Census Demographics Data Sources

First, we load all the libraries we need.

In [22]:
```
from census import Census
from config import census_key
import gmaps
import numpy as np
import pandas as pd
import requests
import time
from us import states
from scipy.stats import linregress
from matplotlib import pyplot as plt
from pprint import pprint
from uszipcode import SearchEngine
```

We then pull Census data including Population, Median Household Income, Per Capita Income by Zip Code and set up a dataframe.

In [23]:
```
#Import Key census data from census web site
c = Census(census_key, year=2018)
census_data = c.acs5.get(("B01003_001E", "B19019_001E","B19301_001E"), {'for': 'zip code tabulation area:*'})
census_pd = pd.DataFrame(census_data)
census_pd = census_pd.rename(columns={"B01003_001E": "Population",
                                      "B19019_001E": "Median Household Income",
                                      "B19301_001E": "Per Capita Income",
                                      "zip code tabulation area": "Zipcode"})
census_pd
```

Out[23]:

	Population	Median Household Income	Per Capita Income	Zipcode
0	17242.0	13092.0	6999.0	00601
1	38442.0	16358.0	9277.0	00602
2	48814.0	16603.0	11307.0	00603
3	6437.0	12832.0	5943.0	00606
4	27073.0	19309.0	10220.0	00610
...	...	...	...	...
33115	363.0	NaN	NaN	87515
33116	9.0	NaN	NaN	87518
33117	2896.0	NaN	NaN	87511
33118	245.0	NaN	NaN	87578
33119	18756.0	NaN	NaN	87532

33120 rows × 4 columns

A quick view of the json to see what fields we need to pull

```
In [29]: #View census data format
         c.acs5.tables()
```

```
  'variables': 'https://api.census.gov/data/2018/acs/acs5/groups/B07410.json'},
 {'name': 'B25067',
  'description': 'AGGREGATE GROSS RENT (DOLLARS) BY MEALS INCLUDED IN RENT',
  'variables': 'https://api.census.gov/data/2018/acs/acs5/groups/B25067.json'},
 {'name': 'B07411',
  'description': 'MEDIAN INCOME IN THE PAST 12 MONTHS (IN 2018 INFLATION-ADJUSTED DOLLARS) BY
GEOGRAPHICAL MOBILITY IN THE PAST YEAR FOR RESIDENCE 1 YEAR AGO IN THE UNITED STATES',
  'variables': 'https://api.census.gov/data/2018/acs/acs5/groups/B07411.json'},
 {'name': 'B27003',
  'description': 'PUBLIC HEALTH INSURANCE STATUS BY SEX BY AGE',
  'variables': 'https://api.census.gov/data/2018/acs/acs5/groups/B27003.json'},
 {'name': 'B25069',
  'description': 'INCLUSION OF UTILITIES IN RENT',
  'variables': 'https://api.census.gov/data/2018/acs/acs5/groups/B25069.json'},
 {'name': 'B14007F',
  'description': 'SCHOOL ENROLLMENT BY DETAILED LEVEL OF SCHOOL FOR THE POPULATION 3 YEARS AN
D OVER (SOME OTHER RACE ALONE)',
  'variables': 'https://api.census.gov/data/2018/acs/acs5/groups/B14007F.json'},
 {'name': 'B14007G',
  'description': 'SCHOOL ENROLLMENT BY DETAILED LEVEL OF SCHOOL FOR THE POPULATION 3 YEARS AN
```

Export the file as a CSV to Excel

```
In [27]: #Export the csv file
         census_pd.to_csv("census_data_output.csv", index=False, header=True)
```

Since the Census data is grouped by Zip code, and we want the data in County and State, we used the Pypi api's SearchEngine to run all 33 thousand plus zip codes through a loop to associate each zip code with its corresponding County and State.

```
In [34]: #Census data is based on zipcode. We used SearchEngine website to add County & State data to al
         state =[]
         county = []
         mhi = []
         la_sq_mile = []
         zipcode_list = []

         for z in census_pd["Zipcode"]:
             search_zip = SearchEngine(simple_zipcode=True) # set simple_zipcode=False to use rich info
             zipcodes_all = search_zip.by_zipcode(z)
             zipcode_dict = zipcodes_all.to_dict()
             state.append(zipcode_dict["state"])
             county.append(zipcode_dict["county"])
             mhi.append(zipcode_dict["median_household_income"])
             la_sq_mile.append(zipcode_dict["land_area_in_sqmi"])
             zipcode_list.append(zipcode_dict["zipcode"])
```

I quick look at the json to confirm the fields we needed.

```
In [40]: pprint(zipcode_dict)
```

```
{'area_code_list': ['505'],
 'bounds_east': -105.905502,
 'bounds_north': 36.116784,
 'bounds_south': 35.925094,
 'bounds_west': -106.329845,
 'common_city_list': ['Espanola'],
 'county': 'Santa Fe County',
 'housing_units': 7929,
 'land_area_in_sqmi': 113.84,
 'lat': 36.0,
 'lng': -106.1,
 'major_city': 'Espanola',
 'median_home_value': 161800,
 'median_household_income': 40274,
 'occupied_housing_units': 7158,
 'population': 18223,
 'population_density': 160.0,
 'post_office_city': 'Espanola, NM',
 'radius_in_miles': 12.0,
 'state': 'NM',
 'timezone': 'Mountain',
 'water_area_in_sqmi': 0.35,
 'zipcode': '87532',
 'zipcode_type': 'Standard'}
```

