CS 649 Big Data: Tools and Methods
Fall Semester, 2021
Doc 16 Assignment 1 Comments
Mar 16, 2021

General Comments

Before you turn in notebook to your boss or as an assignment

Restart the kernel
Run all the cells in order from top to bottom

You can run the notebook and turn it in with answers

Functions are your friend

Avoid magic values

```
weekly_counts = confirmed_deaths_states.loc[:, '2020-01-27'::7]

first_sunday = '2020-01-26'
```

Commenting

X = X + 1 // Add 1 to X

Commenting

X = X + 1 // Add 1 to X

```
def group_and_locate(casedeath): #function to get county ID grouped weekly total values from raw data
    casedeath = casedeath[casedeath['countyFIPS']!=0] #for not including Statewide Unollocated (countyFIPS =
    casedeaths = casedeath.drop(["State", "StateFIPS"], axis=1) #drop 'State' and 'StateFIPS' labels
    casedeaths = casedeaths.groupby('countyFIPS').sum().loc[:,'2020-01-26'::7] # Groupy by countyFIPS
                                                    (county IDs) and include every 7th value after 2020-01-26
    return casedeaths
```

Commenting

X = X + 1 // Add 1 to X

```
def group_and_locate(casedeath):
    casedeath = casedeath[casedeath['countyFIPS']!=0]
    casedeaths = casedeath.drop(["State", "StateFIPS"], axis=1) #drop 'State' and 'StateFIPS' labels
    casedeaths = casedeaths.groupby('countyFIPS').sum().loc[:,'2020-01-26'::7] # Groupy by countyFIPS
                                        (county IDs) and include every 7th value after 2020-01-26
    return casedeaths
```

Commenting

X = X + 1 // Add 1 to X

```
def group_and_locate(casedeath):
    casedeath = casedeath[casedeath['countyFIPS']!=0]
    casedeaths = casedeath.drop(["State", "StateFIPS"], axis=1)
    casedeaths = casedeaths.groupby('countyFIPS').sum().loc[:,'2020-01-26'::7] # Groupy by countyFIPS
                                        (county IDs) and include every 7th value after 2020-01-26
    return casedeaths
```

Commenting

X = X + 1 // Add 1 to X

```
def group_and_locate(casedeath): #function to get county ID grouped weekly total values from raw data
    casedeath = casedeath[casedeath['countyFIPS']!=0] #for not including Statewide Unollocated (countyFIPS =
    casedeaths = casedeath.drop(["State", "StateFIPS"], axis=1) #drop 'State' and 'StateFIPS' labels
    casedeaths = casedeaths.groupby('countyFIPS').sum().loc[:,'2020-01-26'::7] # Groupy by countyFIPS
                                                    (county IDs) and include every 7th value after 2020-01-26
    return casedeaths
```

```
def group_and_locate(casedeath):
    unallocated_removed = casedeath[casedeath['countyFIPS']!=0]
    state_columns_removed = unallocated_removed.drop(["State", "StateFIPS"], axis=1)
    weekly_values = state_columns_removed.groupby('countyFIPS').sum().loc[:,'2020-01-26'::7]
    return weekly_values
```

Commenting

Comment on the why not the what

If the what is complex or obscure

- Consider a separate function for the what
- Function name indicates what it does
- Document the what inside

```
def data_clean_merge(df1,df2):
    '''
        This function takes two data frames and cleans the not needed columns and
              merge them on the 'countyFIPS' as key.
        paramaters:
        df1 : Cases or Deaths Data Frame
        df2 : Population Data Frame

        Returns:
        a merged data frame of df1 and df2 by='countyFIPS'
    '''
    df1 = df1[df1['countyFIPS']!=0]
    df1.drop('StateFIPS',axis=1,inplace=True)
    df1 = df1.groupby('countyFIPS').sum()
    df1.drop(columns=df1.columns[0:4],axis=1,inplace=True)
    df1 = df1.iloc[:,::7].diff(axis=1)
    df2 = df2[df2['countyFIPS']!=0]
    merged_df = df1.merge(df2[['countyFIPS','County Name','State','population',]],on='countyFIPS')
    return merged_df
```

What is data_frame? list?

```
def new_weekly_deaths(data_frame, list):
    confirmed_deaths_states = data_frame.loc[data_frame['State'].isin(states)]
    weekly_counts = confirmed_deaths_states.loc[:, '2020-01-27'::7]
    counties = confirmed_deaths_states.loc[:, 'County Name':'State']
    df = pd.merge(counties, weekly_counts, left_index=True, right_index=True)
    df_short = pd.melt(df, id_vars=['County Name', 'State'], var_name='Date', value_name='Cases')
    new_deaths = df_short.groupby(['State', 'Date'])['Cases'].sum().diff()
    new_deaths.loc[new_cases.isna()] = 2.0
    new_deaths['NV', '2020-01-27'] = 0
    new_deaths['OR', '2020-01-27'] = 0
    new_deaths['WA', '2020-01-27'] = 1
    new_deaths.unstack()
    return new_deaths
```

Documenting

```
def new_weekly_deaths(data_frame, list):
        bunch of code
```

Without knowing the structure of data_frame & list how can we know what function does

What do the names data_frame & list tell us?

Document your data structures

Why?

```
confirmed_cases = confirmed_cases.drop(['2020-01-22', '2020-01-23','2020-01-24', '2020-01-25',
'2020-01-26'], axis=1)
```

What are these date?

plt.xticks(np.arange(0, 53, step=10), ['1/27/20', '4/06/20', '6/15/20', '8/24/20', '11/2/20', '1/11/21'])

Run you Notebooks after restarting the Kernel

```
county_pop = county_pop[county_pop.population != 0]
county_pop['countypop_per_100K'] = county_pop['population'].div(100000)
county_pop=county_pop.drop(columns=['population','County Name'])
county_pop
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call las
<ipython-input-18-f2eedabd3072> in <module>
     18 # Allocated states has 0 and can therefore be eliminated
     19
---> 20 county_pop = county_pop[county_pop.population != 0]
     21 county_pop['countypop_per_100K'] =
county_pop['population'].div(100000)
     22 county_pop=county_pop.drop(columns=['population','County Name'])

NameError: name 'county_pop' is not defined
```

A1

In [2]:
```
confirmed_cases = pd.read_csv('/Users/whitney/Courses/649/Spring21/assignments/assignment1/covid_confirmed_usafa

#display(confirmed_cases)

confirmed_cases = confirmed_cases.drop(['2020-01-22', '2020-01-23','2020-01-24', '2020-01-25', '2020-01-26'], a

#display(confirmed_cases)

```

In [3]:
```
confirmed_cases = confirmed_cases.drop(['countyFIPS', 'StateFIPS'], axis = 1)

# confirmed_cases
```

In [4]:
```
confirmed_cases = confirmed_cases.query('State in ["CA", "OR", "WA", "NV"]')

# confirmed_cases
```

In [5]:
```
confirmed_cases = confirmed_cases.drop(['County Name'], axis = 1)

# confirmed_cases
```

In [6]:
```
confirmed_cases = confirmed_cases.groupby('State').sum()
# confirmed_cases
```

In [7]:
```
weeklycounts = confirmed_cases.loc[:, '2020-02-02'::7]
weeklycounts.columns = weeklycounts.columns.map(lambda d: datetime.strptime(d, '%Y-%m-%d') - timedelta(days=6))
# weeklycounts
```

In [8]:
```
sample_weekly = weeklycounts

# sample_weekly
```

A3

In [12]:
```
covid_deaths = pd.read_csv('covid_deaths_usafacts.csv')
# covid_deaths

```

In [13]:
```
covid_deaths = covid_deaths.drop(['2020-01-22', '2020-01-23','2020-01-24', '2020-01-25', '2020-01-26'], axis=1)
# covid_deaths
```

In [14]:
```
covid_deaths = covid_deaths.drop(['countyFIPS', 'StateFIPS'], axis = 1)
```

In [15]:
```
# covid_deaths
```

In [16]:
```
covid_deaths = covid_deaths.query('State in ["CA", "OR", "WA", "NV"]')
covid_deaths = covid_deaths.drop(['County Name'], axis = 1)
# covid_deaths
```

In [17]:
```
covid_deaths = covid_deaths.groupby('State').sum()
```

In [18]:
```

weeklydeaths = covid_deaths.loc[:, '2020-02-02'::7]
weeklydeaths.columns = weeklydeaths.columns.map(lambda d: datetime.strptime(d, '%Y-%m-%d') - timedelta(days=6))
# weeklydeaths
```

In [19]:
```
sample_deaths = weeklydeaths
sample_deaths.T
sample_deaths.reset_index()
```

A1, A3

Same Calculation - just different input

Why all the repeated code?

```
def new_weekly_deaths(data_frame, list):
    confirmed_deaths_states = data_frame.loc[data_frame['State'].isin(states)]
    weekly_counts = confirmed_deaths_states.loc[:, '2020-01-27'::7]
    counties = confirmed_deaths_states.loc[:, 'County Name':'State']
    df = pd.merge(counties, weekly_counts, left_index=True, right_index=True)
    df_short = pd.melt(df, id_vars=['County Name', 'State'], var_name='Date', value_name='Cases')
    new_deaths = df_short.groupby(['State', 'Date'])['Cases'].sum().diff()
    new_deaths.loc[new_cases.isna()] = 2.0
    new_deaths['NV', '2020-01-27'] = 0
    new_deaths['OR', '2020-01-27'] = 0
    new_deaths['WA', '2020-01-27'] = 1
    new_deaths.unstack()
    return new_deaths
```

What is Different?

```
def new_weekly_state_cases(data_frame, list):
    confirmed_state_cases = data_frame.loc[data_frame['State'].isin(states)]
    weekly_state_cases = confirmed_state_cases.loc[:, '2020-01-27'::7]
    counties = confirmed_state_cases.loc[:, 'County Name':'State']
    df = pd.merge(counties, weekly_state_cases, left_index=True, right_index=True)
    df_short = pd.melt(df, id_vars=['County Name', 'State'], var_name='Date', value_name='Cases')
    new_cases = df_short.groupby(['State', 'Date'])['Cases'].sum().diff()
    new_cases.loc[new_cases.isna()] = 2.0
    new_cases['NV', '2020-01-27'] = 0
    new_cases['OR', '2020-01-27'] = 0
    new_cases['WA', '2020-01-27'] = 1
    new_cases.unstack()
    return new_cases
```

Why Repeat the function?

```
# WA weekly new COViD-19 cases
plt.figure(figsize=(10,6))
# Function weekly_new_cases() takes in a Series object as an argument and returns a DataFrame object.
plt.plot(weekly_new_cases(washington_new_cases).values, '-or')
plt.xlabel('Weeks: 01/27/20 - 02/07/21 (Monday - Sunday)', fontsize=15);
plt.ylabel('Number of New COVID-19 Cases', fontsize=15);
plt.title('Fig.1-3 Weekly Washington New COVID-19 Cases', fontsize=18)
plt.xticks(np.arange(0, 53, step=10), ['1/27/20', '4/06/20', '6/15/20', '8/24/20', '11/2/20', '1/11/21'])
plt.tight_layout()

# OR weekly new COViD-19 cases
plt.figure(figsize=(10,6))
# Function weekly_new_cases() takes in a Series object as an argument and returns a DataFrame object.
plt.plot(weekly_new_cases(oregon_new_cases).values, '-og')
plt.xlabel('Weeks: 01/27/20 - 02/07/21 (Monday - Sunday)', fontsize=15);
plt.ylabel('Number of New COVID-19 Cases',  fontsize=15);
plt.title('Fig.1-2 Weekly Oregon New COVID-19 Cases', fontsize=18)
plt.xticks(np.arange(0, 53, step=10), ['1/27/20', '4/06/20', '6/15/20', '8/24/20', '11/2/20', '1/11/21'])
plt.tight_layout()
```

