PS4

Yuxuan Geng & George Wang

PS4: Due Sat Nov 2 at 5:00PM Central. Worth 100 points.

Style Points (10 pts)

Submission Steps (10 pts)

• Partner 1 (name and cnet ID): Georeg Wang, gwang613 • Partner 2 (name and cnet ID): Yuxuan Geng, yuxuan1123

This submission is our work alone and complies with the 30538 integrity policy. ***GW YG***

I have uploaded the names of anyone else other than my partner and I worked with on the problem set here.

Late coins used this pset: ***1*** Late coins left after submission: ***3***

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

1. The necessary variables are PRVDR_NUM, FAC_NAME, ZIP_CD, STATE_CD, PRVDR_CTGRY_CD, PRVDR_CTGRY_SBTYP_CD, PGM_TRMNTN_CD, TRMNTN_EXPRTN_DT, and CBSA_URBN_RRL_IND.

2. a.

```
import pandas as pd
import altair as alt
alt.renderers.enable("png")

# Load the 2016 POS data
pos_2016 = pd.read_csv('POS_File_Hospital_Non_Hospital_Facilities_Q4_2016.csv')

# Filter for short-term hospitals
short_term_hospitals = pos_2016[
  (pos_2016['PRVDR_CTGRY_CD'] == 1) &
  (pos_2016['PRVDR_CTGRY_SBTYP_CD'] == 1)
]
```

```
# Count the number
hospital_count = short_term_hospitals.shape[0]

hospital_count
```

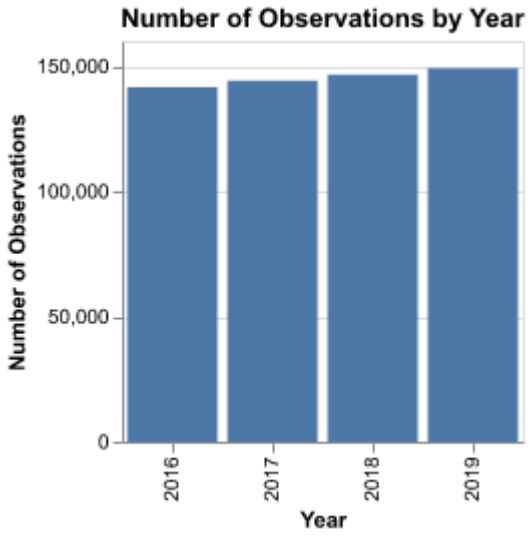
```
7245
```

b.

It does make sense because two datasets (AHA and CMS) might have different methodoligies and classifications. We used data from the American Hospital Association (AHA) for FY2016: https://www.aha.org/system/files/2018-01/Fast%20Facts%202018%20pie%20charts.pdf. The AHA data may exclude certain hospitals that are included in the CMS POS dataset, such as smaller facilities. In addition, the AHA chart is based on FY2016 data published in 2018, while the CMS POS file might have had updates during the same period.

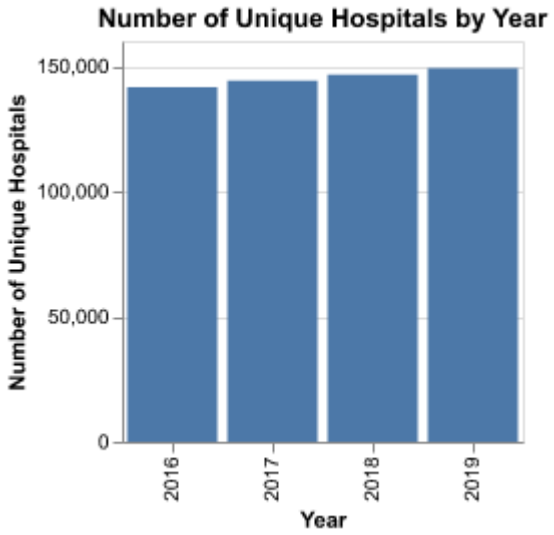
3.

```
# Load POS data from 2017 to 2019
pos_2016['Year'] = 2016

pos_2017 = pd.read_csv('POS_File_Hospital_Non_Hospital_Facilities_Q4_2017.csv', encoding='ISO-8859-1')
pos_2017['Year'] = 2017

pos_2018 = pd.read_csv('POS_File_Hospital_Non_Hospital_Facilities_Q4_2018.csv', encoding='ISO-8859-1')
pos_2018['Year'] = 2018

pos_2019 = pd.read_csv('POS_File_Hospital_Non_Hospital_Facilities_Q4_2019.csv', encoding='ISO-8859-1')
pos_2019['Year'] = 2019

# Appendix them
pos_all_years = pd.concat([pos_2016, pos_2017, pos_2018, pos_2019], ignore_index=True)

# Plot them using altair
import altair as alt

# Calculate the number of observations per year
yearly_counts = pos_all_years['Year'].value_counts().reset_index()
yearly_counts.columns = ['Year', 'Observations']
yearly_counts = yearly_counts.sort_values(by='Year')

# Plot using Altair
chart = alt.Chart(yearly_counts).mark_bar().encode(
    x=alt.X('Year:O', title='Year'),
    y=alt.Y('Observations:Q', title='Number of Observations'),
    tooltip=['Year', 'Observations']
).properties(
    title='Number of Observations by Year',
    width=200,
    height=200
)

chart
```



