DSC-540 Final Project

Chitramoy Mukherjee-DSC540-T304

Date : 09/22/2023

3 data sources and it's descriptions

1. acs2017_county_data.csv : This data file contains US county level census data for year-2017.This dataset is downloaded from kaggle.
2. Wikipedia List of states table contains US state information. Thsi table contains US 50 states information.
3. US Government Data: The US government provides a wide range of public APIs, including data on demographics, economics, and crime. (https://www.census.gov/data/developers/data-sets.html)

Relationship between 3 data sources

US Census Data(acs2017_county_data.csv) and the List of US States table can be linked by geographic location ans state name. We could use the Google Maps API to determine the latitude and longitude for each state in the List of US States table, and then use this information to link the state data to the demographic and economic data in the US Census Data dataset.

Interpretation and operations on dataset to accomplish future milestones

```
    Based on the state name and it's geographic information, we can
merge this 3 datasets after removing the headers from    those. After
the first step will remove the unwanted columns from the datasets
andf then merge those three into one dataset and that dataset could
be used to inform policy makers and economic developers about the
factors that contribute to population growth. It could also be used
to identify states that are at risk of population decline, and to
develop targeted interventions to promote population growth in these
states.

    As a data wrangling project using these datasets would be to
create a dataset that maps the demographics and economic factors of
each US state to the state's population growth rate. This could be
done by linking the US Census Data dataset and the List of US States
table, as described above. Once the datasets are linked, we could use
statistical analysis to calculate the population growth rate for each
state, and then identify correlations between the population growth
rate and demographic and economic factors, such as median income,
poverty rate, and education levels.
```

Data Disctionary for acs2017_county_data.csv :

Data columns (total 37 columns):

Column No. Column Data type Description

---

0 Countyid int64 County identification #
1 State object Name of the state 2 County object Name of the county 3 TotalPop int64 Total population 4 Men int64 Men count 5 Women int64 Women count 6 Hispanic float64 % of population that is Hispanic/Latino 7 White float64 % of population that is white 8 Black float64 % of population that is black 9 Native float64 % of population that is Native American or Native Alaskan 10 Asian float64 % of population that is Asian 11 Pacific float64 % of population that is Native Hawaiian or Pacific Islander 12 VotingAge int64 Voting age in days
13 Income float64 Median household income ( )$14IncomeErrfloat64Medianhouseholdincomeerror()$ 15 IncomePerCap float64 Income per capita ()$16IncomePerCapErrfloat64Incomepercapitaerror()$ 17 Poverty float64 % under poverty level 18 ChildPoverty float64 % of children under poverty level 19 Professional float64 % employed in management, business, science, and arts 20 Service float64 % employed in service jobs 21 Office float64 % employed in sales and office jobs 22 Construction float64 % employed in natural resources, construction, and maintenance 23 Production float64 % employed in production, transportation, and material movement 24 Drive float64 % commuting alone in a car, van, or truck 25 Carpool float64 % carpooling in a car, van, or truck 26 Transit float64 % commuting on public transportation 27 Walk float64 % walking to work 28 OtherTransp float64 % commuting via other means 29 WorkAtHome float64 % working at home 30 MeanCommute float64 Mean commute time (minutes) 31 Employed int64 Number of employed (16+) 32 PrivateWork float64 % employed in private industry 33 PublicWork float64 % employed in public jobs
34 SelfEmployed float64 % self-employed 35 FamilyWork float64 % in unpaid family work 36 Unemployment float64 Unemployment rate (%)

List of states Wikipedia Table data dictionary :

Column No. Column Data type Description

---

1 Postal abbrevation object State Name 2 Cities object Major City bypopulation/state capital 3 Established Date Year state formed 4 Population int64 total state population 5 Total area int64 Total area 6 Land area int64 Total land 7 Water area int64 Total water area

Project subject area

Will apply different data wragling techniques on the source data and merge it to perform the analysis.

As a part of this project we will be merging 3 different dataset of differnt type using a common key(state name) and will perform statistical analysis to identify correlations between crime rates

and demographic and economic factors, such as poverty, unemployment, and education levels.

Data Sources:

1. acs2017_county_data.csv (https://www.kaggle.com/code/alawdisoft/us-census-demographic-data/input?select=acs2017_county_data.csv)
2. The US government provides a wide range of public APIs, including data on demographics, economics, and crime. US Census Bureau provides an API for accessing census data. (https://www.census.gov/data/developers/data-sets.html)
3. his Wikipedia table contains a list of all 50 US states, along with their capitals and population.(https://simple.wikipedia.org/wiki/List_of_U.S._states)

Relationships :

All 3 datasets contain data based on state. The lowest granularity of this 3 dataset data is state name.

Ethical implications and Challenges :

Ethical implications of using US Census Data for a data wrangling project include:

Privacy: The US Census Data contains personal information about individuals and households. It is important to take steps to protect the privacy of this data, such as anonymizing the data or using differential privacy techniques.

Bias: The US Census Data may be biased in certain ways. For example, it may be more difficult to reach certain populations, such as low-income households or immigrant communities. It is important to be aware of these potential biases and to take steps to mitigate them.

Discrimination: The US Census Data could be used to discriminate against certain groups of people. For example, it could be used to target certain groups with marketing messages or to deny them access to services or opportunities. It is important to use the data in a responsible and ethical way to avoid discrimination.

Use differential privacy techniques: Differential privacy is a set of techniques that can be used to protect the privacy of individuals in a dataset while still allowing for accurate analysis.

Some of the challenges that you might face in a US Census Data project include:

Data quality: The US Census Data is a large and complex dataset. It is important to carefully clean and prepare the data before using it for analysis.

Data complexity: The US Census Data contains a wide range of variables. It is important to understand the meaning of the variables and how they can be used for analysis.

Ethical considerations: As discussed above, there are a number of ethical considerations that must be taken into account when using US Census Data. It is important to design your project in a way that respects the privacy of the data and avoids bias and discrimination.

Milestone-2

Apply 5 transformations to acs2017_county_data.csv dataset

In [142…

```
import pandas as pd #Linear Algebra
import numpy as np #Data Processing
import seaborn as sns #Visualization
import matplotlib.pyplot as plt #Visualization
import pandasql as psql
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_17608\2747350820.py in <module>
      3 import seaborn as sns #Visualization
      4 import matplotlib.pyplot as plt #Visualization
----> 5 import pandasql as psql

ModuleNotFoundError: No module named 'pandasql'
```

In [123…

```
import pandas as pd

# Load the CSV file
file_path = 'C:\\Users\\14024\\OneDrive\\Desktop\\MS-DSC\\DSC-540\\DSC-540 Project\\Mi
data = pd.read_csv(file_path)

# Optionally, you can also display the first few rows to verify the new headers
print(df.head())
```

```
   CountyId    State          County  TotalPop        Men      Women  \
0      1001  Alabama  Autauga County     55036  48.875282  51.124718
1      1003  Alabama  Baldwin County    203360  48.941286  51.058714
2      1005  Alabama  Barbour County     26201  53.341476  46.658524
3      1007  Alabama     Bibb County     22580  54.255979  45.744021
4      1009  Alabama   Blount County     57667  49.404339  50.595661

   Hispanic  White  Black  Asian  ...  OtherTransp  WorkAtHome  MeanCommute  \
0       2.7   75.4   18.9    0.9  ...          1.3         2.5         25.8
1       4.4   83.1    9.5    0.7  ...          1.1         5.6         27.0
2       4.2   45.7   47.8    0.6  ...          1.7         1.3         23.4
3       2.4   74.6   22.0    0.0  ...          1.7         1.5         30.0
4       9.0   87.4    1.5    0.1  ...          0.4         2.1         35.0

    Employed  PrivateWork  PublicWork  SelfEmployed  FamilyWork  Unemployment  \
0  43.811323         74.1        20.2           5.6         0.1           5.2
1  44.023899         80.7        12.9           6.3         0.1           5.5
2  33.884203         74.1        19.1           6.5         0.3          12.4
3  36.186891         76.0        17.4           6.3         0.3           8.2
4  37.074930         83.9        11.9           4.0         0.1           4.9

   OtherRace
0        0.3
1        0.8
2        0.2
3        0.4
4        0.3

[5 rows x 36 columns]
```

In [124…

```
# Modify the column headers with prefix "US_2017_"
data_census_2017 =  data.add_prefix('US_2017_')
data_census_2017.head()

# Modifying header/column name with US_2017_ to identify the data corresponds to US an
```

Out[124]:

	US_2017_CountyId	US_2017_State	US_2017_County	US_2017_TotalPop	US_2017_Men	US_2017_Wo
0	1001	Alabama	Autauga County	55036	26899	28
1	1003	Alabama	Baldwin County	203360	99527	103
2	1005	Alabama	Barbour County	26201	13976	12
3	1007	Alabama	Bibb County	22580	12251	10
4	1009	Alabama	Blount County	57667	28490	29

