PS4

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): Yuna Baek / ybaek / 12370259
    - Partner 2 (name and cnet ID): Lizzy Diaz / lizzydiaz / endiaz
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: ***YB**, ***LD**
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_** (Yuna), **_1_** (Lizzy)
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

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

```
# Setup
import pandas as pd
import altair as alt
alt.renderers.enable("png")
import numpy as np
import time
import warnings
import os
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
```

1. Read through the rest of the problem set and look through the data dictionary to identify which variables you will need to complete the exercise, and use the tool on data.cms.gov into restrict to those variables ("Manage Columns") before exporting ("Export"). Download this for 2016 and call it pos2016.csv. What are the variables you pulled?

PRVDR_CTGRY_CD, PRVDR_CTGRY_SBTYP_CD, PRVDR_NUM, PGM_TRMNTN_CD, FAC_NAME, ZIP_CD

PRVDR_CTGRY_CD: Identifies the type of provider participating in the Medicare/Medicaid program. 01=Hospital

PRVDR_CTGRY_SBTYP_CD: Identifies the subtype of the provider, within the primary category. Used in reporting to show the breakdown of provider categories, mainly for hospital and SNFs. 01 = short term

PRVDR_NUM: Six or ten position identification number that is assigned to a certified provider. This is the CMS Certification Number.

PGM_TRMNTN_CD: Indicates the current termination status for the provider.

00=ACTIVE PROVIDER

01=VOLUNTARY-MERGER, CLOSURE

02=VOLUNTARY-DISSATISFACTION WITH REIMBURSEMENT

03=VOLUNTARY-RISK OF INVOLUNTARY TERMINATION

04=VOLUNTARY-OTHER REASON FOR WITHDRAWAL

05=INVOLUNTARY-FAILURE TO MEET HEALTH/SAFETY REQ

06=INVOLUNTARY-FAILURE TO MEET AGREEMENT

07=OTHER-PROVIDER STATUS CHANGE

08=NONPAYMENT OF FEES - CLIA Only

09=REV/UNSUCCESSFUL PARTICIPATION IN PT - CLIA Only

10=REV/OTHER REASON - CLIA Only

11=INCOMPLETE CLIA APPLICATION INFORMATION - CLIA Only

FAC_NAME: Name of the provider certified to participate in the Medicare and/or Medicaid programs.

ZIP_CD: Five-digit ZIP code for a provider's physical address.

2. We want to focus on short-term hospitals. These are identified as facilities with provider type code 01 and subtype code 01. Subset your data to these facilities. How many hospitals are reported in this data? Does this number make sense? Cross-reference with other sources and cite the number you compared it to. If it differs, why do you think it could differ?

```
# Change path variable to read in files from local
path = r"C:\Users\User\OneDrive - The University of Chicago\4_DAP-2\PS4 Data"
# path = "/Users/yunabaek/Desktop/3. Python II/problem-set-4-lizzy-yuna/"

def load_dataframes(path):
    pos_16 = pd.read_csv(os.path.join(path, 'pos2016.csv'),
 ↪  encoding='ISO-8859-1')
    pos_17 = pd.read_csv(os.path.join(path, 'pos2017.csv'),
 ↪  encoding='ISO-8859-1')
    pos_18 = pd.read_csv(os.path.join(path, 'pos2018.csv'),
 ↪  encoding='ISO-8859-1')
    pos_19 = pd.read_csv(os.path.join(path, 'pos2019.csv'),
 ↪  encoding='ISO-8859-1')
    return pos_16, pos_17, pos_18, pos_19

pos_16, pos_17, pos_18, pos_19 = load_dataframes(path)

# This is initially how we read answered question 1. We changed it to the
 ↪  above code to make switching between partners much easier.
# path = "/Users/yunabaek/Desktop/3. Python II/problem-set-4-lizzy-yuna/"
# pos_16 = pd.read_csv("pos2016.csv")
pos_16.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 141557 entries, 0 to 141556
Data columns (total 6 columns):
```

```
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   PRVDR_CTGRY_SBTYP_CD  140550 non-null  float64
 1   PRVDR_CTGRY_CD        141557 non-null  int64
 2   FAC_NAME              141557 non-null  object
 3   PRVDR_NUM             141557 non-null  object
 4   PGM_TRMNTN_CD         141557 non-null  int64
 5   ZIP_CD                141287 non-null  float64
dtypes: float64(2), int64(2), object(2)
memory usage: 6.5+ MB
```

```
pos_16 = pos_16[(pos_16['PRVDR_CTGRY_SBTYP_CD'] == 1) &
 ↪  (pos_16['PRVDR_CTGRY_CD'] == 1)]

unique_fac_names = len(pos_16['FAC_NAME'].unique())
number_rows = len(pos_16)
print(f"There are {number_rows} rows and {unique_fac_names} unique
 ↪  hospitals.")
```

```
There are 7245 rows and 6770 unique hospitals.
```

a. There are 6,770 hospitals reported in the data when subsetted by hospital name. This seems like an overestimate, which could be because the data have redundancies (the same hospital under different names, for example).

b. The issue brief from KFF states that there are nearly 5000 short-term hospitals acute care hospitals in the United States (Wishner et al., 2016). This difference may be due to the different definitions in short-term hospitals. It could also be due to differences in when the data were reported.

3. Repeat the previous 3 steps with 2017Q4, 2018Q4, and 2019Q4 and then append them together. Plot the number of observations in your dataset by year.

