30538 Problem Set 4

Yuliana Zhang and Dale Jin

2001-10-30

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):
    - Partner 2 (name and cnet ID):
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: **___** **___**
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: **___** Late coins left after submission: **___**
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)

```
import pandas as pd
import altair as alt
alt.renderers.enable("png")
import geopandas as gpd
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')
```

1.

I pulled the following 7 variables:

PRVDR_CTGRY_SBTYP_CD (1 = short term), PRVDR_CTGRY_CD (1 = hospitals), PRVDR_NUM, PGM_TRMNTN_CD (00 = active provider), ZIP_CD, TRMNTN_EXPRTN_DT (not necessarily need), FAC_NAME.

2.

```
# Load 2016 CSV file
pos2016 = pd.read_csv("pos2016.csv")
pos2016['year'] = 2016

# Define a function to focus on short term hospitals
def filter_short_term_hospitals(data):
    return data[
        (data['PRVDR_CTGRY_SBTYP_CD'] == 1) &
        (data['PRVDR_CTGRY_CD'] == 1)
    ]
```

```
# Subset 2016
short_term_pos2016 = filter_short_term_hospitals(pos2016)
```

a.

```
# Count the number of short-term hospitals
hospitals_2016 = len(short_term_pos2016)
print(f'There are {hospitals_2016} short-term hospitals in 2016')

# Check whether there are closed hospitals
for cell in short_term_pos2016['PGM_TRMNTN_CD']:
    if cell == 1:
        print('This number of 2016 short-term hospitals does not make sense
        ↪  as it contains closed hospitals')
        break
```

```
There are 7245 short-term hospitals in 2016
This number of 2016 short-term hospitals does not make sense as it contains
closed hospitals
```

b. From the backgroud article provided in the assignment, "there are nearly 5,000 short-term, acute care hospitals in the United States", ref from "https://www.kff.org/report-section/a-look-at-rural-hospital-closures-and-implications-for-access-to-care-three-case-studies-issue-brief/". I think these numbers are different because the dataset contains a lot closed hospitals.

3.

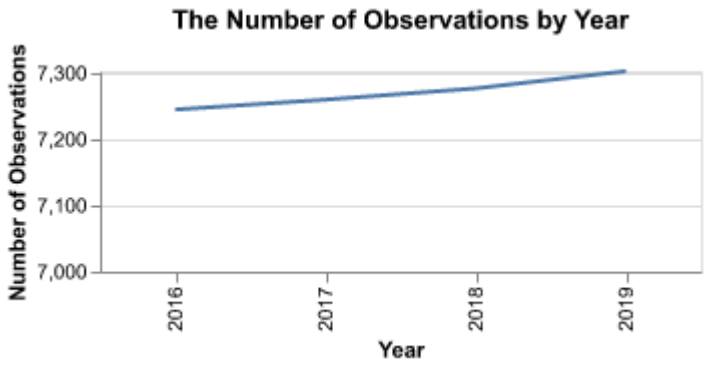
```
# Load 2017 data
pos2017 = pd.read_csv("pos2017.csv")
pos2017['year'] = 2017
# Subset 2017
short_term_pos2017 = filter_short_term_hospitals(pos2017)
# Count the number of short-term hospitals
hospitals_2017 = len(short_term_pos2017)
print(f'There are {hospitals_2017} short-term hospitals in 2017')
# Check whether there are closed hospitals
for cell in short_term_pos2017['PGM_TRMNTN_CD']:
    if cell == 1:
        print('This number of 2017 short-term hospitals does not make sense
        ↪  as it contains closed hospitals')
```

```
        break

# Load 2018 data
pos2018 = pd.read_csv("pos2018.csv")
pos2018['year'] = 2018
# Subset 2018
short_term_pos2018 = filter_short_term_hospitals(pos2018)# Count the number
 ↪  of short-term hospitals
hospitals_2018 = len(short_term_pos2018)
print(f'There are {hospitals_2018} short-term hospitals in 2018')
# Check whether there are closed hospitals
for cell in short_term_pos2018['PGM_TRMNTN_CD']:
    if cell == 1:
        print('This number of 2018 short-term hospitals does not make sense
         ↪  as it contains closed hospitals')
        break

# Load 2019 data
pos2019 = pd.read_csv("pos2019.csv")
pos2019['year'] = 2019
# Subset 2019
short_term_pos2019 = filter_short_term_hospitals(pos2019)
# Count the number of short-term hospitals
hospitals_2019 = len(short_term_pos2019)
print(f'There are {hospitals_2019} short-term hospitals in 2019')
# Check whether there are closed hospitals
for cell in short_term_pos2019['PGM_TRMNTN_CD']:
    if cell == 1:
        print('This number of 2019 short-term hospitals does not make sense
         ↪  as it contains closed hospitals')
        break
```

```
There are 7260 short-term hospitals in 2017
This number of 2017 short-term hospitals does not make sense as it contains
closed hospitals
There are 7277 short-term hospitals in 2018
This number of 2018 short-term hospitals does not make sense as it contains
closed hospitals
There are 7303 short-term hospitals in 2019
This number of 2019 short-term hospitals does not make sense as it contains
closed hospitals
```