5 rows × 37 columns

In [125…

```
# Grouping the data based on US_2017_state and US_2017_county column.

vars_to_merge = [x for x in data_census_2017.columns if x not in ['US_2017_CountyId',

data_census_agg = pd.DataFrame(data_census_2017.groupby(['US_2017_State', 'US_2017_Cou
data_census_agg.head(5)
```

Out[125]:

US_2017_State	US_2017_County	US_2017_TotalPop	US_2017_Men	US_2017_Women	US_2017_Hispanic
Alabama	**Autauga County**	55036	26899	28137	2.7
	Baldwin County	203360	99527	103833	4.4
	Barbour County	26201	13976	12225	4.2
	Bibb County	22580	12251	10329	2.4
	Blount County	57667	28490	29177	9.0

5 rows × 34 columns

In [129…

```
# check duplicates if any in the dataset
data_census_agg.duplicated().sum()

# There is no duplicate rows in data based on the columns present in the dataset.
```

Out[129]:

```
0
```

In [130…

```
# summary of zero values
zero_values = (data_census_agg == 0).sum()
print("Zero values:\n", zero_values)

# There are several columns in the dataset wth 0 values. However, when considering the
# will be good drop the rows that have 0 values. This because the TotalPop column repr
# and including rows with 0 values would not contribute meaningful insights to the ana
```

```
Zero values:
 US_2017_TotalPop              0
US_2017_Men                    0
US_2017_Women                  0
US_2017_Hispanic              13
US_2017_White                  3
US_2017_Black                189
US_2017_Native               481
US_2017_Asian                388
US_2017_Pacific             2393
US_2017_VotingAgeCitizen       0
US_2017_Income                 0
US_2017_IncomeErr              0
US_2017_IncomePerCap           0
US_2017_IncomePerCapErr        0
US_2017_Poverty                0
US_2017_ChildPoverty           9
US_2017_Professional           0
US_2017_Service                1
US_2017_Office                 0
US_2017_Construction           1
US_2017_Production             1
US_2017_Drive                  0
US_2017_Carpool                1
US_2017_Transit              640
US_2017_Walk                  14
US_2017_OtherTransp           72
US_2017_WorkAtHome             6
US_2017_MeanCommute            0
US_2017_Employed               0
US_2017_PrivateWork            0
US_2017_PublicWork             0
US_2017_SelfEmployed           1
US_2017_FamilyWork           610
US_2017_Unemployment          11
dtype: int64
```

In [131…

```
# since population percentage of 'Native' and 'Pacific' is very less, we can merge the
data_census_agg['OtherRace'] = data_census_agg['US_2017_Native'] + data_census_agg['US
data_census_agg.drop(['US_2017_Native', 'US_2017_Pacific'], axis=1, inplace=True)
data_census_agg.head()
```

Out[131]:

US_2017_State	US_2017_County	US_2017_TotalPop	US_2017_Men	US_2017_Women	US_2017_Hispanic
Alabama	**Autauga County**	55036	26899	28137	2.7
	Baldwin County	203360	99527	103833	4.4
	Barbour County	26201	13976	12225	4.2
	Bibb County	22580	12251	10329	2.4
	Blount County	57667	28490	29177	9.0

5 rows × 33 columns

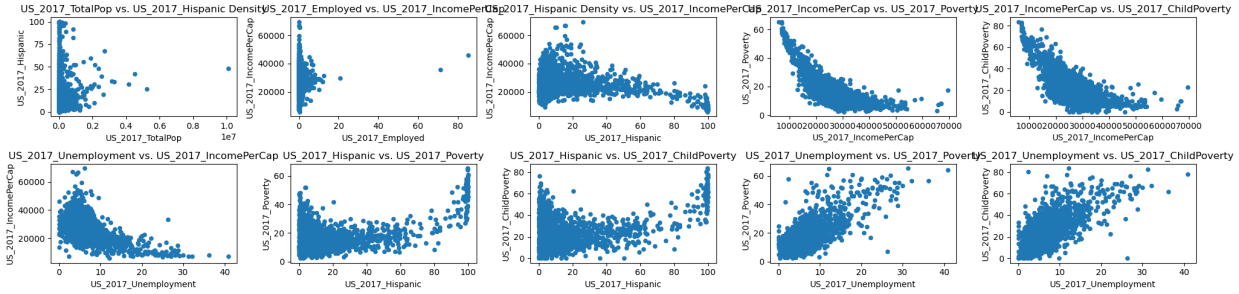
In [137…
```
# change absolute columns to percentage
absolutes = ['US_2017_Men','US_2017_Women','US_2017_VotingAgeCitizen','US_2017_Employe
data_census_agg[absolutes] = data_census_agg[absolutes].div(data_census_agg["US_2017_T
data_census_agg.head()
```

Out[137]:

US_2017_State	US_2017_County	US_2017_TotalPop	US_2017_Men	US_2017_Women	US_2017_Hispanic
Alabama	Autauga County	55036	0.088806	0.092893	2.7
	Baldwin County	203360	0.024066	0.025108	4.4
	Barbour County	26201	0.203586	0.178079	4.2
	Bibb County	22580	0.240283	0.202586	2.4
	Blount County	57667	0.085672	0.087738	9.0

5 rows × 33 columns

In [140…
```
# The plot shows how all these variables related each other.We can use this plot to un
# between demographic variables and find outliers. We can inspect the supected data po
fig, axes = plt.subplots(nrows=2, ncols=5)
fig.set_figheight(5)
fig.set_figwidth(20)

data_census_agg.plot(ax=axes[0,0],x='US_2017_TotalPop', y='US_2017_Hispanic', kind = '
data_census_agg.plot(ax=axes[0,1],x='US_2017_Employed', y='US_2017_IncomePerCap', kind
data_census_agg.plot(ax=axes[0,2],x='US_2017_Hispanic', y='US_2017_IncomePerCap', kind
data_census_agg.plot(ax=axes[0,3], x='US_2017_IncomePerCap', y='US_2017_Poverty', kind
data_census_agg.plot(ax=axes[0,4], x='US_2017_IncomePerCap', y='US_2017_ChildPoverty',
data_census_agg.plot(ax=axes[1,0], x='US_2017_Unemployment', y='US_2017_IncomePerCap',
data_census_agg.plot(ax=axes[1,1], x='US_2017_Hispanic', y='US_2017_Poverty', kind = '
data_census_agg.plot(ax=axes[1,2], x='US_2017_Hispanic', y='US_2017_ChildPoverty', kin
data_census_agg.plot(ax=axes[1,3], x='US_2017_Unemployment', y='US_2017_Poverty', kind
data_census_agg.plot(ax=axes[1,4], x='US_2017_Unemployment', y='US_2017_ChildPoverty',

plt.tight_layout()
```



Milestone-3

Apply 5 transformations to https://simple.wikipedia.org/wiki/List_of_U.S._states data

Check duplicates in the data based on states name

In [5]:
```
import pandas as pd
import requests
from bs4 import BeautifulSoup
# URL of the Wikipedia page of List of US States
url = "https://simple.wikipedia.org/wiki/List_of_U.S._states"

# Request to fetch the page content
response = requests.get(url)

if response.status_code == 200:
# Parse the page content with BeautifulSoup
soup = BeautifulSoup(response.text, "html.parser")

# Find the table containing the list of U.S. states
state_table = soup.find("table", {"class": "wikitable"})

if state_table:
# Initialize a list to store state names
        state_names = []

# Extract state names from the table
        for row in state_table.find_all("tr"):
            cells = row.find_all("td")
            if len(cells) > 1:
                state_name = cells[0].get_text(strip=True)
                state_names.append(state_name)

# Find duplicates in the list of state names
        duplicates = [name for name in state_names if state_names.count(name) > 1]

if duplicates:
    print("Duplicate state names:")
    for duplicate in set(duplicates):
        print(duplicate)
else:
            print("No duplicate state names found.")
else:
        print("No state table found on the page.")
else:
    print("Failed to retrieve the page. Status code:", response.status_code)
```

```
No duplicate state names found.
```