Loading 17-19 data (Note: This is the code we originally wrote. Changed in favor of code written above in 2a).

```
# pos_17 = pd.read_csv("pos2017.csv")
# encountered error for 2018, 2019: 'utf-8' codec can't decode byte 0x98 in
 ↪  position 11050: invalid start byte
# pos_18 = pd.read_csv("pos2018.csv", encoding='ISO-8859-1')
# pos_19 = pd.read_csv("pos2019.csv", encoding='ISO-8859-1')
```

Subsetting data

```
pos_17 = pos_17[(pos_17['PRVDR_CTGRY_SBTYP_CD'] == 1) &
 ↪  (pos_17['PRVDR_CTGRY_CD'] == 1)]

pos_18 = pos_18[(pos_18['PRVDR_CTGRY_SBTYP_CD'] == 1) &
 ↪  (pos_18['PRVDR_CTGRY_CD'] == 1)]

pos_19 = pos_19[(pos_19['PRVDR_CTGRY_SBTYP_CD'] == 1) &
 ↪  (pos_19['PRVDR_CTGRY_CD'] == 1)]
```

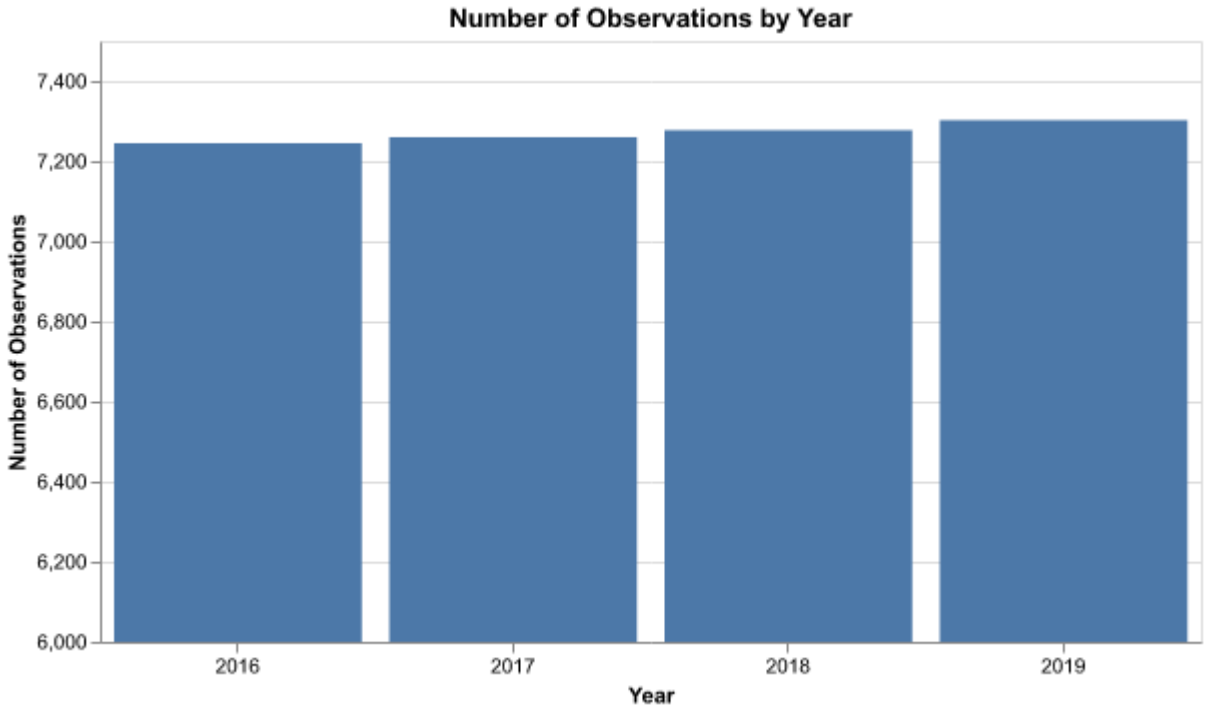
Appending data

```
pos_16['Year'] = 2016
pos_17['Year'] = 2017
pos_18['Year'] = 2018
pos_19['Year'] = 2019
pos_df = pd.concat([pos_16, pos_17, pos_18, pos_19])
pos_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 29085 entries, 0 to 139518
Data columns (total 7 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   PRVDR_CTGRY_SBTYP_CD  29085 non-null  float64
 1   PRVDR_CTGRY_CD        29085 non-null  int64
 2   FAC_NAME              29085 non-null  object
 3   PRVDR_NUM             29085 non-null  object
 4   PGM_TRMNTN_CD         29085 non-null  int64
 5   ZIP_CD                29085 non-null  float64
 6   Year                  29085 non-null  int64
dtypes: float64(2), int64(3), object(2)
memory usage: 1.8+ MB
```

```
obs_df = pos_df['Year'].value_counts().reset_index()
obs_df.columns = ['Year', 'Observations']

obs_plot = alt.Chart(obs_df).mark_bar().encode(
    x=alt.X('Year:O', title='Year',
    axis=alt.Axis(labelAngle=0)),
```

```
    y=alt.Y('Observations:Q', title = 'Number of Observations',
    scale=alt.Scale(domain=[6000, 7500])),
).properties(
    title= 'Number of Observations by Year',
    width=550,
    height=300
).configure_scale(
    clamp=True
)

obs_plot
```

