PSET-4

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): Kishika Mahajan; kishika
   - Partner 2 (name and cnet ID): Nidhi Srivastava; nsrivastava1
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: KM , NS
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

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

1. The variables I pulled are PRVDR_CTGRY_SBTYP_CD, PRVDR_CTGRY_CD, PRVDR_NUM, PGM_TRMNTN_CD, FAC_NAME and ZIP_CD
2. Importing the dataset

```
import pandas as pd
import altair as alt
```

```
hospitals_2016 = pd.read_csv("/Users/kishikamahajan/Desktop/pos2016.csv")
#hospitals_2016 = pd.read_csv("/Users/nidhi/Desktop/Data and Programming
↪  Python 2/POS_File_Hospital_Non_Hospital_Facilities_Q4_2016.csv")
hospitals_2016.head()
hospitals_2016.head()
```

```python
# Subsetting the hospitals
short_term_hospitals_2016 = hospitals_2016[(hospitals_2016["PRVDR_CTGRY_CD"]
 ↪ == 1) & (hospitals_2016["PRVDR_CTGRY_SBTYP_CD"] == 1)]

# Adding a column for the year
short_term_hospitals_2016["YEAR"] = 2016

short_term_hospitals_2016.shape[0]
```

```
/var/folders/w0/cccpmsxn11z4l65wxv069v7m0000gn/T/ipykernel_6725/1797289996.py:10:
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

7245
```

```
a. The number of hospitals reported in this data are 7245. This number does
seem to be pretty high
b. The American Hospital Association (AHA) reports around 6000 total
hospitals in the U.S., with only about 5000 being community hospitals. The
difference can be because there can be potential duplicates. The datatset
might also include hospitals which have been closed.
```

3. Importing the datasets

```python
# FOR 2017
hospitals_2017 = pd.read_csv("/Users/kishikamahajan/Desktop/pos2017.csv")
#hospitals_2017 = pd.read_csv("/Users/nidhi/Desktop/Data and Programming
 ↪ Python 2/POS_File_Hospital_Non_Hospital_Facilities_Q4_2017.csv")

hospitals_2017.head()

# Subsetting the hospitals
short_term_hospitals_2017 = hospitals_2017[(hospitals_2017["PRVDR_CTGRY_CD"]
 ↪ == 1) & (hospitals_2017["PRVDR_CTGRY_SBTYP_CD"] == 1)]
```

```
# Adding a column for the year
short_term_hospitals_2017["YEAR"] = 2017

print(f"The short term hospitals in 2017 were
 ↪  {short_term_hospitals_2017.shape[0]}")
```

```
The short term hospitals in 2017 were 7260
```

```
/var/folders/w0/cccpmsxn11z4l65wxv069v7m0000gn/T/ipykernel_6725/3417132224.py:11:
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
```

The same logic about the number of hospitals applies from above.

```
# FOR 2018
hospitals_2018 = pd.read_csv("/Users/kishikamahajan/Desktop/pos2018.csv" ,
 ↪  encoding="ISO-8859-1")
#hospitals_2018 = pd.read_csv("/Users/nidhi/Desktop/Data and Programming
 ↪  Python 2/POS_File_Hospital_Non_Hospital_Facilities_Q4_2018.csv" ,
 ↪  encoding="ISO-8859-1")
hospitals_2018.head()

# Subsetting the hospitals
short_term_hospitals_2018 = hospitals_2018[(hospitals_2018["PRVDR_CTGRY_CD"]
 ↪  == 1) & (hospitals_2018["PRVDR_CTGRY_SBTYP_CD"] == 1)]

# Adding a column for the year
short_term_hospitals_2018["YEAR"] = 2018

print(f"The short term hospitals in 2018 were
 ↪  {short_term_hospitals_2018.shape[0]}")
```

```
The short term hospitals in 2018 were 7277
```

```
/var/folders/w0/cccpmsxn11z4l65wxv069v7m0000gn/T/ipykernel_6725/3261341874.py:10:
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
```