Scrape data with specific headers from the website data

In [24]:
```
# Check request status
if response.status_code == 200:
# Parse the HTML content of the page
soup = BeautifulSoup(response.text, 'html.parser')

# Find the table containing the data
    table = soup.find("table")

# Initialize a list to store the state data
    state_data = []

 # Iterate through the rows of the table skipping the header row
    for row in table.find_all("tr")[1:]:
```

```
        columns = row.find_all("td")
        if len(columns) >= 4:
            # Extract the data based on the headers
            state_name = columns[0].text.strip()
            capital = columns[1].text.strip()
            established_date = columns[3].text.strip()

 # Create a dictionary for each state
            state_info = {
                "State": state_name,
                "Capital": capital,
                "Established": established_date
            }

# Append the state dictionary to the state_data list
            state_data.append(state_info)

# Print the scraped data with headers
    for state in state_data:
        print("State:", state["State"])
        print("Capital:", state["Capital"])
        print("Established:", state["Established"])
        print("\n")
else:
    print("Failed to retrieve the web page. Status code:", response.status_code)
```

```
State: AL
Capital: Montgomery
Established: Dec 14, 1819


State: AK
Capital: Juneau
Established: Jan 3, 1959


State: AZ
Capital: Phoenix
Established: 7,151,502


State: AR
Capital: Little Rock
Established: 3,011,524


State: CA
Capital: Sacramento
Established: Sep 9, 1850


State: CO
Capital: Denver
Established: 5,773,714


State: CT
Capital: Hartford
Established: Jan 9, 1788


State: DE
Capital: Dover
Established: Dec 7, 1787


State: FL
Capital: Tallahassee
Established: Mar 3, 1845


State: GA
Capital: Atlanta
Established: 10,711,908


State: HI
Capital: Honolulu
Established: 1,455,271


State: ID
Capital: Boise
Established: 1,839,106
```

```
State: IL
Capital: Springfield
Established: Dec 3, 1818


State: IN
Capital: Indianapolis
Established: 6,785,528


State: IA
Capital: Des Moines
Established: 3,190,369


State: KS
Capital: Topeka
Established: Jan 29, 1861


State: KY
Capital: Frankfort
Established: Jun 1, 1792


State: LA
Capital: Baton Rouge
Established: Apr 30, 1812


State: ME
Capital: Augusta
Established: Mar 15, 1820


State: MD
Capital: Annapolis
Established: Apr 28, 1788


State: MA
Capital: Boston
Established: 7,029,917


State: MI
Capital: Lansing
Established: Jan 26, 1837


State: MN
Capital: Saint Paul
Established: May 11, 1858


State: MS
Capital: Jackson
Established: 2,961,279
```

```
State: MO
Capital: Jefferson City
Established: Aug 10, 1821


State: MT
Capital: Helena
Established: Nov 8, 1889


State: NE
Capital: Lincoln
Established: Mar 1, 1867


State: NV
Capital: Carson City
Established: Oct 31, 1864


State: NH
Capital: Concord
Established: Jun 21, 1788


State: NJ
Capital: Trenton
Established: Dec 18, 1787


State: NM
Capital: Santa Fe
Established: Jan 6, 1912


State: NY
Capital: Albany
Established: Jul 26, 1788


State: NC
Capital: Raleigh
Established: Nov 21, 1789


State: ND
Capital: Bismarck
Established: Nov 2, 1889


State: OH
Capital: Columbus
Established: 11,799,448


State: OK
Capital: Oklahoma City
Established: 3,959,353
```

```
State: OR
Capital: Salem
Established: Feb 14, 1859


State: PA
Capital: Harrisburg
Established: Dec 12, 1787


State: RI
Capital: Providence
Established: 1,097,379


State: SC
Capital: Columbia
Established: May 23, 1788


State: SD
Capital: Pierre
Established: Nov 2, 1889


State: TN
Capital: Nashville
Established: 6,910,840


State: TX
Capital: Austin
Established: Dec 29, 1845


State: UT
Capital: Salt Lake City
Established: 3,271,616


State: VT
Capital: Montpelier
Established: Mar 4, 1791


State: VA
Capital: Richmond
Established: Jun 25, 1788


State: WA
Capital: Olympia
Established: Nov 11, 1889


State: WV
Capital: Charleston
Established: 1,793,716
```

```
State: WI
Capital: Madison
Established: May 29, 1848


State: WY
Capital: Cheyenne
Established: 576,851
```

scrape the specified columns data and present it in a more readable tabular format

In [35]:

```
# Send an HTTP GET request to the URL
response = requests.get(url)

# Check if the request was successful
if response.status_code == 200:
    # Parse the HTML content of the page
    soup = BeautifulSoup(response.text, 'html.parser')

    # Find the table containing the data (you may need to inspect the HTML to find the
    table = soup.find("table")

    # Initialize a list to store the state data
    state_data = []

    # Iterate through the rows of the table skipping the header row
    for row in table.find_all("tr")[1:]:
        columns = row.find_all("td")
        if len(columns) >= 4:
            # Extract the state information
            state_name = columns[0].text.strip()
            capital = columns[1].text.strip()
            Population = columns[4].text.strip()

            # Create a tuple for each state
            state_info = (state_name, capital, Population)

            # Append the state tuple to the state_data list
            state_data.append(state_info)

# Print the state data in a more readable tabular format with State, Capital, and Popu
    print("{:<30} {:<30} {:<20}".format("State", "Capital", "Population"))
    for state in state_data:
        state_name, capital, Population = state
        print("{:<30} {:<30} {:<20}".format(state_name, capital, Population))

else:
    print("Failed to retrieve the web page. Status code:", response.status_code)
```

State	Capital	Population
AL	Montgomery	5,024,279
AK	Juneau	733,391
AZ	Phoenix	113,990
AR	Little Rock	53,179
CA	Sacramento	39,538,223
CO	Denver	104,094
CT	Hartford	3,605,944
DE	Dover	989,948
FL	Tallahassee	21,538,187
GA	Atlanta	59,425
HI	Honolulu	10,932
ID	Boise	83,569
IL	Springfield	12,812,508
IN	Indianapolis	36,420
IA	Des Moines	56,273
KS	Topeka	2,937,880
KY	Frankfort	4,505,836
LA	Baton Rouge	4,657,757
ME	Augusta	1,362,359
MD	Annapolis	6,177,224
MA	Boston	10,554
MI	Lansing	10,077,331
MN	Saint Paul	5,706,494
MS	Jackson	48,432
MO	Jefferson City	6,154,913
MT	Helena	1,084,225
NE	Lincoln	1,961,504
NV	Carson City	3,104,614
NH	Concord	1,377,529
NJ	Trenton	9,288,994
NM	Santa Fe	2,117,522
NY	Albany	20,201,249
NC	Raleigh	10,439,388
ND	Bismarck	779,094
OH	Columbus	44,826
OK	Oklahoma City	69,899
OR	Salem	4,237,256
PA	Harrisburg	13,002,700
RI	Providence	1,545
SC	Columbia	5,118,425
SD	Pierre	886,667
TN	Nashville	42,144
TX	Austin	29,145,505
UT	Salt Lake City	84,897
VT	Montpelier	643,077
VA	Richmond	8,631,393
WA	Olympia	7,705,281
WV	Charleston	24,230
WI	Madison	5,893,718
WY	Cheyenne	97,813

Fix casing or inconsistent values in https://simple.wikipedia.org/wiki/List_of_U.S._states

In [40]:
```
# Check if the request was successful
if response.status_code == 200:
    # Parse the HTML content of the page
    soup = BeautifulSoup(response.text, 'html.parser')
```

```
# Find the table containing the data
    table = soup.find("table")

# Initialize a list to store the state data
    state_data = []

# Iterate through the rows of the table skipping header
    for row in table.find_all("tr")[1:]:
        columns = row.find_all("td")
        if len(columns) >= 4:
            # Extract the state information
            state_name = columns[0].text.strip().title()  # Convert to title case
            capital = columns[1].text.strip().title()  # Convert to title case
            established_date = columns[3].text.strip()

 # Create a tuple for each state
            state_info = (state_name, capital, established_date)

 # Append the state tuple to the state_data list
            state_data.append(state_info)

# Print the state data with standardized casing
    print("{:<30} {:<30} {:<20}".format("State", "Capital", "Established Date"))
    for state in state_data:
        state_name, capital, established_date = state
        print("{:<30} {:<30} {:<20}".format(state_name, capital, established_date))

else:
    print("Failed to retrieve the web page. Status code:", response.status_code)
```

State	Capital	Established Date
Al	Montgomery	Dec 14, 1819
Ak	Juneau	Jan 3, 1959
Az	Phoenix	7,151,502
Ar	Little Rock	3,011,524
Ca	Sacramento	Sep 9, 1850
Co	Denver	5,773,714
Ct	Hartford	Jan 9, 1788
De	Dover	Dec 7, 1787
Fl	Tallahassee	Mar 3, 1845
Ga	Atlanta	10,711,908
Hi	Honolulu	1,455,271
Id	Boise	1,839,106
Il	Springfield	Dec 3, 1818
In	Indianapolis	6,785,528
Ia	Des Moines	3,190,369
Ks	Topeka	Jan 29, 1861
Ky	Frankfort	Jun 1, 1792
La	Baton Rouge	Apr 30, 1812
Me	Augusta	Mar 15, 1820
Md	Annapolis	Apr 28, 1788
Ma	Boston	7,029,917
Mi	Lansing	Jan 26, 1837
Mn	Saint Paul	May 11, 1858
Ms	Jackson	2,961,279
Mo	Jefferson City	Aug 10, 1821
Mt	Helena	Nov 8, 1889
Ne	Lincoln	Mar 1, 1867
Nv	Carson City	Oct 31, 1864
Nh	Concord	Jun 21, 1788
Nj	Trenton	Dec 18, 1787
Nm	Santa Fe	Jan 6, 1912
Ny	Albany	Jul 26, 1788
Nc	Raleigh	Nov 21, 1789
Nd	Bismarck	Nov 2, 1889
Oh	Columbus	11,799,448
Ok	Oklahoma City	3,959,353
Or	Salem	Feb 14, 1859
Pa	Harrisburg	Dec 12, 1787
Ri	Providence	1,097,379
Sc	Columbia	May 23, 1788
Sd	Pierre	Nov 2, 1889
Tn	Nashville	6,910,840
Tx	Austin	Dec 29, 1845
Ut	Salt Lake City	3,271,616
Vt	Montpelier	Mar 4, 1791
Va	Richmond	Jun 25, 1788
Wa	Olympia	Nov 11, 1889
Wv	Charleston	1,793,716
Wi	Madison	May 29, 1848
Wy	Cheyenne	576,851