Why the repeated code

Now can use one function

```
def print_state(state_df, state_name):
    plt.figure(figsize=(10,6))
    # Function weekly_new_cases() takes in a Series object as an argument and returns a DataFrame object.
    plt.plot(weekly_new_cases(state_df).values, '-og')
    plt.xlabel('Weeks: 01/27/20 - 02/07/21 (Monday - Sunday)', fontsize=15);
    plt.ylabel('Number of New COVID-19 Cases',  fontsize=15);
    plt.title('Weekly Oregon New COVID-19 Cases', fontsize=18)
    plt.xticks(np.arange(0, 53, step=10), ['1/27/20', '4/06/20', '6/15/20', '8/24/20', '11/2/20', '1/11/21'])
    plt.tight_layout()
```

Question 1

```
def in_states(df):
    states = ['CA', 'OR', 'WA', 'NV']

    new_df = df.loc[df['State'].isin(states)]
    new_df = new_df.drop(columns=['countyFIPS', 'StateFIPS'])
    new_df = new_df.groupby('State').sum()
    return new_df
```

Why restrict to 4 state

```
def weekly_new_data(df):
    first_monday = '2020-01-26'

    new_df = df.loc[:,first_monday::7]
    new_df = new_df.diff(axis=1).drop(columns=[first_monday])
    return new_df

cases_in_CA_OR_WA_NV = weekly_new_data(in_states(cases))
cases_in_CA_OR_WA_NV.T.plot.line()
```

Why hard code the date?

pandas.DataFrame.diff

Calculates the difference of a Dataframe element compared with another element in the Dataframe (default is element in previous row).

```
>>> df = pd.DataFrame({'a': [1, 2, 3, 4, 5, 6],
...                    'b': [1, 1, 2, 3, 5, 8],
...                    'c': [1, 4, 9, 16, 25, 36]})
>>> df
   a  b   c
0  1  1   1
1  2  1   4
2  3  2   9
3  4  3  16
4  5  5  25
5  6  8  36
```

```
>>> df.diff(axis=1)
    a  b   c
0 NaN  0   0
1 NaN -1   3
2 NaN -1   7
3 NaN -1  13
4 NaN  0  20
5 NaN  2  28
```

```
new_df = df[df.columns[4:]]
x, y = largest_values(new_df,20)
print("Covid confirmed rate\tdate\t\t\tcounty")
for j,k in zip(x,y):
    print(new_df.iloc[j,k], end = "\t")
    print(new_df.columns[k], end = "\t")
    print(df.iloc[j,1])
```

What is new_df?

What is x?, y?

Names

```
cnfmCnty = confirmData.drop(0).drop(columns=['State', 'StateFIPS'])
cnfmCnty = cnfmCnty - cnfmCnty.shift(axis=1)
cnfmCnty = cnfmCnty.drop(columns=cnfmCnty.columns[0:5])
```

How can cnfmCnty represent all three?

What does this do?

```
cnfmCnty = confirmData.drop(0).drop(columns=['State', 'StateFIPS'])
```

What does this do?

```
confirmData.drop(0)
```

Names

```
cnfmCnty = confirmData.drop(0).drop(columns=['State', 'StateFIPS'])
cnfmCnty = cnfmCnty - cnfmCnty.shift(axis=1)
cnfmCnty = cnfmCnty.drop(columns=cnfmCnty.columns[0:5])
```

```
no_unallocated = confirmData.drop(0)
no_state_labels = no_unallocated.drop(columns=['State', 'StateFIPS'])
new_cases = no_state_labels - no_state_labels.shift(axis=1)
daily_data_only = new_cases.drop(columns= new_cases.columns[0:5])
```

```
no_unallocated = remove_unallocated(confirmData)
no_state_labels = remove_state_labels(no_unallocated)
new_cases = no_state_labels - no_state_labels.shift(axis=1)
daily_data_only = remove_text_colums(new_cases)
```

```
cnfmCnty = confirmData.drop(0).drop(columns=['State', 'StateFIPS'])
cnfmCnty = cnfmCnty - cnfmCnty.shift(axis=1)
cnfmCnty = cnfmCnty.drop(columns=cnfmCnty.columns[0:5])
```

```
TypeError                                 Traceback (most recent call last)
<ipython-input-6-96529e0efaaa> in <module>
      1 cnfmCnty = confirmData.drop(0).drop(columns=['State', 'StateFIPS'])
----> 2 cnfmCnty = cnfmCnty - cnfmCnty.shift(axis=1)
      3 cnfmCnty = cnfmCnty.drop(columns=cnfmCnty.columns[0:5])

TypeError: unsupported operand type(s) for -: 'int' and 'str'
```

```
# Corrected your error above -5
cnfmCnty = confirmData.drop(0).drop(columns=['State', 'StateFIPS'])
cnfmCnty = cnfmCnty.drop(columns=cnfmCnty.columns[0:5])
cnfmCnty = cnfmCnty - cnfmCnty.shift(axis=1)
```

In [22]:
```
# Fetch the total number of covid deaths in each county
total_death_counts = df_covid_deaths_grouped_by_county.loc[:, '2021-02-07'::1]
counties_states = df_covid_deaths_by_county.loc[:, 'countyFIPS':'StateFIPS'] #states
df_total_death_counts = pd.merge(counties_states, total_death_counts, left_index=True, right_index=True)
df_total_death_counts.rename(columns={'2021-02-07':'Total Deaths'}, inplace=True)
df_total_death_counts
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-22-d2883ec406b2> in <module>
      1 # Fetch the total number of covid deaths in each county
      2 total_death_counts = df_covid_deaths_grouped_by_county.loc[:, '2021-02-07'::1]
----> 3 counties_states = df_covid_deaths_by_county.loc[:, 'countyFIPS':'StateFIPS'] #states
      4 df_total_death_counts = pd.merge(counties_states, total_death_counts, left_index=True, right_index=True)
      5 df_total_death_counts.rename(columns={'2021-02-07':'Total Deaths'}, inplace=True)

NameError: name 'df_covid_deaths_by_county' is not defined
```

Corrected your typo in above -5

In [23]:
```
#Corrected your typo in above -5
total_death_counts = df_covid_deaths_grouped_by_county.loc[:, '2021-02-07'::1]
counties_states = df_covid_deaths_grouped_by_county.loc[:, 'countyFIPS':'StateFIPS'] #states
df_total_death_counts = pd.merge(counties_states, total_death_counts, left_index=True, right_index=True)
df_total_death_counts.rename(columns={'2021-02-07':'Total Deaths'}, inplace=True)
df_total_death_counts
```

```
def group_and_locate(casedeath): #function to get county ID grouped weekly total values from raw data
    casedeath = casedeath[casedeath['countyFIPS']!=0] #for not including Statewide Unollocated (countyFIPS = 0)
    casedeaths = casedeath.drop(["State", "StateFIPS"], axis=1) #drop 'State' and 'StateFIPS' labels
    casedeaths = casedeaths.groupby('countyFIPS').sum().loc[:,'2020-01-26'::7] # Groupy by countyFIPS
                                                    (county IDs) and include every 7th value after 2020-01-26
    return casedeaths
```

```
def weekly_cases(casedeath):
    no_unallocated = casedeath[casedeath['countyFIPS']!=0]
    no_state_stateFIPS = no_unallocated.drop(["State", "StateFIPS"], axis=1)
    weekly_cases = no_state_stateFIPS.groupby('countyFIPS').sum().loc[:,'2020-01-26'::7]
    return weekly_cases
```

What is the valid input?

Using Python Commenting Standards

```
def merge_with_pop(casedeath, pop):
    '''merge cases or deaths data to population data (every 7th day) and getting difference for new cases
    '''
    casedeaths = group_and_locate(casedeath)
    pop = pop[pop['countyFIPS']!=0]

    #weekly new values
    casedeathdiff = casedeaths.diff(axis=1).dropna(axis=1)

    casedeathandpop = casedeathdiff.merge(pop[['countyFIPS','County Name', 'population',]],
                                          on='countyFIPS')
    return casedeathandpop
```

```
def divide_date_values(casedeath):
    df = casedeath
    df.loc[:,'2020-02-02':'2021-02-07'] =
            (casedeath.loc[:,'2020-02-02':'2021-02-07']*100000).div(casedeath['population'],axis=0)
    return df #we will need it to get rates

def merge_divide(casedeath, pop):
   return divide_date_values(merge_with_pop(casedeath,pop))
```

What is casedeath?

What does this do?

df = casedeath

Why these dates?

df.loc[:,'2020-02-02':'2021-02-07']

Can we run the code now to get updated results?

```
def top20_pop(casedeath, pop): #function to get top 20 result
    merged_data = merge_divide(casedeath, pop).drop(['population','countyFIPS'],axis=1)

    merged_data_melted = pd.melt(merged_data, id_vars=['County Name'],
                                                    var_name='Date', value_name='Rate')

    top20_popul = merged_data_melted.max(axis=1).sort_values(ascending=False).head(20)

    result = pd.DataFrame(columns=['Date','County Name','Rate'])

    #getting indexes of top20 to get the locations within merged_data_melted and take values from there
to make new datafraem
    indexlist = top20_popul.index

    for i in range(0,20):
        result = result.append({'Date':merged_data_melted['Date'][indexlist[i]],'
                                County Name':merged_data_melted['County Name'][indexlist[i]],
                                'Rate':merged_data_melted['Rate'][indexlist[i]]},
                                ignore_index=True)
    return result
```

melt

```
>>> df = pd.DataFrame({'A': {0: 'a', 1: 'b', 2: 'c'},
...                    'B': {0: 1, 1: 3, 2: 5},
...                    'C': {0: 2, 1: 4, 2: 6}})
>>> df
   A  B  C
0  a  1  2
1  b  3  4
2  c  5  6
```

```
>>> pd.melt(df, id_vars=['A'], value_vars=['B'])
   A variable  value
0  a        B      1
1  b        B      3
2  c        B      5
```

```
>>> pd.melt(df, id_vars=['A'], value_vars=['B', 'C'])
   A variable  value
0  a        B      1
1  b        B      3
2  c        B      5
3  a        C      2
4  b        C      4
5  c        C      6
```

What does this do?

```
def top20_pop_example(casedeath, pop): #function to get top 20 result
    merged_data = merge_divide(casedeath, pop).drop(['population','countyFIPS'],axis=1)
    merged_data_melted = pd.melt(merged_data, id_vars=['County Name'], var_name='Date',
                                                                    value_name='Rate')
    top20_popul = merged_data_melted.max(axis=1).sort_values(ascending=False).head(20)
    return top20_popul

top20_pop_example(cases, population)
```

Had to Step through function

```
cases = pd.read_csv(path+"/covid_confirmed_usafacts.csv")
population = pd.read_csv(path+"/covid_county_population_usafacts.csv")
deaths = pd.read_csv(path+"covid_deaths_usafacts.csv")

def top20_pop_example(casedeath, pop): #function to get top 20 result
    merged_data = merge_divide(casedeath, pop).drop(['population','countyFIPS'],axis=1)
    return merged_data
top20_pop_example(cases, population)
```

	2020-02-02	2020-02-09	2020-02-16	2020-02-23	2020-03-01	2020-03-08	2020-03-15
0	0.0	0.0	0.0	0.0	0.00000	0.00000	10.739408	10.739408	

```
cases = pd.read_csv(path+"/covid_confirmed_usafacts.csv") # original raw data of new cases
population = pd.read_csv(path+"/covid_county_population_usafacts.csv") # original raw data of population
deaths = pd.read_csv(path+"covid_deaths_usafacts.csv") # original raw data of deaths

def top20_pop_example(casedeath, pop): #function to get top 20 result
    merged_data = merge_divide(casedeath, pop).drop(['population','countyFIPS'],axis=1)
    merged_data_melted = pd.melt(merged_data, id_vars=['County Name'],
                                 var_name='Date', value_name='Rate')
    return merged_data_melted

top20_pop_example(cases, population)
```

