PS4 Yuting Meng and Yunzhou Guo

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): Yuting Meng, yutingm
   - Partner 2 (name and cnet ID): Yunzhou Guo, guoy
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: YM, YG
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: 3
7. Knit your `ps4.qmd` to an PDF file to make `ps4.pdf`,
   - The PDF should not be more than 25 pages. Use `head()` and re-size figures when appropriate.
8. (Partner 1): push `ps4.qmd` and `ps4.pdf` to your github repo.
9. (Partner 1): submit `ps4.pdf` via Gradescope. Add your partner on Gradescope.
10. (Partner 1): tag your submission in Gradescope

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

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

1.

```python
import pandas as pd
pos2016 =
 ↪  pd.read_csv("POS_File_Hospital_Non_Hospital_Facilities_Q4_2016.csv")
print (pos2016.head())
print(pos2016.columns)
```

```
   PRVDR_CTGRY_SBTYP_CD  PRVDR_CTGRY_CD   CITY_NAME  \
0                   1.0               1      DOTHAN
1                   1.0               1  BRIDGEPORT
2                   1.0               1        BOAZ
3                   1.0               1    FLORENCE
4                   1.0               1         OPP

                           FAC_NAME PRVDR_NUM STATE_CD   ZIP_CD
0  SOUTHEAST ALABAMA MEDICAL CENTER    010001       AL  36301.0
1            NORTH JACKSON HOSPITAL    010004       AL  35740.0
2      MARSHALL MEDICAL CENTER SOUTH    010005       AL  35957.0
3     ELIZA COFFEE MEMORIAL HOSPITAL    010006       AL  35631.0
4          MIZELL MEMORIAL HOSPITAL    010007       AL  36467.0
Index(['PRVDR_CTGRY_SBTYP_CD', 'PRVDR_CTGRY_CD', 'CITY_NAME', 'FAC_NAME',
       'PRVDR_NUM', 'STATE_CD', 'ZIP_CD'],
      dtype='object')
```

The variables that I pulled are provider type code, provider subtype code, city name, facility name, provider number, state code, zip code.

2. a.

```
import pandas as pd
short_term_hospitals = pos2016[(pos2016['PRVDR_CTGRY_CD'] == 1) &
 ↪  (pos2016['PRVDR_CTGRY_SBTYP_CD'] == 1)]
num_hospitals = len(short_term_hospitals)
print(f"Number of short-term hospitals in 2016: {num_hospitals}")
```

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

b.

I used the American Hospital Association (AHA), which provides comprehensive
statistics on U.S. hospitals. According to the AHA's "Fast Facts on U.S.
Hospitals" for 2018, there were 5,534 registered hospitals in the U.S. in
2016.

link:
https://www.aha.org/system/files/2018-02/2018-aha-hospital-fast-facts.pdf
```

3.

```
file_names = [
    "POS_File_Hospital_Non_Hospital_Facilities_Q4_2016.csv",
    "POS_File_Hospital_Non_Hospital_Facilities_Q4_2017.csv",
    "POS_File_Hospital_Non_Hospital_Facilities_Q4_2018.csv",
    "POS_File_Hospital_Non_Hospital_Facilities_Q4_2019.csv"
]
years = [2016, 2017, 2018, 2019]
pos_data = []

for year, file in zip(years, file_names):
    try:
        pos_year = pd.read_csv(file, encoding='ISO-8859-1')

        short_term = pos_year[(pos_year['PRVDR_CTGRY_CD'] == 1) &
                              (pos_year['PRVDR_CTGRY_SBTYP_CD'] == 1)]

        short_term['Year'] = year

        pos_data.append(short_term)

    except UnicodeDecodeError as e:
        print(f"Could not read {file} due to encoding issues: {e}")
```

```
pos_all = pd.concat(pos_data, ignore_index=True)
print(pos_all.head())
```

```
/var/folders/gc/ws7810v915ng_kb_x8xx72n80000gn/T/ipykernel_84505/3337511794.py:17:
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['Year'] = year
/var/folders/gc/ws7810v915ng_kb_x8xx72n80000gn/T/ipykernel_84505/3337511794.py:17:
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['Year'] = year

   PRVDR_CTGRY_SBTYP_CD  PRVDR_CTGRY_CD   CITY_NAME  \
0                   1.0               1      DOTHAN
1                   1.0               1  BRIDGEPORT
2                   1.0               1        BOAZ
3                   1.0               1    FLORENCE
4                   1.0               1         OPP

                           FAC_NAME PRVDR_NUM STATE_CD   ZIP_CD  Year
0  SOUTHEAST ALABAMA MEDICAL CENTER    010001       AL  36301.0  2016
1            NORTH JACKSON HOSPITAL    010004       AL  35740.0  2016
2     MARSHALL MEDICAL CENTER SOUTH    010005       AL  35957.0  2016
3    ELIZA COFFEE MEMORIAL HOSPITAL    010006       AL  35631.0  2016
4          MIZELL MEMORIAL HOSPITAL    010007       AL  36467.0  2016

/var/folders/gc/ws7810v915ng_kb_x8xx72n80000gn/T/ipykernel_84505/3337511794.py:17:
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['Year'] = year
/var/folders/gc/ws7810v915ng_kb_x8xx72n80000gn/T/ipykernel_84505/3337511794.py:17:
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-versu
  short_term['Year'] = year
```

```python
import altair as alt

hospital_counts_by_year =
↪  pos_all.groupby('Year').size().reset_index(name='Count')

chart = alt.Chart(hospital_counts_by_year).mark_bar(size=15).encode(
    x=alt.X('Year:O', title='Year'),
    y=alt.Y('Count:Q', title='Number of Observations'),
    tooltip=['Year', 'Count']
).properties(
    width=170,
    title="Number of Short-Term Hospitals by Year (2016-2019)"
)

text = chart.mark_text(
    align='center',
    baseline='bottom',
    dy=-5
).encode(
    text='Count:Q'
)

chart+text
```

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

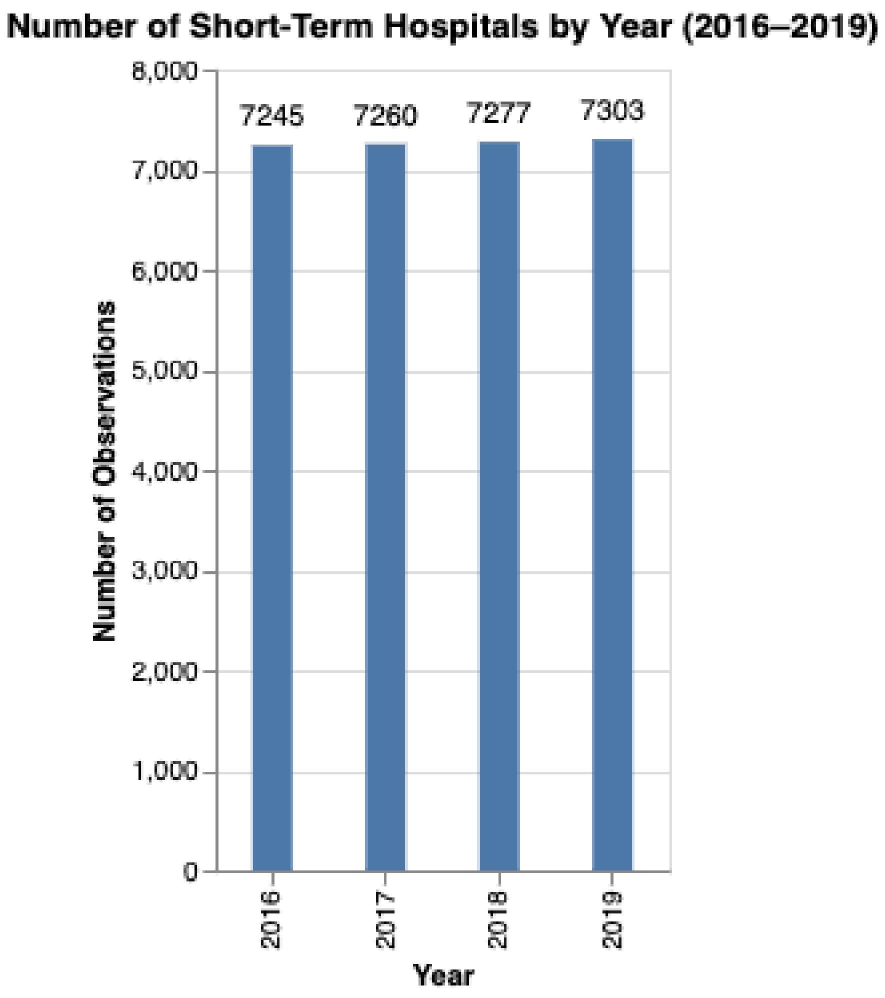

Figure 1: Number of short term hospitals by year

4.

a.

```
unique_hospitals_by_year =
↪  pos_all.groupby('Year')['PRVDR_NUM'].nunique().reset_index(name='Unique_Hospitals')


unique_hospitals_chart =
↪  alt.Chart(unique_hospitals_by_year).mark_bar(size=30).encode(x=alt.X('Year:O',
↪  title='Year'),
y=alt.Y('Unique_Hospitals:Q', title='Number of Unique Hospitals'),
    tooltip=['Year', 'Unique_Hospitals'] ).properties(
        width = 170,
         title="Number of Unique Short-Term Hospitals by Year (2016-2019)")

unique_hospitals_chart
```

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

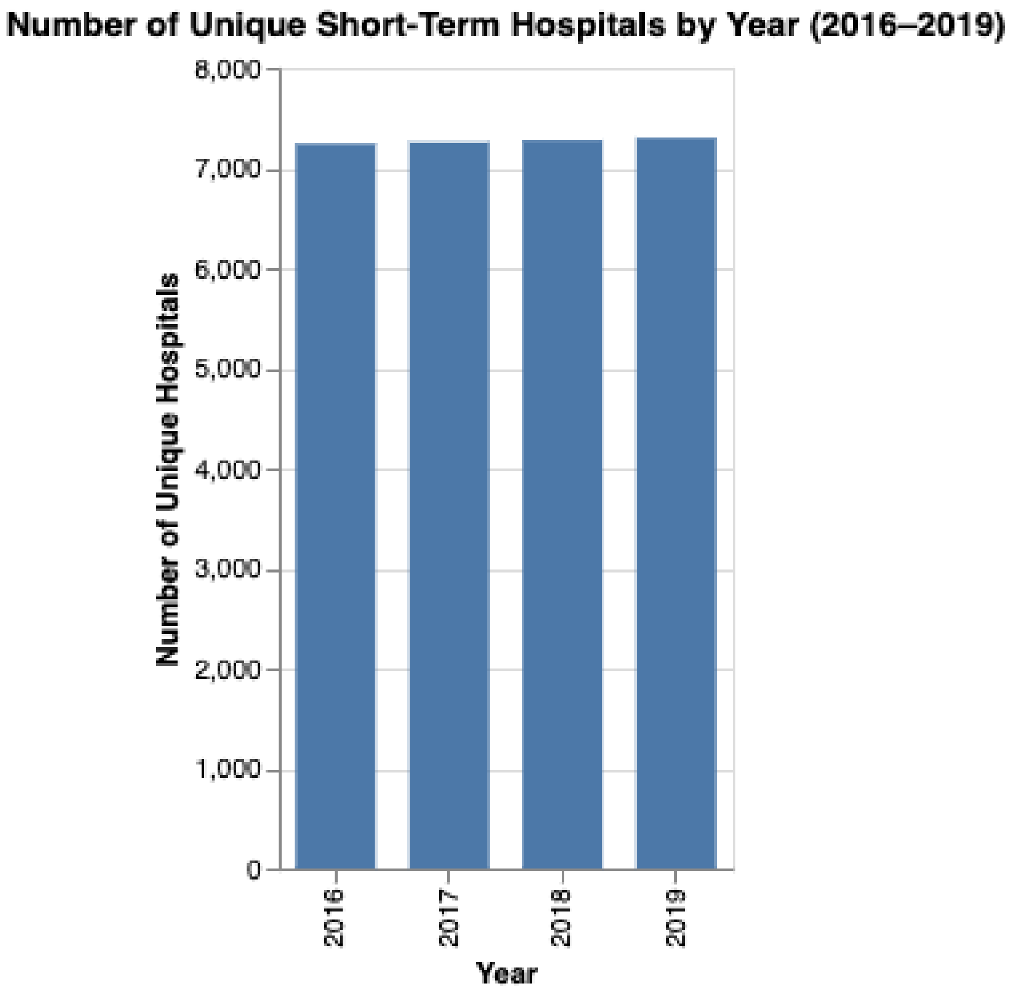

Figure 2: Number of Unique Short-term hospitals by year

b.

The presence of multiple entries for some hospitals in the same year suggests that hospitals may have multiple records within a year. This could be due to:

Multiple service types or specializations under the same certification number.

Data entries from different quarters if the dataset was compiled from quarterly reports.

Administrative updates that result in duplicate entries for the same hospital.

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

1.

Finding closures based on 'PRVDR_NUM'

```
import pandas as pd

pos2016 =
↪  pd.read_csv("POS_File_Hospital_Non_Hospital_Facilities_Q4_2016.csv",
↪  encoding="ISO-8859-1")
pos2017 =
↪  pd.read_csv("POS_File_Hospital_Non_Hospital_Facilities_Q4_2017.csv",
↪  encoding="ISO-8859-1")
pos2018 =
↪  pd.read_csv("POS_File_Hospital_Non_Hospital_Facilities_Q4_2018.csv",
↪  encoding="ISO-8859-1")
pos2019 =
↪  pd.read_csv("POS_File_Hospital_Non_Hospital_Facilities_Q4_2019.csv",
↪  encoding="ISO-8859-1")

print("Columns in 2016 data:", pos2016.columns.tolist())

active_2016 = pos2016[['FAC_NAME', 'ZIP_CD','STATE_CD','PRVDR_NUM']].copy()

def find_closures(active_df, *args):
    closures = active_df.copy()
    closures['Suspected_Closure_Year'] = None

    for year, df in enumerate(args, start=2017):
        closures.loc[~closures['PRVDR_NUM'].isin(df['PRVDR_NUM']),
↪  'Suspected_Closure_Year'] = year

    return closures.dropna(subset=['Suspected_Closure_Year'])
```

```python
suspected_closures = find_closures(active_2016, pos2017, pos2018, pos2019)

print("Total suspected closures:", len(suspected_closures))
print(suspected_closures.sort_values('FAC_NAME').head(10))
```

```
Columns in 2016 data: ['PRVDR_CTGRY_SBTYP_CD', 'PRVDR_CTGRY_CD', 'CITY_NAME',
'FAC_NAME', 'PRVDR_NUM', 'STATE_CD', 'ZIP_CD']
Total suspected closures: 3
                                                  FAC_NAME   ZIP_CD STATE_CD
                                                  \
11268   ADVENTIST HEALTH COMMUNITY CARE-REEDLEY WOMEN'...  93654.0       CA
117766                        REHABILITATION & HEALTHCARE  77521.0       TX
68832                    VETERANS ADMINISTRATION HOSPITAL  63125.0       MO

       PRVDR_NUM Suspected_Closure_Year
11268     058985                   2019
117766    45F481                   2019
68832     262307                   2019
```

There are 3 hospitals that fit this definition.

2.

```python
print(suspected_closures.sort_values('FAC_NAME')[['FAC_NAME',
 ↪  'Suspected_Closure_Year']].head(10))
```

```
                                                  FAC_NAME  \
11268   ADVENTIST HEALTH COMMUNITY CARE-REEDLEY WOMEN'...
117766                        REHABILITATION & HEALTHCARE
68832                    VETERANS ADMINISTRATION HOSPITAL

       Suspected_Closure_Year
11268                    2019
117766                   2019
68832                    2019
```

3.

```python
zip_counts_2016 = pos2016.groupby('ZIP_CD').size()
zip_counts_2017 = pos2017.groupby('ZIP_CD').size()
zip_counts_2018 = pos2018.groupby('ZIP_CD').size()
zip_counts_2019 = pos2019.groupby('ZIP_CD').size()

suspected_closures['is_merger'] = suspected_closures.apply(
    lambda row: (
        zip_counts_2016.get(row['ZIP_CD'], 0) ==
 ↪  zip_counts_2017.get(row['ZIP_CD'], 0) or
        zip_counts_2017.get(row['ZIP_CD'], 0) ==
 ↪  zip_counts_2018.get(row['ZIP_CD'], 0) or
        zip_counts_2018.get(row['ZIP_CD'], 0) ==
 ↪  zip_counts_2019.get(row['ZIP_CD'], 0)
    ), axis=1
)

filtered_closures = suspected_closures[~suspected_closures['is_merger']]

print("Total closures after filtering mergers:", len(filtered_closures))
print(filtered_closures.sort_values('FAC_NAME')[['FAC_NAME', 'ZIP_CD',
 ↪  'Suspected_Closure_Year']].head(10))
```

```
Total closures after filtering mergers: 0
Empty DataFrame
Columns: [FAC_NAME, ZIP_CD, Suspected_Closure_Year]
Index: []
```

```
a. Among the suspected closures, how many hospitals fit this definition of
potentially being a merger/acquisition?
b. After filtering out these potential mergers or acquisitions, 0 hospitals
remain in the list of closures.
c. Since there are no remaining hospitals after filtering, the sorted list of
corrected hospital closures is empty.
```

Download Census zip code shapefile (10 pt)

1. a.

   The five type of files are shp, dbf, shx, prj and xml. .shp (Shapefile) - The main file containing the geometry of shapes, such as points, lines, or polygons, for geographic features like ZIP code boundaries.

.shx (Shape Index Format) - An index file that enables quick access to the geometry data in the .shp file.

.dbf (dBASE Table) - A table file containing attribute data for each shape, including properties like ZIP codes, which complements the .shp file for map creation.

.prj (Projection) - A file that defines the projection and coordinate system, ensuring geographic data aligns correctly with other spatial datasets.

.xml (Metadata) - A metadata file that provides information about the dataset, including its source, creation date, and content details.

b.

The largest file is the .shp file (837.5 MB), as it contains the geometry data. The .dbf file (6.4 MB) is the next largest, containing attribute information. The .shx file (265 KB) is smaller, as it only contains an index. The .prj file (165 bytes) and the .xml file (16 KB) are small because they only contain projection and metadata information.

2.

```python
import geopandas as gpd

zip_codes = gpd.read_file("gz_2010_us_860_00_500k.shp", engine = "pyogrio")

# Display the first few rows to confirm loading
print(zip_codes.head())
```

```
            GEO_ID  ZCTA5   NAME   LSAD  CENSUSAREA  \
0  8600000US01040  01040  01040  ZCTA5      21.281
1  8600000US01050  01050  01050  ZCTA5      38.329
2  8600000US01053  01053  01053  ZCTA5       5.131
3  8600000US01056  01056  01056  ZCTA5      27.205
4  8600000US01057  01057  01057  ZCTA5      44.907

                                            geometry
0  POLYGON ((-72.62734 42.16203, -72.62764 42.162...
1  POLYGON ((-72.95393 42.34379, -72.95385 42.343...
2  POLYGON ((-72.68286 42.37002, -72.68287 42.369...
3  POLYGON ((-72.39529 42.18476, -72.39653 42.183...
4  MULTIPOLYGON (((-72.39191 42.08066, -72.39077 ...
```

```python
import json
```

```
texas_zip_codes = zip_codes[zip_codes['ZCTA5'].str.startswith(('75', '76',
↪  '77', '78'))]

pos2016['ZIP_CD'] = pos2016['ZIP_CD'].astype(str).str.replace('.0', '',
↪  regex=False)

hospitals_count = pos2016['ZIP_CD'].value_counts().reset_index()
hospitals_count.columns = ['ZCTA5', 'hospital_count']

texas_hospitals = texas_zip_codes.merge(hospitals_count, on='ZCTA5',
↪  how='left').fillna(0)
texas_hospitals['hospital_count'] =
↪  texas_hospitals['hospital_count'].astype(int)

print(texas_hospitals[['ZCTA5', 'hospital_count']].head())

texas_hospitals = texas_hospitals.to_crs("EPSG:4326")
geojson_data = json.loads(texas_hospitals.to_json())

choropleth =
↪  alt.Chart(alt.Data(values=geojson_data['features'])).mark_geoshape().encode(
    color=alt.Color('properties.hospital_count:Q', title="Number of
↪  Hospitals"),
    tooltip=[
        alt.Tooltip('properties.ZCTA5:O', title="ZIP Code"),
        alt.Tooltip('properties.hospital_count:Q', title="Hospital Count")

    ]
).properties(
    width=300,
    height=400,
    title="Hospitals per ZIP Code in Texas (2016)"
)

choropleth
```

```
   ZCTA5  hospital_count
0  78624              21
1  78626              25
2  78628              11
3  78631               0
4  78632               0
```

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

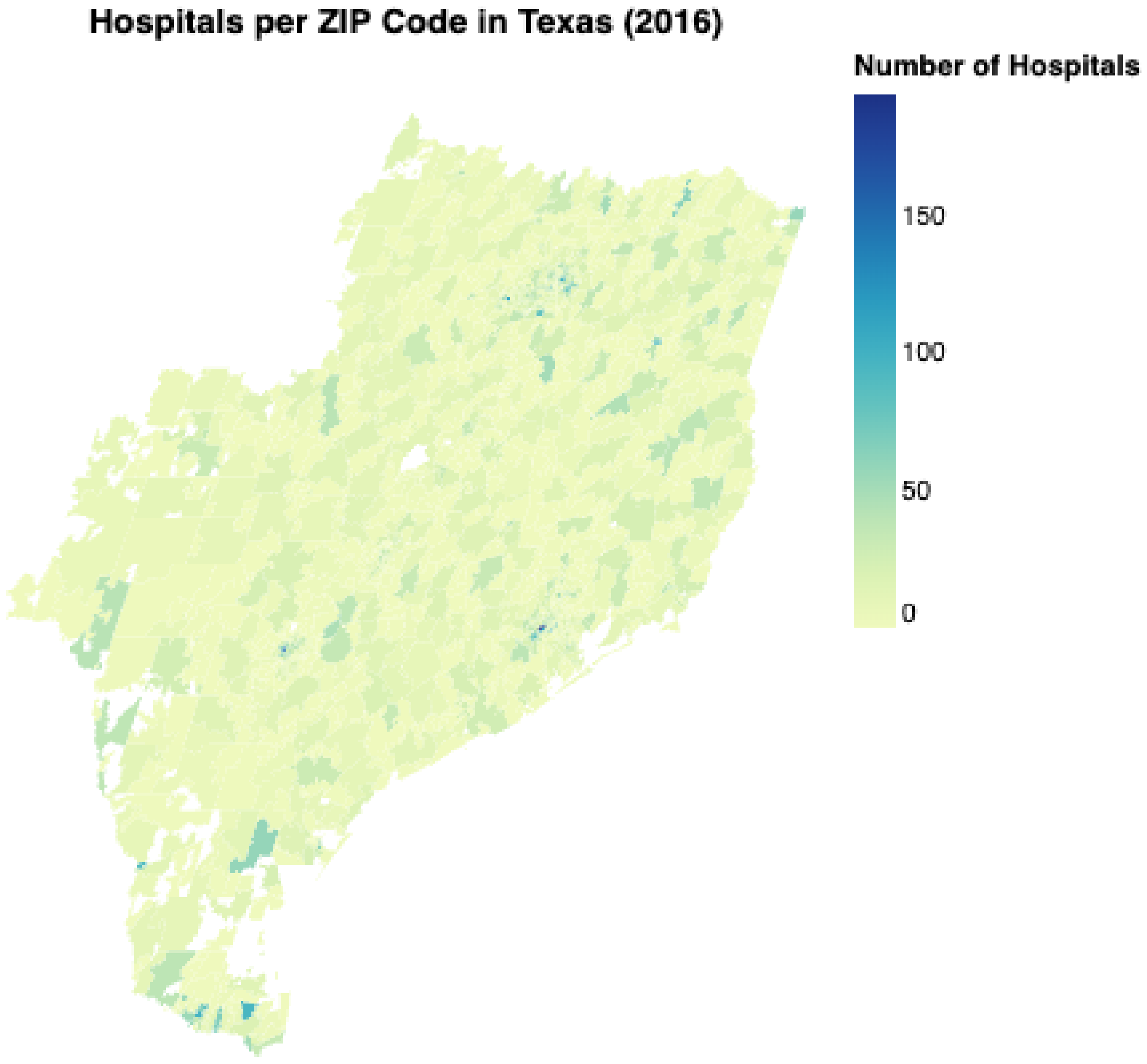

Figure 3: Choropleth

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

1.

```python
zip_codes['centroid'] = zip_codes.geometry.centroid
zips_all_centroids = zip_codes[['ZCTA5', 'centroid']]
print(f"Dimensions of zips_all_centroids: {zips_all_centroids.shape}")
```

```
/var/folders/gc/ws7810v915ng_kb_x8xx72n80000gn/T/ipykernel_84505/2749478377.py:1:
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.

  zip_codes['centroid'] = zip_codes.geometry.centroid

Dimensions of zips_all_centroids: (33120, 2)
```

2.

```python
texas_prefixes = ('75', '76', '77', '78')
border_states_prefixes = texas_prefixes + ('70', '71', '72', '73', '79',
↪  '80', '81', '82')

zips_texas_centroids =
↪  zips_all_centroids[zips_all_centroids['ZCTA5'].str.startswith(texas_prefixes)]
zips_texas_borderstates_centroids =
↪  zips_all_centroids[zips_all_centroids['ZCTA5'].str.startswith(border_states_prefixes)]

print(f"Unique ZIP codes in Texas subset:
↪  {zips_texas_centroids['ZCTA5'].nunique()}")
print(f"Unique ZIP codes in Texas and bordering states subset:
↪  {zips_texas_borderstates_centroids['ZCTA5'].nunique()}")
```

```
Unique ZIP codes in Texas subset: 1614
Unique ZIP codes in Texas and bordering states subset: 4034
```

3.

```python
pos2016['ZIP_CD'] = pos2016['ZIP_CD'].astype(str).str.zfill(5)
hospitals_in_2016 = pos2016[['ZIP_CD']].drop_duplicates()

zips_withhospital_centroids = zips_texas_borderstates_centroids.merge(
    hospitals_in_2016, left_on='ZCTA5', right_on='ZIP_CD', how='inner'
)
print(f"Total ZIP codes with at least one hospital:
↪  {len(zips_withhospital_centroids)}")
```

```
Total ZIP codes with at least one hospital: 2216
```

4.

a.

```python
import time
from shapely.geometry import Point
from geopy.distance import geodesic
# 1.4a Calculate distance for a subset of 10 zip codes
subset_zips = zips_texas_centroids.head(10)
start_time = time.time()

subset_zips['nearest_hospital_dist'] = subset_zips['centroid'].apply(
    lambda zip_point: zips_withhospital_centroids['centroid'].apply(
        lambda hospital_point: geodesic(
            (zip_point.y, zip_point.x),
            (hospital_point.y, hospital_point.x)
        ).miles
    ).min()
)

subset_time = time.time() - start_time
print(f"Time taken for subset (10 ZIP codes): {subset_time} seconds")
```

```
Time taken for subset (10 ZIP codes): 2.8513271808624268 seconds
```

```
/Users/mengyuting/Documents/GitHub/problem-set-4-yuting-yunzhou/.venv/lib/python3.8/site-pac
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-versu
  super().__setitem__(key, value)
```

It is faster than what I expected, around 5 seconds.

b.

start_time = time.time()

zips_texas_centroids[‘nearest_hospital_dist’] = zips_texas_centroids[‘centroid’].apply(lambda zip_point: zips_withhospital_centroids[‘centroid’].apply(lambda hospital_point: geodesic((zip_point.y, zip_point.x), (hospital_point.y, hospital_point.x)).miles).min())

full_time = time.time() - start_time print(f"Total time for full calculation: {full_time} seconds")

c.

```
print(zip_codes.crs)
```

```
EPSG:4269
```

The units are in latitude and longitude in degrees as units.

```
print("Columns in zips_texas_centroids:", zips_texas_centroids.columns)
print(zips_texas_centroids.head())

print("Columns in zips_withhospital_centroids:",
 ↪  zips_withhospital_centroids.columns)
print(zips_withhospital_centroids.head())
```

```
Columns in zips_texas_centroids: Index(['ZCTA5', 'centroid'], dtype='object')
      ZCTA5                    centroid
9207  78624  POINT (-98.87707 30.28160)
9208  78626  POINT (-97.59733 30.66535)
9209  78628  POINT (-97.75112 30.64108)
9210  78631  POINT (-99.30528 30.33772)
9211  78632  POINT (-97.47045 29.69633)
Columns in zips_withhospital_centroids: Index(['ZCTA5', 'centroid',
'ZIP_CD'], dtype='object')
   ZCTA5                    centroid ZIP_CD
0  70003  POINT (-90.21397 29.99864)  70003
1  70032  POINT (-89.99779 29.95816)  70032
2  70039  POINT (-90.38906 29.88151)  70039
3  70043  POINT (-89.96276 29.94804)  70043
4  70047  POINT (-90.36588 29.96953)  70047
```

zips_texas_centroids = gpd.GeoDataFrame(zips_texas_centroids, geometry='centroid') zips_withhospital_centroids = gpd.GeoDataFrame(zips_withhospital_centroids, geometry='centroid')

zips_texas_centroids = zips_texas_centroids.to_crs("EPSG:3857") zips_withhospital_centroids = zips_withhospital_centroids.to_crs("EPSG:3857")

def calculate_nearest_distance(row, hospitals_gdf): nearest_geom = nearest_points(row.geometry, hospitals_gdf.unary_union)[1] distance_meters = row.geometry.distance(nearest_geom) distance_miles = distance_meters * 0.000621371
return distance_miles

zips_texas_centroids['nearest_hospital_dist_miles'] = zips_texas_centroids.apply(calculate_nearest_distance, hospitals_gdf=zips_withhospital_centroids, axis=1)

from shapely.ops import nearest_points def calculate_nearest_distance(row, hospitals_gdf): # Find the nearest point in hospitals_gdf for each ZIP code point nearest_geom = nearest_points(row.geometry, hospitals_gdf.unary_union)[1] # Calculate the distance in meters and convert to miles distance_meters = row.geometry.distance(nearest_geom) distance_miles = distance_meters * 0.000621371 # Convert meters to miles return distance_miles

zips_texas_centroids['nearest_hospital_dist_miles'] = zips_texas_centroids.apply(calculate_nearest_distance, hospitals_gdf=zips_withhospital_centroids, axis=1)

print(zips_texas_centroids[['ZCTA5', 'nearest_hospital_dist_miles']])

I am not able to run this code chunk since it keeps giving me error message of "geometry."

Effects of closures on access in Texas (15 pts)

1.

```
for year, pos_data in zip([2017, 2018, 2019], [pos2017, pos2018, pos2019]):
    duplicates = pos_data['PRVDR_NUM'].duplicated().sum()
    print(f"Year {year}: {duplicates} duplicates found in PRVDR_NUM")

def find_closures_with_merger_check(active_df, *args):
    closures = active_df.copy()
    closures['Suspected_Closure_Year'] = None

    for year, df in enumerate(args, start=2017):
        df_unique = df.drop_duplicates(subset=['PRVDR_NUM'])  # Remove
 ↪  duplicates for clarity
        closures.loc[~closures['PRVDR_NUM'].isin(df_unique['PRVDR_NUM']),
 ↪  'Suspected_Closure_Year'] = year

    return closures.dropna(subset=['Suspected_Closure_Year'])

suspected_closures = find_closures_with_merger_check(active_2016, pos2017,
 ↪  pos2018, pos2019)
```

```
texas_closures = suspected_closures[suspected_closures['STATE_CD'] == 'TX']

print("Texas closures identified:")
print(texas_closures)
```

```
Year 2017: 0 duplicates found in PRVDR_NUM
Year 2018: 0 duplicates found in PRVDR_NUM
Year 2019: 0 duplicates found in PRVDR_NUM
Texas closures identified:
                           FAC_NAME   ZIP_CD STATE_CD PRVDR_NUM  \
117766  REHABILITATION & HEALTHCARE  77521.0       TX    45F481

       Suspected_Closure_Year
117766                   2019
```

2.

import json import altair as alt

texas_closures = filtered_closures[filtered_closures['STATE_CD'] == 'TX'] closures_by_zip = texas_closures.groupby('ZIP_CD').size().reset_index(name='Closure_Count')

texas_zip_data = zip_codes[zip_codes['ZCTA5'].str.startswith(('75', '76', '77', '78'))] closures_geo = texas_zip_data.merge(closures_by_zip, left_on='ZCTA5', right_on='ZIP_CD', how='left') closures_geo['Closure_Count'] = closures_geo['Closure_Count'].fillna(0).astype(int)

closures_geo_no_geom = closures_geo.drop(columns='geometry') geojson_data = json.loads(closures_geo_no_geom.to_json())

choropleth = alt.Chart(alt.Data(values=geojson_data['features'])).mark_geoshape().encode(color=alt.Color('properties.Closure_Count:Q', title="Closures"), tooltip=[alt.Tooltip('properties.ZCTA5:O', title="ZIP Code"), alt.Tooltip('properties.Closure_Count:Q', title="Closure Count")]).properties(width=600, height=400, title="Texas ZIP Codes Directly Affected by Hospital Closures (2016–2019)")

choropleth.display() print(f"Total directly affected ZIP codes in Texas: {closures_geo[closures_geo['Closure_Co
> 0].shape[0]}")

I am also having issues with this graph. The error message displays: AttributeError: No geometry data set (expected in column 'geometry').

3.

import geopandas as gpd from shapely.geometry import Point from shapely.ops import nearest_points

directly_affected_geo = closures_geo[closures_geo['Closure_Count'] > 0] directly_affected_geo = gpd.GeoDataFrame(directly_affected_geo, geometry='geometry') directly_affected_geo = directly_affected_geo.set_crs("EPSG:4269")

directly_affected_geo = directly_affected_geo.to_crs("EPSG:3857") directly_affected_geo['buffered'] = directly_affected_geo.buffer(16093.4)

texas_zip_data = texas_zip_data.to_crs("EPSG:3857") indirectly_affected = gpd.sjoin(texas_zip_data, directly_affected_geo[['buffered']], op='intersects')

indirectly_affected = indirectly_affected[~indirectly_affected['ZCTA5'].isin(directly_affected_geo['ZCTA5'])]

indirectly_affected_count = indirectly_affected['ZCTA5'].nunique() print(f"Total indirectly affected ZIP codes in Texas: {indirectly_affected_count}")

Reflecting on the exercise (10 pts)

1.

Partner 1 Reflection: Addressing Incorrectly Identified Closures The "first-pass" approach of identifying hospital closures based on termination in the dataset has limitations. Hospitals may change certification numbers or rebrand due to mergers, acquisitions, or administrative adjustments without actually ceasing operations. This could lead to a false identification of closures. Additionally, data gaps or errors may cause hospitals to appear inactive in the dataset when they are operational. Temporary closures due to renovations or emergencies could also be misclassified as permanent closures. These issues highlight the need for additional data sources, such as local health department records, to verify suspected closures, allowing for a more accurate classification.

Moreover, some hospitals may switch from general hospital classification to specialty facilities or outpatient centers, meaning they continue operating under a different certification type. This could lead to an incomplete picture of access loss if these transformations aren't recognized in the data. To improve accuracy, it may be beneficial to flag facilities undergoing changes in certification and implement follow-up checks to account for these transformations. By incorporating hospital capacity and service volume data, the analysis could further distinguish true closures from changes in operation type. These enhancements would help minimize misclassification and ensure a more accurate reflection of hospital access.

2.

Identifying ZIP codes affected by closures based solely on proximity provides a basic measure of access, but it doesn't fully capture the impact of closures on healthcare availability. While the 10-mile radius approach covers geographic closeness, it doesn't account for hospital capacity, services offered, or patient load. As a result, ZIP codes within this radius may still experience limited healthcare resources, especially in densely populated areas with high healthcare needs. By incorporating factors like hospital bed counts, service volume, and demographic information (such as elderly population or prevalence of chronic illnesses), we could better measure the true impact of closures on ZIP code-level access.

Furthermore, this proximity-based approach doesn't consider travel barriers or variations in healthcare service types. Factors like transportation availability, travel time, and geographic obstacles can significantly affect a community's ability to access hospitals even within a 10-mile radius. For example, the closure of a trauma center might disproportionately affect nearby ZIP codes lacking similar specialty services. By using travel time instead of distance and accounting for different hospital services, we could improve the accuracy of this access measure, providing a clearer picture of the healthcare landscape following closures.