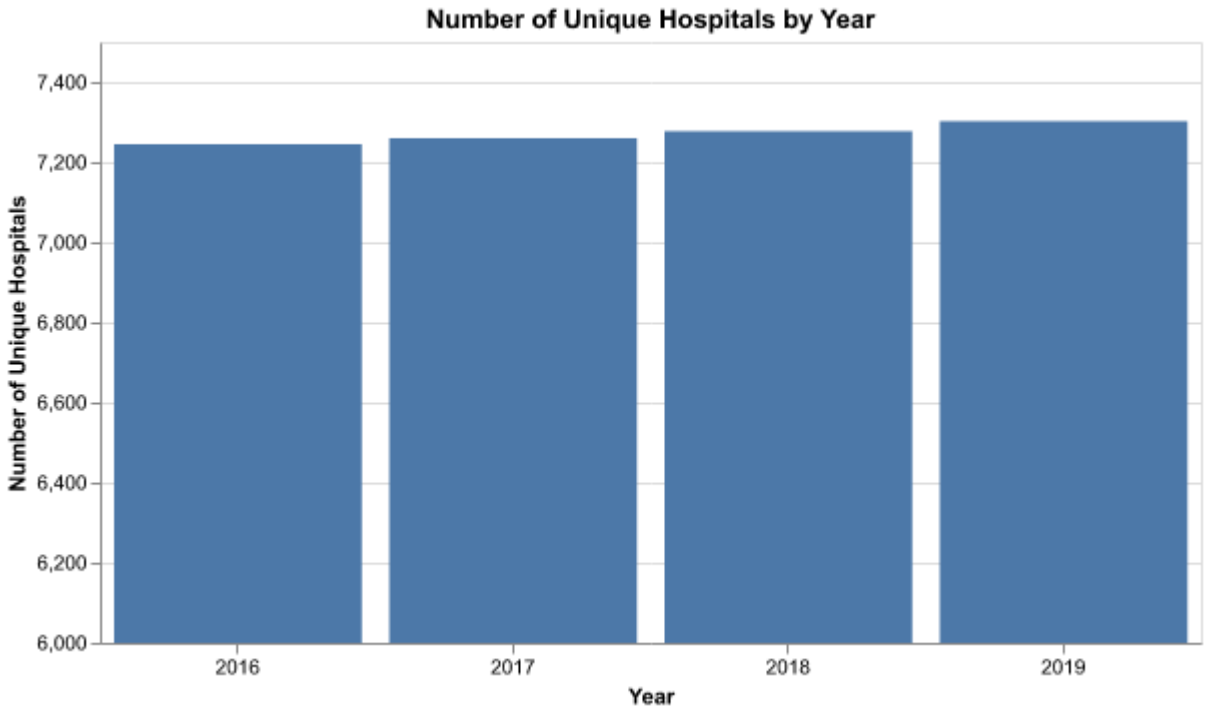


4. Each hospital is identified by its CMS certification number.

    a. Plot the number of unique hospitals in your dataset per year.

```
cms_df = pos_df.groupby('Year')['PRVDR_NUM'].nunique().reset_index()
cms_df.columns = ['Year', 'Unique_CMS']

cms_plot = alt.Chart(cms_df).mark_bar().encode(
    x=alt.X('Year:O', title='Year',
```

```
    axis=alt.Axis(labelAngle=0)),
    y=alt.Y('Unique_CMS:Q', title = 'Number of Unique Hospitals',
        scale=alt.Scale(domain=[6000, 7500]))
).properties(
    title= 'Number of Unique Hospitals by Year',
    width=550,
    height=300
).configure_scale(
    clamp=True
)
cms_plot
```



b. Compare this to your plot in the previous step. What does this tell you about the structure of the data?

The two plots are almost identical, suggesting that the data structure is consistent. There is a stable number of unique records per hospital over the years, indicating that the dataset effectively captures hospital activities without significant duplication. This implies that the dataset is likely well-maintained. The slight upward trend across both plots may reflect real growth in the number of hospitals or an increase in healthcare activity over the years. It may be useful to explore external factors influencing this trend, such as policy changes or demographic shifts.

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

1. In this question, I am treating each provider number as a unique hospital. This is because I ran into an issue where some hospitals, for each year, had both a 0 and 1 termination code, and different provider numbers. Using the provided definition, there were 174 hospitals open in 2016 but closed by 2019.

Sources: https://learnpython.com/blog/python-set-operations/

```
# Find active hospitals in 2016 (PGM_TRMNTN_CD==0)
active_2016 = pos_df[(pos_df['Year'] == 2016) & (pos_df['PGM_TRMNTN_CD'] ==
↪  0)]['PRVDR_NUM'].unique()

# Filter hospitals that have non-zero PGM_TRMNTN_CD in 2017, 2018, or 2019
closed_hospitals = pos_df[(pos_df['PRVDR_NUM'].isin(active_2016)) &
    (pos_df['Year'].isin([2017, 2018, 2019])) &
    (pos_df['PGM_TRMNTN_CD'] != 0)]

# Find hospitals present in data in 2016, but not in data after that
present_after_2016 = pos_df[pos_df['Year'].isin([2017, 2018,
↪  2019])]['PRVDR_NUM'].unique()
potential_closures = set(active_2016) - set(present_after_2016)
print(potential_closures)
# empty, implying no hospitals are in the data that are only 2016
```

```
set()
```

```
# Find suspected year of closure
year_closed =
↪  closed_hospitals.groupby('PRVDR_NUM')['Year'].min().reset_index()
year_closed.columns = ['PRVDR_NUM', 'YR_CLOSED']

suspected_closed = pos_df[pos_df['PRVDR_NUM'].isin(year_closed['PRVDR_NUM'])]
# drop duplicate hospitals, by PRVDR_NUM
suspected_closed = suspected_closed.drop_duplicates(subset=['PRVDR_NUM'])
# Merge with suspected year of closure
suspected_closed = suspected_closed.merge(year_closed, on="PRVDR_NUM")
# keep only desired columns
suspected_closed = suspected_closed[['FAC_NAME', 'ZIP_CD', 'YR_CLOSED']]
suspected_closed.head(3)

print(f'There are {len(suspected_closed)} hospitals that fit this
↪  definition.')
```

```
There are 174 hospitals that fit this definition.
```

