Introduction

The New York Times has published their COVID-19 tracking data on [GitHub (https://github.com/nytimes/covid-19-data)](https://github.com/nytimes/covid-19-data). The data is simple and tidy and the county-level data includes the FIPS (Federal Information Processing Standards) code for each.

Using FIPS as the common reference point, I attempted to marry COVID-19 infection rates with county median income and population information.

This is the result.

COVID-19 Infection Data

In [33]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import requests
import datetime as dt
%matplotlib inline
```

In [34]:
```
# COVID data source from NYT: https://github.com/nytimes/covid-19-data
url = 'https://raw.githubusercontent.com/nytimes/covid-19-data/master/us
-counties.csv'
r = requests.get(url, allow_redirects=True)
open('us_county_covid.csv', 'wb').write(r.content)
```

Out[34]: 12276695

In [35]:
```
# Create Dataframe from COVID dataset
# Keep the FIPS codes as string

covid_df = pd.read_csv('us_county_covid.csv', dtype={'fips':str}, parse_
dates=['date'])
covid_df.describe
```

Out[35]:
```
<bound method NDFrame.describe of               date      county        s
tate   fips  cases  deaths
0      2020-01-21   Snohomish  Washington  53061      1       0
1      2020-01-22   Snohomish  Washington  53061      1       0
2      2020-01-23   Snohomish  Washington  53061      1       0
3      2020-01-24        Cook    Illinois  17031      1       0
4      2020-01-24   Snohomish  Washington  53061      1       0
...           ...         ...         ...    ...    ...     ...
310251 2020-07-08  Sweetwater     Wyoming  56037    124       0
310252 2020-07-08       Teton     Wyoming  56039    149       1
310253 2020-07-08       Uinta     Wyoming  56041    192       0
310254 2020-07-08    Washakie     Wyoming  56043     42       5
310255 2020-07-08      Weston     Wyoming  56045      1       0

[310256 rows x 6 columns]>
```

Geographic Exceptions

This dataset has some geographic exceptions, noted [here (https://github.com/nytimes/covid-19-data#geographic-exceptions)](https://github.com/nytimes/covid-19-data#geographic-exceptions), including:

- Joplin, MO
- Kansas City, MO
- New York city

County FIPS are not assigned to these locations, so I will assign a fake FIPS code for each to use when comparing to income and population data.

Fake FIPS Codes

- NYC: 100001
- KC, MO: 100002
- Joplin, MO: 100003

In [36]:
```
covid_df.loc[(covid_df.county == 'New York City') & (covid_df.state ==
'New York'), 'fips'] = '100001'
covid_df.loc[(covid_df.county == 'Kansas City') & (covid_df.state == 'Mi
ssouri'), 'fips'] = '100002'
covid_df.loc[(covid_df.county == 'Joplin') & (covid_df.state == 'Missour
i'), 'fips'] = '100003'
```

The dataset also contains cases where the county is "unknown." We cannot use those cases in this analysis, so anything else that does not have a FIPS code should be dropped.

In [37]:
```
covid_df = covid_df.dropna(subset=['fips'])
```

Income Data

Next: need a source for median income data per county with FIPS codes. Source: [USDA Economic Research Service (https://www.ers.usda.gov/data-products/county-level-data-sets/download-data/)](https://www.ers.usda.gov/data-products/county-level-data-sets/download-data/)

In [38]:
```
# Download Excel file with median county income
url = 'https://www.ers.usda.gov/webdocs/DataFiles/48747/Unemployment.xl
s?v=1887.2'
r = requests.get(url, allow_redirects=True)
open('us_county_income.xls', 'wb').write(r.content)
```

Out[38]:
```
2322944
```

In [39]:
```
# Import income dataset
temp_income_df = pd.read_excel (r'us_county_income.xls', sheet_name='Une
mployment Med HH Income')
temp_income_df.dtypes
```

Out[39]:
```
Unemployment and median household income for the U.S., States, and coun
ties, 2000-19    object
Unnamed: 1
object
Unnamed: 2
object
Unnamed: 3
object
Unnamed: 4
object

...
Unnamed: 83
object
Unnamed: 84
object
Unnamed: 85
object
Unnamed: 86
object
Unnamed: 87
object
Length: 88, dtype: object
```

In [40]:

```
# The Excel file has lots of historical data that we don't need, so we w
ill only pull in the columns needed for FIPS and median income
# Also, skip the first 7 rows because this is a horribly formatted docum
ent
income_df = temp_income_df.iloc[ 7: , [0,2,3,86]]
income_df.columns = ['fips', 'county', 'rural_urban_code', 'median_incom
e']
income_df
```

Out[40]:

	fips	county	rural_urban_code	median_income
7	00000	United States	NaN	61937
8	01000	Alabama	NaN	49881
9	01001	Autauga County, AL	2	59338
10	01003	Baldwin County, AL	3	57588
11	01005	Barbour County, AL	6	34382
...	...	...	...	...
3277	72145	Vega Baja Municipio, PR	1	NaN
3278	72147	Vieques Municipio, PR	7	NaN
3279	72149	Villalba Municipio, PR	2	NaN
3280	72151	Yabucoa Municipio, PR	1	NaN
3281	72153	Yauco Municipio, PR	2	NaN

3275 rows × 4 columns

In [41]:
```
# The COVID dataset doesn't break down Puerto Rico by county and the inc
ome dataset doesn't have median income for the regions of PR, so we'll d
rop PR from the income dataset
# (Sorry Puerto Rico ❤️🇵🇷 )
income_df = income_df.dropna(subset=['median_income'])

