checking[
    (closures_zip_checking['Year of Suspected Closure'] == 2018) &
    (closures_zip_checking['zip_count_2018'] <= closures_zip_checking['zip_count_2019'])]
```

```
# concat to full dataset
potential_mergers = pd.concat(
    [potential_mergers_2017, potential_mergers_2018])
```

a.

```
# get numbers of potential merges
number_potential_merges = len(potential_mergers)
print(f'Number of hospitals potentially being a merger/acquisition: {number_potential_merges}')
```

```
Number of hospitals potentially being a merger/acquisition: 27
```

b.

```
# remove potential merges
corrected_closures = full_closures.merge(potential_mergers[[
    'FAC_NAME', 'ZIP_CD', 'Year of Suspected Closure']],
    on=['FAC_NAME', 'ZIP_CD', 'Year of Suspected Closure'],
    how='left', indicator=True)

corrected_closures = corrected_closures[corrected_closures['_merge'] == 'left_only']
number_correct_closure = len(corrected_closures)
print(f'Number of hospitals left after correction: {number_correct_closure}')
```

```
Number of hospitals left after correction: 150
```

c.

```
# sorted by name
sorted_correct_closure = corrected_closures.sort_values(by='FAC_NAME')
sorted_correct_closure[['FAC_NAME', 'Year of Suspected Closure']].head(10)
```

	FAC_NAME	Year of Suspected Closure
160	(CLOSED) HEALTHSOUTH CHATTANOOGA REHAB HOSPITAL	2019
0	ABRAZO MARYVALE CAMPUS	2017
73	AFFINITY MEDICAL CENTER	2018
29	ALLEGIANCE SPECIALTY HOSPITAL OF KILGORE	2017

	FAC_NAME	Year of Suspected Closure
132	ALLIANCE LAIRD HOSPITAL	2019
149	ALLIANCEHEALTH DEACONESS	2019
3	ARKANSAS VALLEY REGIONAL MEDICAL CENTER	2017
142	ATRIUM HEALTH KINGS MOUNTAIN	2019
11	BANNER CHURCHILL COMMUNITY HOSPITAL	2017
42	BANNER PAYSON MEDICAL CENTER	2018

Download Census zip code shapefile (10 pt)

1. a. The five file types are .xml, .shx, .shp, .prj, and .dbf.

gz_2010_us_860_00_500k.xml is an XML file, containing metadata for the 2010 cartographic boundary polygon shapefiles representing 5-Digit ZIP Code Tabulation Areas(ZCTAs) in the U.S published by the U.S. Census Bureau. gz_2010_us_860_00_500k.shx is a shape index file with positional index for the accompanying .shp file, allowing for efficient access to geometric data for efficient spatial querying. gz_2010_us_860_00_500k.shp is the core file of the shapefile package, containing the geometric representations of geographic features, such as points, lines, and polygons, defining the actual shapes and locations. gz_2010_us_860_00_500k.prj is a small projection file, describing the Coordinate Reference System (CRS) and projection used for the spatial data, necessary for spatial accuracy. gz_2010_us_860_00_500k.dbf holds attribute information like population, area, or other demographic details corresponding to the geometric shapes in the .shp file, providing context and depth to the geometrical information.

b.

After unzipping, the gz_2010_us_860_00_500k.xml file is 16 KB, gz_2010_us_860_00_500k.shx is 265 KB, gz_2010_us_860_00_500k.shp is 837.5 MB, gz_2010_us_860_00_500k.prj is 165 bytes, and gz_2010_us_860_00_500k.dbf is 6.4 MB.

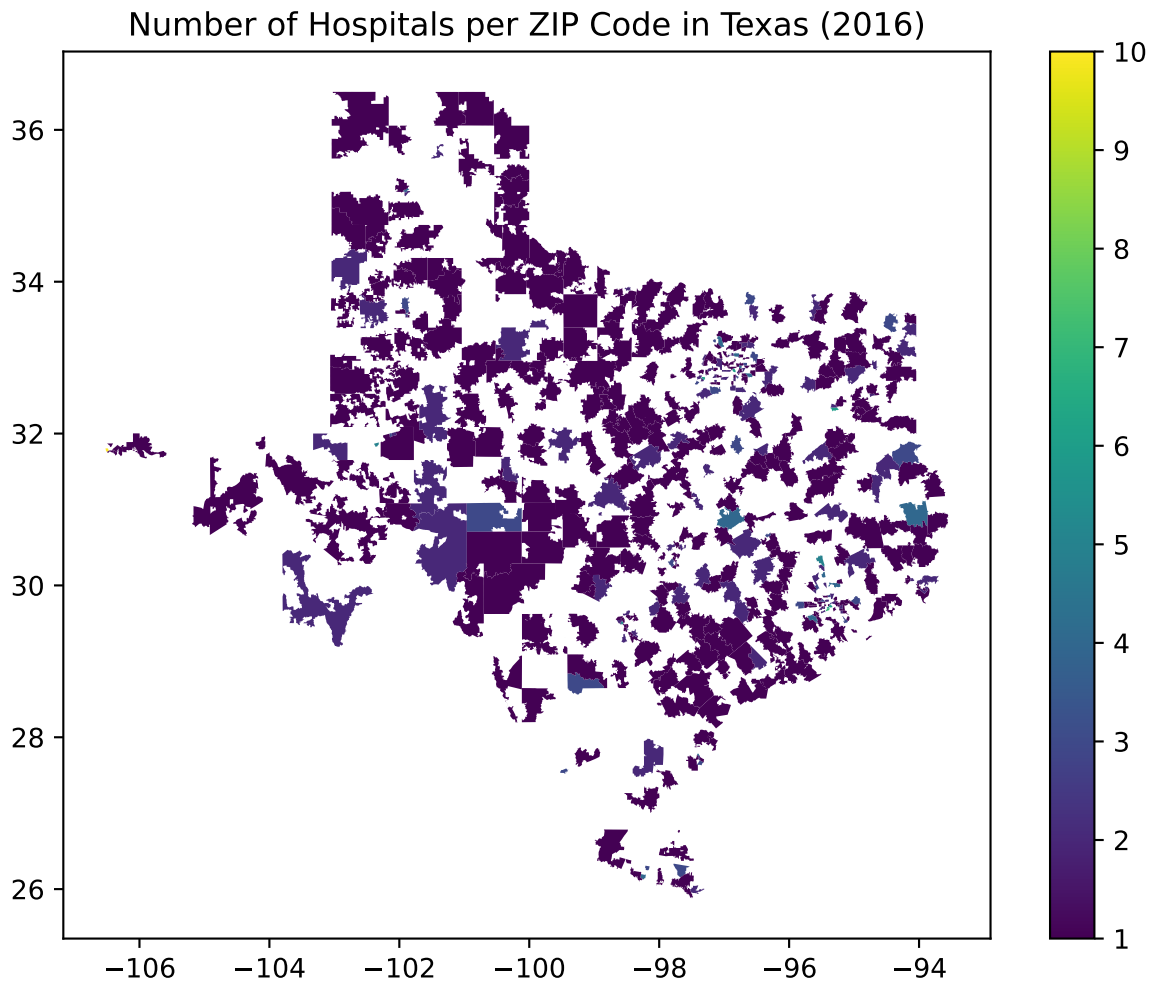
2.