	County Name	Date	Rate
0	Autauga County	2020-02-02	0.000000
1	Baldwin County	2020-02-02	0.000000
2	Barbour County	2020-02-02	0.000000
3	Bibb County	2020-02-02	0.000000
4	Blount County	2020-02-02	0.000000

What is the Output?

```
def top20_pop_example(casedeath, pop): #function to get top 20 result
    merged_data = merge_divide(casedeath, pop).drop(['population','countyFIPS'],axis=1)
    merged_data_melted = pd.melt(merged_data, id_vars=['County Name'], var_name='Date',
                                                                    value_name='Rate')
    top20_popul = merged_data_melted.max(axis=1).sort_values(ascending=False).head(20)
    return top20_popul

top20_pop_example(cases, population)
```

```
43358     11503.013116
88328     10009.498931
162512     9134.814406
128240     8335.614563
129079     7688.500247
```

```
pop_df = pd.read_csv(PATH+"/covid_county_population_usafacts.csv")
case_df = pd.read_csv(PATH+"/covid_confirmed_usafacts.csv")
merged_data = data_clean_merge(case_df,pop_df)
merged_data.columns
```

```
Index(['countyFIPS', '2020-01-26', '2020-02-02', '2020-02-09', '2020-02-16',
       '2020-02-23', '2020-03-01', '2020-03-08', '2020-03-15', '2020-03-22',
       '2020-03-29', '2020-04-05', '2020-04-12', '2020-04-19', '2020-04-26',
       '2020-05-03', '2020-05-10', '2020-05-17', '2020-05-24', '2020-05-31',
       '2020-06-07', '2020-06-14', '2020-06-21', '2020-06-28', '2020-07-05',
       '2020-07-12', '2020-07-19', '2020-07-26', '2020-08-02', '2020-08-09',
       '2020-08-16', '2020-08-23', '2020-08-30', '2020-09-06', '2020-09-13',
       '2020-09-20', '2020-09-27', '2020-10-04', '2020-10-11', '2020-10-18',
       '2020-10-25', '2020-11-01', '2020-11-08', '2020-11-15', '2020-11-22',
       '2020-11-29', '2020-12-06', '2020-12-13', '2020-12-20', '2020-12-27',
       '2021-01-03', '2021-01-10', '2021-01-17', '2021-01-24', '2021-01-31',
       '2021-02-07', 'County Name', 'State', 'population'],
      dtype='object')
```

```
pop_df = pd.read_csv(PATH+"/covid_county_population_usafacts.csv")
case_df = pd.read_csv(PATH+"/covid_confirmed_usafacts.csv")
merged_data = data_clean_merge(case_df,pop_df)
merged_data.loc[:,'2020-01-26':'2021-02-07'].idxmax(axis=1)
```

```
0       2021-01-10
1       2020-12-20
2       2021-01-10
3       2020-12-20
4       2020-12-20
           ...
3137    2020-11-29
3138    2021-01-17
3139    2020-11-22
3140    2020-12-06
3141    2020-11-08
Length: 3142, dtype: object
```

