CasaccioDSC540 - Final Project

February 24, 2024

0.0.1 Author: Alysen Casaccio

0.0.2 Due Date: 3/2/2024

0.0.3 Milestone Document 1 - 5 - Final Project

0.0.4 Milestone 1: Finding the Data (See Word Document)

0.0.5 Milestone 2: Cleaning and Formatting Flat File Source

I have a total of five flat files for this project. I am hoping the data within them can come together to discover to tell a compelling story about the current opioid usages and deaths in the US today.

[94]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import requests
from bs4 import BeautifulSoup
from datetime import datetime
import json
import dask.dataframe as dd
import sqlite3
import seaborn as sns
```

[2]:
```
# defining file paths for all my flat files, CSV and XLSX
file_paths = {
    'deathrates': r'C:\Users\alyse\OneDrive\Documents\Bellevue University\DSC␣
 ↪540 - Data Preparation\Final Project Data\drugoverdosedeathrates2019.csv',
    'opioidproviders': r'C:\Users\alyse\OneDrive\Documents\Bellevue␣
 ↪University\DSC 540 - Data Preparation\Final Project Data\opioidproviders2023.
 ↪csv',
    'demographic_counts': r'C:\Users\alyse\OneDrive\Documents\Bellevue␣
 ↪University\DSC 540 - Data Preparation\Final Project␣
 ↪Data\totalmedicaredeathsbydemographic.xlsx',
    'month_counts': r'C:\Users\alyse\OneDrive\Documents\Bellevue University\DSC␣
 ↪540 - Data Preparation\Final Project Data\totalmedicaredeathsbymonth.xlsx',
    'state_counts': r'C:\Users\alyse\OneDrive\Documents\Bellevue University\DSC␣
 ↪540 - Data Preparation\Final Project Data\totalmedicaredeathsbystate.xlsx'}

# generating dataframes
```

```
dataframes = {}
for name, file_path in file_paths.items():
    if file_path.lower().endswith('.csv'):
        dataframes[name] = pd.read_csv(file_path)
    elif file_path.lower().endswith('.xlsx'):
        dataframes[name] = pd.read_excel(file_path)

# assigning dataframes to variables for ease of coding
deathrates = dataframes['deathrates']
opioidproviders = dataframes['opioidproviders']
demographic_counts = dataframes['demographic_counts']
month_counts = dataframes['month_counts']
state_counts = dataframes['state_counts']
```

[3]:
```
# checking the head and shape of each dataframe
for name, df in dataframes.items():
    print(f"DataFrame: {name}")
    print(df.head())
    print(f"Shape: {df.shape}")
    print("\n")
```

```
DataFrame: deathrates
                    INDICATOR                     PANEL  PANEL_NUM  \
0  Drug overdose death rates  All drug overdose deaths          0
1  Drug overdose death rates  All drug overdose deaths          0
2  Drug overdose death rates  All drug overdose deaths          0
3  Drug overdose death rates  All drug overdose deaths          0
4  Drug overdose death rates  All drug overdose deaths          0

                                                UNIT  UNIT_NUM STUB_NAME  \
0  Deaths per 100,000 resident population, age-ad…         1     Total
1  Deaths per 100,000 resident population, age-ad…         1     Total
2  Deaths per 100,000 resident population, age-ad…         1     Total
3  Deaths per 100,000 resident population, age-ad…         1     Total
4  Deaths per 100,000 resident population, age-ad…         1     Total

   STUB_NAME_NUM   STUB_LABEL  STUB_LABEL_NUM  YEAR  YEAR_NUM       AGE  \
0              0  All persons             0.1  1999         1  All ages
1              0  All persons             0.1  2000         2  All ages
2              0  All persons             0.1  2001         3  All ages
3              0  All persons             0.1  2002         4  All ages
4              0  All persons             0.1  2003         5  All ages

   AGE_NUM  ESTIMATE FLAG
0      1.1       6.1  NaN
1      1.1       6.2  NaN
2      1.1       6.8  NaN
3      1.1       8.2  NaN
```

```
4      1.1       8.9  NaN
Shape: (6228, 15)


DataFrame: opioidproviders
                     NPI                        PROVIDER NAME  \
0  1003081399 1013055110  BAART BEHAVIORAL HEALTH SERVICES IN
1             1003150004                 AMS OF WISCONSIN LLC
2             1003362484                         BHG XLII LLC
3             1003368945                         RTS EDGEWOOD
4             1003571647        METRO TREATMENT OF FLORIDA LP

              ADDRESS LINE 1                  ADDRESS LINE 2            CITY  \
0            617 COMSTOCK RD                           STE 5          BERLIN
1      9532 E 16 FRONTAGE RD                         STE 100        ONALASKA
2      5715 PRINCESS ANNE RD                             NaN  VIRGINIA BEACH
3       2205 PULASKI HIGHWAY                             NaN        EDGEWOOD
4  1241 BLANDING BLVD, STE 5  NEW SEASON TREATMENT CENTER 21     ORANGE PARK

  STATE         ZIP MEDICARE ID EFFECTIVE DATE       PHONE
0    VT  05602-8498                   1/1/2020  8022232003
1    WI  54650-6742                   1/1/2020  9202322332
2    VA  23462-3222                   1/1/2020  7579620748
3    MD       21040                 10/13/2020  4434569001
4    FL  32065-5908                   1/1/2020  9046700820
Shape: (1452, 9)


DataFrame: demographic_counts
                                MDCR ENROLL AB 34 Unnamed: 1  \
0  Medicare Deaths:  Total, Original Medicare, an…        NaN
1  Beneficiaries, by Demographic Characteristics,…        NaN
2                       Demographic Characteristic      Total
3                                            BLANK        NaN
4                                            Total    2337988

         Unnamed: 2                              Unnamed: 3
0               NaN                                     NaN
1               NaN                                     NaN
2  Original Medicare  Medicare Advantage and Other Health Plans
3               NaN                                     NaN
4           1505119                                  832869
Shape: (36, 4)


DataFrame: month_counts
                                MDCR ENROLL AB 33 Unnamed: 1 Unnamed: 2  \
0  Medicare Deaths:  Total (Original Medicare and…        NaN        NaN
```

```
1                                                           Year      Total    January
2                                                          BLANK        NaN        NaN
3                                                           2014    2144287     199003
4                                                           2015    2220438     222132

  Unnamed: 3 Unnamed: 4 Unnamed: 5 Unnamed: 6 Unnamed: 7 Unnamed: 8  \
0        NaN        NaN        NaN        NaN        NaN        NaN
1   February      March      April        May       June       July
2        NaN        NaN        NaN        NaN        NaN        NaN
3     174146     187917     177001     177522     166394     170137
4     189056     200318     184375     181780     170834     175098

  Unnamed: 9 Unnamed: 10 Unnamed: 11 Unnamed: 12 Unnamed: 13
0        NaN         NaN         NaN         NaN         NaN
1     August   September     October    November    December
2        NaN         NaN         NaN         NaN         NaN
3     168593      167220      178505      176032      201817
4     172904      170027      182253      179780      191881
Shape: (11, 14)


DataFrame: state_counts
                                       MDCR ENROLL AB 35 Unnamed: 1 Unnamed: 2  \
0  Medicare Deaths:  Total (Original Medicare and…        NaN        NaN
1          by Area of Residence, Calendar Year 2019        NaN        NaN
2                                 Area of Residence      Total       Aged
3                                             BLANK        NaN        NaN
4                                         All Areas    2337988    2138655

  Unnamed: 3
0        NaN
1        NaN
2   Disabled
3        NaN
4     199333
Shape: (72, 4)
```

Opioid Providers Flat File - Data Exploration I could already see areas where cleanup of the data would be useful. I wanted to begin with the simplest df first, opioidproviders.
I noticed there were two columns that seemed to represent NPI numbers, which could serve as a good unique identifier for each row, provided there wasn't confusion between the two columns. Here I took a closer look:

[4]:
```
# viewing the headers of the first two columns
headers_first_two = opioidproviders.columns[:2]
```

```python
print("Headers of the first two columns:")
print(headers_first_two)

# viewing the first 20 rows of the first two columns
first_two_columns_data = opioidproviders.iloc[:, :2]
print("\nFirst 20 rows of the first two columns:")
print(first_two_columns_data.head(20))
```

```
Headers of the first two columns:
Index(['NPI', 'PROVIDER NAME'], dtype='object')

First 20 rows of the first two columns:
                       NPI                    PROVIDER NAME
0   1003081399 1013055110  BAART BEHAVIORAL HEALTH SERVICES IN
1              1003150004              AMS OF WISCONSIN LLC
2              1003362484                      BHG XLII LLC
3              1003368945                      RTS EDGEWOOD
4              1003571647       METRO TREATMENT OF FLORIDA LP
5   1003581174 1326713314           PREMIER CARE OF OHIO, LLC
6              1003583733  AFFINITY HEALTHCARE GROUP CHERRY HI
7              1003947193  WEST TEXAS COUNSELING & REHABILITAT
8              1003953548            ALLIANCE RECOVERY CENTER
9              1003953548            ALLIANCE RECOVERY CENTER
10             1003953548            ALLIANCE RECOVERY CENTER
11             1003958976            WESTERN PACIFIC MED-CORP
12             1003960022   SOUTHEASTERN COUNCIL ON ALCOHOLISM
13             1003969767       RICHMOND TREATMENT CENTER LLC
14             1003972654         AEGIS TREATMENT CENTERS LLC
15             1013051606                          WCHS INC
16             1013055110  BAART BEHAVIORAL HEALTH SERVICES IN
17             1013060714                  EAST INDIANA LLC
18             1013141977              DISCOVERY HOUSE BC LLC
19             1013345065     MONTEFIORE MOUNT VERNON HOSPITAL
```

I mistakenly thought the first column was without a header and the 2nd column was NPIs. It turns out, the first column was NPIs and in some cases, there was more than one NPI, separated by a space. In considering what I had seen of NPIs in the past, it seemed likely that the NPIs represented were primarily the NPI associated with the practice, as this is more widely utilized in payor spaces and the provider names all appear to be practice names. I checked some CMS websites regarding the presence of more than one practice NPI and found that practice groups are allowed a second NPI if they chose, to represent a separate functional location, like a lab processing department.

Since I wasn't yet sure if this would cause analysis issues later, I decided to split the columns into primary and secondary NPI columns. Before doing that, however, I wanted a closer look at some rows with more than one NPI.

[5]:

```python
# filtering rows with two NPIs in the first column based on character length
rows_with_two_npis = opioidproviders[opioidproviders['NPI'].str.len() > 10]

# displaying the first 20 rows
print("Rows with two NPIs in the first column:")
print(rows_with_two_npis.head(20))
```

```
Rows with two NPIs in the first column:
                                                   NPI  \
0                                1003081399 1013055110
5                                1003581174 1326713314
93                               1043641293 1851957021
104  1053458885 1184992331 1205035425 1386781953 15…
114                              1063012334 1396343844
133                              1063922060 1841782042
139                              1073679445 1487889655
154                   1083945125 1124579875 1578821153
174                              1104184670 1487889655
223                              1144386442 1487889655
229                              1144684697 1700250784
238                   1154691368 1487709671 1578805776
267                              1174544092 1962423889
274                              1174798524 1790837128
275                              1174925010 1497260202
367        1235674409 1295787661 1386895985 1669424024
383                   1255349726 1720480668 1861906687
446                              1306277082 1639362254
464                              1316503584 1942728282
473                              1326283706 1750865804

                         PROVIDER NAME             ADDRESS LINE 1  \
0    BAART BEHAVIORAL HEALTH SERVICES IN            617 COMSTOCK RD
5              PREMIER CARE OF OHIO, LLC     2632 WOODMAN CENTER CT
93             PREMIER CARE OF OHIO, LLC             1380 DUBLIN RD
104  ADDICTION RESEARCH AND TREATMENT IN            2158 SOLANO WAY
114            PREMIER CARE OF OHIO, LLC            5 SEVERANCE CIR
133  REDEEM HEALTHCARE AND MEDICAL SYSTE          917 N CAROLINE ST
139                 APT FOUNDATION, INC               352 STATE ST
154  PRINCE GEORGE'S COUNTY HEALTH DEPAR           3003 HOSPITAL DR
174                 APT FOUNDATION, INC             54 RAMSDELL ST
223                 APT FOUNDATION, INC           495 CONGRESS AVE
229  MILWAUKEE HEALTH SERVICES SYSTEM LL     3440 OAKWOOD HILLS PKWY
238    LEXINGTON CENTER FOR RECOVERY INC            41 PAGE PARK DR
267       BI-VALLEY MEDICAL CLINIC, INC.             310 HARRIS AVE
274  MILWAUKEE HEALTH SERVICES SYSTEM LL             4800 S 10TH ST
275     HUNTINGTON TREATMENT CENTER, LLC                135 4TH AVE
367  START TREATMENT & RECOVERY CENTERS,  119-121 WEST 124TH STREET
383          CHARLESTON TREATMENT CENTER         2157 GREENBRIER ST
```

446	OKLAHOMA TREATMENT SERVICES, LLC	3445 S SHERIDAN RD
464	PREMIER CARE OF OHIO, LLC	2727 SAINT JOHNS RD
473	C.O.R.E. MEDICAL CLINIC, INC	2100 CAPITOL AVE

	ADDRESS LINE 2	CITY	STATE	ZIP	\
0	STE 5	BERLIN	VT	05602-8498	
5	NaN	KETTERING	OH	45420-1477	
93	STE 100	COLUMBUS	OH	43215-1025	
104	NaN	CONCORD	CA	94520-4700	
114	STE 101	CLEVELAND HEIGHTS	OH	44118-1513	
133	NaN	BALTIMORE	MD	21205-1000	
139	NaN	NORTH HAVEN	CT	06473-3108	
154	NaN	CHEVERLY	MD	20785-1194	
174	NaN	NEW HAVEN	CT	06515-1616	
223	NaN	NEW HAVEN	CT	06519-1312	
229	NaN	EAU CLAIRE	WI	54701-7698	
238	STE 200	POUGHKEEPSIE	NY	12603-7500	
267	STE A, E, F, G	SACRAMENTO	CA	95823-2627	
274	UNIT 1	MILWAUKEE	WI	53221-2412	
275	NaN	HUNTINGTON	WV	25701-1219	
367	NaN	NEW YORK	NY	10027	
383	NaN	CHARLESTON	WV	25311-9623	
446	NaN	TULSA	OK	74145-1105	
464	STE D	LIMA	OH	45804-4029	
473	NaN	SACRAMENTO	CA	95816-5721	

	MEDICARE ID	EFFECTIVE DATE	PHONE
0		1/1/2020	8022232003
5		1/1/2020	9377397100
93		1/1/2020	6144887117
104		1/1/2020	3232420500
114		1/1/2020	2168598300
133		7/15/2020	4105221030
139		1/1/2020	2037814600x1703
154		3/10/2020	3018837814
174		1/1/2020	2037814600
223		1/1/2020	2037814600x1703
229		1/1/2020	7152142525
238		1/1/2020	8454862850x2010
267		1/1/2020	9166496793
274		1/1/2020	4147449052
275		1/1/2020	3045255691
367		1/1/2020	2129322676
383		1/1/2020	3043445924
446		3/26/2020	9186103366
464		1/1/2020	5679409145
473		1/1/2020	9164424985x305

That was surprising. I found quite a few rows with more than two NPIs, which made me curious about how many were really represented.

[6]:
```python
# finding the N longest values by character length
def top_n_longest_values(df, column_name, n):
    # filtering out NaN values and sorting by character length in descending␣
↪order
    sorted_df = df.dropna(subset=[column_name]).sort_values(by=column_name,␣
↪key=lambda x: x.str.len(), ascending=False)
    return sorted_df.head(n)

# finding the 5 longest values in the 'NPI' column
top_5_longest = top_n_longest_values(opioidproviders, 'NPI', 5)

# printing the rows with the 5 longest 'NPI' values
print("Top 5 longest NPI values:")
for idx, row in top_5_longest.iterrows():
    print(f"Row {idx}: {row['NPI']}\n")
```

```
Top 5 longest NPI values:
Row 104: 1053458885 1184992331 1205035425 1386781953 1508904780 1538206297
1659419828 1679062434 1679921241 1730226465 1730226473 1780722033

Row 367: 1235674409 1295787661 1386895985 1669424024

Row 1185: 1801298740 1891201703 1982614772

Row 383: 1255349726 1720480668 1861906687

Row 238: 1154691368 1487709671 1578805776
```

After seeing the longest NPIs in the column, it seemed evident these were all practice locations. If these were individual provider NPIs (which could cause duplicate issues when the same provider was working at more than one practice), we would be seeing significantly higher numbers of NPIs in a given row.
I decided to run a check for duplicates and leave the multiple NPIs alone.

[7]:
```python
# checking for duplicates in the dataframe
duplicates = opioidproviders[opioidproviders.duplicated(keep=False)]

# counting the number of duplicate rows
num_duplicates = len(duplicates)

# printing the number of duplicate rows
print(f"Number of duplicate rows in 'opioidproviders' DataFrame:␣
↪{num_duplicates}")
```

```
# displaying the duplicate rows
if num_duplicates > 0:
    print("\nDuplicate Rows:")
    print(duplicates)
```

```
Number of duplicate rows in 'opioidproviders' DataFrame: 0
```

I found no duplicates at all, which wasn't surprising for a file coming from CMS. Their data tends to be fairly clean.
Next, I wanted to see where the NaN values were falling across the different columns.

[8]:
```
# counting NaN values in each column of the dataframe
nan_counts = opioidproviders.isna().sum()

# printing the count of NaN values for each column
print("NaN counts in each column of 'opioidproviders' DataFrame:")
print(nan_counts)
```

```
NaN counts in each column of 'opioidproviders' DataFrame:
NPI                              0
PROVIDER NAME                    0
ADDRESS LINE 1                   0
ADDRESS LINE 2                 904
CITY                             0
STATE                            0
ZIP                              0
MEDICARE ID EFFECTIVE DATE       0
PHONE                            0
dtype: int64
```

Opioid Providers Flat File - Adjustments I was finding this dataset to be impressively clean. There were no NaN values outside of the Address Line 2 column, and when reviewing the head I found that address line 2 wass only used to store the practice's suite or office number if they had one. Since I wouldn't be using that data for any analysis, I opted to simply drop the column entirely.

[9]:
```
# dropping the 'ADDRESS LINE 2' column from the dataframe
opioidproviders = opioidproviders.drop(columns=['ADDRESS LINE 2'])

# verifying that the column has been dropped
print("DataFrame after dropping 'ADDRESS LINE 2' column:")
print(opioidproviders.head())
```

```
DataFrame after dropping 'ADDRESS LINE 2' column:
                     NPI                        PROVIDER NAME  \
0  1003081399 1013055110  BAART BEHAVIORAL HEALTH SERVICES IN
1             1003150004                AMS OF WISCONSIN LLC
2             1003362484                        BHG XLII LLC
3             1003368945                        RTS EDGEWOOD
```

```
4              1003571647        METRO TREATMENT OF FLORIDA LP

               ADDRESS LINE 1            CITY STATE         ZIP  \
0            617 COMSTOCK RD           BERLIN    VT  05602-8498
1       9532 E 16 FRONTAGE RD         ONALASKA    WI  54650-6742
2       5715 PRINCESS ANNE RD   VIRGINIA BEACH    VA  23462-3222
3         2205 PULASKI HIGHWAY        EDGEWOOD    MD       21040
4   1241 BLANDING BLVD, STE 5      ORANGE PARK    FL  32065-5908

  MEDICARE ID EFFECTIVE DATE       PHONE
0                   1/1/2020  8022232003
1                   1/1/2020  9202322332
2                   1/1/2020  7579620748
3                 10/13/2020  4434569001
4                   1/1/2020  9046700820
```

I was happy with this and am hoping I will be able to connect these practices listed as opioid providers with my other datasets somehow.
I also wanted to change the column names to a lowercase formatting to better align with what I would use in the other dataframes.

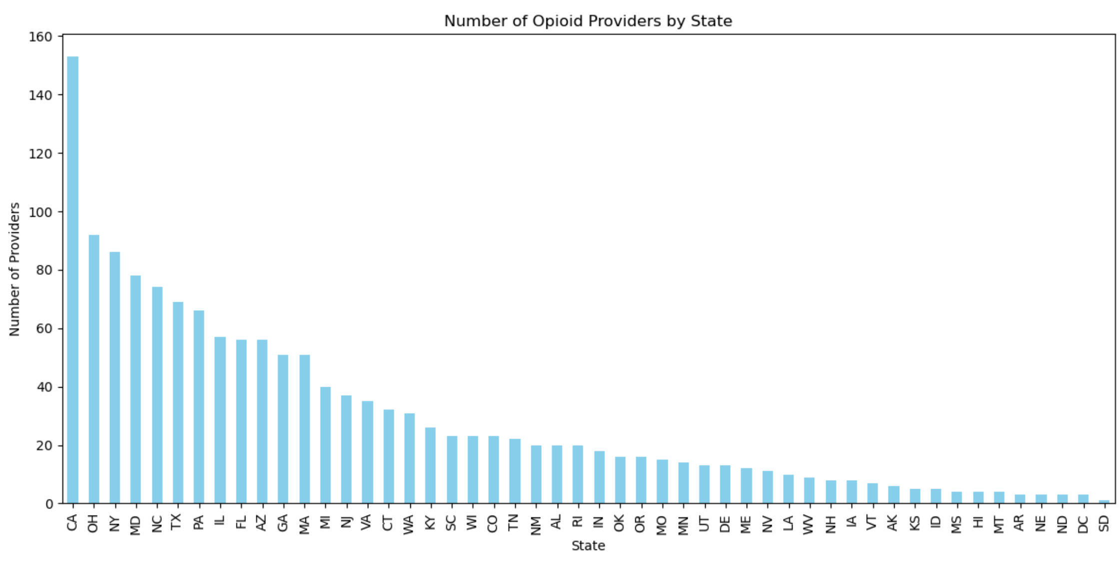
[10]:
```python
# defining a dictionary to map the old column names to the new lowercase names
column_mapping = {
    'NPI': 'practice_npis',
    'PROVIDER NAME': 'practice_name',
    'ADDRESS LINE 1': 'address',
    'CITY': 'city',
    'STATE': 'state',
    'ZIP': 'zip',
    'MEDICARE ID EFFECTIVE DATE': 'medicare_date',
    'PHONE': 'phone_number'}

# renaming the columns with the new names
opioidproviders = opioidproviders.rename(columns=column_mapping)

# checking the updated names
print("DataFrame with Updated Column Names:")
print(opioidproviders.head())
```

```
DataFrame with Updated Column Names:
            practice_npis                        practice_name  \
0   1003081399 1013055110   BAART BEHAVIORAL HEALTH SERVICES IN
1              1003150004                 AMS OF WISCONSIN LLC
2              1003362484                         BHG XLII LLC
3              1003368945                         RTS EDGEWOOD
4              1003571647        METRO TREATMENT OF FLORIDA LP

                   address            city state         zip medicare_date  \
0            617 COMSTOCK RD           BERLIN    VT  05602-8498      1/1/2020
```

```
1       9532 E 16 FRONTAGE RD         ONALASKA    WI  54650-6742       1/1/2020
2       5715 PRINCESS ANNE RD  VIRGINIA BEACH    VA  23462-3222       1/1/2020
3        2205 PULASKI HIGHWAY         EDGEWOOD    MD       21040    10/13/2020
4  1241 BLANDING BLVD, STE 5      ORANGE PARK    FL  32065-5908       1/1/2020

  phone_number
0   8022232003
1   9202322332
2   7579620748
3   4434569001
4   9046700820
```

As a last thought, now that the data was clean and formatted the way I wanted it, I thought I would pull a count of how many opioid providers were in each State, just as an easy query to look at before moving on.

[11]:
```python
# grouping the data by state and counting the number of providers in each state
state_counts = opioidproviders['state'].value_counts()

# creating a bar plot to compare the number of opioid providers in different␣
↪states
plt.figure(figsize=(12, 6))
state_counts.plot(kind='bar', color='skyblue')
plt.title('Number of Opioid Providers by State')
plt.xlabel('State')
plt.ylabel('Number of Providers')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



I didn't yet have any files that could help interpret and adjust this count by population, and that became clearly evident as a gap after I ran this bar plot.
It seemed a good spot to stop digging in the opioid provider file, however, since the data was clean and I needed to check my other data sources. I imagined I would return to this later.