4. a.

```
# Calculate unique hospitals per year
unique_hospitals_per_year = pos_all_years.groupby('Year')['PRVDR_NUM'].nunique().reset_index()
unique_hospitals_per_year.columns = ['Year', 'Unique Hospitals']

# Create the Altair chart
unique_hospital_chart = alt.Chart(unique_hospitals_per_year).mark_bar().encode(
    x=alt.X('Year:O', title='Year'),
    y=alt.Y('Unique Hospitals:Q', title='Number of Unique Hospitals'),
    tooltip=['Year', 'Unique Hospitals']
).properties(
    title='Number of Unique Hospitals by Year',
    width=200,
    height=200
)

unique_hospital_chart
```



b.

Two plots are eseentially the same, showing that each row in the dataset likely corresponds to a unique hospital entry each year, even it is closed. It can be useful for analyzing hospital trends over time as no repeated hospital records within each year.

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

1.

```
# Filter active hospitals in 2016
active_2016 = pos_2016[pos_2016['PGM_TRMNTN_CD'] == 0]

# Initialize a list to collect suspected closures
closures = []

# Check each year suspected closures
for year in range(2017, 2020):
    # Check if the hospital exists and is active in the next year, merging based on PRVDR_NUM
    active_2016 = active_2016.merge(
        pos_all_years[pos_all_years['Year'] == year][['PRVDR_NUM', 'PGM_TRMNTN_CD']],
        on='PRVDR_NUM',
        how='left',
        suffixes=('', f'_{year}')
    ).reset_index(drop=True)

    # Identify hospitals that are no longer active
    suspected_closures = active_2016[
        (active_2016[f'PGM_TRMNTN_CD_{year}'] != 0)
    ]

    # Store those suspected closures
    suspected_closures['CLS_YR'] = year
```

```
    closures.append(suspected_closures[['PRVDR_NUM', 'FAC_NAME', 'ZIP_CD',
↪   'CLS_YR']].reset_index(drop=True))

# Combine all suspected closures and remove duplicates
closures_df = pd.concat(closures).drop_duplicates(subset='PRVDR_NUM')

# Count the number of unique suspected closures
num_suspected_closures = closures_df.shape[0]

# Display
num_suspected_closures
```

```
/var/folders/k2/prgbv7z97knbd104r93pncfc0000gp/T/ipykernel_52980/474959087.py:23: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/var/folders/k2/prgbv7z97knbd104r93pncfc0000gp/T/ipykernel_52980/474959087.py:23: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/var/folders/k2/prgbv7z97knbd104r93pncfc0000gp/T/ipykernel_52980/474959087.py:23: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

5492
```

2.

```
# Sort by FAC_NAME
sorted_closures_df = closures_df.sort_values(by='FAC_NAME').head(10)

# Display
sorted_closures_df[['FAC_NAME', 'CLS_YR']]
```

	FAC_NAME	CLS_YR
4076	(CLOSED) REFLECTIONS TREATMENT AGENCY	2019
5132	1ST CHOICE HEALTHCARE SERVICES INC	2019
1782	1ST CHOICE HOME HEALTH	2017
952	1ST FAMILY HOME HEALTHCARE, INC	2018

	FAC_NAME	CLS_YR
656	1ST HOME HEALTH CARE INC	2018
1249	21ST CENTURY HOME HEALTH AGENCY	2017
2390	247 HOME HEALTH CARE	2019
937	24SEVEN HEALTH CARE SERVICES, INC	2018
1430	25 EAST SAME DAY SURGERY CENTE	2019
1372	3 ANGELS HOME HEALTH	2019