2.

```
suspected_closed[['FAC_NAME',
↪  'YR_CLOSED']].sort_values(by=['FAC_NAME']).head(10)
```

	FAC_NAME	YR_CLOSED
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

3. However, not all suspected hospital closures are true closures. For example, in the case of a merger, a CMS certification number will be appear to be "terminated," but then the hospital re-appear under a similar name/address with a new CMS certification number in the next year. As a first pass to address this, remove any suspected hospital closures that are in zip codes where the number of active hospitals does not decrease in the year after the suspected closure.

Sources: https://www.geeksforgeeks.org/apply-function-to-every-row-in-a-pandas-dataframe/

https://discovery.cs.illinois.edu/guides/DataFrame-Fundamentals/run-a-function-on-rows-dataframe/

https://flexiple.com/python/if-not-python

ChatGPT: I couldn't get the function to run row-wise properly. It suggested to alter my code to assign zip_code and year to their own variables. I also used it to figure out how to properly extract the number of hospitals in year_before and year_after (year_after['Hospital_Count'].values[0]).

```
active_hospitals = pos_df[pos_df['PGM_TRMNTN_CD'] == 0]
hospitals_per_year = active_hospitals.groupby(['ZIP_CD',
 ↪  'Year']).size().reset_index(name='Hospital_Count')
hospitals_per_year['Diff_Count'] =
 ↪  hospitals_per_year.groupby('ZIP_CD')['Hospital_Count'].diff()

def check_decrease(row):
    zip_code = row['ZIP_CD']
    year = row['YR_CLOSED']
    year_before = hospitals_per_year[(hospitals_per_year['ZIP_CD'] ==
 ↪  zip_code) & (hospitals_per_year['Year'] == year)]
    year_after = hospitals_per_year[(hospitals_per_year['ZIP_CD'] ==
 ↪  zip_code) & (hospitals_per_year['Year'] == year + 1)]

    if not year_before.empty and not year_after.empty:
        return year_after['Hospital_Count'].values[0] <
         ↪  year_before['Hospital_Count'].values[0]
    return True

# Apply check_decrease to suspected_closed and filter
filtered_suspected_closed =
 ↪  suspected_closed[suspected_closed.apply(check_decrease, axis=1)]
```

a.

```
filtered_closed_names = filtered_suspected_closed['FAC_NAME']
merger_acquisition =
 ↪  suspected_closed[~suspected_closed['FAC_NAME'].isin(filtered_closed_names)]

print(f'Among the suspected closures, {len(merger_acquisition)} are suspected
 ↪  to be mergers and acquisitions.')
```

```
Among the suspected closures, 27 are suspected to be mergers and
acquisitions.
```

b.

```
print(f'After correcting for this, we have {len(filtered_suspected_closed)}
 ↪  left defined as closures.')
```

```
After correcting for this, we have 147 left defined as closures.
```

c.

```
filtered_suspected_closed.sort_values(by=['FAC_NAME']).head(10)
```

	FAC_NAME	ZIP_CD	YR_CLOSED
4	ABRAZO MARYVALE CAMPUS	85031.0	2017
97	AFFINITY MEDICAL CENTER	44646.0	2018
140	ALLEGIANCE SPECIALTY HOSPITAL OF KILGORE	75662.0	2017
62	ALLIANCE LAIRD HOSPITAL	39365.0	2019
101	ALLIANCEHEALTH DEACONESS	73112.0	2019
26	ANNE BATES LEACH EYE HOSPITAL	33136.0	2019
21	ARKANSAS VALLEY REGIONAL MEDICAL CENTER	81050.0	2017
69	BANNER CHURCHILL COMMUNITY HOSPITAL	89406.0	2017
5	BANNER PAYSON MEDICAL CENTER	85541.0	2018
115	BARIX CLINICS OF PENNSYLVANIA	19047.0	2019

Download Census zip code shapefile (10 pt)

1. This is non-tabular data.

    a. What are the five file types and what type of information is in each file?