# We also only want counties - not states, so we can drop anything whose
rural_urban_code is NaN
# Then we can drop that column entirely
income_df = income_df.dropna(subset=['rural_urban_code'])
income_df = income_df.drop(columns=['rural_urban_code'])
income_df
```

Out[41]:

	fips	county	median_income
9	01001	Autauga County, AL	59338
10	01003	Baldwin County, AL	57588
11	01005	Barbour County, AL	34382
12	01007	Bibb County, AL	46064
13	01009	Blount County, AL	50412
...	...	...	...
3198	56037	Sweetwater County, WY	73315
3199	56039	Teton County, WY	99087
3200	56041	Uinta County, WY	63401
3201	56043	Washakie County, WY	55190
3202	56045	Weston County, WY	54319

3141 rows × 3 columns

In [42]:
```
# Finally, we need to add in the median incomes for our three fake FIPS
 codes
# This data comes from the US Census Bureau
# https://www.census.gov/quickfacts/fact/table/newyorkcitynewyork,kansas
citycitymissouri,joplincitymissouri/AGE295219

d = {'fips': ['100001', '100002', '100003'],
     'county': ['New York City, NY', 'Kansas City, MO', 'Joplin, MO'],
     'median_income': [60762, 52405, 42782]}
extra_cities = pd.DataFrame(data=d)
income_df = income_df.append(extra_cities, ignore_index=True)
income_df
```

Out[42]:

	fips	county	median_income
0	01001	Autauga County, AL	59338
1	01003	Baldwin County, AL	57588
2	01005	Barbour County, AL	34382
3	01007	Bibb County, AL	46064
4	01009	Blount County, AL	50412
...	...	...	...
3139	56043	Washakie County, WY	55190
3140	56045	Weston County, WY	54319
3141	100001	New York City, NY	60762
3142	100002	Kansas City, MO	52405
3143	100003	Joplin, MO	42782

3144 rows × 3 columns

Population Data

Then, need a source for population data per county with FIPS codes. Source: [USDA Economic Research Service (https://www.ers.usda.gov/data-products/county-level-data-sets/download-data/)](https://www.ers.usda.gov/data-products/county-level-data-sets/download-data/)

In [43]:
```
# Download Excel file with county population
url = 'https://www.ers.usda.gov/webdocs/DataFiles/48747/PopulationEstima
tes.xls?v=1887.2'
r = requests.get(url, allow_redirects=True)
open('us_population.xls', 'wb').write(r.content)
```

Out[43]: 5619712

In [44]:
```
# Import population dataset
temp_pop_df = pd.read_excel (r'us_population.xls', sheet_name='Populatio
n Estimates 2010-19')
temp_pop_df.dtypes
```

Out[44]:
```
Population estimates for the U.S., States, and counties, 2010-19 (see t
he second tab in this workbook for variable name descriptions)    objec
t
Unnamed: 1
object
Unnamed: 2
object
Unnamed: 3
object
Unnamed: 4
object

...
Unnamed: 160
object
Unnamed: 161
object
Unnamed: 162
object
Unnamed: 163
object
Unnamed: 164
object
Length: 165, dtype: object
```

In [45]:

```
# This Excel file also has lots of historical data that we don't need, s
o we will only pull in the columns needed for FIPS and median income
# Also, skip the first 2 rows because this is a horribly formatted docum
ent
# And cut Puerto Rico out from the end of the file (sorry again)
pop_df = temp_pop_df.iloc[ 2:3196 , [0,2,3,19]]
pop_df.columns = ['fips', 'county', 'rural_urban_code', 'population_201
9']
pop_df
```

Out[45]:

	fips	county	rural_urban_code	population_2019
2	00000	United States	NaN	328239523
3	01000	Alabama	NaN	4903185
4	01001	Autauga County	2	55869
5	01003	Baldwin County	4	223234
6	01005	Barbour County	6	24686
...	...	...	...	...
3191	56037	Sweetwater County	5	42343
3192	56039	Teton County	7	23464
3193	56041	Uinta County	7	20226
3194	56043	Washakie County	7	7805
3195	56045	Weston County	7	6927

3194 rows × 4 columns

In [46]:
```
# We also only want counties - not states, so we can drop anything whose
rural_urban_code is NaN
# Then we can drop that column entirely
pop_df = pop_df.dropna(subset=['rural_urban_code'])
pop_df = pop_df.drop(columns=['rural_urban_code'])
pop_df
```

Out[46]:

	fips	county	population_2019
4	01001	Autauga County	55869
5	01003	Baldwin County	223234
6	01005	Barbour County	24686
7	01007	Bibb County	22394
8	01009	Blount County	57826
...	...	...	...
3191	56037	Sweetwater County	42343
3192	56039	Teton County	23464
3193	56041	Uinta County	20226
3194	56043	Washakie County	7805
3195	56045	Weston County	6927

3137 rows × 3 columns

In [47]:
```
# Finally, we need to add in the populations for our three fake FIPS cod
es
# This data comes from the US Census Bureau
# https://www.census.gov/quickfacts/fact/table/newyorkcitynewyork,kansas
citycitymissouri,joplincitymissouri/AGE295219

