Problem Set 4

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

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): Ashirwad Wakade - ashirwad
- Partner 2 (name and cnet ID): Anand Kshirsagar -anandkshirsagar

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: ** Ashirwad Wakade and Anand Kshirsagar**

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: **2** 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

Style Points (10 pts)

Submission Steps (10 pts)

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

1

The Variables taken are

PRVDR_CTGRY_SBTYP_CD - Provider Category Subtype Code

PRVDR_CTGRY_CD - Provider Category Code

FAC_NAME - Facility Name

PRVDR_NUM - CMS Certification Number

PGM_TRMNTN_CD - Termination Code

ZIP_CD - ZIP Code

2

```
#loading required packages
import pandas as pd
import altair as alt
import geopandas as gpd
from shapely.geometry import Point, Polygon

#importing dataset
pos2016  =
 ↪  pd.read_csv('/Users/afreenwala/Documents/GitHub/problem-set-4-anand-and-ashirwad/pos2016
#subeseting data for short term hospital
short_term_hospital_2016 = pos2016[(pos2016['PRVDR_CTGRY_CD'] == 1) &
 ↪  (pos2016['PRVDR_CTGRY_SBTYP_CD'] == 1)]
```

a.

```
#Getting number of hospitals
no_hospitals_2016 = short_term_hospital_2016['PRVDR_NUM'].nunique()
print(f"Number of short-term hospitals in 2016 are: {no_hospitals_2016}")
```

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

b

According to the NHS, there were 3,436 short-term hospitals in 2016—a notably lower figure compared to CMS data, which lists 7,245. This discrepancy may stem from differing definitions of short-term hospitals and Medicaid/Medicare eligibility criteria.

3

```
#Loading each dataset
pos2017 =
 ↪  pd.read_csv('/Users/afreenwala/Documents/GitHub/problem-set-4-anand-and-ashirwad/pos
 ↪  year wise/pos2017.csv', encoding='ISO-8859-1')
pos2018 =
 ↪  pd.read_csv('/Users/afreenwala/Documents/GitHub/problem-set-4-anand-and-ashirwad/pos
 ↪  year wise/pos2018.csv', encoding='ISO-8859-1')
pos2019 =
 ↪  pd.read_csv('/Users/afreenwala/Documents/GitHub/problem-set-4-anand-and-ashirwad/pos
 ↪  year wise/pos2019.csv', encoding='ISO-8859-1')

#filtering for short term hospitals
short_term_hospital_2017 = pos2017[(pos2017['PRVDR_CTGRY_CD'] == 1) &
 ↪  (pos2017['PRVDR_CTGRY_SBTYP_CD'] == 1)]
short_term_hospital_2018 = pos2018[(pos2018['PRVDR_CTGRY_CD'] == 1) &
 ↪  (pos2018['PRVDR_CTGRY_SBTYP_CD'] == 1)]
short_term_hospital_2019 = pos2019[(pos2019['PRVDR_CTGRY_CD'] == 1) &
 ↪  (pos2019['PRVDR_CTGRY_SBTYP_CD'] == 1)]

#appending the filtered ones
# Add a Year column to each year's DataFrame
short_term_hospital_2016['Year'] = 2016
short_term_hospital_2017['Year'] = 2017
short_term_hospital_2018['Year'] = 2018
short_term_hospital_2019['Year'] = 2019

# Concatenate all years, starting with 2016
all_short_term_hospitals = pd.concat(
    [short_term_hospital_2016, short_term_hospital_2017,
 ↪  short_term_hospital_2018, short_term_hospital_2019],
    ignore_index=True
)

#ensuring ZIP_CD is string
all_short_term_hospitals['ZIP_CD'] =
 ↪  all_short_term_hospitals['ZIP_CD'].astype(str).str.split('.').str[0].str.zfill(5)
```

```
#Plotting the number of observations by year
data = {
    'Year': ['2017', '2018', '2019'],
    'Count': [len(short_term_hospital_2017), len(short_term_hospital_2018),
     ↪  len(short_term_hospital_2019)]
}
df_counts = pd.DataFrame(data)
# Create the bar chart using Altair
chart = alt.Chart(df_counts).mark_bar().encode(
    x='Year:O',
    y='Count:Q',
).properties(
    title='Number of Short-term Hospitals by Year'
)
```

```
/var/folders/c8/5p3gfwn56xv4bsh3gd84skjm0000gn/T/ipykernel_37699/954323245.py:13:
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
  short_term_hospital_2016['Year'] = 2016
/var/folders/c8/5p3gfwn56xv4bsh3gd84skjm0000gn/T/ipykernel_37699/954323245.py:14:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  short_term_hospital_2017['Year'] = 2017
/var/folders/c8/5p3gfwn56xv4bsh3gd84skjm0000gn/T/ipykernel_37699/954323245.py:15:
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
  short_term_hospital_2018['Year'] = 2018
```

```
/var/folders/c8/5p3gfwn56xv4bsh3gd84skjm0000gn/T/ipykernel_37699/954323245.py:16:
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
  short_term_hospital_2019['Year'] = 2019
```

