Data 200 Final Project - Covid-19

Friendly reminders:

- Some parts of this code utilize parallel computing based on the `Joblib` and may become aggressive in CPU and RAM usage
- This notebook imports the `missingno` library to construct visualizations of missing values, one may have to install it beforehand

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from datetime import datetime, timedelta, date

from sklearn import linear_model
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_validate
from sklearn.feature_selection import SelectFromModel
from sklearn.kernel_ridge import KernelRidge
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

np.random.seed(42)
%matplotlib inline
```

In [2]:
```
# Please make sure that this library is already installed
import missingno as msno
```

1. Load files and EDA

In [3]:
```
# Reading csv files

df1 = pd.read_csv("abridged_couties.csv")
df1['countyFIPS'] = pd.to_numeric(df1['countyFIPS'], errors='coerce')
df2 = pd.read_csv("time_series_covid19_confirmed_US.csv")
df3 = pd.read_csv("time_series_covid19_deaths_US.csv")
```

Overview

- All the key independent variables (features) are included in the counties dataset
- All the dependent variables are included in the time series datasets
- So the primary objective of the EDA is to merge the counties dataset with the time series datasets, and prepare for a dataset that is ready to use

1.1 Examine counties data

In [4]:
```
print('Shape of the counties dataset:', df1.shape)
df1.head()
```

```
Shape of the counties dataset: (3244, 87)
```

Out[4]:

	countyFIPS	STATEFP	COUNTYFP	CountyName	StateName	State	lat	lon	POP_LA
0	1001.0	1.0	1.0	Autauga	AL	Alabama	32.540091	-86.645649	32
1	1003.0	1.0	3.0	Baldwin	AL	Alabama	30.738314	-87.726272	30
2	1005.0	1.0	5.0	Barbour	AL	Alabama	31.874030	-85.397327	31
3	1007.0	1.0	7.0	Bibb	AL	Alabama	32.999024	-87.125260	33
4	1009.0	1.0	9.0	Blount	AL	Alabama	33.990440	-86.562711	33

5 rows × 87 columns

In [5]:
```python
df1.keys()
```

Out[5]:
```
Index(['countyFIPS', 'STATEFP', 'COUNTYFP', 'CountyName', 'StateName', 'State',
       'lat', 'lon', 'POP_LATITUDE', 'POP_LONGITUDE', 'CensusRegionName',
       'CensusDivisionName', 'Rural-UrbanContinuumCode2013',
       'PopulationEstimate2018', 'PopTotalMale2017', 'PopTotalFemale2017',
       'FracMale2017', 'PopulationEstimate65+2017',
       'PopulationDensityperSqMile2010', 'CensusPopulation2010',
       'MedianAge2010', '#EligibleforMedicare2018',
       'MedicareEnrollment,AgedTot2017', '3-YrDiabetes2015-17',
       'DiabetesPercentage', 'HeartDiseaseMortality', 'StrokeMortality',
       'Smokers_Percentage', 'RespMortalityRate2014', '#FTEHospitalTotal2017',
       'TotalM.D.'s,TotNon-FedandFed2017', '#HospParticipatinginNetwork2017',
       '#Hospitals', '#ICU_beds', 'dem_to_rep_ratio', 'PopMale<52010',
       'PopFmle<52010', 'PopMale5-92010', 'PopFmle5-92010', 'PopMale10-142010',
       'PopFmle10-142010', 'PopMale15-192010', 'PopFmle15-192010',
       'PopMale20-242010', 'PopFmle20-242010', 'PopMale25-292010',
       'PopFmle25-292010', 'PopMale30-342010', 'PopFmle30-342010',
       'PopMale35-442010', 'PopFmle35-442010', 'PopMale45-542010',
       'PopFmle45-542010', 'PopMale55-592010', 'PopFmle55-592010',
       'PopMale60-642010', 'PopFmle60-642010', 'PopMale65-742010',
       'PopFmle65-742010', 'PopMale75-842010', 'PopFmle75-842010',
       'PopMale>842010', 'PopFmle>842010', '3-YrMortalityAge<1Year2015-17',
       '3-YrMortalityAge1-4Years2015-17', '3-YrMortalityAge5-14Years2015-17',
       '3-YrMortalityAge15-24Years2015-17',
       '3-YrMortalityAge25-34Years2015-17',
       '3-YrMortalityAge35-44Years2015-17',
       '3-YrMortalityAge45-54Years2015-17',
       '3-YrMortalityAge55-64Years2015-17',
       '3-YrMortalityAge65-74Years2015-17',
       '3-YrMortalityAge75-84Years2015-17', '3-YrMortalityAge85+Years2015-17',
       'mortality2015-17Estimated', 'stay at home', '>50 gatherings',
       '>500 gatherings', 'public schools', 'restaurant dine-in',
       'entertainment/gym', 'federal guidelines', 'foreign travel ban',
       'SVIPercentile', 'HPSAShortage', 'HPSAServedPop', 'HPSAUnderservedPop'],
      dtype='object')
```

1.1.1 Find the primary key

In [6]:
```python
df1['countyFIPS'].dropna().size
```

Out[6]:
```
3242
```

In [7]:
```python
# Examine observations with duplicated countyFIPS
df1.loc[df1['countyFIPS'].duplicated(keep=False)]
```

Out[7]:

	countyFIPS	STATEFP	COUNTYFP	CountyName	StateName	State	lat	lon	POP_LATITUDE	F
3143	60020.0	NaN	NaN	Manua	AS	NaN	NaN	NaN	NaN	
3144	60020.0	NaN	NaN	Ofu	AS	NaN	NaN	NaN	NaN	
3145	60020.0	NaN	NaN	Olosega	AS	NaN	NaN	NaN	NaN	
3147	66010.0	NaN	NaN	Cocos Island	MP	NaN	NaN	NaN	NaN	
3148	66010.0	NaN	NaN	Guam	GU	NaN	NaN	NaN	NaN	
3151	69120.0	NaN	NaN	Aguijan	MP	NaN	NaN	NaN	NaN	
3152	69120.0	NaN	NaN	Tinian	MP	NaN	NaN	NaN	NaN	
3242	NaN	NaN	NaN	New York City	NY	NaN	NaN	NaN	NaN	
3243	NaN	NaN	NaN	Kansas City	MO	NaN	NaN	NaN	NaN	

9 rows × 87 columns

Summary of this part:

- The `countyFIPS` can be used as the primary key for this dataset.
- Although there are some duplicates, those observations are either to be outside of the US mainland (e.g. Guam), or to be unions of different counties (e.g. New York City).
    - Since this analysis is at the county level of all the states, these observations with duplicated `countyFIPS` will be dropped later.

1.1.2 Handle missing values

In [8]:

```
# Examine variables with more than 50 missing values

df1_missing = pd.DataFrame(df1.isna().sum())
df1_missing[df1_missing[0]>50]
```

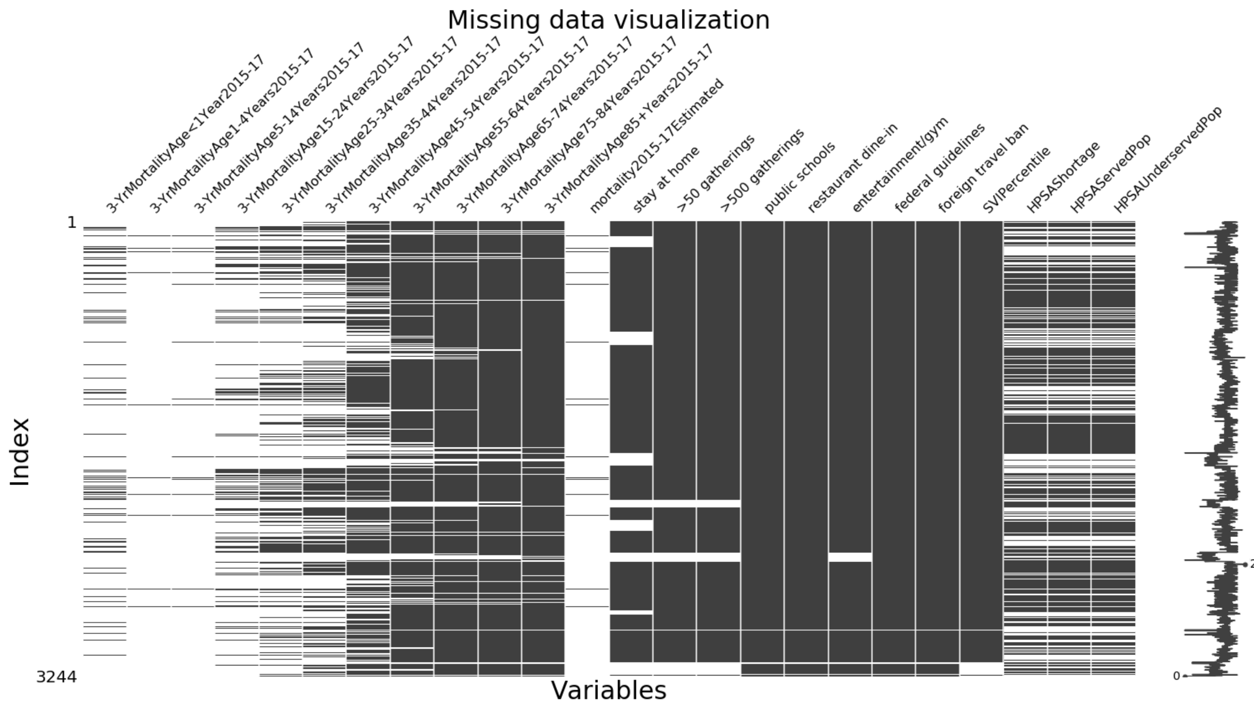
Out[8]:

	0
State	169
lat	169
lon	169
CensusRegionName	98
CensusDivisionName	98
3-YrDiabetes2015-17	1744
Smokers_Percentage	103
RespMortalityRate2014	103
#Hospitals	103
#ICU_beds	103

	0
dem_to_rep_ratio	129
3-YrMortalityAge<1Year2015-17	2772
3-YrMortalityAge1-4Years2015-17	3179
3-YrMortalityAge5-14Years2015-17	3149
3-YrMortalityAge15-24Years2015-17	2610
3-YrMortalityAge25-34Years2015-17	2270
3-YrMortalityAge35-44Years2015-17	1916
3-YrMortalityAge45-54Years2015-17	1071
3-YrMortalityAge55-64Years2015-17	492
3-YrMortalityAge65-74Years2015-17	330
3-YrMortalityAge75-84Years2015-17	237
3-YrMortalityAge85+Years2015-17	181
mortality2015-17Estimated	3149
stay at home	592
>50 gatherings	221
>500 gatherings	221
entertainment/gym	90
SVIPercentile	104
HPSAShortage	1141
HPSAServedPop	1141
HPSAUnderservedPop	1141

It seems that most missing values occur in the variables starting from `3-YrMortalityAge<1Year2015-17` . Below is a visualization of missing values.