- Database file (dbf): attribute information for spatial features. Row = feature (point, line, etc.), column = specific attribute (name, etc.).
- Projection file (prj): describes Coordinate Reference System (CRS) of the shapefile.
- Shapefile (shp): feature geometrics data for spatial features, including coordinates.
- Shape index file (shx): positional index
- Extensible markup language (xml): metadata about spatial data

    b. It will be useful going forward to have a sense going forward of which files are big versus small. After unzipping, how big is each of the datasets?

```
import os

file_types = ['.dbf', '.prj', '.shp', '.shx', '.xml']
# directory = '/Users/yunabaek/Desktop/3. Python II/gz_2010_us_860_00_500k'
directory = r'C:\Users\User\OneDrive - The University of
 ↪  Chicago\4_DAP-2\gz_2010_us_860_00_500k'
for file in os.listdir(directory):
```

```
    if any(file.endswith(ext) for ext in file_types):
        file_path = os.path.join(directory, file)
        file_size = os.path.getsize(file_path)
        print(f"{file}: {file_size} bytes")
```

```
gz_2010_us_860_00_500k.dbf: 6425474 bytes
gz_2010_us_860_00_500k.prj: 165 bytes
gz_2010_us_860_00_500k.shp: 837544580 bytes
gz_2010_us_860_00_500k.shx: 265060 bytes
gz_2010_us_860_00_500k.xml: 15639 bytes
```

2. Load the zip code shapefile and restrict to Texas zip codes. (Hint: you can identify which state a zip code is in using the first 2-3 numbers in the zip code. Then calculate the number of hospitals per zip code in 2016 based on the cleaned POS file from the previous step. Plot a choropleth of the number of hospitals by zip code in Texas.

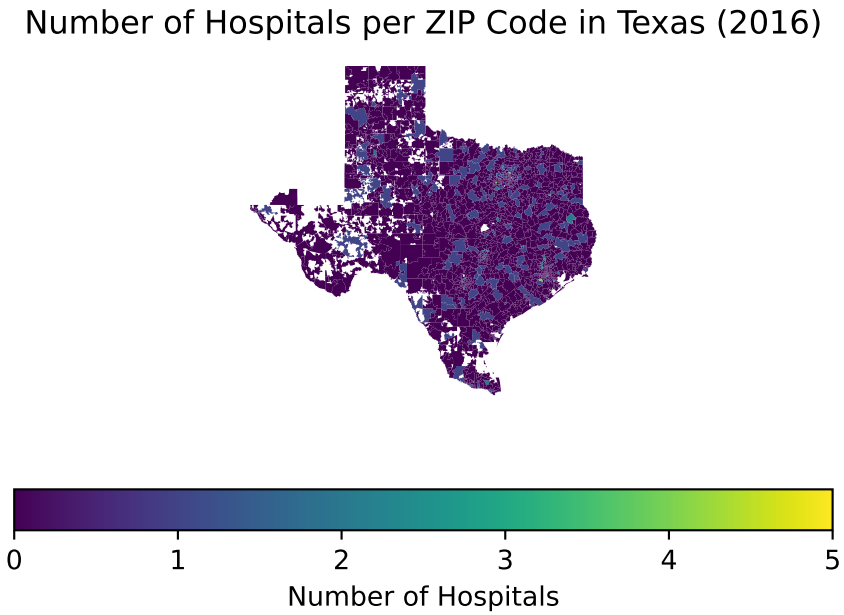
```
import geopandas as gpd
# import fiona
import pyogrio
shppath = r"C:\Users\User\OneDrive - The University of
 ↪  Chicago\4_DAP-2\gz_2010_us_860_00_500k"
# shppath = '/Users/yunabaek/Desktop/3. Python
 ↪  II/problem-set-4-lizzy-yuna/gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.shp'
data_shp = gpd.read_file(shppath)
texas_shp = data_shp[data_shp['ZCTA5'].str.startswith(('75', '76', '77',
 ↪  '78', '79'))]
```

```
# Filtering shapefile
# Converting into string
hospitals_per_year['ZIP_CD'] = hospitals_per_year['ZIP_CD'].astype(str)

# Subsetting for 2016
hospitals_2016 = hospitals_per_year[
    (hospitals_per_year['Year'] == 2016) &
    (hospitals_per_year['ZIP_CD'].str.startswith(('75', '76', '77', '78',
 ↪  '79')))
]

# Aggregating per zip-code count
hospitals_count =
 ↪  hospitals_2016.groupby('ZIP_CD')['Hospital_Count'].sum().reset_index()
hospitals_count.columns = ['ZCTA5', 'Number_of_Hospitals']
```

```
# Fixing variable types before merging
hospitals_count['ZCTA5'] =
 ↪  hospitals_count['ZCTA5'].astype(str).str.replace('.0', '').astype(int)
hospitals_count['ZCTA5'] = hospitals_count['ZCTA5'].astype(int)
texas_shp['ZCTA5'] = texas_shp['ZCTA5'].astype(int)
```

Source: https://geopandas.org/en/stable/docs/user_guide/mapping.html

```
# Merging the Texas shapefile with the hospital counts
texas_hospitals = texas_shp.merge(hospitals_count, on='ZCTA5', how='left')

# Fill NaN values with 0 (if any ZIP codes have no hospitals)
texas_hospitals['Number_of_Hospitals'].fillna(0, inplace=True)

texas_hospitals.plot(column='Number_of_Hospitals', legend=True,
                     cmap='viridis',
                     legend_kwds={'label': "Number of Hospitals",
                                  'orientation': "horizontal"},
                     vmin=0, vmax=5)

plt.title('Number of Hospitals per ZIP Code in Texas (2016)')
plt.axis('off')
plt.show()
```



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

1. Create a GeoDataFrame for the centroid of each zip code nationally: zips_all_centroids. What are the dimensions of the resulting GeoDataFrame and what do each of the columns mean?