4 a

```
 #Prepare the data for unique hospitals
unique_hospitals = {
    'Year': ['2017', '2018', '2019'],
    'Unique_Hospitals': [
        len(short_term_hospital_2017['PRVDR_NUM'].unique()),
        len(short_term_hospital_2018['PRVDR_NUM'].unique()),
        len(short_term_hospital_2019['PRVDR_NUM'].unique())
    ]
}

df_unique_hospitals = pd.DataFrame(unique_hospitals)

# Create the bar chart for unique hospitals using Altair
unique_chart = alt.Chart(df_unique_hospitals).mark_bar().encode(
    x='Year:O',
    y='Unique_Hospitals:Q',
).properties(
    title='Number of Unique Hospitals by Year'
)
unique_chart
```

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

b

The alignment between total observations and unique hospitals suggests a stable dataset, with few hospitals entering or leaving each year. This steadiness implies that, despite occasional closures or ownership changes, the number of active short-term hospitals remains largely consistent.

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

1

```
# Filter for active hospitals in 2016
active_2016 =
 ↪  short_term_hospital_2016[short_term_hospital_2016['PGM_TRMNTN_CD'] ==
 ↪  0][['PRVDR_NUM', 'FAC_NAME', 'ZIP_CD']]

#Merge with 2017 Data to Find Closures in 2017
merged_2017 =
 ↪  active_2016.merge(short_term_hospital_2017[short_term_hospital_2017['PGM_TRMNTN_CD']
 ↪  == 0][['PRVDR_NUM']], on=['PRVDR_NUM'], how='left', indicator=True)
closed_2017 = merged_2017[merged_2017['_merge'] == 'left_only'].copy()
closed_2017['Closure_Year'] = 2017

#Repeat for 2018 to Identify Additional Closures
merged_2018 =
 ↪  active_2016.merge(short_term_hospital_2018[short_term_hospital_2018['PGM_TRMNTN_CD']
 ↪  == 0][['PRVDR_NUM']], on=['PRVDR_NUM'], how='left', indicator=True)
closed_2018 = merged_2018[merged_2018['_merge'] == 'left_only'].copy()
closed_2018['Closure_Year'] = 2018

#Repeat for 2019 to Identify Additional Closures
merged_2019 =
 ↪  active_2016.merge(short_term_hospital_2019[short_term_hospital_2019['PGM_TRMNTN_CD']
 ↪  == 0][['PRVDR_NUM']], on=['PRVDR_NUM'], how='left', indicator=True)
closed_2019 = merged_2019[merged_2019['_merge'] == 'left_only'].copy()
closed_2019['Closure_Year'] = 2019

closed_hospitals = pd.concat([closed_2017[['PRVDR_NUM', 'ZIP_CD', 'FAC_NAME',
 ↪  'Closure_Year']],
                              closed_2018[['PRVDR_NUM', 'ZIP_CD', 'FAC_NAME',
 ↪  'Closure_Year']],
                              closed_2019[['PRVDR_NUM', 'ZIP_CD', 'FAC_NAME',
 ↪  'Closure_Year']]]).drop_duplicates(subset=['PRVDR_NUM'])

print(f"Total number of hospitals suspected to have closed:
 ↪  {closed_hospitals.shape[0]}")
```

```
Total number of hospitals suspected to have closed: 174
```