```
In [9]:  msno.matrix(df1.iloc[:,63:])
         plt.xlabel('Variables', fontdict={'fontsize': 30})
         plt.ylabel('Index', fontdict={'fontsize': 30})
         plt.title('Missing data visualization', fontdict={'fontsize': 30});
```



Handle mortality missing values

```
In [10]: mortality = ['3-YrMortalityAge<1Year2015-17',
                '3-YrMortalityAge1-4Years2015-17', '3-YrMortalityAge5-14Years2015-17',
                '3-YrMortalityAge15-24Years2015-17',
                '3-YrMortalityAge25-34Years2015-17',
                '3-YrMortalityAge35-44Years2015-17',
                '3-YrMortalityAge45-54Years2015-17',
                '3-YrMortalityAge55-64Years2015-17',
                '3-YrMortalityAge65-74Years2015-17',
                '3-YrMortalityAge75-84Years2015-17', '3-YrMortalityAge85+Years2015-17']
```

Observation:

- The mortality values can be way greater than 100, implying that these are head counts
- For older age groups, there are less missing mortality values
- This suggests that the missing values are caused by the fact that no death in the age group is recorded, so I replace all the missing values with 0
- Moreover, head counts will be highly correlated with the population, so I replace them with ratio of the head counts to the population
- `mortality2015-17Estimated` has too many missing values, and seems to be highly correlated with other mortality values, so I decided to drop it

```
In [11]: # fill missing mortality values with 0
         df1[mortality] = df1[mortality].fillna(0)

         # Change head counts to ratios
         for i in mortality:
             df1[i] = df1[i]/df1['PopulationEstimate2018']
```

Handle other missing values

Observation:

- There are no clear structure for the rest of the main missing values, so I replace them with the average values at the state level.

In [12]:
```
# Replace missing values with average values at state level
missing = {}
state_average = {}
missing_index = {}
for i in ['3-YrDiabetes2015-17', 'HPSAShortage', 'HPSAServedPop', 'HPSAUnderservedPop']
    # Capture average values at the state level
    state_average[i] = df1.groupby('StateName').mean()[i]

    # Find the states of each missing value
    missing[i] = list(df1.loc[df1[i].isna(),'StateName'].unique())

    # Find the indices of each missing value
    missing_index[i] = df1.loc[df1[i].isna(),'StateName'].index

    # Fill missing values with state average
    df1.loc[missing_index[i], i] = state_average[i].loc[df1.loc[missing_index[i], 'Stat
```

Summary of this part

- I explored the structures of the missing values, and handled them appropriately
- For the rest of the missing values, I plan to drop those, because they are either not used (e.g. `lat` and `lon`) or minor comparing to the total observation number (e.g. `Smokers_Percentage`)

1.1.3 Preliminary feature engineering

Population breakdown

Observation:

- The population breakdown would be highly correlated with the total population, so these data are transformed to ratios

In [13]:
```
pop_breakdown = ['PopMale<52010',
       'PopFmle<52010', 'PopMale5-92010', 'PopFmle5-92010', 'PopMale10-142010',
       'PopFmle10-142010', 'PopMale15-192010', 'PopFmle15-192010',
       'PopMale20-242010', 'PopFmle20-242010', 'PopMale25-292010',
       'PopFmle25-292010', 'PopMale30-342010', 'PopFmle30-342010',
       'PopMale35-442010', 'PopFmle35-442010', 'PopMale45-542010',
       'PopFmle45-542010', 'PopMale55-592010', 'PopFmle55-592010',
       'PopMale60-642010', 'PopFmle60-642010', 'PopMale65-742010',
       'PopFmle65-742010', 'PopMale75-842010', 'PopFmle75-842010',
       'PopMale>842010', 'PopFmle>842010']
```

In [14]:
```
for i in pop_breakdown:
    df1[i] = df1[i]/df1['CensusPopulation2010']
```

In [15]:
```
df1['PopulationEstimate65+2017'] = df1['PopulationEstimate65+2017']/df1['PopulationEsti
```

Policy variables

Observation:

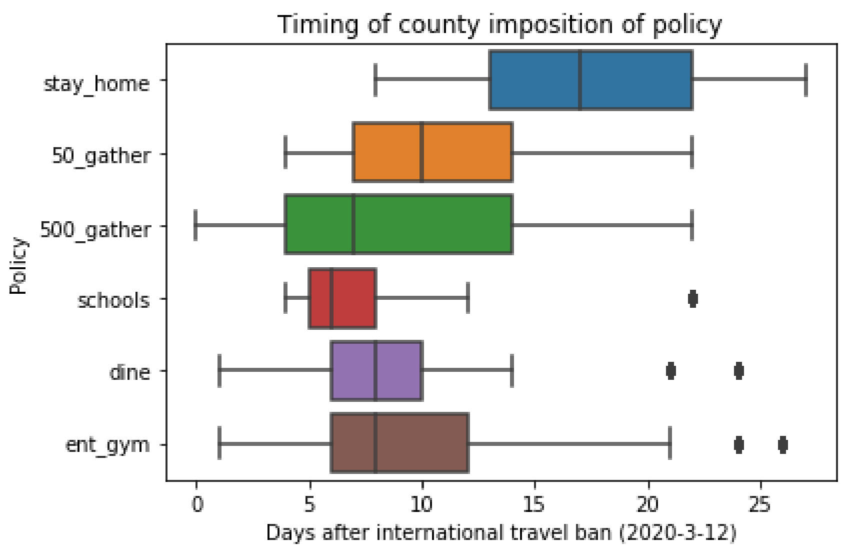
- The policy variables are in terms of the Proleptic Gregorian calendar, I will transform these to usual dates
- Before doing that, it would be interesting to see how states respond differently to the Covid-19
  - I provide a visualization of how many days after the federal guideline (3/12/2020) do states impletement those policies

In [16]:
```
policies = ['stay at home',
             '>50 gatherings',
             '>500 gatherings',
             'public schools',
             'restaurant dine-in',
             'entertainment/gym',
            ]
```

In [17]:
```
policy_days = ['stay_home',
                '50_gather',
                '500_gather',
                'schools',
                'dine',
                'ent_gym'
               ]
```

In [18]:
```
# federal imposition of international travel ban on 737495
for i,j in zip(policies, policy_days):
    df1[j] = df1[i] - 737495
```

In [19]:
```
sns.boxplot(data=df1[policy_days], orient='h')
plt.ylabel('Policy')
plt.xlabel('Days after international travel ban (2020-3-12)')
plt.title('Timing of county imposition of policy');
```



Observation:

- For the states who have the corresponding policies, there is some variation between the first implementation dates
- `stay_home` seems to be the strictest and therefore is the lastest to be adopted by states in general

- Because of the variation, it would be interesting to see whether these policies do have some effect in containing the Covid-19, which motivates the study

The cells below are to transform the Proleptic Gregorian calendar dates to usual dates

In [20]:
```
def to_date(days):
    '''
    Function
    --------
    to_date
        Transform Proleptic Gregorian calendar dates to usual dates

    Parameters
    ----------
    days :
        Proleptic Gregorian calendar dates

    Returns
    -------
    date:
        Usual date in datetime format
    '''

    try:
        date = datetime(1,1,1) + timedelta(days=days)
    except ValueError:
        date = np.nan
    return date
```

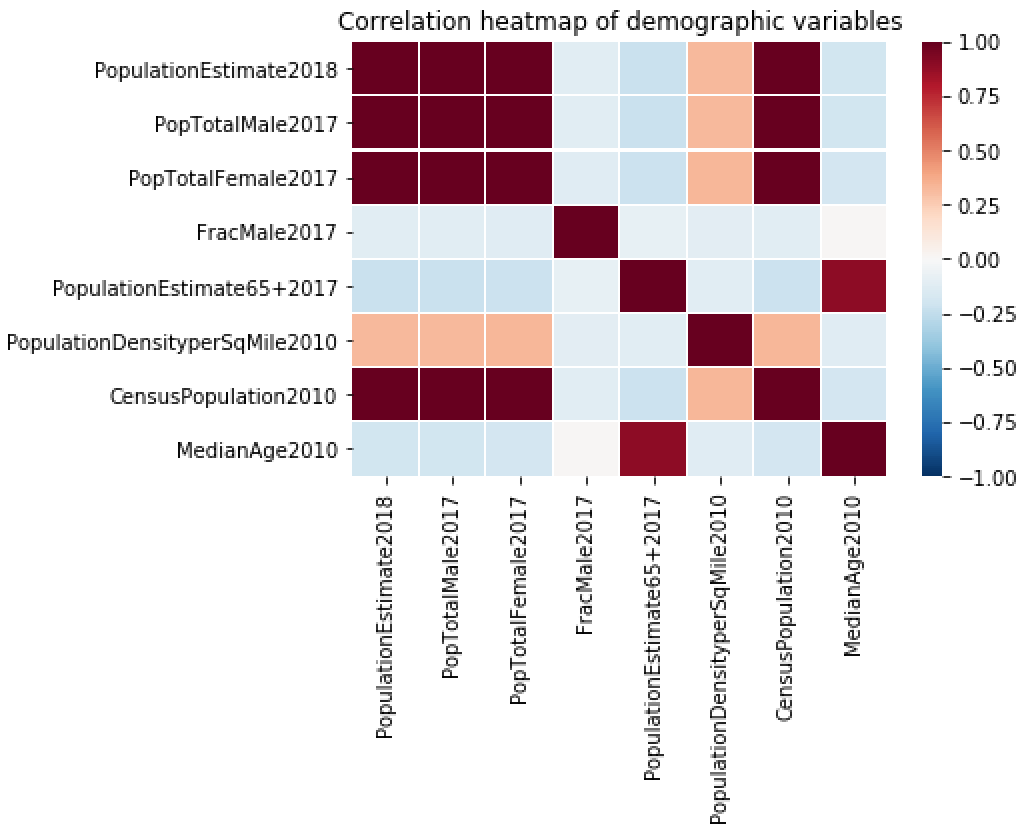
In [21]:
```
for policy in policies:
    df1[policy] = df1[policy].map(to_date)
```

Handle multicollinearity

The cells below examine multicollinearity issues by correlation visualization

In [22]:
```
demographics = [
    'PopulationEstimate2018', 'PopTotalMale2017', 'PopTotalFemale2017',
       'FracMale2017', 'PopulationEstimate65+2017',
       'PopulationDensityperSqMile2010', 'CensusPopulation2010',
       'MedianAge2010'
]