Identify outliers and bad data from https://simple.wikipedia.org/wiki/List_of_U.S._states

In [45]:
```
# Check if the request was successful
if response.status_code == 200:
    # Parse the HTML content of the page
    soup = BeautifulSoup(response.text, 'html.parser')
```

```
    # Find the table containing the data
    table = soup.find("table")

    # Initialize lists to store missing data and data inconsistencies
    missing_data = []
    data_inconsistencies = []

    # Iterate through the rows of the table skipping header row
    for row in table.find_all("tr")[1:]:
        columns = row.find_all("td")
        if len(columns) >= 4:
            # Extract the state information
            state_name = columns[0].text.strip()
            capital = columns[1].text.strip()
            largest_city = columns[2].text.strip()
            established_date = columns[3].text.strip()

            # Check for missing data
            if not state_name or not capital or not largest_city or not established_da
                missing_data.append(state_name)

# Print missing or data inconsistency issues
    if missing_data:
        print("Missing data:")
        for state in missing_data:
            print(state)

    if data_inconsistencies:
        print("Data inconsistencies:")
        for inconsistency in data_inconsistencies:
            print(inconsistency)

else:
    print("Failed to retrieve the web page. Status code:", response.status_code)
```

Top 10 states based on Population from https://simple.wikipedia.org/wiki/List_of_U.S._states#List_of_states

In [48]:

```
# Check if the request was successful
if response.status_code == 200:
    # Parse the HTML content of the page
    soup = BeautifulSoup(response.text, 'html.parser')

# Find the table containing the data (you may need to inspect the HTML to find the cor
    table = soup.find("table")

# Initialize a list to store the state data
    state_data = []

        for row in table.find_all("tr")[1:]:
        columns = row.find_all("td")
        if len(columns) >= 6:
            # Extract the state information
            state_name = columns[0].text.strip()
            capital = columns[2].text.strip()
            population = columns[4].text.strip()
```

```
# Check population is numeric and remove "," from data
            try:
                population = int(population.replace(',', ''))
            except ValueError:
                population = 0

# Create a tuple for each state
            state_info = (state_name, capital, population)

# Append the state tuple to the state_data list
            state_data.append(state_info)

# Sort the states by population in descending order
    state_data.sort(key=lambda x: x[2], reverse=True)

    # Print the top 10 states based on population
    print("{:<30} {:<30} {:<20}".format("State", "Capital", "Population"))
    for state in state_data[:10]:
        state_name, capital, population = state
        print("{:<30} {:<30} {:<20}".format(state_name, capital, population))

else:
    print("Failed to retrieve the web page. Status code:", response.status_code)
```

State	Capital	Population
CA	Los Angeles	39538223
TX	Houston	29145505
FL	Jacksonville	21538187
NY	New York City	20201249
PA	Philadelphia	13002700
IL	Chicago	12812508
NC	Charlotte	10439388
MI	Detroit	10077331
NJ	Newark	9288994
VA	Virginia Beach	8631393

DSC-540 Project Milestone-4

Chitramoy Mukherjee

Apply 5 transformations to https://api.census.gov/data/2017/ecnclcust

In [3]:
```
# Import modules
import matplotlib.pyplot as plt
import pandas as pd
import os
import requests
```

In [4]:
```
 # Construct the full API link using the API_KEY received through key request

def generate_api_link(api_key, dataset, variables, geography):
    base_url = "https://api.census.gov/data"
    endpoint = f"{dataset}"
    params = {
        "get": f"{variables}",
        "for": f"{geography}",
        "key": api_key
```

```
    }

    api_link = f"{base_url}/{endpoint}?{'&'.join([f'{k}={v}' for k, v in params.items(

    return api_link

if __name__ == "__main__":
    census_api_key = "6f54e9fd4f7eef82cba525eb1d738c0da4048c66"
    census_dataset = "2017/ecnclcust"
    census_variables = "NAICS2017_LABEL,NAME,GEO_ID,TYPOP,TYPOP_LABEL,TAXSTAT_LABEL,TA
    census_geography = "state:*&NAICS2017"

    api_link = generate_api_link(census_api_key, census_dataset, census_variables, cer

    census_data = fetch_census_data(census_api_url)

    if census_data:

        for record in census_data[1:6]:  # Printing first 5 rows , skipping the header
            print(record)
    else:
        print("No data retrieved.")
```

```
['Legal services', 'Alaska', '0400000US02', '00', 'All establishments', 'Establishmen
ts subject to federal income tax', 'T', '270019', '5411', '02']
['Legal services', 'Alabama', '0400000US01', '00', 'All establishments', 'Establishme
nts subject to federal income tax', 'T', '2759330', '5411', '01']
['Legal services', 'Wyoming', '0400000US56', '00', 'All establishments', 'Establishme
nts subject to federal income tax', 'T', '266271', '5411', '56']
['Legal services', 'Arkansas', '0400000US05', '00', 'All establishments', 'Establishm
ents subject to federal income tax', 'T', '0', '5411', '05']
['Legal services', 'Arizona', '0400000US04', '00', 'All establishments', 'Establishme
nts subject to federal income tax', 'T', '3485673', '5411', '04']
```

Census Data API: Variables in /data/2017/ecnbasic/variables

NAICS2017_LABEL : Type of Business/Service

NAME : State Name

GEO_ID : Geographic identifier code

TYPOP : Type of operation code

TYPOP_LABEL : Wholesale Trade

TAXSTAT_LABEL : Wholesale Trad labels

TAXSTAT : Tax status code

RCPTOT : Sales, value of shipments, or revenue

NAICS2017 : 2017 NAICS code

state : State code

Ethical implecations on census API data

Census data is often used to allocate resources, determine political representation, and make policy decisions. There is an ethical obligation to ensure that the data collection and analysis processes are fair, unbiased, and do not disproportionately disadvantage specific communities or demographics.

Openness and transparency in the data collection and analysis processes are critical. The public should have access to information about how the census is conducted, what data is collected, and how it is used.

There is an ethical responsibility to collect accurate and reliable data. Inaccurate data can lead to incorrect policy decisions, misallocation of resources, and unfair treatment of certain groups or regions.

One of the primary ethical considerations is the protection of individual privacy. Census data often includes personal information, and there is a responsibility to ensure that the data is collected, stored, and used in a way that respects individuals' privacy rights.

In [7]:
```
# Find duplicates based on 'NAICS2017_LABEL', 'NAME', 'GEO_ID','CLASSCUST_LABEL','CLAS

def fetch_census_data(api_url):
    response = requests.get(api_url)

    if response.status_code == 200:
        data = response.json()
        return data
    else:
        print(f"Error: {response.status_code}")
        return None

if __name__ == "__main__":
    census_api_url = "https://api.census.gov/data/2017/ecnclcust?get=NAICS2017_LABEL,N

    census_data = fetch_census_data(census_api_url)

    if census_data:
        df = pd.DataFrame(census_data[1:], columns=census_data[0])
# Select columns for finding duplicates
        selected_columns = ['NAICS2017_LABEL', 'NAME', 'GEO_ID', 'state']


        duplicates = df[df.duplicated(subset=selected_columns, keep=False)] # Find and

        if not duplicates.empty:
            print("Duplicate Records:")
            print(duplicates)
        else:
            print("No duplicates found.")
    else:
        print("No data retrieved.")
```

```
No duplicates found.
```

In [ ]:
```
# Check for Null value in key fields NAME and state column
# Make the API request
response = requests.get(url, params=params)
```

```
# Check if the request was successful (status code 200)
if response.status_code == 200:
    # Parse the JSON response
    data = response.json()

    # Create a DataFrame from the API response
    df = pd.DataFrame(data[1:], columns=data[0])

    # Check for null values in 'NAME' and 'state' columns
    null_values_name = df['NAME'].isnull().sum()
    null_values_state = df['state'].isnull().sum()

    print(f"Null values in 'NAME': {null_values_name}")
    print(f"Null values in 'state': {null_values_state}")

else:
    print(f"Error: {response.status_code}")
    print(response.text)
```