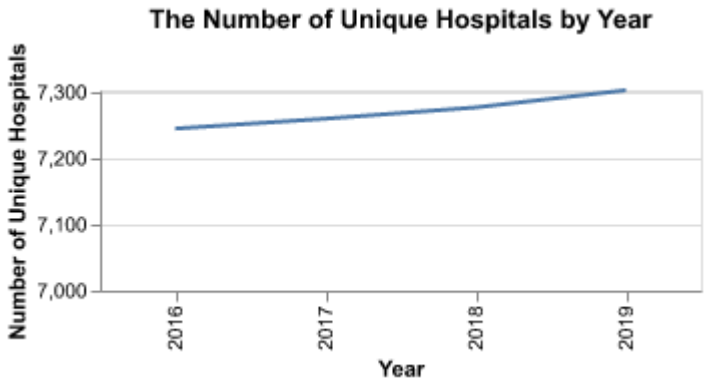
Same reasons as question 2.

```
# ref from
 ↪  https://www.askpython.com/python-modules/pandas/combine-csv-files-using-python
# Append 2016-2019 data
short_term_pos2016_2019 = pd.concat([short_term_pos2016, short_term_pos2017,
 ↪  short_term_pos2018, short_term_pos2019], ignore_index=True)

# Build the overall dataframe
pos = short_term_pos2016_2019

# Plot the number of observations by year.
## Rebuild data frame for chart
observations = pos.groupby(['year']).size().reset_index(name = 'count')

chart = alt.Chart(observations, title = 'The Number of Observations by
 ↪  Year').mark_line().encode(
    alt.X('year:O', title = 'Year'),
    alt.Y('count:Q', title = 'Number of Observations', scale =
 ↪  alt.Scale(domainMin = 7000)),
).properties(
    width = 300,
    height = 100
)
chart.show()
```



4.a

```
# Filter unique hospitals by dropping duplicates based on 'CMS_ID' and 'Year'
unique_hospitals = pos.drop_duplicates(subset = ['PRVDR_NUM', 'year'])

# Count unique hospitals per year
unique_hospitals =
 ↪  unique_hospitals.groupby('year')['PRVDR_NUM'].size().reset_index(name =
 ↪  'unique_hospitals')

# Plot the number of unique CMS by year.
chart = alt.Chart(unique_hospitals, title = 'The Number of Unique Hospitals
 ↪  by Year').mark_line().encode(
    alt.X('year:O', title = 'Year'),
    alt.Y('unique_hospitals:Q', title = 'Number of Unique Hospitals', scale =
 ↪  alt.Scale(domainMin = 7000)),
).properties(
    width = 300,
    height = 100
)
chart.show()
```



4.b

These two graphs are almost the same, which shows that the dataset does not contain repeated hospitals in each year. Thus, the reason for question 2 are solely based on the closed hospitals.

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

1.

```
pos['ZIP_CD'] = pos['ZIP_CD'].astype(str) # convert zipcode to str

# find the close pos after 2016, subgroup
close = pos[(pos["PGM_TRMNTN_CD"] == 1) & (pos["year"] > 2016)]
close = close.drop_duplicates(subset=["FAC_NAME"], keep="first")
close = close[["FAC_NAME", "ZIP_CD", "year"]]

print(len(close), "hospitals that were active in 2016 were suspected to have
 ↪  closed by 2019.")
```

```
2051 hospitals that were active in 2016 were suspected to have closed by
2019.
```