2

```
#droopinng duplicates
closed_hospitals = pd.concat([closed_2017[['PRVDR_NUM', 'ZIP_CD', 'FAC_NAME',
 ↪  'Closure_Year']],
                              closed_2018[['PRVDR_NUM', 'ZIP_CD', 'FAC_NAME',
 ↪  'Closure_Year']],
                              closed_2019[['PRVDR_NUM', 'ZIP_CD', 'FAC_NAME',
 ↪  'Closure_Year']]]).drop_duplicates(subset=['PRVDR_NUM'])

# Drop duplicates based on PRVDR_NUM while keeping Closure_Year
closed_hospitals_with_names =
 ↪  closed_hospitals.drop_duplicates(subset='PRVDR_NUM')

# Sort by FAC_NAME and select relevant columns
sorted_closed_hospitals = closed_hospitals_with_names[['FAC_NAME',
 ↪  'Closure_Year']].sort_values(by='FAC_NAME').head(10)

# Display the sorted list with FAC_NAME and Closure_Year
print("First 10 hospitals suspected to have closed:")
print(sorted_closed_hospitals)
```

```
First 10 hospitals suspected to have closed:
                                          FAC_NAME  Closure_Year
93                           ABRAZO MARYVALE CAMPUS          2017
299       ADVENTIST MEDICAL CENTER - CENTRAL VALLEY          2017
2276                        AFFINITY MEDICAL CENTER          2018
2019  ALBANY MEDICAL CENTER / SOUTH CLINICAL CAMPUS          2017
2955       ALLEGIANCE SPECIALTY HOSPITAL OF KILGORE          2017
1682                         ALLIANCE LAIRD HOSPITAL          2019
2345                       ALLIANCEHEALTH DEACONESS          2019
720                   ANNE BATES LEACH EYE HOSPITAL          2019
528         ARKANSAS VALLEY REGIONAL MEDICAL CENTER          2017
1801            BANNER CHURCHILL COMMUNITY HOSPITAL          2017
```

3

```
# Count active hospitals by ZIP code for each year
zip_counts_2016 = pos2016[pos2016['PGM_TRMNTN_CD'] ==
 ↪  0].groupby('ZIP_CD').size()
zip_counts_2017 = pos2017[pos2017['PGM_TRMNTN_CD'] ==
 ↪  0].groupby('ZIP_CD').size()
zip_counts_2018 = pos2018[pos2018['PGM_TRMNTN_CD'] ==
 ↪  0].groupby('ZIP_CD').size()
```

```
zip_counts_2019 = pos2019[pos2019['PGM_TRMNTN_CD'] ==
↪  0].groupby('ZIP_CD').size()

#Check for potential mergers/acquisitions among suspected closures
potential_mergers = []

for _, row in closed_hospitals.iterrows():
    zip_code = row['ZIP_CD']
    closure_year = row['Closure_Year']

    # Get the active hospital counts for the closure ZIP code in the closure
    ↪  year and the following year
    if closure_year == 2017:
        if zip_counts_2018.get(zip_code, 0) >= zip_counts_2017.get(zip_code,
        ↪  0):
            potential_mergers.append(row['PRVDR_NUM'])
    elif closure_year == 2018:
        if zip_counts_2019.get(zip_code, 0) >= zip_counts_2018.get(zip_code,
        ↪  0):
            potential_mergers.append(row['PRVDR_NUM'])
    elif closure_year == 2016:
        if zip_counts_2017.get(zip_code, 0) >= zip_counts_2016.get(zip_code,
        ↪  0):
            potential_mergers.append(row['PRVDR_NUM'])

#Filter out suspected closures that are potential mergers
filtered_closed_hospitals =
↪  closed_hospitals[~closed_hospitals['PRVDR_NUM'].isin(potential_mergers)]
```

a

```
# a. Number of hospitals fitting the merger/acquisition definition
num_potential_mergers = len(potential_mergers)
print(f"Number of hospitals fitting the merger/acquisition definition:
↪  {num_potential_mergers}")
```

```
Number of hospitals fitting the merger/acquisition definition: 83
```