d = {'fips': ['100001', '100002', '100003'],
     'county': ['New York City, NY', 'Kansas City, MO', 'Joplin, MO'],
     'population_2019': [8336817, 495327, 50925]}
extra_cities = pd.DataFrame(data=d)
pop_df = pop_df.append(extra_cities, ignore_index=True)
pop_df
```

Out[47]:

	fips	county	population_2019
0	01001	Autauga County	55869
1	01003	Baldwin County	223234
2	01005	Barbour County	24686
3	01007	Bibb County	22394
4	01009	Blount County	57826
...	...	...	...
3135	56043	Washakie County	7805
3136	56045	Weston County	6927
3137	100001	New York City, NY	8336817
3138	100002	Kansas City, MO	495327
3139	100003	Joplin, MO	50925

3140 rows × 3 columns

Merging the Datasets

Now that the three datasets have been cleaned up, it's time to merge them together into one giant dataset

In [48]:
```
# Merge income and population dataframes together based on FIPS
income_pop_df = pd.merge(income_df, pop_df[['fips', 'population_2019']],
on='fips')
income_pop_df
```

Out[48]:

	fips	county	median_income	population_2019
0	01001	Autauga County, AL	59338	55869
1	01003	Baldwin County, AL	57588	223234
2	01005	Barbour County, AL	34382	24686
3	01007	Bibb County, AL	46064	22394
4	01009	Blount County, AL	50412	57826
...	...	...	...	...
3134	56043	Washakie County, WY	55190	7805
3135	56045	Weston County, WY	54319	6927
3136	100001	New York City, NY	60762	8336817
3137	100002	Kansas City, MO	52405	495327
3138	100003	Joplin, MO	42782	50925

3139 rows × 4 columns

In [49]:
```
# Now merge all three together
combined_df = pd.merge(covid_df, income_pop_df[['fips', 'median_income',
'population_2019']], on='fips')
#combined_df.to_csv('combined.csv')
combined_df.dtypes
```

Out[49]:
```
date                  datetime64[ns]
county                        object
state                         object
fips                          object
cases                          int64
deaths                         int64
median_income                 object
population_2019               object
dtype: object
```

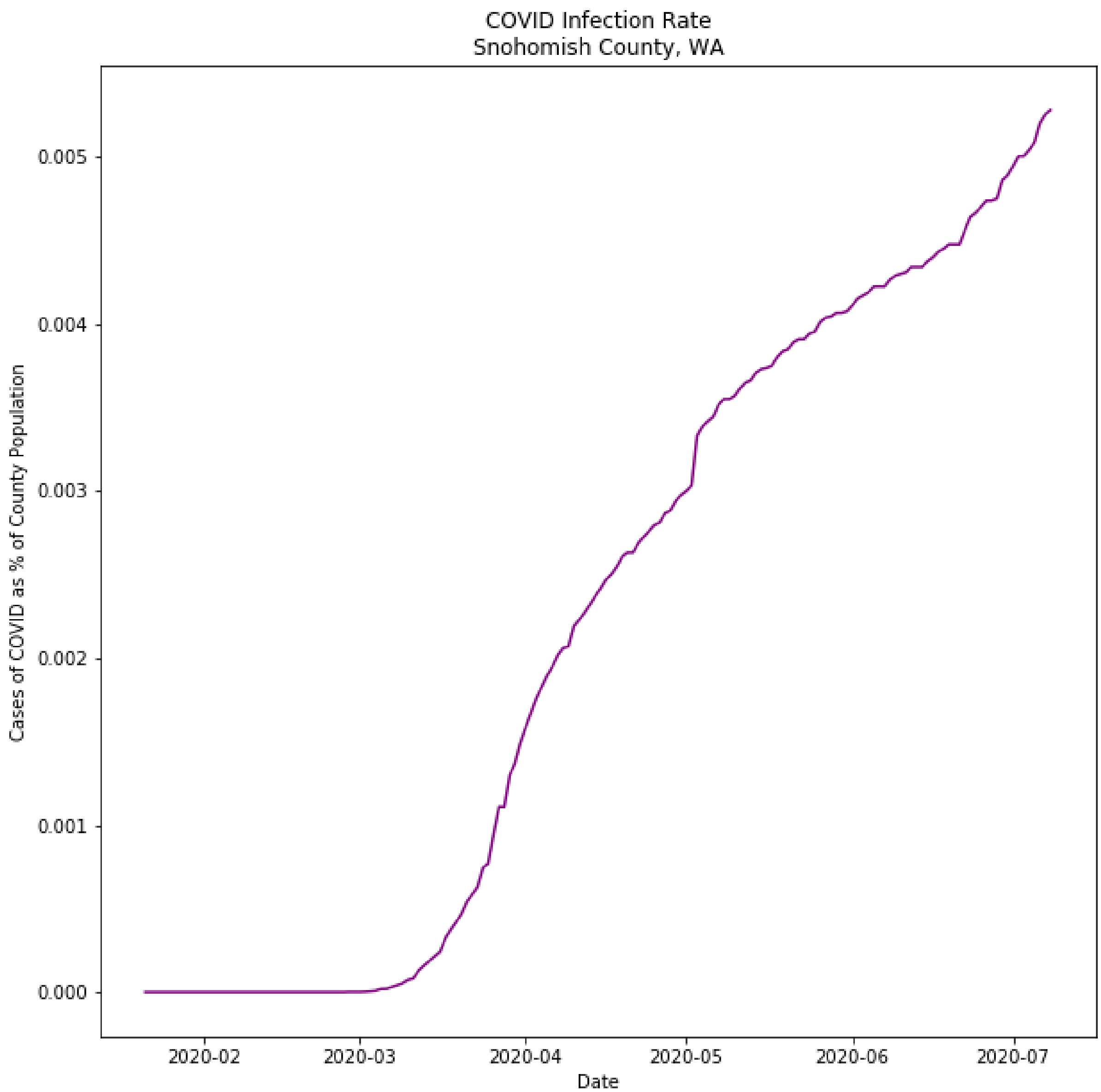
In [50]:
```
# Test with a DF for just one county
snohomish_df = combined_df[combined_df['fips']=='53061']
snohomish_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 170 entries, 0 to 169
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   date             170 non-null    datetime64[ns]
 1   county           170 non-null    object
 2   state            170 non-null    object
 3   fips             170 non-null    object
 4   cases            170 non-null    int64
 5   deaths           170 non-null    int64
 6   median_income    170 non-null    object
 7   population_2019  170 non-null    object
dtypes: datetime64[ns](1), int64(2), object(5)
memory usage: 12.0+ KB
```

Plotting Data

With a combined dataframe, we can try to plot some data

In [51]:
```
# Test a plot
# Create figure and plot space
fig, ax = plt.subplots(figsize=(10, 10))