The same logic about the number of hospitals applies from above.

```
# FOR 2019
hospitals_2019 = pd.read_csv("/Users/kishikamahajan/Desktop/pos2019.csv" ,
↪ encoding="ISO-8859-1")
#hospitals_2019 = pd.read_csv("/Users/nidhi/Desktop/Data and Programming
↪ Python 2/POS_File_Hospital_Non_Hospital_Facilities_Q4_2019.csv" ,
↪ encoding="ISO-8859-1")
hospitals_2019.head()

# Subsetting the hospitals
short_term_hospitals_2019 = hospitals_2019[(hospitals_2019["PRVDR_CTGRY_CD"]
↪ == 1) & (hospitals_2019["PRVDR_CTGRY_SBTYP_CD"] == 1)]

# Adding a column for the year
short_term_hospitals_2019["YEAR"] = 2019

print(f"The short term hospitals in 2019 were
↪ {short_term_hospitals_2019.shape[0]}")
```

```
The short term hospitals in 2019 were 7303
```

```
/var/folders/w0/cccpmsxn11z4l65wxv069v7m0000gn/T/ipykernel_6725/1856361383.py:10:
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
```

The same logic about the number of hospitals applies from above.

Appending the datasets together

```python
combined_short_term_hospitals = pd.concat([short_term_hospitals_2016,
    short_term_hospitals_2017, short_term_hospitals_2018,
    short_term_hospitals_2019], ignore_index=True)
combined_short_term_hospitals.head()
```

	PRVDR_CTGRY_SBTYP_CD	PRVDR_CTGRY_CD	FAC_NAME
0	1.0	1	SOUTHEAST ALABAMA MEDICAL CENT
1	1.0	1	NORTH JACKSON HOSPITAL
2	1.0	1	MARSHALL MEDICAL CENTER SOUTH
3	1.0	1	ELIZA COFFEE MEMORIAL HOSPITAL
4	1.0	1	MIZELL MEMORIAL HOSPITAL

Plotting the number of hospitals by year

```python
# grouping by year
hospitals_by_year =
    combined_short_term_hospitals.groupby("YEAR").size().reset_index(name =
    "number_of_hospitals")

# plotting the number of observations by year
alt.Chart(hospitals_by_year).mark_bar().encode(
    alt.X("YEAR:N" , title = "Year"),
    alt.Y("number_of_hospitals" , title = "Number of Hospitals")
)
```

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

4. Plotting the unique number of hospitals by year

   a.

```python
unique_hospitals_by_year = (

    combined_short_term_hospitals.groupby("YEAR")["PRVDR_NUM"].nunique().reset_index(name
    = "unique_hospitals")
)

# plotting the unique number of hospitals
alt.Chart(unique_hospitals_by_year).mark_bar().encode(
    alt.X("YEAR:N" , title = "Year"),
```

```
    alt.Y("unique_hospitals" , title = "Number of Unique Hospitals")
)
```

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

```
b. As can be seen, the two plots are extremely similar.
There is very little or no duplication of hospitals within each year. Hence,
each hospital (identified by its CMS certification number) appears only once
per year in the dataset.
```

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

1. There are 174 hospitals that fit the definition.

```
#Identifying unique column
unique_columns = [col for col in short_term_hospitals_2016.columns if
 ↪  short_term_hospitals_2016[col].is_unique]
print("Columns with all unique values:", unique_columns)
```

```
Columns with all unique values: ['PRVDR_NUM']
```

```
#Identifying unique column
unique_columns = [col for col in short_term_hospitals_2016.columns if
 ↪  short_term_hospitals_2016[col].is_unique]
print("Columns with all unique values:", unique_columns)
```

```
Columns with all unique values: ['PRVDR_NUM']
```

```
termination_active_2016 =
 ↪  short_term_hospitals_2016[short_term_hospitals_2016['PGM_TRMNTN_CD']==0]

termination_active_2016_s = termination_active_2016[['PGM_TRMNTN_CD',
 ↪  'FAC_NAME', 'ZIP_CD','YEAR','PRVDR_NUM']]

def terminated_hospitals(years, short_term_hospital_mapping):
    # Empty list to store results
    terminated_hospitals_list = []

    for year in years:
        short_term_hospital_year = short_term_hospital_mapping[year]
```

```
        # Creating a subset of data by selecting columns
        short_term_hospital_year_s =
 ↪  short_term_hospital_year[['PGM_TRMNTN_CD', 'FAC_NAME',
 ↪  'ZIP_CD','YEAR','PRVDR_NUM']]

        # Merging the dataset with 2016 Active hospitals
        merged_data_year = pd.merge(termination_active_2016_s,
 ↪  short_term_hospital_year_s, on='PRVDR_NUM', how='outer', indicator=True)

        # Filtering terminated hospitals
        terminated_hospitals_year =
 ↪  merged_data_year[(merged_data_year['_merge'] == 'left_only') |
 ↪  ((merged_data_year['PGM_TRMNTN_CD_y'] != 0) & (merged_data_year['_merge']
 ↪  == 'both' ))]

        # Store the results in the list
        terminated_hospitals_list.append({
            'year': year,
            'terminated_hospitals': terminated_hospitals_year
        })

    return terminated_hospitals_list

# Define the years and the mapping of short-term hospitals
years = [2017, 2018, 2019]

short_term_hospitals_mapping = {
    2017: short_term_hospitals_2017,
    2018: short_term_hospitals_2018,
    2019: short_term_hospitals_2019,
}

# Call the function
terminated_hospitals_result = terminated_hospitals(years,
 ↪  short_term_hospitals_mapping)

# Print the results
for entry in terminated_hospitals_result:
   print(f"{entry['year']}:", len(entry['terminated_hospitals']))
```

```
2017: 40
```

```
2018: 98
2019: 174
```

```
all_terminated_hospitals = pd.concat([entry['terminated_hospitals'] for entry
↪ in terminated_hospitals_result], ignore_index=True)
all_terminated_hospitals.head()
```