Death Rates Flat File – Exploration I reviewed the head and shape of this file again and felt uncertain about the meaning of all the columns being presented there. Before looking into each row, I wanted to understand the number of unique values for each column.

[12]:
```
# displaying the count of unique values in each column of the 'deathrates'␣
↪dataframe
unique_value_counts = deathrates.nunique()

# printing the count of unique values for each column
print("Count of Unique Values in Each Column of 'deathrates' DataFrame:")
print(unique_value_counts)
```

```
Count of Unique Values in Each Column of 'deathrates' DataFrame:
INDICATOR           1
PANEL               6
PANEL_NUM           6
UNIT                2
UNIT_NUM            2
STUB_NAME           8
STUB_NAME_NUM       6
STUB_LABEL         52
STUB_LABEL_NUM     50
YEAR               20
YEAR_NUM           20
AGE                10
AGE_NUM            10
ESTIMATE          322
FLAG                1
dtype: int64
```

The CDC source documentation described the "INDICATOR" column as "the measure being estimated", so I wanted to see the single unique value in the column.

[13]:
```
# viewing the unique value
unique_values_indicator = deathrates['INDICATOR'].unique()
print("Unique Values in the 'INDICATOR' Column:")
print(unique_values_indicator)
```

```
Unique Values in the 'INDICATOR' Column:
['Drug overdose death rates']
```

The documentation described the "PANEL" and "PANEL_NUM" columns as submeasures of the indicator and numeric codes associated with those.

[14]:
```python
# viewing unique values in the 'PANEL' and 'PANEL_NUM' columns
unique_values_panel = deathrates['PANEL'].unique()
unique_values_panel_num = deathrates['PANEL_NUM'].unique()

print("Unique Values in the 'PANEL' Column:")
print(unique_values_panel)
print("\nUnique Values in the 'PANEL_NUM' Column:")
print(unique_values_panel_num)
```

```
Unique Values in the 'PANEL' Column:
['All drug overdose deaths' 'Drug overdose deaths involving any opioid'
 'Drug overdose deaths involving natural and semisynthetic opioids'
 'Drug overdose deaths involving methadone'
 'Drug overdose deaths involving other synthetic opioids (other than methadone)'
 'Drug overdose deaths involving heroin']

Unique Values in the 'PANEL_NUM' Column:
[0 1 2 3 4 5]
```

I was starting to develop an understanding of the data, and was considering different ways to describe it in the header.
In the meantime, I continued to review the columns.

According to the documentation, Unit and Unit_Num described the unit of measurement where stub_name and stub_name_num described the population category, where the stub_label and stub_label_num was described as the population subgroup associated with the 'estimate'. These all seemed to be ways of slicing/dicing demographic groupings, so I wanted to take a look at those next.

[15]:
```python
# viewing unique values in the 'UNIT', 'UNIT_NUM','STUB_NAME','STUB_NAME_NUM',␣
↪'STUB_LABEL', and 'STUB_LABEL_NUM' columns
unique_values_unit = deathrates['UNIT'].unique()
unique_values_unit_num = deathrates['UNIT_NUM'].unique()
unique_values_stub_name = deathrates['STUB_NAME'].unique()
unique_values_stub_name_num = deathrates['STUB_NAME_NUM'].unique()
unique_values_stub_label = deathrates['STUB_LABEL'].unique()
unique_values_stub_label_num = deathrates['STUB_LABEL_NUM'].unique()

print("Unique Values in the 'UNIT' Column:")
print(unique_values_unit)
print("\nUnique Values in the 'UNIT_NUM' Column:")
print(unique_values_unit_num)
print("\nUnique Values in the 'STUB_NAME' Column:")
print(unique_values_stub_name)
print("\nUnique Values in the 'STUB_NAME_NUM' Column:")
print(unique_values_stub_name_num)
print("\nUnique Values in the 'STUB_LABEL' Column:")
print(unique_values_stub_label)
```

```python
print("\nUnique Values in the 'STUB_LABEL_NUM' Column:")
print(unique_values_stub_label_num)
```

```
Unique Values in the 'UNIT' Column:
['Deaths per 100,000 resident population, age-adjusted'
 'Deaths per 100,000 resident population, crude']

Unique Values in the 'UNIT_NUM' Column:
[1 2]

Unique Values in the 'STUB_NAME' Column:
['Total' 'Sex' 'Sex and race' 'Sex and race and Hispanic origin' 'Age'
 'Sex and age' 'Sex and race (single race)'
 'Sex and race and Hispanic origin (single race)']

Unique Values in the 'STUB_NAME_NUM' Column:
[0 2 4 5 1 3]

Unique Values in the 'STUB_LABEL' Column:
['All persons' 'Male' 'Female' 'Male: White'
 'Male: Black or African American'
 'Male: American Indian or Alaska Native'
 'Male: Asian or Pacific Islander' 'Female: White'
 'Female: Black or African American'
 'Female: American Indian or Alaska Native'
 'Female: Asian or Pacific Islander' 'Male: Hispanic or Latino: All races'
 'Male: Not Hispanic or Latino: White'
 'Male: Not Hispanic or Latino: Black'
 'Male: Not Hispanic or Latino: American Indian or Alaska Native'
 'Male: Not Hispanic or Latino: Asian or Pacific Islander'
 'Female: Hispanic or Latino: All races'
 'Female: Not Hispanic or Latino: White'
 'Female: Not Hispanic or Latino: Black'
 'Female: Not Hispanic or Latino: American Indian or Alaska Native'
 'Female: Not Hispanic or Latino: Asian or Pacific Islander'
 'Under 15 years' '15-24 years' '25-34 years' '35-44 years' '45-54 years'
 '55-64 years' '65-74 years' '75-84 years' '85 years and over'
 'Male: Under 15 years' 'Male: 15-24 years' 'Male: 25-34 years'
 'Male: 35-44 years' 'Male: 45-54 years' 'Male: 55-64 years'
 'Male: 65-74 years' 'Male: 75-84 years' 'Male: 85 years and over'
 'Female: Under 15 years' 'Female: 15-24 years' 'Female: 25-34 years'
 'Female: 35-44 years' 'Female: 45-54 years' 'Female: 55-64 years'
 'Female: 65-74 years' 'Female: 75-84 years' 'Female: 85 years and over'
 'Male: Not Hispanic or Latino: Asian'
 'Male: Not Hispanic or Latino: Native Hawaiian or Other Pacific Islander'
 'Female: Not Hispanic or Latino: Asian'
 'Female: Not Hispanic or Latino: Native Hawaiian or Other Pacific Islander']
```

```
Unique Values in the 'STUB_LABEL_NUM' Column:
[0.1  2.1  2.2  4.1  4.2  4.3  4.4  4.5  4.6  4.7  4.8  5.1  5.2  5.3
 5.4  5.5  5.6  5.7  5.8  5.9  5.91 1.1  1.2  1.3  1.4  1.5  1.6  1.7
 1.8  1.9  3.11 3.12 3.13 3.14 3.15 3.16 3.17 3.18 3.19 3.21 3.22 3.23
 3.24 3.25 3.26 3.27 3.28 3.29 5.92 5.93]
```

The STUB_LABEL and STUB_LABEL_NUM columns had an interesting numbering and clustering convention that wasn't necessarily how I would have organized this data, but I could see where it helped control the final number of rows while still providing the same granularity of information. It was definitely something I hadn't seen before and would need time to consider.

The next four columns I wanted to investigate were YEAR, YEAR_NUM, AGE, and AGE_NUM. I didn't expect any surprises here, and hoped the age groupings would be useful in conjunction with my other dataframes. One concern I was developing was that the granularity of data in the files so far wasn't as specific as I had hoped. I could always add a file and I knew that New York State has deidentified discharge data available publicly, which would include patients who were discharged due to demise, but those would be specific to the state and to only those patients who died while in an inpatient setting.
It would have to be something I could consider after examining all the data, rather than just these flat files.

[16]:
```python
# viewing unique values in the 'YEAR', 'YEAR_NUM', 'AGE', and 'AGE_NUM' columns
unique_values_year = deathrates['YEAR'].unique()
unique_values_year_num = deathrates['YEAR_NUM'].unique()
unique_values_age = deathrates['AGE'].unique()
unique_values_age_num = deathrates['AGE_NUM'].unique()

print("Unique Values in the 'YEAR' Column:")
print(unique_values_year)
print("\nUnique Values in the 'YEAR_NUM' Column:")
print(unique_values_year_num)
print("\nUnique Values in the 'AGE' Column:")
print(unique_values_age)
print("\nUnique Values in the 'AGE_NUM' Column:")
print(unique_values_age_num)
```

```
Unique Values in the 'YEAR' Column:
[1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012
 2013 2014 2015 2016 2017 2018]

Unique Values in the 'YEAR_NUM' Column:
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20]

Unique Values in the 'AGE' Column:
['All ages' 'Under 15 years' '15-24 years' '25-34 years' '35-44 years'
 '45-54 years' '55-64 years' '65-74 years' '75-84 years'
 '85 years and over']

Unique Values in the 'AGE_NUM' Column:
```

```
[1.1  1.2  1.3  1.4  1.5  1.6  1.7  1.8  1.9  1.91]
```

Since the deepest meaning of the dataset here is in the estimate column, there was one more column I wanted to check the values for and that was FLAG.

```python
[17]: # viewing the unique value in the 'FLAG' column
unique_value_flag = deathrates['FLAG'].unique()
print("Unique Value in the 'FLAG' Column:")
print(unique_value_flag)
```

```
Unique Value in the 'FLAG' Column:
[nan '*']
```

Getting to the meat of the dataframe, I wanted to take a look at a sample of unique values from the ESTIMATE column. I didn't need to see all of the unique values, just a selection.
I also wanted to see how the information in each row corresponded to the sample of unique estimates.

```python
[18]: # getting 20 unique values from the 'ESTIMATE' column
unique_estimate_values = deathrates['ESTIMATE'].unique()

# creating a list to store the selected rows
selected_rows = []

# iterating through the unique values and selecting a row for each value
for value in unique_estimate_values[:20]:  # picking the first 20 unique values
    row = deathrates[deathrates['ESTIMATE'] == value].iloc[0]  # picking the␣
 ↪first row for each value
    selected_rows.append(row)

# making a dataframe from the selected rows
selected_df = pd.DataFrame(selected_rows)

# printing the results
print("Selected Rows with All Column Values:")
print(selected_df)
```

```
Selected Rows with All Column Values:
                      INDICATOR                     PANEL  PANEL_NUM  \
0   Drug overdose death rates  All drug overdose deaths          0
1   Drug overdose death rates  All drug overdose deaths          0
2   Drug overdose death rates  All drug overdose deaths          0
3   Drug overdose death rates  All drug overdose deaths          0
4   Drug overdose death rates  All drug overdose deaths          0
5   Drug overdose death rates  All drug overdose deaths          0
6   Drug overdose death rates  All drug overdose deaths          0
7   Drug overdose death rates  All drug overdose deaths          0
8   Drug overdose death rates  All drug overdose deaths          0
11  Drug overdose death rates  All drug overdose deaths          0
12  Drug overdose death rates  All drug overdose deaths          0
13  Drug overdose death rates  All drug overdose deaths          0
```

```
14  Drug overdose death rates  All drug overdose deaths         0
15  Drug overdose death rates  All drug overdose deaths         0
16  Drug overdose death rates  All drug overdose deaths         0
17  Drug overdose death rates  All drug overdose deaths         0
18  Drug overdose death rates  All drug overdose deaths         0
20  Drug overdose death rates  All drug overdose deaths         0
21  Drug overdose death rates  All drug overdose deaths         0
22  Drug overdose death rates  All drug overdose deaths         0

                                                 UNIT  UNIT_NUM STUB_NAME  \
0   Deaths per 100,000 resident population, age-ad…         1     Total
1   Deaths per 100,000 resident population, age-ad…         1     Total
2   Deaths per 100,000 resident population, age-ad…         1     Total
3   Deaths per 100,000 resident population, age-ad…         1     Total
4   Deaths per 100,000 resident population, age-ad…         1     Total
5   Deaths per 100,000 resident population, age-ad…         1     Total
6   Deaths per 100,000 resident population, age-ad…         1     Total
7   Deaths per 100,000 resident population, age-ad…         1     Total
8   Deaths per 100,000 resident population, age-ad…         1     Total
11  Deaths per 100,000 resident population, age-ad…         1     Total
12  Deaths per 100,000 resident population, age-ad…         1     Total
13  Deaths per 100,000 resident population, age-ad…         1     Total
14  Deaths per 100,000 resident population, age-ad…         1     Total
15  Deaths per 100,000 resident population, age-ad…         1     Total
16  Deaths per 100,000 resident population, age-ad…         1     Total
17  Deaths per 100,000 resident population, age-ad…         1     Total
18  Deaths per 100,000 resident population, age-ad…         1     Total
20  Deaths per 100,000 resident population, age-ad…         1       Sex
21  Deaths per 100,000 resident population, age-ad…         1       Sex
22  Deaths per 100,000 resident population, age-ad…         1       Sex

    STUB_NAME_NUM   STUB_LABEL  STUB_LABEL_NUM  YEAR  YEAR_NUM       AGE  \
0               0  All persons             0.1  1999         1  All ages
1               0  All persons             0.1  2000         2  All ages
2               0  All persons             0.1  2001         3  All ages
3               0  All persons             0.1  2002         4  All ages
4               0  All persons             0.1  2003         5  All ages
5               0  All persons             0.1  2004         6  All ages
6               0  All persons             0.1  2005         7  All ages
7               0  All persons             0.1  2006         8  All ages
8               0  All persons             0.1  2007         9  All ages
11              0  All persons             0.1  2010        12  All ages
12              0  All persons             0.1  2011        13  All ages
13              0  All persons             0.1  2012        14  All ages
14              0  All persons             0.1  2013        15  All ages
15              0  All persons             0.1  2014        16  All ages
16              0  All persons             0.1  2015        17  All ages
17              0  All persons             0.1  2016        18  All ages
```

```
18              0  All persons              0.1  2017        19  All ages
20              2         Male              2.1  2000         2  All ages
21              2         Male              2.1  2001         3  All ages
22              2         Male              2.1  2002         4  All ages

    AGE_NUM  ESTIMATE  FLAG
0       1.1       6.1   NaN
1       1.1       6.2   NaN
2       1.1       6.8   NaN
3       1.1       8.2   NaN
4       1.1       8.9   NaN
5       1.1       9.4   NaN
6       1.1      10.1   NaN
7       1.1      11.5   NaN
8       1.1      11.9   NaN
11      1.1      12.3   NaN
12      1.1      13.2   NaN
13      1.1      13.1   NaN
14      1.1      13.8   NaN
15      1.1      14.7   NaN
16      1.1      16.3   NaN
17      1.1      19.8   NaN
18      1.1      21.7   NaN
20      1.1       8.3   NaN
21      1.1       9.0   NaN
22      1.1      10.6   NaN
```

My interpretation of what I selected was:
There were 20 rows which displayed the measurement of drug overdose death rates in the US.
These 20 rows included all types of drug overdose, and all estimates in the sample had been age-adjusted.
The first 18 rows represented total population, so no demographic break-out, where the last three rows were males only.
All rows represented all age groupings, and the first 18 rows represented the estimates for the years 1999 - 2017, with one year per row.
The last three rows of males only represented data from 2000, 2001, and 2002.

I was interested in the sign that drug overdose death estimates had been steadily increasing since 1999, but I needed to do a bit of cleanup before digging too deeply there.
At least I now clearly understood what the data held in this file.

Death Rates Flat File – Adjustments Here is what I decided to adjust for this file:
The indicator column has only told us that every record is a drug overdose death rate, which we know, so I dropped that column.
The panel column gives the drug overdose type, so I want to rename the header and simplify the values to make all of it more intuitive.
The panel_num column could be useful since I don't want to have to run queries by text-matching the values, so I'll leave it, but will change the header to match my lowercase formatting.

The unit column has lengthy values as well. I renamed the header and streamlined the values. The unit_num column assigns a numeric value to one of two values in the unit column, and I didn't feel that was necessary, so I dropped the column.
The stub_name column header was renamed and the values were streamlined, so the stub_name_num column was no longer necessary and was dropped.
The stub_label column header was also renamed along with the stub_label_num header to better align. I felt the numeric assignment could be helpful given the amount of detail in this column.
The year and age headers were renamed for formatting, but the values looked good in both, and the year_num and age_num columns were both dropped as the associations only added confusion. The estimate column was renamed for formatting as well, and the flag column dropped as it brought no additional data or value.

[19]:
```
# dropping the 'INDICATOR' column
deathrates = deathrates.drop(columns=['INDICATOR'])

# renaming the 'PANEL' column to 'overdose_type'
deathrates = deathrates.rename(columns={'PANEL': 'overdose_type'})
```

[20]:
```
# replacing values in the 'overdose_type' column (used to be PANEL)
deathrates['overdose_type'] = deathrates['overdose_type'].replace({
    'All drug overdose deaths': 'all overdoses',
    'Drug overdose deaths involving any opioid': 'any opioid',
    'Drug overdose deaths involving natural and semisynthetic opioids':␣
 ↪'natural or semisynthetic',
    'Drug overdose deaths involving methadone': 'methadone',
    'Drug overdose deaths involving other synthetic opioids (other than␣
 ↪methadone)': 'non-methadone synthetic',
    'Drug overdose deaths involving heroin': 'heroin'})
```

[21]:
```
# renaming the 'PANEL_NUM' column header to 'overdose_type_num'
deathrates = deathrates.rename(columns={'PANEL_NUM': 'overdose_type_num'})
```

[22]:
```
# renaming the 'UNIT' column header to 'deathsper100k'
deathrates = deathrates.rename(columns={'UNIT': 'deathsper100k'})

# replacing values in the 'deathsper100k' column
deathrates['deathsper100k'] = deathrates['deathsper100k'].replace({
    'Deaths per 100,000 resident population, age-adjusted': 'age_adjusted',
    'Deaths per 100,000 resident population, crude': 'crude'})
```

[23]:
```
# dropping the 'UNIT_NUM' column
deathrates = deathrates.drop(columns=['UNIT_NUM'])
```

[24]:
```
# renaming the 'STUB_NAME' column header to 'demographic_name'
deathrates = deathrates.rename(columns={'STUB_NAME': 'demographic_name'})

# dropping the 'STUB_NAME_NUM' column
```

```
deathrates = deathrates.drop(columns=['STUB_NAME_NUM'])

# updating unique values in the 'demographic_name' column
deathrates['demographic_name'] = deathrates['demographic_name'].replace({
    'Total': 'total',
    'Sex': 'gender',
    'Sex and race': 'gender and race',
    'Sex and race and Hispanic origin': 'gender race and ethnicity',
    'Age': 'age group',
    'Sex and age': 'gender and age group',
    'Sex and race (single race)': 'gender and race (single)',
    'Sex and race and Hispanic origin (single race)': 'gender race (single) and␣
 ↪ethnicity'})

# renaming the 'STUB_LABEL' column header to 'demographic_detail'
deathrates = deathrates.rename(columns={'STUB_LABEL': 'demographic_detail'})

# renaming the 'STUB_LABEL_NUM' column header to 'demographic_detail_num'
deathrates = deathrates.rename(columns={'STUB_LABEL_NUM':␣
 ↪'demographic_detail_num'})
```

[25]:
```
# renaming the 'YEAR' column header to 'year'
deathrates = deathrates.rename(columns={'YEAR': 'year'})

# dropping the 'YEAR_NUM' column
deathrates = deathrates.drop(columns=['YEAR_NUM'])
```

[26]:
```
# renaming the 'AGE' column header to 'age_group'
deathrates = deathrates.rename(columns={'AGE': 'age_group'})

# dropping the 'AGE_NUM' column
deathrates = deathrates.drop(columns=['AGE_NUM'])
```

[27]:
```
# renaming the 'ESTIMATE' column header to 'estimate'
deathrates = deathrates.rename(columns={'ESTIMATE': 'estimate'})

# dropping the 'FLAG' column
deathrates = deathrates.drop(columns=['FLAG'])
```

[28]:
```
# checking changes made to the df structure/headers
print("Head of the 'deathrates' DataFrame:")
print(deathrates.head())
print("\nShape of the 'deathrates' DataFrame:")
print(deathrates.shape)

# checking changes made to unique values
unique_overdose_type = deathrates['overdose_type'].unique()
```

```python
print("\nUnique values in the 'overdose_type' column:")
print(unique_overdose_type)

unique_deathsper100k = deathrates['deathsper100k'].unique()
print("\nUnique values in the 'deathsper100k' column:")
print(unique_deathsper100k)

unique_demographic_name = deathrates['demographic_name'].unique()
print("\nUnique values in the 'demographic_name' column:")
print(unique_demographic_name)
```

```
Head of the 'deathrates' DataFrame:
   overdose_type  overdose_type_num deathsper100k demographic_name  \
0  all overdoses                  0  age_adjusted            total
1  all overdoses                  0  age_adjusted            total
2  all overdoses                  0  age_adjusted            total
3  all overdoses                  0  age_adjusted            total
4  all overdoses                  0  age_adjusted            total

  demographic_detail  demographic_detail_num  year age_group  estimate
0        All persons                     0.1  1999  All ages       6.1
1        All persons                     0.1  2000  All ages       6.2
2        All persons                     0.1  2001  All ages       6.8
3        All persons                     0.1  2002  All ages       8.2
4        All persons                     0.1  2003  All ages       8.9

Shape of the 'deathrates' DataFrame:
(6228, 9)

Unique values in the 'overdose_type' column:
['all overdoses' 'any opioid' 'natural or semisynthetic' 'methadone'
 'non-methadone synthetic' 'heroin']

Unique values in the 'deathsper100k' column:
['age_adjusted' 'crude']

Unique values in the 'demographic_name' column:
['total' 'gender' 'gender and race' 'gender race and ethnicity'
 'age group' 'gender and age group' 'gender and race (single)'
 'gender race (single) and ethnicity']
```

Since I kept all 6228 rows and dropped 6 columns, I felt it was a good time to check for NaN values and duplicate records.
In some cases, I would consider records to be duplicative if only a few key values across the columns were identical. In this case, since we had estimated counts, not individual patient records, I would only count a record as a duplicate if all values across the columns were a match.

[29]:
```python
# counting the number of NaN values in each column
nan_count = deathrates.isna().sum()
print("Number of NaN values in each column:")
print(nan_count)

# counting the number of duplicate records
duplicate_count = deathrates.duplicated(keep='first').sum()
print("\nNumber of duplicate records (based on all columns):", duplicate_count)
```

```
Number of NaN values in each column:
overdose_type                0
overdose_type_num            0
deathsper100k                0
demographic_name             0
demographic_detail           0
demographic_detail_num       0
year                         0
age_group                    0
estimate                  1111
dtype: int64

Number of duplicate records (based on all columns): 0
```

Since there were a little over a thousand instances of NaN in the estimate column and I did plan to conduct mathematical operations using those values, I felt it would be safe to change those values from NaN to 0. There were times when that wouldn't be appropriate, but I was hoping it would be a good choice for this dataset.

[30]:
```python
# replacing NaN with zero in the estimate column
deathrates['estimate'] = deathrates['estimate'].fillna(0)
```

Death Rates Flat File – Some Num Tables I did leave a few of the num tables I felt could be useful later. I created visual tables to better display the correlation and naming convention I'll use in other files/data to ensure alignment.

[31]:
```python
# grouping by 'overdose_type' and getting unique 'overdose_type_num' for each␣
↪type
unique_overdose_types = deathrates.
↪groupby('overdose_type')['overdose_type_num'].unique().reset_index()
unique_overdose_types['overdose_type_num'] =␣
↪unique_overdose_types['overdose_type_num'].apply(lambda x: x[0] if len(x) >␣
↪0 else None)

print(unique_overdose_types)
```

```
             overdose_type  overdose_type_num
0            all overdoses                  0
1               any opioid                  1
```

```
2                    heroin                   5
3                 methadone                   3
4   natural or semisynthetic                  2
5    non-methadone synthetic                  4
```

[32]:

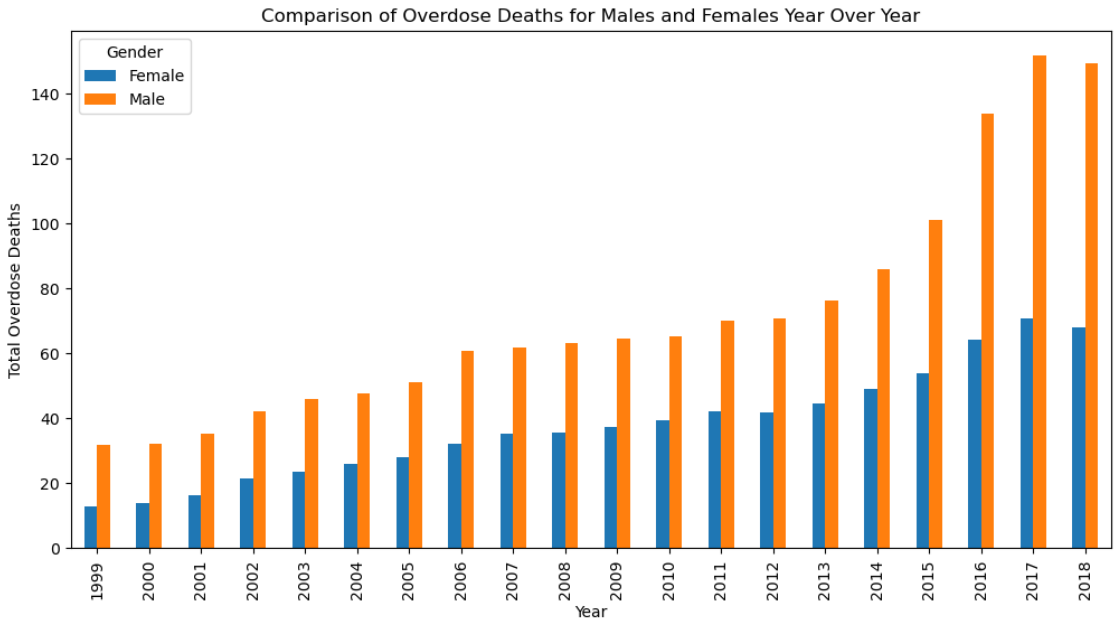
```
# grouping by 'demographic_detail' and getting unique 'demographic_detail_num'
 ↪for each detail
unique_demographic_details = deathrates.
 ↪groupby('demographic_detail')['demographic_detail_num'].unique().
 ↪reset_index()
unique_demographic_details['demographic_detail_num'] =
 ↪unique_demographic_details['demographic_detail_num'].apply(lambda x: x[0] if
 ↪len(x) > 0 else None)

print(unique_demographic_details)
```

```
                                   demographic_detail  demographic_detail_num
0                                         15-24 years                    1.20
1                                         25-34 years                    1.30
2                                         35-44 years                    1.40
3                                         45-54 years                    1.50
4                                         55-64 years                    1.60
5                                         65-74 years                    1.70
6                                         75-84 years                    1.80
7                                   85 years and over                    1.90
8                                         All persons                    0.10
9                                              Female                    2.20
10                                Female: 15-24 years                    3.22
11                                Female: 25-34 years                    3.23
12                                Female: 35-44 years                    3.24
13                                Female: 45-54 years                    3.25
14                                Female: 55-64 years                    3.26
15                                Female: 65-74 years                    3.27
16                                Female: 75-84 years                    3.28
17                          Female: 85 years and over                    3.29
18          Female: American Indian or Alaska Native                    4.70
19                  Female: Asian or Pacific Islander                    4.80
20                  Female: Black or African American                    4.60
21              Female: Hispanic or Latino: All races                    5.60
22  Female: Not Hispanic or Latino: American India…                   5.90
23              Female: Not Hispanic or Latino: Asian                    5.92
24  Female: Not Hispanic or Latino: Asian or Pacif…                   5.91
25              Female: Not Hispanic or Latino: Black                    5.80
26  Female: Not Hispanic or Latino: Native Hawaiia…                   5.93
27              Female: Not Hispanic or Latino: White                    5.70
28                             Female: Under 15 years                    3.21
29                                      Female: White                    4.50
30                                               Male                    2.10
```

```
31                              Male: 15-24 years              3.12
32                              Male: 25-34 years              3.13
33                              Male: 35-44 years              3.14
34                              Male: 45-54 years              3.15
35                              Male: 55-64 years              3.16
36                              Male: 65-74 years              3.17
37                              Male: 75-84 years              3.18
38                        Male: 85 years and over              3.19
39            Male: American Indian or Alaska Native           4.30
40                 Male: Asian or Pacific Islander             4.40
41                 Male: Black or African American             4.20
42              Male: Hispanic or Latino: All races            5.10
43  Male: Not Hispanic or Latino: American Indian …          5.40
44              Male: Not Hispanic or Latino: Asian            5.50
45  Male: Not Hispanic or Latino: Asian or Pacific…          5.50
46              Male: Not Hispanic or Latino: Black            5.30
47  Male: Not Hispanic or Latino: Native Hawaiian …          5.60
48              Male: Not Hispanic or Latino: White            5.20
49                           Male: Under 15 years              3.11
50                                    Male: White              4.10
51                                 Under 15 years              1.10
```

One Last Chart Once my data for overdose death rates had been cleaned, I wanted to test it with a quick visualization. I built a comparison plot to show Male and Female rates year over year for the entire duration of the dataset (1999 - 2017). Here was the result.

[33]:
```python
# filtering the dataframe to include only 'Male' and 'Female' rows
male_female_data = deathrates[deathrates['demographic_detail'].isin(['Male',␣
↪'Female'])]

# grouping the data by 'year' and 'demographic_detail', then summing the␣
↪'estimate' column
grouped_data = male_female_data.groupby(['year',␣
↪'demographic_detail'])['estimate'].sum().unstack()

# plotting the comparison
grouped_data.plot(kind='bar', figsize=(12, 6))
plt.title('Comparison of Overdose Deaths for Males and Females Year Over Year')
plt.xlabel('Year')
plt.ylabel('Total Overdose Deaths')
plt.legend(title='Gender')
plt.show()
```



I did have three more files I still needed to examine and clean, but I believe I've completed at least five adjustments to the dataframes here.
I will continue to explore, prep, and test the dataframes between now and the next milestone, and will keep that code for review/understanding later.
I also plan to review the other datasets I found and decide if there are gaps I want/need to fill in order to find stronger insights.

0.0.6 Milestone Three: Cleaning and Formatting Website Data

The website table I selected was from wikipedia and called, "United States Drug Overdose Death Rates and Totals over Time".
I am least familiar with website data as a source, so I tried to stick to the scope of the project and not 'over-reach' too much, in spite of how interesting I may find the data, just in case I ran into difficulties with it.

[34]:
```
# website my tables of interest are hosted on
url = "https://en.wikipedia.org/wiki/
 ↪United_States_drug_overdose_death_rates_and_totals_over_time"

response = requests.get(url)
soup = BeautifulSoup(response.content, 'html.parser')
tables = soup.find_all('table', {'class': 'wikitable'})

# extracting the first table -- US Drug Overdose Deaths by Year
table = tables[0]
rows = table.find_all('tr')
data = []
```

```
for row in rows:
    cols = row.find_all(['td', 'th'])
    cols = [ele.text.strip() for ele in cols]
    data.append(cols)

wiki_od_deaths = pd.DataFrame(data)
wiki_od_deaths.columns = wiki_od_deaths.iloc[0]  # setting the first row as the
 ↪column names
wiki_od_deaths = wiki_od_deaths.drop(0)  # dropping the first row as it's now
 ↪the header

print(wiki_od_deaths)
```

```
0    Year     Deaths Population(July 1 residents) Crude rate Age adjusted rate
1
2    1968      5,033                  199,533,564        2.5               2.8
3    1969      6,006                  201,568,206        3.0               3.3
4    1970      7,101                  203,458,035        3.5               3.8
5    1971      6,771                  206,782,970        3.3               3.5
6    1972      6,622                  209,237,411        3.2               3.4
7    1973      6,413                  211,361,965        3.0               3.2
8    1974      6,449                  213,436,958        3.0               3.2
9    1975      7,145                  215,457,198        3.3               3.4
10   1976      6,765                  217,615,788        3.1               3.2
11   1977      6,130                  219,808,632        2.8               2.9
12   1978      5,506                  222,102,279        2.5               2.6
13   1979      2,544                  224,635,398        1.1               1.1
14   1980      2,492                  226,624,371        1.1               1.1
15   1981      2,668                  229,487,512        1.2               1.2
16   1982      2,862                  231,701,425        1.2               1.2
17   1983      2,866                  233,781,743        1.2               1.2
18   1984      3,266                  235,922,142        1.4               1.3
19   1985      3,612                  238,005,715        1.5               1.5
20   1986      4,187                  240,189,882        1.7               1.7
21   1987      3,907                  242,395,034        1.6               1.6
22   1988      4,865                  244,651,961        2.0               2.0
23   1989      5,035                  247,001,762        2.0               2.0
24   1990      4,506                  248,922,111        1.8               1.8
25   1991      5,215                  253,088,068        2.1               2.0
26   1992      5,951                  256,606,463        2.3               2.3
27   1993      7,382                  260,024,637        2.8               2.8
28   1994      7,828                  263,241,475        3.0               3.0
29   1995      8,000                  266,386,596        3.0               3.0
30   1996      8,431                  269,540,779        3.1               3.1
31   1997      9,099                  272,776,678        3.3               3.3
32   1998      9,838                  276,032,848        3.6               3.6
```

```
33   1999     16,849                   279,040,168        6.0                 6.1
34   2000     17,415                   281,421,906        6.2                 6.2
35   2001     19,394                   284,968,955        6.8                 6.8
36   2002     23,518                   287,625,193        8.2                 8.2
37   2003     25,785                   290,107,933        8.9                 8.9
38   2004     27,424                   292,805,298        9.4                 9.4
39   2005     29,813                   295,516,599       10.1                10.1
40   2006     34,425                   298,379,912       11.5                11.5
41   2007     36,010                   301,231,207       12.0                11.9
42   2008     36,450                   304,093,966       12.0                11.9
43   2009     37,004                   306,771,529       12.1                11.9
44   2010     38,329                   308,745,538       12.4                12.3
45   2011     41,340                   311,591,917       13.3                13.2
46   2012     41,502                   313,914,040       13.2                13.1
47   2013     43,982                   316,128,839       13.9                13.8
48   2014     47,055                   318,857,056       14.8                14.7
49   2015     52,404                   321,418,820       16.3                16.3
50   2016     63,632                   323,127,513       19.7                19.8
51   2017     70,237                   325,719,178       21.6                21.7
52   2018     67,367                   327,167,434       20.6                20.7
53   2019     70,630                   328,239,523       21.5                21.6
54   2020     91,799                   329,484,123       27.9                28.3
55  Total  1,106,859
```

[35]:
```python
# extracting the second table -- Drug Overdose Death Rates per 100,000
 ↪Population by State (1999-2021)
table = tables[1]
rows = table.find_all('tr')
data = []

for row in rows:
    cols = row.find_all(['td', 'th'])
    cols = [ele.text.strip() for ele in cols]
    data.append(cols)

wiki_od_states = pd.DataFrame(data)
wiki_od_states.columns = wiki_od_states.iloc[0]  # setting the first row as the
 ↪column names
wiki_od_states = wiki_od_states.drop(0)  # dropping the first row as it's now
 ↪the header

print(wiki_od_states)
```

```
0               State  1999  2005  2014  2015  2016  2017  2018  2019  2020  \
1
2           Alabama *   3.9   6.3  15.2  15.7  16.2    18  16.6  16.3  22.3
3            Alaska *   7.5  11.4  16.8    16  16.8  20.2  14.6  17.8    22
4             Arizona  10.6  14.1  12.6    19  20.3  22.2  23.8  26.8  35.8
```

5	Arkansas *	4.4	10.1	18.2	13.8	14	15.5	15.7	13.5	19.1
6	California *	8.1	9	11.1	11.3	11.2	11.7	12.8	15	21.8
7	Colorado *	8	12.7	16.3	15.4	16.6	17.6	16.8	18	24.9
8	Connecticut	9	8.5	17.6	22.1	27.4	30.9	30.7	34.7	39.1
9	Delaware	6.4	7.5	20.9	22	30.8	37	43.8	48	47.3
10	Florida *	6.4	13.5	13.2	16.2	23.7	25.1	22.8	25.5	35
11	Georgia *	3.5	8.2	11.9	12.7	13.3	14.7	13.2	13.1	18
12	Hawaii *	6.5	9.4	10.9	11.3	12.8	13.8	14.3	15.9	18.3
13	Idaho *	5.3	8.1	13.7	14.2	15.2	14.4	14.6	15.1	15.9
14	Illinois	6.7	8.4	13.1	14.1	18.9	21.6	21.3	21.9	28.1
15	Indiana	3.2	9.8	18.2	19.5	24	29.4	25.6	26.6	36.7
16	Iowa	1.9	4.8	8.8	10.3	10.6	11.5	9.6	11.5	14.3
17	Kansas	3.4	9.1	11.7	11.8	11.1	11.8	12.4	14.3	17.4
18	Kentucky	4.9	15.3	24.7	29.9	33.5	37.2	30.9	32.5	49.2
19	Louisiana	4.3	14.7	16.9	19	21.8	24.5	25.4	28.3	42.7
20	Maine	5.3	12.4	16.8	21.2	28.7	34.4	27.9	29.9	39.7
21	Maryland *	11.4	11.4	17.4	20.9	33.2	36.3	37.2	38.2	44.6
22	Massachusetts *	7.5	12	19	25.7	33	31.8	32.8	32.1	33.9
23	Michigan	4.6	9.8	18	20.4	24.4	27.8	26.6	24.4	28.6
24	Minnesota *	2.8	5.4	9.6	10.6	12.5	13.3	11.5	14.2	19
25	Mississippi *	3.2	8.8	11.6	12.3	12.1	12.2	10.8	13.6	21.1
26	Missouri	5	10.7	18.2	17.9	23.6	23.4	27.5	26.9	32.1
27	Montana *	4.6	10.1	12.4	13.8	11.7	11.7	12.2	14.1	15.6
28	Nebraska	2.3	5	7.2	6.9	6.4	8.1	7.4	8.7	11.3
29	Nevada	11.5	18.7	18.4	20.4	21.7	21.6	21.2	20.1	26
30	New Hampshire	4.3	10.7	26.2	34.3	39	37	35.8	32	30.3
31	New Jersey	6.5	9.4	14	16.3	23.2	30	33.1	31.7	32.1
32	New Mexico	15	20.1	27.3	25.3	25.2	24.8	26.7	30.2	39
33	New York *	5	4.8	11.3	13.6	18	19.4	18.4	18.2	25.4
34	North Carolina *	4.6	11.4	13.8	15.8	19.7	24.1	22.4	22.3	30.9
35	North Dakota *	0	0	6.3	8.6	10.6	9.2	10.2	11.4	15.6
36	Ohio *	4.2	10.9	24.6	29.9	39.1	46.3	35.9	38.3	47.2
37	Oklahoma *	5.4	13.8	20.3	19	21.5	20.1	18.4	16.7	19.4
38	Oregon *	6.1	10.4	12.8	12	11.9	12.4	12.6	14	18.7
39	Pennsylvania	8.1	13.2	21.9	26.3	37.9	44.3	36.1	35.6	42.4
40	Rhode Island	5.5	14.3	23.4	28.2	30.8	31	30.1	29.5	38.2
41	South Carolina *	3.7	9.9	14.4	15.7	18.1	20.5	22.6	22.7	34.9
42	South Dakota	0	5.5	7.8	8.4	8.4	8.5	6.9	10.5	10.3
43	Tennessee	6.1	14.5	19.5	22.2	24.5	26.6	27.5	31.2	45.6
44	Texas *	5.4	8.5	9.7	9.4	10.1	10.5	10.4	10.8	14.1
45	Utah *	10.6	19.3	22.4	23.4	22.4	22.3	21.2	18.9	20.5
46	Vermont *	4.7	8.5	13.9	16.7	22.2	23.2	26.6	23.8	32.9
47	Virginia *	5	7.5	11.7	12.4	16.7	17.9	17.1	18.3	26.6
48	Washington *	9.3	13	13.3	14.7	14.5	15.2	14.8	15.8	22
49	West Virginia *	4.1	10.5	35.5	41.5	52	57.8	51.5	52.8	81.4
50	Wisconsin	4	9.3	15.1	15.5	19.3	21.2	19.2	21.1	27.7
51	Wyoming	4.1	4.9	19.4	16.4	17.6	12.2	11.1	14.1	17.4

0	2021
1	
2	30.1
3	35.6
4	38.7
5	22.3
6	26.6
7	31.4
8	42.3
9	54
10	37.5
11	23.5
12	17.3
13	19
14	29
15	43
16	15.3
17	24.3
18	55.6
19	55.9
20	47.1
21	42.8
22	36.8
23	31.5
24	24.5
25	28.4
26	36.5
27	19.5
28	11.4
29	29.2
30	32.3
31	32.4
32	51.6
33	28.7
34	39.2
35	17.2
36	48.1
37	24.4
38	26.8
39	43.2
40	41.7
41	42.8
42	12.6
43	56.6
44	16.8
45	21.1
46	42.3
47	30.5

```
48   28.1
49   90.9
50   31.6
51   18.9
```

[36]:

```python
# extracting the fifth table on the page -- Drug Overdose Deaths by State Over␣
↪Time (1999-2021)
table = tables[4]
rows = table.find_all('tr')
data = []

for row in rows:
    cols = row.find_all(['td', 'th'])
    cols = [ele.text.strip() for ele in cols]
    data.append(cols)

wiki_od_states_raw = pd.DataFrame(data)
wiki_od_states_raw.columns = wiki_od_states_raw.iloc[0]  # setting the first␣
↪row as the column names
wiki_od_states_raw = wiki_od_states_raw.drop(0)  # dropping the first row as␣
↪it's now the header

print(wiki_od_states_raw)
```

```
0             State   1999   2005   2014   2015   2016   2017   2018   2019  \
1
2           Alabama    169    283    723    736    756    835    775    768
3            Alaska     46     79    124    122    128    147    110    132
4           Arizona    511    794    356  1,274  1,382  1,532  1,670  1,907
5          Arkansas    113    269  1,211    392    401    446    444    388
6        California  2,662  3,214  4,521  4,659  4,654  4,868  5,348  6,198
7          Colorado    349    608    899    869    942  1,015    995  1,079
8       Connecticut    310    295    623    800    971  1,072  1,069  1,214
9          Delaware     50     62    189    198    282    338    401    435
10          Florida    997  2,371  2,634  3,228  4,728  5,088  4,698  5,268
11          Georgia    283    738  1,206  1,302  1,394  1,537  1,404  1,408
12           Hawaii     80    126    157    169    191    203    213    242
13            Idaho     64    109    212    218    243    236    250    265
14         Illinois    825  1,067  1,705  1,835  2,411  2,778  2,722  2,790
15          Indiana    191    610  1,172  1,245  1,526  1,852  1,629  1,699
16             Iowa     53    141    264    309    314    341    287    352
17           Kansas     89    241    332    329    313    333    345    403
18         Kentucky    197    638  1,077  1,273  1,419  1,566  1,315  1,380
19        Louisiana    188    661    777    861    996  1,108  1,140  1,267
20            Maine     67    163    216    269    353    424    345    371
21         Maryland    629    656  1,070  1,285  2,044  2,247  2,324  2,369
22    Massachusetts    488    780  1,289  1,724  2,227  2,168  2,241  2,210
23         Michigan    460    985  1,762  1,980  2,347  2,694  2,591  2,385
```

24	Minnesota	136	282	517	581	672	733	636	792
25	Mississippi	87	248	336	351	352	354	310	394
26	Missouri	276	608	1,067	1,066	1,371	1,367	1,610	1,583
27	Montana	41	96	125	138	119	119	125	143
28	Nebraska	39	86	125	126	120	152	138	161
29	Nevada	227	457	545	619	665	676	688	647
30	New Hampshire	54	142	334	422	481	467	452	407
31	New Jersey	557	823	1,253	1,454	2,056	2,685	2,900	2,805
32	New Mexico	266	373	547	501	500	493	537	599
33	New York	959	944	2,300	2,754	3,638	3,921	3,697	3,617
34	North Carolina	366	1,000	1,358	1,567	1,956	2,414	2,259	2,266
35	North Dakota	12	12	43	61	77	68	70	82
36	Ohio	467	1,243	2,744	3,310	4,329	5,111	3,980	4,251
37	Oklahoma	178	478	777	725	813	775	716	645
38	Oregon	210	386	522	505	506	530	547	615
39	Pennsylvania	990	1,613	2,732	3,264	4,627	5,388	4,415	4,377
40	Rhode Island	58	156	247	310	326	320	317	307
41	South Carolina	147	427	701	761	879	1,008	1,125	1,127
42	South Dakota	17	40	63	65	69	73	57	86
43	Tennessee	344	872	1,269	1,457	1,630	1,776	1,823	2,089
44	Texas	1,087	1,910	2,601	2,588	2,831	2,989	3,005	3,136
45	Utah	205	438	603	646	635	650	624	571
46	Vermont	29	53	83	99	125	134	153	133
47	Virginia	366	581	980	1,039	1,405	1,507	1,448	1,547
48	Washington	555	850	979	1,094	1,102	1,169	1,164	1,259
49	West Virginia	75	184	627	725	884	974	856	870
50	Wisconsin	212	518	853	878	1,074	1,177	1,079	1,201
51	Wyoming	20	26	109	96	99	69	66	79

0	2020	2021
1		
2	1,029	1,408
3	160	260
4	2,550	2,730
5	546	637
6	8,908	10,901
7	1,492	1,887
8	1,371	1,552
9	444	513
10	7,231	7,827
11	1,916	2,500
12	274	269
13	287	354
14	3,549	3,762
15	2,321	2,811
16	432	475
17	490	680
18	2,083	2,381

```
19  1,896   2,463
20    496     611
21  2,771   2,737
22  2,302   2,585
23  2,759   3,089
24  1,050   1,356
25    586     787
26  1,875   2,155
27    162     199
28    214     214
29    832     949
30    393     441
31  2,840   3,056
32    784   1,052
33  4,965   5,842
34  3,146   3,981
35    114     124
36  5,204   5,397
37    762     960
38    803   1,171
39  5,168   5,449
40    397     455
41  1,739   2,138
42     83     105
43  3,034   3,813
44  4,172   4,984
45    622     662
46    190     252
47  2,240   2,626
48  1,733   2,264
49  1,330   1,501
50  1,531   1,775
51     99     109
```

The first table didn't have too many issues, but I did want to re-name the headers and drop the total row and the first blank row.

[37]:
```python
# dropping the last row (which holds the total)
wiki_od_deaths = wiki_od_deaths.drop(wiki_od_deaths.index[-1])

# renaming the columns to align with the prior naming convention
new_column_names = {
    'Year': 'year',
    'Deaths': 'deaths',
    'Population(July 1 residents)': 'population_count',
    'Crude rate': 'crude_rate',
    'Age adjusted rate': 'age_adjusted_rate'}
wiki_od_deaths = wiki_od_deaths.rename(columns=new_column_names)
```

```python
print(wiki_od_deaths)
```

```
0   year  deaths population_count crude_rate age_adjusted_rate
1
2   1968   5,033      199,533,564        2.5               2.8
3   1969   6,006      201,568,206        3.0               3.3
4   1970   7,101      203,458,035        3.5               3.8
5   1971   6,771      206,782,970        3.3               3.5
6   1972   6,622      209,237,411        3.2               3.4
7   1973   6,413      211,361,965        3.0               3.2
8   1974   6,449      213,436,958        3.0               3.2
9   1975   7,145      215,457,198        3.3               3.4
10  1976   6,765      217,615,788        3.1               3.2
11  1977   6,130      219,808,632        2.8               2.9
12  1978   5,506      222,102,279        2.5               2.6
13  1979   2,544      224,635,398        1.1               1.1
14  1980   2,492      226,624,371        1.1               1.1
15  1981   2,668      229,487,512        1.2               1.2
16  1982   2,862      231,701,425        1.2               1.2
17  1983   2,866      233,781,743        1.2               1.2
18  1984   3,266      235,922,142        1.4               1.3
19  1985   3,612      238,005,715        1.5               1.5
20  1986   4,187      240,189,882        1.7               1.7
21  1987   3,907      242,395,034        1.6               1.6
22  1988   4,865      244,651,961        2.0               2.0
23  1989   5,035      247,001,762        2.0               2.0
24  1990   4,506      248,922,111        1.8               1.8
25  1991   5,215      253,088,068        2.1               2.0
26  1992   5,951      256,606,463        2.3               2.3
27  1993   7,382      260,024,637        2.8               2.8
28  1994   7,828      263,241,475        3.0               3.0
29  1995   8,000      266,386,596        3.0               3.0
30  1996   8,431      269,540,779        3.1               3.1
31  1997   9,099      272,776,678        3.3               3.3
32  1998   9,838      276,032,848        3.6               3.6
33  1999  16,849      279,040,168        6.0               6.1
34  2000  17,415      281,421,906        6.2               6.2
35  2001  19,394      284,968,955        6.8               6.8
36  2002  23,518      287,625,193        8.2               8.2
37  2003  25,785      290,107,933        8.9               8.9
38  2004  27,424      292,805,298        9.4               9.4
39  2005  29,813      295,516,599       10.1              10.1
40  2006  34,425      298,379,912       11.5              11.5
41  2007  36,010      301,231,207       12.0              11.9
42  2008  36,450      304,093,966       12.0              11.9
43  2009  37,004      306,771,529       12.1              11.9
44  2010  38,329      308,745,538       12.4              12.3
```

```
45  2011  41,340     311,591,917       13.3              13.2
46  2012  41,502     313,914,040       13.2              13.1
47  2013  43,982     316,128,839       13.9              13.8
48  2014  47,055     318,857,056       14.8              14.7
49  2015  52,404     321,418,820       16.3              16.3
50  2016  63,632     323,127,513       19.7              19.8
51  2017  70,237     325,719,178       21.6              21.7
52  2018  67,367     327,167,434       20.6              20.7
53  2019  70,630     328,239,523       21.5              21.6
54  2020  91,799     329,484,123       27.9              28.3
```

The second table needed the header "State" renamed to follow my lowercase naming convention, but I also noticed that the values in that column often had asterisks that would ultimately prevent those values from matching a list of State names in any other column, so I used .strip to remove them and any excess spaces, just in case.

[38]:
```python
# renaming the 'State' column to 'state'
wiki_od_states = wiki_od_states.rename(columns={'State': 'state'})

# removing asterisks and extra spaces from the state names
wiki_od_states['state'] = wiki_od_states['state'].str.replace('*', '').str.
 ↪strip()

# printing df
print(wiki_od_states)
```

```
0             state  1999  2005  2014  2015  2016  2017  2018  2019  2020  2021
1
2           Alabama   3.9   6.3  15.2  15.7  16.2    18  16.6  16.3  22.3  30.1
3            Alaska   7.5  11.4  16.8    16  16.8  20.2  14.6  17.8    22  35.6
4           Arizona  10.6  14.1  12.6    19  20.3  22.2  23.8  26.8  35.8  38.7
5          Arkansas   4.4  10.1  18.2  13.8    14  15.5  15.7  13.5  19.1  22.3
6        California   8.1     9  11.1  11.3  11.2  11.7  12.8    15  21.8  26.6
7          Colorado     8  12.7  16.3  15.4  16.6  17.6  16.8    18  24.9  31.4
8       Connecticut     9   8.5  17.6  22.1  27.4  30.9  30.7  34.7  39.1  42.3
9          Delaware   6.4   7.5  20.9    22  30.8    37  43.8    48  47.3    54
10          Florida   6.4  13.5  13.2  16.2  23.7  25.1  22.8  25.5    35  37.5
11          Georgia   3.5   8.2  11.9  12.7  13.3  14.7  13.2  13.1    18  23.5
12           Hawaii   6.5   9.4  10.9  11.3  12.8  13.8  14.3  15.9  18.3  17.3
13            Idaho   5.3   8.1  13.7  14.2  15.2  14.4  14.6  15.1  15.9    19
14         Illinois   6.7   8.4  13.1  14.1  18.9  21.6  21.3  21.9  28.1    29
15          Indiana   3.2   9.8  18.2  19.5    24  29.4  25.6  26.6  36.7    43
16             Iowa   1.9   4.8   8.8  10.3  10.6  11.5   9.6  11.5  14.3  15.3
17           Kansas   3.4   9.1  11.7  11.8  11.1  11.8  12.4  14.3  17.4  24.3
18         Kentucky   4.9  15.3  24.7  29.9  33.5  37.2  30.9  32.5  49.2  55.6
19        Louisiana   4.3  14.7  16.9    19  21.8  24.5  25.4  28.3  42.7  55.9
20            Maine   5.3  12.4  16.8  21.2  28.7  34.4  27.9  29.9  39.7  47.1
21         Maryland  11.4  11.4  17.4  20.9  33.2  36.3  37.2  38.2  44.6  42.8
```

22	Massachusetts	7.5	12	19	25.7	33	31.8	32.8	32.1	33.9	36.8
23	Michigan	4.6	9.8	18	20.4	24.4	27.8	26.6	24.4	28.6	31.5
24	Minnesota	2.8	5.4	9.6	10.6	12.5	13.3	11.5	14.2	19	24.5
25	Mississippi	3.2	8.8	11.6	12.3	12.1	12.2	10.8	13.6	21.1	28.4
26	Missouri	5	10.7	18.2	17.9	23.6	23.4	27.5	26.9	32.1	36.5
27	Montana	4.6	10.1	12.4	13.8	11.7	11.7	12.2	14.1	15.6	19.5
28	Nebraska	2.3	5	7.2	6.9	6.4	8.1	7.4	8.7	11.3	11.4
29	Nevada	11.5	18.7	18.4	20.4	21.7	21.6	21.2	20.1	26	29.2
30	New Hampshire	4.3	10.7	26.2	34.3	39	37	35.8	32	30.3	32.3
31	New Jersey	6.5	9.4	14	16.3	23.2	30	33.1	31.7	32.1	32.4
32	New Mexico	15	20.1	27.3	25.3	25.2	24.8	26.7	30.2	39	51.6
33	New York	5	4.8	11.3	13.6	18	19.4	18.4	18.2	25.4	28.7
34	North Carolina	4.6	11.4	13.8	15.8	19.7	24.1	22.4	22.3	30.9	39.2
35	North Dakota	0	0	6.3	8.6	10.6	9.2	10.2	11.4	15.6	17.2
36	Ohio	4.2	10.9	24.6	29.9	39.1	46.3	35.9	38.3	47.2	48.1
37	Oklahoma	5.4	13.8	20.3	19	21.5	20.1	18.4	16.7	19.4	24.4
38	Oregon	6.1	10.4	12.8	12	11.9	12.4	12.6	14	18.7	26.8
39	Pennsylvania	8.1	13.2	21.9	26.3	37.9	44.3	36.1	35.6	42.4	43.2
40	Rhode Island	5.5	14.3	23.4	28.2	30.8	31	30.1	29.5	38.2	41.7
41	South Carolina	3.7	9.9	14.4	15.7	18.1	20.5	22.6	22.7	34.9	42.8
42	South Dakota	0	5.5	7.8	8.4	8.4	8.5	6.9	10.5	10.3	12.6
43	Tennessee	6.1	14.5	19.5	22.2	24.5	26.6	27.5	31.2	45.6	56.6
44	Texas	5.4	8.5	9.7	9.4	10.1	10.5	10.4	10.8	14.1	16.8
45	Utah	10.6	19.3	22.4	23.4	22.4	22.3	21.2	18.9	20.5	21.1
46	Vermont	4.7	8.5	13.9	16.7	22.2	23.2	26.6	23.8	32.9	42.3
47	Virginia	5	7.5	11.7	12.4	16.7	17.9	17.1	18.3	26.6	30.5
48	Washington	9.3	13	13.3	14.7	14.5	15.2	14.8	15.8	22	28.1
49	West Virginia	4.1	10.5	35.5	41.5	52	57.8	51.5	52.8	81.4	90.9
50	Wisconsin	4	9.3	15.1	15.5	19.3	21.2	19.2	21.1	27.7	31.6
51	Wyoming	4.1	4.9	19.4	16.4	17.6	12.2	11.1	14.1	17.4	18.9

The final dataframe also needed the header changed to the lowercase naming convention and then I would examine for further cleanup across all the website data.

0.0.7 One More Website

[39]:
```
# requesting one more table from another wiki page
url = "https://en.wikipedia.org/wiki/2020_United_States_census"

response = requests.get(url)
soup = BeautifulSoup(response.content, 'html.parser')
tables = soup.find_all('table', {'class': 'wikitable'})

table = tables[0]  # pulling the state-by-state population data for 2020
rows = table.find_all('tr')
data = []

for row in rows:
```

```python
    cols = row.find_all(['td', 'th'])
    cols = [ele.text.strip() for ele in cols]
    data.append(cols)

population_2020 = pd.DataFrame(data)
population_2020.columns = population_2020.iloc[0]  # setting the first row as␣
 ↪the column names
population_2020 = population_2020.drop(0)  # dropping the first row as it's now␣
 ↪the header

print(population_2020)
```

```
0  Rank/change State  Population(2020)[85] Population(2010)[86]        Change  \
1            1                  California           39,538,223   37,253,956
2            2                       Texas           29,145,505   25,145,561
3            3     1               Florida           21,538,187   18,801,310
4            4     1              New York           20,201,249   19,378,102
5            5     1          Pennsylvania           13,002,700   12,702,379
6            6     1              Illinois           12,812,508   12,830,632
7            7                        Ohio           11,799,448   11,536,504
8            8     1               Georgia           10,711,908    9,687,653
9            9     1        North Carolina           10,439,388    9,535,483
10          10     2              Michigan           10,077,331    9,883,640
11          11                  New Jersey            9,288,994    8,791,894
12          12                    Virginia            8,631,393    8,001,024
13          13                  Washington            7,705,281    6,724,540
14          14     2               Arizona            7,151,502    6,392,017
15          15     1         Massachusetts            7,029,917    6,547,629
16          16     1             Tennessee            6,910,840    6,346,105
17          17     2               Indiana            6,785,528    6,483,802
18          18     1              Maryland            6,177,224    5,773,552
19          19     1              Missouri            6,154,913    5,988,927
20          20                   Wisconsin            5,893,718    5,686,986
21          21     1              Colorado            5,773,714    5,029,196
22          22     1             Minnesota            5,706,494    5,303,925
23          23     1        South Carolina            5,118,425    4,625,364
24          24     1               Alabama            5,024,279    4,779,736
25          25                   Louisiana            4,657,757    4,533,372
26          26                    Kentucky            4,505,836    4,339,367
27          27                      Oregon            4,237,256    3,831,074
28          28                    Oklahoma            3,959,353    3,751,351
29          29                 Connecticut            3,605,944    3,574,097
30          30     4                  Utah            3,271,616    2,763,885
31          31     1                  Iowa            3,190,369    3,046,355
32          32     3                Nevada            3,104,614    2,700,551
33          33     1              Arkansas            3,011,524    2,915,918
34          34     3           Mississippi            2,961,279    2,967,297
```

35	35	2	Kansas	2,937,880	2,853,118
36	36		New Mexico	2,117,522	2,059,179
37	37	1	Nebraska	1,961,504	1,826,341
38	38	1	Idaho	1,839,106	1,567,582
39	39	2	West Virginia	1,793,716	1,852,994
40	40		Hawaii	1,455,271	1,360,301
41	41	1	New Hampshire	1,377,529	1,316,470
42	42	1	Maine	1,362,359	1,328,361
43	43		Rhode Island	1,097,379	1,052,567
44	44		Montana	1,084,225	989,415
45	45		Delaware	989,948	897,934
46	46		South Dakota	886,667	814,180
47	47	1	North Dakota	779,094	672,591
48	48	1	Alaska	733,391	710,231
49	-	-	District of Columbia	689,545	601,723
50	49		Vermont	643,077	625,741
51	50		Wyoming	576,851	563,626
52			United States	331,449,281	308,745,538

0	%change	None
1	2,284,267	6.1%
2	3,999,944	15.9%
3	2,736,877	14.6%
4	823,147	4.3%
5	300,321	2.4%
6	−18,124	−0.1%
7	262,944	2.3%
8	1,024,255	10.6%
9	903,905	9.5%
10	193,691	2.0%
11	497,100	5.7%
12	630,369	7.9%
13	980,741	14.6%
14	759,485	11.9%
15	482,288	7.4%
16	564,735	8.9%
17	301,726	4.6%
18	403,672	7.0%
19	165,986	2.8%
20	206,732	3.6%
21	744,518	14.8%
22	402,569	7.6%
23	493,061	10.7%
24	244,543	5.1%
25	124,385	2.7%
26	166,469	3.8%
27	406,182	10.6%
28	208,002	5.5%

```
29      31,847   0.9%
30     507,731  18.4%
31     144,014   4.7%
32     404,063  15.0%
33      95,606   3.3%
34      -6,018  -0.2%
35      84,762   3.0%
36      58,343   2.8%
37     135,163   7.4%
38     271,524  17.3%
39     -59,278  -3.2%
40      94,970   7.0%
41      61,059   4.6%
42      33,998   2.6%
43      44,812   4.3%
44      94,810   9.6%
45      92,014  10.3%
46      72,487   8.9%
47     106,503  15.8%
48      23,160   3.3%
49      87,822  14.6%
50      17,336   2.8%
51      13,225   2.4%
52  22,703,743   7.4%
```

These columns came into the dataframe slightly incorrect. After a few iterations of exploration, I was able to save the data I actually wanted, which was the state names and the 2020 population data, and ensure they were named correctly.

[40]:
```
# dropping the unwanted columns
population_2020 = population_2020.drop(columns=['Rank/change', 'State',␣
↪'Change', '%change'])

# renaming the remaining columns
population_2020 = population_2020.rename(columns={
    'Population(2020)[85]': 'state',
    'Population(2010)[86]': 'population'})

# printing the df
print(population_2020.head())
```

```
0         state  population   None
1    California  39,538,223   6.1%
2         Texas  29,145,505  15.9%
3       Florida  21,538,187  14.6%
4      New York  20,201,249   4.3%
5  Pennsylvania  13,002,700   2.4%
```

[41]:
```python
# keeping only the 'state' and 'population' columns
population_2020 = population_2020[['state', 'population']]

# printing the df
print(population_2020.head())
```

```
0          state  population
1     California  39,538,223
2          Texas  29,145,505
3        Florida  21,538,187
4       New York  20,201,249
5   Pennsylvania  13,002,700
```

[42]:
```python
# converting the population values to numeric so I could use them in␣
↪calculations later
population_2020['population'] = population_2020['population'].str.replace(',',␣
↪'').astype(int)
```

0.0.8 One More Graph Tonight...

As an example of combined data insights, in my first flat file, I was able to perform counts of opioid providers by state, but all we found in that exploration was that more highly populated states had more providers.
I now have state population counts from the same year as the provider file data.