# Add x-axis and y-axis
ax.plot(snohomish_df['date'],
        snohomish_df['cases']/snohomish_df['population_2019'],
        color='purple')

# Set title and labels for axes
ax.set(xlabel="Date",
       ylabel="Cases of COVID as % of County Population",
       title="COVID Infection Rate\nSnohomish County, WA")

plt.show()
```



Results by Income

Now that we have a dataset with COVID, income, and population data married together we can start to explore a bit.

For reference, the median US Census Bureau (https://bit.ly/320ptAZ) reports the median household income in 2018 was **$61,937**. In the dataset provided here, the county with the lowest median income is Wilcox County, AL ($25,385) and the county with the highest median income is Loudoun County, VA (\$140,382)

In [52]:
```
# Figure out populations of upper income and lower income counties
upper_pop = income_pop_df[income_pop_df['median_income']>61937]['population_2019'].sum()
lower_pop = income_pop_df[income_pop_df['median_income']<=61937]['population_2019'].sum()
print(f'{upper_pop} people live in counties where the median income is higher than the US median.')
print(f'{lower_pop} people live in counties where the median income is lower than the US median.')
```

```
161875067 people live in counties where the median income is higher than the US median.
175232137 people live in counties where the median income is lower than the US median.
```

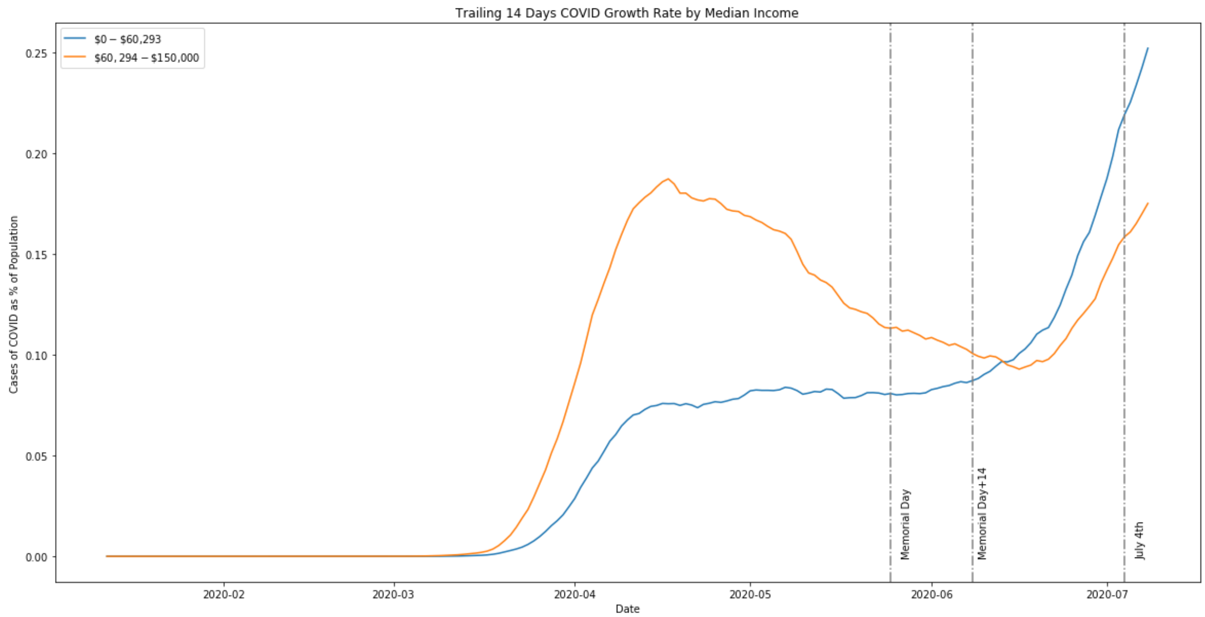
In [53]:
```
# Create a method that accepts an upper and lower salary range and calcu
lates cases for that population