3. a.

```
import pandas as pd

# Define a function to count active hospitals by ZIP code
def count_active_hospitals_by_zip(data, year):
    active_hospitals = data[data['PGM_TRMNTN_CD'] == 00]  # Assuming "01" indicates active
    return active_hospitals.groupby('ZIP_CD').size().rename(f'active_hospitals_{year}')

# Get active hospital counts by ZIP code
count_2016 = count_active_hospitals_by_zip(pos_2016, 2016)
count_2017 = count_active_hospitals_by_zip(pos_2017, 2017)
count_2018 = count_active_hospitals_by_zip(pos_2018, 2018)
count_2019 = count_active_hospitals_by_zip(pos_2019, 2019)

# Merge counts into a single DataFrame
active_counts = pd.DataFrame({
    '2016': count_2016,
    '2017': count_2017,
    '2018': count_2018,
    '2019': count_2019
}).fillna(0)

# Identify suspected mergers/acquisitions following guidence
merger_acquisitions = []
confirmed_closures = []

for index, row in closures_df.iterrows():
    zip_code = row['ZIP_CD']
    closure_year = row['CLS_YR']

    if closure_year < 2019:
        next_year = closure_year + 1
        if active_counts.loc[zip_code, str(next_year)] >= active_counts.loc[zip_code, str(closure_year)]:
            # If the number does not decrease, it's likely a merger/acquisition
            merger_acquisitions.append(row)
        else:
            # If the number decreases, confirm it as a closure
            confirmed_closures.append(row)

# Convert lists to DataFrames
merger_acquisitions_df = pd.DataFrame(merger_acquisitions)
confirmed_closures_df = pd.DataFrame(confirmed_closures)

# Drop duplicate if they have the same facility name and ZIP code
merger_acquisitions_df = merger_acquisitions_df.drop_duplicates(subset=['FAC_NAME', 'ZIP_CD'])
```

```
# Count and display
num_merger_acquisitions = merger_acquisitions_df.shape[0]
print("Number of Hospitals Potentially Closed due to Merger/Acquisition:", num_merger_acquisitions)
print("Potential Mergers/Acquisitions:\n", merger_acquisitions_df[['FAC_NAME', 'ZIP_CD', 'CLS_YR']])
```

```
Number of Hospitals Potentially Closed due to Merger/Acquisition: 2961
Potential Mergers/Acquisitions:
                              FAC_NAME   ZIP_CD  CLS_YR
0              GADSDEN REGIONAL HOSPICE  35901.0    2017
1               SOUTHERNCARE MONTGOMERY  36117.0    2017
2                  SOUTHERNCARE GADSDEN  35901.0    2017
3               SOUTHERNCARE GROVE HILL  36451.0    2017
4                SOUTHERNCARE DEMOPOLIS  36732.0    2017
...                                 ...      ...     ...
3735           SAINT DAVID HOSPICE, INC  91602.0    2018
3736                   SENIORS HOSPICE  91402.0    2018
3738  UNITED ALLIANCE HOSPICE CARE, INC  91506.0    2018
3739                A & A HOSPICE, INC  91401.0    2018
3740       CHARTWELL HOSPICE CARE, INC.  90241.0    2018

[2961 rows x 3 columns]
```

b.

```
# Drop duplicate entries from confirmed closures if they have the same facility name and ZIP code
confirmed_closures_df = confirmed_closures_df.drop_duplicates(subset=['FAC_NAME', 'ZIP_CD'])

# Count and display the number of confirmed closures after removing mergers/acquisitions
num_confirmed_closures = confirmed_closures_df.shape[0]
print("Number of Confirmed Closures after Removing Potential Mergers/Acquisitions:",
↪  num_confirmed_closures)
print("Confirmed Closures:\n", confirmed_closures_df[['FAC_NAME', 'ZIP_CD', 'CLS_YR']])
```

```
Number of Confirmed Closures after Removing Potential Mergers/Acquisitions: 764
Confirmed Closures:
                                          FAC_NAME   ZIP_CD  CLS_YR
6                            SOUTHERNCARE DOTHAN  36303.0    2017
15                        VALLEY HEAD CLINIC, LLC  35967.0    2017
18                            CHEAHA MH/MR CENTER  35160.0    2017
32                                GENTIVA HOSPICE  85711.0    2017
34                  GRACE HOSPICE OF ARIZONA, INC  85282.0    2017
...                                           ...      ...     ...
3661                  TEXAS SENIOR HOMEHEALTH INC  75252.0    2018
3683  MIRACLE HANDS HEALTHCARE SERVICES CORPORATION  77489.0    2018
3691                 HEFTY HEALTHCARE SERVICES INC  77074.0    2018
3693            ASSURANCE HEALTH CARE SERVICES INC  77071.0    2018
3710                         AVID HOME HEALTH INC  78754.0    2018

[764 rows x 3 columns]
```

c.

```
# Sort by facility name and show head
sorted_confirmed_closures = confirmed_closures_df.sort_values(by='FAC_NAME').head(10)

print(sorted_confirmed_closures[['FAC_NAME', 'ZIP_CD', 'CLS_YR']])
```

```
                                  FAC_NAME   ZIP_CD  CLS_YR
656                1ST HOME HEALTH CARE INC  33155.0    2018
937        24SEVEN HEALTH CARE SERVICES, INC  60659.0    2018
1537                      5 STAR HOSPICE LLC  84097.0    2017
3515     A & T MULTI-HEALTHCARE SERVICES LLC  77036.0    2018
2212                           A C L D, INC  11747.0    2018
1705           A&A RELIABLE HOME HEALTH CARE  55104.0    2018
1886  A-1 ADVANTAGE HOME HEALTH SERVICES INC  77027.0    2017
1856             AAA HEALTHCARE SERVICES INC  77036.0    2017
1202                 ABBLE HOME SERVICES LLC  43081.0    2017
1491                 ABICARE HOME HEALTH LLC  75234.0    2017
```

Download Census zip code shapefile (10 pt)

1. a. The .shp file has the shapes (geometry) of the features, such as points. The .shx file is an index that links these shapes to data stored in the .dbf file. The .dbf file has attribute data in a table format, with each row representing to a shape in .shp file. The .prj file has information of the map projection and coordinate system to display a map correctly. The .xml file has metadata, like the dataset's source and description.