	PGM_TRMNTN_CD_x	FAC_NAME_x	ZIP_CD_x	YI
0	0.0	ABRAZO MARYVALE CAMPUS	85031.0	20
1	0.0	ADVENTIST MEDICAL CENTER - CENTRAL VALLEY	93230.0	20
2	0.0	FALLBROOK HOSPITAL DISTRICT	92028.0	20
3	0.0	ARKANSAS VALLEY REGIONAL MEDICAL CENTER	81050.0	20
4	0.0	KEEFE MEMORIAL HOSPITAL	80810.0	20

```
#Group by FAC to check independent hospital closures

test = all_terminated_hospitals.groupby('PRVDR_NUM').agg(
    count =  ('PRVDR_NUM','count'),
    ZIP_CD = ('ZIP_CD_x','first'),
    YEAR = ('YEAR_y','first')
)
test.head()
print("Total number of terminated hospitals is",len(test))
```

```
Total number of terminated hospitals is 174
```

2.

```
all_terminated_hospitals_1 =
↪ all_terminated_hospitals.sort_values(by='FAC_NAME_x', ascending=True)
all_terminated_hospitals_2 =
↪ all_terminated_hospitals_1[['FAC_NAME_x','YEAR_y','ZIP_CD_x']]
first_10_rows = all_terminated_hospitals_2.head(10)
print("First 10 hospital name with year of suspected closure
↪ is:",first_10_rows)
```

```
First 10 hospital name with year of suspected closure is:
FAC_NAME_x  YEAR_y  ZIP_CD_x
0                           ABRAZO MARYVALE CAMPUS    2017   85031.0
142                         ABRAZO MARYVALE CAMPUS    2019   85031.0
```

```
42                             ABRAZO MARYVALE CAMPUS    2018   85031.0
148       ADVENTIST MEDICAL CENTER - CENTRAL VALLEY    2019   93230.0
1         ADVENTIST MEDICAL CENTER - CENTRAL VALLEY    2017   93230.0
46        ADVENTIST MEDICAL CENTER - CENTRAL VALLEY    2018   93230.0
235                         AFFINITY MEDICAL CENTER    2019   44646.0
90                          AFFINITY MEDICAL CENTER    2018   44646.0
81   ALBANY MEDICAL CENTER / SOUTH CLINICAL CAMPUS    2018   12208.0
218  ALBANY MEDICAL CENTER / SOUTH CLINICAL CAMPUS    2019   12208.0
```