# Set a date range from the start of data collection to yesterday
daterange = pd.date_range(dt.date(2020,1,12), (dt.date.today() - dt.time
delta(days=1)))

def cases_by_income(min,max):

    target_df = combined_df[combined_df.median_income>min]
    target_df = target_df[target_df.median_income<=max]

    cases_income_df = pd.DataFrame([], columns=['date', 'cases', '14-da
y', '28-day', 'population'])

    # Calculate population for this group
    group_population = income_pop_df[(income_pop_df.median_income>min)&(
income_pop_df.median_income<=max)]['population_2019'].sum()

    for single_date in daterange:
        cases_for_date = target_df[(target_df['date'] >= single_date) &
((target_df['date'] < single_date+dt.timedelta(days=1)))]['cases'].sum()

        cases_14_ago = target_df[(target_df['date'] >= single_date+dt.ti
medelta(days=-15)) & ((target_df['date'] < single_date+dt.timedelta(days
=-14)))]['cases'].sum()
        cases_28_ago = target_df[(target_df['date'] >= single_date+dt.ti
medelta(days=-29)) & ((target_df['date'] < single_date+dt.timedelta(days
=-28)))]['cases'].sum()

        daily = {'date' : single_date, 'cases' : cases_for_date, '14-da
y' : cases_for_date-cases_14_ago, '28-day' : cases_for_date-cases_28_ago
, 'population' : group_population}
        cases_income_df = cases_income_df.append(daily, ignore_index=Tru
e)

    return cases_income_df
```

In [54]:

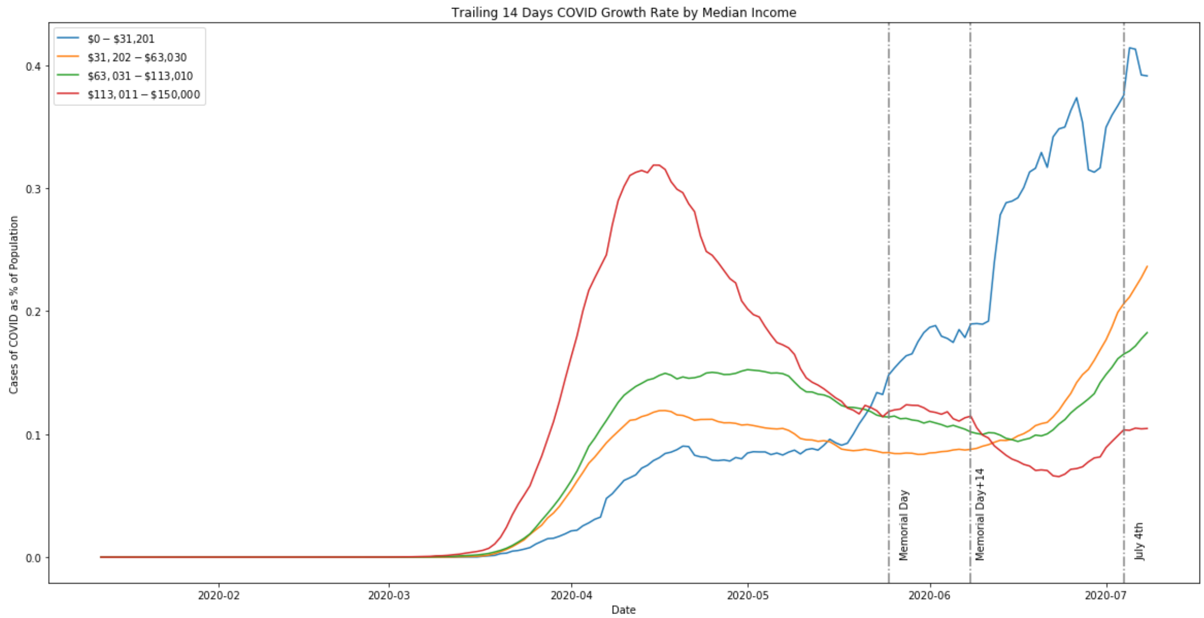
```
# Create a method that accepts an upper and lower population and calcula
tes cases for that population