```
data = gpd.read_file('/Users/boyalin/Documents/GitHub/ppha30538_ps/problem-set-4-boya-zidan/g

# filter to Texas zip codes for both datasets
texas_zip = data[data['ZCTA5'].astype(str).str.startswith(
    ('733', '75', '76', '77', '78', '79'))]
pos_2016_tx = short_term_hospitals_2016[short_term_hospitals_2016['ZIP_CD'].astype(
    str).str.startswith(('733', '75', '76', '77', '78', '79'))]
```

```
hospitals_per_zip = pos_2016_tx['ZIP_CD'].value_counts().reset_index()
hospitals_per_zip.columns = ['ZIP_CD', 'counts']
hospitals_per_zip.head(6)
```

	ZIP_CD	counts
0	79902.0	10
1	77054.0	7
2	75235.0	6
3	76104.0	6
4	77004.0	6
5	77030.0	6

```
# make both datasets have same column name and data type
texas_zip = texas_zip.rename(columns={'ZCTA5': 'ZIP_CD'})
texas_zip['ZIP_CD'] = texas_zip['ZIP_CD'].astype(str)
hospitals_per_zip['ZIP_CD'] = hospitals_per_zip['ZIP_CD'].astype(
    int).astype(str)

# merge the two datasets for plotting
texas_map = texas_zip.merge(hospitals_per_zip, on='ZIP_CD', how='left')

fig, ax = plt.subplots(1, 1, figsize=(8, 6))
texas_map.plot(column='counts', ax=ax, legend=True)
ax.set_title('Number of Hospitals per ZIP Code in Texas (2016)',
             fontdict={'fontsize': '12'})
plt.show()

# Chatgpt reference for merging error, searching you are trying to merge on
# object and float64 columns for key 'ZIP_CD'. If you wish to proceed you should
# use pd.concat, solving with changing datatype
# ChatGDP reference for adding title for the plot
```



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

1.