   b. dbf: 6.4 MB prj: 165 bytes shp: 837.5 MB shx: 265 KB xml: 16 KB

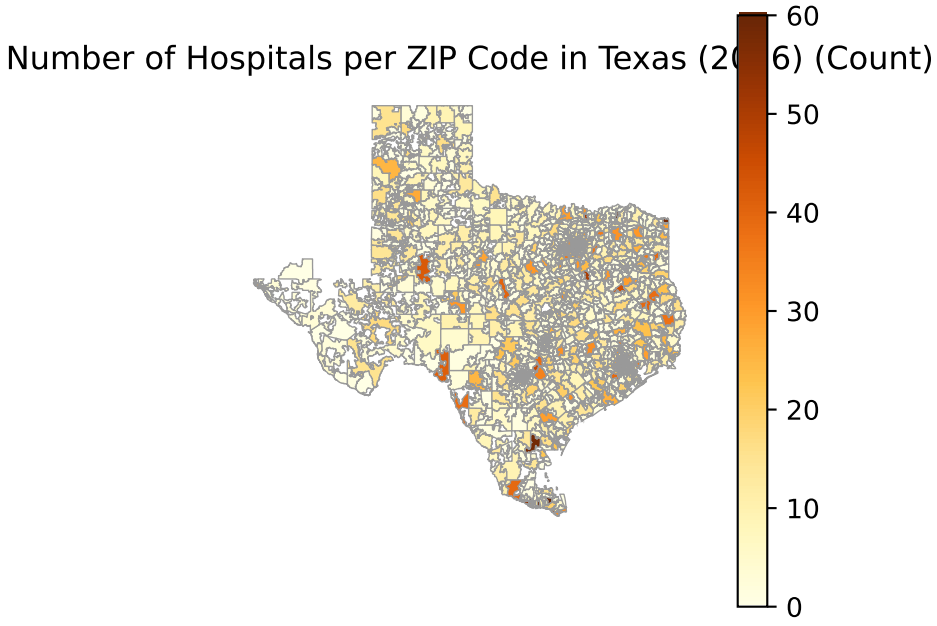
The .shp and .dbf files are the largest because they contain the main geometric and attribute data, respectively. The .prj, .shx, and .xml files are usually smaller.

2.

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

# Load the shapefile for ZIP code boundaries
shapefile_path = '/Users/georgew/Desktop/Fall 2024/PPHA
↪  30538/problem-set-4-yuxuan-george/gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.shp'
zip_codes = gpd.read_file(shapefile_path)

# Clean and convert ZIP codes in `zip_codes` to strings
zip_codes['ZCTA5'] = zip_codes['ZCTA5'].astype(str).str.replace(r'\.0$', '', regex=True)

# Filter ZIP codes for Texas
texas_zip = zip_codes[zip_codes['ZCTA5'].str.startswith(('75', '76', '77', '78', '79'))]

# Filter hospital data for Texas ZIP codes starting with specific prefixes
texas_hospital = pos_2016[(pos_2016['STATE_CD'] == 'TX') &
                          (pos_2016['ZIP_CD'].astype(str).str.replace(r'\.0$', '',
↪  regex=True).str.startswith(('75', '76', '77', '78', '79')))]

# Count hospitals per ZIP code
texas_hospital_num = texas_hospital.groupby('ZIP_CD').size().reset_index(name='hospital_count')

# Convert both ZIP code columns to string for merging
texas_hospital_num['ZIP_CD'] = texas_hospital_num['ZIP_CD'].astype(str).str.replace(r'\.0$', '',
↪  regex=True)
```

```
# Merge ZIP code boundaries with hospital counts
texas_hospital_merged = texas_zip.merge(texas_hospital_num, left_on='ZCTA5', right_on='ZIP_CD',
↪  how='left')

# Fill NaN values in 'hospital_count' with 0
texas_hospital_merged['hospital_count'] = texas_hospital_merged['hospital_count'].fillna(0)

# Print hospital count per zip, and describe the describe
print(texas_hospital_merged[['ZCTA5', 'hospital_count']].head())
print(texas_hospital_merged['hospital_count'].describe())

# Plot the number of hospitals per ZIP code in Texas (color was adjusted to make hot spots more obvious)
fig, ax = plt.subplots(figsize=(4, 4), dpi=50)
texas_hospital_merged.plot(
    column='hospital_count',
    cmap='YlOrBr',
    linewidth=0.5,
    edgecolor='0.6',
    ax=ax,
    legend=True,
    vmin=0,
    vmax=60 # lower max to make maps more obvious
)

ax.set_title("Number of Hospitals per ZIP Code in Texas (2016) (Count)", fontsize=12)
ax.set_axis_off()

# Show plot
plt.show()
```

```
   ZCTA5  hospital_count
0  78624            21.0
1  78626            25.0
2  78628            11.0
3  78631             0.0
4  78632             0.0
count    1935.000000
mean        8.119897
std        14.024321
min         0.000000
25%         0.000000
50%         2.000000
75%        11.000000
max       195.000000
Name: hospital_count, dtype: float64
```



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

1.