2.

```
close_sorted = close.sort_values(by = "FAC_NAME",
 ↪  ascending=True).reset_index(drop=True)
close_sorted.head(10)
```

	FAC_NAME	ZIP_CD	year
0	(CLOSED) BAPTIST HOSPITAL FOR WOMEN	37934.0	2017
1	(CLOSED) BAPTIST HOSPITAL OF ROANE COUNTY	37854.0	2017
2	(CLOSED) BAPTIST MEMORIAL HOSPITAL LAUDERDALE	38063.0	2017
3	(CLOSED) BAPTIST MEMORIAL HOSPITAL-COLLIERVILLE	38017.0	2017
4	(CLOSED) BAPTIST REHABILITATION GERMANTOWN	38138.0	2019
5	(CLOSED) BAPTIST WOMEN'S TREATMENT CENTER	37236.0	2019
6	(CLOSED) BAPTIST WOMENS TREATMENT CTR MURFREES...	37130.0	2019
7	(CLOSED) CAMDEN GENERAL HOSPITAL	38320.0	2017
8	(CLOSED) COLUMBIA WHITWELL MEDICAL CENTER INC	37397.0	2017
9	(CLOSED) COLUMIBA EAST RIDGE HOSP	37412.0	2017

3.

```
# number of active hospital by zipcode
active = pos[(pos["PGM_TRMNTN_CD"] == 0) & (pos["year"] > 2016)]
active_zip = active.groupby(["ZIP_CD", "year"]).agg({
    "FAC_NAME" : "count",
}).reset_index()

# number of active hospitals does not decrease
```

```
active_zip['future'] = active_zip.groupby('ZIP_CD')['FAC_NAME'].shift(-1)
active_diff = active_zip[active_zip["future"] >= active_zip["FAC_NAME"]]

# remove any suspected hospital closures
close["suspect"] = close["ZIP_CD"].isin(active_diff["ZIP_CD"])
active_clean = close[close["suspect"] == False]
```

b.

```
suspected_closure = close[close["suspect"]]
print(len(suspected_closure), "hospitals fit this definition of potentially
 ↪  being a merger/acquisition.")

print(len(active_clean), "hospitals have left after correcting for this.")
```

```
633 hospitals fit this definition of potentially being a merger/acquisition.
1418 hospitals have left after correcting for this.
```

c.

```
active_clean_sorted = active_clean.sort_values(by = "FAC_NAME",
 ↪  ascending=True).reset_index(drop=True)
active_clean_sorted.head(10)
```

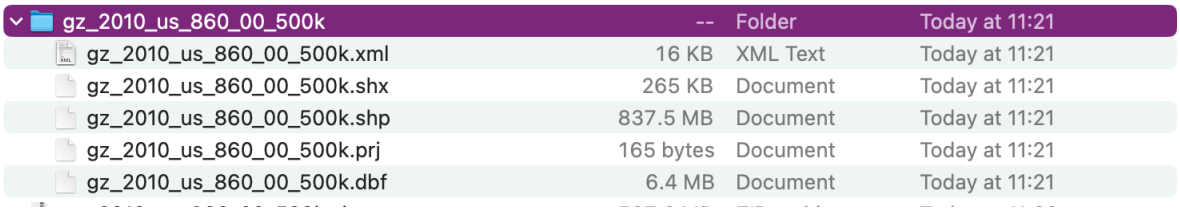
	FAC_NAME	ZIP_CD	year	suspect
0	(CLOSED) BAPTIST HOSPITAL FOR WOMEN	37934.0	2017	False
1	(CLOSED) BAPTIST HOSPITAL OF ROANE COUNTY	37854.0	2017	False
2	(CLOSED) BAPTIST MEMORIAL HOSPITAL LAUDERDALE	38063.0	2017	False
3	(CLOSED) BAPTIST MEMORIAL HOSPITAL-COLLIERVILLE	38017.0	2017	False
4	(CLOSED) BAPTIST REHABILITATION GERMANTOWN	38138.0	2019	False
5	(CLOSED) BAPTIST WOMENS TREATMENT CTR MURFREES...	37130.0	2019	False
6	(CLOSED) CAMDEN GENERAL HOSPITAL	38320.0	2017	False
7	(CLOSED) COLUMBIA WHITWELL MEDICAL CENTER INC	37397.0	2017	False
8	(CLOSED) COLUMIBA EAST RIDGE HOSP	37412.0	2017	False
9	(CLOSED) COPPER BASIN MEDICAL CENTER	37317.0	2017	False

Download Census zip code shapefile (10 pt)

1.a

The files in the folder contain the following 5 type files:

```
from IPython.display import Image
Image("file_type.png")
```



.shp (Shapefile): This is the main file that stores the geometric shape data.