Scale hides Other States Pattern

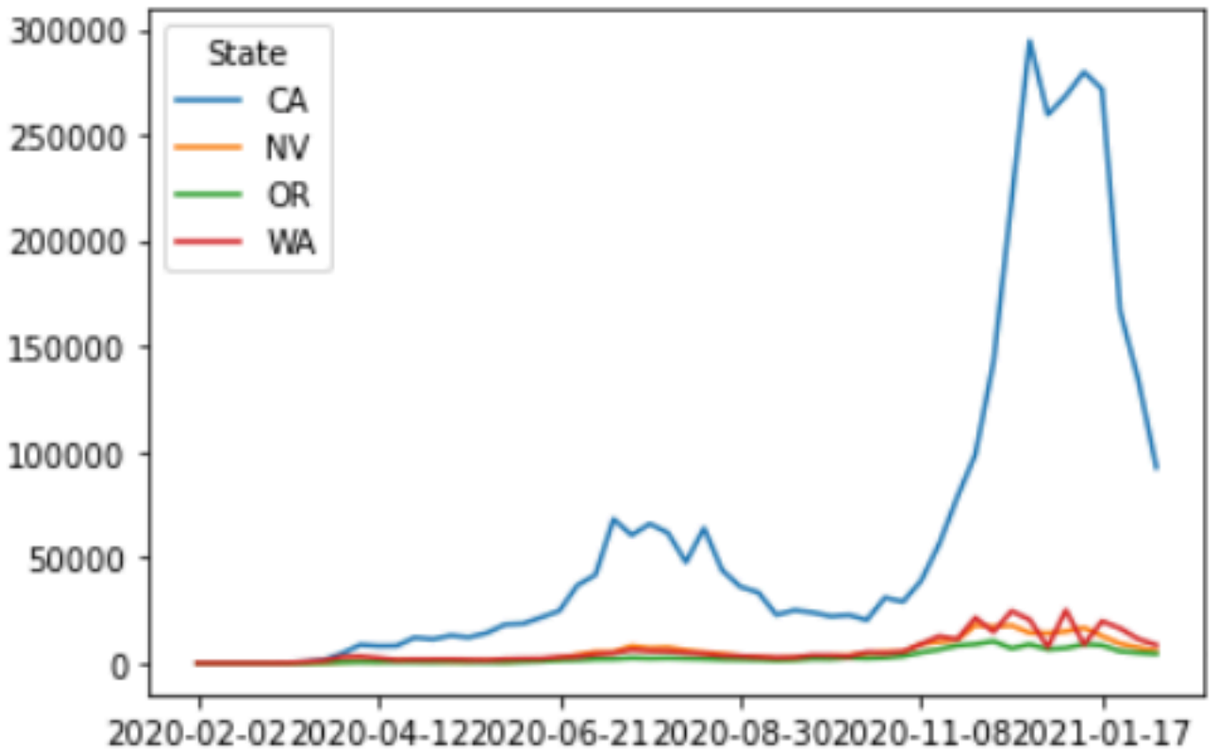


Fig.1-4 Weekly Nevada New COVID-19 Cases

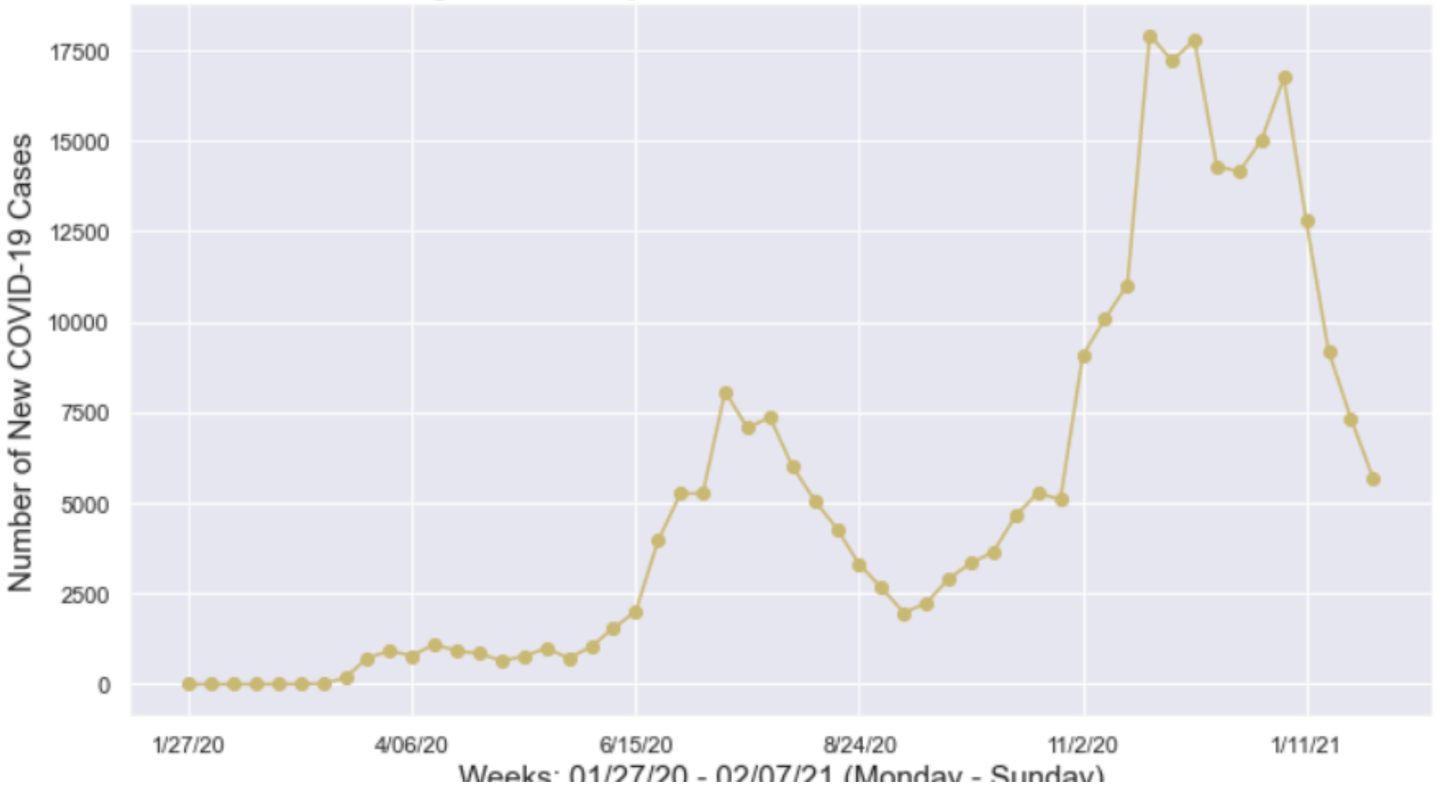
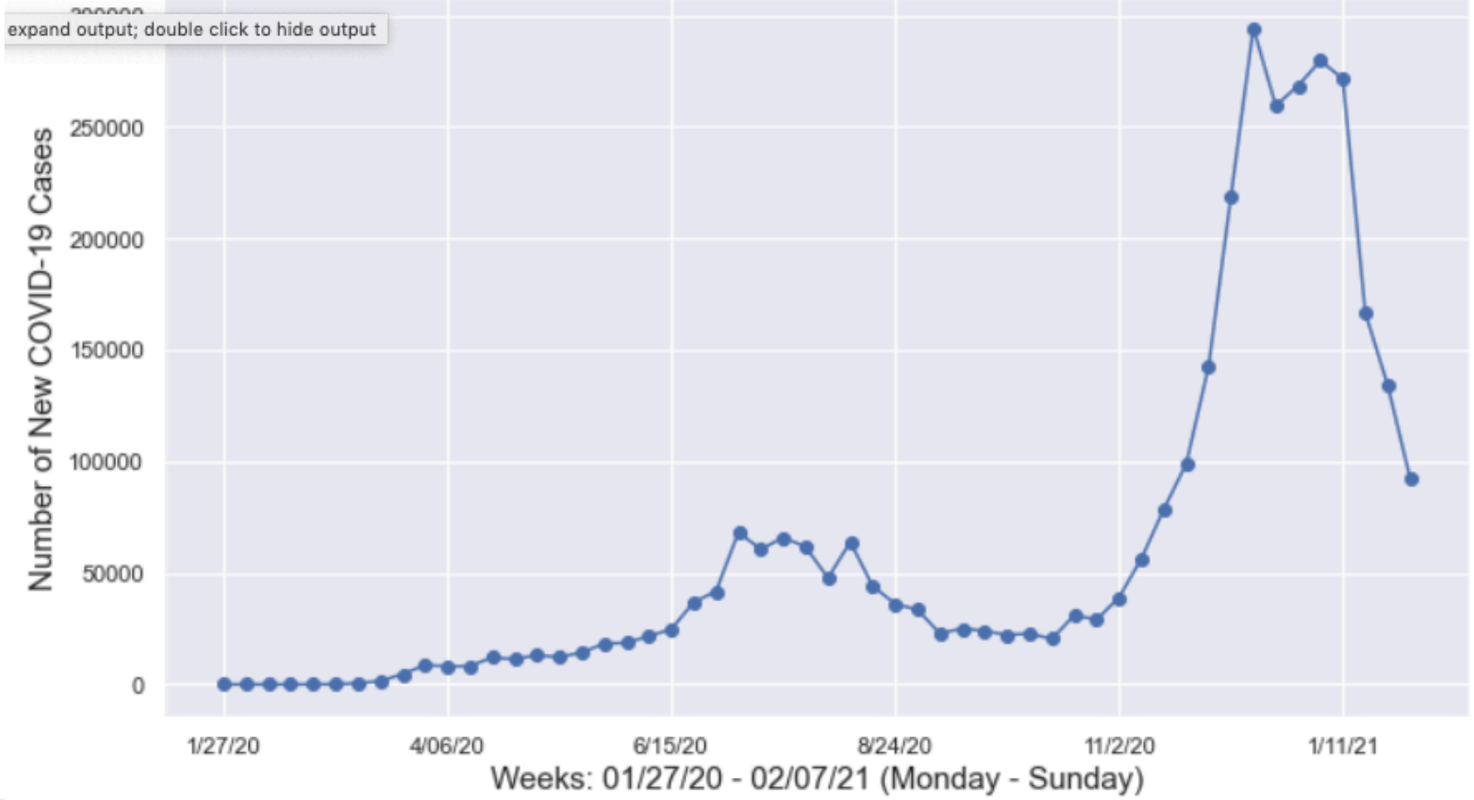
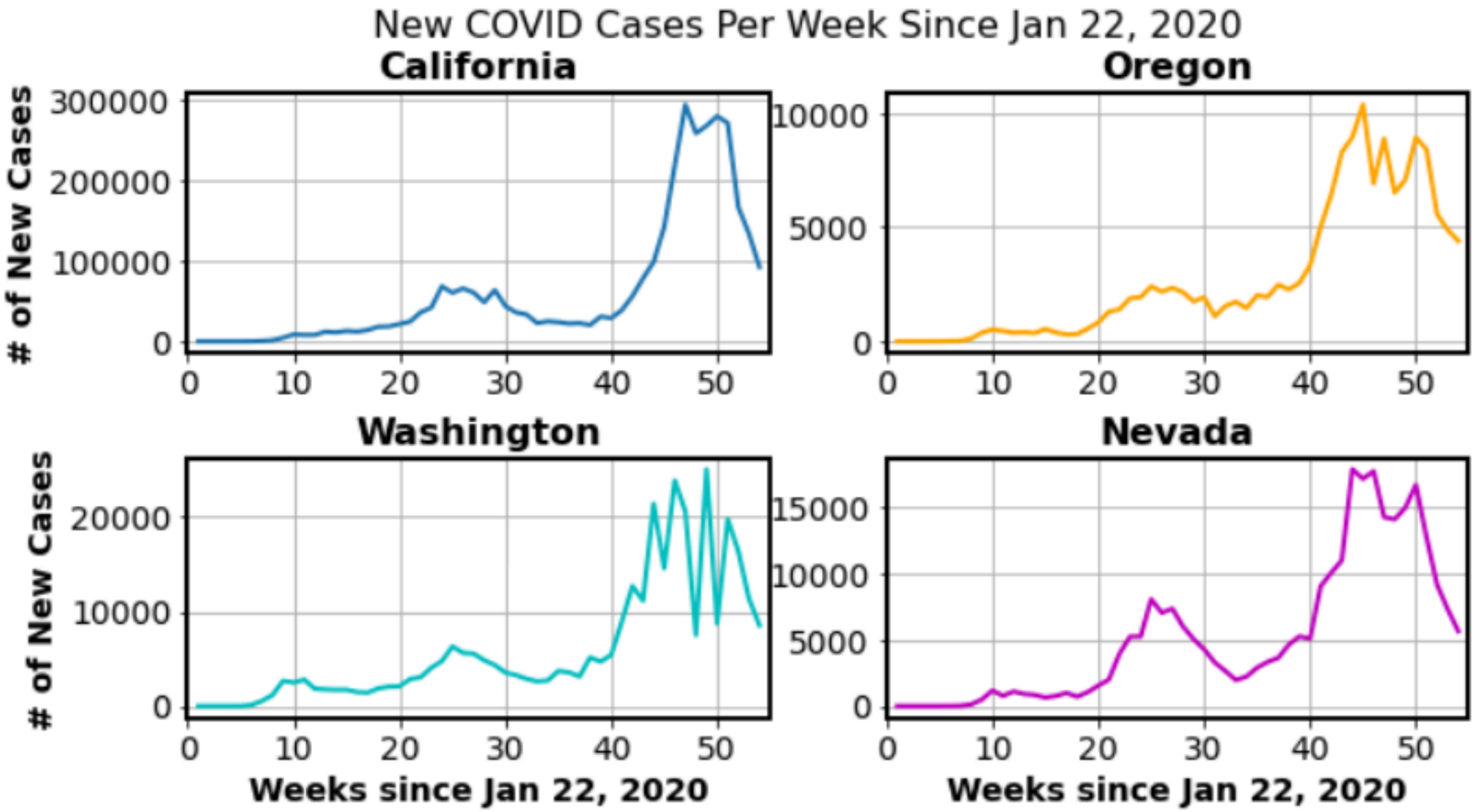
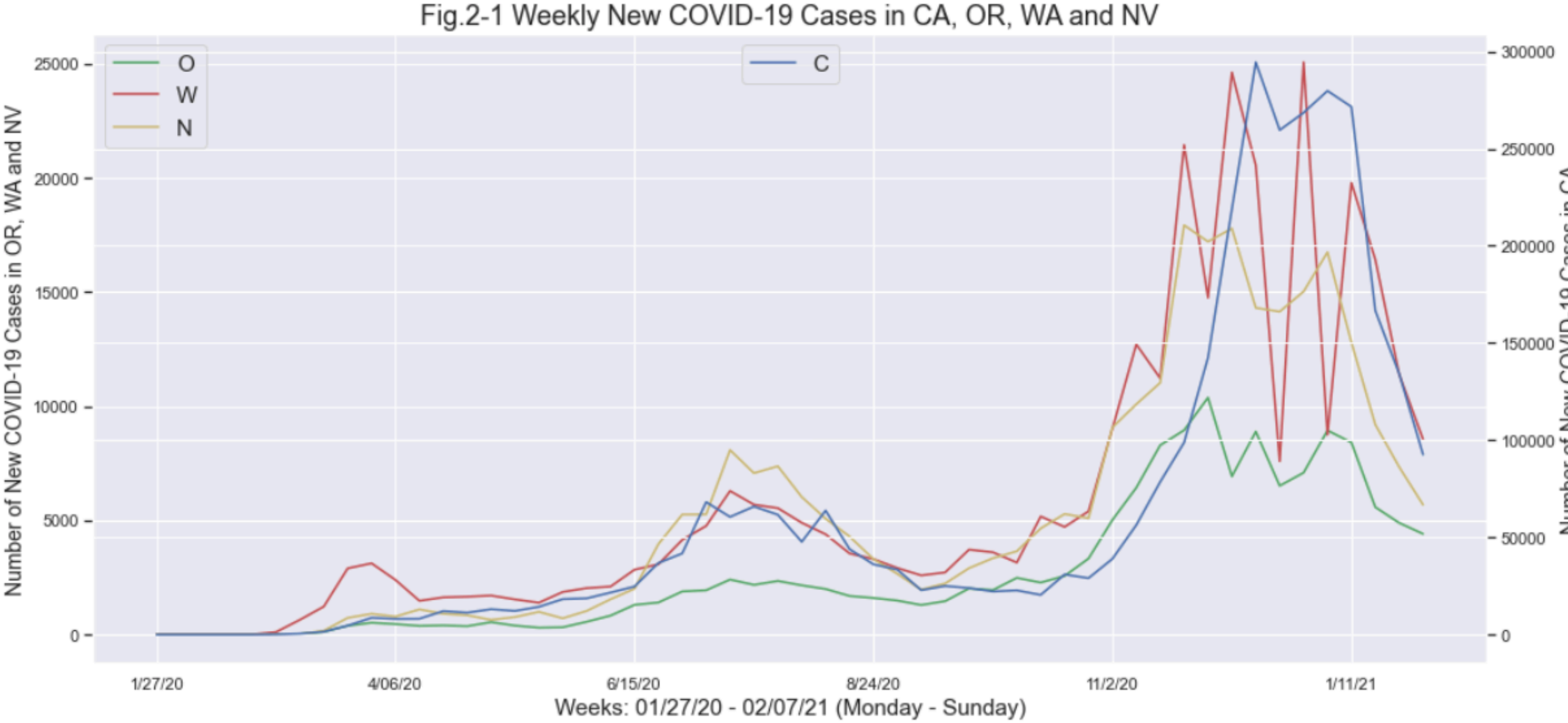


Fig.1-1 Weekly California New COVID-19 Cases

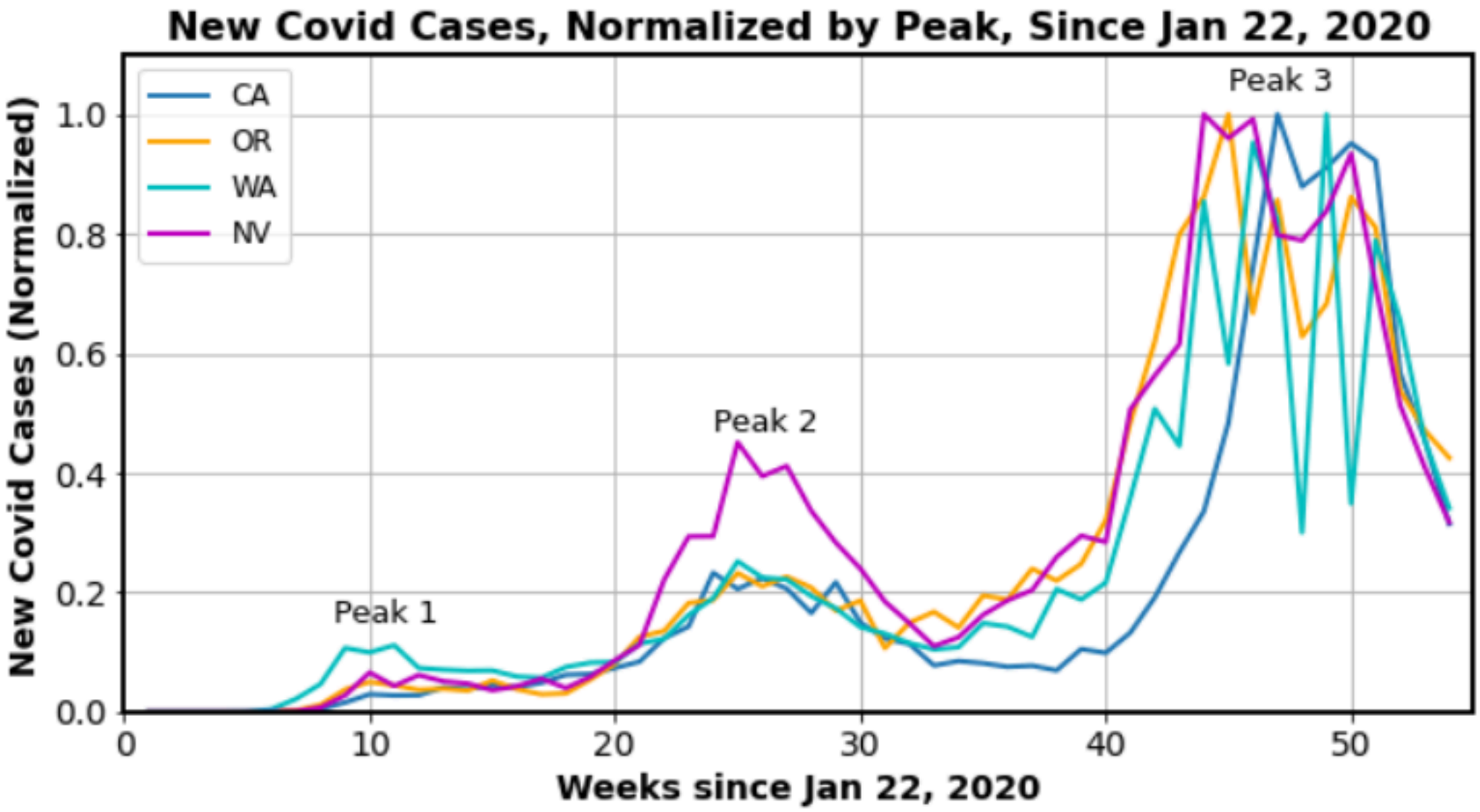



New COVID Cases Per Week Since Jan 22, 2020
California
Oregon
Washington
Nevada
# of New Cases
Weeks since Jan 22, 2020
0
10
20
30
40
50
100000
200000
300000
5000
10000
15000
20000