```
# Load the shapefile for ZIP code boundaries
shapefile_path = '/Users/georgew/Desktop/Fall 2024/PPHA
↪  30538/problem-set-4-yuxuan-george/gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.shp'
zip_codes = gpd.read_file(shapefile_path)

# Calculate the centroids of each ZIP code polygon and create a GeoDataFrame
zips_all_centroids = zip_codes.copy()
zips_all_centroids['geometry'] = zip_codes.centroid

# Display dimensions and columns of the resulting GeoDataFrame
dimensions = zips_all_centroids.shape
columns = zips_all_centroids.columns

print(f"Dimensions of the GeoDataFrame: {dimensions}")
print(f"Columns: {columns}")
```

```
/var/folders/k2/prgbv7z97knbd104r93pncfc0000gp/T/ipykernel_52980/2933873005.py:7: UserWarning:

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


Dimensions of the GeoDataFrame: (33120, 6)
Columns: Index(['GEO_ID', 'ZCTA5', 'NAME', 'LSAD', 'CENSUSAREA', 'geometry'], dtype='object')
```

The zips_all_centroids GeoDataFrame has 3,120 rows and 6 columns, representing ZIP code centroids across the U.S. The columns include GEO_ID (unique geographic identifier), ZCTA5 (5-digit ZIP code), NAME (ZIP code label), LSAD (type of geographic area), CENSUSAREA (area size), and geometry (centroid coordinates as point geometries). This data allows for analyzing or visualizing central points of ZIP code areas.

2.

```
# Load the zips_all_centroids
shapefile_path = '/Users/georgew/Desktop/Fall 2024/PPHA
↪  30538/problem-set-4-yuxuan-george/gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.shp'
zips_all_centroids = gpd.read_file(shapefile_path)

# Calculate centroids
zips_all_centroids['geometry'] = zips_all_centroids.centroid

# Texas ZIP code prefixes
texas_prefixes = ('75', '76', '77', '78', '79')

# Bordering states ZIP code prefixes
border_states_prefixes = texas_prefixes + ('73', '74', '70', '71', '72', '86', '87', '88')

# Create gdf for all ZIP codes in Texas
zips_texas_centroids = zips_all_centroids[zips_all_centroids['ZCTA5'].str.startswith(texas_prefixes)]

# Create gdf for all ZIP codes in Texas or bordering states
zips_texas_borderstates_centroids =
↪  zips_all_centroids[zips_all_centroids['ZCTA5'].str.startswith(border_states_prefixes)]

# Get the number of unique ZIP codes in each subset
unique_texas_zip_codes = zips_texas_centroids['ZCTA5'].nunique()
unique_borderstates_zip_codes = zips_texas_borderstates_centroids['ZCTA5'].nunique()

unique_texas_zip_codes, unique_borderstates_zip_codes
```

```
/var/folders/k2/prgbv7z97knbd104r93pncfc0000gp/T/ipykernel_52980/2562787517.py:6: UserWarning:

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


(1935, 4160)
```

There are 1,935 unique ZIP codes in Texas and 4,160 unique ZIP codes in Texas or bordering states.

3.

```
# Merge the gdf containing Texas and bordering states ZIP code centroids
zips_with_hospital = zips_texas_borderstates_centroids.merge(
    texas_hospital_num,
    left_on='ZCTA5',
    right_on='ZIP_CD',
    how='left'
)
```

```
# Filter to keep only ZIP codes with at least 1 hospital in 2016
zips_withhospital_centroids = zips_with_hospital[zips_with_hospital['hospital_count'] > 0]
```

I used a left merge on the ZIP code variable (ZCTA5 from the ZIP code GeoDataFrame and ZIP_CD from the hospital dataset).

4. a.

```
import geopandas as gpd
import time
from shapely.ops import nearest_points

# Subset to 10 ZIP codes from Texas
subset_texas_centroids = zips_texas_centroids.sample(10)

# Start timer to measure time
start_time = time.time()

# Initialize
nearest_distances = []

# Calculate the distance to the nearest ZIP code with a hospital for each ZIP code
for _, row in subset_texas_centroids.iterrows():
    origin_point = row['geometry']
    nearest_point = nearest_points(origin_point, zips_withhospital_centroids.unary_union)[1]
    distance = origin_point.distance(nearest_point)
    nearest_distances.append(distance)

# End
end_time = time.time()

# Calculate the elapsed time
elapsed_time = end_time - start_time
print(f"Time taken for 10 ZIP codes: {elapsed_time:.2f} seconds")

# Estimate time for the entire dataset (linear scaling)
total_time_estimate = (elapsed_time / 10) * len(zips_texas_centroids)
print(f"Estimated time for entire dataset: {total_time_estimate / 60:.2f} minutes")
```

```
Time taken for 10 ZIP codes: 0.01 seconds
Estimated time for entire dataset: 0.03 minutes

/var/folders/k2/prgbv7z97knbd104r93pncfc0000gp/T/ipykernel_52980/3423632555.py:17: DeprecationWarning:

The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
```

The following time change every time. Time taken for 10 ZIP codes: 0.01 seconds Estimated time for entire dataset: 0.02 minutes

b.

```
# Start the timer to measure computation time
start_time = time.time()

# Calculate the distance in the full dataset
nearest_distances = []
for _, row in zips_texas_centroids.iterrows():
    origin_point = row['geometry']
    nearest_point = nearest_points(origin_point, zips_withhospital_centroids.unary_union)[1]
    distance = origin_point.distance(nearest_point)
    nearest_distances.append(distance)

# End the timer
end_time = time.time()

# Calculate the elapsed time
elapsed_time = end_time - start_time
print(f"Time taken for full dataset: {elapsed_time / 60:.2f} minutes")
```

```
/var/folders/k2/prgbv7z97knbd104r93pncfc0000gp/T/ipykernel_52980/2796076469.py:8: DeprecationWarning:

The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.