```
# calculate centroid
zips_all_centroids = data.copy()
zips_all_centroids['centroid'] = zips_all_centroids['geometry'].centroid
```

```
/var/folders/r5/j3b0xjqs7h9bw5yxznft8cl00000gn/T/ipykernel_12263/531323553.py:3: UserWarning

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

```
# get dimension
zips_all_centroids.shape
```

```
(33120, 7)
```

```
# get column names
zips_all_centroids.columns
zips_all_centroids.head(5)
```

	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

The dimension of the dataframe is that it contains 33120 observations and 7 columns. "GEO_ID": identified ID for each area. "ZCTA5": ZIP Code Tabulation Area (ZCTA) for each zipcode area. "NAME": seems like same to ZCTA5, area names. "LSAD": Legal/Statistical Area Description, description of what kind of area notation are using for dataset. "CENSUSAREA": the area of the ZCTA as calculated by the Census. "geometry": polygon or multipolygon shaping information. "centroid": centroid point of the ZIP code polygon area.

2.

```
# identified prefixes
texas_prefixes = ('733', '75', '76', '77', '78', '79')
bordering_prefixes = ('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', '733',
                      '75', '76', '77', '78', '79')
# subset Texas data with prefixes
zips_texas_centroids = zips_all_centroids[zips_all_centroids['ZCTA5'].str.startswith(
    texas_prefixes)]
# subset Texas and borders data with prefixes
zips_texas_borderstates_centroids = zips_all_centroids[zips_all_centroids[
```

```
    'ZCTA5'].str.startswith(bordering_prefixes)]
# get the number of unique zip codes
num_unique_zip_tx = zips_texas_centroids['ZCTA5'].nunique()
num_unique_zip_all = zips_texas_borderstates_centroids['ZCTA5'].nunique()

print(f'Number of unique zip codes in Texas: {num_unique_zip_tx}')
print(
    f'Number of unique zip codes in Texas and bordering state: {num_unique_zip_all}')
```

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

3.

```
# filter zip codes with at least one hospital
hospitals_per_zip_1 = hospitals_per_zip[hospitals_per_zip['counts'] >= 1]
# rename for merge
hospitals_per_zip_1 = hospitals_per_zip_1.rename(columns={'ZIP_CD': 'ZCTA5'})
# merge two dataset
zips_withhospital_centroids = zips_texas_borderstates_centroids.merge(
    hospitals_per_zip_1, on='ZCTA5', how='inner')
```

I decide to do a inner merger, which only keeps the zipcode areas that have at least 1 hospitals and also located in in Texas or a bordering state, which menas the zipcodes that appear in both dataset. I am merging based on 'ZCTA5', which is the zipcode.

4.

```
from shapely.geometry import Polygon, Point, MultiPoint
from datetime import datetime
```

a.

```
subset = zips_texas_centroids.head(10)

# calculate time for the for loop of measuring distance for subset
start = datetime.now()
distances = []
for index, row in subset.iterrows():
    nearest_hospital = zips_withhospital_centroids.distance(
        row['centroid']).min()
```

```
    distances.append((row['ZCTA5'], nearest_hospital))
end = datetime.now()
print('Duration:{}'.format(end-start))

# Referencing stackoverflow for for loop by rows, https://stackoverflow.com/questions
# /16476924/how-can-i-iterate-over-rows-in-a-pandas-dataframe/55557758#55557758
# Searching how to do for loop row by row
# StackOverflow Referencing for calculating distance, searching distance to centroid:
# https://stackoverflow.com/questions/30740046/calculate-distance-to-nearest-feature-with-geo
```

```
Duration:0:00:00.132642
```

```
/var/folders/r5/j3b0xjqs7h9bw5yxznft8cl00000gn/T/ipykernel_12263/90401724.py:7: UserWarning:

  nearest_hospital = zips_withhospital_centroids.distance(
```

```
# estimate time for full dataset
number_in_ziptx = len(zips_texas_centroids)
number_in_ziptx
total_duration = number_in_ziptx/10 * (end-start)
total_duration
```

```
datetime.timedelta(seconds=25, microseconds=666227)
```

