Pset4

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): Andrew White awhite6
    - Partner 2 (name and cnet ID): Justine Silverstein silversteinj
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: JS AW
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: 1 and 2 for Justine, 1 and 1 for Andrew
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 packages

```
import pandas as pd
import numpy as np
import altair as alt
import os
import json
alt.renderers.enable("png")
import shapely
import geopandas as gpd
```

1.

```
#raw_health =
 ↪  r"/Users/justinesilverstein/Desktop/problem-set-4-andrew-justine"

raw_health = r"C:/Users/andre/Documents/GitHub/problem-set-4-andrew-justine"
path_health16 = os.path.join(raw_health, "pos2016.csv")
health16 = pd.read_csv(path_health16)

path_health17 = os.path.join(raw_health, "pos2017.csv")
health17 = pd.read_csv(path_health17)

path_health18 = os.path.join(raw_health, "pos2018.csv")
health18 = pd.read_csv(path_health18, encoding = "ISO-8859-1")

path_health19 = os.path.join(raw_health, "pos2019.csv")
health19 = pd.read_csv(path_health19, encoding = "ISO-8859-1")
```

```
C:\Users\andre\AppData\Local\Temp\ipykernel_9792\3728273300.py:8:
DtypeWarning: Columns (11) have mixed types. Specify dtype option on import
or set low_memory=False.
  health17 = pd.read_csv(path_health17)
```

Attribution: I asked ChatGPT how to solve the unicode errors I was encountering, and I was advised to try out different encodings, including "ISO-8859-1".

Q1 Answer:

```
variables_list = health16.columns

print("These are the variables I selected: ", variables_list)
```

```
These are the variables I selected:  Index(['PRVDR_CTGRY_SBTYP_CD',
'PRVDR_CTGRY_CD', 'FAC_NAME', 'PRVDR_NUM',
       'STATE_CD', 'SSA_STATE_CD', 'ST_ADR', 'PGM_TRMNTN_CD',
       'TRMNTN_EXPRTN_DT', 'ZIP_CD', 'FIPS_STATE_CD', 'FY_END_MO_DAY_CD'],
      dtype='object')
```

2. Function for subsetting the datasets to short term hospitals

```
def short_term(X):
  Y = X[(X["PRVDR_CTGRY_SBTYP_CD"] == 1) & (X["PRVDR_CTGRY_CD"] == 1)]
  return(Y)
```

Run function on health16, creating a count of hospitals

```
short16 = short_term(health16)

print("There are ", len(short16), " hospitals in this dataset")
```

```
There are  7245  hospitals in this dataset
```

a. There are 7,245 short term
   hospitals in the 2016 dataset. This number seems
   a bit low.
b. Upon re-reading some of the article provided in this pset and taking
   a look at data from Statista
   (https://www.statista.com/statistics/185843/number-of-all-hospitals-in-the-us/)
   I see that this number appears to be, in fact, a bit high. The
   article provided in the dataset places the number of short term,
   acute care hospitals at around 5000, while in the Statology
   explainer the total number of U.S. hospitals in 2016 is pegged
   at around 5,500. Some potential causes for this discrepancy may include
   varying definitions of hospital, and particularly short term hospital;
   the fact that, potentially, the data from Statista and the Kaiser

Family Foundation consists of hospital numbers as they stood at a different time than the time in which the data were recorded for this dataset (e.g.: this dataset covers Q4, perhaps other sources gather data for Q1? Or average over an entire year.)

3.

short term function on health17

```
short17 = short_term(health17)
```

short term function on health18

```
short18 = short_term(health18)
```

```
short19 = short_term(health19)
```

Appending the data together

```
# add a "year" column to each dataset, for future disagg
def add_column(X, the_year):
  year_col = []
  i = 0
  while i < len(X):
    year_col.append(the_year)
    i += 1
  X["Year"] = year_col
  return(X)

#run on each dataframe
add_column(short16, 2016)

add_column(short17, 2017)

add_column(short18, 2018)

add_column(short19, 2019)
```

```
C:\Users\andre\AppData\Local\Temp\ipykernel_9792\2695025422.py:8:
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
  X["Year"] = year_col
C:\Users\andre\AppData\Local\Temp\ipykernel_9792\2695025422.py:8:
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
  X["Year"] = year_col
C:\Users\andre\AppData\Local\Temp\ipykernel_9792\2695025422.py:8:
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
  X["Year"] = year_col
C:\Users\andre\AppData\Local\Temp\ipykernel_9792\2695025422.py:8:
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
  X["Year"] = year_col
```