census_breakdown = [
    'PopMale<52010',
       'PopFmle<52010', 'PopMale5-92010', 'PopFmle5-92010', 'PopMale10-142010',
       'PopFmle10-142010', 'PopMale15-192010', 'PopFmle15-192010',
       'PopMale20-242010', 'PopFmle20-242010', 'PopMale25-292010',
       'PopFmle25-292010', 'PopMale30-342010', 'PopFmle30-342010',
       'PopMale35-442010', 'PopFmle35-442010', 'PopMale45-542010',
       'PopFmle45-542010', 'PopMale55-592010', 'PopFmle55-592010',
       'PopMale60-642010', 'PopFmle60-642010', 'PopMale65-742010',
       'PopFmle65-742010', 'PopMale75-842010', 'PopFmle75-842010',
       'PopMale>842010', 'PopFmle>842010'
]

health = [
    '#EligibleforMedicare2018',
       'MedicareEnrollment,AgedTot2017', '3-YrDiabetes2015-17',
       'DiabetesPercentage', 'HeartDiseaseMortality', 'StrokeMortality',
       'Smokers_Percentage', 'RespMortalityRate2014', '#FTEHospitalTotal2017',
       'TotalM.D.\'s,TotNon-FedandFed2017', '#HospParticipatinginNetwork2017',
       '#Hospitals', '#ICU_beds'
]

mortality = [
    '3-YrMortalityAge<1Year2015-17',
       '3-YrMortalityAge1-4Years2015-17', '3-YrMortalityAge5-14Years2015-17',
       '3-YrMortalityAge15-24Years2015-17',
       '3-YrMortalityAge25-34Years2015-17',
       '3-YrMortalityAge35-44Years2015-17',
       '3-YrMortalityAge45-54Years2015-17',
       '3-YrMortalityAge55-64Years2015-17',
       '3-YrMortalityAge65-74Years2015-17',
       '3-YrMortalityAge75-84Years2015-17', '3-YrMortalityAge85+Years2015-17',
       'mortality2015-17Estimated'
]
```

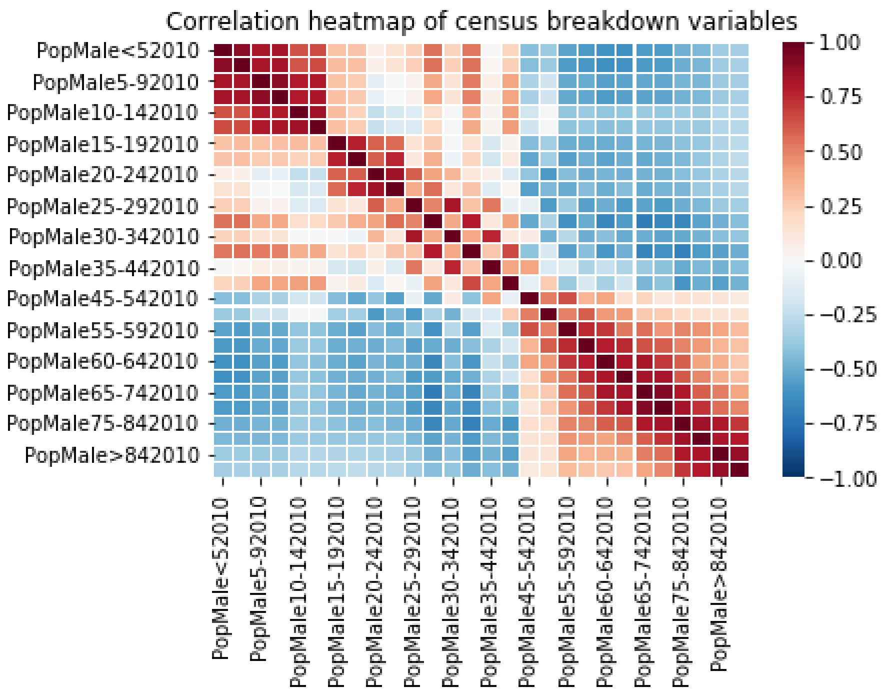
In [23]:
```
corr_demo = df1[demographics].corr()
sns.heatmap(corr_demo, vmin=-1, vmax=1, cmap="RdBu_r", lw=.1)
plt.title('Correlation heatmap of demographic variables');
```



Observation:

- As expected `PopTotalMale2017` , `PopTotalFemale2017` , `CensusPopulation2010` are highly correlated with `PopulationEstimate2018` -
- So `PopTotalMale2017` , `PopTotalFemale2017` , `CensusPopulation2010` will be removed

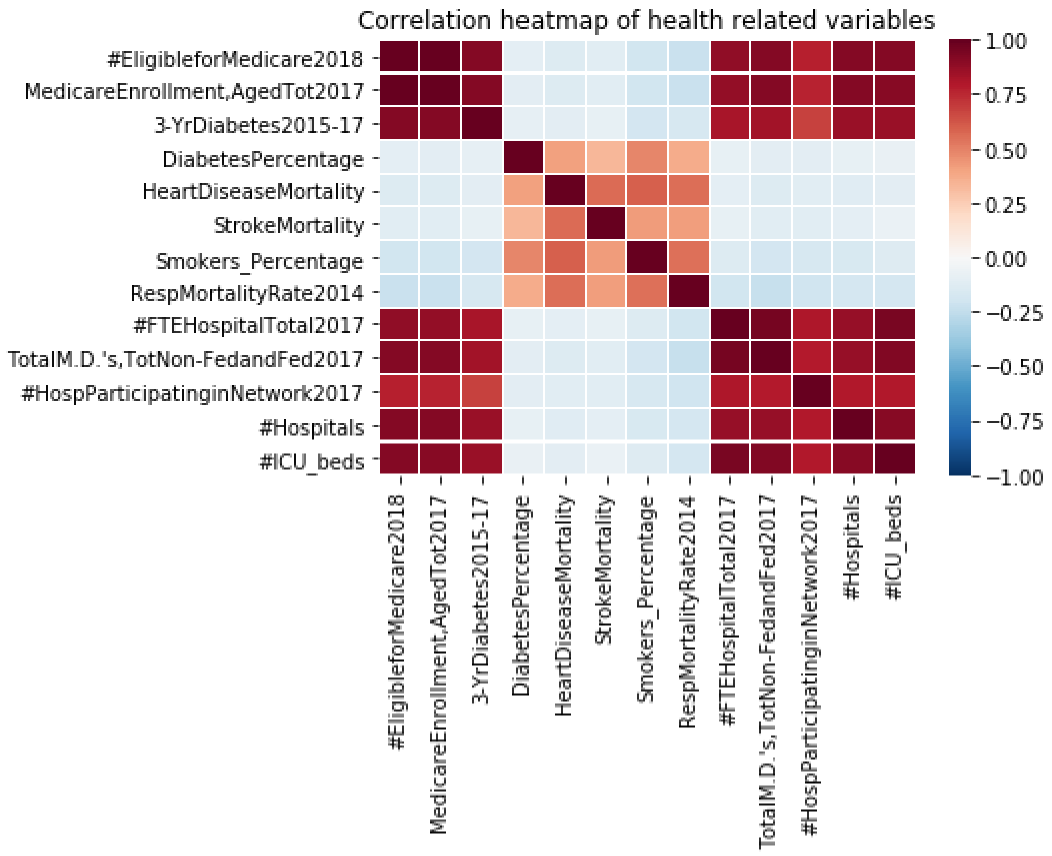
In [24]:
```
corr_census_breakdown = df1[census_breakdown].corr()
sns.heatmap(corr_census_breakdown, vmin=-1, vmax=1, cmap="RdBu_r", lw=.05)
plt.title('Correlation heatmap of census breakdown variables');
```



Observation:

- These variables are either positively or negatively correlated, but not perfectly
- I think these varriables seem to be fine, so I decide to keep all of them

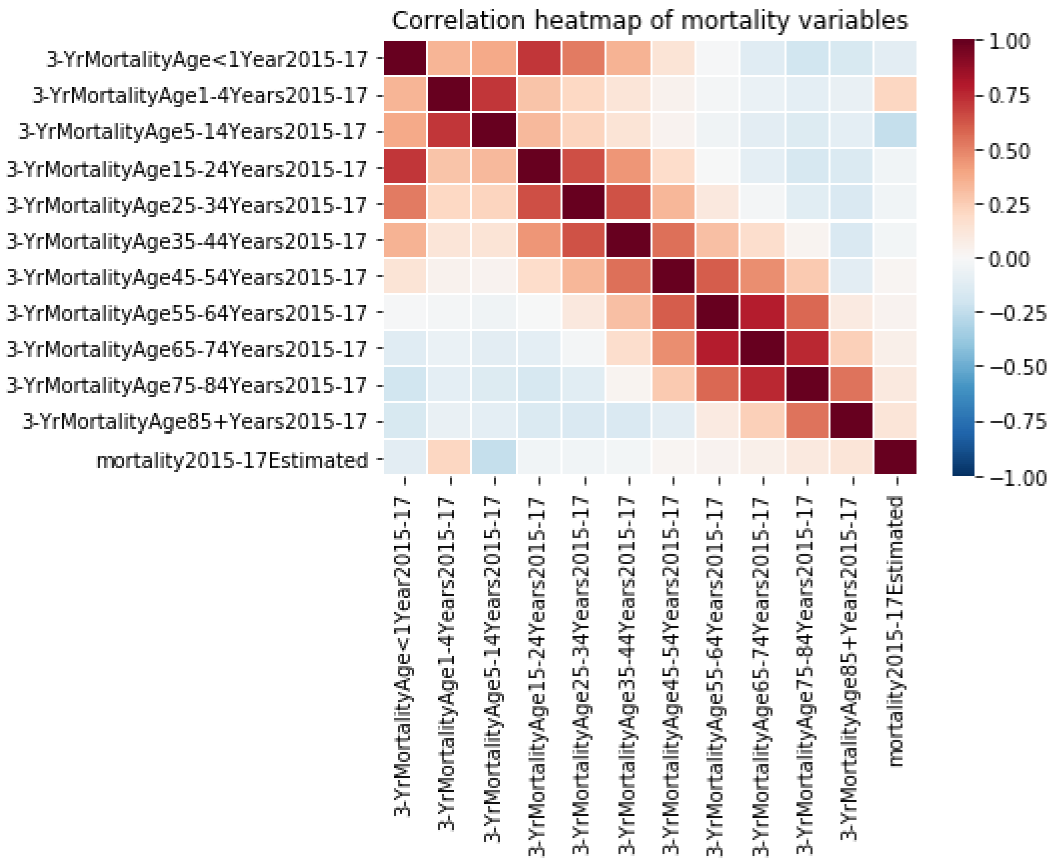
In [25]:
```
corr_health = df1[health].corr()
sns.heatmap(corr_health, vmin=-1, vmax=1, cmap="RdBu_r", lw=.1)
plt.title('Correlation heatmap of health related variables');
```



Observation:

- Many variables are almost perfectly positively correlated
- I decide to keep `MedicareEnrollment,AgedTot2017` , `3-YrDiabetes2015-17` , `DiabetesPercentage` , `HeartDiseaseMortality` , `StrokeMortality` , `Smokers_Percentage` , `RespMortalityRate2014` , and `#HospParticipatinginNetwork2017`

In [26]:
```
corr_mortality = df1[mortality].corr()
# plt.figure(figsize=(10,10))
sns.heatmap(corr_mortality, vmin=-1, vmax=1, cmap="RdBu_r", lw=.1)
plt.title('Correlation heatmap of mortality variables');
```

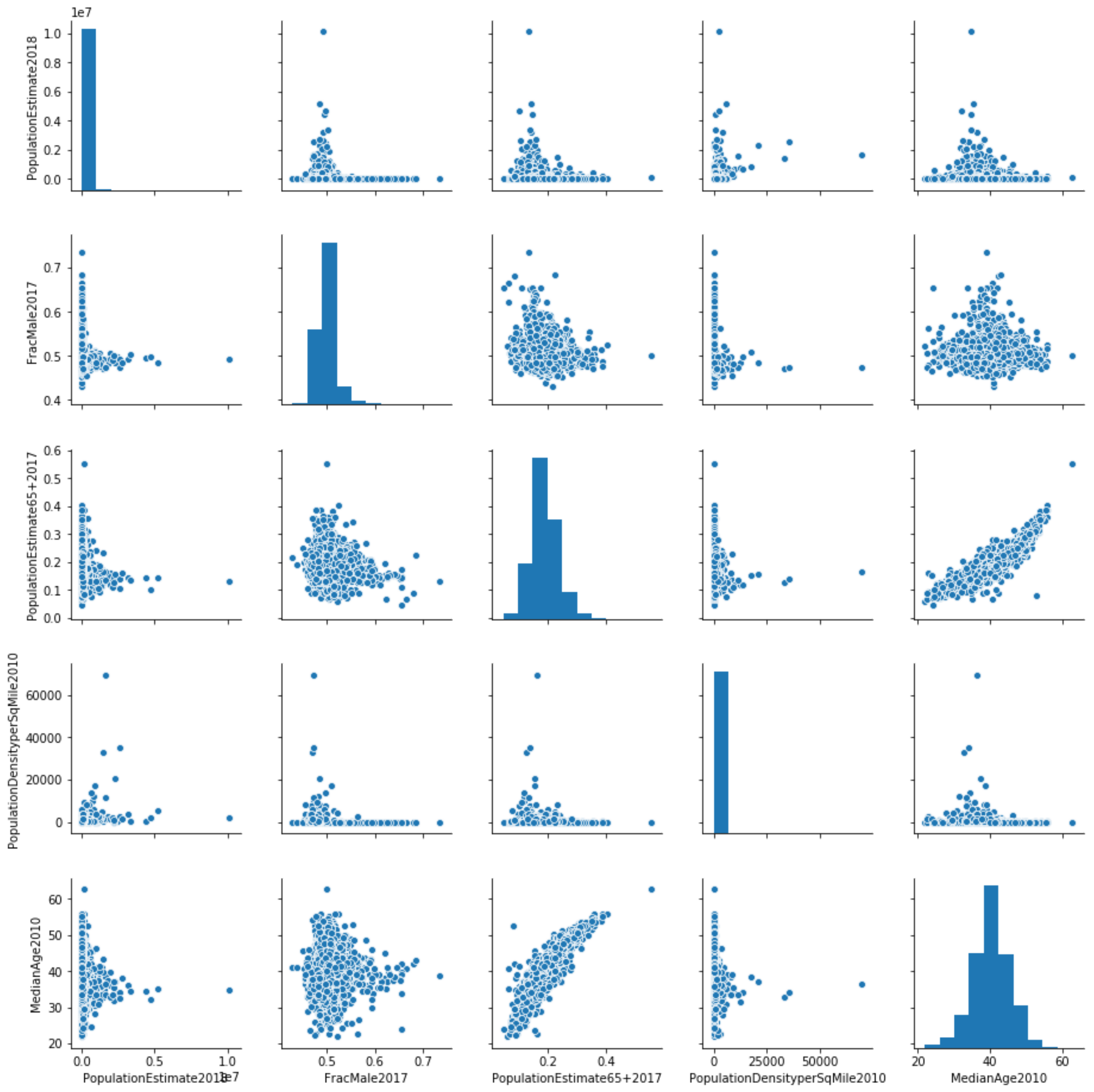


Observation:

- The mortality variables seem to be highly correlated to each other
- I decide to keep only `3-YrMortalityAge<1Year2015-17` , `3-YrMortalityAge1-4Years2015-17` , and `3-YrMortalityAge5-14Years2015-17`

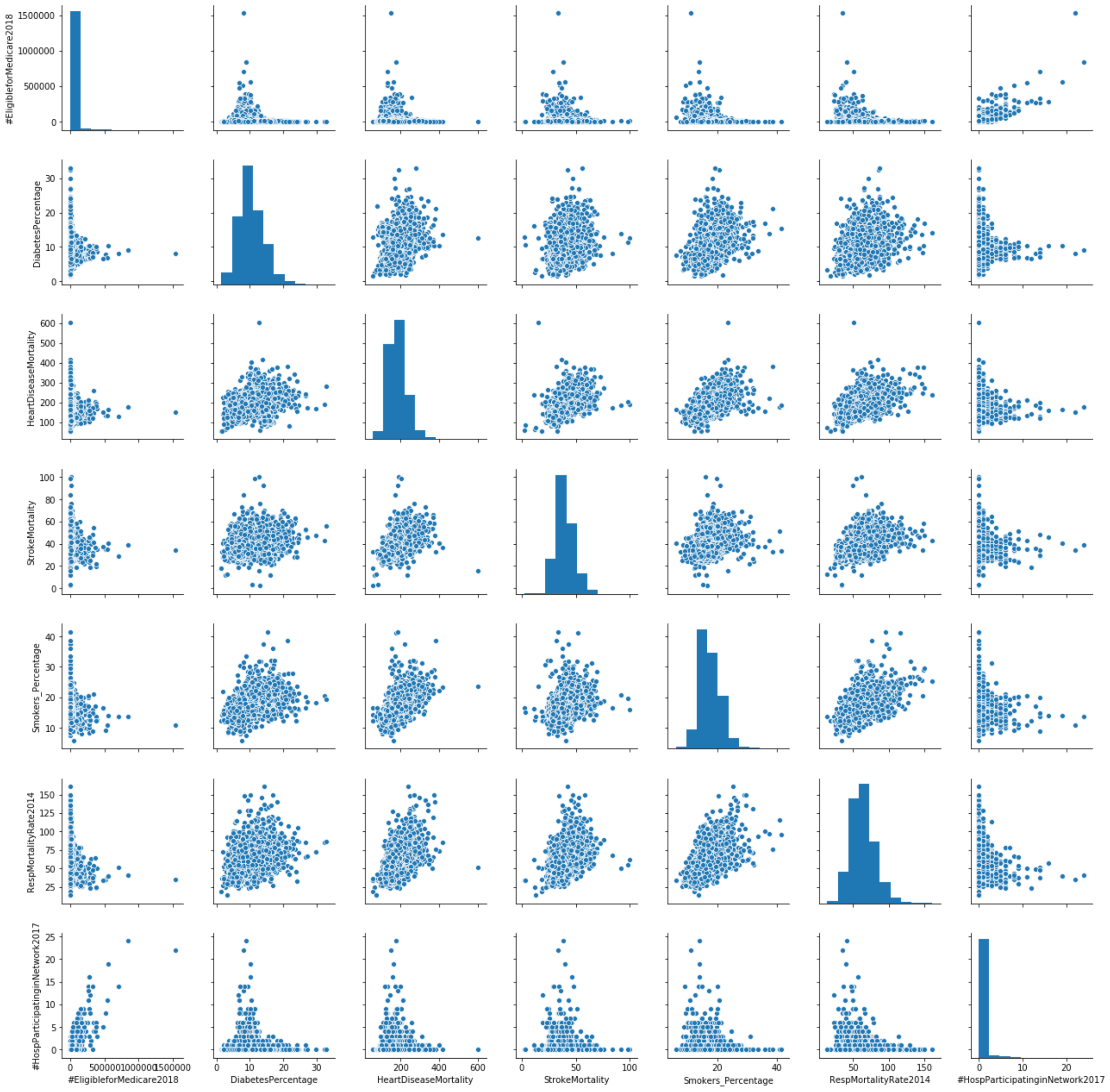
In [27]:
```
demographics_revised = [
    'PopulationEstimate2018',
       'FracMale2017', 'PopulationEstimate65+2017',
       'PopulationDensityperSqMile2010',
       'MedianAge2010'
]
```

In [28]:
```
sns.pairplot(df1[demographics_revised]);
```



```
In [29]: health_revised = [
             '#EligibleforMedicare2018',
                'DiabetesPercentage', 'HeartDiseaseMortality', 'StrokeMortality',
                'Smokers_Percentage', 'RespMortalityRate2014',
                '#HospParticipatinginNetwork2017'
         ]
```

```
In [30]: sns.pairplot(df1[health_revised]);
```



Summary of this part

- I have removed redundant features
- After the feature engineering described above, the counties data are ready to use

1.2 Examine confirmed data

In [31]:
```
# Only the US data will be used
df2 = df2[df2['iso2']=='US']
df2.head()
```

Out[31]:

	UID	iso2	iso3	code3	FIPS	Admin2	Province_State	Country_Region	Lat	Long_
5	84001001	US	USA	840	1001.0	Autauga	Alabama	US	32.539527	-86.644082
6	84001003	US	USA	840	1003.0	Baldwin	Alabama	US	30.727750	-87.722071
7	84001005	US	USA	840	1005.0	Barbour	Alabama	US	31.868263	-85.387129
8	84001007	US	USA	840	1007.0	Bibb	Alabama	US	32.996421	-87.125115

	UID	iso2	iso3	code3	FIPS	Admin2	Province_State	Country_Region	Lat	Long_
9	84001009	US	USA	840	1009.0	Blount	Alabama	US	33.982109	-86.567906

5 rows × 99 columns

In [32]:
```
df2.keys()
```

Out[32]:
```
Index(['UID', 'iso2', 'iso3', 'code3', 'FIPS', 'Admin2', 'Province_State',
       'Country_Region', 'Lat', 'Long_', 'Combined_Key', '1/22/20', '1/23/20',
       '1/24/20', '1/25/20', '1/26/20', '1/27/20', '1/28/20', '1/29/20',
       '1/30/20', '1/31/20', '2/1/20', '2/2/20', '2/3/20', '2/4/20', '2/5/20',
       '2/6/20', '2/7/20', '2/8/20', '2/9/20', '2/10/20', '2/11/20', '2/12/20',
       '2/13/20', '2/14/20', '2/15/20', '2/16/20', '2/17/20', '2/18/20',
       '2/19/20', '2/20/20', '2/21/20', '2/22/20', '2/23/20', '2/24/20',
       '2/25/20', '2/26/20', '2/27/20', '2/28/20', '2/29/20', '3/1/20',
       '3/2/20', '3/3/20', '3/4/20', '3/5/20', '3/6/20', '3/7/20', '3/8/20',
       '3/9/20', '3/10/20', '3/11/20', '3/12/20', '3/13/20', '3/14/20',
       '3/15/20', '3/16/20', '3/17/20', '3/18/20', '3/19/20', '3/20/20',
       '3/21/20', '3/22/20', '3/23/20', '3/24/20', '3/25/20', '3/26/20',
       '3/27/20', '3/28/20', '3/29/20', '3/30/20', '3/31/20', '4/1/20',
       '4/2/20', '4/3/20', '4/4/20', '4/5/20', '4/6/20', '4/7/20', '4/8/20',
       '4/9/20', '4/10/20', '4/11/20', '4/12/20', '4/13/20', '4/14/20',
       '4/15/20', '4/16/20', '4/17/20', '4/18/20'],
      dtype='object')
```

In [33]:
```
df2['Province_State'].unique()
```

Out[33]:
```
array(['Alabama', 'Alaska', 'Arizona', 'Arkansas', 'California',
       'Colorado', 'Connecticut', 'Delaware', 'District of Columbia',
       'Florida', 'Georgia', 'Hawaii', 'Idaho', 'Illinois', 'Indiana',
       'Iowa', 'Kansas', 'Kentucky', 'Louisiana', 'Maine', 'Maryland',
       'Massachusetts', 'Michigan', 'Minnesota', 'Mississippi',
       'Missouri', 'Montana', 'Nebraska', 'Nevada', 'New Hampshire',
       'New Jersey', 'New Mexico', 'New York', 'North Carolina',
       'North Dakota', 'Ohio', 'Oklahoma', 'Oregon', 'Pennsylvania',
       'Rhode Island', 'South Carolina', 'South Dakota', 'Tennessee',
       'Texas', 'Utah', 'Vermont', 'Virginia', 'Washington',
       'West Virginia', 'Wisconsin', 'Wyoming', 'Diamond Princess',
       'Grand Princess'], dtype=object)
```

Observation:

- `FIPS` is the foreign key to be used to merge with the counties dataset
- `Diamond Princess` and `Grand Princess` are considered nuisance
  - However, since these observations don't have a `FIPS`, they will be dropped when merging with counties dataset

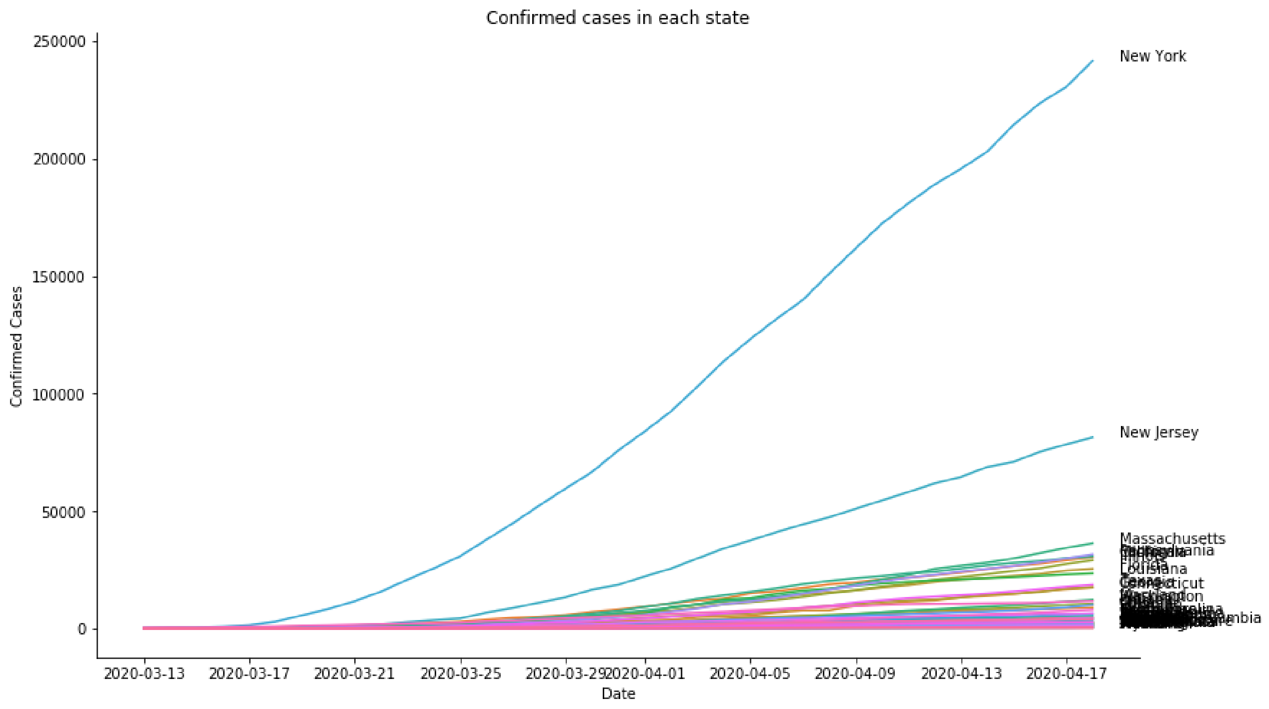
1.2.1 Visualization of total confirmed cases of each state per day

In [34]:
```
# Reshape the original dataset as panel data
confirmed = df2.drop(['UID', 'iso2', 'iso3', 'code3', 'FIPS', 'Admin2',
                      'Country_Region', 'Lat', 'Long_', 'Combined_Key'],1).set_index('P

confirmed_state = confirmed.stack().rename('Confirmed').reset_index()

confirmed_state['Date'] = pd.to_datetime(confirmed_state['level_1'])

# I consider the dates after the federal guideline
```

```
confirmed_state = confirmed_state[confirmed_state['Date']>'2020-03-12']

# Remove the nuisance "Diamond Princess" and "Grand Princess" observations
confirmed_state = confirmed_state[~confirmed_state['Province_State'].isin(['Diamond Pri

# Sum at the state level
confirmed_state = confirmed_state.groupby(['Province_State', 'Date']).sum().reset_index
```

In [35]:
```
# Prepare for the labels in the visualization
labels_confirmed = confirmed_state[confirmed_state['Date']=='2020-04-18'].groupby('Prov
```

In [36]:
```
plt.figure(figsize=(13,8))
ax = sns.lineplot(x='Date', y='Confirmed', hue='Province_State',
             data=confirmed_state, ci=None, legend=None)

for n in range(len(labels_confirmed)):
    i,j = labels_confirmed.iloc[n]
    ax.text(x=pd.to_datetime('2020-4-19'), y=j, s=i)

plt.title('Confirmed cases in each state')
plt.xlabel('Date')
plt.ylabel('Confirmed Cases')
sns.despine();
```



Summary of this part

- New York and New Jersey have way more cases than other states
- These differences may not be captured by the features in the counties data, so it's worthwhile to include state fixed effect using one-hot encoding

1.2.2 Prepare for a dataset to be merged with the counties data

In [37]:
```
# Keep the variables that will be used
confirmed2 = df2.drop(['UID', 'iso2', 'iso3', 'code3', 'Province_State', 'Admin2',
                      'Country_Region', 'Lat', 'Long_', 'Combined_Key'],1).set_index('F
```

```
# Compute the first difference, reshape as panel data
confirmed_fips = confirmed2.stack().rename('Confirmed_delta').diff().reset_index()
confirmed_fips['Confirmed_delta'] = confirmed_fips['Confirmed_delta'].map(lambda x: max

# Clean the date format
confirmed_fips['Date'] = pd.to_datetime(confirmed_fips['level_1'])
confirmed_fips = confirmed_fips.drop('level_1',1)

# Rename the foreign key
confirmed_fips = confirmed_fips.rename(columns={'FIPS': 'countyFIPS'})

# Drop the observations with FIPS missing
confirmed_fips = confirmed_fips.dropna()

confirmed_fips.head()
```

Out[37]:

	countyFIPS	Confirmed_delta	Date
1	1001.0	0.0	2020-01-23
2	1001.0	0.0	2020-01-24
3	1001.0	0.0	2020-01-25
4	1001.0	0.0	2020-01-26
5	1001.0	0.0	2020-01-27

1.3 Examine the deaths data

Since this dataset is largely similar to the confirmed data, I will omit most of the comments for simplicity

In [38]:
```
# Only the US data will be used
df3 = df3[df3['iso2']=='US']
df3.head()
```

Out[38]:

	UID	iso2	iso3	code3	FIPS	Admin2	Province_State	Country_Region	Lat	Long_
5	84001001	US	USA	840	1001.0	Autauga	Alabama	US	32.539527	-86.644082
6	84001003	US	USA	840	1003.0	Baldwin	Alabama	US	30.727750	-87.722071
7	84001005	US	USA	840	1005.0	Barbour	Alabama	US	31.868263	-85.387129
8	84001007	US	USA	840	1007.0	Bibb	Alabama	US	32.996421	-87.125115
9	84001009	US	USA	840	1009.0	Blount	Alabama	US	33.982109	-86.567906

5 rows × 100 columns

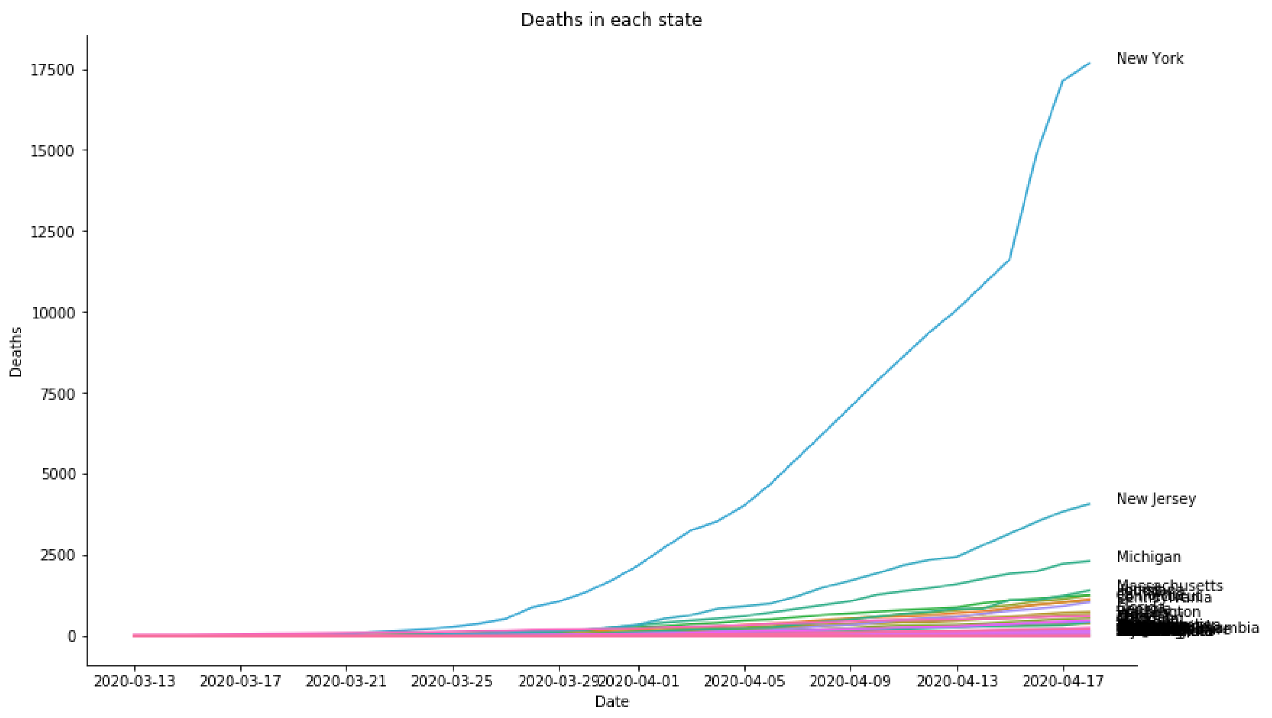
1.3.1 Visualization of total deaths of each state per day

In [39]:
```
death = df3.drop(['UID', 'iso2', 'iso3', 'code3', 'FIPS', 'Admin2',
                  'Country_Region', 'Lat', 'Long_', 'Combined_Key', 'Population'],1

death_state = death.stack().rename('Death').reset_index()
```

```
death_state['Date'] = pd.to_datetime(death_state['level_1'])

death_state = death_state[death_state['Date']>'2020-03-12']

death_state = death_state[~death_state['Province_State'].isin(['Diamond Princess', 'Gra

death_state = death_state.groupby(['Province_State', 'Date']).sum().reset_index()
```

In [40]:
```
labels_death = death_state[death_state['Date']=='2020-04-18'].groupby('Province_State')
```

In [41]:
```
plt.figure(figsize=(13,8))
ax = sns.lineplot(x='Date', y='Death', hue='Province_State',
             data=death_state, ci=None, legend=None)

for n in range(len(labels_death)):
    i,j = labels_death.iloc[n]
    ax.text(x=pd.to_datetime('2020-4-19'), y=j, s=i)

plt.title('Deaths in each state')
plt.xlabel('Date')
plt.ylabel('Deaths')
sns.despine();
```



1.3.2 Prepare for a dataset to be merged with the counties data

In [42]:
```
death2 = df3.drop(['UID', 'iso2', 'iso3', 'code3', 'Province_State', 'Admin2',
                      'Country_Region', 'Lat', 'Long_', 'Combined_Key','Population'],1)
death_fips = death2.stack().rename('Deaths_delta').diff().reset_index()
death_fips['Deaths_delta'] = death_fips['Deaths_delta'].map(lambda x: max(x,0))
death_fips['Date'] = pd.to_datetime(death_fips['level_1'])
death_fips = death_fips.drop('level_1',1)
death_fips = death_fips.rename(columns={'FIPS': 'countyFIPS'})

death_fips = death_fips.dropna()
death_fips['Days'] = (death_fips['Date']-pd.to_datetime('2020-01-22')).dt.days
death_fips.head()
```

Out[42]:

	countyFIPS	Deaths_delta	Date	Days
1	1001.0	0.0	2020-01-23	1
2	1001.0	0.0	2020-01-24	2
3	1001.0	0.0	2020-01-25	3
4	1001.0	0.0	2020-01-26	4
5	1001.0	0.0	2020-01-27	5

2. Merge datasets and more feature engineering

In [43]:

```
data = pd.merge(confirmed_fips, death_fips)
data = pd.merge(data, df1, on='countyFIPS', how='left')

# Keep the data after the federal guideline date
data = data[data['Date']>'2020-3-11']

# Drop observations without county/state information
data = data[(data['StateName'].notnull()) & (data['countyFIPS'].notnull())]
```

2.1 Policy variables

- The policy dates are ordinal qualitative data, so tranformation is need in order to extract useful information
  - One can to one-hot encoding to extract day-specific information, but there would be too many vairables (because of too many days), so I don't see this efficient
  - Note that the federal guideline starts from 3/12/2020 (737495), I transform all the policy variables to how many says since the federal guideline began
  - A missing value indicates that the county does not implement this policy, so the corresponding number of days will be replaced with 0

In [44]:

```
for i,j in zip(policies, policy_days):
    data[j] = (data['Date']-data[i]).dt.days
    data.loc[data[j]<0, j] = 0

data[policy_days] = data[policy_days].fillna(0)
```

In [45]:

```
data[policy_days].describe()
```

Out[45]:

	stay_home	50_gather	500_gather	schools	dine	ent_gym
count	119320.000000	119320.000000	119320.000000	119320.000000	119320.000000	119320.000000
mean	4.725679	9.239809	10.790672	12.048282	11.076584	10.186130
std	6.928790	9.325456	10.219542	10.084381	9.853800	9.789852
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	2.000000	0.000000	0.000000
50%	0.000000	7.000000	9.000000	11.000000	10.000000	8.000000

	stay_home	50_gather	500_gather	schools	dine	ent_gym
75%	9.000000	17.000000	19.000000	21.000000	19.000000	18.000000
max	29.000000	33.000000	37.000000	33.000000	36.000000	36.000000

2.2 One-hot encoding for states

In [46]:

```
data = pd.concat([data, pd.get_dummies(data['StateName'], drop_first=True)], 1)
# data['NY'] = (data['StateName'] == 'NY')
# data['NJ'] = (data['StateName'] == 'NJ')
```

2.3 Construct the final dataset for modelling

In [47]:

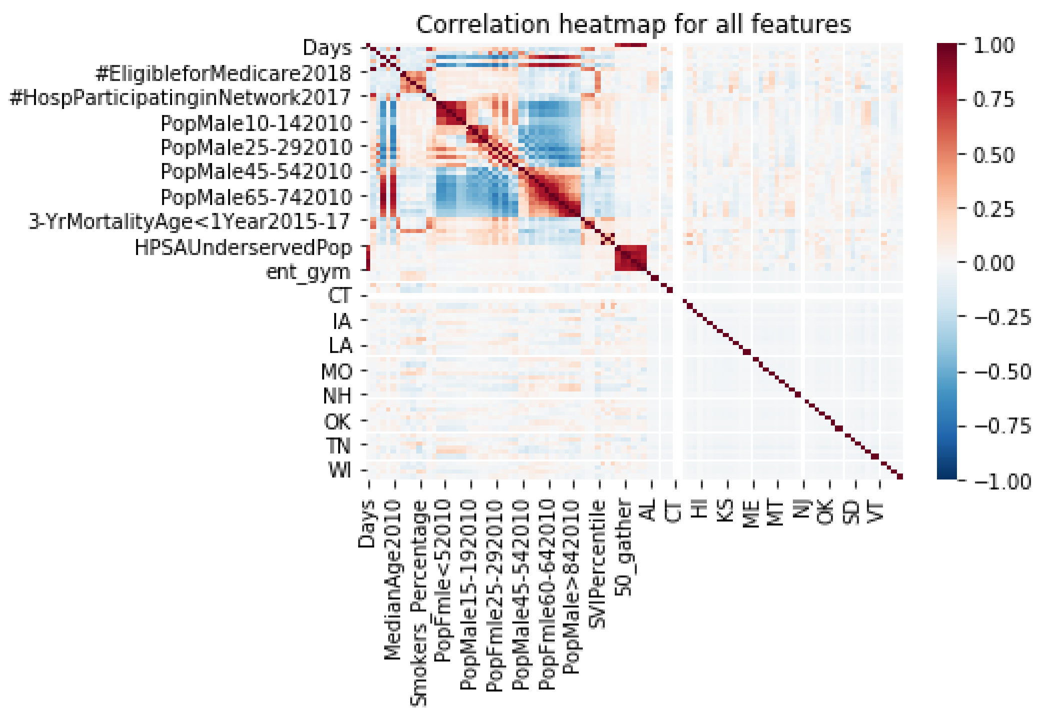
```
# Include all the features

features = ['Days', 'PopulationEstimate2018',
        'FracMale2017', 'PopulationEstimate65+2017',
        'PopulationDensityperSqMile2010',
        'MedianAge2010', '#EligibleforMedicare2018',
        'DiabetesPercentage', 'HeartDiseaseMortality', 'StrokeMortality',
        'Smokers_Percentage', 'RespMortalityRate2014',
        '#HospParticipatinginNetwork2017', 'dem_to_rep_ratio', 'PopMale<52010',
        'PopFmle<52010', 'PopMale5-92010', 'PopFmle5-92010', 'PopMale10-142010',
        'PopFmle10-142010', 'PopMale15-192010', 'PopFmle15-192010',
        'PopMale20-242010', 'PopFmle20-242010', 'PopMale25-292010',
        'PopFmle25-292010', 'PopMale30-342010', 'PopFmle30-342010',
        'PopMale35-442010', 'PopFmle35-442010', 'PopMale45-542010',
        'PopFmle45-542010', 'PopMale55-592010', 'PopFmle55-592010',
        'PopMale60-642010', 'PopFmle60-642010', 'PopMale65-742010',
        'PopFmle65-742010', 'PopMale75-842010', 'PopFmle75-842010',
        'PopMale>842010', 'PopFmle>842010', '3-YrMortalityAge<1Year2015-17',
        '3-YrMortalityAge1-4Years2015-17', '3-YrMortalityAge5-14Years2015-17',
        'SVIPercentile', 'HPSAShortage', 'HPSAServedPop', 'HPSAUnderservedPop',
        'stay_home', '50_gather', '500_gather', 'schools', 'dine', 'ent_gym', 'AL', 'AR'
        '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']
```

In [48]:

```
# Construct the "cleaned_data" as the final dataset for modelling
cleaned_data = data[['Confirmed_delta','Deaths_delta', 'Date', 'CountyName', 'StateName

# In order to utilize log-linear models, I add a small number to confirmed and deaths t
cleaned_data['Confirmed_delta'] = cleaned_data['Confirmed_delta'] + 1e-7
cleaned_data['Deaths_delta'] = cleaned_data['Deaths_delta'] + 1e-7
```

In [49]:

```
# Sanity check before delivering the dataset
sns.heatmap(cleaned_data[features].corr(),vmin=-1,vmax=1,cmap='RdBu_r')
plt.title('Correlation heatmap for all features');
```



Observations:

- The features seem to be in a good shape, so I am ready to finalize the dataset.

2.4 Train test split

In [50]:
```
# Use 1/3 data as the test data
X_train, X_test, confirmed_train, confirmed_test, deaths_train, deaths_test = \
train_test_split(cleaned_data[features], cleaned_data['Confirmed_delta'],
                 cleaned_data['Deaths_delta'], test_size=0.33, random_state=42)
```

3. Model Selection

Overview:

- Both confirmed cases and deaths are discrete quantitative vairables, and can be as large as thousands
- The prediction problem is more appropriate to be considered as a regression problem
  - If it's considered as a multiclass classification problem, I will have thousands of classes. The model won't be well behaved
- For a regression problem, the simplest model would be OLS.
  - So I start from simple OLS, and gradually add up model complexity to fit the data
- To measure a model's performance:
  - 2-fold cross validation is implemented
  - Since I have a regression problem, the averages of both the training and validation r-squared's of all cross-validation sets are reported

3.1 Simple OLS

In [51]:
```
reg = linear_model.LinearRegression(normalize=True, n_jobs=-1)
```

In [52]:
```
score = cross_validate(reg, X_train, confirmed_train, cv=2, scoring='r2', return_train_
print('OLS: Confirmed')
print('Training r-squared:', score['train_score'].mean())
print('Validation r-squared:', score['test_score'].mean())

score = cross_validate(reg, X_train, deaths_train, cv=2, scoring='r2', return_train_sco
print('\nOLS: Deaths')
print('Training r-squared:', score['train_score'].mean())
print('Validation r-squared:', score['test_score'].mean())
```

```
OLS: Confirmed
Training r-squared: 0.3909565782301141
Validation r-squared: 0.36007762592893594

OLS: Deaths
Training r-squared: 0.1571141426317253
Validation r-squared: 0.09637372316467013
```

Observations:

- The prediction performance is very poor
- The r-squared's suggest that I underfit the data, so complexity is needed
- I come up with two ways to add complexity:
  - I can make this model more advanced, such as a GLM (generalized linear model) or a non-linear model
  - I can create more complicated features, such as the second-order polynomial features

3.2 Log linear

- One reason that simple OLS performs poorly might be that OLS can make negative predictions, but the counts data cannot be negative
- So I decide to try out the log linear model in order to restrict the dependent variable to be positive:

$$y = \exp(X\beta + \varepsilon) \implies \log y = X\beta + \varepsilon$$

In [53]:
```
score = cross_validate(reg, X_train, np.log(confirmed_train), cv=2, scoring='r2', retur
print('Log linear: Confirmed')
print('Training r-squared:', score['train_score'].mean())
print('Validation r-squared:', score['test_score'].mean())

score = cross_validate(reg, X_train, np.log(deaths_train), cv=2, scoring='r2', return_t
print('\nLog linear: Deaths')
print('Training r-squared:', score['train_score'].mean())
print('Validation r-squared:', score['test_score'].mean())
```

```
Log linear: Confirmed
Training r-squared: 0.3739229845324598
Validation r-squared: 0.37043691666428125

Log linear: Deaths
Training r-squared: 0.2097651126872943
Validation r-squared: 0.20242304417402696
```

Observations:

- The log linear model does seem to help to some extent, so it's worth to consider it in the modelling process
- However, the r-squared's still suggest that the prediction is poor

3.3 OLS with second-order polynomial features

In [54]:
```
print('Number of all second-order polynomial features:', PolynomialFeatures(2).fit(X_tr
```

```
Number of all second-order polynomial features: 5671
```

- After including all the second order and interaction terms, there will be more than 5000 features
- Very likely that OLS without regularization will overfit the data, but I think it's worth a try

In [55]:
```
poly_reg = Pipeline([
  ('polynomial_features', PolynomialFeatures(2)),
  ('linear_model', linear_model.LinearRegression(normalize=True, n_jobs=-1))
])
```

In [56]:
```
score = cross_validate(poly_reg, X_train, confirmed_train, cv=2, scoring='r2', return_t
print('OLS with polynomial features: Confirmed')
print('Training r-squared:', score['train_score'].mean())
print('Validation r-squared:', score['test_score'].mean())

score = cross_validate(poly_reg, X_train, deaths_train, cv=2, scoring='r2', return_trai
print('\nOLS with polynomial features: Deaths')
print('Training r-squared:', score['train_score'].mean())
print('Validation r-squared:', score['test_score'].mean())
```

```
OLS with polynomial features: Confirmed
Training r-squared: 0.854220045546282
Validation r-squared: -634892052.1018882

OLS with polynomial features: Deaths
Training r-squared: 0.4795652862748853
Validation r-squared: -22116344237.841408
```

Observations:

- The validation r-squared's suggest that I insanely overfit the data
- Regularization is needed

3.4 Lasso with second-order polynomial features

- Since I may have too many features, and Lasso has the functionality of feature selection, I decide to implement Lasso
- The hyperparameter, alpha, is tuned through 3-fold cross-validation

In [57]:
```
poly_lasso = Pipeline([
  ('polynomial_features', PolynomialFeatures(2)),
  ('linear_model', linear_model.LassoCV(n_alphas=10, normalize=True, max_iter=1000000,
])
```

In [58]:
```
score = cross_validate(poly_lasso, X_train, confirmed_train, cv=2, scoring='r2', return
print('Lasso with polynomial features: Confirmed')
print('Training r-squared:', score['train_score'].mean())
print('Validation r-squared:', score['test_score'].mean())

score = cross_validate(poly_lasso, X_train, deaths_train, cv=2, scoring='r2', return_tr
print('\nLasso with polynomial features: Deaths')
print('Training r-squared:', score['train_score'].mean())
print('Validation r-squared:', score['test_score'].mean())
```

```
Lasso with polynomial features: Confirmed
Training r-squared: 0.8230187952255568
Validation r-squared: 0.7942997010394165

Lasso with polynomial features: Deaths
Training r-squared: 0.24026543165581032
Validation r-squared: 0.13910356073632646
```

In [ ]:
```
score = cross_validate(poly_lasso, X_train, np.log(confirmed_train), cv=2, scoring='r2'
print('Log linear Lasso with polynomial features: Confirmed')
print('Training r-squared:', score['train_score'].mean())
print('Validation r-squared:', score['test_score'].mean())

score = cross_validate(poly_lasso, X_train, np.log(deaths_train), cv=2, scoring='r2', r
print('\nLog linear Lasso with polynomial features: Deaths')
print('Training r-squared:', score['train_score'].mean())
print('Validation r-squared:', score['test_score'].mean())
```

Observations:

- $L1$ regularization reduces a bit training r-squared, but the validation r-squared looks much better
- The performance on `confirmed` is better than `deaths`
- Log linearization doesn't seem to help in the Lasso case
- Overall, given the time I have and the class material of Data 100/200, I think the Lasso model with polynomial features without log linearization is good to go

3.5 Ridge with second-order polynomial features

- Since Lasso turns out to have a decent performance, it may be also worthwhile to try the Ridge regression
- The hyperparameter alpha is tuned through 3-fold cross validation

In [57]:
```
poly_ridge = Pipeline([
  ('polynomial_features', PolynomialFeatures(2)),
  ('linear_model', linear_model.RidgeCV(alphas=(.1,1,10), normalize=True, cv=3))
])
```

In [62]:
```
score = cross_validate(poly_ridge, X_train, confirmed_train, cv=2, scoring='r2', return
print('Ridge with polynomial features: Confirmed')
print('Training r-squared:', score['train_score'].mean())
print('Validation r-squared:', score['test_score'].mean())

score = cross_validate(poly_ridge, X_train, deaths_train, cv=2, scoring='r2', return_tr
print('\nRidge with polynomial features: Deaths')
print('Training r-squared:', score['train_score'].mean())
print('Validation r-squared:', score['test_score'].mean())
```

```
Ridge with polynomial features: Confirmed
```

```
Training r-squared: 0.7909532096300802
Validation r-squared: 0.7824834684710544

Ridge with polynomial features: Deaths
Training r-squared: 0.2535918179656203
Validation r-squared: 0.2263477165603991
```

In [63]:
```
score = cross_validate(poly_ridge, X_train, np.log(confirmed_train), cv=2, scoring='r2'
print('Log linear Ridge with polynomial features: Confirmed')
print('Training r-squared:', score['train_score'].mean())
print('Validation r-squared:', score['test_score'].mean())

score = cross_validate(poly_ridge, X_train, np.log(death_train), cv=2, scoring='r2', re
print('\nLog linear Ridge with polynomial features: Deaths')
print('Training r-squared:', score['train_score'].mean())
print('Validation r-squared:', score['test_score'].mean())
```

```
Log linear Ridge with polynomial features: Confirmed
Training r-squared: 0.4188682137091883
Validation r-squared: 0.41323619574162757

Log linear Ridge with polynomial features: Deaths
Training r-squared: 0.4188682137091883
Validation r-squared: 0.41323619574162757
```

Observations:

- The performance of Ridge is similar to Lasso, but is slightly worse for all cases
- So, Lasso is preferred in this case

3.6 Kernel ridge

- As I mentioned earlier, to increase the model complexity, I can either add more features or make the model more advanced
- I have tried extensively with the second-order polynomial features above, now I switch gear to try a non-linear model, which is Kernel regression
- I decide to use the rbf kernel with $L2$ regularization
- The alpha can be tuned by cross validation. But due to the long computing time, I omit the cross validation process and let alpha=0.1

In [59]:
```
kernel_ridge = KernelRidge(alpha=.1, kernel="rbf")
```

In [ ]:
```
score = cross_validate(kernel_ridge, X_train, confirmed_train, cv=2, scoring='r2', retu
print('Kernel ridge: Confirmed')
print('Training r-squared:', score['train_score'].mean())
print('Validation r-squared:', score['test_score'].mean())

score = cross_validate(kernel_ridge, X_train, deaths_train, cv=2, scoring='r2', return_
print('\nKernel ridge: Deaths')
print('Training r-squared:', score['train_score'].mean())
print('Validation r-squared:', score['test_score'].mean())
```

In [63]:
```
score = cross_validate(kernel_ridge, X_train, np.log(confirmed_train), cv=2, scoring='r
print('Log linear Kernel ridge: Confirmed')
print('Training r-squared:', score['train_score'].mean())
print('Validation r-squared:', score['test_score'].mean())
```

```
score = cross_validate(kernel_ridge, X_train, np.log(deaths_train), cv=2, scoring='r2',
print('\nLog linear Kernel ridge: Deaths')
print('Training r-squared:', score['train_score'].mean())
print('Validation r-squared:', score['test_score'].mean())
```

```
Log linear Kernel ridge: Confirmed
Training r-squared: 0.9841151845566585
Validation r-squared: 0.8416335956004222

Log linear Kernel ridge: Deaths
Training r-squared: 0.9841151845566585
Validation r-squared: 0.8416335956004222
```

Observations:

- The performance of Kernel ridge outperforms all the linear models for all cases
- The validation r-squared's suggest that I slightly overfit the data, but I still have a decent performance

3.7 Final model selection

- Based on the experiments I have so far, the Kernel ridge model has the best performance
- Unfortunately, the model takes too much time to train (>6 hours). Due to time constraint, I have to use the second-best, Lasso with polynomial features without log linearization
- Before shipping the model, let's train the model using the entire training data and check of its performance on the testing data

In [60]:
```
confirmed_model = poly_lasso
deaths_model = poly_lasso
```

In [ ]:
```
# Fit the final model to all the confirmed training data
confirmed_model.fit(X_train, confirmed_train);
```

In [ ]:
```
# Report testing performance for confirmed
confirmed_train_score = confirmed_model.score(X_train, confirmed_train)
confirmed_test_score = confirmed_model.score(X_test, confirmed_test)
print('Final model: Confirmed')
print('Training r-squared:', confirmed_train_score)
print('Testing r-squared:', confirmed_test_score)
```

In [ ]:
```
# Fit the kernel_ridge model to all the deaths training data
deaths_model.fit(X_train, deaths_train);
```

In [ ]:
```
deaths_train_score = deaths_model.score(X_train, deaths_train)
deaths_test_score = deaths_model.score(X_test, deaths_test)
print('\nFinal model: Deaths')
print('Training r-squared:', deaths_train_score)
print('Testing r-squared:', deaths_test_score)
```

The r-squared values suggest that this model has reasonable performance on both the confirmed and deaths datasets.

3.8 Ship the model

Now, I am ready to ship the model. I train the model using all the data available.

In [61]:
```
# Fit the final model to all the training data
confirmed_model.fit(cleaned_data[features], cleaned_data['Confirmed_delta'])
deaths_model.fit(cleaned_data[features], cleaned_data['Deaths_delta']);
```

4. Counterfactual: What if there's no policy?

- My question:
  If the corresponding policies to Covid-19 are implemented 2 weeks in advance, what will happen to the confirmed and deaths counts for all the counties in California?
- Since I don't have a randomized controlled trial (RCT), I cannot declare that the inference is *causal*. Instead, the analysis only suggests the *correlation* between policy implementation and confirmed/deaths count
  - In other words, I am trying to answer a different but related question: What would be the predicted confirmed and death counts if observe the counties with exactly the same demographics as the counties in California, with the Covid-19 corresponding policies adopted 14 days in advance?
- To do this:
  - I first create a counterfactual dataset with all the counties in the United States, but shut down all policy variables (=0)
  - I predict the confirmed and deaths counts for these counterfactual counties
  - then compare the differences and conclude my findings

In [105…
```
calif_factual = cleaned_data[cleaned_data['StateName']=='CA'].copy()
calif_counterfactual = calif_factual.copy()
calif_counterfactual[policy_days] = calif_counterfactual[policy_days] + 14
```

In [106…
```
calif_counterfactual['Confirmed_delta_hat'] = confirmed_model.predict(calif_counterfact
calif_counterfactual['Deaths_delta_hat'] = deaths_model.predict(calif_counterfactual[fe
```

In [107…
```
confirmed_factual = calif_factual.groupby('CountyName')['Confirmed_delta'].sum().sum()
confirmed_counterfactual = calif_counterfactual.groupby('CountyName')['Confirmed_delta_

print('Confirmed difference:', np.round(confirmed_factual - confirmed_counterfactual))
```

```
Confirmed difference: 29744.0
```

In [108…
```
deaths_factual = calif_factual.groupby('CountyName')['Deaths_delta'].sum().sum()
deaths_counterfactual = calif_counterfactual.groupby('CountyName')['Deaths_delta_hat'].

print('Death difference:', np.round(deaths_factual - deaths_counterfactual))
```

```
Death difference: 538.0
```

In [120…
```
fl_factual = cleaned_data[cleaned_data['StateName']=='FL'].copy()
fl_counterfactual = fl_factual.copy()
fl_counterfactual[policy_days] = fl_counterfactual[policy_days] + 14
fl_counterfactual['Confirmed_delta_hat'] = confirmed_model.predict(fl_counterfactual[fe
fl_counterfactual['Deaths_delta_hat'] = deaths_model.predict(fl_counterfactual[features

confirmed_factual = fl_factual.groupby('CountyName')['Confirmed_delta'].sum().sum()
confirmed_counterfactual = fl_counterfactual.groupby('CountyName')['Confirmed_delta_hat

print('Confirmed difference:', np.round(confirmed_factual - confirmed_counterfactual))
```

```
deaths_factual = fl_factual.groupby('CountyName')['Deaths_delta'].sum().sum()
deaths_counterfactual = fl_counterfactual.groupby('CountyName')['Deaths_delta_hat'].sum

print('Death difference:', np.round(deaths_factual - deaths_counterfactual))
```

```
Confirmed difference: 24812.0
Death difference: 50.0
```

In [119…

```
tx_factual = cleaned_data[cleaned_data['StateName']=='TX'].copy()
tx_counterfactual = tx_factual.copy()
tx_counterfactual[policy_days] = tx_counterfactual[policy_days] + 14
tx_counterfactual['Confirmed_delta_hat'] = confirmed_model.predict(tx_counterfactual[fe
tx_counterfactual['Deaths_delta_hat'] = deaths_model.predict(tx_counterfactual[features

confirmed_factual = tx_factual.groupby('CountyName')['Confirmed_delta'].sum().sum()
confirmed_counterfactual = tx_counterfactual.groupby('CountyName')['Confirmed_delta_hat

print('Confirmed difference:', np.round(confirmed_factual - confirmed_counterfactual))
deaths_factual = tx_factual.groupby('CountyName')['Deaths_delta'].sum().sum()
deaths_counterfactual = tx_counterfactual.groupby('CountyName')['Deaths_delta_hat'].sum

print('Death difference:', np.round(deaths_factual - deaths_counterfactual))
```

```
Confirmed difference: 16105.0
Death difference: -2184.0
```

Findings:

If I observe the counties with exactly the same demographics as the counties in California but with the Covid-19 corresponding policies adopted 2 weeks in advance, I would have around 29000 less confirmed cases and around 500 deaths.