3.

```python
def correct_for_mergers(closure_df, master_df):
    """
    Remove suspected closures that might be mergers/acquisitions
    """
    # Calculate active hospitals per zip code per year from master data
    zip_counts = {}
    for year in master_df['YEAR'].unique():
        active_hospitals = master_df[
            (master_df['YEAR'] == year) &
            (master_df['PGM_TRMNTN_CD'] == 0)
        ]
        zip_counts[year] = active_hospitals.groupby('ZIP_CD').size()

    # Filter out potential mergers
    valid_closures = []

    for _, closure in closure_df.iterrows():
        zip_code = closure['ZIP_CD']
        closure_year = closure['YEAR']

        # Get count of hospitals in zip code before and after closure
        try:
            pre_count = zip_counts[closure_year - 1].get(zip_code, 0)
            post_count = zip_counts[closure_year].get(zip_code, 0)

            # Only keep if number of hospitals decreased
            if post_count < pre_count:
                valid_closures.append(closure)
        except KeyError:
            # If we don't have data for the year, skip this closure
            print(f"Warning: Missing data for ZIP {zip_code} in year
             ↪  {closure_year}")
```

```python
            continue

    valid_closures_df = pd.DataFrame(valid_closures)

    # Add summary statistics
    print(f"\nClosure Analysis Summary:")
    print(f"Total suspected closures: {len(closure_df)}")
    print(f"Confirmed closures: {len(valid_closures_df)}")
    print(f"Potential mergers/acquisitions: {len(closure_df) -
     ↪  len(valid_closures_df)}")

    return valid_closures_df

valid_closures = correct_for_mergers(test,combined_short_term_hospitals)
valid_closures.head()
```

```
Closure Analysis Summary:
Total suspected closures: 174
Confirmed closures: 166
Potential mergers/acquisitions: 8
```

	count	ZIP_CD	YEAR
010032	1.0	36278.0	2019.0
010047	1.0	36033.0	2019.0
010146	2.0	36265.0	2018.0
010172	2.0	35611.0	2018.0
030001	3.0	85031.0	2017.0

Download Census zip code shapefile (10 pt)

1. a. The five file types are .dbf, .prj, .shp, .shx and .xml.

   .shp (Shape Format):

   ::: {.cell execution_count=16} `{.python .cell-code} # reading .shp import geopandas as gpd shapefile = gpd.read_file("/Users/kishikamahajan/Desktop/gz_2010_us_86 shapefile.head()`

   ::: {.cell-output .cell-output-display execution_count=211}

	GEO_ID	ZCTA5	NAME	LSAD	CENSUSAREA	geometry
0	8600000US01040	01040	01040	ZCTA5	21.281	POLYGON ((-72.62734 42.16203, -72.62
1	8600000US01050	01050	01050	ZCTA5	38.329	POLYGON ((-72.95393 42.34379, -72.95
2	8600000US01053	01053	01053	ZCTA5	5.131	POLYGON ((-72.68286 42.37002, -72.68
3	8600000US01056	01056	01056	ZCTA5	27.205	POLYGON ((-72.39529 42.18476, -72.39
4	8600000US01057	01057	01057	ZCTA5	44.907	MULTIPOLYGON (((-72.39191 42.0806

::: :::

This file contains the geometric data of the features. In particular, it contains information like the geo_id, name, census area and the coordinates of the polygon.

.shx (Shape Index Format):

::: {.cell execution_count=17} `{.python .cell-code} # reading .shx shapeindex = gpd.read_file("/Users/kishikamahajan/Desktop/gz_2010_us_860_00_500k/gz_2010_us_860_00_ shapeindex.head()`

::: {.cell-output .cell-output-display execution_count=212}

	GEO_ID	ZCTA5	NAME	LSAD	CENSUSAREA	geometry
0	8600000US01040	01040	01040	ZCTA5	21.281	POLYGON ((-72.62734 42.16203, -72.62
1	8600000US01050	01050	01050	ZCTA5	38.329	POLYGON ((-72.95393 42.34379, -72.95
2	8600000US01053	01053	01053	ZCTA5	5.131	POLYGON ((-72.68286 42.37002, -72.68
3	8600000US01056	01056	01056	ZCTA5	27.205	POLYGON ((-72.39529 42.18476, -72.39
4	8600000US01057	01057	01057	ZCTA5	44.907	MULTIPOLYGON (((-72.39191 42.0806

::: :::

This file contains similar information as the shapefile but more generally, it provides a positional index of the geometry. It helps locate specific geometries quickly within the .shp file.