Using multiple Scales

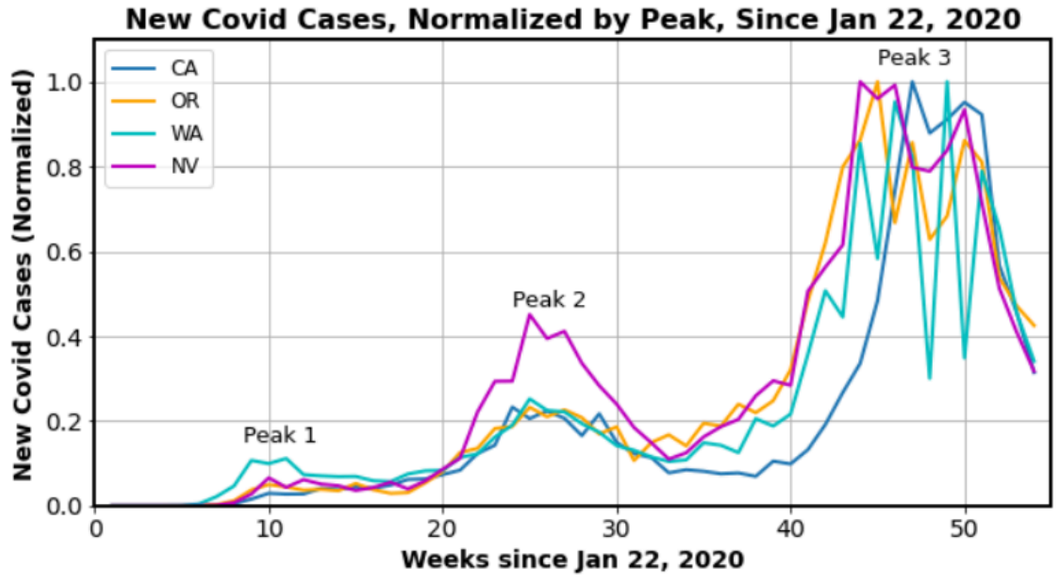


Normalized

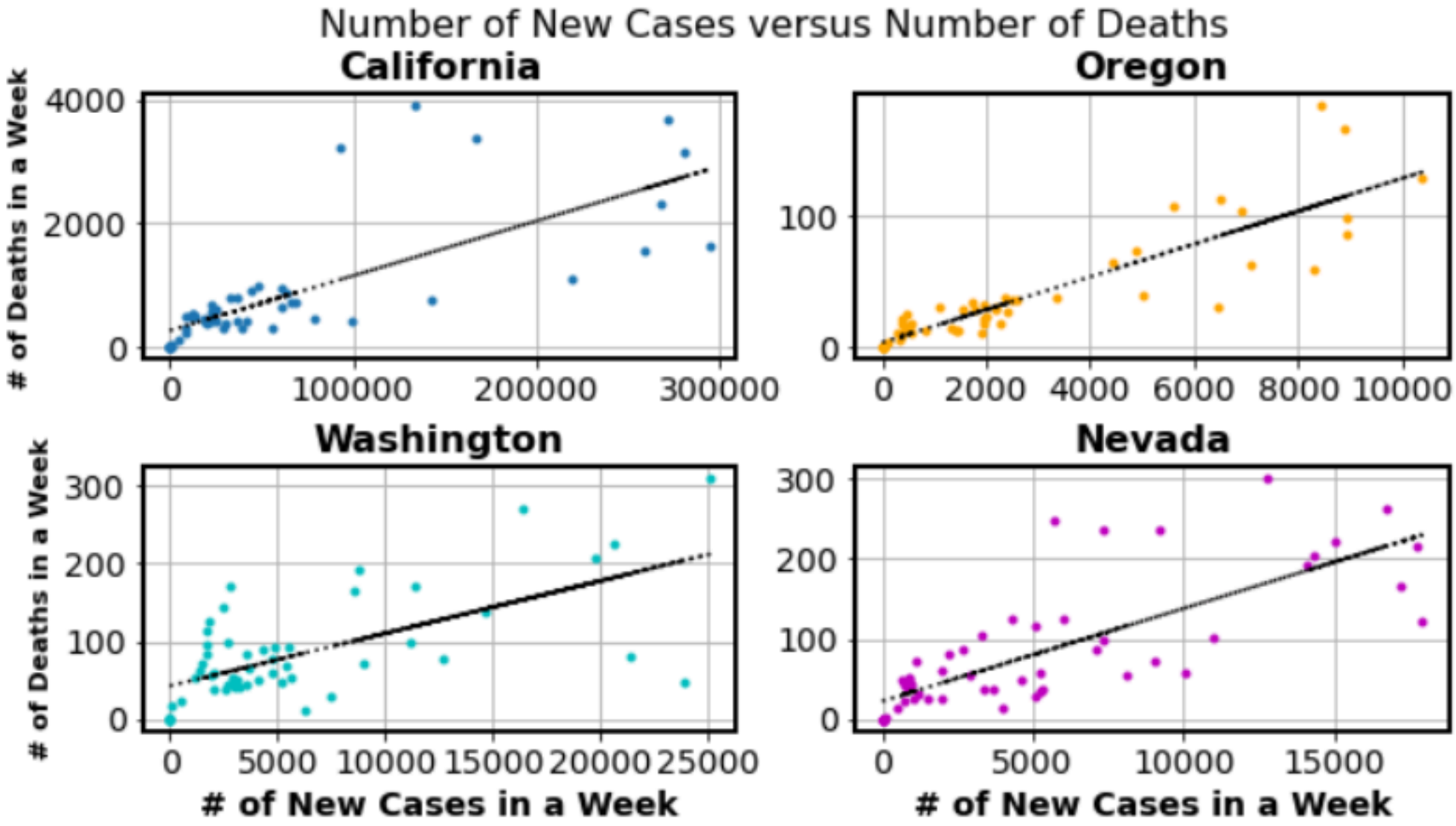

New Covid Cases, Normalized by Peak, Since Jan 22, 2020
CA
OR
WA
NV
Peak 1
Peak 2
Peak 3
New Covid Cases (Normalized)
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
50
Weeks since Jan 22, 2020

The Code

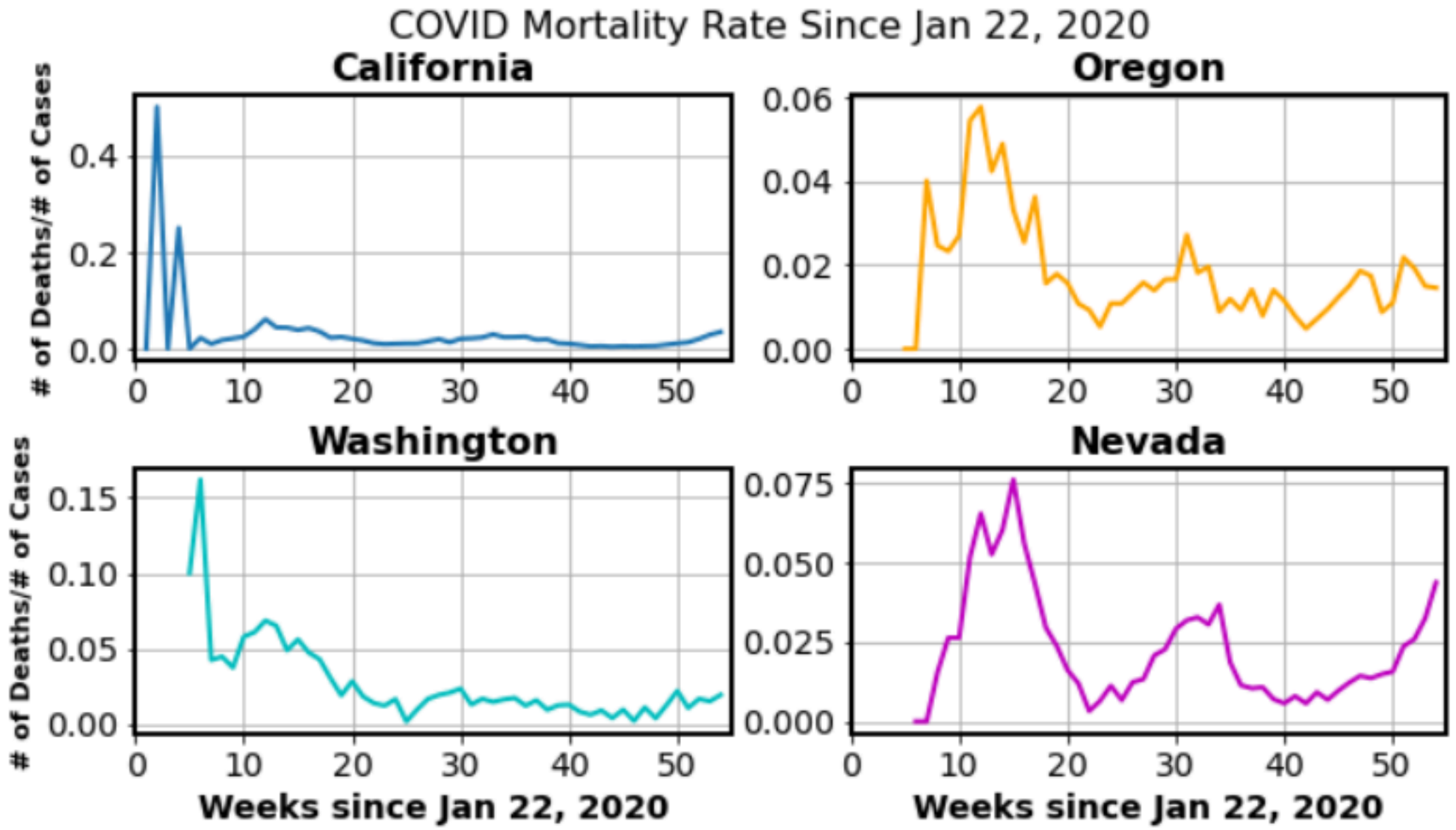
```
weeks = np.arange(0, 56, 1)

plt.title('New Covid Cases, Normalized by Peak, Since Jan 22, 2020')
plt.plot(weeks[1:-1], weekly_CA_cases[1:-1]/weekly_CA_cases[1:-1].max(), lw = 2, label = 'CA')
plt.plot(weeks[1:-1], weekly_OR_cases[1:-1]/weekly_OR_cases[1:-1].max(), lw = 2, c = 'orange', label = 'OR')
plt.plot(weeks[1:-1], weekly_WA_cases[1:-1]/weekly_WA_cases[1:-1].max(), lw = 2, c = 'c', label = 'WA')
plt.plot(weeks[1:-1], weekly_NV_cases[1:-1]/weekly_NV_cases[1:-1].max(), lw = 2, c = 'm', label = 'NV')
plt.text(8.5, 0.15, 'Peak 1', fontsize=13)
plt.text(24, 0.47, 'Peak 2', fontsize=13)
plt.text(45, 1.04, 'Peak 3', fontsize=13)
plt.ylabel('New Covid Cases (Normalized)')
plt.xlabel('Weeks since Jan 22, 2020')
plt.xlim([0, 55])
plt.ylim([0, 1.1])
plt.legend(fontsize=12)
plt.grid()
```



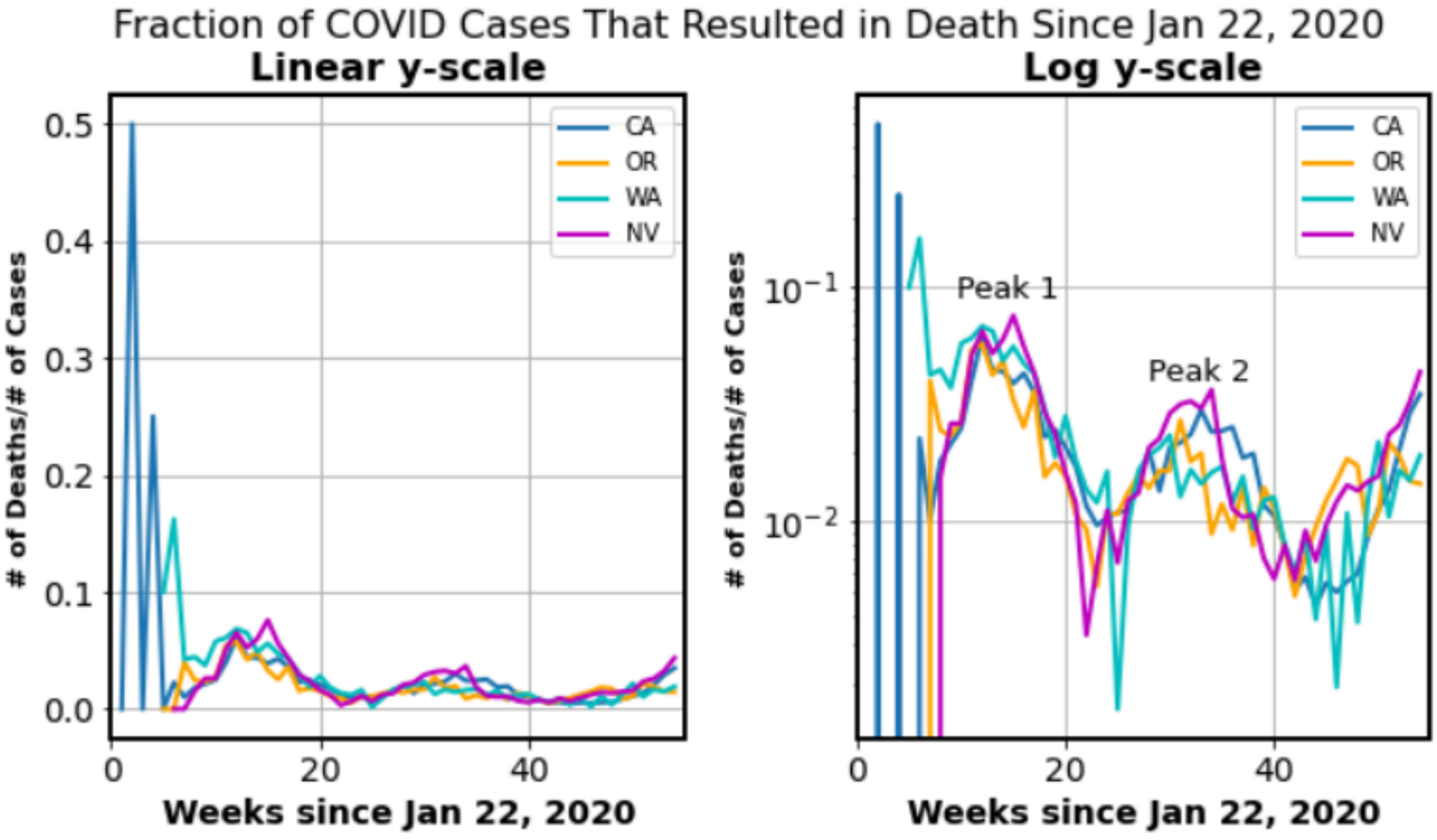
Cases vs Deaths



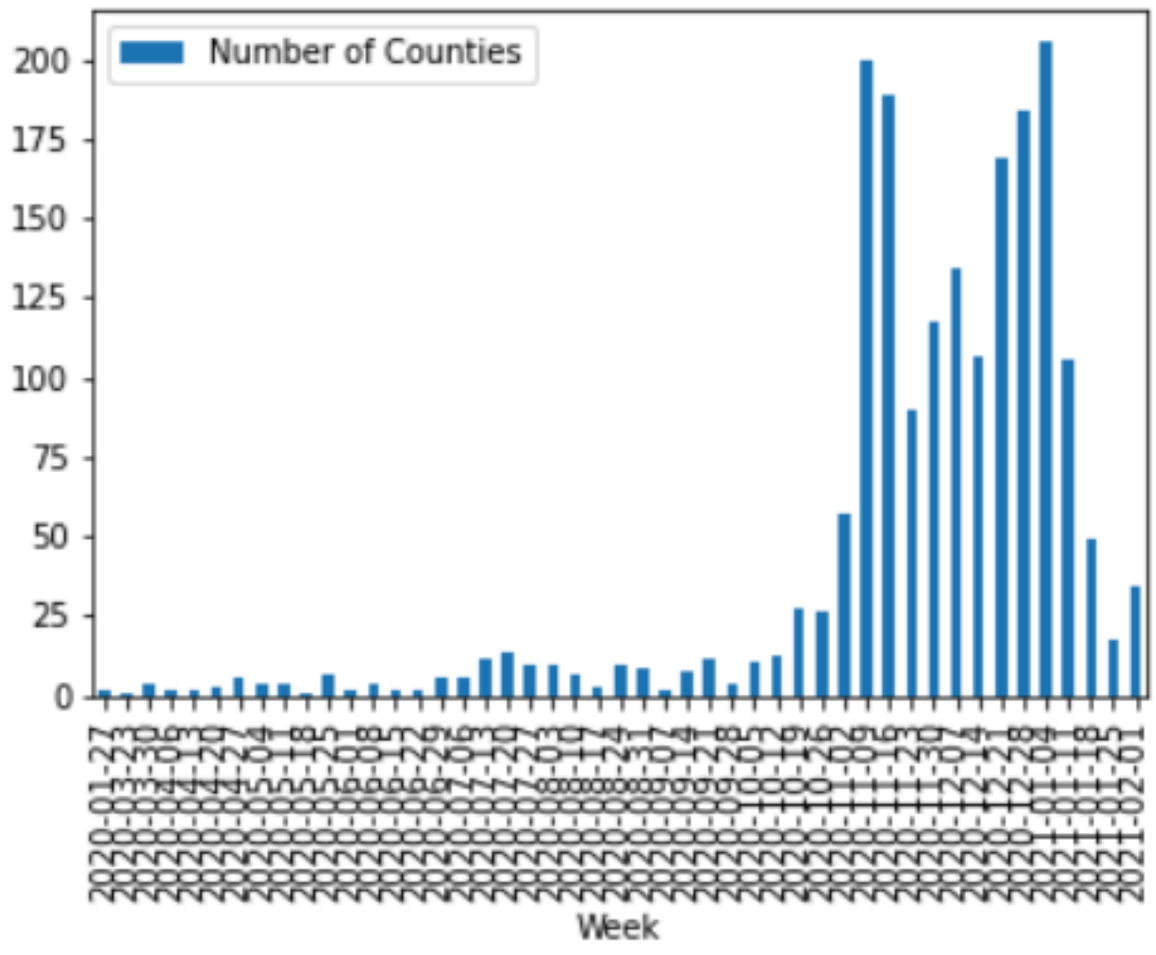
Mortality Rates



Using Log Scale



Week labels are Not Readable



How can there be more deaths than People?

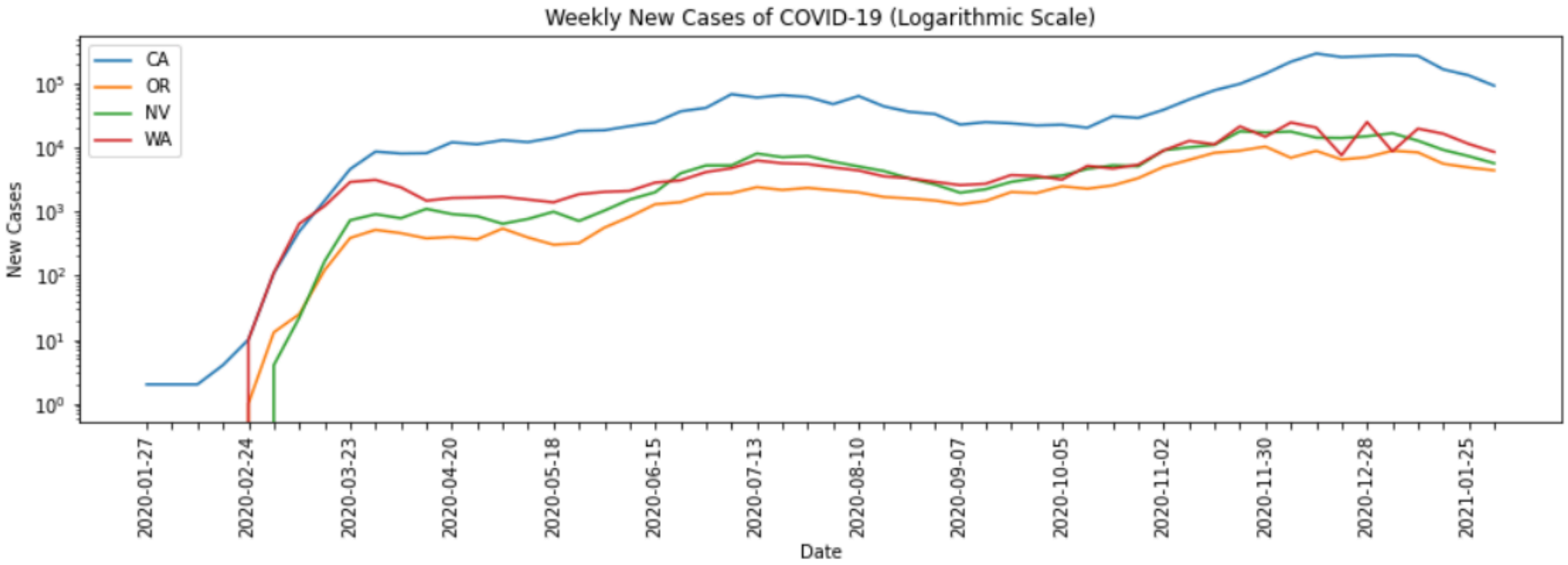
	countyFIPS	Total Cases	Total Deaths	population	percent
2031	38083	101	1511	1315	107.224335
261	8033	63	2250	2055	106.423358
1711	31115	40	507	664	70.331325
2398	46075	82	593	903	56.589147