b

```
# b. Number of hospitals left after correcting for mergers
num_remaining_hospitals = filtered_closed_hospitals.shape[0]
print(f"Number of hospitals remaining after correcting for mergers:
 ↪  {num_remaining_hospitals}")
print(num_remaining_hospitals)
```

```
Number of hospitals remaining after correcting for mergers: 91
91
```

c

```
# c. Sort by facility name and report the first 10 rows
sorted_filtered_hospitals =
 ↪  filtered_closed_hospitals.sort_values(by='FAC_NAME').head(10)
print("First 10 hospitals suspected to have closed after correction for
 ↪  mergers:")
print(sorted_filtered_hospitals[['FAC_NAME', 'ZIP_CD', 'Closure_Year']])
```

```
First 10 hospitals suspected to have closed after correction for mergers:
                                                FAC_NAME   ZIP_CD
                                                Closure_Year
2955           ALLEGIANCE SPECIALTY HOSPITAL OF KILGORE  75662.0
2017
1682                             ALLIANCE LAIRD HOSPITAL  39365.0
2019
2345                            ALLIANCEHEALTH DEACONESS  73112.0
2019
720                         ANNE BATES LEACH EYE HOSPITAL  33136.0
2019
2579                        BARIX CLINICS OF PENNSYLVANIA  19047.0
2019
3406               BAY AREA REGIONAL MEDICAL CENTER, LLC  77598.0
2018
3407                     BAYLOR EMERGENCY MEDICAL CENTER  75087.0
2019
3398  BAYLOR SCOTT & WHITE EMERGENCY MEDICAL CENTER ...  78613.0
2019
2278                           BELMONT COMMUNITY HOSPITAL  43906.0
2019
1773                              BIG SKY MEDICAL CENTER  59716.0
2019
```

Download Census zip code shapefile (10 pt)

1 a The five file types are .dbf - it has attribution information

.prj - describes the Coordinate Reference System (CRS)

.shp - it feature geometrics

.shx - it has has a positional index

.xml - optional metadata file that provides additional information about the shapefile's contents and purpose

b

```
import os
folder_path =
 ↪  '/Users/afreenwala/Documents/GitHub/problem-set-4-anand-and-ashirwad/gz_2010_us_860_00_5
# List all files in the folder and get their sizes
for file_name in os.listdir(folder_path):
    file_path = os.path.join(folder_path, file_name)
    if os.path.isfile(file_path):
        file_size = os.path.getsize(file_path)  # Get size in bytes
        print(f"{file_name}: {file_size / (1024 * 1024):.2f} MB")
```

```
gz_2010_us_860_00_500k.prj: 0.00 MB
gz_2010_us_860_00_500k.shx: 0.25 MB
gz_2010_us_860_00_500k.shp: 798.74 MB
gz_2010_us_860_00_500k.dbf: 6.13 MB
gz_2010_us_860_00_500k.xml: 0.01 MB
```

2

```
#importing packages
import geopandas as gpd
import pandas as pd

#Load the shapefile and restrict to Texas ZIP codes
shapefile_path =
 ↪  '/Users/afreenwala/Documents/GitHub/problem-set-4-anand-and-ashirwad/gz_2010_us_860_00_5
zipcodes_gdf = gpd.read_file(shapefile_path)
zipcodes_gdf = zipcodes_gdf.rename(columns={'ZCTA5': 'ZIP_CD'}) #rename ZCTA5
 ↪  to ZIP_CD
zipcodes_gdf['ZIP_CD'] = zipcodes_gdf['ZIP_CD'].astype(str) #Ensure ZIP_CD is
 ↪  a string
```

```
# Filter ZIP codes for Texas using the first digit '7' (typically for Texas
 ↪  ZIP codes)
texas_zipcodes = zipcodes_gdf[zipcodes_gdf['ZIP_CD'].str.startswith(('733',
 ↪  '75', '76', '77', '78', '79'))]

#texas zip code and POS data
hospitals_texas =
 ↪  all_short_term_hospitals[all_short_term_hospitals['ZIP_CD'].str.startswith(('733',
 ↪  '75', '76', '77', '78', '79'))]

#hospitals per zip
hospitals_per_zipcode =
 ↪  hospitals_texas.groupby('ZIP_CD').size().reset_index(name='hospital_count')

#merging hospitals and zipcode shape file
texas_zipcodes = texas_zipcodes.merge(hospitals_per_zipcode, on='ZIP_CD',
 ↪  how='left').fillna(0)
```

```
import matplotlib.pyplot as plt
# Plot a choropleth of the number of hospitals per ZIP code in Texas
fig, ax = plt.subplots(1, 1, figsize=(12, 10))
texas_zipcodes.plot(column='hospital_count',
                        cmap='Reds',
                        linewidth=0.8,
                        ax=ax,
                        edgecolor='0.8',
                        legend=True)

ax.set_title("Number of Hospitals per ZIP Code in Texas (2016)")
plt.show()
```