.dbf (Attribute Format):

::: {.cell execution_count=18} `{.python .cell-code} # reading .dbf attributefile = gpd.read_file("/Users/kishikamahajan/Desktop/gz_2010_us_860_00_500k/gz_2010_us_860_00_ attributefile.head()`

::: {.cell-output .cell-output-display execution_count=213}

	GEO_ID	ZCTA5	NAME	LSAD	CENSUSAREA	geometry
0	8600000US01040	01040	01040	ZCTA5	21.281	POLYGON ((-72.62734 42.16203, -72.62

	GEO_ID	ZCTA5	NAME	LSAD	CENSUSAREA	geometry
1	8600000US01050	01050	01050	ZCTA5	38.329	POLYGON ((-72.95393 42.34379, -72.95
2	8600000US01053	01053	01053	ZCTA5	5.131	POLYGON ((-72.68286 42.37002, -72.68
3	8600000US01056	01056	01056	ZCTA5	27.205	POLYGON ((-72.39529 42.18476, -72.39
4	8600000US01057	01057	01057	ZCTA5	44.907	MULTIPOLYGON (((-72.39191 42.0806

::: :::

While the information is the same, this file contains attribute data for each feature, stored in tabular format. It includes columns with each attribute associated with the data.

.prj (Projection Format):

::: {.cell execution_count=19} "' {.python .cell-code} # reading .prj from pyproj import CRS

with open("/Users/kishikamahajan/Desktop/gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.prj "r") as prj_file: prj_text = prj_file.read() crs = CRS.from_wkt(prj_text)

print(f"The CRS of the file is {crs}") "'

::: {.cell-output .cell-output-stdout} `The CRS of the file is GEOGCS["GCS_North_American_1983",DA`
::: :::

The projection file defines the coordinate system and projection information for the shapefile which is essential for accurately mapping spatial data. It ensures that spatial data aligns correctly with other geographic layers.

.xml (Metadata format): The .xml file in a shapefile set contains metadata about the dataset, offering detailed information about the data's contents, source, creation, and structure.

b. **.shp:**

::: {.cell execution_count=20} "' {.python .cell-code} import os

file_path_shp = "/Users/kishikamahajan/Desktop/gz_2010_us_860_00_500k/gz_2010_us_860_00_5 file_size_shp = os.path.getsize(file_path_shp) file_size_shp_mb = file_size_shp / (1024 * 1024)

print(f"File size of shapefile: {file_size_shp_mb:.2f} MB") "'

::: {.cell-output .cell-output-stdout} `File size of shapefile: 798.74 MB` ::: :::

.shx:

::: {.cell execution_count=21} "' {.python .cell-code} file_path_shx = "/Users/kishikamahajan/Desktop/
file_size_shx = os.path.getsize(file_path_shx) file_size_shx_mb = file_size_shx / (1024 * 1024)

print(f"File size of shx: {file_size_shx_mb:.2f} MB") "'

::: {.cell-output .cell-output-stdout} `File size of shx: 0.25 MB` ::: :::

.dbf:

::: {.cell execution_count=22} "' {.python .cell-code} file_path_dbf = "/Users/kishikamahajan/Desktop/
file_size_dbf = os.path.getsize(file_path_dbf) file_size_dbf_mb = file_size_dbf / (1024 * 1024)

print(f"File size of dbf: {file_size_dbf_mb:.2f} MB") "'

::: {.cell-output .cell-output-stdout} `File size of dbf: 6.13 MB` ::: :::

.prj:

::: {.cell execution_count=23} "' {.python .cell-code} file_path_prj = "/Users/kishikamahajan/Desktop/
file_size_prj = os.path.getsize(file_path_prj) file_size_prj_kb = file_size_prj / 1024

print(f"File size of prj: {file_size_prj_kb:.2f} KB") "'

::: {.cell-output .cell-output-stdout} `File size of prj: 0.16 KB` ::: :::

.xml: This file is 16 KB.

::: {.cell execution_count=24} "' {.python .cell-code} file_path_xml = "/Users/kishikamahajan/Desktop/
file_size_xml = os.path.getsize(file_path_xml) file_size_xml_kb = file_size_xml / 1024

print(f"File size of xml: {file_size_xml_kb:.2f} KB") "'

::: {.cell-output .cell-output-stdout} `File size of xml: 15.27 KB` ::: :::

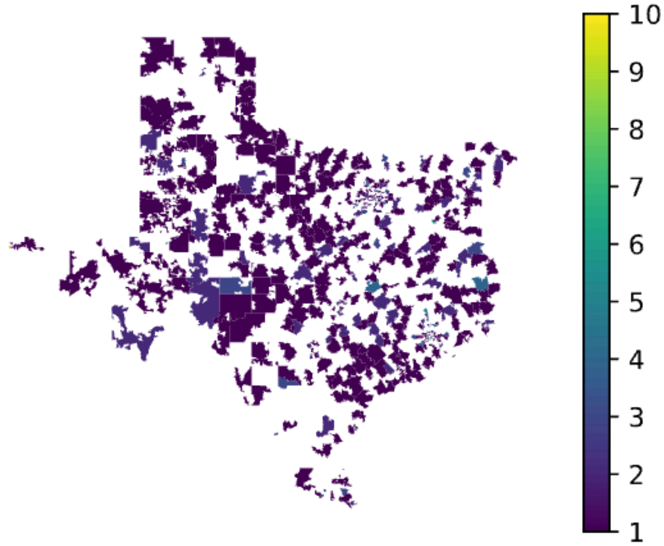
2. Restricting zipcodes only to Texas by including only those zipcodes that start with "75", "76", "77", "78", "79"

```
# keeping only those observations in shapefiles which start with the above

texas_zip_codes = shapefile[shapefile["ZCTA5"].str.startswith(("75", "76",
    "77", "78", "79", "718", "885", "733"))]
texas_zip_codes = texas_zip_codes.rename(columns={"ZCTA5": "ZIP_CD"})
texas_zip_codes.head()
```

	GEO_ID	ZIP_CD	NAME	LSAD	CENSUSAREA	geometry
9127	8600000US71801	71801	71801	ZCTA5	301.823	POLYGON ((-93.69731 33.69854, -
9128	8600000US71822	71822	71822	ZCTA5	254.088	POLYGON ((-94.3768 33.64818, -9
9129	8600000US71825	71825	71825	ZCTA5	24.475	MULTIPOLYGON (((-93.51207 33.
9130	8600000US71832	71832	71832	ZCTA5	176.594	POLYGON ((-94.20633 34.10465, -
9131	8600000US71834	71834	71834	ZCTA5	148.667	POLYGON ((-94.04296 33.01922, -

```
# Convert ZIP_CD to string and remove decimals
short_term_hospitals_2016["ZIP_CD"] =
 ↪  short_term_hospitals_2016["ZIP_CD"].astype(str).str.split('.').str[0]

# Clean up the Texas shapefile ZIP codes
texas_zip_codes["ZIP_CD"] = texas_zip_codes["ZIP_CD"].astype(str)

# grouping short term hospitals on the basis of zipcodes
short_term_hospitals_2016_grouped =
 ↪  short_term_hospitals_2016.groupby("ZIP_CD").size().reset_index(name =
 ↪  "count")

# merging texas zipcodes with short term hospitals
texas_hospitals_geo =
 ↪  texas_zip_codes.merge(short_term_hospitals_2016_grouped, on = "ZIP_CD",
 ↪  how = "left")

texas_hospitals_geo.plot(column = "count", legend = True).set_axis_off()
```

```
/var/folders/w0/cccpmsxn11z4l65wxv069v7m0000gn/T/ipykernel_6725/953405146.py:2:
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
```



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

1. Calculating the zipcode centroids

```
zips_all_centroids = shapefile.copy()
zips_all_centroids["geometry"] = zips_all_centroids.geometry.centroid
zips_all_centroids = zips_all_centroids.reset_index(drop=True)
```

```
/var/folders/w0/cccpmsxn11z4l65wxv069v7m0000gn/T/ipykernel_6725/3656193247.py:2:
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.
```

```
# Viewing the dimensions
dimensions = zips_all_centroids.shape
columns = zips_all_centroids.columns

print("Dimensions of the GeoDataFrame:", dimensions)
print("Columns in the GeoDataFrame:", columns)
```

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

Meaning of columns:

GEO_ID: This is a unique identifier for each ZIP code area in the dataset. **ZCTA5:** This represents the zipcode. **NAME:** This is a reptition of the previous column and essentially shows the zipcode again. **LSAD:** This stands for Legal/Statistical Area Description and describes the type of area that the ZIP code represents. In this case, it is labeled as ZCTA5, indicating that it is a standard ZIP Code Tabulation Area. **CENSUSAREA:** This column indicates the area size of the ZIP code regions. **geometry:** This contains the geometric representation of the ZIP code's centroid.

2. Making a subset of zipcodes in Texas

```
zips_texas_centroids =
↪ zips_all_centroids[zips_all_centroids["ZCTA5"].str.startswith(("75",
↪ "76", "77", "78", "79", "718", "885", "733"))]

unique_texas_zipcodes_count = zips_texas_centroids["ZCTA5"].nunique()
print(f"The number of unique zipcodes are {unique_texas_zipcodes_count}")
```

```
The number of unique zipcodes are 1968
```

Making a subset of zipcodes in Texas and in bordering states

```
zips_texas_borderstates_centroids =
↪ zips_all_centroids[zips_all_centroids["ZCTA5"].str.startswith(("75",
↪ "76", "77", "78", "79", "718", "885", "733", "70", "71", "72", "73",
↪ "87", "88"))]

unique_texas_bs_zipcodes_count =
↪ zips_texas_borderstates_centroids["ZCTA5"].nunique()
print(f"The number of unique zipcodes are {unique_texas_bs_zipcodes_count}")
```

```
The number of unique zipcodes are 3724
```

3.

```python
# getting short-term active hospitals in 2016
active_2016 =
↪  short_term_hospitals_2016[short_term_hospitals_2016['PGM_TRMNTN_CD']==0]

active_2016_grouped = active_2016.groupby("ZIP_CD").size().reset_index(name =
↪  "count")

zips_withhospital_centroids =
↪  zips_texas_borderstates_centroids.merge(active_2016_grouped,
↪  left_on='ZCTA5', right_on='ZIP_CD',  how='inner')
```

I decided to do an inner merge and I merged it on the basis of the zipcode column.

4. a. Subsetting to 10 zipcodes

   ::: {.cell execution_count=32} "' {.python .cell-code} import time

   zips_texas_centroids = zips_texas_centroids.to_crs(epsg=2272) zips_withhospital_centroids = zips_withhospital_centroids.to_crs(epsg=2272)

   zips_texas_centroids_subset = zips_texas_centroids.sample(n=10)

   start_time = time.time()

   nearest_hospitals = gpd.sjoin_nearest(zips_texas_centroids_subset, zips_withhospital_centroids, how='inner',
   distance_col="distance")

   end_time = time.time()

   time_taken = end_time - start_time

   print("Time taken for 10 zipcodes:" , time_taken,"seconds.") "'

   ::: {.cell-output .cell-output-stdout} `Time taken for 10 zipcodes: 0.007531881332397461 seconds.` ::: :::

   For the entire dataset it will take approximately:

   ::: {.cell execution_count=33} `{.python .cell-code}  time_approx = (zips_texas_centroids.shap`
   `print("Approx time that should be taken for the whole dataset:" , time_approx , "seconds.")`

   ::: {.cell-output .cell-output-stdout} `Approx time that should be taken for the whole dataset: 1.4822742462158205 seconds.` ::: :::

   b. Doing this on the whole dataset

::: {.cell execution_count=34} “‘ {.python .cell-code} start_time = time.time()

nearest_hospitals = gpd.sjoin_nearest(zips_texas_centroids, zips_withhospital_centroids, how=‘inner’,
distance_col=“distance”)

end_time = time.time()

time_taken = end_time - start_time

print(“Time taken for all zipcodes:” , time_taken,“seconds.”) “‘

::: {.cell-output .cell-output-stdout} `Time taken for all zipcodes: 0.017024993896484375 seconds.` ::: :::

It took lesser than what we had anticipated. I would say it is significantly lesser as essentially, since we’re dealing with seconds, this difference can be seen as significant.

c.

::: {.cell execution_count=35} “‘ {.python .cell-code} # reading .prj from pyproj import CRS

with open(“/Users/kishikamahajan/Desktop/gz_2010_us_860_00_500k/gz_2010_us_860_00_500k.prj
“r”) as prj_file: prj_text = prj_file.read() crs = CRS.from_wkt(prj_text)

print(f”The CRS of the file is {crs}“)
“‘

::: {.cell-output .cell-output-stdout} `The CRS of the file is GEOGCS["GCS_North_American_1983",DA`
::: :::

The units in the prj file are degrees - 0.017453292519943295. This in miles will be approximately 0.310281 miles.

5. a.