	PRVDR_CTGRY_SBTYP_CD	PRVDR_CTGRY_CD	FAC_NAME
0	1.0	1	SOUTHEAST ALABAMA MEDICAL C
1	1.0	1	NORTH JACKSON HOSPITAL
2	1.0	1	MARSHALL MEDICAL CENTERS SO
3	1.0	1	NORTH ALABAMA MEDICAL CENT
4	1.0	1	MIZELL MEMORIAL HOSPITAL
...	...	...	...
139514	1.0	1	CROCKETT MEDICAL CENTER
139515	1.0	1	EL PASO LTAC HOSPTIAL
139516	1.0	1	BAYLOR SCOTT & WHITE MEDICA
139517	1.0	1	THE HEIGHTS HOSPITAL
139518	1.0	1	SOUTHCROSS HOSPITAL

	PRVDR_CTGRY_SBTYP_CD	PRVDR_CTGRY_CD	FAC_NAME

Attribution: I asked my code why my len(X) and length of index values were not equal, and was advised to set i equal to 0 instead of 1.

Concat the dataframes together

```
concat_1 = pd.concat([short16, short17], axis = 0)

concat_2 = pd.concat([concat_1, short18], axis = 0)

df_health = pd.concat([concat_2, short19], axis = 0)

df_health.reset_index(drop = True)
```

	PRVDR_CTGRY_SBTYP_CD	PRVDR_CTGRY_CD	FAC_NAME
0	1.0	1	SOUTHEAST ALABAMA MEDICAL C
1	1.0	1	NORTH JACKSON HOSPITAL
2	1.0	1	MARSHALL MEDICAL CENTER SOU
3	1.0	1	ELIZA COFFEE MEMORIAL HOSPITA
4	1.0	1	MIZELL MEMORIAL HOSPITAL
...	...	...	...
29080	1.0	1	CROCKETT MEDICAL CENTER
29081	1.0	1	EL PASO LTAC HOSPTIAL
29082	1.0	1	BAYLOR SCOTT & WHITE MEDICAL
29083	1.0	1	THE HEIGHTS HOSPITAL
29084	1.0	1	SOUTHCROSS HOSPITAL

Attribution: https://datacarpentry.org/python-ecology-lesson/05-merging-data.html

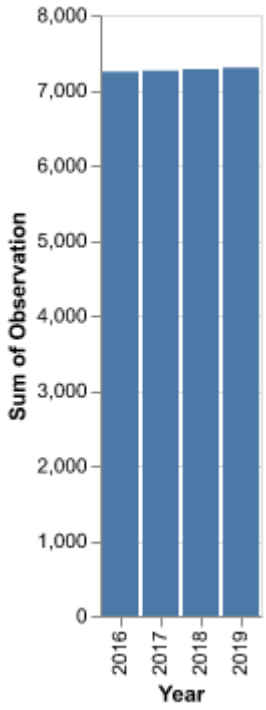
Code

```
def  one_dummy(C):
  observation = []
  i = 0
  #produces a 1 for every entry
  while 1 > 0 and i < C:
    observation.append(1)
    i += 1
  return(observation)
```

```
#add observation column to df
df_health["Observation"] = one_dummy(len(df_health))
```

Attribution: modifying my own code from Pset 1 Graph observations by year

```
alt.data_transformers.enable("vegafusion")

alt.Chart(df_health).mark_bar().encode(
  alt.X("Year:N"),
  alt.Y("sum(Observation)")
)
```



4. a.

```
# create a function to iterate through PRVDR_NUM
def unique_comp(column):
  unique_box = []
  for entry in column:
    if entry not in unique_box:
```

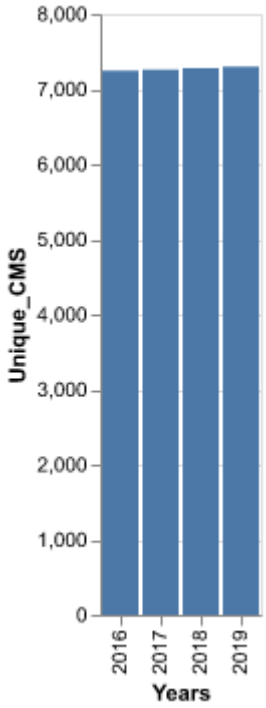
```
      unique_box.append(entry)
    else:
      pass
  return(unique_box)

#save list
unique_list = unique_comp(df_health["PRVDR_NUM"])
```

Filter by year in order to make numbers for each year, save those numbers to a df

```
# make series with the four years
the_years = [2016, 2017, 2018, 2019]

#2016
unique_len_16 = len(unique_comp(df_health["PRVDR_NUM"][df_health["Year"] ==
 ↪  2016]))

#2017
unique_len_17 = len(unique_comp(df_health["PRVDR_NUM"][df_health["Year"] ==
 ↪  2017]))

#2018
unique_len_18 = len(unique_comp(df_health["PRVDR_NUM"][df_health["Year"] ==
 ↪  2018]))

#2019
unique_len_19 = len(unique_comp(df_health["PRVDR_NUM"][df_health["Year"] ==
 ↪  2019]))

# make the lens into a list
len_list = [unique_len_16, unique_len_17, unique_len_18, unique_len_19]

#form into dataframe

the_data = {
  "Years": the_years,
   "Unique_CMS": len_list
}

df_CMS = pd.DataFrame(the_data)
```

Plot the graph

```
alt.Chart(df_CMS).mark_bar().encode(
  alt.X("Years:N"),
  alt.Y("Unique_CMS")
)
```



Attribution: https://www.geeksforgeeks.org/different-ways-to-create-pandas-dataframe/ b. These plots suggest that, disaggregated by year, each observation is equal to a unique hospital. In addition, the number of unique hospitals in 2019 is exactly the same as the number of unique hospitals for the dataset overall, with each year appearing to have all unique observations, indicating that either more hospitals are being opened than the number that are closing down, or, perhaps more likely, that there is an issue with the way we are analyzing this data.

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

1. Create a list of all hospitals that were active in 2016 that were suspected to have closed by 2019. Record the facility name and zip of each hospital as well as the year of suspected closure. How many hospitals fit in this definition?

3,130 hospitals fit this definition, out of over 7000 total hospitals.

```
active_2016 = df_health[(df_health['PGM_TRMNTN_CD'] == 0) &
↪  (df_health['Year'] == 2016)]

def find_suspected_closures(active_2016, df_health):
    suspected_closures = []
    active_facilities = set(active_2016['FAC_NAME'])

    for year in range(2017, 2020):
        closed_facilities = df_health[(df_health['Year'] == year) &
↪  (df_health['PGM_TRMNTN_CD'] != 0)]

        for facility in active_facilities:
            if facility not in closed_facilities['FAC_NAME'].values:
                suspected_closures.append({
                    'FAC_NAME': facility,
                    'ZIP_CD': active_2016.loc[active_2016['FAC_NAME'] ==
                    ↪  facility, 'ZIP_CD'].values[0],
                    'Year': year
                })
    return pd.DataFrame(suspected_closures)

suspected_closures = find_suspected_closures(active_2016, df_health)
print(len(suspected_closures.loc[suspected_closures.groupby('FAC_NAME')['Year'].idxmin()]))
```

```
3130
```

2. Sort this list of hospitals by name and report the names and year of suspected closure for the first 10 rows.

```
print(suspected_closures.loc[suspected_closures.groupby('FAC_NAME')['Year'].idxmin()].sort_va
↪  'Year']].head(10))
```

```
                                          FAC_NAME  Year
441   (CLOSED) BEHAVIORAL HEALTHCARE CENTER AT CLARK  2017
132                      ABBEVILLE GENERAL HOSPITAL  2017
3024                   ABBOTT NORTHWESTERN HOSPITAL  2017
693                 ABILENE REGIONAL MEDICAL CENTER  2017
496                      ABINGTON MEMORIAL HOSPITAL  2017
1809                        ABRAZO ARROWHEAD CAMPUS  2017
2889                          ABRAZO CENTRAL CAMPUS  2017
81                          ABRAZO SCOTTSDALE CAMPUS  2017
```

```
1128                              ABRAZO WEST CAMPUS  2017
2266                          ACADIA GENERAL HOSPITAL  2017
```

3. Not all closures are true closures, remove any suspected closures that are zipcodes where the number does not decrease in the year after the suspected closure.

    a. Among the suspected closures, how many hospitals fit this definition? After correcting, how many do you have left?