In [ ]:

```
# Rename the NAME column as STATE_NAME and state AS STATE_CD and display data
# Make the API request
response = requests.get(url, params=params)

# Check if the request was successful (status code 200)
if response.status_code == 200:
    # Parse the JSON response
    data = response.json()

    # Create a DataFrame from the API response
    df = pd.DataFrame(data[1:], columns=data[0])

    # Handle missing values by dropping rows with missing values
    df = df.dropna()

    # Rename columns
    df = df.rename(columns={'NAME': 'STATE_NAME', 'state': 'STATE_CD'})

    # Display the first 100 rows of the transformed DataFrame
    print(df.head())

else:
    print(f"Error: {response.status_code}")
    print(response.text)
```

In [ ]:
```
# Convert numeric columns to appropriate types

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

# Create a DataFrame from the API response
df = pd.DataFrame(data[1:], columns=data[0])

numeric_columns = ['TYPOP', 'TAXSTAT', 'RCPTOT']
df[numeric_columns] = df[numeric_columns].apply(pd.to_numeric, errors='coerce')


print("Summary Statistics:")# Display summary statistics to identify outliers
print(df.describe())
```

In [ ]:
```
# Make the API request
response = requests.get(url, params=params)

# Check if the request was successful (status code 200)
if response.status_code == 200:
    # Parse the JSON response
    data = response.json()

# Create a DataFrame from the API response
    df = pd.DataFrame(data[1:], columns=data[0])

# Convert string columns to lowercase for consistency
    string_columns = ['NAICS2017_LABEL', 'NAME', 'TYPOP_LABEL', 'TAXSTAT_LABEL']
    df[string_columns] = df[string_columns].apply(lambda x: x.str.lower())

# Display the first few records of the DataFrame
    print(df.head())

else:
    print(f"Error: {response.status_code}")
    print(response.text)
```

DSC-540 Project Milestone-5

Chitramoy Mukherjee

Store various source data into sqlite DB and Create visualiztion using Python plots

In [16]:
```
import pandas as pd
import sqlite3
import requests
```

In [19]:
```
# Load acs2017_county_data.csv data into acs2017_county_data table in dsc540.db sqlite

# Set the csv_file_path and sqlite_db_path variable
csv_file_path = 'C:\\Users\\14024\\OneDrive\\Desktop\\MS-DSC\\DSC-540\\DSC-540 Project
sqlite_db_path = 'C:\\Users\\14024\sqllite3\\Mydatabase\\dsc540.db'

# Read data from CSV file into a pandas DataFrame
```

```
df = pd.read_csv(csv_file_path)

# Connect to SQLite database
conn = sqlite3.connect(sqlite_db_path)

# Write the data to the SQLite table
df.to_sql('acs2017_county_data', conn, index=False, if_exists='replace')

# Commit and close the connection
conn.commit()
conn.close()

# Read data from CSV file into a pandas DataFrame
df = pd.read_csv(csv_file_path)

# Connect to SQLite database
conn = sqlite3.connect(sqlite_db_path)

# Write the data to the SQLite table
df.to_sql('acs2017_county_data', conn, index=False, if_exists='replace')

# Commit and close the connection
conn.commit()
conn.close()

# Connect to SQLite database again
conn = sqlite3.connect(sqlite_db_path)

# Select 10 rows from the table
query = "SELECT * FROM acs2017_county_data LIMIT 10;"
result = pd.read_sql_query(query, conn)

# Display the result
print(result)

# Close the connection
conn.close()
```

```
   CountyId    State           County  TotalPop    Men   Women  Hispanic  \
0      1001  Alabama   Autauga County      55036  26899   28137       2.7
1      1003  Alabama   Baldwin County     203360  99527  103833       4.4
2      1005  Alabama   Barbour County      26201  13976   12225       4.2
3      1007  Alabama      Bibb County      22580  12251   10329       2.4
4      1009  Alabama    Blount County      57667  28490   29177       9.0
5      1011  Alabama   Bullock County      10478   5616    4862       0.3
6      1013  Alabama    Butler County      20126   9416   10710       0.3
7      1015  Alabama   Calhoun County     115527  55593   59934       3.6
8      1017  Alabama  Chambers County      33895  16320   17575       2.2
9      1019  Alabama  Cherokee County      25855  12862   12993       1.6

   White  Black  Native  ...  Walk  OtherTransp  WorkAtHome  MeanCommute  \
0   75.4   18.9     0.3  ...   0.6          1.3         2.5         25.8
1   83.1    9.5     0.8  ...   0.8          1.1         5.6         27.0
2   45.7   47.8     0.2  ...   2.2          1.7         1.3         23.4
3   74.6   22.0     0.4  ...   0.3          1.7         1.5         30.0
4   87.4    1.5     0.3  ...   0.4          0.4         2.1         35.0
5   21.6   75.6     1.0  ...   6.2          1.7         3.0         29.8
6   52.2   44.7     0.1  ...   0.9          0.9         2.0         23.2
7   72.7   20.4     0.2  ...   1.3          1.1         3.2         24.8
8   56.2   39.3     0.3  ...   0.6          0.5         2.0         23.6
9   91.8    5.0     0.5  ...   0.3          0.3         2.0         26.5

   Employed  PrivateWork  PublicWork  SelfEmployed  FamilyWork  Unemployment
0     24112         74.1        20.2           5.6         0.1           5.2
1     89527         80.7        12.9           6.3         0.1           5.5
2      8878         74.1        19.1           6.5         0.3          12.4
3      8171         76.0        17.4           6.3         0.3           8.2
4     21380         83.9        11.9           4.0         0.1           4.9
5      4290         81.4        13.6           5.0         0.0          12.1
6      7727         79.1        15.3           5.3         0.3           7.6
7     47392         74.9        19.9           5.1         0.1          10.1
8     14527         84.5        11.8           3.7         0.0           6.4
9      9879         74.8        17.1           8.1         0.0           5.3

[10 rows x 37 columns]
```

In [20]:

```
# Exported the https://simple.wikipedia.org/wiki/List_of_U.S._states data into excel f
# us_state_detl table in dsc540.db sqlite db and display 10 rows.

# Excel file path
excel_file_path = 'C:\\Users\\14024\\OneDrive\\Desktop\\MS-DSC\\DSC-540\\DSC-540 Proje

# Read Excel file into a pandas DataFrame
df = pd.read_excel(excel_file_path)

# Connect to SQLite database
conn = sqlite3.connect('C:\\Users\\14024\sqllite3\\Mydatabase\\dsc540.db')
cursor = conn.cursor()

# Create the table if it doesn't exist
cursor.execute('''
    CREATE TABLE IF NOT EXISTS us_state_detl (
        State TEXT PRIMARY KEY,
        State_cd TEXT,
        Capital TEXT,
        Largest_City TEXT,
        Population INTEGER,
        Total_Area INTEGER,
```

```
        Land_Area INTEGER,
        Water_Area INTEGER,
        Number_of_Reps INTEGER
    )
''')

# Insert data into the table
df.to_sql('us_state_detl', conn, if_exists='replace', index=False)

# Select 10 rows from the table
query = "SELECT *  FROM us_state_detl limit 10;"
result = pd.read_sql_query(query, conn)

# Display the result
print(result)

# Commit changes and close the connection
conn.commit()
conn.close()
```

```
         State State_cd      Capital  Largest_City  Population  Total_Area  \
0      Alabama       AL   Montgomery    Huntsville     5024279       52420
1       Alaska       AK       Juneau     Anchorage      733391      665384
2      Arizona       AZ      Phoenix       Phoenix     7151502      113990
3     Arkansas       AR  Little Rock   Little Rock     3011524       53179
4   California       CA   Sacramento   Los Angeles    39538223      163695
5     Colorado       CO       Denver        Denver     5773714      104094
6  Connecticut       CT     Hartford    Bridgeport     3605944        5543
7     Delaware       DE        Dover    Wilmington      989948        2489
8      Florida       FL  Tallahassee  Jacksonville    21538187       65758
9      Georgia       GA      Atlanta       Atlanta    10711908       59425

   Land_Area  Water_Area  Number_of_Reps
0      50645        1775               7
1     570641       94743               1
2     113594         396               9
3      52035        1143               4
4     155779        7916              52
5     103642         452               8
6       4842         701               5
7       1949         540               1
8      53625       12133              28
9      57513        1912              14
```