::: {.cell execution_count=36} “‘ {.python .cell-code} zips_texas_centroids = zips_texas_centroids.to_crs(epsg=2272) zips_withhospital_centroids = zips_withhospital_centroids.to_

nearest_hospitals = gpd.sjoin_nearest(zips_texas_centroids, zips_withhospital_centroids, how=‘inner’, distance_col=“distance”)

nearest_hospitals[‘distance_miles’] = nearest_hospitals[‘distance’] / 1609.34 “‘ :::

b.

::: {.cell execution_count=37} “‘ {.python .cell-code} import altair as alt

average_distance = nearest_hospitals.groupby(‘ZIP_CD’)[‘distance_miles’].mean().reset_index()

average_distance.rename(columns={‘distance_miles’: ‘average_distance_miles’}, inplace=True)

overall_average_distance = nearest_hospitals[‘distance_miles’].mean()

Print the overall average distance print(f”The overall average distance to the nearest hospital is {overall_average_distance:.2f} miles.”) “‘

::: {.cell-output .cell-output-stdout} `The overall average distance to the nearest hospital is 45.12 miles.` ::: :::

The number somehow make sense but ideally, we would expect the distance to be even smaller as we would expect that there are hospitals quite nearby from each zipcode.

c. Plotting value for each zipcode

::: {.cell execution_count=38} `{.python .cell-code} alt.Chart(average_distance).mark_bar().e`
`alt.X("average_distance_miles" , title = "Average distance in miles"),`
`alt.Y("ZIP_CD" , title = "Zipcodes") )`

::: {.cell-output .cell-output-display execution_count=233} `alt.Chart(...)` ::: :::

Effects of closures on access in Texas (15 pts)

1.

```
closures_by_zipcode =
 ↪  valid_closures.groupby('ZIP_CD').size().reset_index(name="count")

zips_texas_centroids["ZCTA5"] = zips_texas_centroids["ZCTA5"].astype(str)
closures_by_zipcode["ZIP_CD"] = closures_by_zipcode["ZIP_CD"].astype(str)

texas_closures = closures_by_zipcode.merge(
    zips_texas_centroids,
    left_on='ZIP_CD',
    right_on='ZCTA5',
    how='inner'
)

texas_closures.head()
```

ZIP_CD	count	GEO_ID	ZCTA5	NAME	LSAD	CENSUSAREA	geometry

2.
3.
4.

Reflecting on the exercise (10 pts)

Here's a more concise version of the potential issues with identifying hospital closures:

Data Collection Timing: Regular data collection schedules might miss closure events, requiring a more robust system for continuous data updates. Closure Type Distinction: Temporary closures (due to renovation, financial recovery) vs permanent closures can lead to overestimation of actual hospital closures. Identity Changes: Mergers and acquisitions can alter hospital identifiers, potentially misidentifying closures. Geographic Data Accuracy: Incorrect geocoding or changing zip-code boundaries can lead to misattribution of closure locations. Data Collection Methods: Machine learning could improve tracking hospital activity and closure data reliability.

Partner 2: Identifying zip codes affected hospital closure is a good start point when we have such a large dataset in hand. However, it may sometime miss to give a true reality of hospital closures given that there could be various factors affecting closure of hospitals: First of all, not all ZIP codes are uniquely distributed in terms of resources and population. Areas with higher populations, older adults, or those with chronic health needs may experience a greater impact from closures. Adjusting for these factors can help prioritise areas with higher healthcare needs.Secondly, ZIP codes have a geographical factor and its impact on public health structurally attached to it. Some places are more prone to public diseases and hence demand more hospitals. While some area would demand less and have lesser footfall relatively. Thus, geographical factors can have huge impact on hospital operations. Transportation and Accessibility: Evaluate the availability of public transportation or other means of reaching nearby hospitals, especially in rural areas where closures may significantly reduce accessible options. A measure that considers public transit access, car ownership rates, or alternative transportation services could improve the assessment. Socioeconomic Indicators: Socioeconomic factors, such as income levels or insurance coverage, influence access to healthcare. Low-income areas may face greater challenges with closures, as residents may have fewer alternatives. Including socioeconomic indicators can highlight where closures are likely to exacerbate existing health disparities.

Thus overall we need to also include geographical factors, deomgraphy, economic distribution, transportation to assess the real reasoning for hospital closures.