Time taken for full dataset: 0.02 minutes
```

c.

The CRS is GCS North American 1983 (NAD83). UNIT[“Degree”,0.0174532925199433295]. The unit is in degrees.

```
# Add the nearest distances in degrees to the gdf
zips_texas_centroids = zips_texas_centroids.copy()  # Avoid SettingWithCopyWarning
zips_texas_centroids['nearest_hospital_distance_degrees'] = nearest_distances

zips_texas_centroids['nearest_hospital_distance_miles'] =
↪  zips_texas_centroids['nearest_hospital_distance_degrees'] * 69
```

5. a. Now, it is in miles.

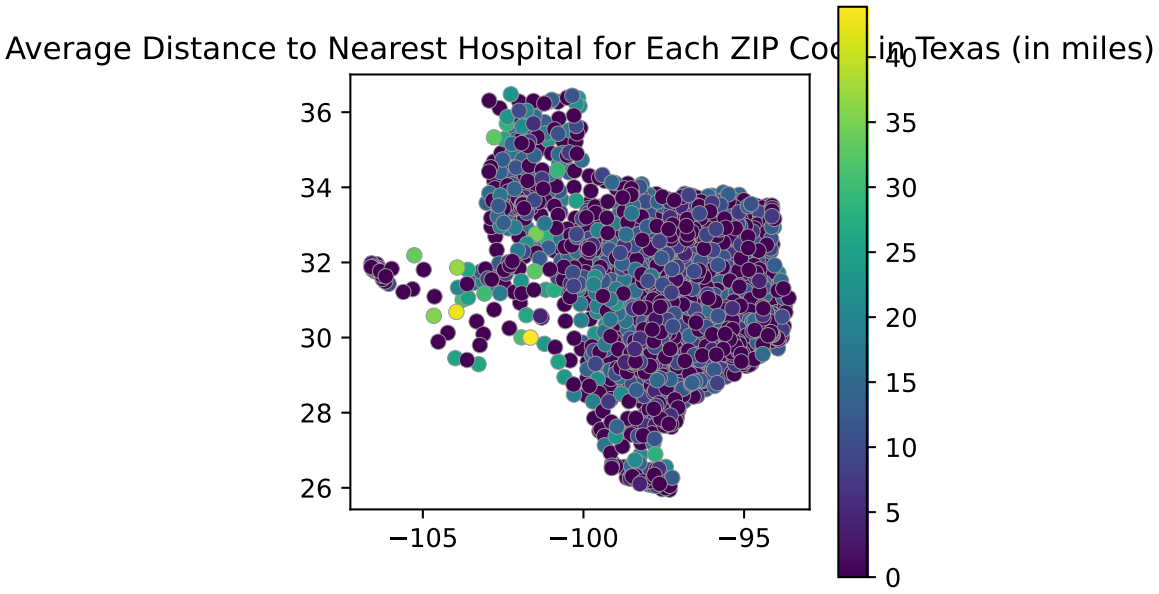
   b.

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

```
Average distance to the nearest hospital (in miles): 4.23
```

Average distance to the nearest hospital (in miles): 4.23. This makes sense beucase many urban areas have hospitals in ver close distance that can balance out some far distance from hositals in rural areas. c.

```
fig, ax = plt.subplots(figsize=(4, 4), dpi=50)
zips_texas_centroids.plot(
    column='nearest_hospital_distance_miles',
    linewidth=0.3,
    ax=ax,
    edgecolor='0.6',
    legend=True
)
plt.title("Average Distance to Nearest Hospital for Each ZIP Code in Texas (in miles)")
plt.show()
```



Effects of closures on access in Texas (15 pts)

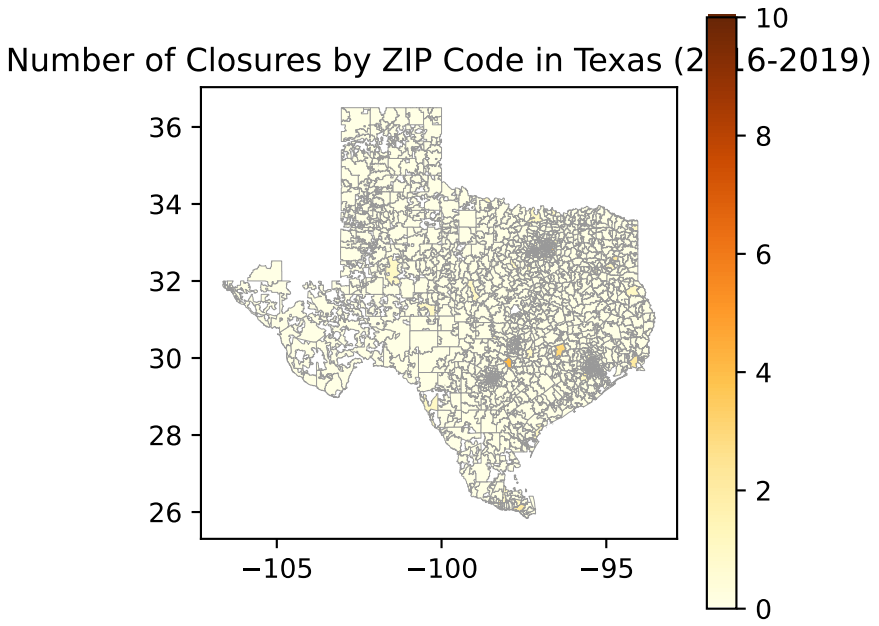
1.

```
# Convert to strings
confirmed_closures_df['ZIP_CD'] = confirmed_closures_df['ZIP_CD'].astype(str)

# Filter the closures
closures_texas = confirmed_closures_df[
    (confirmed_closures_df['ZIP_CD'].str.startswith(('75', '76', '77', '78', '79')))
]

# Group by ZIP code and count closrues
closures_by_zip = closures_texas.groupby('ZIP_CD').size().reset_index(name='Number_of_Closures')

# Display the table of the number of closures by ZIP code
closures_by_zip.sort_values(by='Number_of_Closures', ascending=False, inplace=True)
print(closures_by_zip)
```

```
     ZIP_CD  Number_of_Closures
43  77036.0                  30
56  77477.0                  14
49  77074.0                  11
46  77063.0                   5
7   75150.0                   4
..      ...                 ...
41  77030.0                   1
1   75044.0                   1
45  77060.0                   1
48  77072.0                   1
85  79905.0                   1

[86 rows x 2 columns]
```

2.

```
closures_by_zip['ZIP_CD'] = closures_by_zip['ZIP_CD'].astype(str).str.replace(r'\.0$', '', regex=True)

# Merge Texas ZIP code boundaries
closures_choropleth = texas_zip.merge(closures_by_zip, left_on='ZCTA5', right_on='ZIP_CD', how='left')

# Fill NaN values in 'Number_of_Closures' with 0
closures_choropleth['Number_of_Closures'] = closures_choropleth['Number_of_Closures'].fillna(0)

# Plot the choropleth showing affected ZIP codes
fig, ax = plt.subplots(figsize=(4, 4), dpi=50)
closures_choropleth.plot(
    column='Number_of_Closures',
    linewidth=0.2,
    ax=ax,
    cmap='YlOrBr',
    edgecolor='0.6',
    legend=True,
    vmin = 0,
    vmax = 10 # lower max to make the map more obvious
)
plt.title("Number of Closures by ZIP Code in Texas (2016-2019)")
plt.show()

# Count the number of directly affected ZIP codes
directly_affected_zip_codes = closures_choropleth[closures_choropleth['Number_of_Closures'] > 0].shape[0]
print(f"Number of directly affected ZIP codes in Texas: {directly_affected_zip_codes}")
```




```
Number of directly affected ZIP codes in Texas: 86
```

3.

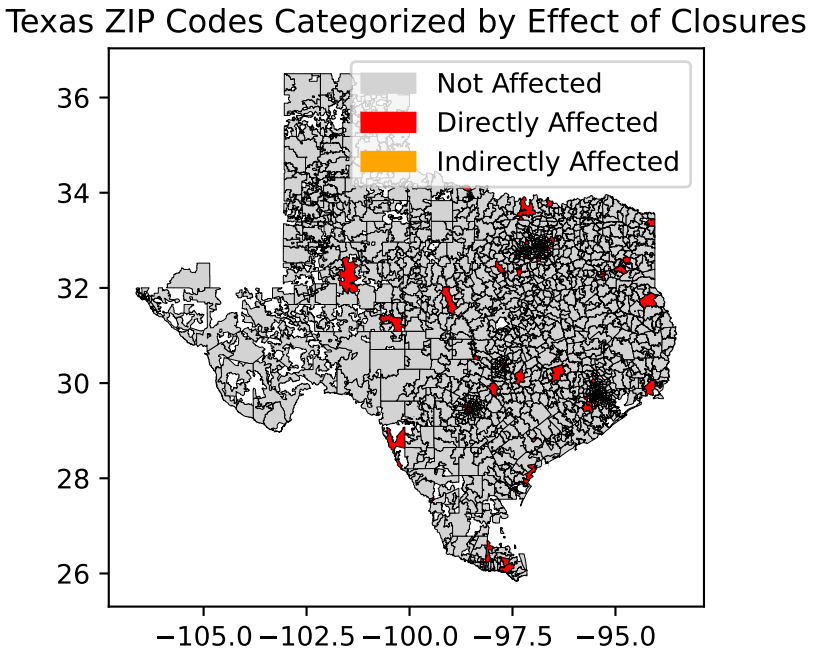
```python
# Filter to get directly affected ZIP codes
directly_affected_gdf = closures_choropleth[closures_choropleth['Number_of_Closures'] > 0]

# Reproject to a CRS
if directly_affected_gdf.crs.is_geographic:
    directly_affected_gdf = directly_affected_gdf.to_crs(epsg=3857)

# Create a 10-mile buffer around directly affected ZIP codes (approx. 16093.4 meters)
directly_affected_buffer = directly_affected_gdf.copy()
directly_affected_buffer['geometry'] = directly_affected_buffer.geometry.buffer(16093.4)

# Reproject back to original CRS for further operations
if directly_affected_buffer.crs != texas_zip.crs:
    directly_affected_buffer = directly_affected_buffer.to_crs(texas_zip.crs)