This is my attempt at aligning the state values data to use as a key so I can instead show population-adjusted opioid provider counts:

[43]:
```python
unique_states = opioidproviders['state'].unique()
print("Unique state abbreviations in opioidproviders:")
print(unique_states)

unique_pop_states = population_2020['state'].unique()
print("Unique states in population_20202:")
print(unique_pop_states)
```

```
Unique state abbreviations in opioidproviders:
['VT' 'WI' 'VA' 'MD' 'FL' 'OH' 'NJ' 'TX' 'GA' 'CA' 'CT' 'IN' 'WA' 'PA'
 'NY' 'CO' 'MI' 'AZ' 'MA' 'IL' 'KY' 'MN' 'MT' 'NC' 'AL' 'DE' 'RI' 'NE'
 'TN' 'NH' 'MO' 'LA' 'ME' 'AR' 'OK' 'SC' 'UT' 'WV' 'MS' 'NM' 'AK' 'NV'
 'OR' 'KS' 'HI' 'ND' 'IA' 'DC' 'ID' 'SD']
Unique states in population_20202:
['California' 'Texas' 'Florida' 'New York' 'Pennsylvania' 'Illinois'
 'Ohio' 'Georgia' 'North Carolina' 'Michigan' 'New Jersey' 'Virginia'
 'Washington' 'Arizona' 'Massachusetts' 'Tennessee' 'Indiana' 'Maryland'
 'Missouri' 'Wisconsin' 'Colorado' 'Minnesota' 'South Carolina' 'Alabama'
 'Louisiana' 'Kentucky' 'Oregon' 'Oklahoma' 'Connecticut' 'Utah' 'Iowa'
 'Nevada' 'Arkansas' 'Mississippi' 'Kansas' 'New Mexico' 'Nebraska'
 'Idaho' 'West Virginia' 'Hawaii' 'New Hampshire' 'Maine' 'Rhode Island'
```

```
 'Montana' 'Delaware' 'South Dakota' 'North Dakota' 'Alaska'
 'District of Columbia' 'Vermont' 'Wyoming' 'United States']
```

I attemped to map the state initials to the full state names to align these two columns so I could use them as a key.
I ran into a couple of difficulties when the two lists didn't have the same values – I am dropping the 'United States' row, which serves as a total in the population_2020 dataframe, and then will sort them both alphabetically to ensure my code captures all it needs to.

[44]:
```
# removing the 'United States' row from the population_2020 DataFrame
population_2020 = population_2020[population_2020['state'] != 'United States']
```

[45]:
```
# printing unique state abbreviations in opioidproviders in alphabetical order
unique_states = sorted(opioidproviders['state'].unique())
print("Unique state abbreviations in opioidproviders:")
print(unique_states)

# printing unique states in population_2020 in alphabetical order
unique_pop_states = sorted(population_2020['state'].unique())
print("Unique states in population_2020:")
print(unique_pop_states)
```

```
Unique state abbreviations in opioidproviders:
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'IA',
'ID', 'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO', 'MS',
'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM', 'NV', 'NY', 'OH', 'OK', 'OR', 'PA',
'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV']
Unique states in population_2020:
['Alabama', 'Alaska', 'Arizona', 'Arkansas', 'California', 'Colorado',
'Connecticut', 'Delaware', 'District of Columbia', 'Florida', 'Georgia',
'Hawaii', 'Idaho', 'Illinois', 'Indiana', 'Iowa', 'Kansas', 'Kentucky',
'Louisiana', 'Maine', 'Maryland', 'Massachusetts', 'Michigan', 'Minnesota',
'Mississippi', 'Missouri', 'Montana', 'Nebraska', 'Nevada', 'New Hampshire',
'New Jersey', 'New Mexico', 'New York', 'North Carolina', 'North Dakota',
'Ohio', 'Oklahoma', 'Oregon', 'Pennsylvania', 'Rhode Island', 'South Carolina',
'South Dakota', 'Tennessee', 'Texas', 'Utah', 'Vermont', 'Virginia',
'Washington', 'West Virginia', 'Wisconsin', 'Wyoming']
```

[46]:
```
# mapping state abbreviations to full names
state_mapping = {
    'AL': 'Alabama', 'AK': 'Alaska', 'AZ': 'Arizona', 'AR': 'Arkansas', 'CA':␣
 ↪'California',
    'CO': 'Colorado', 'CT': 'Connecticut', 'DC': 'District of Columbia', 'DE':␣
 ↪'Delaware', 'FL': 'Florida', 'GA': 'Georgia',
    'HI': 'Hawaii', 'ID': 'Idaho', 'IL': 'Illinois', 'IN': 'Indiana', 'IA':␣
 ↪'Iowa',
    'KS': 'Kansas', 'KY': 'Kentucky', 'LA': 'Louisiana', 'ME': 'Maine', 'MD':␣
 ↪'Maryland',
```

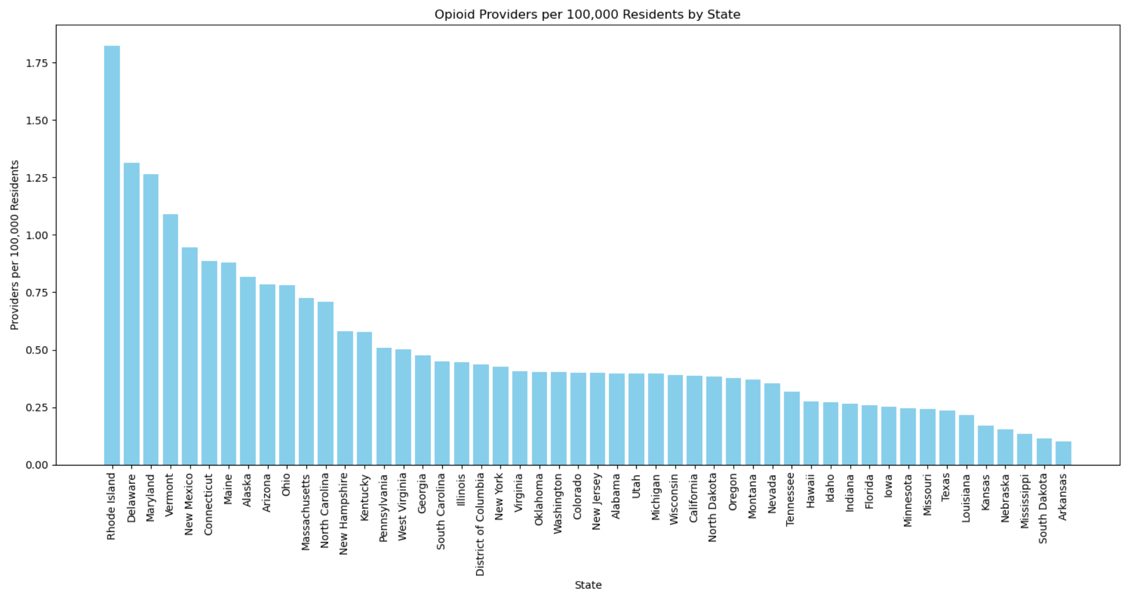
```
    'MA': 'Massachusetts', 'MI': 'Michigan', 'MN': 'Minnesota', 'MS':␣
↪'Mississippi', 'MO': 'Missouri',
    'MT': 'Montana', 'NE': 'Nebraska', 'NV': 'Nevada', 'NH': 'New Hampshire',␣
↪'NJ': 'New Jersey',
    'NM': 'New Mexico', 'NY': 'New York', 'NC': 'North Carolina', 'ND': 'North␣
↪Dakota', 'OH': 'Ohio',
    'OK': 'Oklahoma', 'OR': 'Oregon', 'PA': 'Pennsylvania', 'RI': 'Rhode␣
↪Island', 'SC': 'South Carolina',
    'SD': 'South Dakota', 'TN': 'Tennessee', 'TX': 'Texas', 'UT': 'Utah', 'VT':␣
↪'Vermont',
    'VA': 'Virginia', 'WA': 'Washington', 'WV': 'West Virginia', 'WI':␣
↪'Wisconsin', 'WY': 'Wyoming'}

# converting state abbreviations in opioidproviders to full names
opioidproviders['state'] = opioidproviders['state'].map(state_mapping)
```

[47]:
```
# rechecking values
unique_states = opioidproviders['state'].unique()
print("Unique state abbreviations in opioidproviders:")
print(unique_states)

unique_pop_states = population_2020['state'].unique()
print("Unique states in population_20202:")
print(unique_pop_states)
```

```
Unique state abbreviations in opioidproviders:
['Vermont' 'Wisconsin' 'Virginia' 'Maryland' 'Florida' 'Ohio' 'New Jersey'
 'Texas' 'Georgia' 'California' 'Connecticut' 'Indiana' 'Washington'
 'Pennsylvania' 'New York' 'Colorado' 'Michigan' 'Arizona' 'Massachusetts'
 'Illinois' 'Kentucky' 'Minnesota' 'Montana' 'North Carolina' 'Alabama'
 'Delaware' 'Rhode Island' 'Nebraska' 'Tennessee' 'New Hampshire'
 'Missouri' 'Louisiana' 'Maine' 'Arkansas' 'Oklahoma' 'South Carolina'
 'Utah' 'West Virginia' 'Mississippi' 'New Mexico' 'Alaska' 'Nevada'
 'Oregon' 'Kansas' 'Hawaii' 'North Dakota' 'Iowa' 'District of Columbia'
 'Idaho' 'South Dakota']
Unique states in population_20202:
['California' 'Texas' 'Florida' 'New York' 'Pennsylvania' 'Illinois'
 'Ohio' 'Georgia' 'North Carolina' 'Michigan' 'New Jersey' 'Virginia'
 'Washington' 'Arizona' 'Massachusetts' 'Tennessee' 'Indiana' 'Maryland'
 'Missouri' 'Wisconsin' 'Colorado' 'Minnesota' 'South Carolina' 'Alabama'
 'Louisiana' 'Kentucky' 'Oregon' 'Oklahoma' 'Connecticut' 'Utah' 'Iowa'
 'Nevada' 'Arkansas' 'Mississippi' 'Kansas' 'New Mexico' 'Nebraska'
 'Idaho' 'West Virginia' 'Hawaii' 'New Hampshire' 'Maine' 'Rhode Island'
 'Montana' 'Delaware' 'South Dakota' 'North Dakota' 'Alaska'
 'District of Columbia' 'Vermont' 'Wyoming']
```

Now I am ready to group/count by state, merge the dataframes, handle missing provider counts (Wyoming has no opioid providers in the dataset), and calculate the amount per 100,000 population so we can see the population-adjusted graph!

[48]:
```
# grouping opioidproviders by state and doing the count
provider_counts = opioidproviders['state'].value_counts()

# moving provider counts to a df
provider_counts_df = provider_counts.reset_index()
provider_counts_df.columns = ['state', 'provider_count']

# merging the dataframes on state
merged_data = pd.merge(provider_counts_df, population_2020, on='state',␣
 ↪how='left')

# filling NaN values in provider_count with 0
merged_data['provider_count'] = merged_data['provider_count'].fillna(0)

# calculate the number of providers per 100,000 people
merged_data['providers_per_100k'] = (merged_data['provider_count'] /␣
 ↪merged_data['population']) * 100000

# plotting the graph
plt.figure(figsize=(15, 8))
merged_data.sort_values('providers_per_100k', ascending=False, inplace=True)
plt.bar(merged_data['state'], merged_data['providers_per_100k'],␣
 ↪color='skyblue')
plt.title('Opioid Providers per 100,000 Residents by State')
plt.xlabel('State')
plt.ylabel('Providers per 100,000 Residents')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



0.0.9 List of Data Transformations Made Above

Quite a lot of the website data I brought into my new dataframes was quite clean, but it still required some massaging to make it usable for my needs. Here is a list of data transformations I made during this milestone:

1. Using the top row to create column headers and then dropping the first row.

2. Dropping the last (total) row from the first dataframe.
3. Adjusting header titles to match my previous lowercase naming convention in the first dataframe.
4. Using .strip to remove excess spaces and characters in values of state names.
5. Using .drop to remove unwanted columns or rows from the population table and adjusting headers to correct columns.
6. Keeping only the first two columns as a way to drop the last unnamed column that Pandas displays as 'None', but cannot be used to drop using that as a header.
7. Aligning the formatting of values in the 'state' column of the opioidproviders df to match the capital case full state names used in other dataframes.
8. Converting the population values to numeric to allow for calculations (and other dataframes will still need this done later).
9. Merging dataframes using 'state' as a key after the values were aligned.

0.0.10 Some Thoughts and Ethical Concerns

Since the website data has been loaded, read, and cleaned, I wanted to circle back to the other three dataframes I created from the flat files and get those cleaned as well. I am really hoping the data comes together in the way I expected to gather some interesting insights.
Before I moved onto that, however, I noticed I hadn't addressed the ethical concerns with using or cleaning this kind of data.

There are some legislative documents in place that help protect our healthcare information, and of course, any data relating to public health or especially PHI has to be treated with the utmost care and consideration of the real human beings behind the thousands of data points. I've worked in HIPAA legislation for a number of years and it has been my experience that many healthcare organizations still struggle to understand enough of the HIPAA Security Rule to make more than a 'good faith' effort at protecting our data. This means much of our healthcare data is in the hands of the electronic health record vendors, who do tend to take it more seriously as they all have nationwide Business Associate Agreements with healthcare organizations, and thus, could also be fined for negligence and failure to protect our healthcare data.

Opinions vary about how protected U.S. healthcare data actually is, however, when you back up and view the larger, global and digital picture.

When considering my own ethical concerns, so far the most sensitive data I have here is practice NPI. This means with the right granularity of data, we do have the potential to identify and highlight prescribing patterns of individual practices, some of which may be better at following current 'best practices' regarding opioid prescribing than others. The truth is that clinical best practices regarding opioid usage has fundamentally shifted over the past 30-50 years and I am certain we have providers in practice who have contributed to the opioid crisis, without realizing the damage those earlier practices may have caused.

I suspect I am not going to reveal anything here in this project that SureScripts, Epic, or even the Walgreens analysts don't already know.
Regardless, the most important thing is to always self-check the ethics of any changes made and to always remember that even in "counts" of de-identified healthcare data, we are still talking about the lives of real human beings, patients and providers, so the data as well as the information and insights discovered should always be treated with the utmost care.

0.0.11 Next Steps

I have reviewed the additional flat files from our prior milestone that I didn't get to in the assignment and I'm finding they may not have much in the way of additional information, so I may remove them from my final version. I am excited to connect the APIs I found, however, as those records seem to be the most robust source of prescribing behavior data.
The challenge I will be faced with, I think, is pulling this all together in ways that allow us to gain insights as a result of the combined data.

0.0.12 Milestone Four: - Connecting to an API & Cleaning/Formatting

My first API is quite large, and contains information on prescription drugs provided to Medicare beneficiaries enrolled in Part D coverage by physicians. The documentation says it has 25.5 million rows of data across 22 columns, so my strategy for my initial data exploration here will be to use a small subset of the data to understand the structure, missing values, and try to plan my larger cleaning and transformation steps.

Connecting to the API – Iteration One This is an API call for 2017 data only (the full dataset has nine years of data), and through some initial exploration, I decided on these specific columns and will do some clean-up on the full dataset once loaded. I discovered that the dataset was quite large, and loading/reading 2 million rows through an API was a different animal than doing the same with a flat file.

I did the first request for only the first 100 records.
It is also looking like I'll need to convert the state abbreviations to full state names again, which was a little challenging to do with my website data, so I'm hoping I can simply re-use some of that prior code.

[49]:
```
# API endpoint for 2017 data only
url = "https://data.cms.gov/data-api/v1/dataset/
 ↪04b93a42-c533-4e5c-8df9-a8f254886cde/data"

# detailing columns of interest
columns = [
    "Prscrbr_NPI",
    "Prscrbr_Last_Org_Name",
    "Prscrbr_First_Name",
    "Prscrbr_Type",
    "Prscrbr_Type_Src",
    "Prscrbr_State_Abrvtn",
    "Gnrc_Name",
    "Brnd_Name",
    "Tot_Clms",
    "Tot_30day_Fills",
    "Tot_Day_Suply",
    "Tot_Drug_Cst",
    "Tot_Benes"]

# columns need formatting as a comma-separated list, per API documentation
columns_param = ",".join(columns)

# parameters for the API request as the total database is 25.5m rows --␣
 ↪mitigation against memory issues
params = {
    "size": 100,  # number of records per request
    "offset": 0,  # start at the beginning of the dataset
    "column": columns_param,}  # formatted as a comma-separated list

# making the API request
response = requests.get(url, params=params)

# checking if the request was successful
if response.status_code == 200:
    data = response.json()         # Converting the response to JSON
    part_d = pd.DataFrame(data)    # converting the JSON data to a df

    # printing the first few rows of the df
    print(part_d.head())
else:
    print(f"Failed to fetch data. Status code: {response.status_code}")
```

```
  Prscrbr_NPI Prscrbr_Last_Org_Name Prscrbr_First_Name       Prscrbr_Type  \
0  1003000126             Enkeshafi            Ardalan  Internal Medicine
1  1003000126             Enkeshafi            Ardalan  Internal Medicine
2  1003000126             Enkeshafi            Ardalan  Internal Medicine
3  1003000126             Enkeshafi            Ardalan  Internal Medicine
4  1003000126             Enkeshafi            Ardalan  Internal Medicine

  Prscrbr_Type_Src Prscrbr_State_Abrvtn             Gnrc_Name  \
0                S                   MD   Amlodipine Besylate
1                S                   MD  Atorvastatin Calcium
2                S                   MD            Cephalexin
3                S                   MD     Ciprofloxacin Hcl
4                S                   MD   Doxycycline Hyclate

              Brnd_Name Tot_Clms Tot_30day_Fills Tot_Day_Suply Tot_Drug_Cst  \
0   Amlodipine Besylate       13              13           390        59.21
1  Atorvastatin Calcium       27              27           765       259.48
2            Cephalexin       17              17           123        98.99
3     Ciprofloxacin Hcl       12              12            95       120.43
4   Doxycycline Hyclate       17              17           105       300.76

  Tot_Benes
0
1        11
2        17
3        11
4        17
```

[50]:
```python
print(part_d.shape)
```

```
(100, 13)
```

[51]:
```python
# mapping state abbreviations to full name in capital case
part_d['Prscrbr_State_Abrvtn'] = part_d['Prscrbr_State_Abrvtn'].
 ↪map(state_mapping)
```

[52]:
```python
print(part_d.head())
```

```
  Prscrbr_NPI Prscrbr_Last_Org_Name Prscrbr_First_Name       Prscrbr_Type  \
0  1003000126             Enkeshafi            Ardalan  Internal Medicine
1  1003000126             Enkeshafi            Ardalan  Internal Medicine
2  1003000126             Enkeshafi            Ardalan  Internal Medicine
3  1003000126             Enkeshafi            Ardalan  Internal Medicine
4  1003000126             Enkeshafi            Ardalan  Internal Medicine

  Prscrbr_Type_Src Prscrbr_State_Abrvtn             Gnrc_Name  \
0                S             Maryland   Amlodipine Besylate
1                S             Maryland  Atorvastatin Calcium
2                S             Maryland            Cephalexin
```

```
3                 S             Maryland     Ciprofloxacin Hcl
4                 S             Maryland   Doxycycline Hyclate

              Brnd_Name Tot_Clms Tot_30day_Fills Tot_Day_Suply Tot_Drug_Cst  \
0   Amlodipine Besylate       13              13           390        59.21
1  Atorvastatin Calcium       27              27           765       259.48
2            Cephalexin       17              17           123        98.99
3     Ciprofloxacin Hcl       12              12            95       120.43
4   Doxycycline Hyclate       17              17           105       300.76

  Tot_Benes
0
1        11
2        17
3        11
4        17
```

Now that I'm confident about doing some of this clean-up, I'll take another iteration of requesting the data through the API, only this time I'll get all of it in chunks. The API documentation allows for requests of 5000 rows at a time, so we'll iterate through until we have all of the 2017 Medicare Part D data.

[ ]:
```python
# defining the function to fetch data in 4,000 row batches
def fetch_data(offset, size):
    url = "https://data.cms.gov/data-api/v1/dataset/
 ↪04b93a42-c533-4e5c-8df9-a8f254886cde/data"  # API endpoint for 2017 only
    columns = [
        "Prscrbr_NPI",
        "Prscrbr_Last_Org_Name",
        "Prscrbr_First_Name",
        "Prscrbr_Type",
        "Prscrbr_Type_Src",
        "Prscrbr_State_Abrvtn",
        "Gnrc_Name",
        "Brnd_Name",
        "Tot_Clms",
        "Tot_30day_Fills",
        "Tot_Day_Suply",
        "Tot_Drug_Cst",
        "Tot_Benes"]
    params = {
        "size": size,
        "offset": offset,
        "column": ",".join(columns),}
    response = requests.get(url, params=params)
    return response

# initializing variables for paging
```

```
size = 4000  # Adjusted to be below the max size allowed
offset = 0
all_data = []  # List to store all fetched data
start_time = datetime.now()

# loop to fetch data until complete
while True:
    response = fetch_data(offset, size)
    # printing progress immediately after fetching data
    print(f"Fetched {size} rows starting from offset {offset}.")
    if response.status_code == 200:
        data = response.json()
        if not data:  # break the loop if no data is returned
            break
        all_data.extend(data)
        print(f"Successfully fetched batch starting at offset {offset}. Total␣
↪rows fetched: {len(all_data)}")
        offset += size  # Prepare for the next batch
    else:
        print(f"Failed to fetch data at offset {offset}. Status code: {response.
↪status_code}")
        break  # exit loop when there's a failure

end_time = datetime.now()
duration = end_time - start_time
print(f"Completed. Duration: {duration}. Total rows fetched: {len(all_data)}")

# converting the collected data to a df
medicare_partd = pd.DataFrame(all_data)     # this line gave a memory error
```

0.0.13 The Down Sides of Being "Data-Greedy"

The code worked beautifully to retrieve all the data and provide me with regular updates as the retrieval iterations worked. In total, the data took nearly 15 hours to retrieve to complete and it fetched a total of 25,209,130 rows.
Panda dataframes didn't have the memory to convert the data into a single df, so I tried a variety of approaches and learned a LOT while trying to work with this large dataset.

Some of the attempts included:
Dumping the data into a JSON file as a backup and learning/exploring the use of Dask to clean the data before moving it to SQLite.
Dask still gave me memory errors, so I tried moving the data from the JSON to parquet files.
I reduced the chunksize multiple times in the hopes of managing the memory issue, and eventually started looking not to the packages, but my own hardware.
My system has great internet speeds and the CPU had no issues, but I was reaching the limit of my system's memory.
I looked up ways to extend the memory using things like external storage and other workarounds, but these solutions were starting to feel excessive.

I took a day to re-consider my approach and realized there was really no reason to be processing/cleaning ALL of the data, when my project only focused on opioid management. I had started to become interested in medications prescribed to patients 65+ that were highly discouraged by the American Geriatric Society, and opioids can be one of them, depending on the need and length of use.
Essentially, I had to step away from the project to realize I was falling into my own mental scope creep.

I deleted the JSON, re-wrote my API request, and this time, filtered the request to only pull rows related to opioids.

0.0.14 API Successes

Due to the size of the initial request, I was a little concerned that even with my medication filter, I would run into size and memory issues.
I wrote a test-request to pull the first 100 rows of each medication match, just to see if it would be manageable.

[53]:
```
def fetch_data_filtered_by_medication(medication, size=100):
    base_url = "https://data.cms.gov/data-api/v1/dataset/
↪04b93a42-c533-4e5c-8df9-a8f254886cde/data"
    params = {
        "filter[Gnrc_Name]": medication,
        "size": size,  # fetching only 100 rows that match the medication
        "offset": 0}       # starting at the beginning of the dataset
    response = requests.get(base_url, params=params)
    return response

medications = [
    "fentanyl", "methadone hydrochloride", "morphine sulfate",
    "oxymorphone hydrochloride", "hydrocodone", "oxycodone",
    "codeine", "morphine", "tapentadol", "methadone",
    "buprenorphine", "meperidine", "isonipecaine",
    "pethidine", "dihydromorphinone"]

all_data_filter = []

for medication in medications:
    response = fetch_data_filtered_by_medication(medication)
    if response.status_code == 200:
        data = response.json()
        all_data_filter.extend(data)
        print(f"Data fetched for medication: {medication}")
    else:
        print(f"Failed to fetch data for {medication}. Status code: {response.
↪status_code}")

# checking how many records were fetched
```

```python
print(f"Total records fetched: {len(all_data_filter)}")
```

```
Data fetched for medication: fentanyl
Data fetched for medication: methadone hydrochloride
Data fetched for medication: morphine sulfate
Data fetched for medication: oxymorphone hydrochloride
Data fetched for medication: hydrocodone
Data fetched for medication: oxycodone
Data fetched for medication: codeine
Data fetched for medication: morphine
Data fetched for medication: tapentadol
Data fetched for medication: methadone
Data fetched for medication: buprenorphine
Data fetched for medication: meperidine
Data fetched for medication: isonipecaine
Data fetched for medication: pethidine
Data fetched for medication: dihydromorphinone
Total records fetched: 300
```

I then did some early exploration of the data from a Pandas DF so I was *sure* I knew what I might be bringing in before initiating the full, filtered API request.

[54]:
```python
# converting the list of dictionaries into a Pandas DF
test_meds = pd.DataFrame(all_data_filter)

# displaying the first few rows, the structure/summary
print(test_meds.head())
print(test_meds.info())
```

```
  Prscrbr_NPI Prscrbr_Last_Org_Name Prscrbr_First_Name Prscrbr_City  \
0  1003000142                Khalil             Rashid       Toledo
1  1003000407               Girardi              David   Brookville
2  1003000530              Semonche             Amanda   Quakertown
3  1003001363               Stevens            Charles    El Centro
4  1003002312               Hopkins           Patricia       Quincy

  Prscrbr_State_Abrvtn Prscrbr_State_FIPS       Prscrbr_Type Prscrbr_Type_Src  \
0                   OH                 39     Anesthesiology                S
1                   PA                 42    Family Practice                S
2                   PA                 42  Internal Medicine                S
3                   CA                 06     Anesthesiology                S
4                   MA                 25       Rheumatology                S

  Brnd_Name Gnrc_Name  … Tot_Day_Suply Tot_Drug_Cst Tot_Benes  \
0  Fentanyl  Fentanyl  …          2495      6981.67        19
1  Fentanyl  Fentanyl  …           232      1067.07
2  Fentanyl  Fentanyl  …           630      1147.95
3  Fentanyl  Fentanyl  …          1965      8565.05        37
4  Fentanyl  Fentanyl  …           740      2221.23
```

```
  GE65_Sprsn_Flag GE65_Tot_Clms GE65_Tot_30day_Fills GE65_Tot_Drug_Cst  \
0                            26                 26.1           2201.03
1                            14                   14           1067.07
2               *
3                            34                   34           3489.36
4                            13                   13           1747.77

  GE65_Tot_Day_Suply GE65_Bene_Sprsn_Flag GE65_Tot_Benes
0                769                    #
1                232                    *
2                                       *
3               1005                                  20
4                390                    *

[5 rows x 22 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300 entries, 0 to 299
Data columns (total 22 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Prscrbr_NPI            300 non-null    object
 1   Prscrbr_Last_Org_Name  300 non-null    object
 2   Prscrbr_First_Name     300 non-null    object
 3   Prscrbr_City           300 non-null    object
 4   Prscrbr_State_Abrvtn   300 non-null    object
 5   Prscrbr_State_FIPS     300 non-null    object
 6   Prscrbr_Type           300 non-null    object
 7   Prscrbr_Type_Src       300 non-null    object
 8   Brnd_Name              300 non-null    object
 9   Gnrc_Name              300 non-null    object
 10  Tot_Clms               300 non-null    object
 11  Tot_30day_Fills        300 non-null    object
 12  Tot_Day_Suply          300 non-null    object
 13  Tot_Drug_Cst           300 non-null    object
 14  Tot_Benes              300 non-null    object
 15  GE65_Sprsn_Flag        300 non-null    object
 16  GE65_Tot_Clms          300 non-null    object
 17  GE65_Tot_30day_Fills   300 non-null    object
 18  GE65_Tot_Drug_Cst      300 non-null    object
 19  GE65_Tot_Day_Suply     300 non-null    object
 20  GE65_Bene_Sprsn_Flag   300 non-null    object
 21  GE65_Tot_Benes         300 non-null    object
dtypes: object(22)
memory usage: 51.7+ KB
None
```

[55]:
```
# a quick check to make sure the code ran as expected:
print(test_meds['Gnrc_Name'].value_counts())
```

```
Gnrc_Name
Fentanyl            100
Morphine Sulfate    100
Buprenorphine       100
Name: count, dtype: int64
```

[57]:
```
# I even converted to excel to just take a look at it all -- clearly, I was␣
↪being cautious.
test_meds.to_excel(r'C:\Users\alyse\OneDrive\Documents\Bellevue University\DSC␣
↪540 - Data Preparation\test_meds.xlsx', index=False, engine='openpyxl')
```

0.0.15 The Winning API Request

I finally felt confident enough to write and run the new, filtered API request.
It retrieved 136K rows, and this was significantly easier to work with in Pandas.

[58]:
```
def fetch_data_opioids(medication, size, offset):
    base_url = "https://data.cms.gov/data-api/v1/dataset/
↪04b93a42-c533-4e5c-8df9-a8f254886cde/data"
    params = {
        "filter[Gnrc_Name]": medication,
        "size": size,
        "offset": offset}
    response = requests.get(base_url, params=params)
    return response

medications = [
    "fentanyl", "methadone hydrochloride", "morphine sulfate",
    "oxymorphone hydrochloride", "hydrocodone", "oxycodone",
    "codeine", "morphine", "tapentadol", "methadone",
    "buprenorphine", "meperidine", "isonipecaine",
    "pethidine", "dihydromorphinone"]

all_data_filter = []

for medication in medications:
    print(f"Fetching data for medication: {medication}")
    offset = 0
    while True:
        response = fetch_data_opioids(medication, 4000, offset)
        if response.status_code == 200:
            data = response.json()
            if not data:  # breaking the loop if no data is returned
                break
            all_data_filter.extend(data)
```

```
            print(f"Successfully fetched batch for {medication} starting at␣
↪offset {offset}. Total rows fetched so far for all medications:␣
↪{len(all_data_filter)}")
            offset += 4000  # grabbing the next batch from the last line␣
↪retrieved
        else:
            print(f"Failed to fetch data for {medication} at offset {offset}.␣
↪Status code: {response.status_code}")
            break  # exiting loop when there's a failure

print(f"Total records fetched for all medications: {len(all_data_filter)}")
```

```
Fetching data for medication: fentanyl
Successfully fetched batch for fentanyl starting at offset 0. Total rows fetched
so far for all medications: 4000
Successfully fetched batch for fentanyl starting at offset 4000. Total rows
fetched so far for all medications: 8000
Successfully fetched batch for fentanyl starting at offset 8000. Total rows
fetched so far for all medications: 12000
Successfully fetched batch for fentanyl starting at offset 12000. Total rows
fetched so far for all medications: 16000
Successfully fetched batch for fentanyl starting at offset 16000. Total rows
fetched so far for all medications: 20000
Successfully fetched batch for fentanyl starting at offset 20000. Total rows
fetched so far for all medications: 24000
Successfully fetched batch for fentanyl starting at offset 24000. Total rows
fetched so far for all medications: 28000
Successfully fetched batch for fentanyl starting at offset 28000. Total rows
fetched so far for all medications: 32000
Successfully fetched batch for fentanyl starting at offset 32000. Total rows
fetched so far for all medications: 36000
Successfully fetched batch for fentanyl starting at offset 36000. Total rows
fetched so far for all medications: 40000
Successfully fetched batch for fentanyl starting at offset 40000. Total rows
fetched so far for all medications: 44000
Successfully fetched batch for fentanyl starting at offset 44000. Total rows
fetched so far for all medications: 48000
Successfully fetched batch for fentanyl starting at offset 48000. Total rows
fetched so far for all medications: 52000
Successfully fetched batch for fentanyl starting at offset 52000. Total rows
fetched so far for all medications: 55215
Fetching data for medication: methadone hydrochloride
Fetching data for medication: morphine sulfate
Successfully fetched batch for morphine sulfate starting at offset 0. Total rows
fetched so far for all medications: 59215
Successfully fetched batch for morphine sulfate starting at offset 4000. Total
rows fetched so far for all medications: 63215
```

```
Successfully fetched batch for morphine sulfate starting at offset 8000. Total
rows fetched so far for all medications: 67215
Successfully fetched batch for morphine sulfate starting at offset 12000. Total
rows fetched so far for all medications: 71215
Successfully fetched batch for morphine sulfate starting at offset 16000. Total
rows fetched so far for all medications: 75215
Successfully fetched batch for morphine sulfate starting at offset 20000. Total
rows fetched so far for all medications: 79215
Successfully fetched batch for morphine sulfate starting at offset 24000. Total
rows fetched so far for all medications: 83215
Successfully fetched batch for morphine sulfate starting at offset 28000. Total
rows fetched so far for all medications: 87215
Successfully fetched batch for morphine sulfate starting at offset 32000. Total
rows fetched so far for all medications: 91215
Successfully fetched batch for morphine sulfate starting at offset 36000. Total
rows fetched so far for all medications: 95215
Successfully fetched batch for morphine sulfate starting at offset 40000. Total
rows fetched so far for all medications: 99215
Successfully fetched batch for morphine sulfate starting at offset 44000. Total
rows fetched so far for all medications: 103215
Successfully fetched batch for morphine sulfate starting at offset 48000. Total
rows fetched so far for all medications: 107215
Successfully fetched batch for morphine sulfate starting at offset 52000. Total
rows fetched so far for all medications: 111215
Successfully fetched batch for morphine sulfate starting at offset 56000. Total
rows fetched so far for all medications: 115215
Successfully fetched batch for morphine sulfate starting at offset 60000. Total
rows fetched so far for all medications: 119215
Successfully fetched batch for morphine sulfate starting at offset 64000. Total
rows fetched so far for all medications: 123215
Successfully fetched batch for morphine sulfate starting at offset 68000. Total
rows fetched so far for all medications: 127215
Successfully fetched batch for morphine sulfate starting at offset 72000. Total
rows fetched so far for all medications: 130416
Fetching data for medication: oxymorphone hydrochloride
Fetching data for medication: hydrocodone
Fetching data for medication: oxycodone
Fetching data for medication: codeine
Fetching data for medication: morphine
Fetching data for medication: tapentadol
Fetching data for medication: methadone
Fetching data for medication: buprenorphine
Successfully fetched batch for buprenorphine starting at offset 0. Total rows
fetched so far for all medications: 134416
Successfully fetched batch for buprenorphine starting at offset 4000. Total rows
fetched so far for all medications: 136784
Fetching data for medication: meperidine
Fetching data for medication: isonipecaine
```

```
Fetching data for medication: pethidine
Fetching data for medication: dihydromorphinone
Total records fetched for all medications: 136784
```

0.0.16 Cleaning the API Data

[60]:
```python
# converting the list of dictionaries into a Pandas DF
opioid_meds = pd.DataFrame(all_data_filter)

# saving the data in Excel as a backup
excel_backup_path = r'C:\Users\alyse\OneDrive\Documents\Bellevue University\DSC␣
↪540 - Data Preparation\opioid_meds_backup.xlsx'
opioid_meds.to_excel(excel_backup_path, index=False)
```

0.0.17 Reloading the DFs for Milestone 5

[56]:
```python
# defining the path to my Excel backup so I don't have to re-download through␣
↪the API again
excel_backup_path = 'C:/Users/alyse/OneDrive/Documents/Bellevue University/DSC␣
↪540 - Data Preparation/opioid_meds_backup.xlsx'

# loading the data into a dataframe
opioid_meds = pd.read_excel(excel_backup_path)

# confirming the load
print(opioid_meds.head())
```

```
   Prscrbr_NPI Prscrbr_Last_Org_Name Prscrbr_First_Name Prscrbr_City  \
0   1003000142                Khalil             Rashid       Toledo
1   1003000407               Girardi              David   Brookville
2   1003000530              Semonche             Amanda   Quakertown
3   1003001363               Stevens            Charles    El Centro
4   1003002312               Hopkins           Patricia       Quincy

  Prscrbr_State_Abrvtn Prscrbr_State_FIPS       Prscrbr_Type Prscrbr_Type_Src  \
0                   OH                 39     Anesthesiology                S
1                   PA                 42    Family Practice                S
2                   PA                 42  Internal Medicine                S
3                   CA                 06     Anesthesiology                S
4                   MA                 25       Rheumatology                S

  Brnd_Name Gnrc_Name  …  Tot_Day_Suply  Tot_Drug_Cst  Tot_Benes  \
0  Fentanyl  Fentanyl  …           2495       6981.67       19.0
1  Fentanyl  Fentanyl  …            232       1067.07        NaN
2  Fentanyl  Fentanyl  …            630       1147.95        NaN
3  Fentanyl  Fentanyl  …           1965       8565.05       37.0
4  Fentanyl  Fentanyl  …            740       2221.23        NaN
```

```
   GE65_Sprsn_Flag  GE65_Tot_Clms GE65_Tot_30day_Fills  GE65_Tot_Drug_Cst  \
0              NaN           26.0                 26.1            2201.03
1              NaN           14.0                 14.0            1067.07
2                *            NaN                  NaN                NaN
3              NaN           34.0                 34.0            3489.36
4              NaN           13.0                 13.0            1747.77

   GE65_Tot_Day_Suply  GE65_Bene_Sprsn_Flag  GE65_Tot_Benes
0               769.0                     #             NaN
1               232.0                     *             NaN
2                 NaN                     *             NaN
3              1005.0                   NaN            20.0
4               390.0                     *             NaN

[5 rows x 22 columns]
```

[57]:
```python
# displaying the first few rows, the structure/summary of the df
print(opioid_meds.head())
print(opioid_meds.info())
```

```
   Prscrbr_NPI Prscrbr_Last_Org_Name Prscrbr_First_Name Prscrbr_City  \
0   1003000142                Khalil             Rashid       Toledo
1   1003000407               Girardi              David   Brookville
2   1003000530              Semonche             Amanda   Quakertown
3   1003001363               Stevens            Charles    El Centro
4   1003002312               Hopkins           Patricia       Quincy

  Prscrbr_State_Abrvtn Prscrbr_State_FIPS       Prscrbr_Type Prscrbr_Type_Src  \
0                   OH                 39     Anesthesiology                S
1                   PA                 42    Family Practice                S
2                   PA                 42  Internal Medicine                S
3                   CA                 06     Anesthesiology                S
4                   MA                 25       Rheumatology                S

  Brnd_Name Gnrc_Name  …  Tot_Day_Suply  Tot_Drug_Cst  Tot_Benes  \
0  Fentanyl  Fentanyl  …           2495       6981.67       19.0
1  Fentanyl  Fentanyl  …            232       1067.07        NaN
2  Fentanyl  Fentanyl  …            630       1147.95        NaN
3  Fentanyl  Fentanyl  …           1965       8565.05       37.0
4  Fentanyl  Fentanyl  …            740       2221.23        NaN

   GE65_Sprsn_Flag  GE65_Tot_Clms GE65_Tot_30day_Fills  GE65_Tot_Drug_Cst  \
0              NaN           26.0                 26.1            2201.03
1              NaN           14.0                 14.0            1067.07
2                *            NaN                  NaN                NaN
3              NaN           34.0                 34.0            3489.36
4              NaN           13.0                 13.0            1747.77
```

```
   GE65_Tot_Day_Suply  GE65_Bene_Sprsn_Flag  GE65_Tot_Benes
0               769.0                     #             NaN
1               232.0                     *             NaN
2                 NaN                     *             NaN
3              1005.0                   NaN            20.0
4               390.0                     *             NaN

[5 rows x 22 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 136784 entries, 0 to 136783
Data columns (total 22 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   Prscrbr_NPI           136784 non-null  int64
 1   Prscrbr_Last_Org_Name 136784 non-null  object
 2   Prscrbr_First_Name    136784 non-null  object
 3   Prscrbr_City          136784 non-null  object
 4   Prscrbr_State_Abrvtn  136784 non-null  object
 5   Prscrbr_State_FIPS    136784 non-null  object
 6   Prscrbr_Type          136784 non-null  object
 7   Prscrbr_Type_Src      136784 non-null  object
 8   Brnd_Name             136784 non-null  object
 9   Gnrc_Name             136784 non-null  object
 10  Tot_Clms              136784 non-null  int64
 11  Tot_30day_Fills       136784 non-null  float64
 12  Tot_Day_Suply         136784 non-null  int64
 13  Tot_Drug_Cst          136784 non-null  float64
 14  Tot_Benes             29105 non-null   float64
 15  GE65_Sprsn_Flag       53403 non-null   object
 16  GE65_Tot_Clms         83381 non-null   float64
 17  GE65_Tot_30day_Fills  83381 non-null   float64
 18  GE65_Tot_Drug_Cst     83381 non-null   float64
 19  GE65_Tot_Day_Suply    83381 non-null   float64
 20  GE65_Bene_Sprsn_Flag  128262 non-null  object
 21  GE65_Tot_Benes        8522 non-null    float64
dtypes: float64(8), int64(3), object(11)
memory usage: 23.0+ MB
None
```

[58]:

```python
# columns to be dropped as they aren't relevant to my project, but I included␣
↪them initially as I wanted to be sure.
columns_to_drop = [
    'Prscrbr_State_FIPS',
    'Prscrbr_Type_Src',
    'GE65_Sprsn_Flag',
    'GE65_Tot_Clms',
    'GE65_Tot_30day_Fills',
```

```
    'GE65_Tot_Drug_Cst',
    'GE65_Tot_Day_Suply',
    'GE65_Bene_Sprsn_Flag',
    'GE65_Tot_Benes']

# dropping the listed columns
opioid_meds = opioid_meds.drop(columns=columns_to_drop, axis=1)

# rechecking the head
print(opioid_meds.head())
```

```
   Prscrbr_NPI Prscrbr_Last_Org_Name Prscrbr_First_Name Prscrbr_City  \
0   1003000142                Khalil             Rashid       Toledo
1   1003000407               Girardi              David   Brookville
2   1003000530              Semonche             Amanda   Quakertown
3   1003001363               Stevens            Charles    El Centro
4   1003002312               Hopkins           Patricia       Quincy

  Prscrbr_State_Abrvtn       Prscrbr_Type Brnd_Name Gnrc_Name  Tot_Clms  \
0                   OH     Anesthesiology  Fentanyl  Fentanyl        84
1                   PA    Family Practice  Fentanyl  Fentanyl        14
2                   PA  Internal Medicine  Fentanyl  Fentanyl        21
3                   CA     Anesthesiology  Fentanyl  Fentanyl        67
4                   MA       Rheumatology  Fentanyl  Fentanyl        25

   Tot_30day_Fills  Tot_Day_Suply  Tot_Drug_Cst  Tot_Benes
0             84.1           2495       6981.67       19.0
1             14.0            232       1067.07        NaN
2             21.0            630       1147.95        NaN
3             67.0           1965       8565.05       37.0
4             25.0            740       2221.23        NaN
```

[59]:
```
# getting the datatypes of each column
column_datatypes = opioid_meds.dtypes

# getting the number of unique values for each column
unique_values = opioid_meds.nunique()

# combining the dtypes and unique values into a single df
summary_df = pd.DataFrame({
    'DataType': column_datatypes,
    'UniqueValues': unique_values})

# printing the combined info
print(summary_df)
```

```
                      DataType  UniqueValues
Prscrbr_NPI              int64         82344
```

```
Prscrbr_Last_Org_Name   object         38894
Prscrbr_First_Name      object         12540
Prscrbr_City            object          6940
Prscrbr_State_Abrvtn    object            58
Prscrbr_Type            object            88
Brnd_Name               object            10
Gnrc_Name               object             3
Tot_Clms                 int64           905
Tot_30day_Fills        float64          3367
Tot_Day_Suply            int64          8504
Tot_Drug_Cst           float64        118892
Tot_Benes              float64           249
```

[61]:

```python
# mapping to new column names to align with prior naming convention
rename_columns = {
    'Prscrbr_NPI': 'clinician_npi',
    'Prscrbr_Last_Org_Name': 'clinician_lastname',
    'Prscrbr_First_Name': 'clinician_firstname',
    'Prscrbr_City': 'clinician_city',
    'Prscrbr_State_Abrvtn': 'clinician_state',
    'Prscrbr_Type': 'clinician_type',
    'Brnd_Name': 'brand_name',
    'Gnrc_Name': 'generic_name',
    'Tot_Clms': 'total_claims',
    'Tot_30day_Fills': 'total_30d_fills',
    'Tot_Day_Suply': 'total_day_supply',
    'Tot_Drug_Cst': 'total_cost',
    'Tot_Benes': 'total_beneficiaries'}

# renaming the columns in place
opioid_meds.rename(columns=rename_columns, inplace=True)

# displaying to confirm the change
print(opioid_meds.head())
```

```
   clinician_npi clinician_lastname clinician_firstname clinician_city  \
0     1003000142             Khalil              Rashid         Toledo
1     1003000407            Girardi               David     Brookville
2     1003000530           Semonche              Amanda     Quakertown
3     1003001363            Stevens             Charles      El Centro
4     1003002312            Hopkins            Patricia         Quincy

  clinician_state     clinician_type brand_name generic_name  total_claims  \
0              OH     Anesthesiology   Fentanyl     Fentanyl            84
1              PA    Family Practice   Fentanyl     Fentanyl            14
2              PA  Internal Medicine   Fentanyl     Fentanyl            21
3              CA     Anesthesiology   Fentanyl     Fentanyl            67
4              MA       Rheumatology   Fentanyl     Fentanyl            25
```

```
   total_30d_fills  total_day_supply  total_cost  total_beneficiaries
0             84.1              2495     6981.67                 19.0
1             14.0               232     1067.07                  NaN
2             21.0               630     1147.95                  NaN
3             67.0              1965     8565.05                 37.0
4             25.0               740     2221.23                  NaN
```

[62]:
```
# mapping state abbreviations to full names again to align with other dataframes
opioid_meds['clinician_state'] = opioid_meds['clinician_state'].
 ↪map(state_mapping)
```

[63]:
```
# display the head to confirm
print(opioid_meds.head())
```

```
   clinician_npi clinician_lastname clinician_firstname clinician_city  \
0     1003000142             Khalil              Rashid         Toledo
1     1003000407            Girardi               David     Brookville
2     1003000530           Semonche              Amanda     Quakertown
3     1003001363            Stevens             Charles     El Centro
4     1003002312            Hopkins            Patricia         Quincy

  clinician_state     clinician_type brand_name generic_name  total_claims  \
0            Ohio     Anesthesiology    Fentanyl     Fentanyl            84
1    Pennsylvania    Family Practice    Fentanyl     Fentanyl            14
2    Pennsylvania  Internal Medicine    Fentanyl     Fentanyl            21
3      California     Anesthesiology    Fentanyl     Fentanyl            67
4   Massachusetts       Rheumatology    Fentanyl     Fentanyl            25

   total_30d_fills  total_day_supply  total_cost  total_beneficiaries
0             84.1              2495     6981.67                 19.0
1             14.0               232     1067.07                  NaN
2             21.0               630     1147.95                  NaN
3             67.0              1965     8565.05                 37.0
4             25.0               740     2221.23                  NaN
```

[64]:
```
# running a count of na/NaN value and printing the count for each column
columns_to_check = ['total_claims', 'total_30d_fills', 'total_day_supply',␣
 ↪'total_cost', 'total_beneficiaries']
nan_counts = opioid_meds[columns_to_check].isna().sum()
print(nan_counts)
```

```
total_claims                0
total_30d_fills             0
total_day_supply            0
total_cost                  0
total_beneficiaries    107679
dtype: int64
```

I didn't find any na or NaN values, but I could clearly see that the total_beneficiaries column had multiple blanks.

[65]:
```
# running a count of blank strings and printing the count for each column
def is_blank(x):
    return x == '' or x.isspace() if isinstance(x, str) else False

blank_counts = opioid_meds[columns_to_check].applymap(is_blank).sum()
print(blank_counts)
```

```
total_claims          0
total_30d_fills       0
total_day_supply      0
total_cost            0
total_beneficiaries   0
dtype: int64
```

```
C:\Users\alyse\AppData\Local\Temp\ipykernel_30340\1434482315.py:5:
FutureWarning: DataFrame.applymap has been deprecated. Use DataFrame.map
instead.
  blank_counts = opioid_meds[columns_to_check].applymap(is_blank).sum()
```

[66]:
```
# replacing blank values with 10 for the total_beneficiaries column
opioid_meds['total_beneficiaries'] = opioid_meds['total_beneficiaries'].
 ↪replace(r'^\s*$', 10, regex=True)

# converting the column to numeric
opioid_meds['total_beneficiaries'] = pd.
 ↪to_numeric(opioid_meds['total_beneficiaries'])
```

[67]:
```
# converting the remaining desired columns to numeric
opioid_meds['total_claims'] = pd.to_numeric(opioid_meds['total_claims'],␣
 ↪errors='coerce')
opioid_meds['total_30d_fills'] = pd.to_numeric(opioid_meds['total_30d_fills'],␣
 ↪errors='coerce')
opioid_meds['total_day_supply'] = pd.
 ↪to_numeric(opioid_meds['total_day_supply'], errors='coerce')
opioid_meds['total_cost'] = pd.to_numeric(opioid_meds['total_cost'],␣
 ↪errors='coerce')
```

[68]:
```
# re-checking the datatypes of each column
column_datatypes = opioid_meds.dtypes

# re-counting the number of unique values for each column
unique_values = opioid_meds.nunique()

# combining the dtypes and unique values into a single df
summary_df = pd.DataFrame({
```

```
    'DataType': column_datatypes,
    'UniqueValues': unique_values})

# printing the refreshed info
print(summary_df)
```

```
                      DataType  UniqueValues
clinician_npi            int64         82344
clinician_lastname      object         38894
clinician_firstname     object         12540
clinician_city          object          6940
clinician_state         object            51
clinician_type          object            88
brand_name              object            10
generic_name            object             3
total_claims             int64           905
total_30d_fills        float64          3367
total_day_supply         int64          8504
total_cost             float64        118892
total_beneficiaries    float64           249
```

[69]:

```
# re-printing the head after cleaning
print(opioid_meds.head())
```

```
   clinician_npi clinician_lastname clinician_firstname clinician_city  \
0     1003000142             Khalil              Rashid         Toledo
1     1003000407            Girardi               David     Brookville
2     1003000530           Semonche              Amanda     Quakertown
3     1003001363            Stevens             Charles      El Centro
4     1003002312            Hopkins            Patricia         Quincy

  clinician_state     clinician_type brand_name generic_name  total_claims  \
0            Ohio     Anesthesiology   Fentanyl     Fentanyl            84
1    Pennsylvania    Family Practice   Fentanyl     Fentanyl            14
2    Pennsylvania  Internal Medicine   Fentanyl     Fentanyl            21
3      California     Anesthesiology   Fentanyl     Fentanyl            67
4   Massachusetts       Rheumatology   Fentanyl     Fentanyl            25

   total_30d_fills  total_day_supply  total_cost  total_beneficiaries
0             84.1              2495     6981.67                 19.0
1             14.0               232     1067.07                  NaN
2             21.0               630     1147.95                  NaN
3             67.0              1965     8565.05                 37.0
4             25.0               740     2221.23                  NaN
```

0.0.18 List of Data Transformations

The data transformations I made on the API data included:
1. Nine columns were dropped due to being unnecessary or at least secondary to my project

question. 2. Thirteen columns had their column name changed to match my prior lowercase naming conventions. 3. Values in 'clinician_state' were mapped to convert state abbreviations to full state names in capital case, to match earlier state value decisions. 4. NaN and blank string values were checked for in all columns I preferred to be numeric as I anticipated I may do mathematical calculations. 5. Blank strings were converted to '10' in alignment with API documentation regarding blanks = <11. 6. Five columns were converted from a datatype of object to numeric for consistency and for future mathematical calculation.

0.0.19 Ethical Considerations of Milestone Four

In addition to the ethical considerations mentioned in the prior milestone, I wanted to call out the decision to transform the values in the column titled, 'total_beneficiaries'. The API documentation defines this column as, "The total number of unique Medicare Part D beneficiaries with at least one claim for the drug."
The documentation also addresses the blank fields stating, "Counts fewer than 11 are suppressed and are indicated by a blank."

This left me with a choice to either fill the blanks with a mean, median, or other statistic, fill the blank with a 10, or remove the column entirely.
At this point, I've decided to fill the blanks with the numeric 10, primarily because this number aligns with the maxium possible actual count, so I hope this gives a conservative estimate of the true counts, rather than using a higher average than would have been accurate.
At the same time, if I use this column to draw conclusions, I would need to acknowledge that it is only a threshold that was set.
My primary concern here is that there were nearly 79% of the values for this column initially left blank.

While my '10' solution seems better than using the mean, the large number of blanks indicates this adjustment may also introduce added bias.
I will consider this between now and the final assignment.

0.0.20 Milestone 5: Loading and Using the Database

In this milestone, I needed to load all the cleaned data into individual tables in my SQLite database, and then join the datasets together in Python. Prior to taking this step, I decided to take one more view of each of the data sources/dataframes to ensure that merging would go smoothly.

[70]:
```
# listing all dataframes
dataframes = {name: obj for name, obj in globals().items() if isinstance(obj,␣
↪pd.DataFrame)}

# printing the names
print("List of all DataFrames:")
for name in dataframes:
    print(name)
```

```
List of all DataFrames:
deathrates
opioidproviders
demographic_counts
```

```
month_counts
df
first_two_columns_data
rows_with_two_npis
top_5_longest
duplicates
selected_df
unique_overdose_types
unique_demographic_details
male_female_data
grouped_data
wiki_od_deaths
wiki_od_states
wiki_od_states_raw
population_2020
provider_counts_df
merged_data
part_d
test_meds
opioid_meds
summary_df
```

[71]:

```python
# defining the list of dataframes that may move to SQLite
dataframe_names = ['deathrates', 'opioidproviders', 'wiki_od_deaths',␣
↪'wiki_od_states', 'population_2020', 'opioid_meds']

# printing the column names for each in a loop
for name in dataframe_names:
    df = globals().get(name)  # Attempt to get the DataFrame by name
    if df is not None and isinstance(df, pd.DataFrame):
        print(f"Columns in {name}: {list(df.columns)}")
    else:
        print(f"{name} does not exist or is not a DataFrame.")
```

```
Columns in deathrates: ['overdose_type', 'overdose_type_num', 'deathsper100k',
'demographic_name', 'demographic_detail', 'demographic_detail_num', 'year',
'age_group', 'estimate']
Columns in opioidproviders: ['practice_npis', 'practice_name', 'address',
'city', 'state', 'zip', 'medicare_date', 'phone_number']
Columns in wiki_od_deaths: ['year', 'deaths', 'population_count', 'crude_rate',
'age_adjusted_rate']
Columns in wiki_od_states: ['state', '1999', '2005', '2014', '2015', '2016',
'2017', '2018', '2019', '2020', '2021']
Columns in population_2020: ['state', 'population']
Columns in opioid_meds: ['clinician_npi', 'clinician_lastname',
'clinician_firstname', 'clinician_city', 'clinician_state', 'clinician_type',
'brand_name', 'generic_name', 'total_claims', 'total_30d_fills',
'total_day_supply', 'total_cost', 'total_beneficiaries']
```

0.0.21 Examining Unique Values for Final Standardization

[72]:
```
# deathrates: 'year'
print("First 10 unique years in deathrates:", deathrates['year'].unique()[:10])

# wiki_od_deaths: 'year'
print("First 10 unique years in wiki_od_deaths:", wiki_od_deaths['year'].
 ↪unique()[:10])

# opioidproviders: 'practice_npis'
print("First 10 unique practice_npis in opioidproviders:",␣
 ↪opioidproviders['practice_npis'].unique()[:10])

# opioid_meds: 'clinician_npi'
print("First 10 unique clinician_npi in opioid_meds:",␣
 ↪opioid_meds['clinician_npi'].unique()[:10])

# opioidproviders: 'state'
print("First 10 unique states in opioidproviders:", opioidproviders['state'].
 ↪unique()[:10])

# wiki_od_states: 'state'
print("First 10 unique states in wiki_od_states:", wiki_od_states['state'].
 ↪unique()[:10])

# population_2020: 'state'
print("First 10 unique states in population_2020:", population_2020['state'].
 ↪unique()[:10])

# opioid_meds: 'clinician_state'
print("First 10 unique clinician_states in opioid_meds:",␣
 ↪opioid_meds['clinician_state'].unique()[:10])
```

```
First 10 unique years in deathrates: [1999 2000 2001 2002 2003 2004 2005 2006
2007 2008]
First 10 unique years in wiki_od_deaths: ['' '1968' '1969' '1970' '1971' '1972'
'1973' '1974' '1975' '1976']
First 10 unique practice_npis in opioidproviders: ['1003081399 1013055110'
'1003150004' '1003362484' '1003368945'
 '1003571647' '1003581174 1326713314' '1003583733' '1003947193'
 '1003953548' '1003958976']
First 10 unique clinician_npi in opioid_meds: [1003000142 1003000407 1003000530
1003001363 1003002312 1003005034
 1003007469 1003009218 1003010786 1003010950]
First 10 unique states in opioidproviders: ['Vermont' 'Wisconsin' 'Virginia'
'Maryland' 'Florida' 'Ohio' 'New Jersey'
 'Texas' 'Georgia' 'California']
First 10 unique states in wiki_od_states: ['' 'Alabama' 'Alaska' 'Arizona'
```

```
'Arkansas' 'California' 'Colorado'
 'Connecticut' 'Delaware' 'Florida']
First 10 unique states in population_2020: ['California' 'Texas' 'Florida' 'New
York' 'Pennsylvania' 'Illinois'
 'Ohio' 'Georgia' 'North Carolina' 'Michigan']
First 10 unique clinician_states in opioid_meds: ['Ohio' 'Pennsylvania'
'California' 'Massachusetts' 'Texas' 'Arkansas'
 'New York' 'Oklahoma' 'Mississippi' 'North Dakota']
```

Final Data Cleansing Decisions

[73]:
```python
# converting the clinician_npi column to string
opioid_meds['clinician_npi'] = opioid_meds['clinician_npi'].astype(str)

# checking the change
print(opioid_meds['clinician_npi'].dtype)
```

```
object
```

[74]:
```python
# checking for matching npi at the practice vs clinician level: This would help␣
↪me determine column naming convention

# splitting npis as some values have more than one, separated by a space
opioid_meds['clinician_npi'] = opioid_meds['clinician_npi'].str.split()

# flattening the list of lists into a single list of NPIs
flattened_meds_npis = [npi for sublist in opioid_meds['clinician_npi'] for npi␣
↪in sublist]

# splitting npis in opioidproviders, too
opioid_providers_npis = opioidproviders['practice_npis'].str.split().explode().
↪unique()

# converting both to sets for efficient comparison
set_meds_npis = set(flattened_meds_npis)
set_providers_npis = set(opioid_providers_npis)

# checking for matches
matching_npis = set_meds_npis.intersection(set_providers_npis)

# printing the number of matches found
print(f"Number of unique matching NPIs: {len(matching_npis)}")
```

```
Number of unique matching NPIs: 0
```

Notes on Keys and NPIs Note: Although initially, I was planning to use NPIs as a key across tables, I have since found that the NPIs listed in opioidproviders and the ones listed in opioid_meds are different, one being associated with individual providers and the other being associated with the Tax ID of the practice locations. I did attempt to split/explode to preserve these, but honestly, after a few iterations, I have realized they serve no purpose in my data anymore. I don't have a crosswalk to know which providers are associated with which practices, and as a result, I will be using city and state as my keys and focus more on the regional level aggreggation of the data for this assignment.

Because of this choice, I can simply drop the NPI columns and should be able to upload the rest to SQLite.

[75]:
```python
# dropping the two NPI-related columns
if 'practice_npis' in opioidproviders.columns:
    opioidproviders.drop('practice_npis', axis=1, inplace=True)

if 'clinician_npi' in opioid_meds.columns:
    opioid_meds.drop('clinician_npi', axis=1, inplace=True)
```

[76]:
```python
# renaming the 'clinician_state' column to 'state' in opioid_meds &␣
↪'clinician_city' to 'city'
opioid_meds.rename(columns={'clinician_state': 'state'}, inplace=True)
opioid_meds.rename(columns={'clinician_city': 'city'}, inplace=True)

# verifying the change
print(opioid_meds.columns)
```

```
Index(['clinician_lastname', 'clinician_firstname', 'city', 'state',
       'clinician_type', 'brand_name', 'generic_name', 'total_claims',
       'total_30d_fills', 'total_day_supply', 'total_cost',
       'total_beneficiaries'],
      dtype='object')
```

[77]:
```python
# converting empty 'year' strings to NaN, then to integers for wiki_od_deaths
wiki_od_deaths['year'] = pd.to_numeric(wiki_od_deaths['year'], errors='coerce').
↪fillna(0).astype(int)

# checking the conversion
print(wiki_od_deaths['year'].unique()[:10])
```

```
[   0 1968 1969 1970 1971 1972 1973 1974 1975 1976]
```

[78]:
```python
# checking the first 10 unique cities in opioidproviders
print("First 10 unique cities in opioidproviders:", opioidproviders['city'].
↪unique()[:10])

# checking the first 10 unique cities in opioid_meds
print("First 10 unique cities in opioid_meds:", opioid_meds['city'].unique()[:
↪10])
```

```
First 10 unique cities in opioidproviders: ['BERLIN' 'ONALASKA' 'VIRGINIA BEACH'
'EDGEWOOD' 'ORANGE PARK' 'KETTERING'
 'CHERRY HILL' 'PLANO' 'DECATUR' 'ATHENS']
First 10 unique cities in opioid_meds: ['Toledo' 'Brookville' 'Quakertown' 'El
```

```
Centro' 'Quincy' 'Newark'
 'Killeen' 'Morrilton' 'Amsterdam' 'Fountain Valley']
```

[79]:
```
# converting the 'city' column in opioidproviders to Capital Case
opioidproviders['city'] = opioidproviders['city'].str.title()

# verifying the change
print("First 10 unique cities in opioidproviders after conversion:",␣
↪opioidproviders['city'].unique()[:10])
```

```
First 10 unique cities in opioidproviders after conversion: ['Berlin' 'Onalaska'
'Virginia Beach' 'Edgewood' 'Orange Park' 'Kettering'
 'Cherry Hill' 'Plano' 'Decatur' 'Athens']
```

[80]:
```
# final check of df and columns
dataframe_names = ['deathrates', 'opioidproviders', 'wiki_od_deaths',␣
↪'wiki_od_states', 'population_2020', 'opioid_meds']

# printing the column names for each in a loop
for name in dataframe_names:
    df = globals().get(name)  # Attempt to get the DataFrame by name
    if df is not None and isinstance(df, pd.DataFrame):
        print(f"Columns in {name}: {list(df.columns)}")
    else:
        print(f"{name} does not exist or is not a DataFrame.")
```

```
Columns in deathrates: ['overdose_type', 'overdose_type_num', 'deathsper100k',
'demographic_name', 'demographic_detail', 'demographic_detail_num', 'year',
'age_group', 'estimate']
Columns in opioidproviders: ['practice_name', 'address', 'city', 'state', 'zip',
'medicare_date', 'phone_number']
Columns in wiki_od_deaths: ['year', 'deaths', 'population_count', 'crude_rate',
'age_adjusted_rate']
Columns in wiki_od_states: ['state', '1999', '2005', '2014', '2015', '2016',
'2017', '2018', '2019', '2020', '2021']
Columns in population_2020: ['state', 'population']
Columns in opioid_meds: ['clinician_lastname', 'clinician_firstname', 'city',
'state', 'clinician_type', 'brand_name', 'generic_name', 'total_claims',
'total_30d_fills', 'total_day_supply', 'total_cost', 'total_beneficiaries']
```

0.0.22 Moving to SQLite!

Saving Backups of the Cleaned DFs as CSV files

[81]:
```
# defining the base path for the CSV files
base_path = 'C:/Users/alyse/OneDrive/Documents/Bellevue University/DSC 540 -␣
↪Data Preparation/Final Project Data/Cleaned Backups/'

# appending the filenames
csv_paths = {
```

```
    'deathrates': base_path + 'deathrates_clean.csv',
    'opioidproviders': base_path + 'opioidproviders_clean.csv',
    'wiki_od_deaths': base_path + 'wiki_od_deaths_clean.csv',
    'wiki_od_states': base_path + 'wiki_od_states_clean.csv',
    'population_2020': base_path + 'population_2020_clean.csv',
    'opioid_meds': base_path + 'opioid_meds_clean.csv'}

# saving each DF to a csv file at the path
for df_name, path in csv_paths.items():
    globals()[df_name].to_csv(path, index=False)
```

[82]:
```
# defining the DFs to check
dataframe_names = ['deathrates', 'opioidproviders', 'wiki_od_deaths',␣
 ↪'wiki_od_states', 'population_2020', 'opioid_meds']

# checking each DF for any columns that contain lists -- SQLite does *not* like␣
 ↪those.
for df_name in dataframe_names:
    df = globals()[df_name]  # Get the DataFrame by name
    for column in df.columns:
        if df[column].apply(lambda x: isinstance(x, list)).any():
            print(f"DataFrame '{df_name}' column '{column}' contains list-type␣
 ↪data.")
```

[83]:
```
# connecting to my SQLite database
conn = sqlite3.connect('my_database.db')

# defining tables in SQLite
table_names = {
    'deathrates': 'deathrates',
    'opioidproviders': 'opioidproviders',
    'wiki_od_deaths': 'wiki_od_deaths',
    'wiki_od_states': 'wiki_od_states',
    'population_2020': 'population_2020',
    'opioid_meds': 'opioid_meds'}

# uploading each df as a separate table
for df_name, table_name in table_names.items():
    globals()[df_name].to_sql(table_name, conn, if_exists='replace',␣
 ↪index=False)
```

0.0.23 Consolidating a Dataset

[84]:
```
# reshaping wiki_od_states for year-wise comparisons
wiki_od_states_melted = pd.melt(wiki_od_states, id_vars=['state'],␣
 ↪var_name='year', value_name='state_deaths')
wiki_od_states_melted['year'] = wiki_od_states_melted['year'].astype(int)
```

[85]:
```
# merging opioidproviders with population data
state_level = pd.merge(opioidproviders, population_2020, on='state', how='left')

# merging opioid_meds to add provider prescription data
state_level = pd.merge(state_level, opioid_meds[['city', 'state',␣
↪'total_day_supply', 'total_beneficiaries']], on=['city', 'state'],␣
↪how='left')
```

[87]:
```
# reshaping wiki_od_states to have 'year' and 'state_deaths' columns
wiki_od_states_long = pd.melt(wiki_od_states, id_vars=['state'],␣
↪var_name='year', value_name='state_deaths')

# converting 'year' from string to integer for consistent merging
wiki_od_states_long['year'] = wiki_od_states_long['year'].astype(int)
```

[90]:
```
# ensuring 'year' in deathrates is an int - just to make sure
deathrates['year'] = deathrates['year'].astype(int)

# merging deathrates with nationwide yearly deaths from wiki_od_deaths
temporal_analysis = pd.merge(deathrates, wiki_od_deaths[['year', 'deaths']],␣
↪on='year', how='left')
```

0.0.24 Post-Merging Descriptions

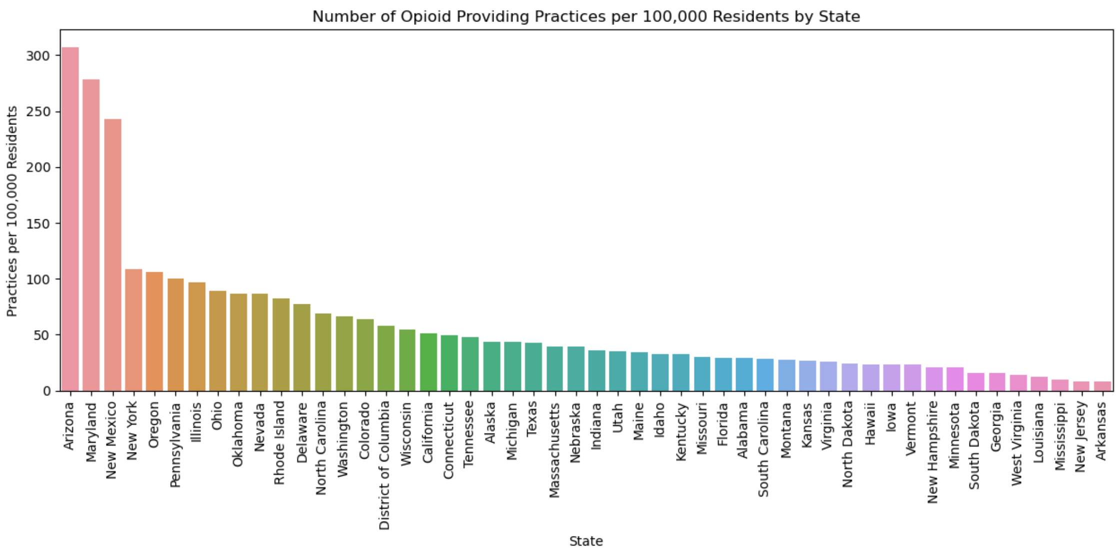
[102]:
```
print(wiki_od_states_melted.columns)     # easier year-by-year and␣
↪state-by-state analyses
print(state_level.columns)     # useful to examine the density of opioid␣
↪prescribing practices
print(wiki_od_states_long.columns)     # prepped for analyses of trends in␣
↪opioid deaths across states over time
print(temporal_analysis.columns)     # national trends in opioid deaths over␣
↪time
```

```
Index(['state', 'year', 'state_deaths'], dtype='object')
Index(['practice_name', 'address', 'city', 'state', 'zip', 'medicare_date',
       'phone_number', 'population', 'total_day_supply', 'total_beneficiaries',
       'practices_per_100k', 'providers_per_100k'],
      dtype='object')
Index(['state', 'year', 'state_deaths'], dtype='object')
Index(['overdose_type', 'overdose_type_num', 'deathsper100k',
       'demographic_name', 'demographic_detail', 'demographic_detail_num',
       'year', 'age_group', 'estimate', 'deaths'],
      dtype='object')
```

0.0.25 Data Visualizations

Data Vizualization One - Bar Graph

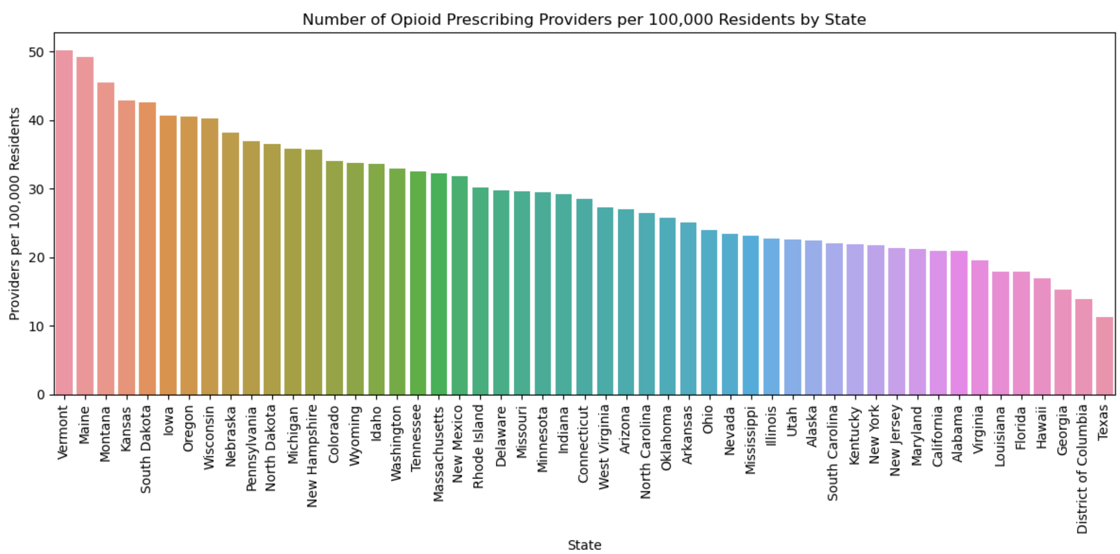
[100]:
```
# calculating practices per 100k
state_level['practices_per_100k'] = (state_level.
 ↪groupby('state')['practice_name'].transform('count') /␣
 ↪state_level['population']) * 100000

# dropping duplicates and sorting
sorted_state_level = state_level.drop_duplicates('state').
 ↪sort_values('practices_per_100k', ascending=False)

# Plot
plt.figure(figsize=(12, 6))
sns.barplot(x='state', y='practices_per_100k', data=sorted_state_level)
plt.xticks(rotation=90)
plt.title('Number of Opioid Providing Practices per 100,000 Residents by State')
plt.xlabel('State')
plt.ylabel('Practices per 100,000 Residents')
plt.tight_layout()
plt.show()
```



Data Visualization Two: Bar Graph

[107]:
```
# creating a unique identifier(ish) for each clinician using firstname,␣
 ↪lastname, and state
opioid_meds['clinician_unique_id'] = opioid_meds['clinician_firstname'] + ' ' +␣
 ↪opioid_meds['clinician_lastname'] + ' ' + opioid_meds['state']

# merging opioid_meds with population data to get state-level population
opioid_meds_with_population = pd.merge(opioid_meds, population_2020,␣
 ↪on='state', how='left')
```

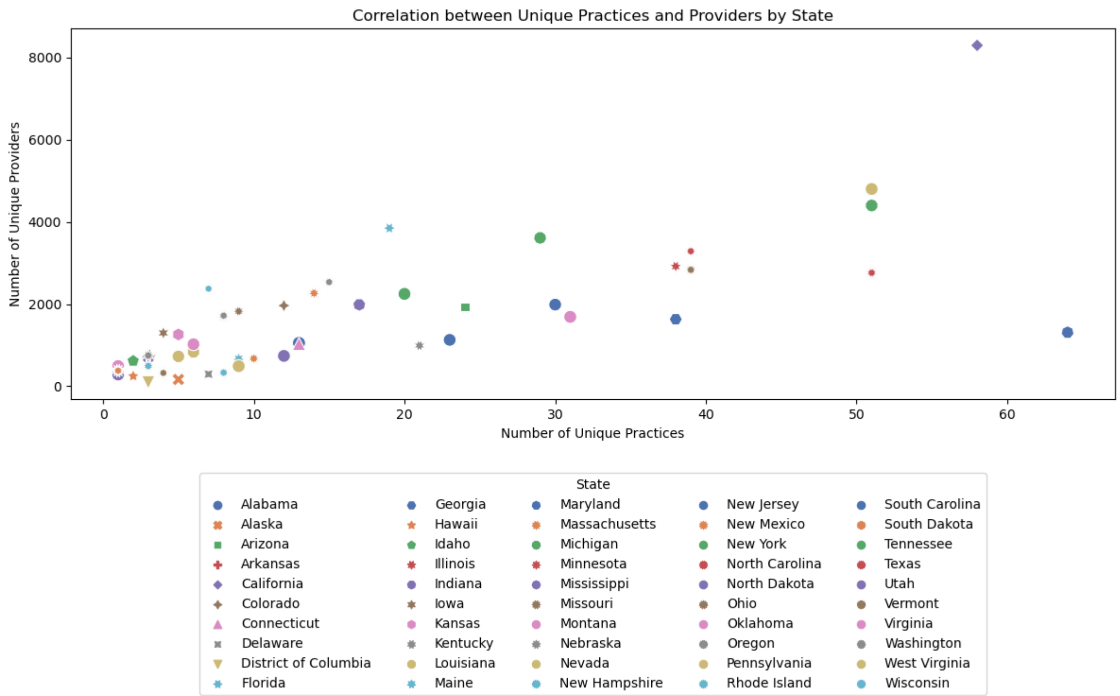
```python
# calculating the number of unique providers per 100,000 residents by state
opioid_meds_with_population['providers_per_100k'] = opioid_meds_with_population.
 ↪groupby('state')['clinician_unique_id'].transform('nunique') /␣
 ↪opioid_meds_with_population['population'] * 100000

# aggregating to get one row per state with the calculated metric
providers_per_100k_by_state = opioid_meds_with_population[['state',␣
 ↪'providers_per_100k']].drop_duplicates('state')

# sorting by providers_per_100k in descending order
sorted_providers = providers_per_100k_by_state.
 ↪sort_values('providers_per_100k', ascending=False)

plt.figure(figsize=(12, 6))
sns.barplot(x='state', y='providers_per_100k', data=sorted_providers)
plt.xticks(rotation=90)
plt.title('Number of Opioid Prescribing Providers per 100,000 Residents by␣
 ↪State')
plt.xlabel('State')
plt.ylabel('Providers per 100,000 Residents')
plt.tight_layout()
plt.show()
```



Visualization Three: Scatter Plot Correlation

[112]:
```python
# counting unique practices by state from opioidproviders
unique_practices_by_state = opioidproviders.groupby('state')['practice_name'].
 ↪nunique().reset_index(name='unique_practices')
```

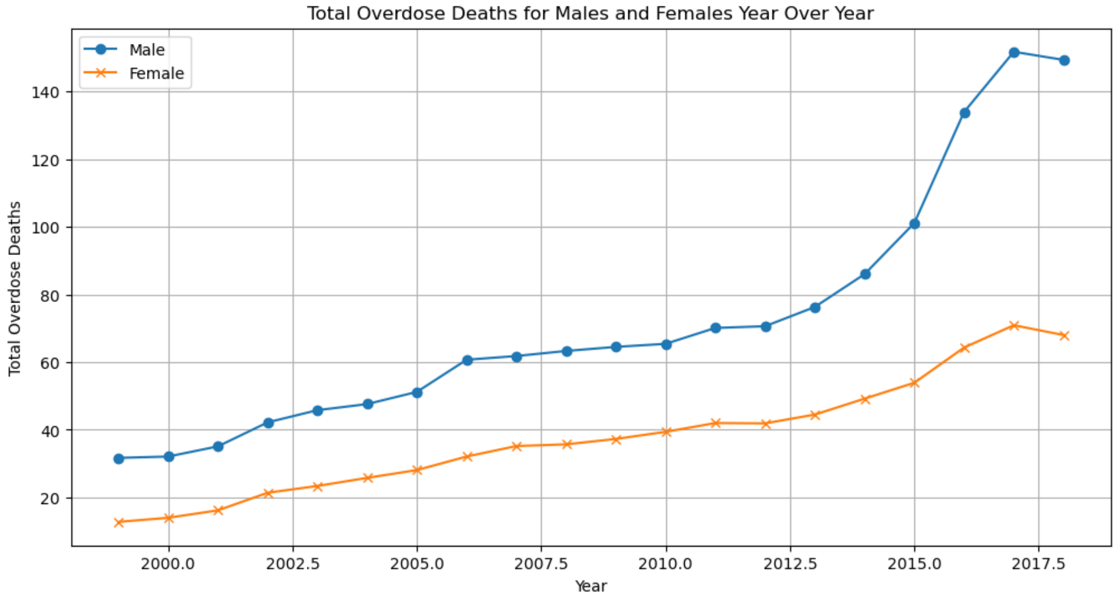
```python
# counting unique providers by state from opioid_meds
unique_providers_by_state = opioid_meds.groupby('state')['clinician_unique_id'].nunique().reset_index(name='unique_providers')

# merging the unique counts by state into a single df
state_counts = pd.merge(unique_practices_by_state, unique_providers_by_state, on='state')

# creating a scatter plot of unique practices vs. unique providers by state
plt.figure(figsize=(12, 8))
sns.scatterplot(data=state_counts, x='unique_practices', y='unique_providers', hue='state', style='state', palette='deep', s=100)
plt.title('Correlation between Unique Practices and Providers by State')
plt.xlabel('Number of Unique Practices')
plt.ylabel('Number of Unique Providers')
plt.legend(title='State', bbox_to_anchor=(0.5, -0.2), loc='upper center', ncol=5, borderaxespad=0.)

plt.tight_layout()
plt.show()

# calculating and printing correlation coefficient
# this is more related to DSC550-Data Mining -- but since I'm here making this plot...¯\_( )_/¯
correlation = state_counts['unique_practices'].corr(state_counts['unique_providers'])
print(f'Pearson Correlation Coefficient: {correlation:.2f}')
```




```
Pearson Correlation Coefficient: 0.74
```

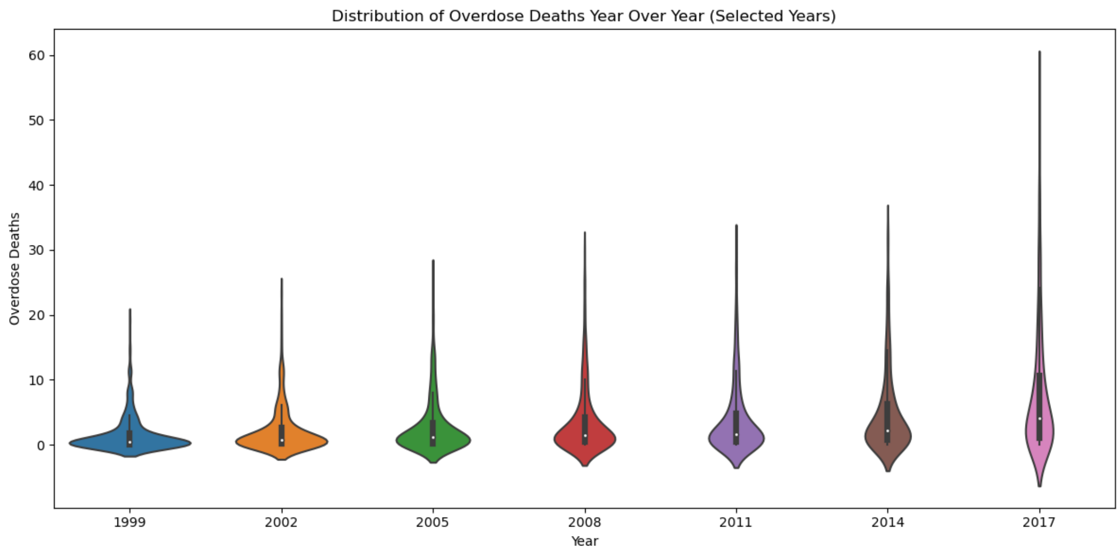
Visualization Four: Line Graph

[114]:
```
# grouping male/female data and total overdose deaths
male_female_data = temporal_analysis[temporal_analysis['demographic_detail'].
 ↪isin(['Male', 'Female'])]
grouped_data = male_female_data.groupby(['year',␣
 ↪'demographic_detail'])['estimate'].sum().unstack()

# plotting
plt.figure(figsize=(12, 6))
plt.plot(grouped_data.index, grouped_data['Male'], label='Male', marker='o')
plt.plot(grouped_data.index, grouped_data['Female'], label='Female', marker='x')

plt.title('Total Overdose Deaths for Males and Females Year Over Year')
plt.xlabel('Year')
plt.ylabel('Total Overdose Deaths')
plt.legend()
plt.grid(True)
plt.show()
```



Visualization Five: Violin Graph

[125]:
```
# selecting years at a 3-year interval to more clearly display distribution␣
↪change
selected_years = list(range(1999, 2019, 3))

# filtering the deathrates df to include only selected years
filtered_deathrates = deathrates[deathrates['year'].isin(selected_years)]

plt.figure(figsize=(12, 6))
sns.violinplot(x="year", y="estimate", data=filtered_deathrates)
plt.title('Distribution of Overdose Deaths Year Over Year (Selected Years)')
plt.xlabel('Year')
plt.ylabel('Overdose Deaths')
plt.tight_layout()
plt.show()
```



[126]:
```python
# closing the connection
conn.close()
```

0.0.26 Summary

This project has been an exercise in the process of data preparation, analysis, and visualization. The project focused on the critical public health issue of opioid prescribing and overdose in the United States. This involved multiple stages, including loading and cleansing data from various sources including CSV files, websites, and APIs, then merging this data into a consolidated SQLite database for comprehensive analysis.

One of the first challenges encountered was one of computing resources, as the large amount of available data far outweighed the scope of the project, necessitating a focused goal and ensuring the API requests reflected the only the project's scope, rather than satisfying additional curisosities.

Next, there were challenges in ensuring the cleanliness and compatibility of the data. This required strong attention to detail in finding and correcting inaccuracies, missing values, and inconsistencies across datasets. For example, my assumption was that I would be able to leverage clinician NPI numbers as unique identifiers, but later found that in one of the datasets, practice NPIs were listed instead.
This required some amount of creativity, so I generated my own unique identifier using the clinician first name, last name, and state. Because there could be clinicians on the list with the same name, from the same state, this isn't the most ideal identifier to use. However, depending on the accuracy needed for the output, something like this may be a good fit for future needs. Creating additional robustness through finding another field to add to this, like a date of birth or some other personally identifiable datapoint, could have further potential. Next time, running some level of analysis surrounding the "clinician match" using the selected identifier could also be useful.

Loading the data into the SQLite database took a few attempts, but once that was done, analyzing the data for connections was possible. This involved creating visualizations to uncover insights into the distribution of opioid prescribing practices and the impact of opioid overdoses across different demographics and states. While the preference would have been to start with a more closely aligned dataset at the most granualar level (individual prescriptions, prescribers, and locations and/or beneficiaries) some interesting insights in the available data were found.

Ethical considerations were also essential to this project. Although the data was deidentified, the data cleansing, particularly in the context of a sensitive topic like opioid use, required a thoughtful approach to ensure that the modifications did not introduce bias or distort the underlying reality.

In summary, this project was both a technical exercise in data manipulation and an exploration of the different challenges in working with health-related data.
This was a fantastic project I found challenging and interesting in many ways. Thanks so much for the class!

[ ]:

```

```