    Working with the definition given, we have 3090 mergers/acquisitions.

```
total_closures =
 ↪  suspected_closures.loc[suspected_closures.groupby('FAC_NAME')['Year'].idxmin()].sort_valu
 ↪  'Year']]
zip_counts = suspected_closures.groupby(['ZIP_CD',
 ↪  'Year']).size().reset_index(name='Observation_Count')

zip_counts = zip_counts.sort_values(['ZIP_CD', 'Year'])

zip_counts['Prev_Observation_Count'] =
 ↪  zip_counts.groupby('ZIP_CD')['Observation_Count'].shift(1)
zip_counts['Change'] = zip_counts['Observation_Count'] -
 ↪  zip_counts['Prev_Observation_Count']

#Filter for ZIP codes with no decrease or no disappearance
no_decrease = zip_counts[(zip_counts['Change'] >= 0) |
 ↪  (zip_counts['Prev_Observation_Count'].notnull())]

result = no_decrease[['ZIP_CD', 'Year', 'Observation_Count']]
print(result)

merged_results = pd.merge(suspected_closures, no_decrease, on='ZIP_CD',
 ↪  how='inner')
mergers = merged_results[['FAC_NAME']].drop_duplicates()
print(len(mergers))
```

```
       ZIP_CD  Year  Observation_Count
1       603.0  2018                  1
2       603.0  2019                  1
4       613.0  2018                  1
5       613.0  2019                  1
7       614.0  2018                  1
...       ...   ...                ...
```

```
8372  99669.0  2019                 1
8374  99701.0  2018                 1
8375  99701.0  2019                 1
8377  99801.0  2018                 1
8378  99801.0  2019                 1

[5546 rows x 3 columns]
3090
```

b. How many hospitals do you have left?
Subtracting the mergers, we have 40 hospitals left.

```
print((len(total_closures) - len(mergers)))
```

```
40
```

c. Sort and report the first 10 rows.

```
print(mergers.sort_values(by='FAC_NAME').head(10))
```

```
                                        FAC_NAME
875   (CLOSED) BEHAVIORAL HEALTHCARE CENTER AT CLARK
263                       ABBEVILLE GENERAL HOSPITAL
5934                    ABBOTT NORTHWESTERN HOSPITAL
1361                 ABILENE REGIONAL MEDICAL CENTER
978                       ABINGTON MEMORIAL HOSPITAL
3550                         ABRAZO ARROWHEAD CAMPUS
5668                           ABRAZO CENTRAL CAMPUS
161                          ABRAZO SCOTTSDALE CAMPUS
2214                              ABRAZO WEST CAMPUS
4444                          ACADIA GENERAL HOSPITAL
```

Download Census zip code shapefile (10 pt)

1. a. The five file types in this download are .dbf files, .prj files, .shp files, .shx files, and .xml file. .dbf Files have attribute information, .shp files have feature geometrics, and .shx files contain a positional index, while .prj files describe the Coordinate Reference System and .xml files are used to store data in hierarchical categories.

Attribution: in-class lecture and https://blog.hubspot.com/website/what-is-xml-file

b. In terms of uncompressed sizes, the largest files are the .shp and .dbf files, at 817,915 and 6,275 KB, followed by the .shx and .xml files at 259 and 16 KB. Coming in last is the .prj file at 1 KB.

2. a.
   b.

3.

Read the data into GeoDataFrame

```
#read the .shp file in
the_path =
 ↪  "C:/Users/andre/Documents/GitHub/problem-set-4-andrew-justine/gz_2010_us_860_00_500k.shp"

#the_path =
 ↪  "/Users/justinesilverstein/Desktop/problem-set-4-andrew-justine/gz_2010_us_860_00_500k.sh

shape_data = gpd.read_file(the_path)
```

Attribution: https://automating-gis-processes.github.io/CSC/notebooks/L2/geopandas-basics.html

Restrict shape_data only to Texas zip codes

First create a list of Texas zipcodes

```
#generate list of Texas zipcode leading numbers
Tex_zip_list = np.linspace(start = 750, stop = 799, num =
 ↪  50).astype(np.int32)
#turn to list object
Tex_zip_list = list(Tex_zip_list)

#add Austin entry to Tex_zip_list
Tex_zip_list.append(733)

string_Tex_list = str(Tex_zip_list)
```

This function is used only for checking zipcodes

```
#make a function to pull out the first 3 letters in the
#string for each entry
def string_puller(df):
  #a receiver for the values
  my_strings = []
  #subset for the relevant column
  A = df["ZCTA5"]
  for i in A:
    #select the first 3 characters in each zipcode
    my_strings.append(i[0:3])
  return(my_strings)
```

Create a column for being in Texas

```
#create a binary variable for in or outside of Texas
def in_Texas(df, condition):
  Texas = []
  for i in df[condition]:
    #select the first 3 characters in each zipcode
    #if they have a match in the zip code list
    #they are in Texas
    if i[0:3] in string_Tex_list:
      Texas.append(1)
    else:
      Texas.append(0)
  df["in_Texas"] = Texas
  return(df)

#run function on shape_data
in_Texas(shape_data, "ZCTA5")
```

	GEO_ID	ZCTA5	NAME	LSAD	CENSUSAREA	geometry
0	8600000US01040	01040	01040	ZCTA5	21.281	POLYGON ((-72.62734 42.16203, -7
1	8600000US01050	01050	01050	ZCTA5	38.329	POLYGON ((-72.95393 42.34379, -7
2	8600000US01053	01053	01053	ZCTA5	5.131	POLYGON ((-72.68286 42.37002, -7
3	8600000US01056	01056	01056	ZCTA5	27.205	POLYGON ((-72.39529 42.18476, -7
4	8600000US01057	01057	01057	ZCTA5	44.907	MULTIPOLYGON (((-72.39191 42.
...	...	...	...	...	...	...
33115	8600000US57340	57340	57340	ZCTA5	79.952	POLYGON ((-97.76724 43.90034, -9
33116	8600000US59910	59910	59910	ZCTA5	55.396	MULTIPOLYGON (((-114.19828 47

	GEO_ID	ZCTA5	NAME	LSAD	CENSUSAREA	geometry
33117	8600000US60004	60004	60004	ZCTA5	11.082	POLYGON ((-87.98511 42.14212, -8
33118	8600000US60048	60048	60048	ZCTA5	27.528	MULTIPOLYGON (((-87.95979 42.
33119	8600000US60102	60102	60102	ZCTA5	14.495	POLYGON ((-88.3694 42.15406, -88

Restrict shape_data to in_Texas

```
Tex_shape = shape_data[shape_data["in_Texas"] == 1]
```

Attribution: https://stackoverflow.com/questions/35928170/is-there-a-numpy-function-for-generating-sequences-similar-to-rs-seq-function

Calculate the number of hospitals by zipcode in 2016

Clean up ZIP_CD data type

```
#convert ZIP_CD to float
df_health["ZIP_CD"] = df_health["ZIP_CD"].astype(float)
#convert ZIP_CD to int
df_health["ZIP_CD"] = df_health["ZIP_CD"].apply(np.int64)
#convert ZIP_CD to string
df_health["ZIP_CD"] = df_health["ZIP_CD"].astype(str)
```

Use function from earlier to subset df_health to in_Texas

```
#add in_Texas column to df_health
in_Texas(df_health, "ZIP_CD")

#subset df_health to in_Texas == 1
Tex_health = df_health[df_health["in_Texas"] == 1]
```

Created a grouped object containing

```
def by_zip(df):
  #subset for 2016
  hosp_2016 = df[df["Year"] == 2016]

  #groupby zipcode, select CMS Identification number, and
  #return a count of the CMS ID numbers associated with
  #each Zipcode
```

```
  grouped = hosp_2016.groupby("ZIP_CD")["PRVDR_NUM"].count()

  #reset index
  grouped = grouped.reset_index()

  #save to dataframe
  df_grouped = pd.DataFrame(grouped)
  return(df_grouped)

#run on Tex_health
CMS_by_zip = by_zip(Tex_health)

print(CMS_by_zip)
```

```
    ZIP_CD  PRVDR_NUM
0      733          4
1    75001          1
2    75010          2
3    75013          1
4    75020          3
..     ...        ...
527  79905          1
528  79915          1
529  79925          1
530  79936          1
531  79938          1

[532 rows x 2 columns]
```

Attribution: https://realpython.com/pandas-groupby/ "Example 1".

Add the CMS Number column from the grouped df to the shape data using pd.merge(). Then make choropleth

```
#add an int version of ZIP_CD to CMS_by_zip
CMS_by_zip["Int_Zip"] = [int(i) for i in CMS_by_zip["ZIP_CD"]]

#convert that int version into a string
CMS_by_zip["String_Zip"] = [str(i) for i in CMS_by_zip["Int_Zip"]]

#comprehension where if ZCTA5 in String_Zip add entry into
#new column
```

```
new_col = [i for i in shape_data["ZCTA5"].values if i in
 ↪  CMS_by_zip["String_Zip"].values]

newer_col = [CMS_row["PRVDR_NUM"] for index, CMS_row in CMS_by_zip.iterrows()
 ↪  if CMS_row["String_Zip"] in new_col]



#subsetting for shape_file ZCTA5 == String_Zip in CMS_by_zip

Texas_hospitals = pd.merge(Tex_shape, CMS_by_zip, left_on = "ZCTA5", right_on
 ↪  = "ZIP_CD", how = "left")
```

Attribution: https://www.statology.org/pandas-merge-on-different-column-names/ I also asked ChatGPT how to use the code suggested in the article above.