# Set a date range from the start of data collection to yesterday
daterange = pd.date_range(dt.date(2020,1,12), (dt.date.today() - dt.time
delta(days=1)))

def cases_by_population(min,max):

    target_df = combined_df[combined_df.population_2019>min]
    target_df = target_df[target_df.population_2019<=max]

    cases_population_df = pd.DataFrame([], columns=['date', 'cases', '14
-day', '28-day', 'population'])

    # Calculate population for this group
    group_population = income_pop_df[(income_pop_df.population_2019>min)
&(income_pop_df.population_2019<=max)]['population_2019'].sum()

    for single_date in daterange:
        cases_for_date = target_df[(target_df['date'] >= single_date) &
((target_df['date'] < single_date+dt.timedelta(days=1)))]['cases'].sum()

        cases_14_ago = target_df[(target_df['date'] >= single_date+dt.ti
medelta(days=-15)) & ((target_df['date'] < single_date+dt.timedelta(days
=-14)))]['cases'].sum()
        cases_28_ago = target_df[(target_df['date'] >= single_date+dt.ti
medelta(days=-29)) & ((target_df['date'] < single_date+dt.timedelta(days
=-28)))]['cases'].sum()

        daily = {'date' : single_date, 'cases' : cases_for_date, '14-da
y' : cases_for_date-cases_14_ago, '28-day' : cases_for_date-cases_28_ago
, 'population' : group_population}
        cases_population_df = cases_population_df.append(daily, ignore_i
ndex=True)

    return cases_population_df
```

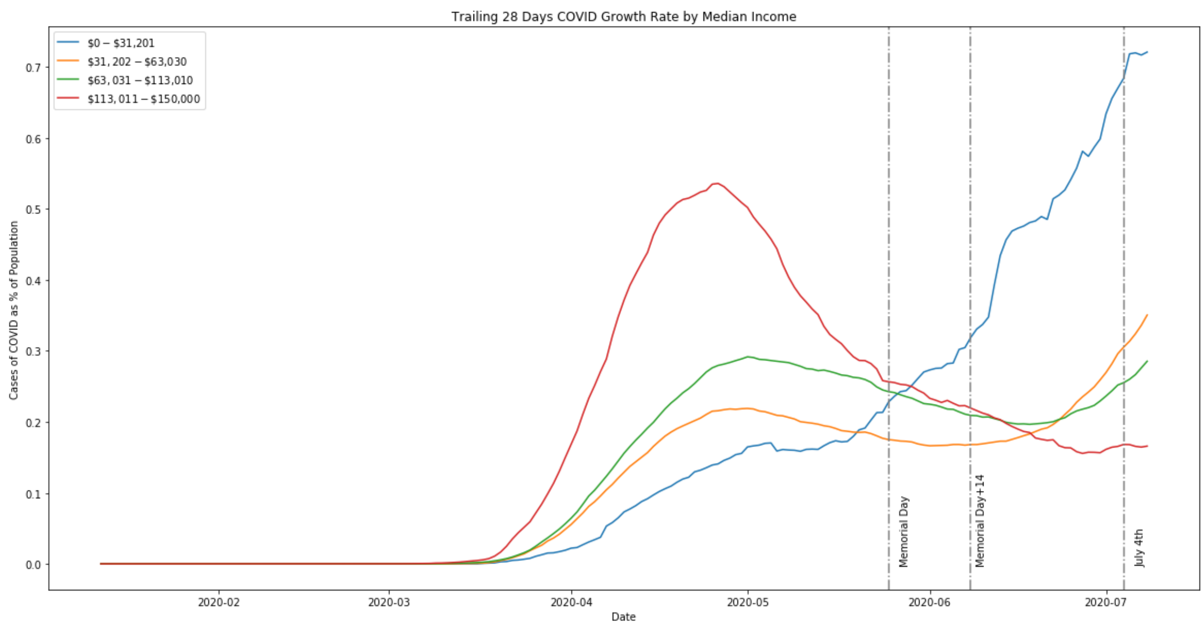
In [55]:
```
# Create a method that plots the change in cases/population for a series
of salary ranges
# Inputs are
# - an array of ranges [low, high]
# - either 14 or 28 for the trailing average
# - 'i' for income or 'p' for population

def plot_by_ranges(ranges, trail_days, metric):
    # Create figure and plot space
    fig, ax = plt.subplots(figsize=(20, 10))

    for range in ranges:

        if metric == 'i':
            range_label = "\$" + "${:,.0f}".format(range[0]) + " - \$" +
"${:,.0f}".format(range[1])
            title_metric = 'Median Income'
            range_df = cases_by_income(range[0],range[1])
        else:
            range_label = "${:,.0f}".format(range[0]) + " - " + "${:,.0
f}".format(range[1])
            title_metric = 'County Population'
            range_df = cases_by_population(range[0],range[1])


        if trail_days == 14:
            ax.plot(range_df['date'],
                    (range_df['14-day']/range_df['population'])*100,
                    label=range_label)
            title_days = "14"
        else:
            ax.plot(range_df['date'],
                    (range_df['28-day']/range_df['population'])*100,
                    label=range_label)
            title_days = "28"