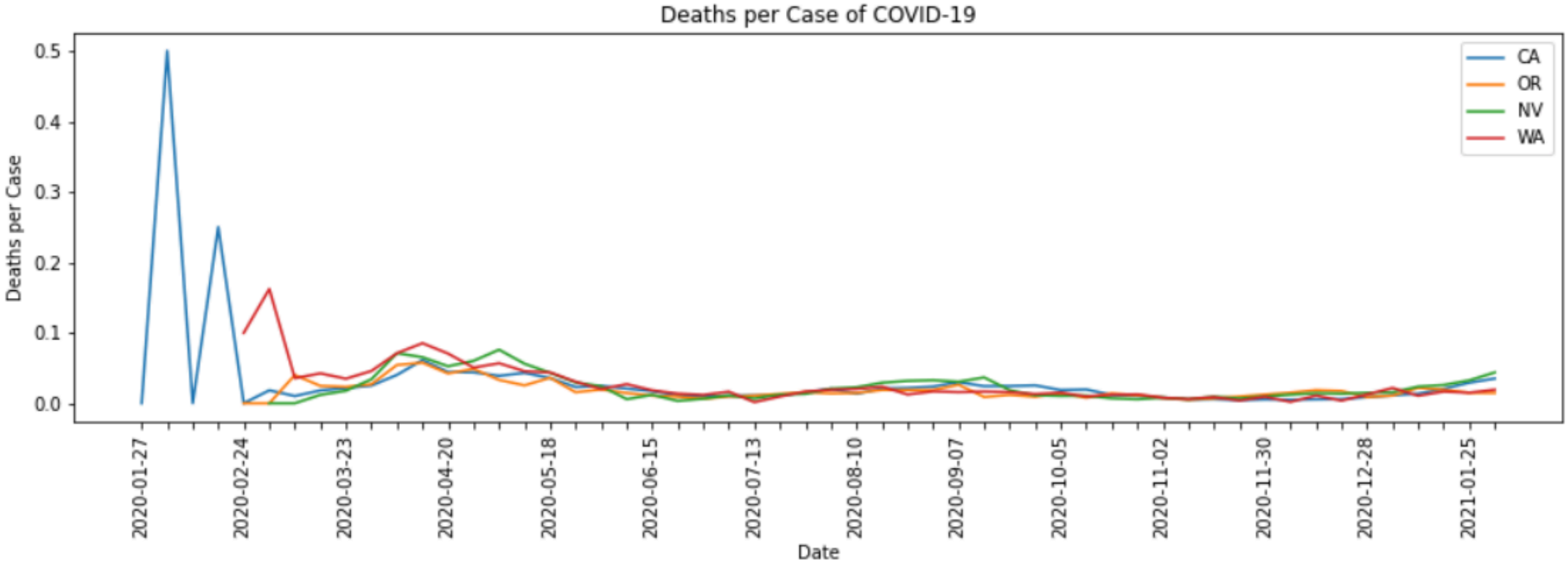
countyFIPS

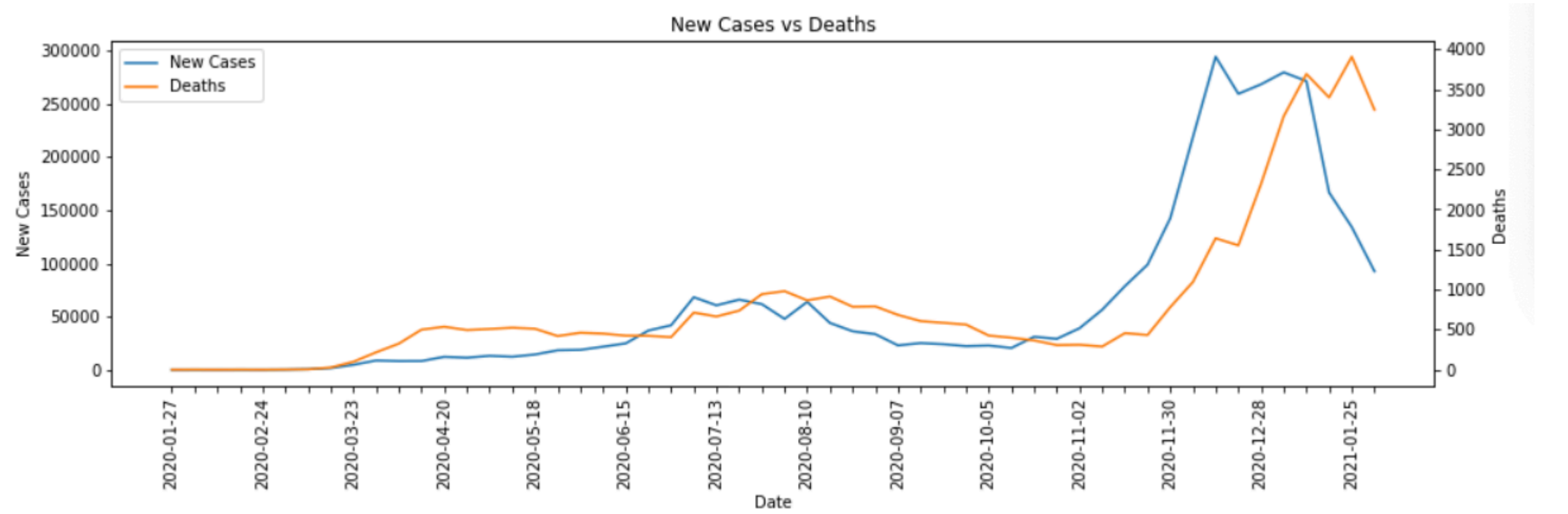
```
8025     31.172482
8011     23.915782
5079     23.802188
47095    23.712815
46041    23.675694
13053    22.798973
20137    22.553431
46009    21.815009
46017    21.549513
47169    20.834443
19021    20.547386
38027    19.991232
40003    19.782342
20053    19.350607
42053    18.705036
5067     18.637263
2050     18.514889
31043    18.353448
12067    18.335516
5077     18.327100
dtype: float64
```

What counties?

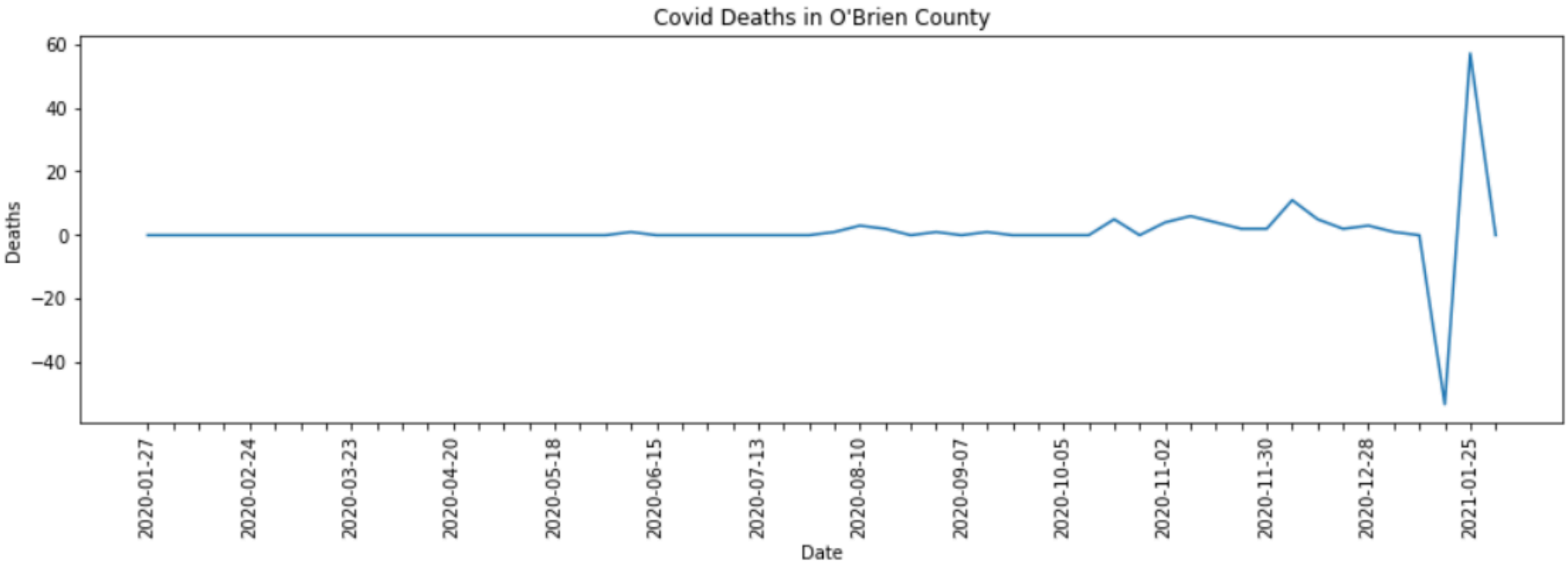
Using Log scale

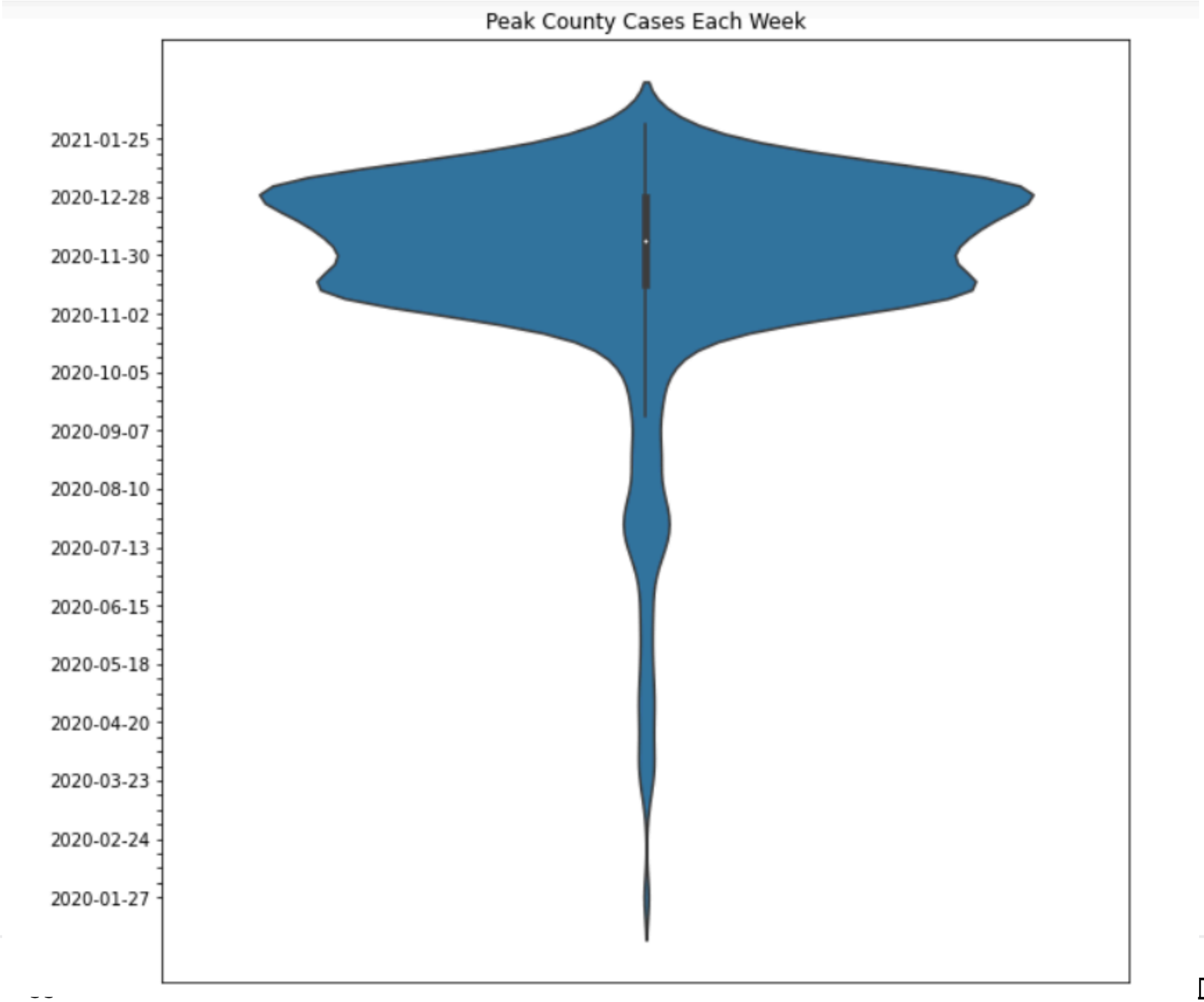


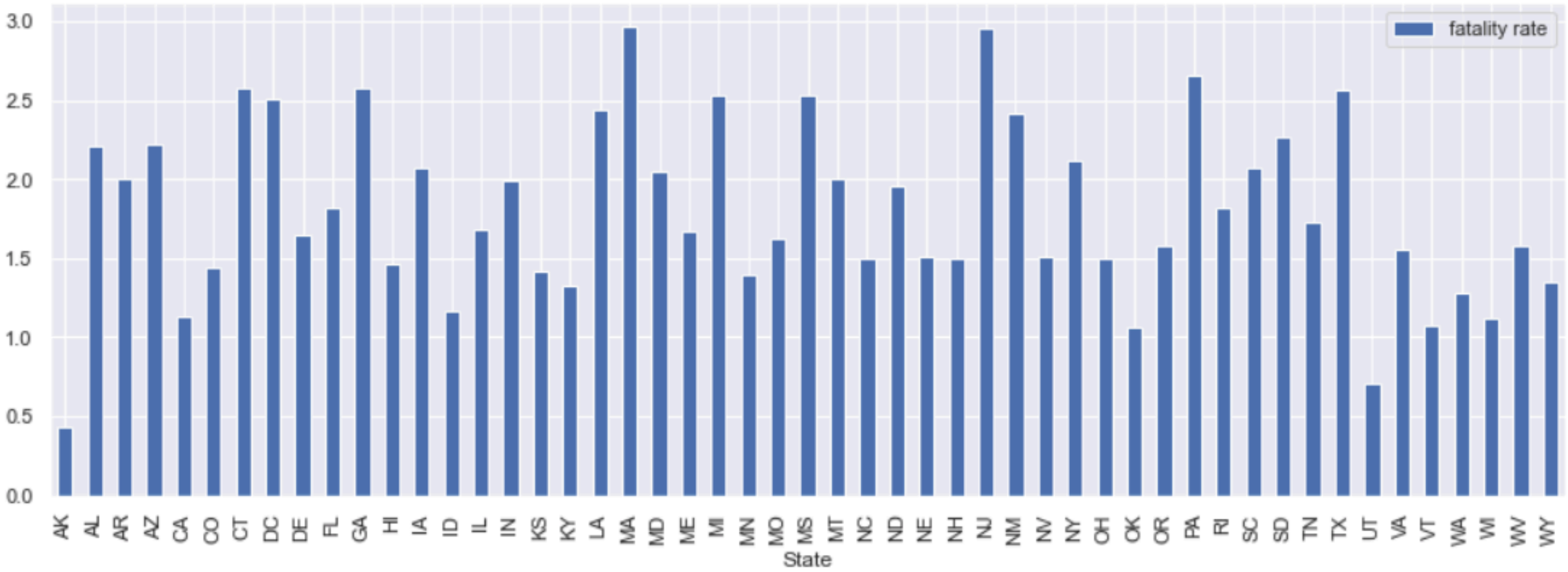

Deaths per Case of COVID-19
Deaths per Case
0.0
0.1
0.2
0.3
0.4
0.5
CA
OR
NV
WA
2020-01-27
2020-02-24
2020-03-23
2020-04-20
2020-05-18
2020-06-15
2020-07-13
2020-08-10
2020-09-07
2020-10-05
2020-11-02
2020-11-30
2020-12-28
2021-01-25
Date


New Cases vs Deaths
New Cases
Deaths
New Cases
0
50000
100000
150000
200000
250000
300000
Deaths
0
500
1000
1500
2000
2500
3000
3500
4000
2020-01-27
2020-02-24
2020-03-23
2020-04-20
2020-05-18
2020-06-15
2020-07-13
2020-08-10
2020-09-07
2020-10-05
2020-11-02
2020-11-30
2020-12-28
2021-01-25
Date

What does this do to death rate?




Peak County Cases Each Week
2021-01-25
2020-12-28
2020-11-30
2020-11-02
2020-10-05
2020-09-07
2020-08-10
2020-07-13
2020-06-15
2020-05-18
2020-04-20
2020-03-23
2020-02-24
2020-01-27

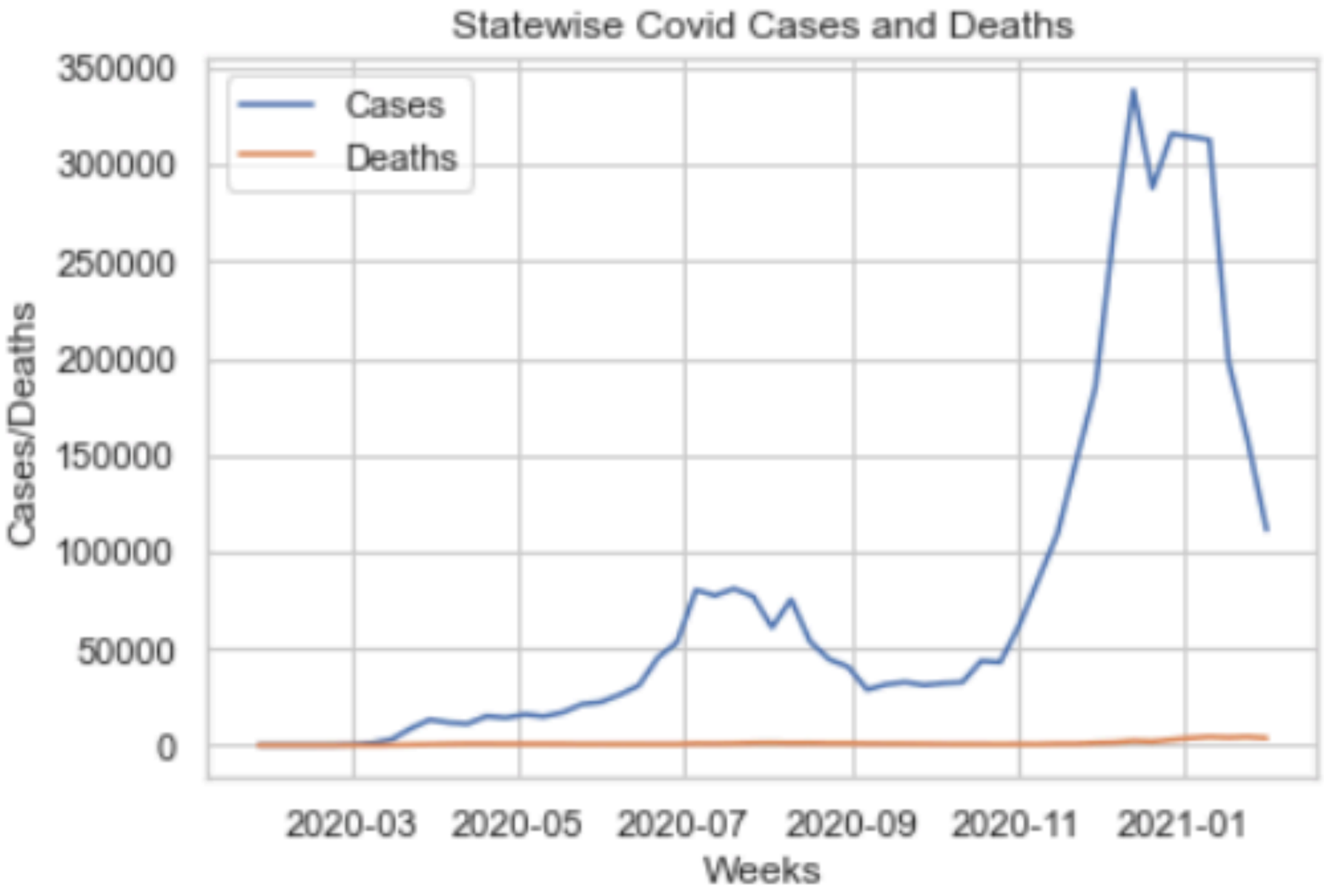

fatality rate
3.0
2.5
2.0
1.5
1.0
0.5
0.0
AK AL AR AZ CA CO CT DC DE FL GA HI IA ID IL IN KS KY LA MA MD ME MI MN MO MS MT NC ND NE NH NJ NM NV NY OH OK OR PA RI SC SD TN TX UT VA VT WA WI WV WY
State

```
# get the weekly count for all the states (CA, OR, WA, NV)
weekly_new_cases_count_df = weekly_new_cases_count_df.swapaxes(0,1)
# find Correlation among the states count
new_cases_corr_matrix = weekly_new_cases_count_df.corr()
print(new_cases_corr_matrix)
```

Table 1

	CA_Count	OR_Count	WA_Count	NV_Count
CA_Count	1.000000	0.833836	0.862291	0.869333
OR_Count	0.833836	1.000000	0.900384	0.940084
WA_Count	0.862291	0.900384	1.000000	0.896007
NV_Count	0.869333	0.940084	0.896007	1.000000

Scale Hides What Is Going On



Q9 - What is the Result

new_df.iloc[:,-1].nlargest(20)

```
396     31.172482
1335    23.915782
336     23.802188
1975    23.712815
3076    23.675694
1452    22.798973
815     22.553431
2447    21.815009
3064    21.549513
1120    20.834443
753     20.547386
1815    19.991232
2901    19.782342
726     19.350607
826     18.705036
330     18.637263
7       18.514889
2757    18.353448
1658    18.335516
```

This understandable

Table 9-3. Top 20 rates of the population that is still alive and has had covid per county in the US

State_x	County Name_x	%Alive
CO	Crowley County	30.869493
CO	Bent County	23.471400
AR	Lincoln County	23.402948
TN	Lake County	23.289624
SD	Dewey County	23.234895
GA	Chattahoochee County	22.765197
KS	Norton County	21.936206
SD	Bon Homme County	21.373714
SD	Buffalo County	20.744139
TN	Trousdale County	20.630982
IA	Buena Vista County	20.326198
OK	Alfalfa County	19.677306
ND	Eddy County	19.676432
KS	Ellsworth County	19.272370
AR	Jackson County	18.410192
AK	Bethel Census Area	18.405308
PA	Forest County	18.393818
NE	Dakota County	18.016578
FL	Lafayette County	18.012349

Bonus - Collection Pipelines

Lays out a sequence of operations that pass a collection of items between them.

"one of the most common, and pleasing, patterns in software"
Martin Fowler

```
def group_and_locate(casedeath):
    casedeath = casedeath[casedeath['countyFIPS']!=0]
    casedeaths = casedeath.drop(["State", "StateFIPS"], axis=1)
    casedeaths = casedeaths.groupby('countyFIPS').sum().loc[:,'2020-01-26'::7]
    return casedeaths
```

Applying multiple operations to a dataframe in a sequence

Java Streams

```
String[] words = {"a", "ab", "abc", "abcd", "bat"};
List<String> wordList = Arrays.asList(words);
List<String> longWords
longWords = wordList.stream()
                        .filter( s -> s.length() > 2)
                        .filter( s -> s.charAt(0) == 'a')
                        .map( s -> s.toUpperCase())
                        .collect( Collectors.toList());
System.out.println(longWords);
```

Julia @as macro

```
weekly_new_cases(coviddf) = @as data coviddf begin
    remove_nondata_cols(data)
    day_totals(data)
    DataFrames.stack(data,1:ncol(data))
    remove_sum_string(data)
    @where(data, issunday.(:variable))
    total_to_new_cases(data)
    shorten_date(data)
end
```

```
remove_nondata_cols(coviddf)

day_totals(remove_nondata_cols(coviddf))

DataFrames.stack(day_totals(remove_nondata_cols(coviddf)),1:ncol(data))
```

Julia @as macro

```
weekly_new_cases(coviddf) = @as data coviddf begin
    remove_nondata_cols(data)
    day_totals(data)
    DataFrames.stack(data,1:ncol(data))
    remove_sum_string(data)
    @where(data, issunday.(:variable))
    total_to_new_cases(data)
    shorten_date(data)
end


function weekly_new_cases(coviddf)
    just_data = remove_nondata_cols(coviddf)
    daily_totals = day_totals(just_data)
    stacked  = DataFrames.stack(data,1:ncol(daily_totals))
    labels_fixed = remove_sum_string(stacked)
    etc....
```

Python Toolz

https://toolz.readthedocs.io/en/latest/streaming-analytics.html

Supports some collection pipelines in Python