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

4.1

```
# Create a copy of the ZIP code GeoDataFrame
zipcode_data = zipcodes_gdf.copy()

# Calculate centroids for each ZIP code geometry
```

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

# Extract longitude and latitude from the centroid points
zipcode_data['longitude'] = zipcode_data['centroid'].x
zipcode_data['latitude'] = zipcode_data['centroid'].y

# Create a GeoDataFrame for ZIP code centroids
zips_all_centroids = gpd.GeoDataFrame(
    zipcode_data,
    geometry=gpd.points_from_xy(zipcode_data.longitude,
 ↪  zipcode_data.latitude),
    crs="EPSG:4326"
)
# Display dimensions and the first few rows
print(f"Dimensions of zips_all_centroids: {zips_all_centroids.shape}")
print(zips_all_centroids.head())
```

4.2.

```
# Filter for Texas ZIP codes only
zips_texas_centroids =
 ↪  zips_all_centroids[zips_all_centroids['ZIP_CD'].str.startswith('7')]

# Filter for Texas and bordering states ZIP codes
border_states_prefixes = ['7', '73', '74', '72', '70', '71', '87', '88']
zips_texas_borderstates_centroids = zips_all_centroids[

 ↪  zips_all_centroids['ZIP_CD'].str.startswith(tuple(border_states_prefixes))
]
# Display the number of unique ZIP codes in each subset
print(f"Number of unique ZIP codes in Texas:
 ↪  {zips_texas_centroids['ZIP_CD'].nunique()}")
print(f"Number of unique ZIP codes in Texas and bordering states:
 ↪  {zips_texas_borderstates_centroids['ZIP_CD'].nunique()}")
```

4.3

```
# Step 1: Count hospitals per ZIP code in 2016
# Assuming combined_data_full contains only records for 2016 and has a
 ↪  'ZIP_CD' column
hospital_counts_2016 =
 ↪  all_short_term_hospitals.groupby('ZIP_CD').size().reset_index(name='hospital_count')
```

```
# Filter to retain ZIP codes with at least one hospital in 2016
hospitals_within_2016 =
 ↪  hospital_counts_2016[hospital_counts_2016['hospital_count'] >
 ↪  0][['ZIP_CD']]

# Step 2: Perform an inner merge with zips_texas_borderstates_centroids on
 ↪  'ZIP_CD'
zips_withhospital_centroids = zips_texas_borderstates_centroids.merge(
    hospitals_within_2016,
    on='ZIP_CD',
    how='inner'  # Inner join to include only ZIP codes with hospitals in
 ↪  2016
)

# Display the first few rows to check the result and count ZIP codes with
 ↪  hospitals
print(zips_withhospital_centroids.head())
print(f"Number of unique ZIP codes with at least one hospital in 2016:
 ↪  {len(zips_withhospital_centroids)}")
```

4.4 a

```
import time
# Step 1: Select a subset of 10 ZIP codes from zips_texas_centroids for
 ↪  testing
zips_texas_sample = zips_texas_centroids.head(10)

# Step 2: Record the start time for the subset distance calculation
start_time = time.time()

# Step 3: Reproject both GeoDataFrames to a suitable projected CRS for
 ↪  accurate distance calculations (UTM Zone for Texas)
zips_texas_sample = zips_texas_sample.to_crs(epsg=32614)  # Reprojecting to
 ↪  EPSG 32614 (UTM Zone)
zips_withhospital_centroids = zips_withhospital_centroids.to_crs(epsg=32614)