The duration for running the subset is around 0.5 seconds. The approximated entire procedure for full dataset will be around 105 seconds.

b.

```
# calculate time for the for loop of measuring distance for full dataset
start2 = datetime.now()
distances_full = []
for index, row in zips_texas_centroids.iterrows():
    nearest_hospital = zips_withhospital_centroids.distance(
        row['centroid']).min()
    distances_full.append((row['ZCTA5'], nearest_hospital))
end2 = datetime.now()
print('Duration for full data for :{}'.format(end2-start2))
```

```
/var/folders/r5/j3b0xjqs7h9bw5yxznft8cl00000gn/T/ipykernel_12263/2546944163.py:5: UserWarning

  nearest_hospital = zips_withhospital_centroids.distance(

Duration for full data for :0:00:25.418426
```

```
# turn the outcome of for loop to a dataset
distances_full = pd.DataFrame(distances_full, columns=['ZCTA5', 'Distance'])

# Refencing ChatGPT for separating strings to df
```

The time for calculating the distance for full dataset is around 100 seconds, which is quite close to the previous estimation.

c.

```
with open(
    "/Users/boyalin/Documents/GitHub/ppha30538_ps/problem-set-4-boya-zidan/gz_2010_us_860_00_
) as prj_file:
    contents = prj_file.read()
    print(contents)

# Referencing ChatGPT for reading prj files
```

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

```
# transfering units
distances_full["distance_miles"]= distances_full["Distance"] *69
# Referencing https://www.usgs.gov/faqs/how-much-distance-does-a-degree-minute-
# and-second-cover-your-maps for changing degree to miles.
```

5.

```
# calculating average distance in miles
average_distance = distances_full['distance_miles'].mean()
print(f'Average distance to the nearest hospital: {average_distance}')
```

```
Average distance to the nearest hospital: 2.962006000084911
```

a.

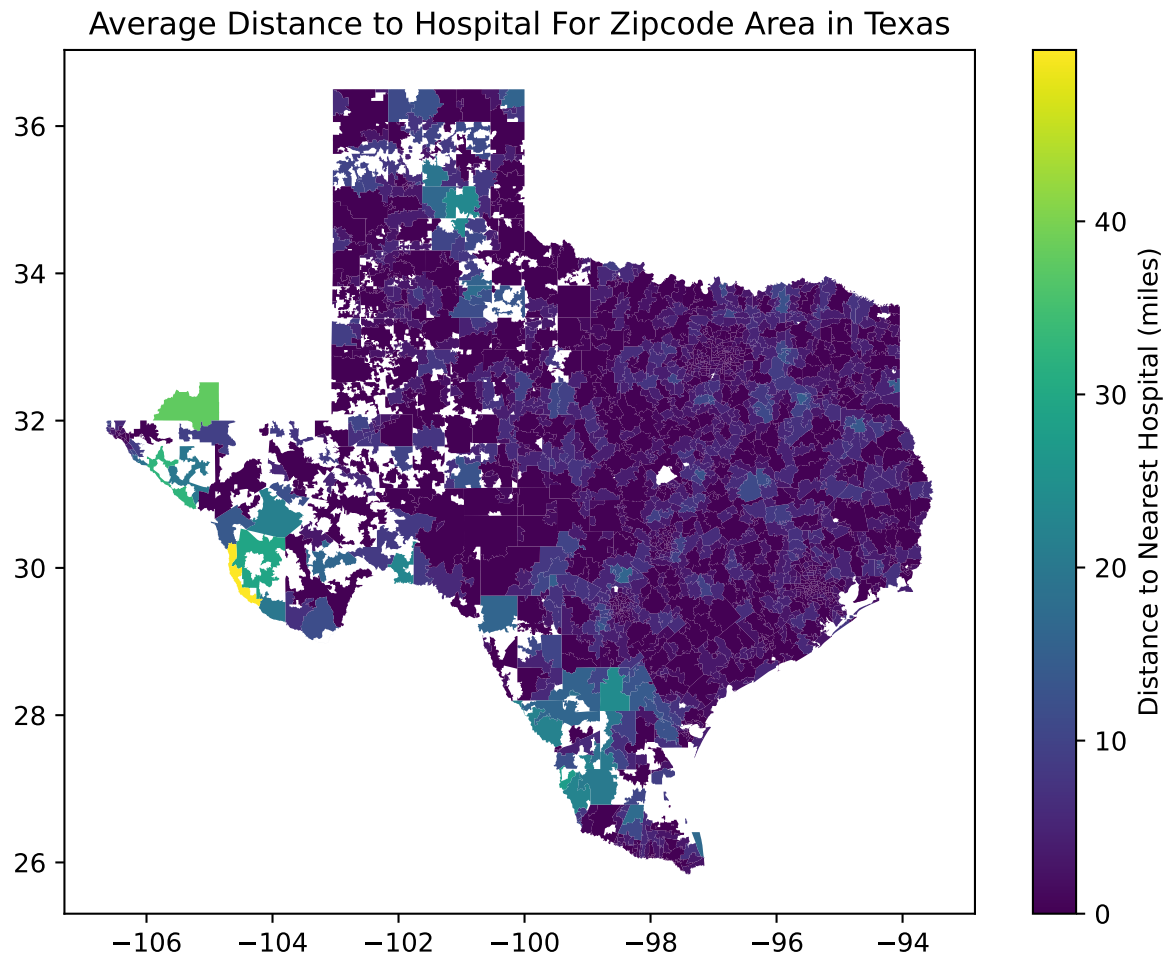
The average distance is calculated in miles.

b.

The average distance to the nearest hospital in Texas is 2.962 miles. This number makes sense. It represents each zip code area is quite closed to hospital on average, which indicates a good healthcare accessibility in Texas.

c.

```
# merge for ploting graphs
texas_distance = zips_texas_centroids.merge(
    distances_full, on='ZCTA5', how='left')

fig, ax = plt.subplots(1, 1, figsize=(8, 6))
texas_distance.plot(column='distance_miles', ax=ax, legend=True, legend_kwds={
    'label': "Distance to Nearest Hospital (miles)"})
ax.set_title('Average Distance to Hospital For Zipcode Area in Texas',
             fontdict={'fontsize': '12'})
plt.show()
```



Effects of closures on access in Texas (15 pts)

1.

```
# find zips codes for hospital closures in Texas
texas_closures = corrected_closures[corrected_closures['ZIP_CD'].astype(
    str).str.startswith(('733', '75', '76', '77', '78', '79'))]
# group by zip codes and count closures for each
zip_closure_counts = texas_closures.groupby(
    'ZIP_CD').size().reset_index(name='Closure_Counts')
print(zip_closure_counts.head(10))

# make a summary table for number of closures and number of zip codes
```

```
zip_closure_summary = zip_closure_counts[
    'Closure_Counts'].value_counts().reset_index()
zip_closure_summary.columns = ['Number of Closures', 'Number of ZIP Codes']
print(zip_closure_summary)
# ChatGDP reference for displaying a table summarizing count values
```

```
    ZIP_CD  Closure_Counts
0  75042.0               1
1  75051.0               1
2  75087.0               1
3  75140.0               1
4  75235.0               1
5  75390.0               1
6  75601.0               1
7  75662.0               1
8  75835.0               1
9  75862.0               1
   Number of Closures  Number of ZIP Codes
0                   1                   28
```

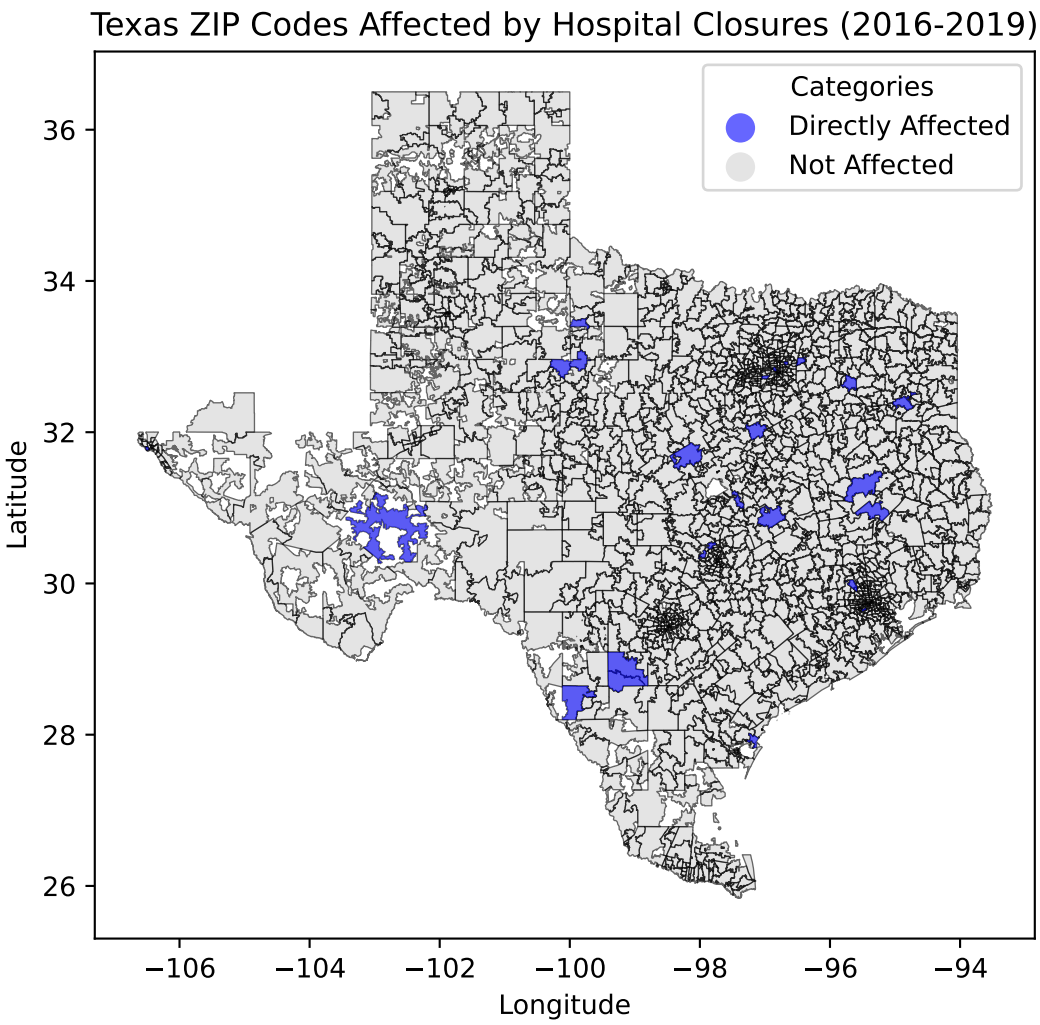
2.

```
# change the data type for ZIP_CD columns in texas_closures
texas_closures['ZIP_CD'] = texas_closures['ZIP_CD'].astype(
    str).str.split('.').str[0].str.strip()
# ChatGDP reference for extracting the integer part from float data
# and converting it to a string

# find unique Texas zip codes and count for that
affected_zip_codes = texas_closures['ZIP_CD'].unique()
affected_texas_zips = texas_distance[texas_distance['ZCTA5'].isin(
    affected_zip_codes)]
print(f'Number of directly affected ZIP codes: {len(affected_zip_codes)}')
```

```
Number of directly affected ZIP codes: 28

/var/folders/r5/j3b0xjqs7h9bw5yxznft8cl00000gn/T/ipykernel_12263/1660976515.py:2: SettingWitl
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
  texas_closures['ZIP_CD'] = texas_closures['ZIP_CD'].astype(
```

```
# plot the choropleth
fig, ax = plt.subplots(1, 1, figsize=(8, 6))
texas_distance.plot(ax=ax, legend=True, column='distance_miles',
                    color='lightgrey', alpha=0.6, edgecolor='black', linewidth=0.5)

affected_texas_zips.plot(ax=ax, color='blue', alpha=0.6)
# ChatGDP reference for formating choropleth

ax.set_title(
    'Texas ZIP Codes Affected by Hospital Closures (2016-2019)', fontsize=12)
ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')

# add legend for clarity
plt.scatter([], [], color='blue', label='Directly Affected', s=100, alpha=0.6)
plt.scatter([], [], color='lightgrey', label='Not Affected', s=100, alpha=0.6)
ax.legend(title='Categories')
# ChatGDP reference for adding legend to the plot

plt.show()
```

```
/var/folders/r5/j3b0xjqs7h9bw5yxznft8cl00000gn/T/ipykernel_12263/1323109098.py:3: UserWarning
  texas_distance.plot(ax=ax, legend=True, column='distance_miles',
```