- ZCTA5: The zip code
- geometry: The points representing the shape of each zip code.
- centroid: The centroid point coordinates, which are an arithmetic mean position of all the points in a polygon.

```
data_shp['centroid'] = data_shp['geometry'].centroid
zips_all_centroids = data_shp[['ZCTA5', 'geometry','centroid']]
zips_all_centroids.set_geometry('centroid')

print(f'The dimensions of the dataframe are: {zips_all_centroids.shape}')
```

```
The dimensions of the dataframe are: (33120, 3)
```

2. Create two GeoDataFrames as subsets of zips_all_centroids. First, create all zip codes in Texas: zips_texas_centroids.

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

print(f'There are {zips_texas_centroids['ZCTA5'].nunique()} unique zip codes
 ↪  in this subset.')
```

```
There are 1935 unique zip codes in this subset.
```

Then, create all zip codes in Texas or a bordering state: zips_texas_borderstates_centroids.

```
# Define zip code prefixes for Texas and neighboring states
prefixes = ([str(x) for x in range (75, 79)]  +
    [str(x) for x in range (716, 729)] +
    [str(x) for x in range (700, 715)] +
    [str(x) for x in range (870, 884)] +
    [str(x) for x in range (73, 74)]
)
zips_texas_borderstates_centroids =
 ↪  zips_all_centroids[zips_all_centroids['ZCTA5'].str.startswith(tuple(prefixes))]

print(f'There are {zips_texas_borderstates_centroids['ZCTA5'].nunique()}
 ↪  unique zip codes in this subset.')
```

```
There are 3350 unique zip codes in this subset.
```

3. Then create a subset of zips_texas_borderstates_centroids that contains only the zip codes with at least 1 hospital in 2016. Call the resulting GeoDataFrame zips_withhospital_centroids What kind of merge did you decide to do, and what variable are you merging on?

```
active_2016_df = pos_df[(pos_df['Year'] == 2016) & (pos_df['PGM_TRMNTN_CD']
 ↪  == 0)]
# Fix the zip code format
active_2016_df['ZIP_CD'] =
 ↪  active_2016_df['ZIP_CD'].astype(str).str.split('.').str[0]

zips_withhospital_centroids = zips_texas_borderstates_centroids.merge(
    active_2016_df[['ZIP_CD']],
    left_on='ZCTA5',
    right_on='ZIP_CD',
```

```
    how='inner'
)
```

I used an inner merge, merging on the zip code variables in the two datasets. This will give only the rows with matching zip codes in both datasets. The active_2016_df contains information for all active hospitals, so if there is a zip code in that dataset which matches a zip code in the Texas and bordering states dataset, it will appear in zips_withhospital_centroids.

4. For each zip code in zips_texas_centroids, calculate the distance to the nearest zip code with at least one hospital in zips_withhospital_centroids.

    a.

```
# Subset to 10 zip codes
zips_texas_subset = zips_texas_centroids[:10]
from timeit import default_timer as timer
start = timer()
nearest_hospital_subset = gpd.sjoin_nearest(
    zips_texas_subset,
    zips_withhospital_centroids,
    how='inner', distance_col="distance"
)
end = timer()
time = (end - start)

print(f'It took {time:.2f} seconds to run the subsetted merge.')
estimate_mins = (time*(zips_texas_centroids.shape[0]/10))/60
print(f'I estimate the entire procedure will take {estimate_mins:.2f}
 ↪  minutes.')
```

```
It took 5.21 seconds to run the subsetted merge.
I estimate the entire procedure will take 16.79 minutes.
```

b.

```
start = timer()
nearest_hospital = gpd.sjoin_nearest(
    zips_texas_centroids,
    zips_withhospital_centroids,
    how='inner', distance_col="distance"
)
end = timer()
```

```
time = (end - start)
print(f'It took {time/60:.2f} minutes to run the full merge.')
```

```
It took 12.60 minutes to run the full merge.
```

c. According to the .prj file, the units are in "degree." According to [this site](https://www.sco.wisc.edu/2022/01/21/how-big-is-a-degree/), 1 degree is approximately equal to 69.4 miles.

```
nearest_hospital['distance'] = nearest_hospital['distance']*69.4
```

5. Calculate the average distance to the nearest hospital for each zip code in Texas.

```
average_distance =
 ↪  nearest_hospital.groupby(['ZCTA5_left'])['distance'].mean().reset_index()
average_distance = average_distance.rename(columns={'ZCTA5_left':'zip'})
average_distance.head(3)
```

	zip	distance
0	75001	0.0
1	75002	0.0
2	75006	0.0

a. The units are in miles.

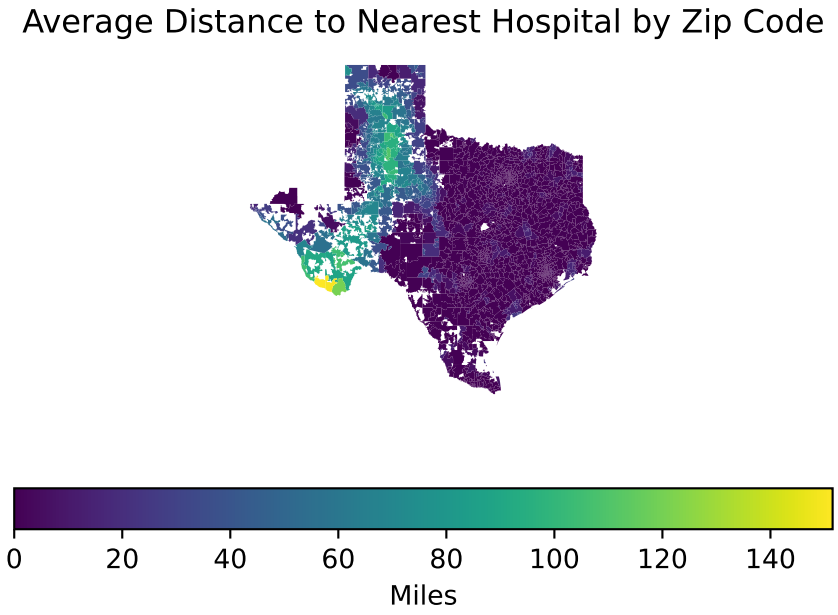
b.