# Step 4: Calculate the minimum distance from each ZIP code in the sample to
 ↪  the nearest hospital location
zips_texas_sample['nearest_hospital_distance'] =
 ↪  zips_texas_sample.geometry.apply(
    lambda geom: zips_withhospital_centroids.distance(geom).min()
```

```
)

# Step 5: Record the end time and calculate the duration for the sample
end_time = time.time()
subset_time_taken = end_time - start_time
print(f"Time taken to compute distances for the sample:
 ↪  {subset_time_taken:.4f} seconds")

# Step 6: Estimate total time for calculating distances for all ZIP codes in
 ↪  zips_texas_centroids based on the sample time
estimated_total_time = subset_time_taken * (len(zips_texas_centroids) /
 ↪  len(zips_texas_sample))
print(f"Estimated time for the entire dataset: {estimated_total_time:.4f}
 ↪  seconds")
```

b

```
# Full calculation start time
start_time_full = time.time()

# Reproject all Texas ZIP codes to UTM zone for accurate distance
zips_texas_centroids = zips_texas_centroids.to_crs(epsg=32614)

# Calculate nearest hospital distance for each ZIP code in Texas
zips_texas_centroids['nearest_hospital_distance'] =
 ↪  zips_texas_centroids.geometry.apply(
    lambda geom: zips_withhospital_centroids.distance(geom).min()
)

# Full calculation end time
end_time_full = time.time()
time_taken_full = end_time_full - start_time_full

# Display time taken and compare to estimate
print(f"Time taken for full calculation: {time_taken_full:.4f} seconds")
print(f"Estimated vs Actual: {estimated_total_time:.4f} vs
 ↪  {time_taken_full:.4f} seconds")
```

c

```
# Define the path to the .prj file for the Texas ZIP code shapefile
prj_file_path =
 ↪  r"/Users/afreenwala/Documents/GitHub/problem-set-4-anand-and-ashirwad/gz_2010_us_860_00_5

# Read the CRS information from the .prj file
with open(prj_file_path, 'r') as prj_file:
    crs_info = prj_file.read()

# Display CRS information
print("CRS Information:")
print(crs_info)

# Check if units are in meters; typically, UTM projections use meters
# Convert distances to miles if units are meters
if "meters" in crs_info.lower():
    # Conversion factor: 1 meter   0.000621371 miles
    zips_texas_centroids['nearest_hospital_distance_miles'] = (
        zips_texas_centroids['nearest_hospital_distance'] * 0.000621371
    )

    # Display first few rows to confirm distance conversion
    print(zips_texas_centroids[['ZIP_CD',
     ↪  'nearest_hospital_distance_miles']].head())
```

4.5

a

```
# Calculate distance to nearest hospital
zips_texas_centroids['nearest_hospital_distance'] =
 ↪  zips_texas_centroids.geometry.apply(
    lambda x: zips_withhospital_centroids.distance(x).min()
)
# Report the unit
print("The unit of 'nearest_hospital_distance' is in meters.")
```

b

```
# Average distance by ZIP code
avg_distance_by_zip =
 ↪  zips_texas_centroids.groupby('ZIP_CD')['nearest_hospital_distance'].mean().reset_index()
```

```
# Convert to miles
avg_distance_by_zip['avg_distance_to_hospital_miles'] =
 ↪  avg_distance_by_zip['nearest_hospital_distance'] / 1609.34

# Report average distance
average_distance =
 ↪  avg_distance_by_zip['avg_distance_to_hospital_miles'].mean()
print(f"Average distance to nearest hospital in miles: {average_distance:.2f}
 ↪  miles")
# Reasoning: Assess whether this average distance makes sense based on
 ↪  geographical context.
```

c

```
# Merge for mapping
zips_map = zips_texas_centroids.merge(avg_distance_by_zip, on='ZIP_CD')

# Map the values
fig, ax = plt.subplots(1, 1, figsize=(12, 10))
zips_map.boundary.plot(ax=ax, linewidth=1)
zips_map.plot(column='avg_distance_to_hospital_miles', ax=ax, legend=True,
               legend_kwds={'label': "Avg Distance to Nearest Hospital
 ↪  (Miles)", 'orientation': "horizontal"},
               cmap='OrRd', missing_kwds={'color': 'lightgrey'})
plt.title('Avg Distance to Nearest Hospital by ZIP Code in Texas')
plt.show()
```

Effects of closures on access in Texas (15 pts)

5.1

```
# Step 1: Convert ZIP_CD in filtered_closed_hospitals to string, removing any
 ↪  decimal point if necessary
filtered_closed_hospitals['ZIP_CD'] =
 ↪  filtered_closed_hospitals['ZIP_CD'].astype(int).astype(str)

# Step 2: Create a list of Texas ZIP codes from texas_zipcodes
texas_zipcodes_list = texas_zipcodes['ZIP_CD'].unique()
```

```
# Step 3: Filter the closures DataFrame to include only those in Texas
closures_texas =
 ↪  filtered_closed_hospitals[filtered_closed_hospitals['ZIP_CD'].isin(texas_zipcodes_list)]

# Step 4: Count the number of closures by ZIP code
closures_count_by_zip =
 ↪  closures_texas.groupby('ZIP_CD').size().reset_index(name='Number_of_Closures')

# Step 5: Display the results
print("Number of closures by ZIP code in Texas:")
print(closures_count_by_zip)
```

5.2

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt

# Step 1: Count the number of closures by ZIP code
closures_by_zip =
 ↪  filtered_closed_hospitals['ZIP_CD'].value_counts().reset_index()
closures_by_zip.columns = ['ZIP_CD', 'Number_of_Closures']

# Step 2: Merge with Texas ZIP codes GeoDataFrame
# Assuming you already have the Texas ZIP codes GeoDataFrame as zipcodes_gdf
affected_zipcodes_gdf = zipcodes_gdf.merge(closures_by_zip, on='ZIP_CD',
 ↪  how='left')

# Fill NaN values for zip codes with no closures
affected_zipcodes_gdf['Number_of_Closures'].fillna(0, inplace=True)

# Filter for Texas ZIP codes only (if necessary)
# This assumes that zipcodes_gdf contains only Texas ZIP codes
# If not, you should filter it here
texas_zipcodes = ['733', '75', '76', '77', '78', '79']  # Add any other Texas
 ↪  ZIP code prefixes if needed
affected_zipcodes_gdf =
 ↪  affected_zipcodes_gdf[affected_zipcodes_gdf['ZIP_CD'].str.startswith(tuple(texas_zipcodes

# Step 3: Plot the Choropleth
fig, ax = plt.subplots(1, 1, figsize=(12, 10))
```

```
# Plot only Texas boundary
affected_zipcodes_gdf.boundary.plot(ax=ax, linewidth=1, color='black')
affected_zipcodes_gdf.plot(column='Number_of_Closures', ax=ax, legend=True,
                           legend_kwds={'label': "Number of Closures",
 ↪  'orientation': "horizontal"},
                           cmap='OrRd', missing_kwds={'color': 'lightgrey'})

# Set axis limits to focus on Texas
ax.set_xlim(-106, -93)  # Longitude limits for Texas
ax.set_ylim(25, 37)     # Latitude limits for Texas

plt.title('Texas ZIP Codes Affected by Hospital Closures (2016-2019)')
plt.show()

# Step 4: Count the number of directly affected zip codes
affected_zip_code_count =
 ↪  affected_zipcodes_gdf[affected_zipcodes_gdf['Number_of_Closures'] >
 ↪  0].shape[0]
print(f"Number of directly affected ZIP codes in Texas:
 ↪  {affected_zip_code_count}")
```

5.3

```
import geopandas as gpd

# Step 1: Create a GeoDataFrame for Directly Affected Zip Codes
# Assuming affected_zipcodes_gdf is your GeoDataFrame containing affected zip
 ↪  codes
directly_affected_gdf =
 ↪  affected_zipcodes_gdf[affected_zipcodes_gdf['Number_of_Closures'] >
 ↪  0].copy()

# Ensure geometries are present and valid
if 'geometry' not in directly_affected_gdf.columns:
    raise ValueError("The directly affected GeoDataFrame must contain a
     ↪  'geometry' column.")

# Step 2: Set the coordinate reference system (CRS) to a projected system for
 ↪  accurate distance measurements
# If you want to change the CRS, use to_crs() for transformation
# If you just want to replace the CRS, use allow_override=True
```

```
try:
    directly_affected_gdf = directly_affected_gdf.set_crs('EPSG:4326',
 ↪  allow_override=True)
except ValueError:
    directly_affected_gdf = directly_affected_gdf.to_crs(epsg=26914)  # Use a
 ↪  suitable projected CRS for Texas

# Step 3: Create a 10-Mile Buffer
buffer_distance = 10 * 1609.34  # Convert miles to meters
directly_affected_buffer = directly_affected_gdf.buffer(buffer_distance)

# Create a new GeoDataFrame for the buffer
buffer_gdf = gpd.GeoDataFrame(geometry=directly_affected_buffer,
 ↪  crs=directly_affected_gdf.crs)

# Step 4: Load the overall Texas ZIP code shapefile
# Assuming zipcodes_gdf is already loaded and contains Texas ZIP codes
zipcodes_gdf = zipcodes_gdf.to_crs(directly_affected_gdf.crs)  # Ensure the
 ↪  same CRS for the join

# Perform the spatial join to find indirectly affected ZIP codes
indirectly_affected_zipcodes = gpd.sjoin(zipcodes_gdf, buffer_gdf,
 ↪  how='inner', predicate='intersects')

# Count the number of indirectly affected ZIP codes
indirectly_affected_count = indirectly_affected_zipcodes['ZIP_CD'].nunique()
 ↪  # Unique ZIP codes

print(f"Number of indirectly affected ZIP codes in Texas:
 ↪  {indirectly_affected_count}")
```

5.4

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


# Step 1: Load the Texas ZIP code GeoDataFrame (make sure it contains only
 ↪  Texas ZIP codes)
# Assuming you have already filtered to Texas ZIP codes
# If you haven't loaded or filtered yet, do so here
```

```
texas_zipcodes = zipcodes_gdf[zipcodes_gdf['ZIP_CD'].str.startswith(('733',
 ↪  '75', '76', '77', '78', '79'))]

# Step 2: Create categories for ZIP codes
texas_zipcodes['category'] = 'Not Affected'  # Default category

# Directly affected ZIP codes
directly_affected_zipcodes = directly_affected_gdf['ZIP_CD'].unique()
texas_zipcodes.loc[texas_zipcodes['ZIP_CD'].isin(directly_affected_zipcodes),
 ↪  'category'] = 'Directly Affected'

# Indirectly affected ZIP codes
indirectly_affected_zipcodes =
 ↪  indirectly_affected_zipcodes['ZIP_CD'].unique()
texas_zipcodes.loc[texas_zipcodes['ZIP_CD'].isin(indirectly_affected_zipcodes),
 ↪  'category'] = 'Indirectly Affected'

# Step 3: Plotting the Choropleth Map
# Set color mapping for categories
color_map = {'Directly Affected': 'red', 'Indirectly Affected': 'orange',
 ↪  'Not Affected': 'lightgrey'}

# Create the plot
fig, ax = plt.subplots(1, 1, figsize=(12, 10))

# Plot Texas ZIP codes only
texas_zipcodes.plot(column='category', ax=ax,
 ↪  color=texas_zipcodes['category'].map(color_map), legend=True)

# Add a title and labels
plt.title('Texas ZIP Codes Affected by Hospital Closures (2016-2019)',
 ↪  fontsize=15)
plt.xlabel('Longitude', fontsize=12)
plt.ylabel('Latitude', fontsize=12)

# Remove axis for better visualization
plt.axis('off')

# Show the plot
plt.show()
```

Reflecting on the exercise (10 pts)

1. The problem with eliminating closures in zipcodes based on eqauting counts is that there are cases when a hospital may close and a new entirely diffrent hospital open ups. With the first pass model these closures will not be counted. However, there may be more complex conditions we can apply to fix this problem. For example, we can see those cases where a hospital closes and another opens and see if the names are similar and only then not count those as closures.

2. Looking at closures is not necessarily a metric of hospital accessibility even though it is a signal. I think this method shoud be supplemented by looking at overall numbers of hospitals per-captia. This can also be supplmented by looking at changes over time with both overall numbers and hospital accessibility.