3.

```
# create a GeoDataFrame of the directly affected zip codes
directly_affected_zips = affected_texas_zips.copy()
directly_affected_zips = gpd.GeoDataFrame(
    directly_affected_zips, geometry='geometry')

# Convert to the appropriate CRS (EPSG:3857)
texas_distance = texas_distance.to_crs(epsg=3857)
directly_affected_zips = directly_affected_zips.to_crs(epsg=3857)
# ChatGDP reference for ensuring the units of datasets to be consistent

# Create a 10-mile buffer around (1 mile = 1609.34 meters)
```

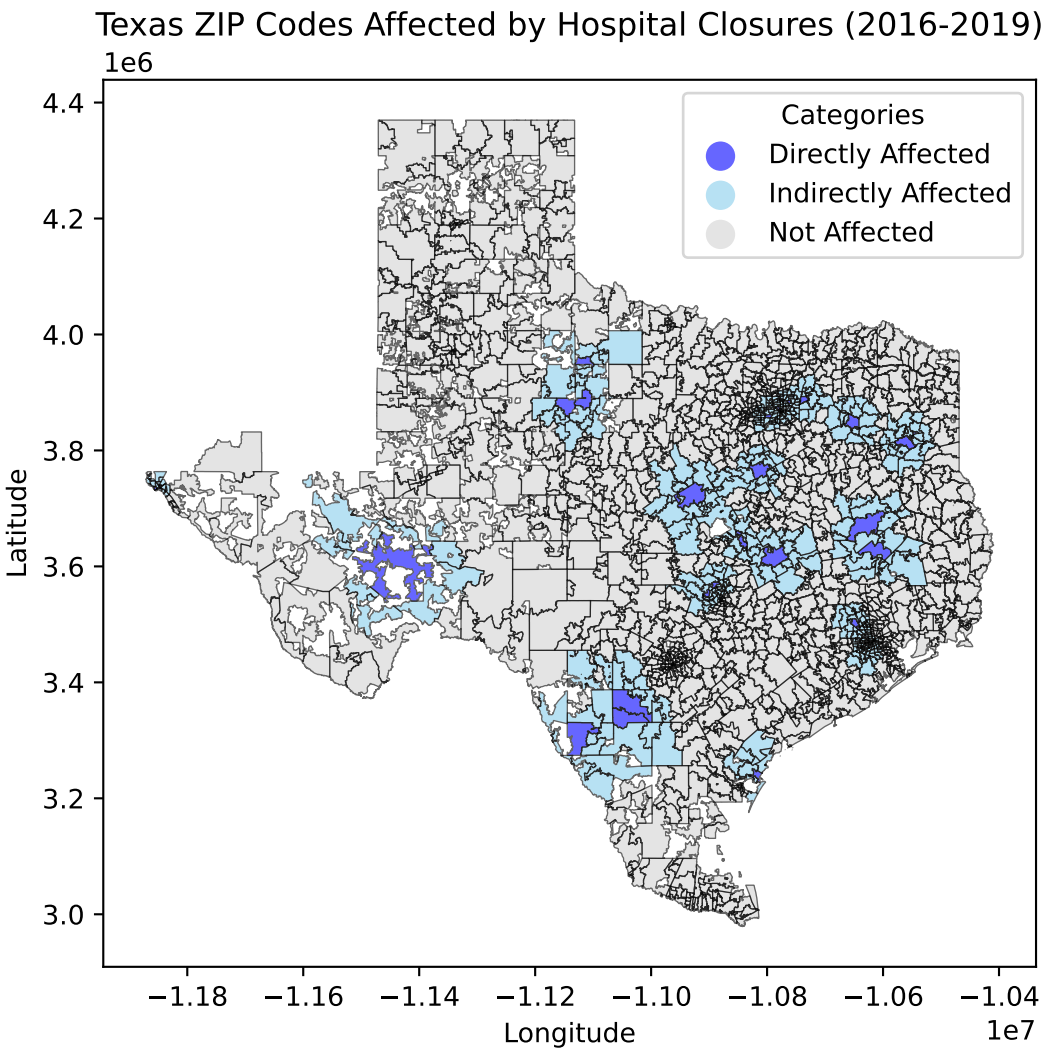
```
buffer_distance = 10 * 1609.34
directly_affected_zips['buffer'] = directly_affected_zips.geometry.buffer(
    buffer_distance)

buffered_affected_zips = gpd.GeoDataFrame(
    directly_affected_zips, geometry='buffer')

# perform spatial join with overall Texas zip codes
indirectly_affected_zips = gpd.sjoin(
    texas_distance, buffered_affected_zips,
    how='inner', predicate='intersects')
# StackOverflow reference for specific usage of gpd.sjoin:
# https://stackoverflow.com/questions/68504942/how-do-geopandas-sjoin-predicate
# -within-and-intersects-differ

indirectly_affected_zips = indirectly_affected_zips.rename(
    columns={'ZCTA5_left': 'ZCTA5',
             'ZCTA5_right': 'buffered_ZCTA5'})
# ChatGDP reference for changing column names

num_indirectly_affected_zips = indirectly_affected_zips['ZCTA5'].nunique()
print(
    f'Total number of indirectly affected ZIP codes: {num_indirectly_affected_zips}')
```

```
Total number of indirectly affected ZIP codes: 505
```