Here's the new dataframe which now includes County and State.

```
In [36]: #Created a new Dataframe with the County and State info.
state_info = pd.DataFrame({"Zip Code":zipcode_list,
                           "State":state,
                          "County":county,
                          "Median Household Income":mhi,
                          "Land Area in Square Miles":la_sq_mile,
                          })
state_info
```

Out[36]:

	Zip Code	State	County	Median Household Income	Land Area in Square Miles
0	00601	PR	Adjuntas Municipio	NaN	NaN
1	00602	PR	Aguada Municipio	NaN	30.61
2	00603	PR	Aguadilla Municipio	NaN	31.61
3	00606	PR	Maricao Municipio	NaN	42.31
4	00610	PR	Aasco Municipio	NaN	35.92
...	...	...	...	...	...
33115	87515	NM	Rio Arriba County	37778.0	132.98
33116	87518	NM	Rio Arriba County	41974.0	89.60
33117	87511	NM	Rio Arriba County	49000.0	55.31
33118	87578	NM	Rio Arriba County	58348.0	43.75
33119	87532	NM	Santa Fe County	40274.0	113.84

33120 rows × 5 columns

And finally, export the csv.

```
In [37]: #Exported the new csv file.
         state_info.to_csv("State_County_MHI_Landarea.csv", index=False, header=True)
```

Further Cleanup and Merging the 2 Census Data Sources

We started with dependencies and setup

```
In [1]: # Dependencies and Setup
        import pandas as pd
        import us

        resource_path = "../../Resources/"
        age_bins_path = f"{resource_path}Age Bins.csv"
        census_path = f"{resource_path}census_data_output.csv"
        land_area_path = f"{resource_path}State_County_MHI_Landarea.csv"
```

We then loaded the file with land area

```
In [2]: area_df = pd.read_csv(land_area_path)
        area_df.head()
```

Out[2]:

	Zip Code	State	County	Median Household Income	Land Area in Square Miles
0	601	PR	Adjuntas Municipio	NaN	NaN
1	602	PR	Aguada Municipio	NaN	30.61
2	603	PR	Aguadilla Municipio	NaN	31.61
3	606	PR	Maricao Municipio	NaN	42.31
4	610	PR	Aasco Municipio	NaN	35.92

We then did some clean-up:

- Remove any rows missing land area (cannot figure population density without that)
- Remove any rows missing Median Household Income
- Oklahoma has 1 zip code without a county name, but since we're not doing a by-county analysis, it doesn't matter
- Remove any rows with a negative Median Household Income (some commercial-only zip codes have this)

- Rename Land Area in Square Miles to Land Area (m2) for brevity
- The results show the same number of rows in all columns:

In [3]:
```
area_df['Median Household Income'].unique()

#Eliminate records with no land area or income figures
clean_area_df = area_df[area_df['Land Area in Square Miles'].notnull()]
clean_area_df = clean_area_df[clean_area_df['Median Household Income'].notnull()]

#Missing county name is insignificant for statewide analysis;
#would need to deal with if drilling down into Oklahoma counties
#clean_area_df = clean_area_df[clean_area_df['County'].isnull()]

#Remove any records that reference a negative income; this could be attributed to zip codes with no residences
clean_area_df = clean_area_df[clean_area_df['Median Household Income'] > 0]

#Clean up column names; this will help with merging further below
clean_area_df.rename(columns={'Land Area in Square Miles' : 'Land Area (m2)'}, inplace=True)

#A useable data source without nulls will have same counts for all columns
#(Again, the missing county is insignificant to statewide aggregation)
clean_area_df.count()
```

Out[3]:
```
Zip Code                  31859
State                     31859
County                    31858
Median Household Income   31859
Land Area (m2)            31859
dtype: int64
```

To ensure this data frame is ready for joining by zip code, we ensure that the number of unique zip codes is the same as the total number of zip codes above

In [4]:
```
#If the result of this expression matches the total row count above,
#then the column values are all unique and zip code can be used to merge
print(len(clean_area_df['Zip Code'].unique()))
```

```
31859
```

Next, we loaded the census information described above

In [5]:
```
census_df = pd.read_csv(census_path)
census_df.head()
```

Out[5]:

	Population	Median Household Income	Per Capita Income	Zipcode	Population density
0	17242.0	13092.0	6999.0	601	0.000053
1	38442.0	16358.0	9277.0	602	0.000118
2	48814.0	16603.0	11307.0	603	0.000150
3	6437.0	12832.0	5943.0	606	0.000020
4	27073.0	19309.0	10220.0	610	0.000083

We did some cleanup on this data source as well:

- Eliminate any rows missing Median Household Income
- Eliminate any rows missing Per Capita Income
- Create Total Income by multiplying Per Capita Income by Population; this will be needed later to aggregate to the state level

- Renamed Zip Code and Population Density colums for consistency
- The results show the same number of rows in all columns:

In [6]:
```
#Eliminate rows that are missing data for any measures (income, population, etc.)
clean_census_df = census_df[census_df['Median Household Income'].notnull()]
clean_census_df = clean_census_df[clean_census_df['Per Capita Income'].notnull()]

#Remove any records that reference a negative income; this could be attributed to zip codes with no residences
clean_census_df = clean_census_df[clean_census_df['Median Household Income'] > 0]
clean_census_df = clean_census_df[clean_census_df['Per Capita Income'] > 0]

clean_census_df['Total Income'] = clean_census_df['Per Capita Income'] * clean_census_df['Population']

#Clean up column names; this will help with merging further below
clean_census_df.rename(columns={'Zipcode' : 'Zip Code',
                                'Population density' : 'Population Density'}, inplace=True)

#With null measures eliminated, they should all return the same count
clean_census_df.count()
```

Out[6]:
```
Population                30888
Median Household Income   30888
Per Capita Income         30888
Zip Code                  30888
Population Density        30888
Total Income              30888
dtype: int64
```

Again, we checked to make sure the data frame is ready for merging by zip code by verifying the unique list of zip codes is the same length as the total list of zip codes

In [7]:
```
#If the result of this expression matches the total row count above,
#then the column values are all unique and zipcode can be used to merge
print(len(clean_census_df['Zip Code'].unique()))
```

```
30888
```

Merge the 2 data sources above on zip code.