```
print(f'The average distance to the nearest hospital for zip codes in \
    Texas is {nearest_hospital.loc[:,"distance"].mean():.2f} miles.')
```

```
The average distance to the nearest hospital for zip codes in     Texas is
6.85 miles.
```

c.

```
# Merge TX zip shapes with avg distance to hospitals
texas_hospitals_distance = zips_texas_centroids.merge(
    average_distance,
    left_on='ZCTA5',
    right_on='zip',
    how='outer'
)

texas_hospitals_distance.plot(column='distance', legend=True,
    cmap='viridis',
    legend_kwds={'label':'Miles',
    'orientation':'horizontal'}
)
plt.title('Average Distance to Nearest Hospital by Zip Code')
plt.axis('off')
plt.show()
```



Effects of closures on access in Texas (15 pts)

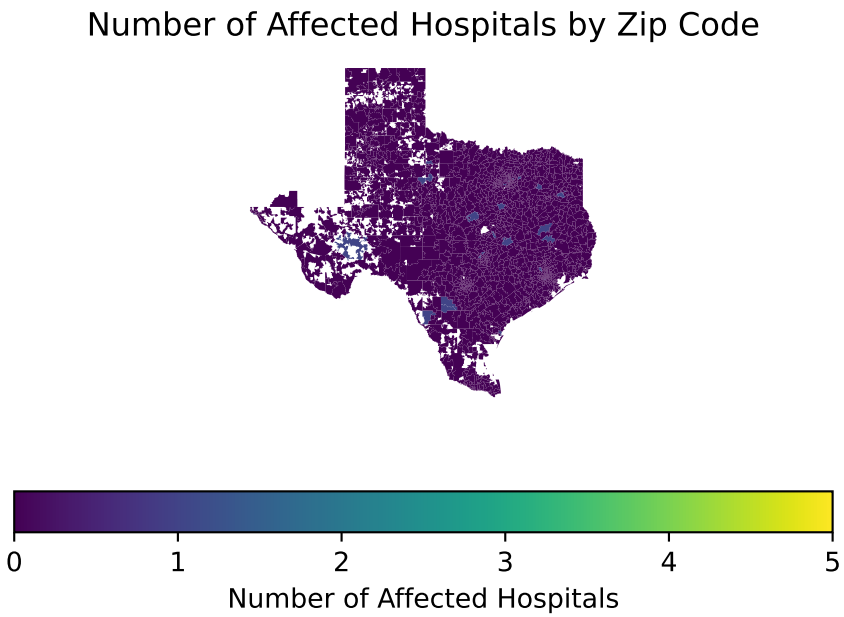
1. Using the corrected hospital closures dataset from the first section, create a list of directly affected zip codes in Texas – that is, those with at least one closure in 2016-2019. Display a table of the number of zip codes vs. the number of closures they experienced.

```
closure_df =
 ↪  pd.DataFrame(filtered_suspected_closed['ZIP_CD'].value_counts().reset_index())
closure_df.columns = ['ZCTA5', 'Count']
closure_df
```

	ZCTA5	Count
0	33136.0	2
1	36278.0	1
2	36265.0	1
3	85031.0	1
4	85541.0	1
...	...	...
141	78613.0	1
142	78017.0	1
143	76520.0	1
144	75087.0	1
145	75140.0	1

2. Plot a choropleth of which Texas zip codes were directly affected by a closure in 2016-2019 – there was at least one closure within the zip code. How many directly affected zip codes are there in Texas?

```
# Merging the Texas shapefile with the closure counts
texas_closure_df = texas_shp.merge(closure_df, on='ZCTA5', how='left')

# Fill NaN values with 0
texas_closure_df['Count'].fillna(0, inplace=True)

texas_closure_df.plot(column='Count', legend=True,
                      cmap='viridis',
                      legend_kwds={'label': "Number of Affected Hospitals",
                                   'orientation': "horizontal"},
                      vmin=0, vmax=5)

plt.title('Number of Affected Hospitals by Zip Code')
plt.axis('off')
plt.show()
```




```
# Count of directly affected zip codes
affected_zip_codes_count = len(texas_closure_df[texas_closure_df['Count'] >
 ↪  0])
print(f'There are {affected_zip_codes_count} zip codes in Texas that were
 ↪  directly affected.')
```

```
There are 28 zip codes in Texas that were directly affected.
```

3. Then identify all the indirectly affected zip codes: Texas zip codes within a 10-mile radius of the directly affected zip codes. To do so, first create a GeoDataFrame of the directly affected zip codes. Then create a 10-mile buffer around them. Then, do a spatial join with the overall Texas zip code shapefile. How many indirectly affected zip codes are there in Texas?

Attribution: https://geopandas.org/en/stable/docs/reference/api/geopandas.GeoDataFrame.set_geometry.html, ChatGPT to resolve error on .set_geometry

```
# Creating a category for affected directly / indirectly / not affected
def affected(df):
    df['affected'] = 'Not Affected'
    df.loc[df['Count'] > 0, 'affected'] = 'Directly Affected'
    return df

texas_affected_df = affected(texas_closure_df)
```

```
# Creating GeoDataFrame
affected_gdf = gpd.GeoDataFrame(
    texas_affected_df[texas_affected_df['Count'] > 0],
    geometry=texas_affected_df.geometry
)
# 10-mile (16093.4 meters) buffer
affected_gdf['buffer'] = affected_gdf.geometry.buffer(16093.4)
affected_gdf.set_geometry('buffer', inplace=True)
```

```
# Spatial join
indirectly_affected_gdf = gpd.sjoin(texas_shp, affected_gdf,
                                    how='inner', predicate='intersects')

# Count the number of indirectly affected zip codes
indirectly_affected_count = indirectly_affected_gdf['ZCTA5_left'].nunique()
print(f'Number of indirectly affected zip codes in Texas:
 ↪  {indirectly_affected_count}')
```

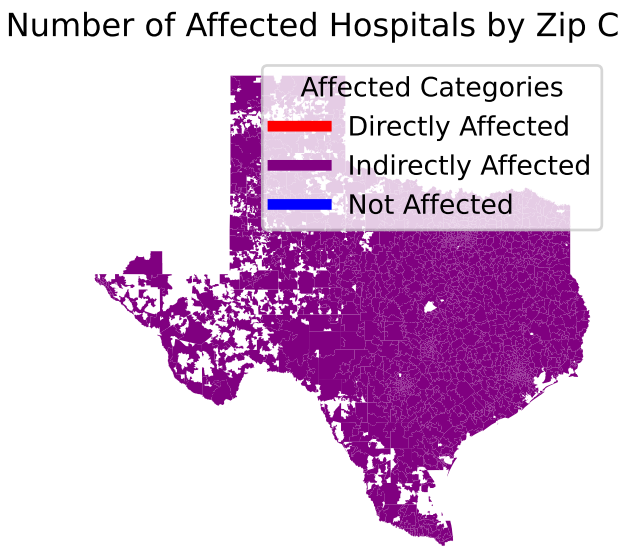
```
Number of indirectly affected zip codes in Texas: 1935
```

4. Make a choropleth plot of the Texas zip codes with a different color for each of the 3 categories: directly affected by a closure, within 10 miles of closure but not directly affected, or not affected.

```
# Adding indirectly affected category
indirectly_affected_zip_codes =
 ↪  indirectly_affected_gdf['ZCTA5_left'].unique()

def affected(df, indirectly_affected_zipcodes):
    df['affected'] = 'Not Affected'
    df.loc[df['Count'] > 0, 'affected'] = 'Directly Affected'
    df.loc[df['ZCTA5'].isin(indirectly_affected_zipcodes), 'affected'] =
 ↪  'Indirectly Affected'
    return df

texas_affected_df = affected(texas_closure_df, indirectly_affected_zip_codes)
```

Attribution: ChatGPT for fixing legend errors

```
color = {
    'Directly Affected': 'red',
    'Indirectly Affected': 'purple',
    'Not Affected': 'blue'
}

texas_affected_df['color'] = texas_affected_df['affected'].map(color)

texas_affected_df.plot(color=texas_affected_df['color'], legend=False)
plt.title('Number of Affected Hospitals by Zip Code')

handles = [
    plt.Line2D([0], [0], color='red', lw=4, label='Directly Affected'),
    plt.Line2D([0], [0], color='purple', lw=4, label='Indirectly Affected'),
    plt.Line2D([0], [0], color='blue', lw=4, label='Not Affected')
]

plt.legend(handles=handles, title="Affected Categories", loc='upper right')

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



Reflecting on the exercise (10 pts)

a. Some potential issues with identifying closures using this "first pass" method are:

- Using the facility name versus the CMS code: A hospital could have changed names but not their code, or vice versa. This could cause issues in correctly identifying and matching hospitals across subsetted data. We should be clear on how codes and names are assigned and can change through time. A hospital name change could inaccurately be interpreted as a closure.
- Location: A hospital might have closed in one zip code and re-opened nearby, but in another zip code. This would be reflected as a closure using our method, but in practice probably shouldn't be.
- The data could be skewed based on when information is collected. A merger occuring in one year might not show up until the next year's data is collected, and would be inaccurately interpreted.

We could do a better job at identifying hospital closures by ensuring that the CMS code does not change for a specific facility across time. If it does, we would want information on why. For example, if the code changed because the hospital switched ownership, we might incorrectly count a closure. We should cross-verify this information with other data sources.

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

Challenges of this measure includes data accuracy and reliance on a single source of information to determine hospital closures. Hospitals may be identified as closed (affected) due to mergers, or there may be issues with identifying and recording temporary closures. Additionally, the 10-mile radius would have different significance in urban vs. rural areas. The type and size of hospitals as well as the range of the service that hospitals cover would also need to be considered. We may be able to improve this measure by tracking long-term trends to measure peramnent closures, or adopt a modified buffer measures to reflect the population of each region.