4.

```
def categorize_zip(zip_code):
    """ function for categoriing zip codes """
    if zip_code in directly_affected_zips['ZCTA5'].values:
        return 'Directly Affected'
    elif zip_code in indirectly_affected_zips['ZCTA5'].values:
        return 'Indirectly Affected'
    else:
        return 'Not Affected'

# apply the function
texas_distance['category'] = texas_distance['ZCTA5'].apply(categorize_zip)

# setting up color mapping for different categories
color_mapping = {
```

```
    'Directly Affected': 'blue',
    'Indirectly Affected': 'skyblue',
    'Not Affected': 'lightgrey'
}
texas_distance['color'] = texas_distance['category'].map(color_mapping)
# ChatGDP reference for setting up color mapping for three categories in my data

# plot the choropleth
fig, ax = plt.subplots(1, 1, figsize=(8, 6))
texas_distance.plot(ax=ax, legend=True,
                    color=texas_distance['color'], alpha=0.6,
                    edgecolor='black', linewidth=0.5)

ax.set_title(
    'Texas ZIP Codes Affected by Hospital Closures (2016-2019)', fontsize=12)
ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
# ChatGDP reference for making a Plot showing each category with its assigned color

plt.scatter([], [], color='blue', label='Directly Affected', s=100, alpha=0.6)
plt.scatter([], [], color='skyblue',
            label='Indirectly Affected', s=100, alpha=0.6)
plt.scatter([], [], color='lightgrey', label='Not Affected', s=100, alpha=0.6)
ax.legend(title='Categories')
# ChatGDP reference for adding legend to the plot

plt.show()
```



Reflecting on the exercise (10 pts)

1.

We currently classify hospitals as suspected closures if they were active in 2016 but do not appear in subsequent years. This assessment relies on a left-only merge to identify hospitals that have disappeared from the dataset, along with the status of the PGM_TRMNTN_CD and potential mergers or acquisitions. However, this approach still has some limitations.

Missing data in any given year can lead to misclassification of hospital closures. Hospitals may also change ownership without a name change, necessitating careful examination of the

ownership change column. Moreover, if a hospital frequently change its names due to various reasons, it can further complicate analysis and inflate closure estimates. More importantly, our reliance on short-term data does not account for hospitals that may reopen or relocate.

To do a better job in confirming hospital closures, we should cross-check additional columns, such as the effective dates of previous ownership changes, and utilize resources like official reports or regional hospital updates. Additionally, if feasible, conducting longitudinal studies will also help us monitor changes over time more effectively.

2.

The zipcodes that are in the zipcodes areas with hospital closed are directly affected since they have less hospital in their zipcode area. The measurement of indirectly affected through buffering based on distance might have some limitations. We set a 10-miles buffer for each directly affected zipcode area, without considering the population density and real world practice. There are might citizens living in a further area but we didn't include with a 10-miles buffer. Furthermore, we should consider the level of affected based on population in the area instead of just direct and indirect based on distance. Some zipcode areas might have more people, then the closure of hospital might have higher influence since more people are affected. We can improve the measurement by including the population density in each affected zipcode area. The buffer distance range should based on a actual geographical situation. For example, if there are citizens living in zipcode areas further away from the direct affected zip codes, we might consider to use a buffer with longer distance.