- Inner join; any missing data makes the whole row irrelevant
- Drop any rows in the resulting data frame that are missing a state; we won't be able to aggregate those into the state level
- Drop any rows missing Population
- The results show the same number of rows in all columns:

In [8]:
```
merge_df = pd.merge(clean_area_df, clean_census_df, on='Zip Code', how='inner', suffixes=('_a', '_b'), validate='one
#merge_df.count()

clean_merge_df = merge_df[merge_df['State'].notnull()]
clean_merge_df = clean_merge_df[clean_merge_df['Population'].notnull()]

clean_merge_df.count()
```

Out[8]:
```
Zip Code                  30636
State                     30636
County                    30635
Median Household Income_a 30636
Land Area (m2)            30636
Population                30636
Median Household Income_b 30636
Per Capita Income         30636
Population Density        30636
Total Income              30636
dtype: int64
```

Group the data frame by state and aggregate our measures by median, sum, or custom function as needed. Also rename land area column to correctly reflect the unit

In [10]:
```
state_grp = clean_merge_df.groupby(['State'])
state_mhia = state_grp['Median Household Income_a'].median()
state_mhib = state_grp['Median Household Income_b'].median()
state_area = state_grp['Land Area (m2)'].sum()
state_pop = state_grp['Population'].sum()
state_pci = state_grp['Total Income'].sum() / state_pop
state_pd = state_pop / state_grp['Land Area (m2)'].sum()

state_df = pd.DataFrame(state_area)
state_df['Population'] = state_pop
state_df['Population Density'] = state_pd
state_df['Per Capita Income'] = state_pci
#state_df['Median Household Income A'] = state_mhia
state_df['Median Household Income'] = state_mhib

state_df.rename(columns={'Land Area (m2)' : 'Land Area (mi2)'}, inplace=True)

state_df.head()
```

Out[10]:

State	Land Area (mi2)	Population	Population Density	Per Capita Income	Median Household Income
AK	211300.81	734417.0	3.475694	35617.249121	51875.0
AL	49481.34	4846150.0	97.938940	26883.102656	41750.0
AR	49590.60	2977621.0	60.044061	25666.556534	40574.0
AZ	89709.56	6940873.0	77.370494	29270.505603	49063.0
CA	96922.44	39076841.0	403.176406	35049.472573	65231.0

Finally, save the results to a CSV for further processing

In [11]:
```
state_df.to_csv("Resources/IncomePopByState.csv")
```

Cases and Deaths Data Source

Import necessary libraries

In [1]:
```
# Dependencies and Setup
import pandas as pd
import numpy as np
import requests
import us

from datetime import date, timedelta
from pprint import pprint
```

Prepare parameters and a result data frame for the results

In [2]:
```
#Initialize data source URL
url = 'https://covid-api.com/api/reports'

#Initialize list of dates: all dates between today and March 1, 2020
#API requires queries by date, so we will need to request one date at a time
start_date = date(2020,3,1)
end_date = date.today()

#Uncomment this assignment to run the original 'bi-monthly' data frame
#dates = ['2020-03-01', '2020-03-15', '2020-04-01', '2020-04-15', '2020-05-01',
#         '2020-05-15', '2020-06-01', '2020-06-15', '2020-07-01', '2020-07-15']

#Uncomment this assignment to run the enhanced 'daily' data frame
dates = [str(start_date + timedelta(n)) for n in range(int((end_date - start_date).days))]

#Initialize data frame
cases_df = pd.DataFrame(columns=[
    'Active',
    'Confirmed',
    'Date',
    'Deaths',
    'Fatality Rate',
    'Recovered',
    'State',
    'State Abbr',
    'Active Diff',
    'Confirmed Diff',
    'Deaths Diff',
    'Recovered Diff',
    'Lat',
    'Lon'
])
```

Out[2]:

Active	Confirmed	Date	Deaths	Fatality Rate	Recovered	State	State Abbr	Active Diff	Confirmed Diff	Deaths Diff	Recovered Diff	Lat	Lon

There are 2 alternate versions of this data frame:

- An original “bi-monthly” data frame that has the 1st and 15th of every month since cases and deaths have been counted (March 2020). This data source omits the 5 “Diff” columns seen above, since the figures are presented as daily differentials and do not make sense in a bi-monthly context.
- An enhanced “daily” data frame that has all dates since March 1, 2020, to date. This frame includes the daily differentials since they make sense with daily data.

Analyze a sample of the results returned by the URL query. For this, we just said USA and a recent date

```
In [3]: query_params = {'date' : '2020-07-22',
                        'iso' : 'USA'}
        response = requests.get(url, params=query_params)
        json = response.json()
        pprint(json)
```

```
{'data': [{'active': 70449,
           'active_diff': 1394,
           'confirmed': 71813,
           'confirmed_diff': 1455,
           'date': '2020-07-22',
           'deaths': 1364,
           'deaths_diff': 61,
           'fatality_rate': 0.019,
           'last_update': '2020-07-23 05:15:04',
           'recovered': 0,
           'recovered_diff': 0,
           'region': {'cities': [{'confirmed': 863,
                                  'confirmed_diff': 18,
                                  'long': '-87.37325922',
                                  'name': 'Winston'}],
                      'iso': 'USA',
                      'lat': '32.3182',
                      'long': '-86.9023',
                      'name': 'US',
                      'province': 'Alabama'}},
          {'active': 2112,
           'active_diff': 88,
           'confirmed': 2131,
```

The data source returns a list, data, in which each element represents a “state” (more on this later). Each data element has a region dictionary with information about the state, includes its counties, packed in the list labeled as cities above. The details of that list have been omitted because we are not using county-level data because our census data by zip code only aggregates cleanly to the state level (zip codes often span multiple counties).

Execute the query for each date, depending on the version being run, and pack each returned relevant data element into the data frame from above.

In [3]:
```
#Loop through all dates established above
for date in dates:

    #Set up the params for USA and the current date, and request
    query_params = {'date' : date,
                    'iso' : 'USA'}
    response = requests.get(url, params=query_params)
    print(f"Retreiving {response.url}...")
    json = response.json()

    try:
        data = json['data']

        #In this API, the 'data' element is a list whose elements represent states
        #Loop through the list to extract by-state information
        for datum in data:

            try:

                #Get the state name from the current 'datum'
                region = datum['region']
                state = region['province']

                #Only proceed with reading if the state is an actual US state
                #Other information is not joinable to income and population data
                #and should be ignored for this project
                if (state in [states.name for states in us.states.STATES]):

                    #print(f"Loading information for {state}...")

                    #Read fields for the current 'state' row
                    active = datum['active']
                    active_diff = datum['active_diff']
                    confirmed = datum['confirmed']
                    confirmed_diff = datum['confirmed_diff']
                    date = datum['date']
                    deaths = datum['deaths']
                    deaths_diff = datum['deaths_diff']
                    fatality_rate = datum['fatality_rate']
                    recovered = datum['recovered']
                    recovered_diff = datum['recovered_diff']
                    state_lat = region['lat']
                    state_lon = region['long']

                    #Create a dictionary with the contents for the current 'state' row
                    data_row = {
                        'Active' : active,
                        'Confirmed' : confirmed,
                        'Date' : date,
                        'Deaths' : deaths,
                        'Fatality Rate' : fatality_rate,
                        'Recovered' : recovered,
                        'State' : state,
                        'State Abbr' : us.states.lookup(state).abbr,
                        'Active Diff' : active_diff,
                        'Confirmed Diff' : confirmed_diff,
                        'Deaths Diff' : deaths_diff,
                        'Recovered Diff' : recovered_diff,
                        'Lat' : state_lat,
                        'Lon' : state_lon
                    }

                    #Append the dictionary as a row to the data frame
                    cases_df = cases_df.append(data_row, ignore_index=True)

            except KeyError: #If a particular data element doesn't have an expected field
                print("A requested field was not found; skipping...")

    except KeyError: #If a particular date result doesn't have an expected field
        print("No data returned; skipping...")

#Show a sample from the data frame to show it is populated
cases_df.head()
```

```
Retreiving https://covid-api.com/api/reports?date=2020-03-01&iso=USA...
Retreiving https://covid-api.com/api/reports?date=2020-03-15&iso=USA...
Retreiving https://covid-api.com/api/reports?date=2020-04-01&iso=USA...
Retreiving https://covid-api.com/api/reports?date=2020-04-15&iso=USA...
Retreiving https://covid-api.com/api/reports?date=2020-05-01&iso=USA...
Retreiving https://covid-api.com/api/reports?date=2020-05-15&iso=USA...
Retreiving https://covid-api.com/api/reports?date=2020-06-01&iso=USA...
Retreiving https://covid-api.com/api/reports?date=2020-06-15&iso=USA...
Retreiving https://covid-api.com/api/reports?date=2020-07-01&iso=USA...
Retreiving https://covid-api.com/api/reports?date=2020-07-15&iso=USA...
```

Out[4]:

	Active	Confirmed	Date	Deaths	Fatality Rate	Recovered	State	State Abbr	Active Diff	Confirmed Diff	Deaths Diff	Recovered Diff	Lat	Lon
0	12	12	2020-03-15	0	0	0	Alabama	AL	6	6	0	0	32.3182	-86.9023
1	1	1	2020-03-15	0	0	0	Alaska	AK	0	0	0	0	61.3707	-152.4044
2	12	13	2020-03-15	0	0	1	Arizona	AZ	1	1	0	0	33.7298	-111.4312
3	16	16	2020-03-15	0	0	0	Arkansas	AR	4	4	0	0	34.9697	-92.3731
4	414	426	2020-03-15	6	0.0141	6	California	CA	85	86	1	0	36.1162	-119.6816

Here, we kept the following fields, as relevant measures to compare against wealth, population density, and age:

- Active
- Confirmed
- Date
- Deaths
- Fatality Rate (for convenience; figured as Deaths / Confirmed)
- Recovered (for convenience; figured as Confirmed – Active – Deaths)
- State
- State's abbreviation (from the py US library)
- Active Diff*
- Confirmed Diff*
- Deaths Diff*
- Recovered Diff*
- Lat/Lon for each state, as provided by this data source

*Differential fields were only kept in the "daily" version of this data source, since they represent daily changes.

We also observed there to be a lot things reported in the data as "states" that were not actually states. Some examples are a lot of the cruise ships at sea that were reporting cases. Since those are not real states that we do not have real census data to compare with, we used the py US library to only extract data pertaining to real states, as seen in the "if" block above.

To retrieve all the data correctly, we needed to loop once for each day needed. This was the reason we initially created a bi-monthly data source to be faster to work with, and then later created a daily data source that was more detailed, but much bigger and taking longer to create.

Within each date, we had to loop through all the data elements returned, determine if each represented a real state, and if so, extract its information needed for the data frame as described above. A pair of nested try/except blocks attempt to catch KeyErrors so that the script can keep processing if a particular date or data element does not contain the dictionary keys expected.

We determined this would be a more optimal way to retrieve all the data than blindly loading everything returned into the data frame and then attempting the drop the junk data out of it (i.e., the rows that would not represent real states)

Afterwards, we checked to ensure all columns have the same amount of data to ensure no row is missing any important measures

```
In [5]: #Check that we have data in all rows of all columns
        #This is particularly important for the measure columns (Cases, Deaths, etc.)
        cases_df.count()

Out[5]: Active            450
        Confirmed         450
        Date              450
        Deaths            450
        Fatality Rate     450
        Recovered         450
        State             450
        State Abbr        450
        Active Diff       450
        Confirmed Diff    450
        Deaths Diff       450
        Recovered Diff    450
        Lat               450
        Lon               450
        dtype: int64
```

Check to make sure we only have 50 unique states in the data frame

```
In [6]: #Check to ensure there is no noise in the State Column
        #Specifically, we should only have 50 unique values
        print(len(cases_df['State'].unique()))

        50
```

Finally, save the results to a csv for convenient access merging to census information and plotting, since especially the daily version takes a long time to load

```
In [7]: #Save results to CSV for separate plotting (Uncomment the version of this line being run)
        #cases_df.to_csv("Resources/CasesByState-Daily.csv", index=False)
        cases_df.to_csv("Resources/CasesByState.csv", index=False)
```

Fully Combined Data Source

Import necessary libraries

In [1]:
```
# Dependencies and Setup
import pandas as pd
import requests
import us
from pprint import pprint

resource_path = "Resources"
cases_by_state_path = f"{resource_path}/CasesByState-Daily.csv"
income_pop_path = f"{resource_path}/IncomePopByState.csv"
```

Read the cases-by-state data source saved in the CSV file described above

In [2]:
```
cases_df = pd.read_csv(cases_by_state_path)
cases_df.head()
```

Out[2]:

	Active	Confirmed	Date	Deaths	Fatality Rate	Recovered	State	State Abbr	Active Diff	Confirmed Diff	Deaths Diff	Recovered Diff	Lat	Lon
0	0	0	2020-03-10	0	0.0000	0	Alaska	AK	0	0	0	0	61.3707	-152.4044
1	5	6	2020-03-10	0	0.0000	1	Arizona	AZ	4	5	0	1	33.7298	-111.4312
2	0	0	2020-03-10	0	0.0000	0	Arkansas	AR	0	0	0	0	34.9697	-92.3731
3	140	144	2020-03-10	2	0.0139	2	California	CA	138	142	2	2	36.1162	-119.6816
4	15	15	2020-03-10	0	0.0000	0	Colorado	CO	0	0	0	0	39.0598	-105.3111

Rename State and State Abbr columns to be more consistent with the census data source, especially since we'll be joining on State abbreviation

In [3]:
```
cases_df.rename(columns={'State' : 'State Name', 'State Abbr' : 'State'}, inplace=True)
cases_df.head()
```

Out[3]:

	Active	Confirmed	Date	Deaths	Fatality Rate	Recovered	State Name	State	Active Diff	Confirmed Diff	Deaths Diff	Recovered Diff	Lat	Lon
0	0	0	2020-03-10	0	0.0000	0	Alaska	AK	0	0	0	0	61.3707	-152.4044
1	5	6	2020-03-10	0	0.0000	1	Arizona	AZ	4	5	0	1	33.7298	-111.4312
2	0	0	2020-03-10	0	0.0000	0	Arkansas	AR	0	0	0	0	34.9697	-92.3731
3	140	144	2020-03-10	2	0.0139	2	California	CA	138	142	2	2	36.1162	-119.6816
4	15	15	2020-03-10	0	0.0000	0	Colorado	CO	0	0	0	0	39.0598	-105.3111

Load the data source with census demographic information from its CSV described above

In [4]:
```
demo_df = pd.read_csv(income_pop_path)
demo_df.head()
```

Out[4]:

	State	Land Area (mi2)	Population	Population Density	Per Capita Income	Median Household Income
0	AK	211300.81	734417.0	3.475694	35617.249121	51875.0
1	AL	49481.34	4846150.0	97.938940	26883.102656	41750.0
2	AR	49590.60	2977621.0	60.044061	25666.556534	40574.0
3	AZ	89709.56	6940873.0	77.370494	29270.505603	49063.0
4	CA	96922.44	39076841.0	403.176406	35049.472573	65231.0

Ensure the census demographics data source is unique by state (51 includes DC)

```
In [5]: print(demo_df.count())
        print(len(demo_df['State'].unique()))
```

```
State                    51
Land Area (mi2)          51
Population               51
Population Density       51
Per Capita Income        51
Median Household Income  51
dtype: int64
51
```

Merge cases to census demographics on state using an inner join since we cannot use rows with missing information on either side. Validate many-to-one because the cases data frame has a list of all 50 states for each date within it.

```
In [6]: all_df = pd.merge(cases_df, demo_df, on='State', how='inner', validate='many_to_one')
        all_df.head(50)
```

Out[6]:

	Active	Confirmed	Date	Deaths	Fatality Rate	Recovered	State Name	State	Active Diff	Confirmed Diff	Deaths Diff	Recovered Diff	Lat	Lon	Land Area (mi2)	Population
0	0	0	2020-03-10	0	0.0000	0	Alaska	AK	0	0	0	0	61.3707	-152.4044	211300.81	734417.0
1	0	0	2020-03-11	0	0.0000	0	Alaska	AK	0	0	0	0	61.3707	-152.4044	211300.81	734417.0
2	0	0	2020-03-12	0	0.0000	0	Alaska	AK	0	0	0	0	61.3707	-152.4044	211300.81	734417.0
3	1	1	2020-03-13	0	0.0000	0	Alaska	AK	1	1	0	0	61.3707	-152.4044	211300.81	734417.0
4	1	1	2020-03-15	0	0.0000	0	Alaska	AK	0	0	0	0	61.3707	-152.4044	211300.81	734417.0
5	1	1	2020-03-16	0	0.0000	0	Alaska	AK	0	0	0	0	61.3707	-152.4044	211300.81	734417.0

Finally, save the resulting data set in a CSV file for use with plotting

```
In [7]: all_df.to_csv("Resources/AllByState-Daily.csv", index=False)
```

Age Groups

For Age groups, we first download the libraries.

```
In [100]: from census import Census
          import gmaps
          import numpy as np
          import pandas as pd
          import requests
          import time
          from us import states
          from scipy.stats import linregress
          import matplotlib.pyplot as plt
```

Load a csv from the CDC website containing info on number of deaths by age bracket for nationwide and state level.

```
In [2]: #Load the csv file from cdc website.
death_by_age = "CDC_Death_by_Age1.csv"
death_df = pd.read_csv(death_by_age, encoding="ISO-8859-1")
death_df
```

Out[2]:

	ï»¿Data as of	Start week	End Week	State	Sex	Age group	COVID-19 Deaths	Total Deaths	Pneumonia Deaths	Pneumonia and COVID-19 Deaths	Influenza Deaths	Pneumonia, Influenza, or COVID-19 Deaths	Footnote
0	7/22/20	2/1/20	7/11/20	United States	All	Under 5 years	20.0	1,547	52	2	41	100	NaN
1	7/22/20	2/1/20	7/11/20	United States	All	5-14 years	16.0	2,379	78	5	49	138	NaN
2	7/22/20	2/1/20	7/11/20	United States	All	15-24 years	190.0	14,810	300	62	51	475	NaN
3	7/22/20	2/1/20	7/11/20	United States	All	25-34 years	935.0	30,885	1,113	416	149	1,768	NaN
4	7/22/20	2/1/20	7/11/20	United States	All	35-44 years	2411.0	43,783	2,433	1,009	242	4,048	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1410	7/22/20	2/1/20	7/11/20	Puerto Rico	Female	75-84 years	0.0	0	0	0	0	0	NaN
1411	7/22/20	2/1/20	7/11/20	Puerto Rico	Female	85 years and over	0.0	0	0	0	0	0	NaN
1412	7/22/20	2/1/20	7/11/20	Puerto Rico	Female	All ages	NaN	434	44	NaN	NaN	52	One or more data cells have counts between 1â€...
1413	7/22/20	2/1/20	7/11/20	Puerto Rico	Unknown	All ages	0.0	NaN	0	0	0	0	One or more data cells have counts between 1â€...
1414	7/22/20	2/1/20	7/11/20	Puerto Rico Total	All	All Ages	29.0	1,380	112	11	NaN	136	One or more data cells have counts between 1â€...

1415 rows × 13 columns

Delete Columns that were no needed

In [3]:
```
#delete columns that are not needed.
del death_df['Total Deaths']
del death_df['Pneumonia Deaths']
del death_df['Pneumonia and COVID-19 Deaths']
del death_df['Influenza Deaths']
del death_df['Pneumonia, Influenza, or COVID-19 Deaths']
death_df
```

Out[3]:

	ï»¿Data as of	Start week	End Week	State	Sex	Age group	COVID-19 Deaths	Footnote
0	7/22/20	2/1/20	7/11/20	United States	All	Under 5 years	20.0	NaN
1	7/22/20	2/1/20	7/11/20	United States	All	5-14 years	16.0	NaN
2	7/22/20	2/1/20	7/11/20	United States	All	15-24 years	190.0	NaN
3	7/22/20	2/1/20	7/11/20	United States	All	25-34 years	935.0	NaN
4	7/22/20	2/1/20	7/11/20	United States	All	35-44 years	2411.0	NaN
...	...	...	...	...	...	...	...	...
1410	7/22/20	2/1/20	7/11/20	Puerto Rico	Female	75-84 years	0.0	NaN
1411	7/22/20	2/1/20	7/11/20	Puerto Rico	Female	85 years and over	0.0	NaN
1412	7/22/20	2/1/20	7/11/20	Puerto Rico	Female	All ages	NaN	One or more data cells have counts between 1â€...
1413	7/22/20	2/1/20	7/11/20	Puerto Rico	Unknown	All ages	0.0	One or more data cells have counts between 1â€...
1414	7/22/20	2/1/20	7/11/20	Puerto Rico Total	All	All Ages	29.0	One or more data cells have counts between 1â€...

1415 rows × 8 columns

Locate just the rows pertaining to national data.

In [4]:
```
#Locate rows that are just for all states, both sexes by age group
death_all_df = death_df.loc[(death_df["State"]=="United States") &
                            (death_df["Sex"]=="All") &
                            (death_df["Age group"]!="All Ages")]
death_all_df.head(20)
```

Out[4]:

	ï»¿Data as of	Start week	End Week	State	Sex	Age group	COVID-19 Deaths	Footnote
0	7/22/20	2/1/20	7/11/20	United States	All	Under 5 years	20.0	NaN
1	7/22/20	2/1/20	7/11/20	United States	All	5-14 years	16.0	NaN
2	7/22/20	2/1/20	7/11/20	United States	All	15-24 years	190.0	NaN
3	7/22/20	2/1/20	7/11/20	United States	All	25-34 years	935.0	NaN
4	7/22/20	2/1/20	7/11/20	United States	All	35-44 years	2411.0	NaN
5	7/22/20	2/1/20	7/11/20	United States	All	45-54 years	6566.0	NaN
6	7/22/20	2/1/20	7/11/20	United States	All	55-64 years	15880.0	NaN
7	7/22/20	2/1/20	7/11/20	United States	All	65-74 years	27167.0	NaN
8	7/22/20	2/1/20	7/11/20	United States	All	75-84 years	34399.0	NaN
9	7/22/20	2/1/20	7/11/20	United States	All	85 years and over	42666.0	NaN

Clean up Column headings

In [5]:
```
#Clean up column names
death_all_us = death_all_df[["ï»¿Data as of",
                             "State", "Age group","COVID-19 Deaths"]]
death_all_us = death_all_us.rename(columns={"ï»¿Data as of": "Data as of",})

death_all_us
```

Out[5]:

	Data as of	State	Age group	COVID-19 Deaths
0	7/22/20	United States	Under 5 years	20.0
1	7/22/20	United States	5-14 years	16.0
2	7/22/20	United States	15-24 years	190.0
3	7/22/20	United States	25-34 years	935.0
4	7/22/20	United States	35-44 years	2411.0
5	7/22/20	United States	45-54 years	6566.0
6	7/22/20	United States	55-64 years	15880.0
7	7/22/20	United States	65-74 years	27167.0
8	7/22/20	United States	75-84 years	34399.0
9	7/22/20	United States	85 years and over	42666.0

Load Census data downloaded from the Census website.

In [7]:
```
#Load Census data from the U.S. Census
census_age = "Age Bins1.csv"
age_df = pd.read_csv(census_age, encoding="ISO-8859-1")
age_df.head(20)
```

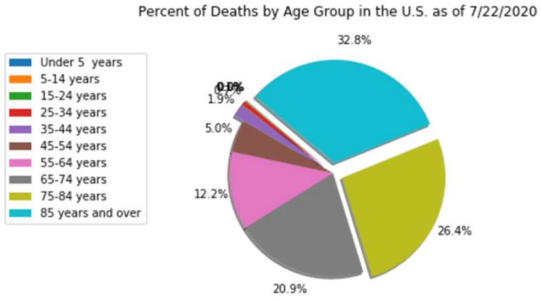
Out[7]:

	Age Bracket	Number of People	Percent of people
0	Under 5 years	19929	6.2
1	5 to 14 years	41055	12.7
2	15 to 24 years	42342	13.1
3	25 to 34 years	44855	13.9
4	35 to 44 years	40660	12.6
5	45 to 54 years	41537	12.9
6	55 to 64 years	41700	12.9
7	65 to 74 years	30366	9.4
8	75 to 84 years	14559	4.5
9	85 years and over	6154	1.9

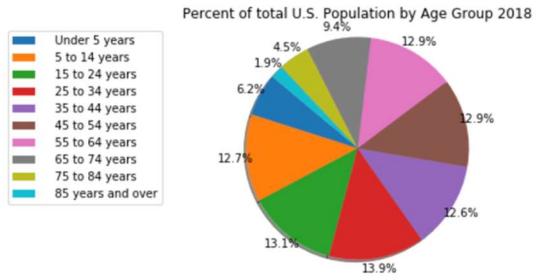
Create two pie charts showing Populations by age bracket and Covid-19 Deaths by age bracket.

In [91]:
```
#Pie Chart of Covid deaths by age group from CDC as f 7/22/2020
covid_deaths = death_all_us["COVID-19 Deaths"]
plt.pie(covid_deaths, explode=(0.1,0.1,0.1,0.1,0.1,0,0,0,0.1,0.1,),  colors=None,
        autopct="%1.1f%%", shadow=True, startangle=140, rotatelabels=False, radius=1, pctdistance = 1.2, labeldistance
plt.axis("equal")
plt.title("Percent of Deaths by Age Group in the U.S. as of 7/22/2020",y = 1.1)
plt.legend(
    loc='best',
    labels=death_all_us["Age group"],
    prop={'size': 10},
    bbox_to_anchor=(0.0, 1),)

plt.savefig("Images/Percent_Death_US_Age_Group.png",bbox_inches='tight')
plt.show()
```



In [97]:
```
#Pie Chart of Total U.S. Population by age group from U.S. Census 2018
pct_age_pop = age_df["Percent of people"]
plt.pie(pct_age_pop, explode=None,  colors=None, autopct="%1.1f%%",shadow=True, startangle=140, rotatelabels=False, rad
plt.axis("equal")
plt.title("Percent of total U.S. Population by Age Group 2018")
plt.legend(
    loc='best',
    labels=age_df["Age Bracket"],
    prop={'size': 10},
    bbox_to_anchor=(0.0, 1),)
plt.savefig("Images/Percent_Pop_US.png",bbox_inches='tight')
plt.show()
```



We searched for the states with the five highest rates of death from Covid-19

```
In [11]: #Sort by highest rates of death by state from covid-19.
state_totals = death_df.loc[(death_df["State"]!="United States") &
                            (death_df["Sex"]=="All")]
state_totals_sort = state_totals.sort_values("COVID-19 Deaths",ascending=False)
state_totals_sort
```

Out[11]:

	ï»¿Data as of	Start week	End Week	State	Sex	Age group	COVID-19 Deaths	Footnote
36	7/22/20	2/1/20	7/11/20	United States Total	All	All Ages	130250.0	NaN
920	7/22/20	2/1/20	7/11/20	New York City Total	All	All Ages	20460.0	NaN
842	7/22/20	2/1/20	7/11/20	New Jersey Total	All	All Ages	13811.0	NaN
894	7/22/20	2/1/20	7/11/20	New York Total	All	All Ages	11242.0	NaN
608	7/22/20	2/1/20	7/11/20	Massachusetts Total	All	All Ages	7753.0	NaN
1076	7/22/20	2/1/20	7/11/20	Pennsylvania Total	All	All Ages	7227.0	NaN
166	7/22/20	2/1/20	7/11/20	California Total	All	All Ages	7100.0	NaN
400	7/22/20	2/1/20	7/11/20	Illinois Total	All	All Ages	6652.0	NaN
634	7/22/20	2/1/20	7/11/20	Michigan Total	All	All Ages	5596.0	NaN
296	7/22/20	2/1/20	7/11/20	Florida Total	All	All Ages	4341.0	NaN
218	7/22/20	2/1/20	7/11/20	Connecticut Total	All	All Ages	4031.0	NaN

To save time we manually updated the csv file to combine data and reimported it.

```
In [62]: #Import clean CSV file for top 5 state deaths
top_states_death = "CDC_Death_by_Age_Hi_Low_States.csv"
top_states_death_df = pd.read_csv(top_states_death, encoding="ISO-8859-1")
top_states_death_df.head()
```

Out[62]:

Cleaned up the column headings.

```
In [63]: #Clean up columns
del top_states_death_df['Sex']
top_states_death_df = top_states_death_df.rename(columns={"ï»¿Data as of": "Data as of",})
top_states_death_df.head()
```

Out[63]:

	Data as of	Start week	End Week	State	Age group	COVID-19 Deaths
0	7/22/20	2/1/20	7/11/20	New York	Under 5 years	0
1	7/22/20	2/1/20	7/11/20	New York	5-14 years	0
2	7/22/20	2/1/20	7/11/20	New York	15-24 years	43
3	7/22/20	2/1/20	7/11/20	New York	25-34 years	227
4	7/22/20	2/1/20	7/11/20	New York	35-44 years	680

Created variables for each of the top five states.

In [68]:
```
#Create variables for each state
ny_deaths = top_states_death_df.loc[(top_states_death_df["State"]=="New York")]
nj_deaths = top_states_death_df.loc[(top_states_death_df["State"]=="New Jersey")]
ma_deaths = top_states_death_df.loc[(top_states_death_df["State"]=="Massachusetts")]
pa_deaths = top_states_death_df.loc[(top_states_death_df["State"]=="Pennsylvania")]
ca_deaths = top_states_death_df.loc[(top_states_death_df["State"]=="California")]
```

Out[68]:

	Data as of	Start week	End Week	State	Age group	COVID-19 Deaths
0	7/22/20	2/1/20	7/11/20	New York	Under 5 years	0
1	7/22/20	2/1/20	7/11/20	New York	5-14 years	0
2	7/22/20	2/1/20	7/11/20	New York	15-24 years	43
3	7/22/20	2/1/20	7/11/20	New York	25-34 years	227
4	7/22/20	2/1/20	7/11/20	New York	35-44 years	680
5	7/22/20	2/1/20	7/11/20	New York	45-54 years	1862
6	7/22/20	2/1/20	7/11/20	New York	55-64 years	4572
7	7/22/20	2/1/20	7/11/20	New York	65-74 years	7231
8	7/22/20	2/1/20	7/11/20	New York	75-84 years	8223
9	7/22/20	2/1/20	7/11/20	New York	85 years and over	8850

And created pie charts for each of the five states with New York given as an example.

In [92]:
```
# Pie Chart of percent of deaths in New York by age group
ny_covid_deaths = ny_deaths["COVID-19 Deaths"]
plt.pie(ny_covid_deaths, explode=(0.1,0.1,0.1,0.1,0.1,0,0,0,0.1,0.1,),  colors=None,
        autopct="%1.1f%%", shadow=True, startangle=140, rotatelabels=False, radius=1, pctdistance = 1.2, labeldistance
plt.axis("equal")
plt.title("Percent of Deaths by Age Group in New York as of 7/22/2020",y = 1.1)
plt.legend(
    loc='best',
    labels=death_all_us["Age group"],
    prop={'size': 10},
    bbox_to_anchor=(0.0, 1),)
plt.savefig("Images/Percent_Death_NY_Age_Group.png",bbox_inches='tight')
plt.show()
```