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

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

The dimensions of the dataframe are 33120 rows and 6 columns. And the columns are printed. For the columns, GEO_ID is the unqiue identifier for a geographic area and serves as the primary key for joining with other datasets. ZCTAS is the Zip Code Tabulation Area, NAME is the name typically associated with the zip. LSAD stands for Legal/Statistical Area Description which indicates the type of area the zip represents (so incorported, unincorporated, etc). CENSUSAREA represents the area size of the zip code region, usually measured in miles or kilometers and provides informations about the geographic extent of the area. The geometry column contains the geometric representation of the zip code area, holding the acutal shape and boundaries of the zip as a geometric object.

```
#zips_all =
 ↪  gpd.read_file('/Users/justinesilverstein/Desktop/problem-set-4-andrew-justine/gz_2010_us

zips_all =
 ↪  gpd.read_file("C:/Users/andre/Documents/GitHub/problem-set-4-andrew-justine")

zips_all_centroids = zips_all.copy()
zips_all_centroids['geometry'] = zips_all_centroids.geometry.centroid
```

```
print(zips_all_centroids.shape)
print(zips_all_centroids.columns)
```

```
C:\Users\andre\AppData\Local\Temp\ipykernel_9792\3016945152.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.

  zips_all_centroids['geometry'] = zips_all_centroids.geometry.centroid

(33120, 6)
Index(['GEO_ID', 'ZCTA5', 'NAME', 'LSAD', 'CENSUSAREA', 'geometry'],
dtype='object')
```

2. Create two GeoDataFrames as subsets of zips_all_centroids. First, create all zip codes in Texas: zips_texas_centroids. Then, create all zip codes in Texas or a bordering state: zips_texas_borderstates_centroids, using the zip code prefixes to make these subsets. How many unique zip codes are in each of these subsets?

There are 402 unique zip codes in Texas and 1314 unique zip codes in the bordering states.

```
zips_all_centroids =
 ↪  gpd.read_file("C:/Users/andre/Documents/GitHub/problem-set-4-andrew-justine/gz_2010_us_8
zips_texas_centroids =
 ↪  zips_all_centroids[zips_all_centroids['ZCTA5'].str.startswith('75')]

unique_texas_zip_codes = zips_texas_centroids['ZCTA5'].nunique()
print(f"Unique ZIP codes in Texas: {unique_texas_zip_codes}")

bordering_states_prefixes = ['87', '73', '72', '70']
zips_texas_borderstates_centroids = zips_all_centroids[

 ↪  zips_all_centroids['ZCTA5'].str.startswith(tuple(bordering_states_prefixes))
]

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

```
Unique ZIP codes in Texas: 402
Unique ZIP codes in Texas and bordering states: 1314
```

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

I first changed the column names to match and then inner merged on the variable ZIP_CD. There are 148 Texas zip codes with at least one hospital.

```
active_2016 = df_health[(df_health['PGM_TRMNTN_CD'] == 0) &
 ↪  (df_health['Year'] == 2016)]
zips_texas_borderstates_centroids =
 ↪  zips_texas_borderstates_centroids.rename(columns={'ZCTA5': 'ZIP_CD'})

hospital_zip_df = active_2016.rename(columns={'ZIP_CD': 'ZIP_CD'})
hospital_zip_df['ZIP_CD'] = hospital_zip_df['ZIP_CD'].astype(str)
zips_texas_borderstates_centroids['ZIP_CD'] =
 ↪  zips_texas_borderstates_centroids['ZIP_CD'].astype(str)

zips_withhospital_centroids = zips_texas_borderstates_centroids.merge(
    hospital_zip_df, on='ZIP_CD', how='inner'
)

print(f'Number of ZIP codes with at least one hospital in 2016:
 ↪  {len(zips_withhospital_centroids)}')
```

```
Number of ZIP codes with at least one hospital in 2016: 148
```

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

   a.This is a computationally-intensive join. Before attempting to do the entire join, subset to 10 zip codes in zips_texas_centroids and try the join. How long did it take? Approximately how long do you estimate the entire procedure will take?

   The code for this question is above. It took 9.98 seconds to compute for 10 zip codes. I think it will take roughly 150 seconds for the entire procedure to complete.

```
import time

def calculate_nearest_hospital_distance(zips, hospitals):
    zips = zips.to_crs(hospitals.crs)
    distances = zips.geometry.apply(
        lambda zip_geom: hospitals.distance(zip_geom).min()
    )
```

```
    return distances

subset_texas_zips = zips_texas_centroids.sample(n=10, random_state=1)
start_time = time.time()
subset_texas_zips['Nearest_Hospital_Distance'] =
 ↪  calculate_nearest_hospital_distance(subset_texas_zips,
 ↪  zips_withhospital_centroids)
end_time = time.time()

print(subset_texas_zips[['ZCTA5', 'Nearest_Hospital_Distance']])
print(f'Time taken for subset calculation: {end_time - start_time:.4f}
 ↪  seconds')
```

```
C:\Users\andre\AppData\Local\Temp\ipykernel_9792\745236431.py:6: UserWarning:
Geometry is in a geographic CRS. Results from 'distance' are likely
incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected
CRS before this operation.

  lambda zip_geom: hospitals.distance(zip_geom).min()

       ZCTA5  Nearest_Hospital_Distance
10784  75450                   1.500020
26800  75044                   1.100082
32915  75424                   0.879122
10739  75158                   1.598125
10564  75024                   0.952822
24874  75701                   2.202935
32579  75662                   1.833904
26486  75253                   1.390721
9376   75703                   2.115168
9409   75951                   0.347204
Time taken for subset calculation: 8.9711 seconds
```

b. Now try doing the full calculation and time how long it takes. How close is it to your estimation?

The code for this question is above. It took 437.47 seconds to do the whole calculation, which was very far from my estimate.

```
import time

def calculate_nearest_hospital_distance(zips, hospitals):
```

```
    zips = zips.to_crs(hospitals.crs)
    distances = zips.geometry.apply(
        lambda zip_geom: hospitals.distance(zip_geom).min()
    )
    return distances

start_time = time.time()
zips_texas_centroids['Nearest_Hospital_Distance'] =
 ↪  calculate_nearest_hospital_distance(zips_texas_centroids,
 ↪  zips_withhospital_centroids)
end_time = time.time()

print(zips_texas_centroids[['ZCTA5', 'Nearest_Hospital_Distance']].head())
print(f'Time taken for full calculation: {end_time - start_time:.4f}
 ↪  seconds')
```

```
C:\Users\andre\AppData\Local\Temp\ipykernel_9792\2563871115.py:6:
UserWarning: Geometry is in a geographic CRS. Results from 'distance' are
likely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a
projected CRS before this operation.

  lambda zip_geom: hospitals.distance(zip_geom).min()
```

```
      ZCTA5  Nearest_Hospital_Distance
9347  75558                  1.859191
9348  75559                  1.382945
9349  75560                  1.593366
9350  75561                  1.357277
9351  75562                  1.765584
Time taken for full calculation: 756.1693 seconds
```

```
C:\Users\andre\AppData\Local\Programs\Python\Python311\Lib\site-packages\geopandas\geodatafra
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
  super().__setitem__(key, value)
```

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

a. What unit is this in? Since we are using geometry distance, it is originally in units or degrees which are not useful for our purposes to gauge distance.

Convert the nearest hospital distances from meters to miles

zips_texas_centroids['Nearest_Hospital_Distance_Miles'] = zips_texas_centroids['Nearest_Hospital_Distanc

average_distance_miles = zips_texas_centroids['Nearest_Hospital_Distance_Miles'].mean()

print(f"Average distance to the nearest hospital for each ZIP code in Texas: {average_distance_miles:.2f} miles")

b. Report the average distance in miles. Does this value make sense?

This value does make sense as all distance in the US is measured in miles, units is not useful to gauge real life distance. After converting to miles, the average is 1.4 miles to the nearest hospital.

```{python}
# Convert the nearest hospital distances from meters to miles
zips_texas_centroids['Nearest_Hospital_Distance_Miles'] =
zips_texas_centroids['Nearest_Hospital_Distance']

average_distance_miles =
zips_texas_centroids['Nearest_Hospital_Distance_Miles'].mean()

print(f"Average distance to the nearest hospital for each ZIP code in Texas:
{average_distance_miles:.2f} miles")

c. Map the value for each zip code.

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 1, figsize=(12, 10))
zips_texas_centroids.plot(column='Nearest_Hospital_Distance_Miles',
                          ax=ax,
                          legend=True,
                          cmap='OrRd',
                          edgecolor='black',
```
```

```
                           legend_kwds={'label': "Distance to Nearest Hospital
 ↪  (Miles)",
                                        'orientation': "horizontal"})

ax.set_title('Distance to Nearest Hospital for Texas ZIP Codes', fontsize=15)
ax.set_xlabel('Longitude', fontsize=12)
ax.set_ylabel('Latitude', fontsize=12)

plt.show()
```

Effects of closures on access in Texas (15 pts)

1. Create df with only "cleaned" closures

```
#merge to bring in zipcodes, but only those caught in total_closures
new_total_closures = pd.merge(total_closures, suspected_closures, on =
 ↪  "FAC_NAME", how = "inner")

#drop duplicates
new_total_closures = new_total_closures.drop_duplicates(subset = "FAC_NAME")

#drop "Year_y"
new_total_closures = new_total_closures.drop(columns = "Year_y")

#change Year_x to Year
new_total_closures = new_total_closures.rename(columns = {"Year_x":"Year"})
```

Use groupby to create list of closures

```
#groupby zipcode
TX_closed = new_total_closures.groupby("ZIP_CD")["FAC_NAME"].count()

#reset index
TX_closed = TX_closed.reset_index()

#to DataFrame
TX_closed = pd.DataFrame(TX_closed)

#rename FAC_NAME to Closure_Count
TX_closed = TX_closed.rename(columns = {"FAC_NAME":"Closure_Count"})
```

Convert ZIP_CD to string

```
#convert ZIP_CD to float
TX_closed["ZIP_CD"] = TX_closed["ZIP_CD"].astype(float)
#convert ZIP_CD to int
TX_closed["ZIP_CD"] = TX_closed["ZIP_CD"].apply(np.int64)
#convert ZIP_CD to string
TX_closed["ZIP_CD"] = TX_closed["ZIP_CD"].astype(str)

#add in_Texas column
in_Texas(TX_closed, "ZIP_CD")
#filter for only Texas
TX_closed = TX_closed[TX_closed["in_Texas"] == 1]

print("This table shows hospital closures in TX by zipcode")

print(TX_closed)
```

2. Perform a merge to create shapefile containing the closure_count data

```
shape_closure = pd.merge(Tex_shape, TX_closed, left_on = "ZCTA5", right_on =
 ↪  "ZIP_CD", how = "left")

#fill in NA values in Closure_Count with 0
shape_closure["Closure_Count"] = shape_closure["Closure_Count"].fillna(0)

#drop ZIP_CD (leaving only ZCTA5) and drop in_Texas_y

shape_closure = shape_closure.drop(columns = "ZIP_CD")

shape_closure = shape_closure.drop(columns = "in_Texas_y")

#rename in_Texas_x to in_Texas

shape_closure = shape_closure.rename(columns = {"in_Texas_x":"in_Texas"})
```

Plot choropleth

```
shape_closure.plot(column = "Closure_Count", legend = True)
```

```
#also print number of directly affected zipcodes
len(shape_closure[shape_closure["Closure_Count"] >= 1])
```

It appears that there are 231 zipcodes in Texas that were directly impacted by a closure, based off of the shape file data we created. However, this is about 20 zipcodes lower than the number implied in the list of TX closures by zipcode. There are a number of potential reasons for this, including potential error in the original data (e.g. zipcodes that are not included in the .shp file) or, perhaps more likely, a consequence of data cleaning choices.

Attribution: I asked ChatGPT why I might have lost some values in using pd.merge() 3.

4.

Reflecting on the exercise (10 pts)

1. Based on my understanding of the methodology, I can see a few different ways in which our first pass may be imperfect. Firstly, it is possible that, due to some of the complexities involved in hospital ownership, some hospitals that we may mark as open may have changed their terms of service, and no longer fall under the category of short term hospital, but not have reported this at the time that they reported their status to the relevant authorities. Additionally, some of the hospital names (FAC_NAME) actually have the word closed in them, for multiple years, and it would probably be worthwhile to select for those hospital names to understand how many there are, and potentially create a comparison between the hospitals that are reported as closed in the dataset and the list of hospitals that we identified as closed in the first pass.

We could also cross-reference data from other sources to develop multiple indicators determining whether a hospital is open or closed, and then create a measure with multiple levels of confidence (e.g. if the hospital is listed as closed by local property taxation records and in this dataset, it will be at level 2, and so on. The objective would be to see if the hospital matches status across data sets.)

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

I think there are issues with how we code and define true closures from mergers/acquisitions. The definition itself is confusng as we compare different observations from a year to year basis but a hospital could still close even if it was a merger. Furthermore, it is unclear how we discuss what we count as a merger, especially if it gets confusing how to calculate when a merger occurred. Also, this does not take into account urban areas vs rural area where

distance makes a huge difference for people. I do not think mergers and acquisitions reflect changes in zip code level hospital access siginifcantly. I think it the type of facility closure is more likely to have an impact.