# Perform spatial join to find indirectly affected ZIP codes
indirectly_affected_gdf = gpd.sjoin(texas_zip, directly_affected_buffer, how='inner',
↪  predicate='intersects')

# Remove duplicates
if 'ZCTA5_left' in indirectly_affected_gdf.columns:
    indirectly_affected_gdf = indirectly_affected_gdf.drop_duplicates(subset='ZCTA5_left')
elif 'ZCTA5' in indirectly_affected_gdf.columns:
    indirectly_affected_gdf = indirectly_affected_gdf.drop_duplicates(subset='ZCTA5')

# Count
num_indirectly_affected = indirectly_affected_gdf.shape[0]
print(f"Number of indirectly affected ZIP codes: {num_indirectly_affected}")
```

```
Number of indirectly affected ZIP codes: 896
```

4.

```
import matplotlib.patches as mpatches

# Ensure texas_zip has a copy to avoid warnings
texas_zip = texas_zip.copy()

# Drop any rows with missing geometries
texas_zip = texas_zip[texas_zip['geometry'].notna()]

# Ensure that CRS is consistent across gdf
if texas_zip.crs != indirectly_affected_gdf.crs:
    indirectly_affected_gdf = indirectly_affected_gdf.to_crs(texas_zip.crs)

# Assign categories to each ZIP code
texas_zip['affected_category'] = 'Not Affected'  # Default to 'Not Affected'

# Mark directly affected ZIP codes
texas_zip.loc[texas_zip['ZCTA5'].isin(directly_affected_gdf['ZCTA5']), 'affected_category'] = 'Directly
↪  Affected'

# Mark indirectly affected ZIP codes
texas_zip.loc[
    (texas_zip['ZCTA5'].isin(indirectly_affected_gdf['ZCTA5_right'])) &
    (texas_zip['affected_category'] != 'Directly Affected'),
    'affected_category'
] = 'Indirectly Affected'

# Drop rows with missing geometries again
texas_zip = texas_zip[texas_zip['geometry'].notna()]

# Create choropleth plot
fig, ax = plt.subplots(figsize=(4, 4), dpi=50)
category_colors = {
    'Not Affected': '#d3d3d3',  # Gray for not affected
    'Directly Affected': '#ff0000',  # Red for directly affected
    'Indirectly Affected': '#ffa500'  # Orange for indirectly affected
}

# Plot each category separately
for category, color in category_colors.items():
    subset = texas_zip[texas_zip['affected_category'] == category]
    if not subset.empty:
        subset.plot(
            ax=ax,
            color=color,
            edgecolor='black',
            linewidth=0.3,
            label=category
        )

# Create custom legend
legend_patches = [
    mpatches.Patch(color=color, label=category)
    for category, color in category_colors.items()
]
```

```
# Add title and legend
plt.title("Texas ZIP Codes Categorized by Effect of Closures")
plt.legend(handles=legend_patches, loc='upper right')
plt.show()
```



Reflecting on the exercise (10 pts)

a. First, there can be delayed data reporting that hospitals might close, but the official count could remain the same due to reporting delays. This could miss closures if we only look at a year to year comparison. In this case, we could track closures over multiple years, including a broader time range to confirm hospital closures with less impact form delay data reporting.

Second, when hospitals merge or are acquired, they may be recorded differently, causing a reduction in count. This can happen even when the facility remains operation after merger. One of the approaches we used is to identify areas where the total hospital count doesn't decrease. To improve this, we could use text-matching algorithms to recognize mergers and acquisitions in the dataset (especially for hospitals with similar names), identifying cases where hospitals change ownership or merge but continue operations. We even can collect more information on the ownership strcuture or the adress to see whether mergers are represented clearly, which can better reflect hospital closures.

Third, we rely on zip code data alone. ZIP code-based hospital counts may not capture nuanced details, such as changes in hospital size or service capacity. To improve this problem, we could consider incorporating other indicators like bed counts to better reflect hospital closures.

b. The current method identifies affected ZIP codes by counting hospitals per ZIP code annually. We listed suspected closures by finding hospitals active in 2016 but inactive in later years, removing cases where hospital counts remained steady, suggesting mergers or acquisitions. This approach provides a straightforward way to detect closures and their immediate impact at the ZIP code level, but also has some limitations.

First, ZIP codes represent administrative boundaries, which may not reflect well actual patient access. Hospitals just outside a ZIP code boundary may still be accessible to residents, but this wouldn't be captured in the current method. To improve this, we could expand the analysis to include nearby ZIP codes or using a distance-based method like within a certain radius to reflect access more accurately, such as using google API to better reflect the distance as an indicator to accessibility.

Second, the current method doesn't account for actual travel distances or times. This will misrepresents access in rural versus urban areas, where 10 miles can represent vastly different travel times. To improve this, we could use GIS tools to calculate driving distances or estimated travel times to the nearest hospital would provide a more coorect accessibility. This could be particularly valuable in areas where a single closure forces patients to travel farther.

Third, we ignore the hospital capacity and services. Not all hospitals have the same capacity or the same services, so the simple count of hospitals might not reflect actual healthcare availability, especially if closures affect larger hospitals or facilities offering specialized care. To improve this, we can include more data on services availability into the analysis to better reflect healthcare access changes with in zip codes.