In [21]:

```
# Load the API data (https://api.census.gov/data/2017/ecnclcust) into api_census_2017_
# and display 10 rows.

# Census API endpoint URL
url = "https://api.census.gov/data/2017/ecnclcust"

# API parameters
params = {
    'get': 'NAICS2017_LABEL,NAME,GEO_ID,TYPOP,TYPOP_LABEL,TAXSTAT_LABEL,TAXSTAT,RCPTOT
    'for': 'state:*',
    'NAICS2017': '5411',
    'key': '6f54e9fd4f7eef82cba525eb1d738c0da4048c66'
}

# Fetch data from the API
response = requests.get(url, params=params)
```

```
data = response.json()

# Connect to SQLite database
conn = sqlite3.connect('C:\\Users\\14024\sqllite3\\Mydatabase\\dsc540.db')
cursor = conn.cursor()

# Create the table if it doesn't exist
cursor.execute('''
    CREATE TABLE IF NOT EXISTS api_census_2017_data (
        NAICS2017_LABEL TEXT,
        NAME TEXT,
        GEO_ID TEXT,
        TYPOP INTEGER,
        TYPOP_LABEL TEXT,
        TAXSTAT_LABEL TEXT,
        TAXSTAT INTEGER,
        RCPTOT INTEGER,
        NAICS2017 INTEGER,
        state TEXT
    )
''')

# Insert data into the table
for row in data[1:]:
    row.append(row.pop())  # Move 'state' to the last position
    cursor.execute('''
        INSERT INTO api_census_2017_data VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
    ''', row)

# Select 10 rows from the table
query = "SELECT * FROM api_census_2017_data LIMIT 10;"
result = pd.read_sql_query(query, conn)

# Display the result
print(result)

# Commit changes and close the connection
conn.commit()
conn.close()
```

```
  NAICS2017_LABEL                 NAME       GEO_ID  TYPOP  \
0  Legal services               Alaska  0400000US02      0
1  Legal services              Alabama  0400000US01      0
2  Legal services              Wyoming  0400000US56      0
3  Legal services             Arkansas  0400000US05      0
4  Legal services              Arizona  0400000US04      0
5  Legal services           California  0400000US06      0
6  Legal services             Colorado  0400000US08      0
7  Legal services          Connecticut  0400000US09      0
8  Legal services  District of Columbia  0400000US11      0
9  Legal services             Delaware  0400000US10      0

          TYPOP_LABEL                                    TAXSTAT_LABEL TAXSTAT  \
0  All establishments  Establishments subject to federal income tax       T
1  All establishments  Establishments subject to federal income tax       T
2  All establishments  Establishments subject to federal income tax       T
3  All establishments  Establishments subject to federal income tax       T
4  All establishments  Establishments subject to federal income tax       T
5  All establishments  Establishments subject to federal income tax       T
6  All establishments  Establishments subject to federal income tax       T
7  All establishments  Establishments subject to federal income tax       T
8  All establishments  Establishments subject to federal income tax       T
9  All establishments  Establishments subject to federal income tax       T

     RCPTOT  NAICS2017 state
0    270019       5411    02
1   2759330       5411    01
2    266271       5411    56
3         0       5411    05
4   3485673       5411    04
5  44033660       5411    06
6   4506165       5411    08
7   2926492       5411    09
8  16750810       5411    11
9         0       5411    10
```

In [22]:

```
# Select specific columns after mergeing data from acs2017_county_data, api_census_201
# dsc540.db sqlite db based on state column.

# Connect to SQLite database
conn = sqlite3.connect('C:\\Users\\14024\sqllite3\\Mydatabase\\dsc540.db')
cursor = conn.cursor()

# SQL query to merge data from three tables based on state column
query = '''
    SELECT
        census.NAICS2017_LABEL,
        census.NAME,
        census.GEO_ID,
        census.RCPTOT,
        acs.County,
        acs.TotalPop,
        acs.Carpool,
        acs.IncomePerCap,
        acs.PrivateWork,
        us.Capital,
        us.Largest_City,
        us.Population,
        us.Total_Area
    FROM
```

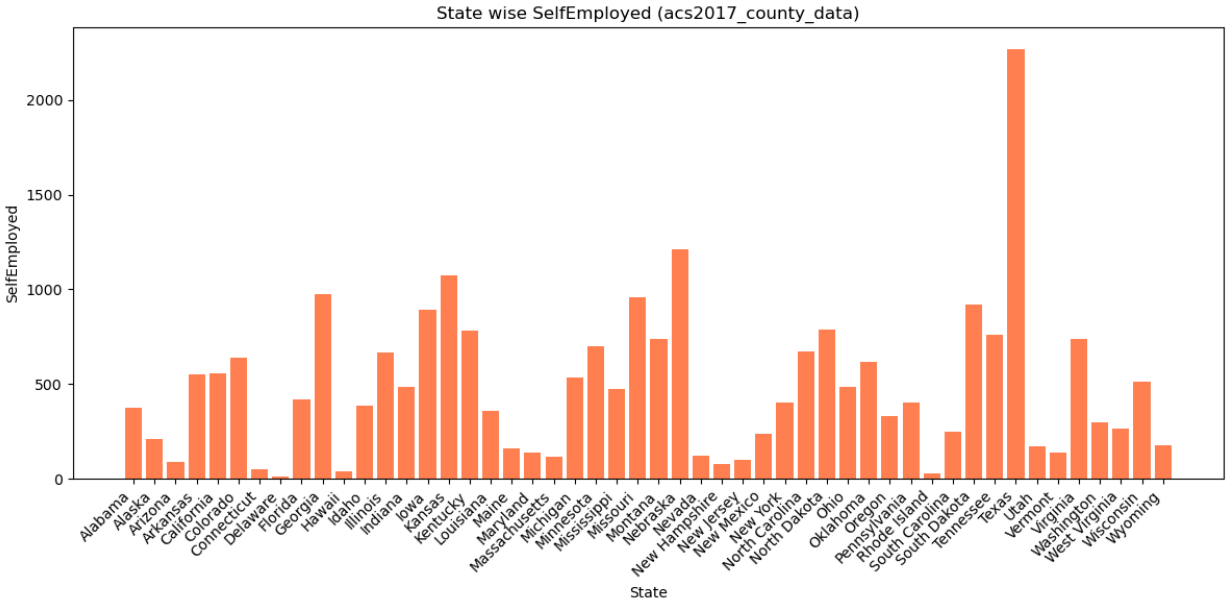
```
        api_census_2017_data census
    JOIN
        acs2017_county_data acs ON UPPER(TRIM(census.NAME)) = UPPER(TRIM(acs.State))
    JOIN
        us_state_detl us ON UPPER(TRIM(acs.State)) = UPPER(TRIM(us.State))
'''

# Execute the query and fetch the result into a pandas DataFrame
result_df = pd.read_sql_query(query, conn)

# Display the first 10 rows
print(result_df.head(10))

# Close the connection
conn.close()
```

```
  NAICS2017_LABEL    NAME       GEO_ID  RCPTOT                        County  \
0  Legal services  Alaska  0400000US02  270019         Aleutians East Borough
1  Legal services  Alaska  0400000US02  270019     Aleutians West Census Area
2  Legal services  Alaska  0400000US02  270019         Anchorage Municipality
3  Legal services  Alaska  0400000US02  270019             Bethel Census Area
4  Legal services  Alaska  0400000US02  270019            Bristol Bay Borough
5  Legal services  Alaska  0400000US02  270019                 Denali Borough
6  Legal services  Alaska  0400000US02  270019         Dillingham Census Area
7  Legal services  Alaska  0400000US02  270019  Fairbanks North Star Borough
8  Legal services  Alaska  0400000US02  270019                 Haines Borough
9  Legal services  Alaska  0400000US02  270019      Hoonah-Angoon Census Area

   TotalPop  Carpool  IncomePerCap  PrivateWork Capital Largest_City  \
0      3338      5.8         31254         70.9  Juneau    Anchorage
1      5784      9.0         35998         79.7  Juneau    Anchorage
2    298225     11.7         38977         73.2  Juneau    Anchorage
3     17957     13.6         18654         51.9  Juneau    Anchorage
4       917     16.0         42002         54.3  Juneau    Anchorage
5      2303      8.8         33084         74.9  Juneau    Anchorage
6      4974     18.3         24647         55.5  Juneau    Anchorage
7    100031     15.0         35328         65.4  Juneau    Anchorage
8      2537     10.3         35907         65.7  Juneau    Anchorage
9      2146      9.1         33704         60.3  Juneau    Anchorage

   Population  Total_Area
0      733391      665384
1      733391      665384
2      733391      665384
3      733391      665384
4      733391      665384
5      733391      665384
6      733391      665384
7      733391      665384
8      733391      665384
9      733391      665384
```

In [23]:

```
# Join acs2017_county_data and  us_state_detl based on State to visualize State wise S

# Connect to SQLite database
conn = sqlite3.connect('C:\\Users\\14024\sqllite3\\Mydatabase\\dsc540.db')
cursor = conn.cursor()

# SQL query to fetch data
query = '''
```

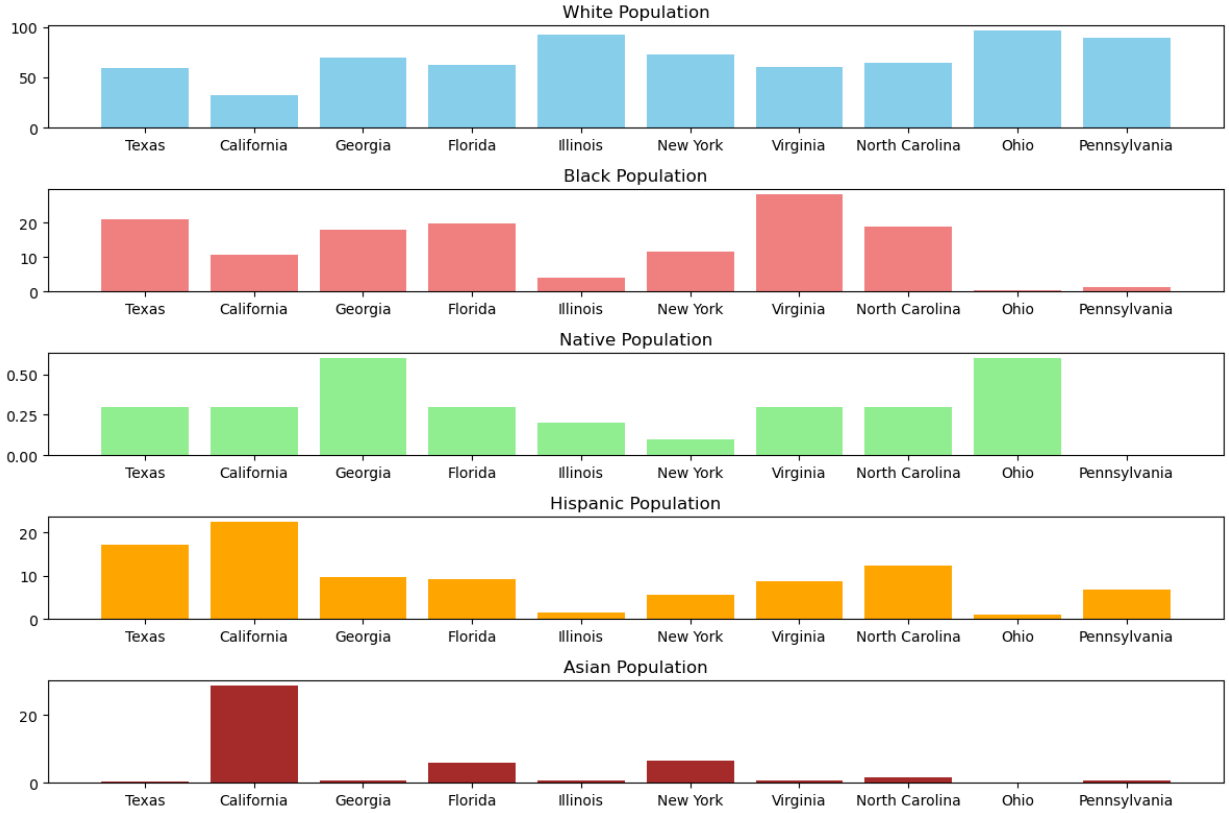
```
    SELECT
        SUM(c.SelfEmployed) as SelfEmployed  ,
        s.State
    FROM
        acs2017_county_data c
    JOIN
        us_state_detl s ON UPPER(TRIM(c.state)) = UPPER(TRIM(s.State))
        group by s.State
'''

# Execute the query and fetch the result into a pandas DataFrame
result_df = pd.read_sql_query(query, conn)

# Close the connection
conn.close()

# Plotting
plt.figure(figsize=(12, 6))
plt.bar(result_df['State'], result_df['SelfEmployed'], color='coral')
plt.title('State wise SelfEmployed (acs2017_county_data)')
plt.xlabel('State')
plt.ylabel('SelfEmployed')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()

# Show the plot
plt.show()
```



In [24]:

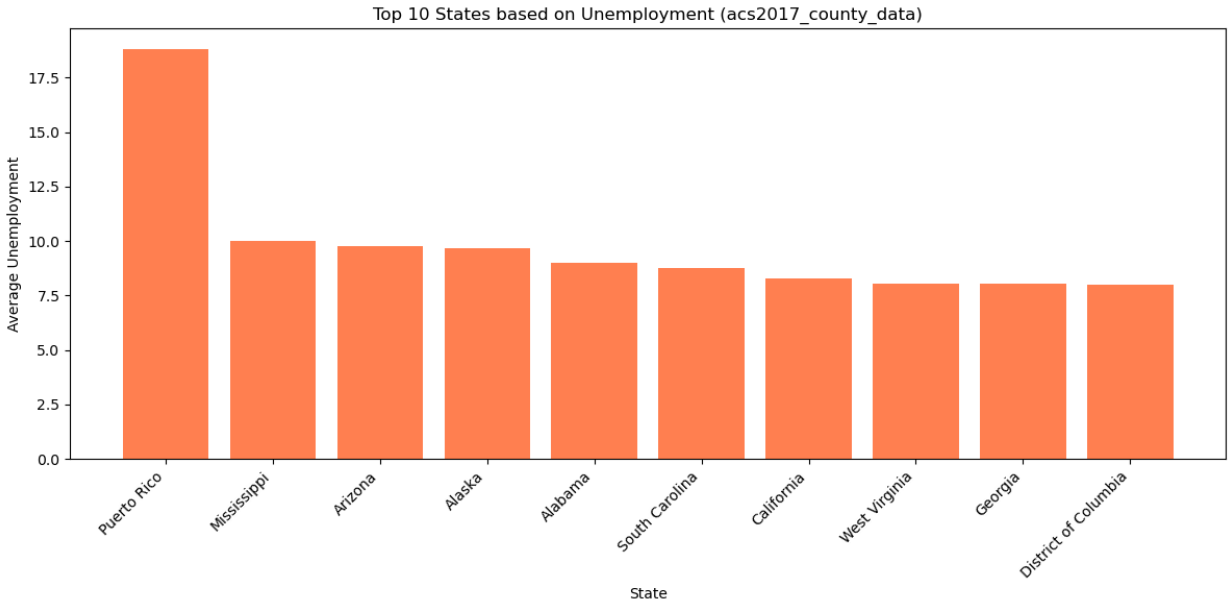
```
# Join acs2017_county_data and  us_state_detl based on State to White, Black, Native,
# for top 10 most populated state. .

# Connect to the SQLite database
conn = sqlite3.connect('C:\\Users\\14024\sqllite3\\Mydatabase\\dsc540.db')  # Replace

# Define the SQL query to select data from the tables and join them
query = '''
    SELECT acs.State, acs.White, acs.Black, acs.Native, acs.Hispanic, acs.Asian, us_st
    FROM acs2017_county_data acs
    JOIN us_state_detl ON acs.State = us_state_detl.State
'''
```

```
# Execute the query and fetch the data into a DataFrame
df = pd.read_sql_query(query, conn)

# Calculate the total population for each state
df['TotalPopulation'] = df['Population'].groupby(df['State']).transform('sum')

# Sort the data by the total population in descending order and select the top 10 stat
top_10_states = df.drop_duplicates(subset=['State']).nlargest(10, 'TotalPopulation')

# Plot bar diagrams for the top 10 states
plt.figure(figsize=(12, 8))

# Bar diagram for White population
plt.subplot(5, 1, 1)
plt.bar(top_10_states['State'], top_10_states['White'], color='skyblue')
plt.title('White Population')

# Bar diagram for Black population
plt.subplot(5, 1, 2)
plt.bar(top_10_states['State'], top_10_states['Black'], color='lightcoral')
plt.title('Black Population')

# Bar diagram for Native population
plt.subplot(5, 1, 3)
plt.bar(top_10_states['State'], top_10_states['Native'], color='lightgreen')
plt.title('Native Population')

# Bar diagram for Hispanic population
plt.subplot(5, 1, 4)
plt.bar(top_10_states['State'], top_10_states['Hispanic'], color='orange')
plt.title('Hispanic Population')

# Bar diagram for Asian population
plt.subplot(5, 1, 5)
plt.bar(top_10_states['State'], top_10_states['Asian'], color='brown')
plt.title('Asian Population')

plt.tight_layout()
plt.show()

# Close the database connection
conn.close()
```



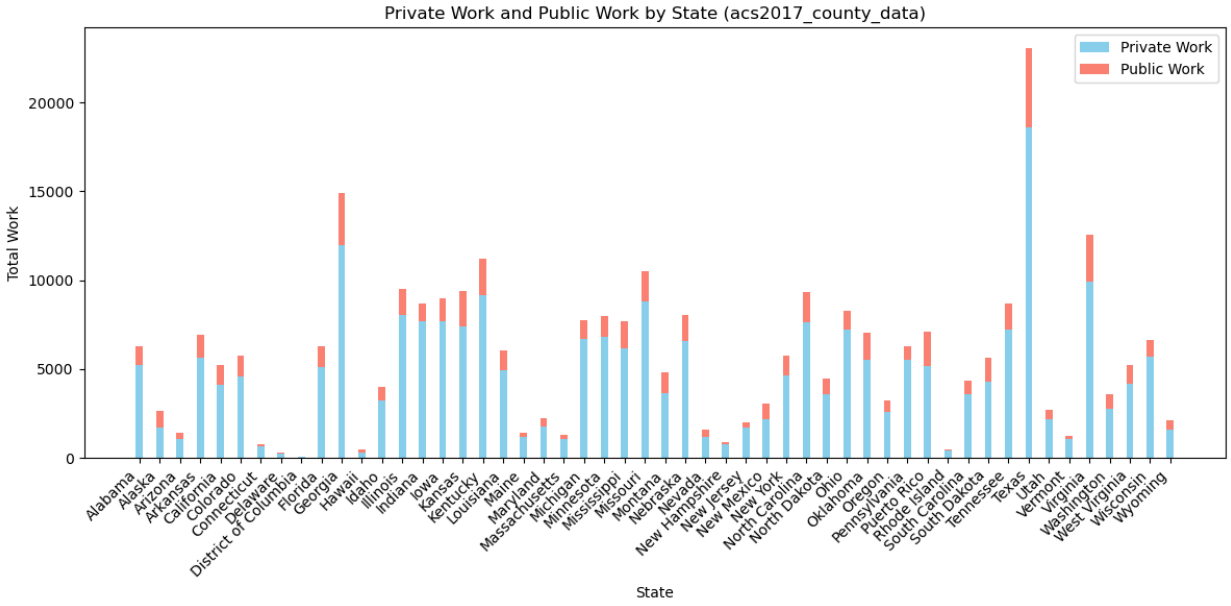
In [25]:
```
# Bar plot for Top 10 States based on Unemployment using acs2017_county_data data.

# Connect to SQLite database
conn = sqlite3.connect('C:\\Users\\14024\sqllite3\\Mydatabase\\dsc540.db')
cursor = conn.cursor()

# SQL query to fetch data
query = '''
    SELECT
        State,
        AVG(Unemployment) AS AvgUnemployment
    FROM
        acs2017_county_data
    GROUP BY
        State
    ORDER BY
        AvgUnemployment DESC
    LIMIT 10
'''

# Execute the query and fetch the result into a pandas DataFrame
result_df = pd.read_sql_query(query, conn)

# Close the connection
conn.close()

# Plotting
plt.figure(figsize=(12, 6))
plt.bar(result_df['State'], result_df['AvgUnemployment'], color='coral')
plt.title('Top 10 States based on Unemployment (acs2017_county_data)')
plt.xlabel('State')
plt.ylabel('Average Unemployment')
plt.xticks(rotation=45, ha='right')
```

```
plt.tight_layout()

# Show the plot
plt.show()
```



In [26]:
```
# Private Work and Public Work by State using acs2017_county_data data.

# Connect to SQLite database
conn = sqlite3.connect('C:\\Users\\14024\sqllite3\\Mydatabase\\dsc540.db')
cursor = conn.cursor()

# SQL query to fetch data
query = '''
    SELECT
        State,
        SUM(PrivateWork) AS TotalPrivateWork,
        SUM(PublicWork) AS TotalPublicWork
    FROM
        acs2017_county_data
    GROUP BY
        State
'''

# Execute the query and fetch the result into a pandas DataFrame
result_df = pd.read_sql_query(query, conn)

# Close the connection
conn.close()

# Plotting
plt.figure(figsize=(12, 6))
bar_width = 0.35

# Bar for Private Work
plt.bar(result_df['State'], result_df['TotalPrivateWork'], bar_width, label='Private W

# Bar for Public Work
plt.bar(result_df['State'], result_df['TotalPublicWork'], bar_width, label='Public Wor

plt.title('Private Work and Public Work by State (acs2017_county_data)')
```

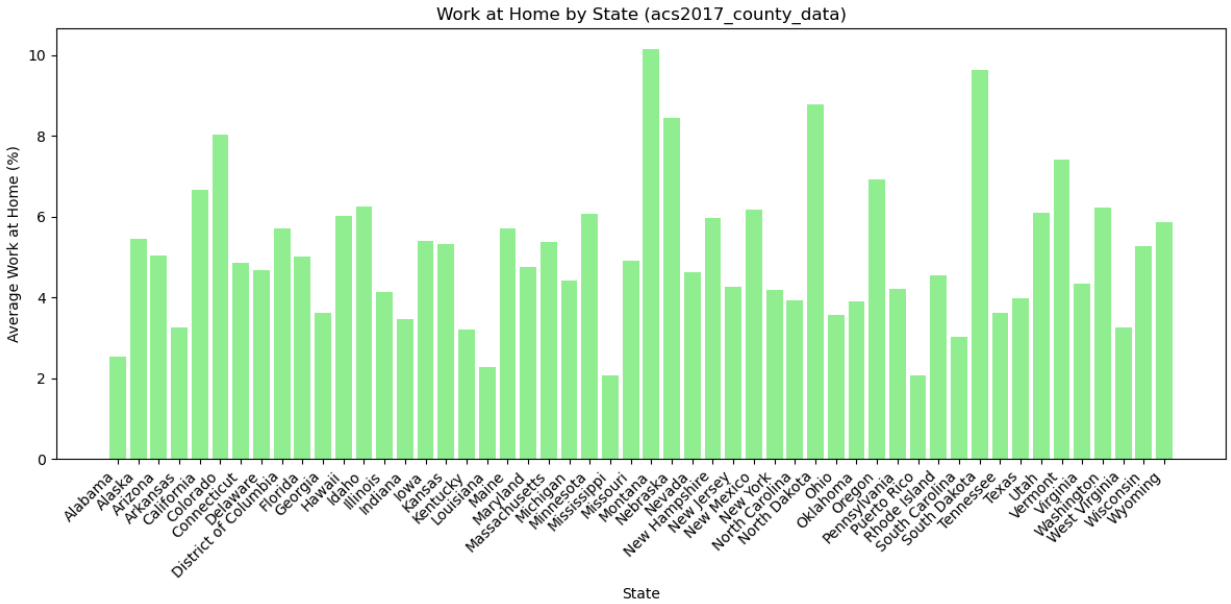
```
plt.xlabel('State')
plt.ylabel('Total Work')
plt.legend()
plt.xticks(rotation=45, ha='right')
plt.tight_layout()

# Show the plot
plt.show()
```



In [27]:

```
# Display Work at Home by State using acs2017_county_data.

# Connect to SQLite database
conn = sqlite3.connect('C:\\Users\\14024\sqllite3\\Mydatabase\\dsc540.db')
cursor = conn.cursor()

# SQL query to fetch data
query = '''
    SELECT
        State,
        AVG(WorkAtHome) AS AvgWorkAtHome
    FROM
        acs2017_county_data
    GROUP BY
        State
'''

# Execute the query and fetch the result into a pandas DataFrame
result_df = pd.read_sql_query(query, conn)

# Close the connection
conn.close()

# Plotting
plt.figure(figsize=(12, 6))
plt.bar(result_df['State'], result_df['AvgWorkAtHome'], color='lightgreen')
plt.title('Work at Home by State (acs2017_county_data)')
plt.xlabel('State')
plt.ylabel('Average Work at Home (%)')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better readability
plt.tight_layout()
```

```
# Show the plot
plt.show()
```



Leraning summary during the course of completion of the Project and Ethical Consideration :

1. Project topic is data wrangling and visualiztion of US Census 2017 data. As a part of this project identified 3 differnt source. Cav data from kaggle. US state related data from wikipedia and performed Census API endpoint URL call.

2. One of the key was to make sure common key between all 3 data sources to make sure they can be merged finally and can be extracted from DB by joining that column for data visualiztion.

3. Performed basic transformations such as Null check in key column, Replace Headers, dataformating to make data readable format, identifying duplicates to avoid cross joins in future data extarction process during vizsualization and fixing casing and inconsistent values from exach data source.

4. As the data sourced from verified source it doesn't need much cleansing and doesn't contain much outliers or Null in it.

5. Used sqlite Db for storing data. Leraned db sqlite db installation and db table creation and storing the dataset or API's data into DB as a part of this project.

6. Merged 3 tables data using the same joining key and created different visualtions using the python bar plots. Visualize the different race population for top 10 states (based on total population). Displayed States wise SelfEmployed people Unemployment using bar diagram. Created bar plot to visulaize Private Work and Public Work by State.

7. While identifying dataset made sure to avoid displaying individual-level data or any information that could lead to the identification of individuals.

8. Clearly communicated the methods used for data collection, processing, and visualization to enhance transparency.

9. Ensured that visualizations accurately represent the underlying data.

10. Was mindful of cultural nuances and sensitivities when representing demographic data, ensuring that visualizations are respectful and do not perpetuate cultural stereotypes.

11. Submitted key request to access API's and use the same key during the API call to make sure sucured data access and use.

12. Created visualizations on summerized data , which is very clear to everyone irrespective of basic python knowledge.

13. Couldn't fetch the us_state table data from wikipedia directly into sqlite db table as the source had multiple subcolumns.