.shx (Shape Index Format): This index file provides the spatial index for locating the features in the .shp file.

.dbf (Database File): This file contains attribute data in a tabular format, which is associated with each geometric shape in the .shp file.

.prj (Projection File): This file contains information about the coordinate system and projection used in the .shp file.

.xml (Metadata): This file may contain metadata information about the shapefile, describing the content, source, creation date, and other relevant details.

1.b

From the image, we can see the sizes of the unzipped files:

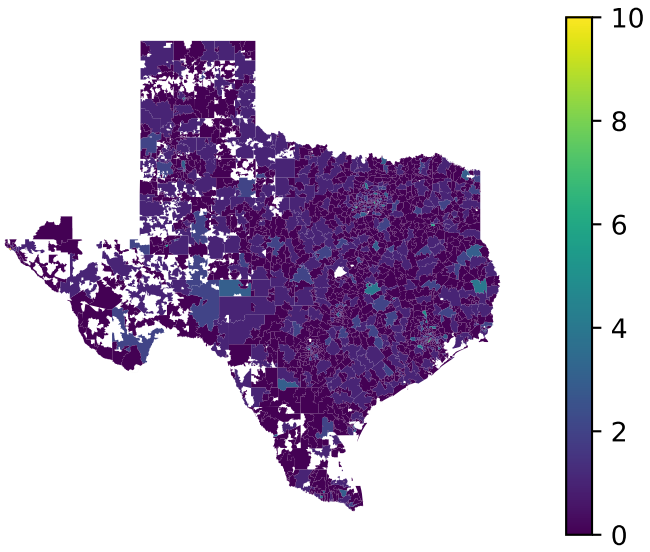
gz_2010_us_860_00_500k.shp: 837.5 MB gz_2010_us_860_00_500k.shx: 265 KB gz_2010_us_860_00_500k.dbf: 6.4 MB gz_2010_us_860_00_500k.prj: 165 bytes gz_2010_us_860_00_500k.xml: 16 KB

This size information indicates that the .shp file is the largest as it holds the spatial data.

2.

```
# Load the shapefile
zip_codes = gpd.read_file("gz_2010_us_860_00_500k.shp")
```

```
# Texas zip code prefixes based on wiki
# Ref from https://simple.wikipedia.org/wiki/List_of_ZIP_Code_prefixes#700s
texas_prefixes = ['733'] + [str(prefix) for prefix in range(750, 800)]
# Filter based on the first three characters of the ZIP column

texas_zip_codes = zip_codes[zip_codes['ZCTA5'].str[:3].isin(texas_prefixes)]
# Ensure consistent string format

texas_zip_codes['ZCTA5'] = texas_zip_codes['ZCTA5'].astype(str).str.strip()

# Convert ZIP_CD to string
short_term_pos2016['ZIP_CD'] = pd.to_numeric(short_term_pos2016['ZIP_CD'],
 ↪  errors='coerce').fillna(0).astype(int)
short_term_pos2016['ZIP_CD'] = short_term_pos2016['ZIP_CD'].astype(str)

# Calculate the number of hospitals per zip code in 2016
hospitals_per_zip =
 ↪  short_term_pos2016.groupby('ZIP_CD').size().reset_index(name =
 ↪  'hospitals')

# Merge with Texas zip code data
texas_hospitals = texas_zip_codes.merge(hospitals_per_zip, left_on='ZCTA5',
 ↪  right_on='ZIP_CD', how='left')
texas_hospitals['hospitals'] = texas_hospitals['hospitals'].fillna(0)

# Draw the choropleth plot
texas_hospitals.plot(column = "hospitals", legend=True).set_axis_off()
```



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

1.

```
# Calculate centroids
zip_codes["centroid"] = zip_codes["geometry"].centroid

# Create a new GeoDataFrame for centroids
zips_all_centroids = zip_codes[["ZCTA5", "centroid"]]
zips_all_centroids = gpd.GeoDataFrame(zips_all_centroids,
 ↪  geometry="centroid")

print("Dimensions of zips_all_centroids:", zips_all_centroids.shape)
```

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

ZCTA5: the ZIP code for each geographic area. Each row is a ZIP code. geometry: each ZIP code's polygonal boundary. centroid: the centroid points of each ZIP code area, representing the center of each ZIP code's polygonal boundary.

2.

```
# all zip codes in Texas
zips_texas_centroids =
 ↪  zips_all_centroids[zips_all_centroids['ZCTA5'].str[:3].isin(texas_prefixes)]

# borderstates zip code prefixes based on wiki
texas_borderstates_prefixes = [str(prefix) for prefix in range(700, 800)] +
 ↪  [str(prefix) for prefix in range(870, 886)]

# all zip codes in Texas or a bordering state
zips_texas_borderstates_centroids =
 ↪  zips_all_centroids[zips_all_centroids['ZCTA5'].str[:3].isin(texas_borderstates_prefixes)]

unique_texas_zips = zips_texas_centroids['ZCTA5'].nunique()
unique_borderstate_zips =
 ↪  zips_texas_borderstates_centroids['ZCTA5'].nunique()

# Display the counts
print(f"Unique ZIP codes in Texas: {unique_texas_zips,
 ↪  len(zips_texas_centroids)}")
print(f"Unique ZIP codes in Texas and bordering states:
 ↪  {unique_borderstate_zips, len(zips_texas_borderstates_centroids)}")
```

```
Unique ZIP codes in Texas: (1935, 1935)
Unique ZIP codes in Texas and bordering states: (4057, 4057)
```

3.

```
# subset of zips_texas_borderstates_centroids
zips_withhospital_centroids =
 ↪  zips_texas_borderstates_centroids.merge(hospitals_per_zip,
 ↪  left_on='ZCTA5', right_on='ZIP_CD', how='left')

# zip codes with at least 1 hospital
zips_withhospital_centroids =
 ↪  zips_withhospital_centroids[zips_withhospital_centroids["hospitals"] >=
 ↪  1]
```

I decide to do a left merge between zips_texas_borderstates_centroids and hospitals_per_zip. This keeps all columns from the zips_texas_borderstates_centroids and only adds hospitals column with rows from hospitals_per_zip where there is a matching ZIP code. I merge on the ZIP code columns, specifically ZCTA5 in zips_texas_borderstates_centroids and ZIP_CD in hospitals_per_zip.

4. a.

```
import time
# subset to 10 zip codes
zips_texas_centroids_subset = zips_texas_centroids.head(10)

start_time = time.time()

# calculate the distance to the nearest zip code
subset_distances = gpd.sjoin_nearest(
    zips_texas_centroids_subset,
    zips_withhospital_centroids,
    how="inner",
    distance_col="distance"
)

end_time = time.time()
subset_time = end_time - start_time

print(f"Time taken for 10 ZIP codes: {subset_time:.2f} seconds")

# estimate the entire procedure
estimate_time = len(zips_texas_centroids) / 10 * subset_time

print(f"Estimated time for entire dataset: {estimate_time:.2f} seconds")
```

```
Time taken for 10 ZIP codes: 0.01 seconds
Estimated time for entire dataset: 1.06 seconds
```

b.

```
start_time = time.time()

# calculate the distance to the nearest zip code
true_distances = gpd.sjoin_nearest(
    zips_texas_centroids,
    zips_withhospital_centroids,
    how="inner",
    distance_col="distance"
)

end_time = time.time()
```

```
true_time = end_time - start_time

print(f"True time for entire dataset: {true_time:.2f} seconds")
```

```
True time for entire dataset: 0.01 seconds
```

The true time is not close to estimation.

```
c.
The unit in .prj file is in ["Degree",0.017453292519943295], which is an
angular unit used in geographic coordinate systems. According to the
internet, 1 degree is about 69 miles.
ref:https://www.nwcg.gov/course/ffm/location/610-latitude-longitude#:~:text=One%20degree%20o
```

5. a. The unit is in "Degree"

   b.

```
average_distance = true_distances["distance"].mean()
distance_mile = average_distance * 69

print(f"The average distance in miles is {distance_mile:.2f}")
```

```
The average distance in miles is 8.86
```

This number make sense.

```
c.
```

Effects of closures on access in Texas (15 pts)

1.
2.
3.
4.

Reflecting on the exercise (10 pts)