    # Set title and labels for axes
    ax.set(xlabel="Date",
           ylabel="Cases of COVID as % of Population",
           title= "Trailing " + title_days + " Days COVID Growth Rate by
" + title_metric)

    plt.axvline(dt.datetime(2020, 5, 25), linestyle="-.", color="gray")
    plt.text(dt.datetime(2020, 5, 27),0,'Memorial Day',rotation=90)

    plt.axvline(dt.datetime(2020, 6, 8), linestyle="-.", color="gray")
    plt.text(dt.datetime(2020, 6, 9),0,'Memorial Day+14',rotation=90)

    plt.axvline(dt.datetime(2020, 7, 4), linestyle="-.", color="gray")
    plt.text(dt.datetime(2020, 7, 6),0,'July 4th',rotation=90)

    plt.legend(loc="upper left")
    plt.show()
```

In [56]:
```
# Divide the country into upper/lower income halves
plot_by_ranges([[0,60293],[60294,150000]],14,'i')
```



In [57]:
```
# Divide the country into quartiles
plot_by_ranges([[0,31201],[31202,63030],[63031,113010],[113011,150000]],
14,'i')
plot_by_ranges([[0,31201],[31202,63030],[63031,113010],[113011,150000]],
28,'i')
```



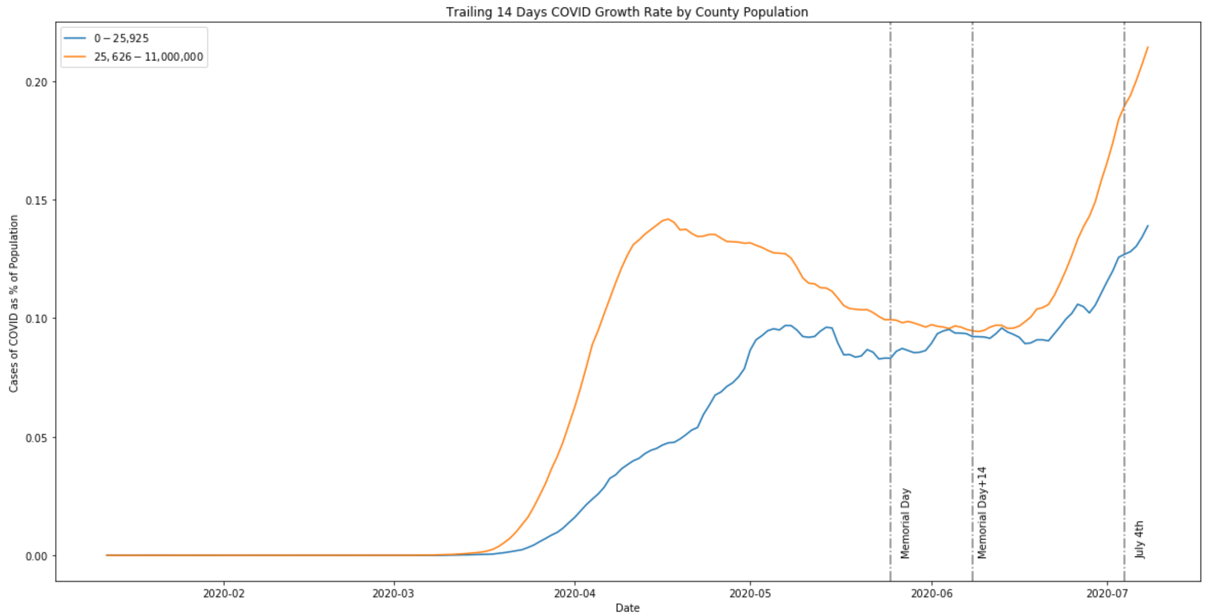


Results by Population

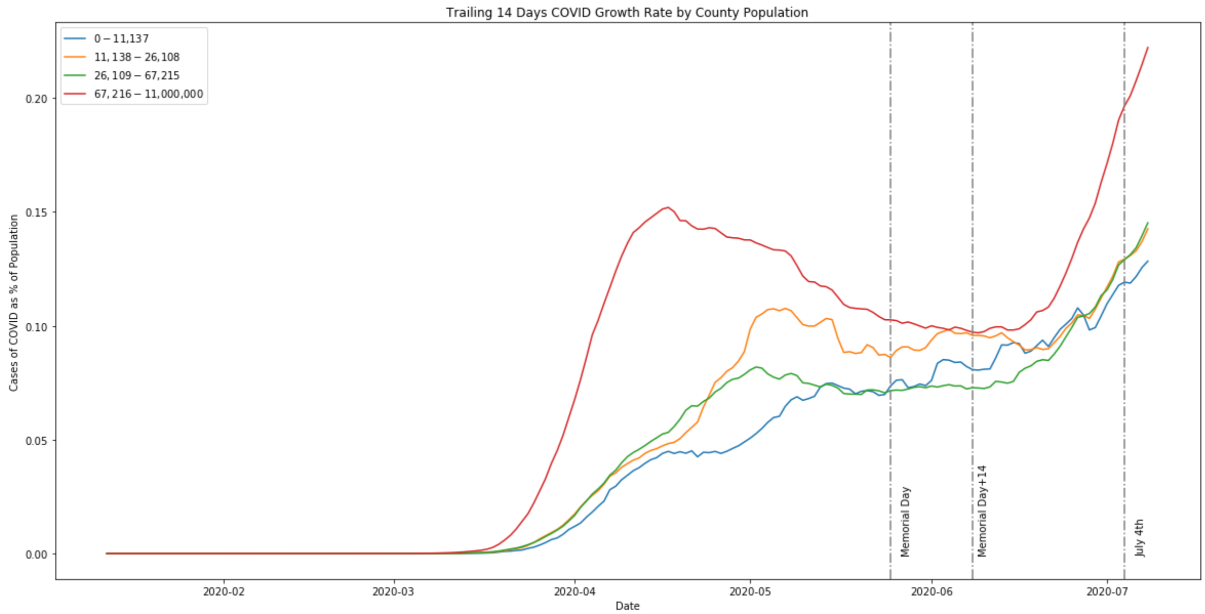
And although this started as an exercise to examine the differences in income level, we can do the same type of analysis based on the population of the counties.

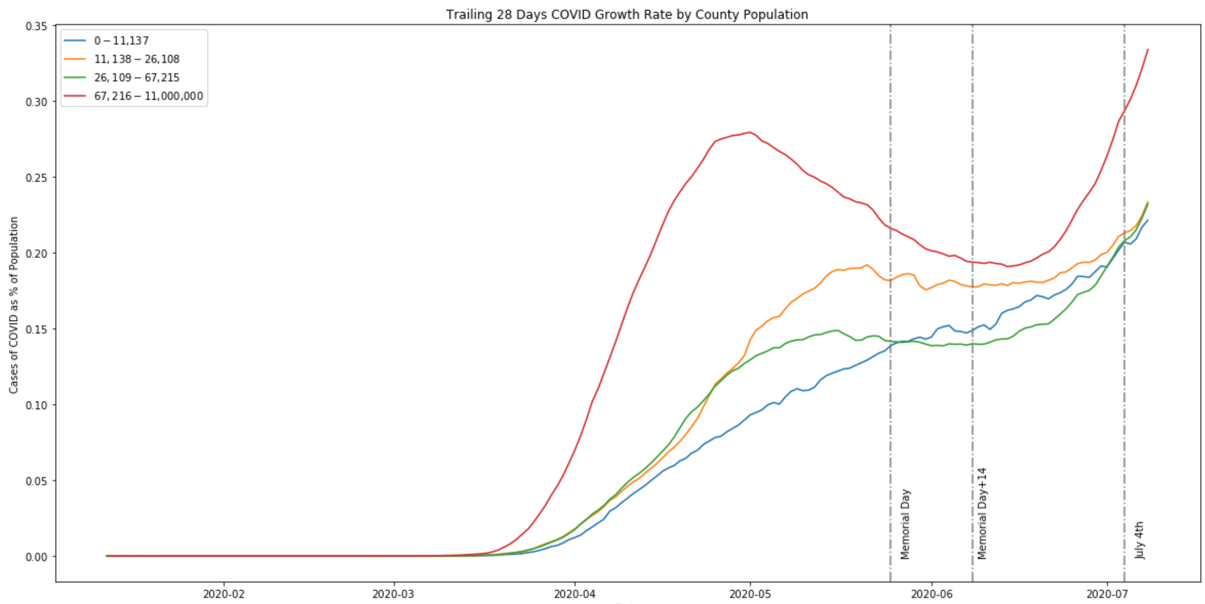
For reference, the median county population in our dataset is 25,925. The least populated county is Kalawao County, HI (86) and the most populated county is Los Angeles County, CA (10,039,107).

In [58]:
```
# Divide the country into upper/lower population halves
plot_by_ranges([[0,25925],[25626,11000000]],14,'p')
```



In [59]:
```
# Divide the country into population quartilesishes
plot_by_ranges([[0,11137],[11138,26108],[26109,67215],[67216,11000000]],
14,'p')
plot_by_ranges([[0,11137],[11138,26108],[26109,67215],[67216,11000000]],
28,'p')
```





In [ ]: