final_proj

May 26, 2020

1 The Covid-19 Pandemic: How Healthcare and Economic Factors Influence Death Rate

Eshani Mehta, Emily Wang, Shauna Hannani

1.0.1 Importing Libraries

We start by importing the libraries we will use later in the project.

[1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline

import json
from urllib.request import urlopen

import seaborn as sns
sns.set(style = "whitegrid",
        color_codes = True,
        font_scale = 1.5)

import plotly.express as px
import plotly.graph_objects as go

from sklearn import linear_model as lm
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn import preprocessing
```

1.0.2 Reading in Data

We then read the datasets that we will use throughout this project. * The counties dataset contains county-based information such as health factors, population, and some economic factors. * The confirmed dataset contains the number of cases in each county through 5/12/20. * The deaths dataset contains the number of deaths in each county through 5/12/20.

We also brought in 2 extra datasets from the USDA's county level data sets (link: https://www.ers.usda.gov/data-products/county-level-data-sets/download-data/). * The poverty dataset contains county-based information such as the percent of people under poverty level. * The unemployment dataset contains county-based information such as the unemployment rate in 2018 and the median household income in 2018.

[2]:
```python
#states = pd.read_csv('finalproj/data/5.12states.csv')
counties = pd.read_csv('abridged_couties.csv')
confirmed = pd.read_csv('new_time_series_covid19_confirmed_US.csv')
deaths = pd.read_csv('new_time_series_covid19_deaths_US.csv')

#Following data sets taken from the USDA's county level data sets
poverty = pd.read_csv('PovertyEstimates.csv')
unemployment = pd.read_csv('Unemployment.csv', thousands=',')
```

[3]:
```python
counties.head()
```

[3]:
```
  countyFIPS  STATEFP  COUNTYFP CountyName StateName    State        lat  \
0      01001      1.0       1.0    Autauga        AL  Alabama  32.540091
1      01003      1.0       3.0    Baldwin        AL  Alabama  30.738314
2      01005      1.0       5.0    Barbour        AL  Alabama  31.874030
3      01007      1.0       7.0       Bibb        AL  Alabama  32.999024
4      01009      1.0       9.0     Blount        AL  Alabama  33.990440

         lon  POP_LATITUDE  POP_LONGITUDE  … >500 gatherings public schools  \
0 -86.645649     32.500389     -86.494165  …        737497.0        737500.0
1 -87.726272     30.548923     -87.762381  …        737497.0        737500.0
2 -85.397327     31.844036     -85.310038  …        737497.0        737500.0
3 -87.125260     33.030921     -87.127659  …        737497.0        737500.0
4 -86.562711     33.955243     -86.591491  …        737497.0        737500.0

   restaurant dine-in  entertainment/gym  federal guidelines  \
0            737503.0           737512.0            737500.0
1            737503.0           737512.0            737500.0
2            737503.0           737512.0            737500.0
3            737503.0           737512.0            737500.0
4            737503.0           737512.0            737500.0

   foreign travel ban  SVIPercentile  HPSAShortage  HPSAServedPop  \
0            737495.0         0.4354           NaN            NaN
1            737495.0         0.2162           NaN            NaN
2            737495.0         0.9959          6.08         5400.0
3            737495.0         0.6003          2.75        14980.0
4            737495.0         0.4242          7.21        31850.0
```

```
   HPSAUnderservedPop
0                 NaN
1                 NaN
2             18241.0
3              6120.0
4             25233.0

[5 rows x 87 columns]
```

[4]:
```python
confirmed.head()
```

[4]:
```
   UID iso2 iso3  code3  FIPS Admin2           Province_State Country_Region  \
0   16   AS  ASM     16  60.0    NaN           American Samoa             US
1  316   GU  GUM    316  66.0    NaN                     Guam             US
2  580   MP  MNP    580  69.0    NaN  Northern Mariana Islands             US
3  630   PR  PRI    630  72.0    NaN              Puerto Rico             US
4  850   VI  VIR    850  78.0    NaN           Virgin Islands             US

       Lat     Long_  … 5/3/20  5/4/20  5/5/20  5/6/20  5/7/20  5/8/20  \
0 -14.2710 -170.1320  …      0       0       0       0       0       0
1  13.4443  144.7937  …    145     145     145     149     149     151
2  15.0979  145.6739  …     14      14      14      15      15      15
3  18.2208  -66.5901  …   1808    1843    1924    1968    2031    2156
4  18.3358  -64.8963  …     66      66      66      66      66      68

   5/9/20  5/10/20  5/11/20  5/12/20
0       0        0        0        0
1     151      151      151      152
2      16       16       19       19
3    2173     2198     2256     2299
4      68       69       69       69

[5 rows x 123 columns]
```

[5]:
```python
deaths.head()
```

[5]:
```
   UID iso2 iso3  code3  FIPS Admin2           Province_State Country_Region  \
0   16   AS  ASM     16  60.0    NaN           American Samoa             US
1  316   GU  GUM    316  66.0    NaN                     Guam             US
2  580   MP  MNP    580  69.0    NaN  Northern Mariana Islands             US
3  630   PR  PRI    630  72.0    NaN              Puerto Rico             US
4  850   VI  VIR    850  78.0    NaN           Virgin Islands             US

       Lat     Long_  … 5/3/20  5/4/20  5/5/20  5/6/20  5/7/20  5/8/20  \
0 -14.2710 -170.1320  …      0       0       0       0       0       0
1  13.4443  144.7937  …      5       5       5       5       5       5
2  15.0979  145.6739  …      2       2       2       2       2       2
```

```
3  18.2208  -66.5901  …     97      97      99      99     102     107
4  18.3358  -64.8963  …      4       4       4       4       4       4

   5/9/20  5/10/20  5/11/20  5/12/20
0       0        0        0        0
1       5        5        5        5
2       2        2        2        2
3     108      111      113      114
4       4        4        5        6

[5 rows x 124 columns]
```

[6]:
```
poverty.head()
```

[6]:
```
   FIPStxt Stabr       Area_name  Rural-urban_Continuum_Code_2003  \
0        0    US   United States                              NaN
1     1000    AL         Alabama                              NaN
2     1001    AL  Autauga County                              2.0
3     1003    AL  Baldwin County                              4.0
4     1005    AL  Barbour County                              6.0

   Urban_Influence_Code_2003  Rural-urban_Continuum_Code_2013  \
0                        NaN                              NaN
1                        NaN                              NaN
2                        2.0                              2.0
3                        5.0                              3.0
4                        6.0                              6.0

   Urban_Influence_Code_2013 POVALL_2018 CI90LBAll_2018 CI90UBALL_2018  …  \
0                        NaN  41,852,315     41,619,366     42,085,264  …
1                        NaN     801,758        785,668        817,848  …
2                        2.0       7,587          6,334          8,840  …
3                        2.0      21,069         17,390         24,748  …
4                        6.0       6,788          5,662          7,914  …

   CI90UB517P_2018  MEDHHINC_2018  CI90LBINC_2018 CI90UBINC_2018 POV04_2018  \
0             17.2         61,937          61,843         62,031  3,758,704
1             23.7         49,881          49,123         50,639     73,915
2             23.9         59,338          53,628         65,048        NaN
3             16.9         57,588          54,437         60,739        NaN
4             45.9         34,382          31,157         37,607        NaN

  CI90LB04_2018  CI90UB04_2018  PCTPOV04_2018  CI90LB04P_2018 CI90UB04P_2018
0     3,714,862      3,802,546           19.5            19.3           19.7
1        69,990         77,840           26.0            24.6           27.4
2           NaN            NaN            NaN             NaN            NaN
3           NaN            NaN            NaN             NaN            NaN
```

```
4          NaN          NaN          NaN          NaN          NaN

[5 rows x 34 columns]
```

[7]:
```
unemployment.head()
```

[7]:
```
   FIPStxt Stabr            area_name  Rural_urban_continuum_code_2013  \
0        0    US        United States                              NaN
1     1000    AL              Alabama                              NaN
2     1001    AL   Autauga County, AL                              2.0
3     1003    AL   Baldwin County, AL                              3.0
4     1005    AL   Barbour County, AL                              6.0

   Urban_influence_code_2013  Metro_2013   Civilian_labor_force_2000   \
0                        NaN         NaN                 142601667.0
1                        NaN         NaN                   2133223.0
2                        2.0         1.0                     21720.0
3                        2.0         1.0                     69533.0
4                        6.0         0.0                     11373.0

    Employed_2000    Unemployed_2000   Unemployment_rate_2000  …  \
0     136904680.0          5696987.0                      4.0  …
1       2035594.0            97629.0                      4.6  …
2         20846.0              874.0                      4.0  …
3         66971.0             2562.0                      3.7  …
4         10748.0              625.0                      5.5  …

   Civilian_labor_force_2018  Employed_2018  Unemployed_2018  \
0                161389026.0    155102319.0        6286707.0
1                  2216627.0      2130845.0          85782.0
2                    26196.0        25261.0            935.0
3                    95233.0        91809.0           3424.0
4                     8414.0         7987.0            427.0

   Unemployment_rate_2018   Civilian_labor_force_2019    Employed_2019   \
0                     3.9                 163100055.0      157115247.0
1                     3.9                   2241747.0        2174483.0
2                     3.6                     26172.0          25458.0
3                     3.6                     97328.0          94675.0
4                     5.1                      8537.0           8213.0

    Unemployed_2019   Unemployment_rate_2019  Median_Household_Income_2018   \
0         5984808.0                      3.7                       61937.0
1           67264.0                      3.0                       49881.0
2             714.0                      2.7                       59338.0
3            2653.0                      2.7                       57588.0
4             324.0                      3.8                       34382.0
```

```
   Med_HH_Income_Percent_of_State_Total_2018
0                                        NaN
1                                      100.0
2                                      119.0
3                                      115.5
4                                       68.9

[5 rows x 88 columns]
```

1.1 Data Cleaning

1.1.1 Merging Relevant Datasets

The first step is creating a merged dataset with each county's information, the number of confirmed cases, the number of deaths, and other relevant information from other datasets.

We create a copy to ensure that the original `counties` dataframe is intact.

Since the "countyFIPS" column currently is type str, we convert it to numbers so we can match it with the "FIPS" columns in `confirmed` and `deaths`. We also drop any empty values for FIPS since all valid counties have a FIPS value.

The FIPS columns serve as a primary key since it's a standardized id to identify counties that is present in most county datasets even online.

We first inner merge `merged` with `confirmed` on the FIPS columns, specifically only looking at the "5/12/20" column since that contains the most updated number of cases per county. We then rename this to a more usable name so it'll be easier to reference later.

We then similarly inner merge the new `merged` with `deaths` on the FIPS columns, specifically only looking at the "5/12/20" column since that contains the most updated number of deaths per county. We then rename this to a more usable name so it'll be easier to reference later.

[8]:
```
merged = counties.copy()

merged["countyFIPS"] = pd.to_numeric(merged["countyFIPS"], errors='coerce')
merged = merged.dropna(subset=["countyFIPS"])

#merge w confirmed
merged = merged.merge(right=confirmed[["FIPS", "5/12/20"]], how='inner',␣
 ↪left_on='countyFIPS', right_on='FIPS')
merged = merged.rename(columns={'5/12/20' : 'confirmed'})

#merged w deaths
merged = merged.merge(right=deaths[["FIPS", "5/12/20"]], how='inner',␣
 ↪left_on='countyFIPS', right_on='FIPS')
merged = merged.rename(columns={'5/12/20' : 'deaths'})
```

```python
#drop irrelevant columns
merged = merged.drop(["FIPS_x", "FIPS_y"], axis = 1)
merged
```

[8]:

```
      countyFIPS  STATEFP  COUNTYFP                           CountyName  \
0         1001.0      1.0       1.0                              Autauga
1         1003.0      1.0       3.0                              Baldwin
2         1005.0      1.0       5.0                              Barbour
3         1007.0      1.0       7.0                                 Bibb
4         1009.0      1.0       9.0                               Blount
…            …       …        …                                    …
3135      2195.0      2.0     195.0                   Petersburg Borough
3136      2198.0      2.0     198.0  Prince of Wales-Hyder Census Area
3137      2230.0      2.0     230.0                 Skagway Municipality
3138      2275.0      2.0     275.0            Wrangell City and Borough
3139     15005.0     15.0       5.0                              Kalawao

     StateName    State        lat         lon  POP_LATITUDE  POP_LONGITUDE  \
0           AL  Alabama  32.540091  -86.645649     32.500389     -86.494165
1           AL  Alabama  30.738314  -87.726272     30.548923     -87.762381
2           AL  Alabama  31.874030  -85.397327     31.844036     -85.310038
3           AL  Alabama  32.999024  -87.125260     33.030921     -87.127659
4           AL  Alabama  33.990440  -86.562711     33.955243     -86.591491
…          …       …         …          …           …             …
3135        AK      NaN        NaN         NaN     56.812712    -133.115025
3136        AK      NaN        NaN         NaN     55.448164    -132.560842
3137        AK      NaN        NaN         NaN     59.464536    -135.311501
3138        AK      NaN        NaN         NaN     56.385821    -132.310837
3139        HI      NaN        NaN         NaN     21.188495    -156.979972

      … restaurant dine-in entertainment/gym  federal guidelines  \
0     …           737503.0          737512.0            737500.0
1     …           737503.0          737512.0            737500.0
2     …           737503.0          737512.0            737500.0
3     …           737503.0          737512.0            737500.0
4     …           737503.0          737512.0            737500.0
…   …                …                 …                   …
3135  …           737501.0          737501.0            737500.0
3136  …           737501.0          737501.0            737500.0
3137  …           737501.0          737501.0            737500.0
3138  …           737501.0          737501.0            737500.0
3139  …           737504.0          737509.0            737500.0

      foreign travel ban  SVIPercentile  HPSAShortage  HPSAServedPop  \
0               737495.0         0.4354           NaN            NaN
1               737495.0         0.2162           NaN            NaN
```

```
2              737495.0        0.9959        6.08        5400.0
3              737495.0        0.6003        2.75       14980.0
4              737495.0        0.4242        7.21       31850.0
…                   …             …           …           …
3135           737495.0        0.6650         NaN           NaN
3136           737495.0        0.7662        1.50        1050.0
3137           737495.0        0.1685        0.69         175.0
3138           737495.0        0.5618         NaN           NaN
3139           737495.0        0.3162         NaN           NaN

      HPSAUnderservedPop  confirmed  deaths
0                    NaN         91       4
1                    NaN        227       7
2                18241.0         67       1
3                 6120.0         46       1
4                25233.0         45       0
…                      …      …       …
3135                 NaN          4       1
3136              5260.0          2       0
3137              2412.0          0       0
3138                 NaN          0       0
3139                 NaN          0       0

[3140 rows x 89 columns]
```

1.1.2 Merging External Datasets

Here, we merge the two USDA datasets in a similar fashion to what we did above. We convert the FIPS columns in `poverty` and `unemployment` to numeric data again and select the columns from each that we want to use in our analysis before performing an inner merge.

[9]:
```
merged2 = merged.copy()
poverty["FIPStxt"] = pd.to_numeric(poverty["FIPStxt"], errors='coerce')
unemployment["FIPStxt"] = pd.to_numeric(unemployment["FIPStxt"],␣
↪errors='coerce')

#merge w poverty to add and rename pov_pct
merged2 = merged2.merge(right=poverty[["FIPStxt", "PCTPOVALL_2018"]],␣
↪how='inner', left_on='countyFIPS', right_on='FIPStxt')
merged2 = merged2.rename(columns={"PCTPOVALL_2018" : "pov_pct"})

#merge w unemployment to add and rename unemploy_rate and␣
↪median_household_income
merged2 = merged2.merge(right=unemployment[["FIPStxt",␣
↪"Unemployment_rate_2018", "Median_Household_Income_2018"]], how='inner',␣
↪left_on='countyFIPS', right_on='FIPStxt')
```

```
merged2 = merged2.rename(columns={"Unemployment_rate_2018" : "unemploy_rate",␣
↪"Median_Household_Income_2018": "med_income"})
```

1.1.3 Adding Relevant Columns and Some Data Cleaning

We create a new copy of all the merged datasets. We want to first add a `case_rate` column, which is just the # of cases divided by the total population of a county. We also multiply this by 100 to make it easier to read.

We want to then add a `death_rate` column, which is the # of deaths divided by the # of confirmed cases. Before we this, we remove the 272 counties that have 0 confirmed cases so we don't have a division by 0 error. Since this project is also aiming to look at effects of certain factors on the case_rate and death_rate, counties with no cases don't add anything to our data. We also multiply this by 100 to make it easier to read.

Other columns we added: * `old` is the percent of the population that is age 65+ * `inMedicare` is the percent of Medicare eligible people who are actually enrolled * `medicare_rate` is the percent of the population that is eligible for Medicare

We also noticed in the dataset that counties with latitude and longitude equal to 0 or missing were other territories or not valid counties or lacking important information, so we remove all instances where that is true.

We decided we should analyze which columns were missing more data than others, which ended up including include ["3-Yr Diabetes", all the "3-YrMortality", "stay at home" (396), "HPSA" (1013)].

We also checked on what states were included in the data after cleaning. The data now covered 48 states (excluding Alaska and Hawaii since they're not continental and including District of Columbia).

[10]:
```
#add case rate
data = merged2.copy()
data["case_rate"] = data["confirmed"] / data["PopulationEstimate2018"]
data["case_rate"] = data["case_rate"] * 100

#add death rate
data = data.loc[data["confirmed"] != 0]
data["death_rate"] = data["deaths"] / data["confirmed"]
data["death_rate"] = data["death_rate"] * 100

#add age column
data["old"] = (data['PopulationEstimate65+2017'] /␣
↪data["PopulationEstimate2018"]) * 100

#add medicare cols
data = data.dropna(subset=["MedicareEnrollment,AgedTot2017"])
data["inMedicare"] = (data["MedicareEnrollment,AgedTot2017"] /␣
↪data["#EligibleforMedicare2018"]) * 100
```

```
data["medicare_rate"] = (data["#EligibleforMedicare2018"] /␣
↪data["PopulationEstimate2018"]) * 100

#add hospitals and icu_beds per 1000 people
data["hospitals/1000ppl"] = (data["#Hospitals"] /␣
↪data["PopulationEstimate2018"]) * 1000
data["icu_beds/1000ppl"] = (data["#ICU_beds"] / data["PopulationEstimate2018"])␣
↪* 1000

#drop null or 0 latitudes
data = data.loc[data["lat"] != 0]
data = data.dropna(subset=["lat"])

data.isna().sum()
#note: columms with many na values include ["3-Yr Diabetes", all the␣
↪"3-YrMortality", "stay at home" (396), "HPSA" (1013)]

data.State.unique()
#48 states (excluding Alaska and Hawaii since they're not continental and␣
↪including District of Columbia)

data
```

[10]:

```
      countyFIPS  STATEFP  COUNTYFP  CountyName StateName     State  \
0         1001.0      1.0       1.0     Autauga        AL   Alabama
1         1003.0      1.0       3.0     Baldwin        AL   Alabama
2         1005.0      1.0       5.0     Barbour        AL   Alabama
3         1007.0      1.0       7.0        Bibb        AL   Alabama
4         1009.0      1.0       9.0      Blount        AL   Alabama
…            …        …        …          …         …         …
3128     56039.0     56.0      39.0       Teton        WY   Wyoming
3129     56041.0     56.0      41.0       Uinta        WY   Wyoming
3130     56043.0     56.0      43.0    Washakie        WY   Wyoming
3132      8014.0      8.0      14.0  Broomfield        CO  Colorado
3133     12086.0     12.0      86.0  Miami-Dade        FL   Florida

            lat         lon  POP_LATITUDE  POP_LONGITUDE  … FIPStxt_y  \
0     32.540091  -86.645649     32.500389     -86.494165  …      1001
1     30.738314  -87.726272     30.548923     -87.762381  …      1003
2     31.874030  -85.397327     31.844036     -85.310038  …      1005
3     32.999024  -87.125260     33.030921     -87.127659  …      1007
4     33.990440  -86.562711     33.955243     -86.591491  …      1009
…           …           …            …             …  …        …
3128  43.713556 -110.570974     43.494174    -110.784353  …     56039
3129  41.289323 -110.553036     41.271860    -110.767519  …     56041
3130  43.909060 -107.679282     44.012142    -107.911552  …     56043
3132  39.963039 -105.058542     39.936888    -105.055491  …      8014
```

```
3133  25.607895  -80.587502    25.774565    -80.298888  …     12086

     unemploy_rate  med_income  case_rate  death_rate        old  inMedicare  \
0              3.6     59338.0   0.163666    4.395604  15.093254   70.338316
1              3.6     57588.0   0.104118    3.083700  19.453541   76.812263
2              5.1     34382.0   0.269282    1.492537  19.119006   70.438557
3              3.9     46064.0   0.205357    2.173913  16.214286   66.088144
4              3.5     50412.0   0.077801    0.000000  17.895920   72.425047
…              …           …          …           …          …          …
3128           2.9     99087.0   0.424592    0.000000  14.509770   83.505976
3129           4.2     63401.0   0.044337    0.000000  13.315927   72.092376
3130           4.1     55190.0   0.101458    0.000000  21.280913   82.449182
3132           2.8     96924.0   0.290181    9.950249  13.072603   73.026762
3133           3.5     52043.0   0.520897    3.510601  15.898321   84.000125

      medicare_rate  hospitals/1000ppl  icu_beds/1000ppl
0         20.573371           0.017985          0.107912
1         24.834650           0.013760          0.233921
2         26.851815           0.040191          0.200957
3         22.892857           0.044643          0.000000
4         22.778354           0.017289          0.103734
…               …                  …                …
3128      16.312118           0.043326          0.259954
3129      17.705306           0.049264          0.295581
3130      25.580216           0.126823          0.000000
3132      17.047079           0.014437          0.288738
3133      17.370847           0.006156          0.214732

[2812 rows x 101 columns]
```

1.2 Exploratory Data Analysis

1.2.1 Effects of the timing of Stay at Home orders on case_rate and death_rate

Since a big portion of the news today is focused on stay at home orders, we wanted to see if there was any obvious effect of when the stay at home orders were put in place and the rates of cases and deaths in a county.

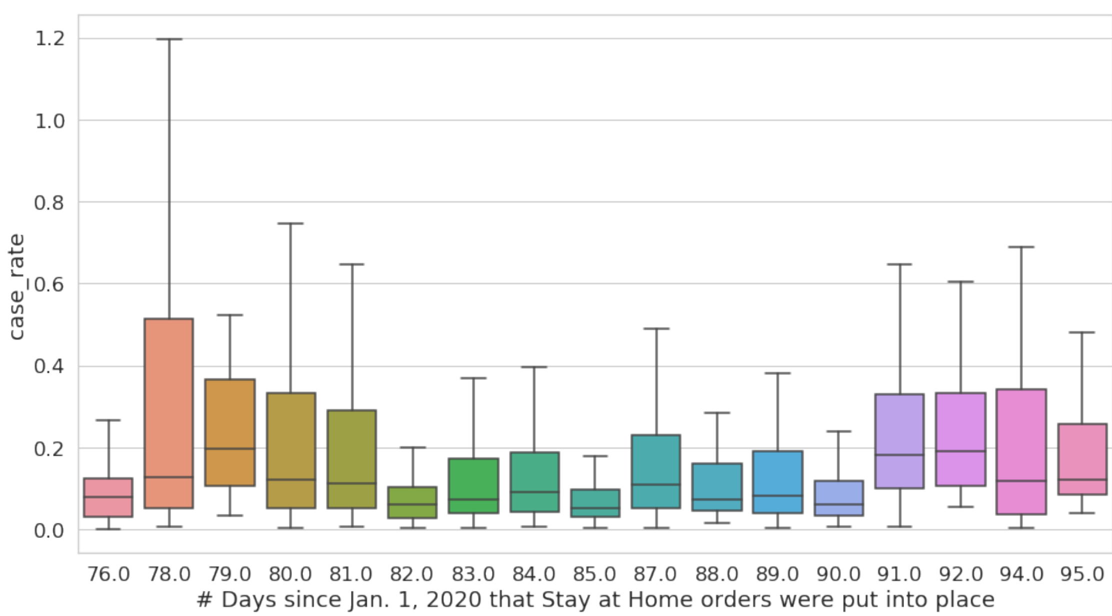
After calculating a more readable number of days for the "stay at home" column, we made boxplots for each of case_rate and death_rate. We decided to hide outliers to make the plot more readable, and there were many outliers that fell way beyond the current graph axes.

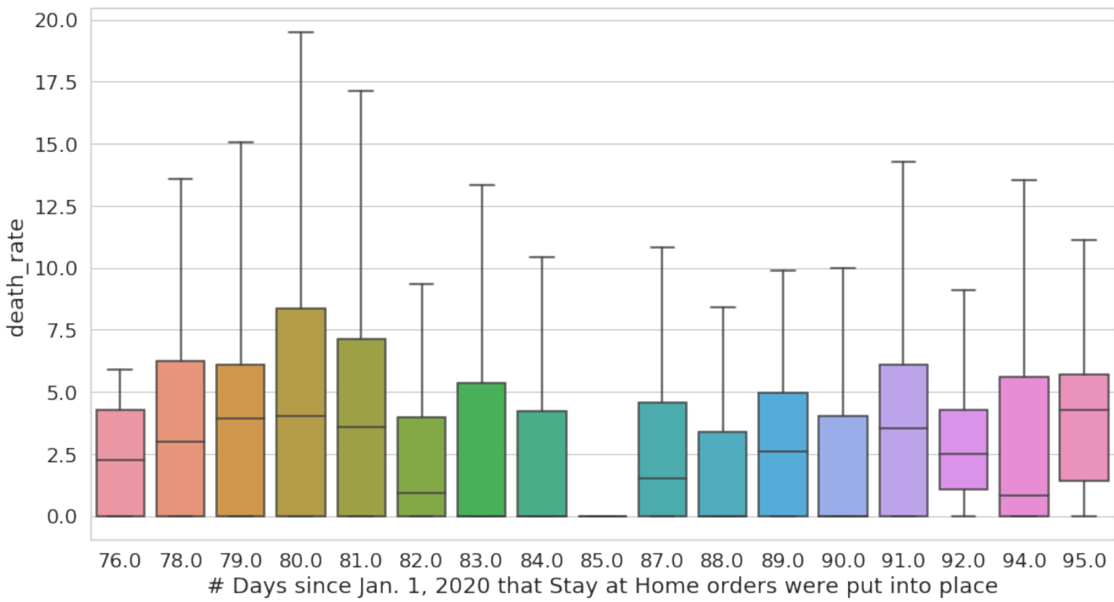
[11]:
```
dates = data.copy()

#737427 is the proleptic Gregorian ordinal of January 1, 2020
#we subtract that so we can see how many days into the new year the order was␣
↪put into place
```

```
dates["stayHome"] = dates["stay at home"] - 737427

#drop all counties where there is no stay at home order established since this␣
↪is about timing
dates = dates.dropna(subset=["stayHome"])

print("Stay at home dates range (in days since Jan. 1, 2020): ", "March 17,␣
↪2020 (", min(dates["stayHome"]), " days) - April 5, 2020 (",␣
↪max(dates["stayHome"]), " days)")

#boxplot of case_rate
plt.figure(figsize = (15, 8))
sns.boxplot(x="stayHome", y = "case_rate", data=dates, showfliers=False)
plt.xlabel("# Days since Jan. 1, 2020 that Stay at Home orders were put into␣
↪place")

#boxplot of death_rate
plt.figure(figsize = (15, 8))
sns.boxplot(x="stayHome", y = "death_rate", data=dates, showfliers=False)
plt.xlabel("# Days since Jan. 1, 2020 that Stay at Home orders were put into␣
↪place")
```

```
Stay at home dates range (in days since Jan. 1, 2020):  March 17, 2020 ( 76.0
days) - April 5, 2020 ( 95.0  days)
```

[11]:
```
Text(0.5, 0, '# Days since Jan. 1, 2020 that Stay at Home orders were put into
place')
```





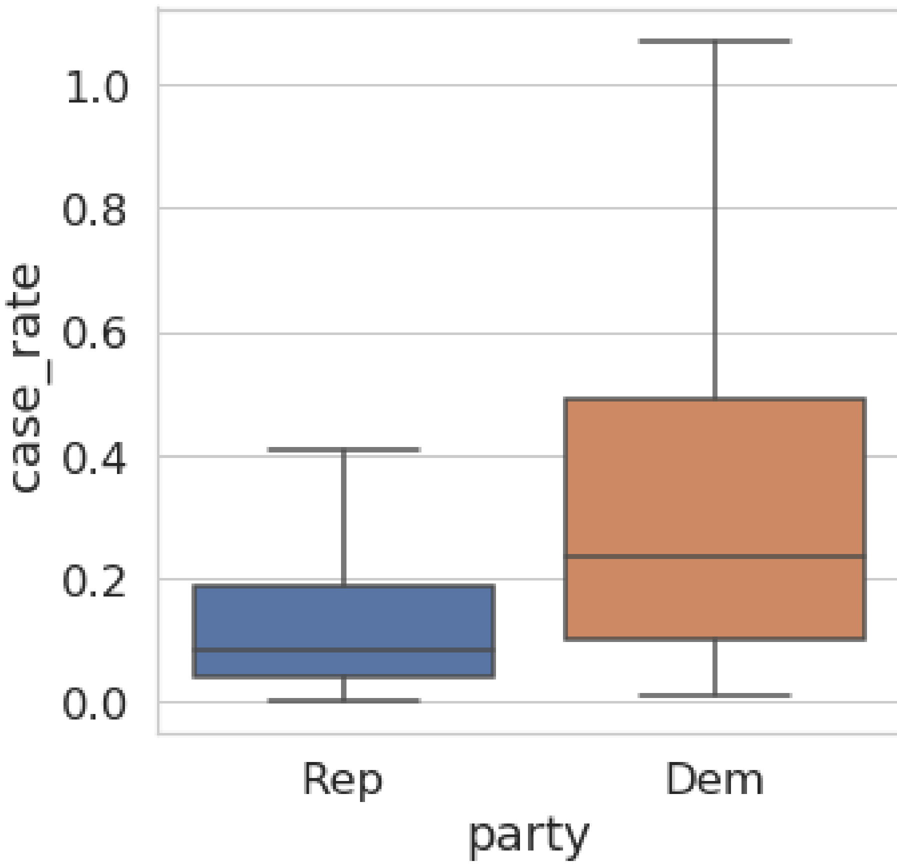
1.2.2 Political Affiliation of County and Correlation with case_rate and death_rate

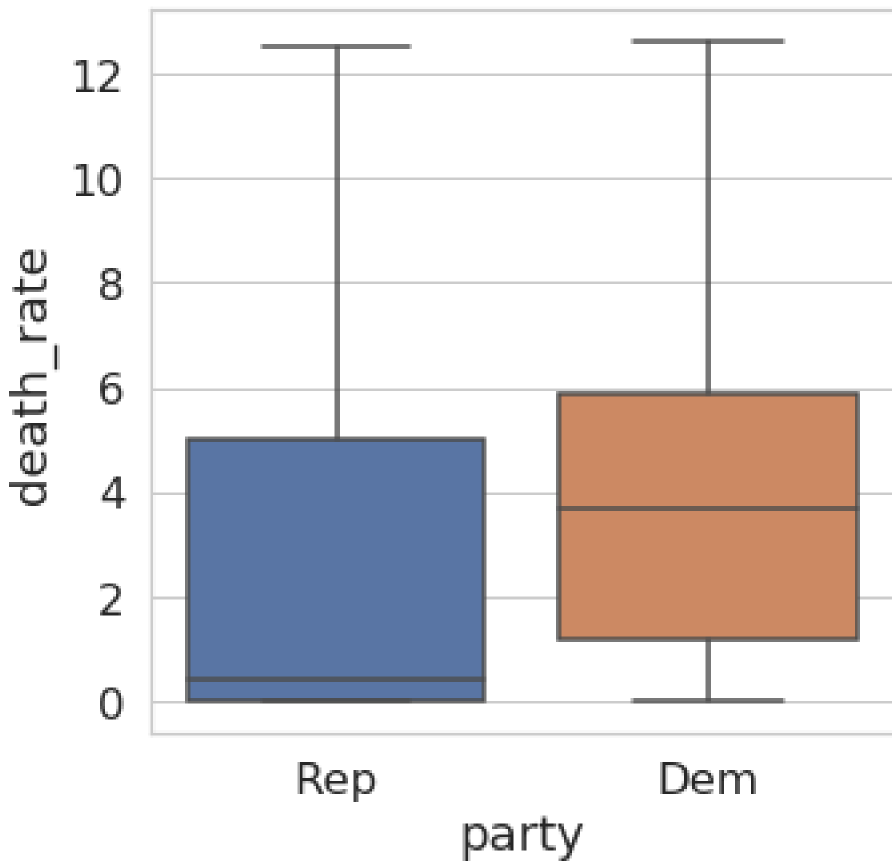
We found the dem_to_rep_ratio column interesting, and since many people on either side have differing opinions about the pandemic and social distancing, we thought it would be interesting to see if there was any correlation to case_rate or death_rate.

We categorized the ratios into democratic and republican and then created boxplots for each factor.

[12]:
```
political = data.copy()
political = political.dropna(subset=["dem_to_rep_ratio"])

#categorize
political.loc[political["dem_to_rep_ratio"] > 1, "party"] = "Dem"
political.loc[political["dem_to_rep_ratio"] < 1, "party"] = "Rep"
political[["party"]]

#make boxplot for case_rate
plt.figure(figsize = (5, 5))
sns.boxplot(x="party", y = "case_rate", data=political, showfliers=False)

#make boxplot for death_rate
plt.figure(figsize = (5, 5))
sns.boxplot(x="party", y = "death_rate", data=political, showfliers=False)
```

[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f380c270710>
```


1.0
0.8
0.6
0.4
0.2
0.0
case_rate
Rep
Dem
party



1.2.3 Exploring counties and the # confirmed cases

Here, we just wanted to know which counties had the most confirmed cases as of 5/12. We weren't surprised to see New York, of course, but many of the other names were unexpected.

We also calculated the mean and median number of confirmed cases per county. Since we already removed the 0 case counties, we were able to get a better value. The mean was obviously influenced by large outliers like the top 10 shown here. The median was interesting since we didn't expect it to be low because we usually only hear about the counties with large numbers of cases on the news. However, it makes sense that most counties don't have as many, especially ones in rural areas.

[13]:
```
print("Mean number of confirmed cases per county: ", np.mean(data["confirmed"]))
print("Median number of confirmed cases per county: ", np.
 ↪median(data["confirmed"]))
print("Total number of counties: ", data.shape[0])

#looking at the counties with the most cases and deaths
most_cases = data.sort_values("confirmed", ascending = False)
most_cases[["CountyName", "StateName", "confirmed"]].head(10)
```

```
Mean number of confirmed cases per county:  480.7528449502134
Median number of confirmed cases per county:  31.0
Total number of counties:  2812
```

[13]:

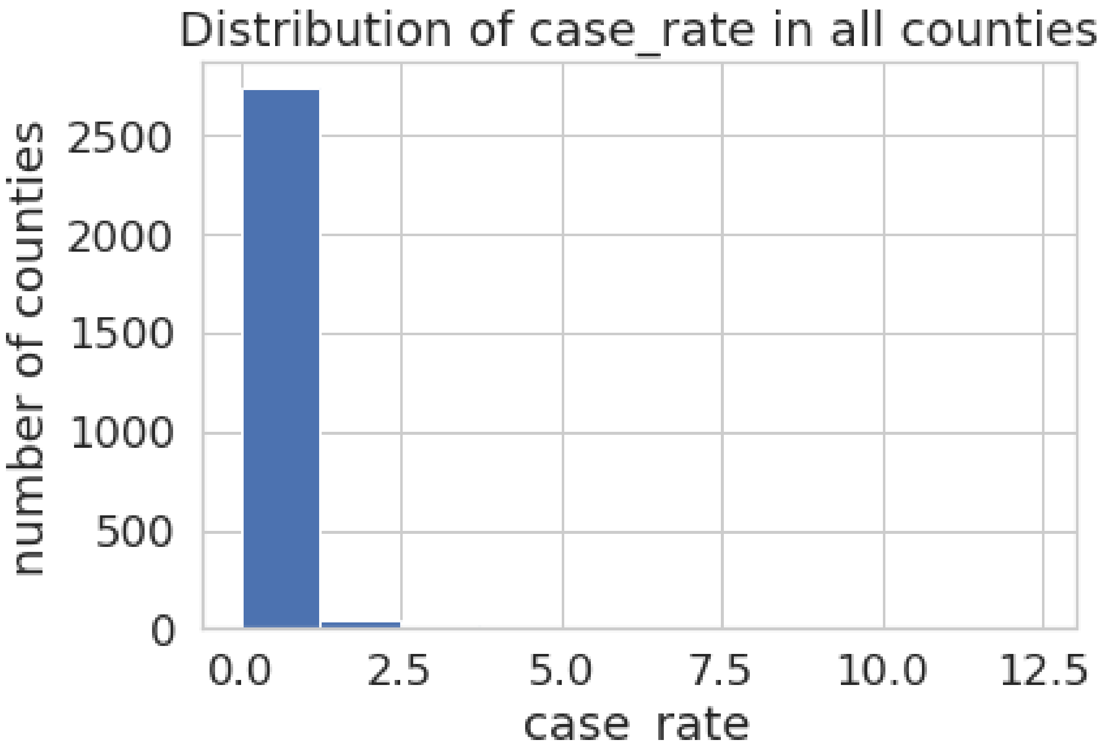
	CountyName	StateName	confirmed
1849	New York	NY	186123
601	Cook	IL	55470
1848	Nassau	NY	38434
1870	Suffolk	NY	37062
198	Los Angeles	CA	33211
1878	Westchester	NY	31472
2285	Philadelphia	PA	18537
1303	Wayne	MI	18274
1216	Middlesex	MA	17953
1773	Hudson	NJ	17677

1.2.4 Distribution of Case Rate

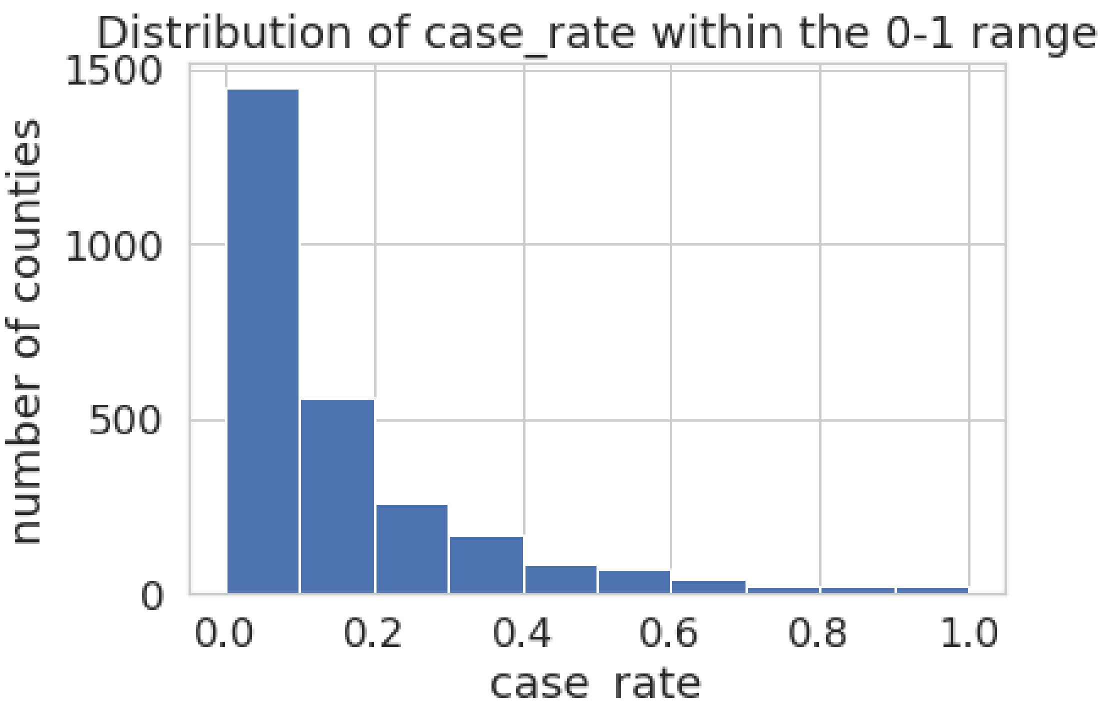
As we considered which factor to predict, we created some histograms to observe the distribution of `case_rate`.

[14]:
```
hist_cases = data.copy()
hist_cases = hist_cases[["case_rate"]]

hist_cases.hist()
plt.title("Distribution of case_rate in all counties")
plt.xlabel("case_rate")
plt.ylabel("number of counties")

hist_cases.hist(range = (0,1))
plt.title("Distribution of case_rate within the 0-1 range")
plt.xlabel("case_rate")
plt.ylabel("number of counties")

hist_cases.hist(range = (0, 0.2))
plt.title("Distribution of case_rate within the 0-0.2 range")
plt.xlabel("case_rate")
plt.ylabel("number of counties")

hist_cases.loc[hist_cases["case_rate"] > 1].shape[0]
```

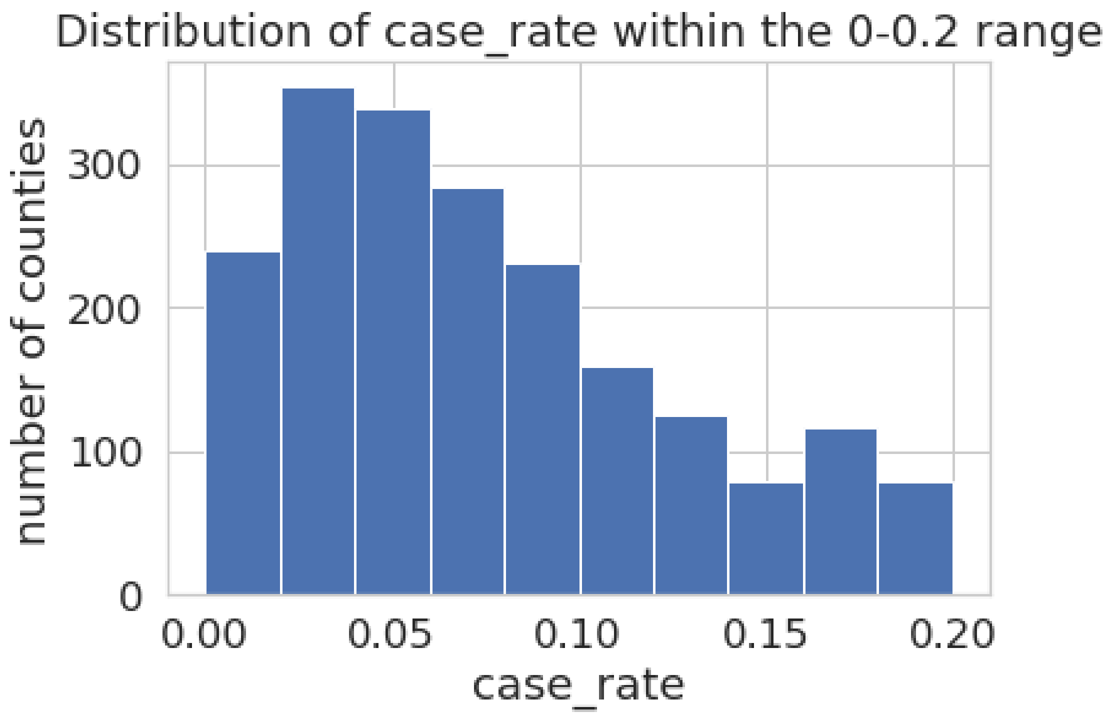
[14]: 110



Distribution of case_rate in all counties
number of counties
2500
2000
1500
1000
500
0
0.0
2.5
5.0
7.5
10.0
12.5
case_rate



Distribution of case_rate within the 0-1 range
number of counties
1500
1000
500
0
0.0
0.2
0.4
0.6
0.8
1.0
case_rate



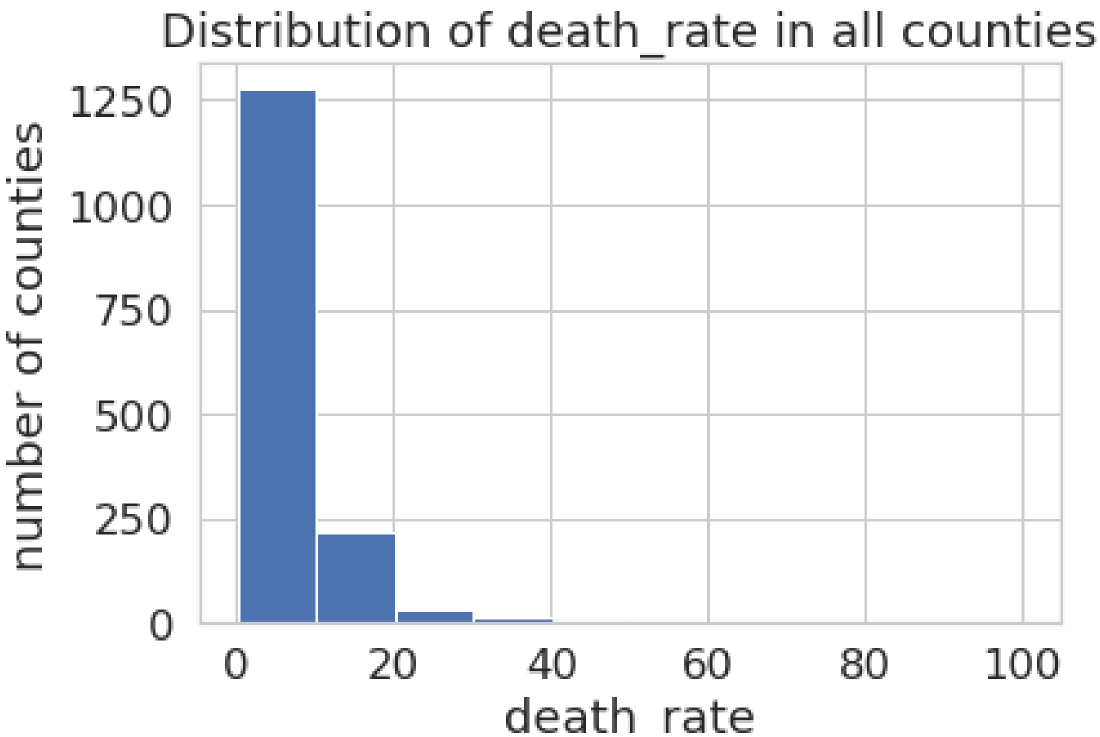
1.2.5 Distribution of Death Rate

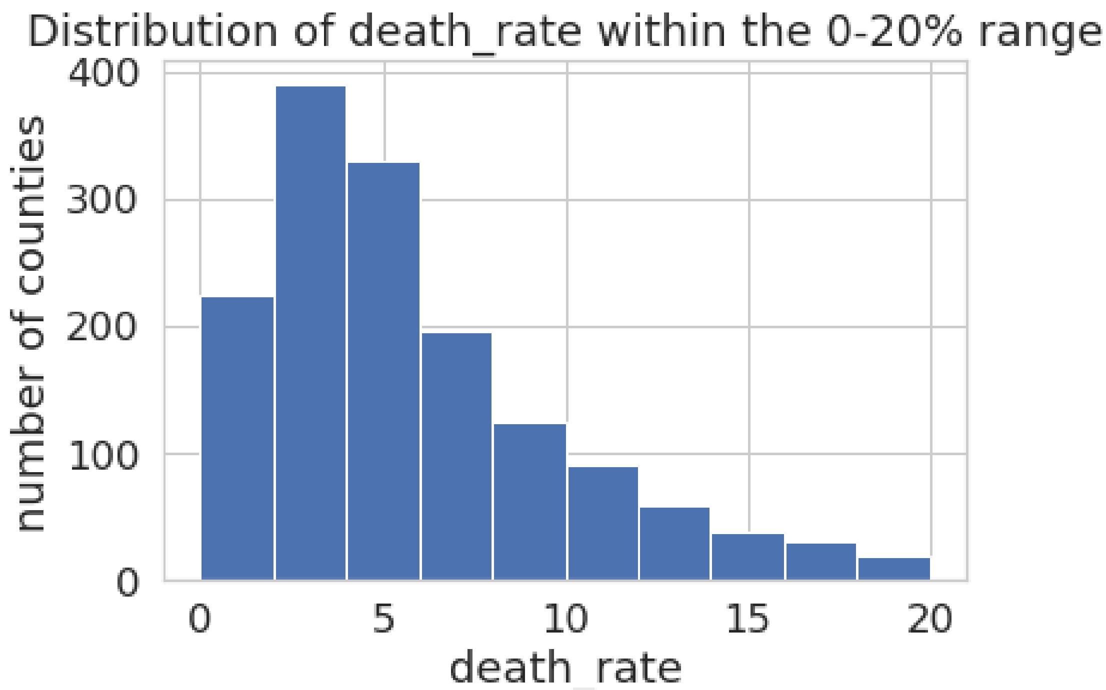
As we considered which factor to predict, we created some histograms to observe the distribution of `death_rate`.

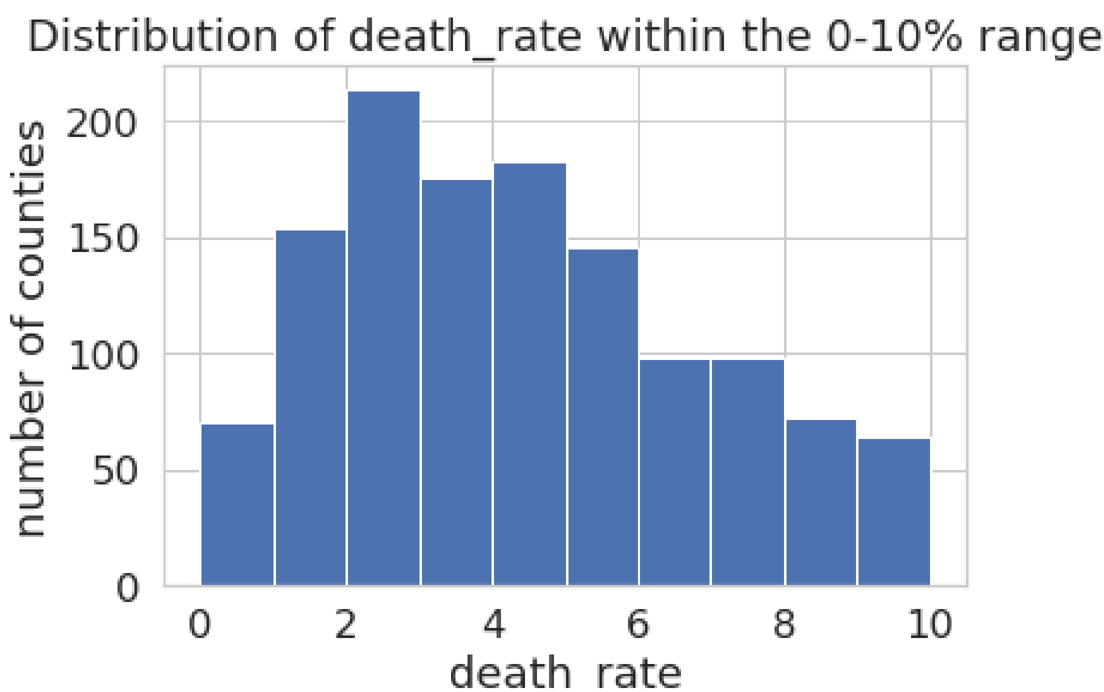
[15]:
```
hist_deaths = data.copy()
hist_deaths = hist_deaths[["death_rate"]]

#removed 0% death rates so we can look closely at the distribution of actual␣
↪death rates
hist_deaths = hist_deaths.loc[hist_deaths["death_rate"] != 0]

hist_deaths.hist()
plt.title("Distribution of death_rate in all counties")
plt.xlabel("death_rate")
plt.ylabel("number of counties")

hist_deaths.hist(range = (0,20))
plt.title("Distribution of death_rate within the 0-20% range")
plt.xlabel("death_rate")
plt.ylabel("number of counties")

hist_deaths.hist(range = (0, 10))
```

```
plt.title("Distribution of death_rate within the 0-10% range")
plt.xlabel("death_rate")
plt.ylabel("number of counties")

hist_deaths.loc[hist_deaths["death_rate"] > 20].shape[0]
```

[15]: 50




Distribution of death_rate within the 0-20% range
400
300
200
100
0
number of counties
0
5
10
15
20
death_rate


Distribution of death_rate within the 0-10% range
200
150
100
50
0
number of counties
0
2
4
6
8
10
death_rate

1.2.6 Geographically mapping counties that have any cases and any deaths

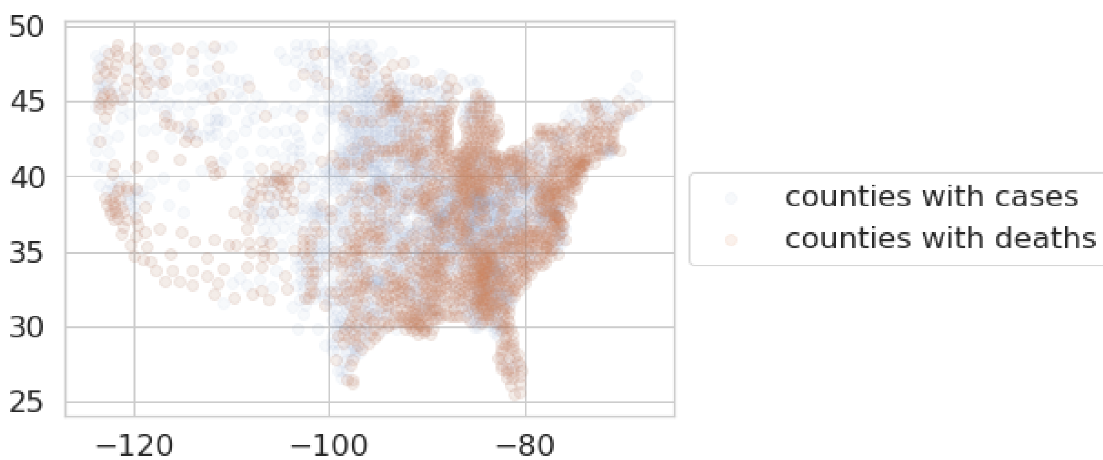
We noticed that there were many counties with no cases or no deaths, and we wanted to visually see where these counties were to see if there was a particular state or region that especially lacked cases or deaths.

We created a scatterplot based on latitude and longitude and plotted all counties with cases first and then all counties with deaths.

Points where you can just see the blue dot shows there are no deaths in the area. We also played with the alpha value so that the opacity was lower and you could see where cases or deaths were especially concentrated.

[16]:
```
maps = data.copy()
#plot all counties with cases
plt.scatter(maps["lon"], maps["lat"], alpha = 0.04)

#remove counties with no deaths and plot the ones with any deaths
maps = maps.loc[maps["deaths"] != 0]
plt.scatter(maps["lon"], maps["lat"], alpha = 0.1)

#add and re-position the legend
plt.legend(["counties with cases", "counties with deaths"], loc="center left",␣
 ↪bbox_to_anchor=(1, 0.5))
```

[16]: `<matplotlib.legend.Legend at 0x7f380c6f1450>`



1.3 Exploratory Data Analysis on Potential Features

These are features we ended up incorporating into our models.

1.3.1 Median Income and Death Rate

We first made a simple line plot between the median income and the death rate, and saw a general negative correlation between the two.

[17]:
```
income = data.copy()

#line plot (hover to show information)
fig = px.line(income, x = "med_income", y = "death_rate",␣
 ↪hover_name="CountyName")
fig.show()
```

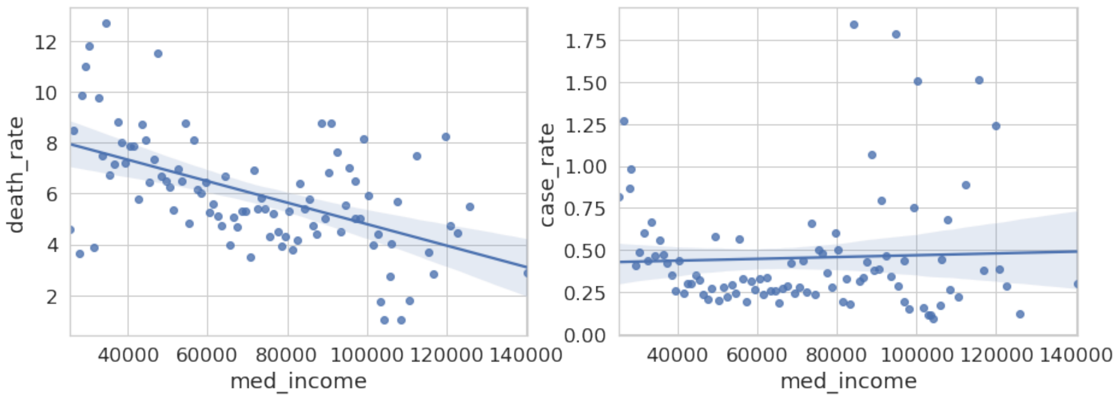
1.3.2 Death Rate and Case Rate for Grouped Median Income

We wanted to get a closer look at that trend, so we did some more thorough investigating. We then grouped the income by the thousands so we would have fewer data points and created a scatterplot with a best fit line to show the correlation even more.

Coupled with the case rate graph, the death rate vs median income graph shows that when income is higher, the death rate of the counties is lower because the case rate stays generally constant throughout all income levels.

[18]:
```
income = income.sort_values(["med_income"])

#add column to group income by the thousands
income["income (in k)"] = income["med_income"] // 1000
income = income[["med_income", "income (in k)", "death_rate", "case_rate"]]

#remove death_rate = 0 since that isn't representative
income = income.loc[income["death_rate"] != 0]

#group by income groups (of thousands)
incomegroup = income.groupby(["income (in k)"]).mean().
 ↪sort_values(["med_income"])

fig, ax = plt.subplots(1,2, figsize=(15,5))

#plot case_rate and death_rate
sns.regplot("med_income", "death_rate", data=incomegroup, ax=ax[0])
sns.regplot("med_income", "case_rate", data=incomegroup, ax=ax[1])
```

[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3808f28a90>
```



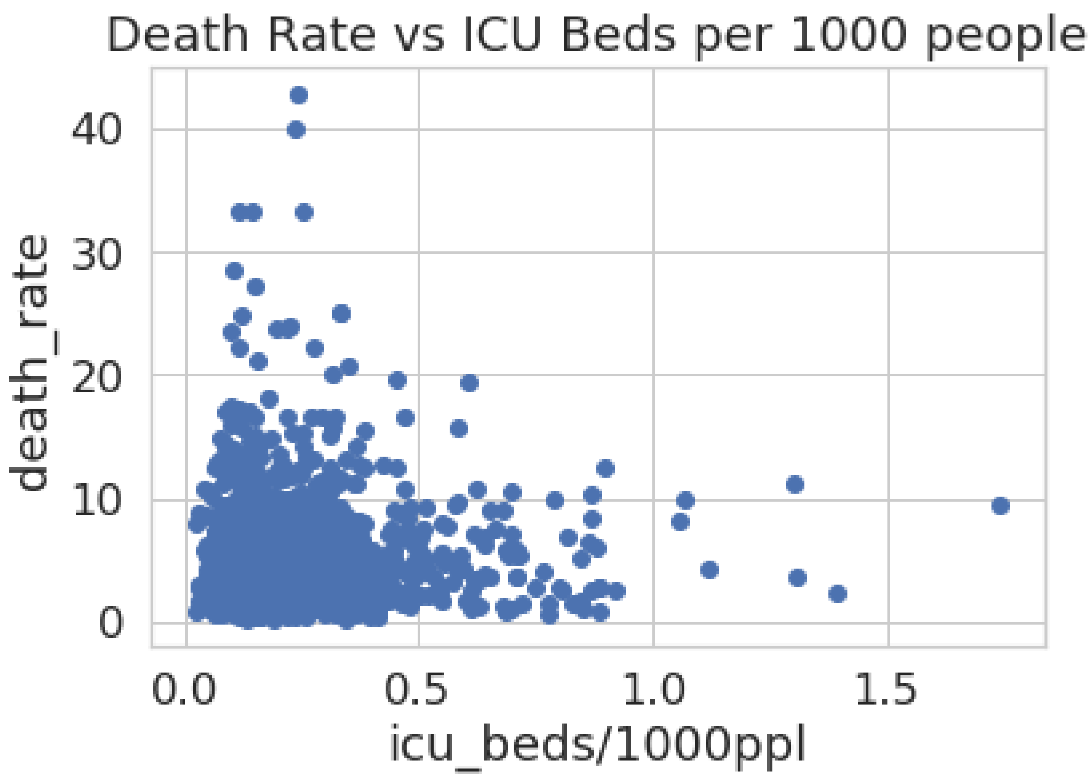
1.3.3 ICU_beds/1000ppl and Death_rate

We then wanted to see if there was a correlation between the number of ICU beds for every 1000 people and death rate, so we created a scatter plot.

The correlation wasn't that strong, so to use this, we'll have to add on more features.

[19]:
```
icu = data.copy()

icu = icu.loc[icu["death_rate"] != 100]
icu = icu.loc[icu["death_rate"] != 0]
icu = icu.loc[icu["icu_beds/1000ppl"] != 0]

plt.scatter("icu_beds/1000ppl", "death_rate", data=icu)
plt.xlabel("icu_beds/1000ppl")
plt.ylabel("death_rate")
plt.title("Death Rate vs ICU Beds per 1000 people")
```

[19]:
```
Text(0.5, 1.0, 'Death Rate vs ICU Beds per 1000 people')
```



1.3.4 Age and Death_Rate

We finally wanted to see if there was a correlation between the age factors and death rate, so we created another scatter plot.

This one shows a positive correlation, where counties with higher fractions of 65+ people have a higher death rate. The following shows a positive correlation (but less strong), where counties with higher median ages have a higher death rate.

[20]:
```
age = data.copy()

age = age.loc[age["death_rate"] != 100]
age = age.loc[age["death_rate"] != 0]

fig = px.scatter(age, x = "old", y = "death_rate", trendline = "ols")
fig.update_layout(
    title="Age vs Death Rate",
    xaxis_title="% of population with age 65+",
    yaxis_title="death rate",
    )
```

[21]:
```
fig2 = px.scatter(age, x = "MedianAge2010", y = "death_rate", trendline = "ols")
fig2.update_layout(
    title="Age vs Death Rate",
    xaxis_title="Median Age",
    yaxis_title="death rate",
    )
```

1.4 Modeling the Data

1.4.1 Data Cleaning

Before we build our model, we decided we needed to further clean our data so the model could be as accurate as possible. We first removed outliers: * We removed counties where the death rate was 100% (there were 2 counties where this was true, both had 1 case and 1 death). These were large outliers and not representative of the remaining data. * We then decided to remove counties where there were more than 10,000 cases since those were also not representative of the data. There were 20 counties where this was true (out of 2810 total now), so we decided the best course of action was to remove these. * The next decision removed a fairly large number of counties. We decided to remove all counties where the death rate was 0 (of which there were 1265). This was because the goal of the model was to predict death_rate based on various factors, and it didn't make sense for the model to predict a value of 0 since the factors were variable and there can't exist any "perfect" values for the factors that would make them have a 0% death rate.

Then, we cut the dataset down to any factors that we may choose to use later in our models so it's easier to read and process. We counted how many NaN values were in each of the chosen columns, and then seeing that they were a negligible amount (1 or 2 counties), we dropped all counties that had any NaN values in our chosen columns.

[22]:
```
print(data.shape[0])
data.columns
```

```
2812
```

[22]:
```
Index(['countyFIPS', 'STATEFP', 'COUNTYFP', 'CountyName', 'StateName', 'State',
       'lat', 'lon', 'POP_LATITUDE', 'POP_LONGITUDE',
       …
       'FIPStxt_y', 'unemploy_rate', 'med_income', 'case_rate', 'death_rate',
       'old', 'inMedicare', 'medicare_rate', 'hospitals/1000ppl',
       'icu_beds/1000ppl'],
      dtype='object', length=101)
```

[23]:
```
traintest = data.copy()

#remove outliers
traintest.loc[traintest["death_rate"] == 100].shape[0] #there are 2 counties␣
↪where this is true
traintest = traintest.loc[traintest["death_rate"] != 100]
```

```
traintest.loc[traintest["confirmed"] > 10000].shape[0] #there are 20 of these␣
↪outliers with significantly more cases
traintest = traintest.loc[traintest["confirmed"] < 10000]

traintest = traintest.loc[traintest["deaths"] > 0.00]

traintest = traintest.rename(columns={"PopulationDensityperSqMile2010":
↪"density"})

traintest = traintest[["confirmed",
                 "deaths",
                 "death_rate",
                 "case_rate",
                "old",
                   "MedianAge2010",
                "density",
                "inMedicare",
                "medicare_rate",
               "pov_pct",
               "unemploy_rate",
               "med_income",
               "lat",
               "lon",
                "FracMale2017",
                "DiabetesPercentage",
                "HeartDiseaseMortality",
                "StrokeMortality",
                "Smokers_Percentage",
                "RespMortalityRate2014",
                "hospitals/1000ppl",
                "icu_beds/1000ppl",
                "SVIPercentile",
                "TotalM.D.'s,TotNon-FedandFed2017",
                   "Rural-UrbanContinuumCode2013",
                "#FTEHospitalTotal2017"]]
#print(traintest.isna().sum())
traintest = traintest.dropna()

traintest = (traintest.loc[traintest["confirmed"] > 5])
traintest.loc[traintest["confirmed"] > 5].shape[0]
```

[23]: 1489

1.4.2 Train-Test Split

Next we split our cleaned dataset into a training data set and a testing data set using sklearn. We put 85% of our data into the training set and 15% into the testing set. We picked a random_state so that the split was pseudorandom.

[24]:
```
train, test = train_test_split(traintest, test_size=0.15, random_state=42)
train_c = train
test_c = test
```

[25]:
```
# from sklearn import linear_model as lm
train_c
```

[25]:
```
      confirmed  deaths  death_rate  case_rate        old  MedianAge2010  \
1765       1558      79    5.070603   0.586974  17.595289           39.9
2116        213      13    6.103286   0.091742  13.942620           37.8
2426         22       1    4.545455   0.078515  20.089222           42.0
2072         39       1    2.564103   0.066662  16.747573           38.4
813           6       1   16.666667   0.038069  21.686441           43.3
…           …     …         …         …         …            …
2240       3453     172    4.981176   0.821845  16.849140           39.1
2617        203       2    0.985222   0.091182  10.239365           30.4
1757         44       2    4.545455   0.057522  19.179533           40.7
3047         34       1    2.941176   0.059277  17.280937           41.0
2236       1526     127    8.322412   0.125241  18.515707           41.3

      density  inMedicare  medicare_rate  pov_pct  …  HeartDiseaseMortality  \
1765    494.1   77.058475      22.260190     12.8  …                  203.7
2116    530.0   80.931568      16.949861      5.2  …                  155.9
2426     47.6   73.813112      28.415418     18.6  …                  315.7
2072    121.3   77.136540      22.420689     11.1  …                  174.5
813      32.6   81.084427      25.626547     11.4  …                  162.4
…         …         …             …      …  …                    …
2240    480.4   79.603396      20.716074     11.4  …                  171.2
2617    231.7   75.262671      13.594693     13.2  …                  131.2
1757    109.1   75.991409      24.955225     10.2  …                  148.0
3047     74.2   78.829787      22.943617      7.6  …                  163.2
2236   1675.7   82.062767      22.006201     11.7  …                  185.8

      StrokeMortality  Smokers_Percentage  RespMortalityRate2014  \
1765             31.8           15.644195                  50.72
2116             36.8           15.228085                  57.85
2426             57.6           22.895309                  61.91
2072             35.1           19.307126                  73.64
813              30.4           15.947325                  51.87
…                 …                 …                    …
2240             50.8           16.583595                  47.62
```

```
2617             36.2           14.147185                    43.77
1757             30.9           16.624635                    64.46
3047             32.1           14.981969                    58.82
2236             35.7           16.991039                    48.80

      hospitals/1000ppl  icu_beds/1000ppl  SVIPercentile  \
1765           0.007535          0.241119         0.7911
2116           0.004307          0.245507         0.0478
2426           0.035689          0.214133         0.7443
2072           0.034186          0.205114         0.4325
813            0.063448          0.000000         0.3596
…                   …                 …            …
2240           0.007140          0.168986         0.6162
2617           0.013475          0.188653         0.4924
1757           0.013073          0.352973         0.1417
3047           0.034869          0.174344         0.1204
2236           0.012311          0.476014         0.2506

      TotalM.D.'s,TotNon-FedandFed2017  Rural-UrbanContinuumCode2013  \
1765                             779.0                           2.0
2116                             651.0                           1.0
2426                              31.0                           6.0
2072                              60.0                           4.0
813                                9.0                           7.0
…                                  …                         …
2240                             890.0                           2.0
2617                             296.0                           1.0
1757                             215.0                           4.0
3047                              59.0                           2.0
2236                            8995.0                           1.0

      #FTEHospitalTotal2017
1765                 6125.0
2116                 1455.0
2426                  283.0
2072                  937.0
813                   173.0
…                      …
2240                 8214.0
2617                 1399.0
1757                 1260.0
3047                  806.0
2236                37225.0

[1265 rows x 26 columns]
```

[26]:
```
test_c
```

```
[26]:       confirmed  deaths  death_rate  case_rate        old  MedianAge2010  \
      1896         35       3    8.571429   0.050342  24.017605           45.8
      483          50       1    2.000000   0.125247  14.155457           36.2
      447        2060      35    1.699029   1.019055  14.447336           34.5
      1906        399      11    2.756892   0.120061  11.769326           31.0
      2111        413      36    8.716707   0.207927  21.151203           42.8
      …           …      …         …          …         …             …
      380          20       1    5.000000   0.241051  14.318428           33.4
      1846       1890     156    8.253968   0.254554  16.835068           38.5
      38           99       2    2.020202   0.107158  19.603407           40.4
      96           22       1    4.545455   0.104275  38.406484           53.9
      2341         59       1    1.694915   0.223502  17.838473           38.8

            density  inMedicare  medicare_rate  pov_pct  …  HeartDiseaseMortality  \
      1896    131.3   79.446079      28.355676      9.8  …                  164.1
      483     115.0   68.404160      19.511034     15.2  …                  210.5
      447     457.5   78.287274      18.490908     13.2  …                  145.8
      1906    489.7   69.597926      16.479403     17.0  …                  188.2
      2111    340.2   79.151284      26.456625     17.6  …                  213.2
      …         …          …             …       …   …                    …
      380      24.7   71.045392      17.524406     26.1  …                  219.3
      1846   1132.6   78.660949      21.450179     14.4  …                  149.3
      38      138.9   75.752346      24.457987     14.0  …                  219.7
      96        4.6   77.438727      29.201820     23.7  …                  182.7
      2341     60.3   70.396666      24.543526     30.0  …                  368.7

            StrokeMortality  Smokers_Percentage  RespMortalityRate2014  \
      1896             40.5           15.646474                  52.63
      483              48.5           17.836136                  98.05
      447              39.9           15.636843                  61.37
      1906             42.5           18.090857                  68.59
      2111             43.0           19.676253                  63.93
      …                 …                  …                    …
      380              49.7           21.822201                 116.44
      1846             32.3           15.151442                  37.67
      38               49.9           16.656004                  75.94
      96               31.4           15.521010                  45.08
      2341             63.8           21.379465                  92.22

            hospitals/1000ppl  icu_beds/1000ppl  SVIPercentile  \
      1896           0.014384          0.115068         0.4255
      483            0.025049          0.000000         0.7134
      447            0.004947          0.420484         0.6618
      1906           0.003009          0.144435         0.8640
      2111           0.010069          0.211452         0.4640
      …                   …                 …              …
      380            0.000000          0.000000         0.9615
```

```
1846          0.006734           0.257248          0.5204
38            0.010824           0.519554          0.4468
96            0.094796           0.142194          0.9236
2341          0.000000           0.000000          0.9869

      TotalM.D.'s,TotNon-FedandFed2017  Rural-UrbanContinuumCode2013  \
1896                             212.0                           4.0
483                               13.0                           3.0
447                              475.0                           3.0
1906                             851.0                           2.0
2111                             290.0                           2.0
…                                …                            …
380                                1.0                           9.0
1846                            4297.0                           1.0
38                               195.0                           3.0
96                                16.0                           6.0
2341                              15.0                           6.0

      #FTEHospitalTotal2017
1896                  903.0
483                   214.0
447                  4546.0
1906                 8580.0
2111                 2707.0
…                      …
380                     0.0
1846                23546.0
38                   1250.0
96                    947.0
2341                    0.0

[224 rows x 26 columns]
```

1.4.3 Defining Functions and Setting Up for the Models

Here, we defined a function to calculate the root mean squared error (RMSE) given actual and predicted values. This was created while referencing the function defined in lecture.

We also initialized a dictionary to hold the models so we can compare them later, and arrays to hold the RMSEs for training data, cross validation, and testing data so we could compare them at the end. This was also created in reference to the Cross Validation lecture.

Since we are predicting death_rate that variable is set (and this structure makes it easy to switch to case_rate or any other factor we want to predict!).

We also find the range of the death_rates in our data so we can contextualize our RMSE later.

[27]:
```
def rmse(actual, predicted):
    return np.sqrt(np.mean((actual - predicted)**2))
```

[28]:
```
def standardize(data):
    return (data - np.mean(data)) / np.std(data)
```

[114]:
```
models = {}
training_rmse = []
validate_rmse = []
test_rmse = []
```

[30]:
```
predicting = "death_rate"

print(min(train_c[predicting]), " - ", max(train_c[predicting]))
print(np.mean(train_c[predicting]))

standard = standardize(train_c[predicting])
print(min(standard), " - ", max(standard))
```

```
0.15384615384615385  -  42.857142857142854
6.01213217346454
-1.2663776434065634  -  7.964735358538478
```

1.5 Feature Engineering

We started the process of evaluating what features are best by creating different pairplots to compare death_rate and case_rate to certain factors. Below is the pairplot we created for some economic factors.

(Note: we created several different pairplots but including all of them caused my kernel to crash, so we're only showing one below)

[31]:
```
#sns.pairplot(train_c[["death_rate", "case_rate", "pov_pct", "unemploy_rate",␣
↪"med_income"]])
```

1.5.1 General Health Features Model

In terms of general health, the factors we took into account were: * the percentage of "old" people in that county (over 65 years old) * the median age of people in the county * the percentage of the population who has diabetes * the number of heart disease mortalities (per 100,000 people) per year of that county (from 2014-2016) * the number of stroke mortalities (per 100,000 people) per year of that county (from 2014-2016) * the percentage of the population who are smokers * the respiratory mortality rate (per 100,000 people) from the year 2014.

[115]:
```
def health_cols(data):
    return data[[predicting,
```

```
                    "old",
                   "MedianAge2010",
                   "DiabetesPercentage",
                    "HeartDiseaseMortality",
                  "StrokeMortality",
                  "Smokers_Percentage",
                  "RespMortalityRate2014"
                  ]]
```

We create the `X_train` and `y_train` variables for the model, and then create the Linear Regression model that we're using.

[116]:
```
X_train_h = health_cols(train_c).drop([predicting], axis = 1)
y_train = train_c[predicting]

regr = lm.LinearRegression()

X_train_h
```

[116]:
```
            old  MedianAge2010  DiabetesPercentage  HeartDiseaseMortality  \
1765  17.595289           39.9                10.4                  203.7
2116  13.942620           37.8                 9.5                  155.9
2426  20.089222           42.0                 9.9                  315.7
2072  16.747573           38.4                11.1                  174.5
813   21.686441           43.3                 6.3                  162.4
…          …              …                  …                     …
2240  16.849140           39.1                11.2                  171.2
2617  10.239365           30.4                 8.5                  131.2
1757  19.179533           40.7                 7.1                  148.0
3047  17.280937           41.0                 9.4                  163.2
2236  18.515707           41.3                 8.1                  185.8

      StrokeMortality  Smokers_Percentage  RespMortalityRate2014
1765             31.8           15.644195                  50.72
2116             36.8           15.228085                  57.85
2426             57.6           22.895309                  61.91
2072             35.1           19.307126                  73.64
813              30.4           15.947325                  51.87
…                 …                 …                      …
2240             50.8           16.583595                  47.62
2617             36.2           14.147185                  43.77
1757             30.9           16.624635                  64.46
3047             32.1           14.981969                  58.82
2236             35.7           16.991039                  48.80

[1265 rows x 7 columns]
```

In the following cells, we fit the model and use it to predict values based on X_train. We also use the previously defined **rmse** function to calculate the training error. We also use the score function built into models, although we mostly used the RMSE to determine and score the model's performance.

[117]:
```
regr.fit(X_train_h, y_train)
y_fitted = regr.predict(X_train_h)
```

[118]:
```
models["health"] = regr
```

[119]:
```
training_error = rmse(y_fitted, y_train)
training_rmse.append(training_error)
training_error
```

[119]:
```
4.429037715031477
```

[120]:
```
print(regr.score(X_train_h, y_train))
```

```
0.08334885794761948
```

Next, we perform cross validation on this model using the **cross_val_score** method from sklearn. Instead of the default scoring, we used one of the built-in scoring methods which was "neg_root_mean_squared_error". Since we didn't actually want the value to be negative we then make them positive again. We then take the mean of these 5 cross validated RMSEs and will use that later to compare.

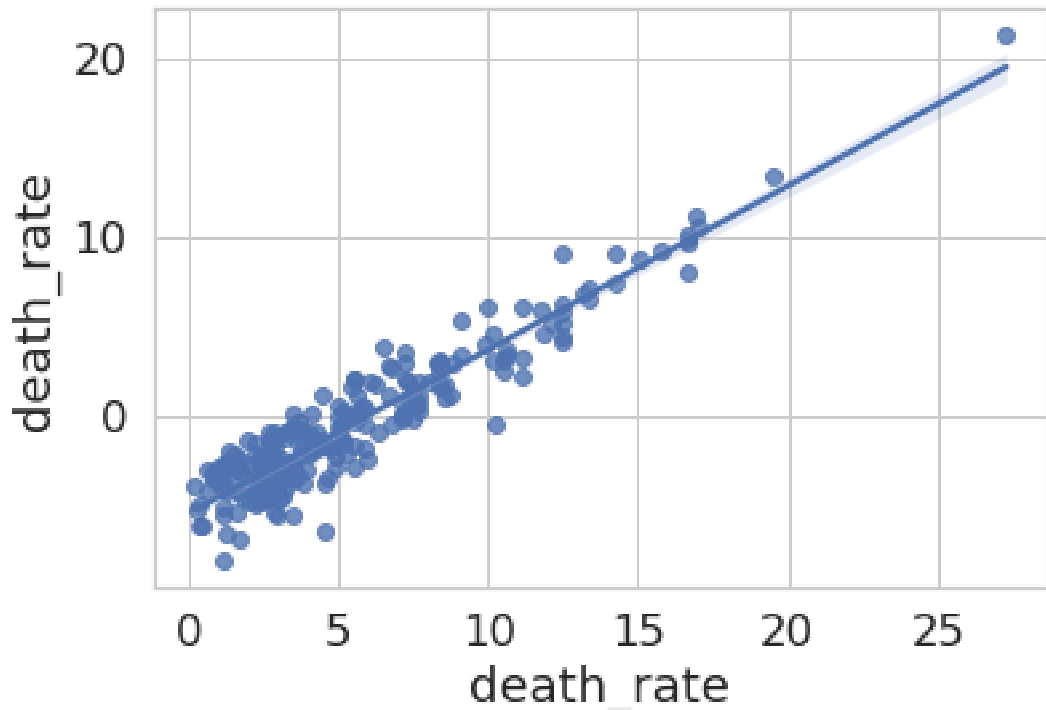
[121]:
```
c_scores = cross_val_score(regr, X_train_h, y_train, cv=5,␣
 ↪scoring="neg_root_mean_squared_error")
c_scores = c_scores * -1
c_scores.mean()
validate_rmse.append(c_scores.mean())
c_scores
```

[121]:
```
array([4.16004279, 5.03249353, 4.18326695, 4.8132614 , 4.14195952])
```

Finally, we predict our values for our testing data set, and calculate the testing RMSE. We also find the residuals and plot them against the actual values (this was referenced from part homeworks and labs).

[122]:
```
X_test_h = health_cols(test_c).drop([predicting], axis = 1)
y_test = test_c[predicting]

y_predicted = regr.predict(X_test_h)
residuals = y_test - y_predicted
ax = sns.regplot(y_test, residuals)

test_error = rmse(y_predicted, y_test)
test_rmse.append(test_error)
test_error
```

[122]: 4.165044006828287



1.5.2 Healthcare System Features Model

In terms of the healthcare system, the factors we took into account were: * the number of hospitals per 1,000 people in the county * the number of ICU beds per 1,000 people in the county * the percent of people enrolled in Medicare who are eligible for Medicare * the number of MDs in each county in 2017 * the number of full-time employees at hospitals in 2017 in the county
* the rural-urban continuum code that determines how rural or urban a county is on a scale from 1 to 9

[123]:
```
def healthcare_cols(data):
    return data[[predicting,
                  "hospitals/1000ppl",
                 "icu_beds/1000ppl",
                 "inMedicare",
                 "TotalM.D.'s,TotNon-FedandFed2017",
                 "#FTEHospitalTotal2017",
                  "Rural-UrbanContinuumCode2013"
                 ]]
```

The model training, error finding, cross validation, and testing process mirrors the one above for the general health features.

[124]:
```
X_train_hc = healthcare_cols(train_c).drop([predicting], axis = 1)
y_train = train_c[predicting]

regr2 = lm.LinearRegression()
```

[125]:
```
regr2.fit(X_train_hc, y_train)
y_fitted = regr2.predict(X_train_hc)
```

[126]:
```
models["healthcare"] = regr2
```

[127]:
```
training_error = rmse(y_fitted, y_train)
training_rmse.append(training_error)
training_error
```

[127]:
```
4.494261786979374
```

[128]:
```
print(regr2.score(X_train_hc, y_train))
```

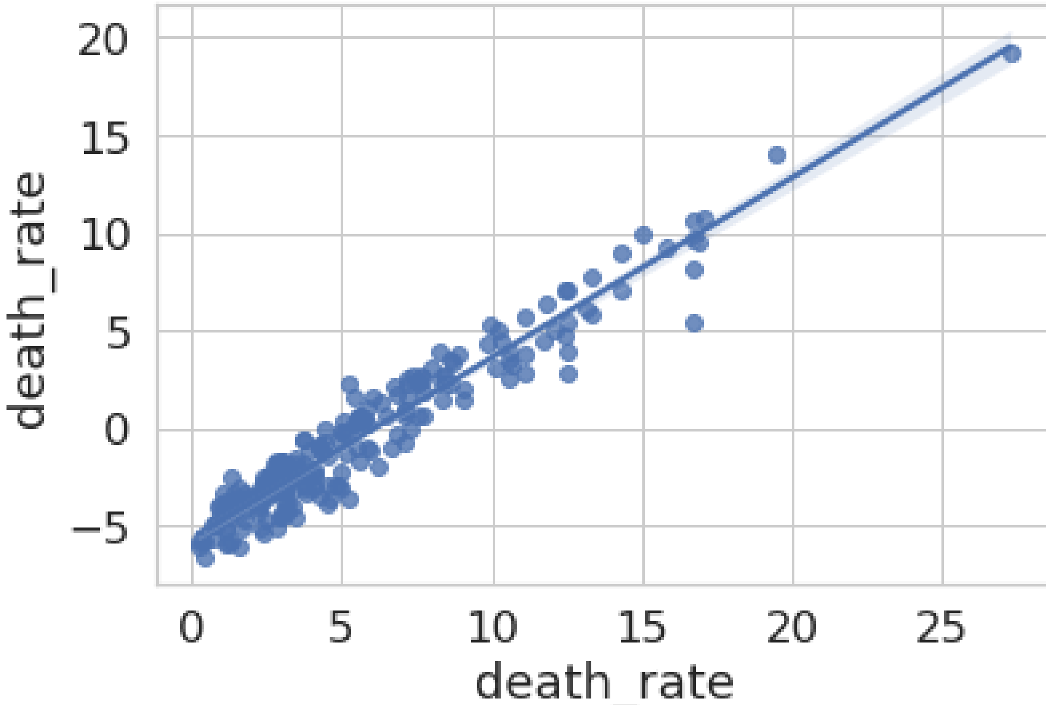
```
0.056152001736684325
```

[129]:
```
c_scores = cross_val_score(regr2, X_train_hc, y_train, cv=5,␣
↪scoring="neg_root_mean_squared_error")
c_scores = c_scores * -1
c_scores.mean()
validate_rmse.append(c_scores.mean())
c_scores
```

[129]:
```
array([4.27377157, 5.04418444, 4.28248583, 4.84825898, 4.12799525])
```

[130]:
```
X_test_hc = healthcare_cols(test_c).drop([predicting], axis = 1)
y_test = test_c[predicting]

y_predicted = regr2.predict(X_test_hc)
residuals = y_test - y_predicted
ax = sns.regplot(y_test, residuals)

test_error = rmse(y_predicted, y_test)
test_rmse.append(test_error)

test_error
```

[130]:
```
4.101943608331466
```



1.5.3 Economic State Features Model

In terms of the economic state, the factors we took into account were: * median income of the county * the unemployment rate of the county * the percentage of poverty in the county * the percentage of people eligible for Medicare in the county * the county's overall percentile ranking indicating the CDC's Social Vulnerability Index (SVI)

[131]:
```
def econ_cols(data):
    return data[[predicting,
                   "SVIPercentile",
                 "medicare_rate",
                  "pov_pct",
                  "unemploy_rate",
                  "med_income"
                 ]]
```

The model training, error finding, cross validation, and testing process mirrors the one above for the general health features.

[132]:
```
X_train_e = econ_cols(train_c).drop([predicting], axis = 1)
y_train = train_c[predicting]

regr3 = lm.LinearRegression()
```

```
[133]: regr3.fit(X_train_e, y_train)
       y_fitted = regr3.predict(X_train_e)
```

```
[134]: models["economic"] = regr3
```

```
[135]: training_error = rmse(y_fitted, y_train)
       training_rmse.append(training_error)
       training_error
```

```
[135]: 4.4769016717887125
```

```
[136]: print(regr3.score(X_train_e, y_train))
```

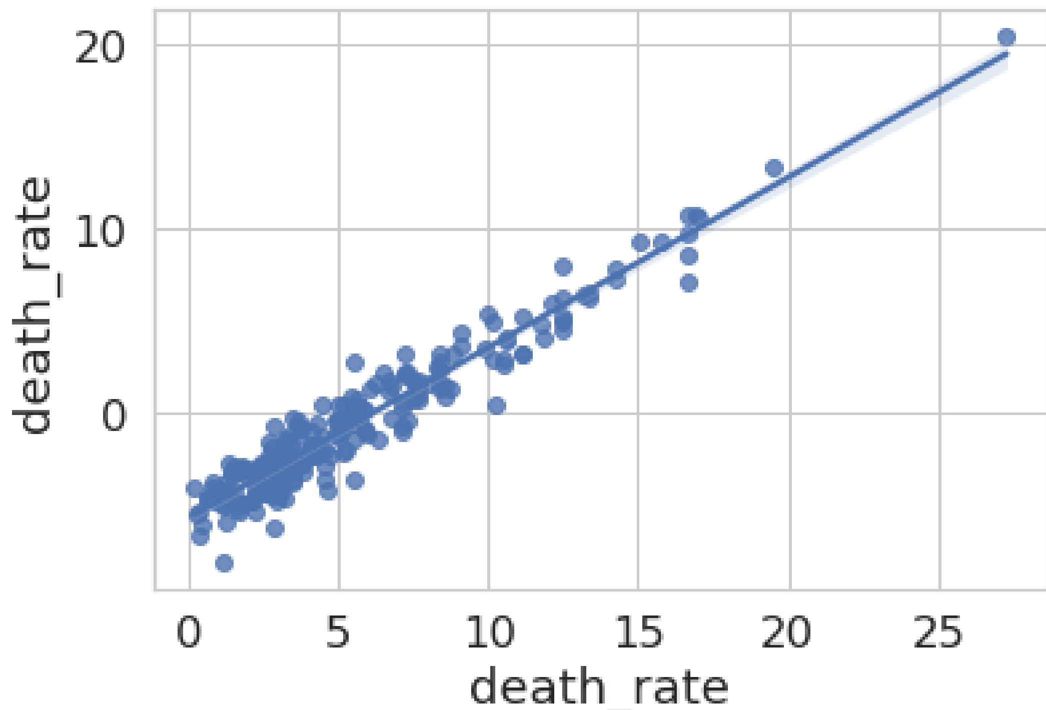
```
0.06342957689439865
```

```
[137]: c_scores = cross_val_score(regr3, X_train_e, y_train, cv=5,␣
        ↪scoring="neg_root_mean_squared_error")
       c_scores = c_scores * -1
       c_scores.mean()
       validate_rmse.append(c_scores.mean())
       c_scores
```

```
[137]: array([4.15907729, 5.1289684 , 4.23665803, 4.81163882, 4.10209386])
```

```
[138]: X_test_e = econ_cols(test_c).drop([predicting], axis = 1)
       y_test = test_c[predicting]

       y_predicted = regr3.predict(X_test_e)
       residuals = y_test - y_predicted
       ax = sns.regplot(y_test, residuals)

       test_error = rmse(y_predicted, y_test)
       test_rmse.append(test_error)

       test_error
```

```
[138]: 4.114396933982724
```



1.5.4 All Features Model

Here we combined all the features from the previous 3 models.

[139]:
```
def all_cols(data):
    return data[[predicting,
                 "old",
                 "DiabetesPercentage",
                  "HeartDiseaseMortality",
                "StrokeMortality",
                "Smokers_Percentage",
                "RespMortalityRate2014",
                 "hospitals/1000ppl",
                "icu_beds/1000ppl",
                "inMedicare",
                "TotalM.D.'s,TotNon-FedandFed2017",
                "#FTEHospitalTotal2017",
                 "Rural-UrbanContinuumCode2013",
                  "SVIPercentile",
                "medicare_rate",
                 "pov_pct",
                 "unemploy_rate",
                 "med_income"
```

```
                    ]]
```

The model training, error finding, cross validation, and testing process mirrors the one above for the general health features.

[140]:
```
X_train_a = all_cols(train_c).drop([predicting], axis = 1)
y_train = train_c[predicting]

regr4 = lm.LinearRegression()
```

[141]:
```
regr4.fit(X_train_a, y_train)
y_fitted = regr4.predict(X_train_a)
```

[142]:
```
models["all"] = regr4
```

[143]:
```
training_error = rmse(y_fitted, y_train)
training_rmse.append(training_error)
training_error
```

[143]:
```
4.368192572487627
```

[144]:
```
print(regr4.score(X_train_a, y_train))
```

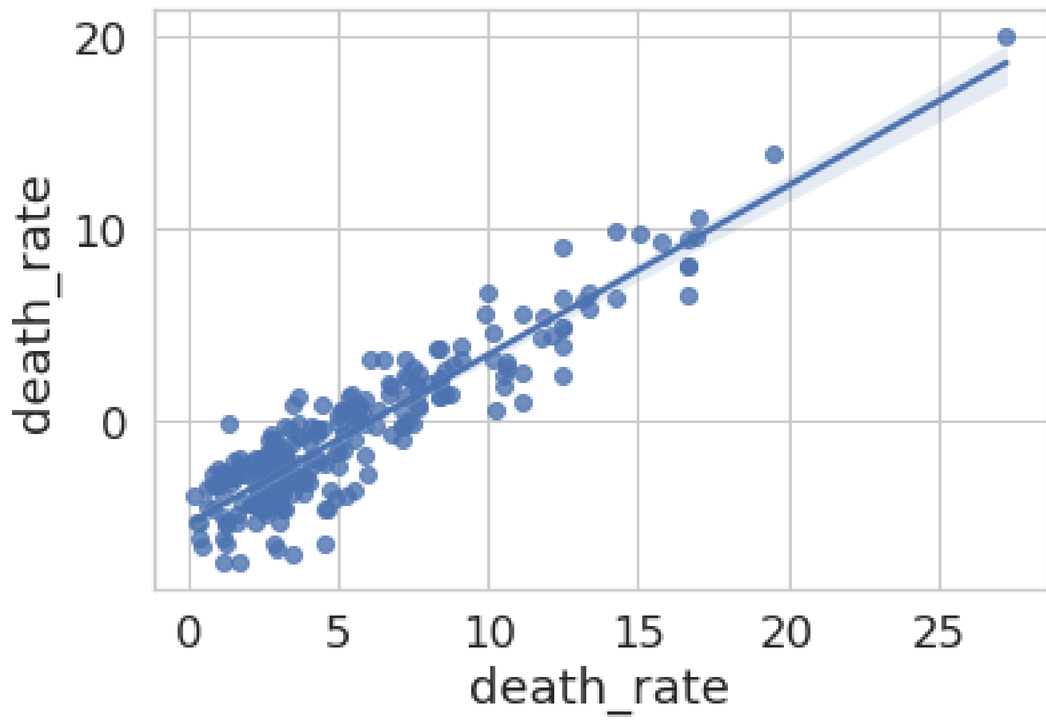
```
0.10836136364220872
```

[145]:
```
c_scores = cross_val_score(regr4, X_train_a, y_train, cv=5,␣
 ↪scoring="neg_root_mean_squared_error")
c_scores = c_scores * -1
c_scores.mean()
validate_rmse.append(c_scores.mean())
c_scores
```

[145]:
```
array([4.23722201, 4.94783012, 4.21891976, 4.72464312, 4.10778179])
```

[146]:
```
X_test_a = all_cols(test_c).drop([predicting], axis = 1)
y_test = test_c[predicting]

y_predicted = regr4.predict(X_test_a)
residuals = y_test - y_predicted
ax = sns.regplot(y_test, residuals)

test_error = rmse(y_predicted, y_test)
test_rmse.append(test_error)

test_error
```

[146]:
```
4.096416655290851
```



1.5.5 Comparing Models

Here we compare the previous 4 models based on the training rmse, the cross validation rmse, and the testing rmse. This process was created by referencing lecture code.

[147]:
```
#adapted from lecture code

def compare_models(models):
    names = list(models.keys())
    fig = go.Figure([
        go.Bar(x = names, y = training_rmse, name="Training RMSE"),
        go.Bar(x = names, y = validate_rmse, name="CV RMSE"),
        go.Bar(x = names, y = test_rmse, name="Test RMSE", opacity=.3)])
    return fig

training_rmse
```

[147]:
```
[4.429037715031477, 4.494261786979374, 4.4769016717887125, 4.368192572487627]
```

[150]:
```
fig = compare_models(models)
fig.update_yaxes(range=[4.05,4.52], title="RMSE")
```

[ ]:
```
```

1.6 Appendix

As a note, we did not think of making an appendix at the end until we saw this on Piazza, so not all our ideas and original failed tests and models and features are in this appendix. We only added old models from the last day of us working on it, so this is not all-encompassing

1.6.1 Baseline Features Model

[66]:
```
def no_cols(data):
    return data[[predicting,
                 "lat"
                ]]
```

[67]:
```
X_train_n = no_cols(train_c).drop([predicting], axis = 1)
y_train = train_c[predicting]

regr0 = lm.LinearRegression()
```

[68]:
```
regr0.fit(X_train_n, y_train)
y_fitted = regr0.predict(X_train_n)
```

[69]:
```
models["none"] = regr0
```

[70]:
```
training_error = rmse(y_fitted, y_train)
training_rmse.append(training_error)
training_error
```

[70]: 4.6197655238922515

[71]:
```
print(regr0.score(X_train_n, y_train))
```

0.002701435567487143

[72]:
```
c_scores = cross_val_score(regr0, X_train_n, y_train, cv=5,␣
↪scoring="neg_root_mean_squared_error")
c_scores = c_scores * -1
print(c_scores.mean())
validate_rmse.append(c_scores.mean())
c_scores
```

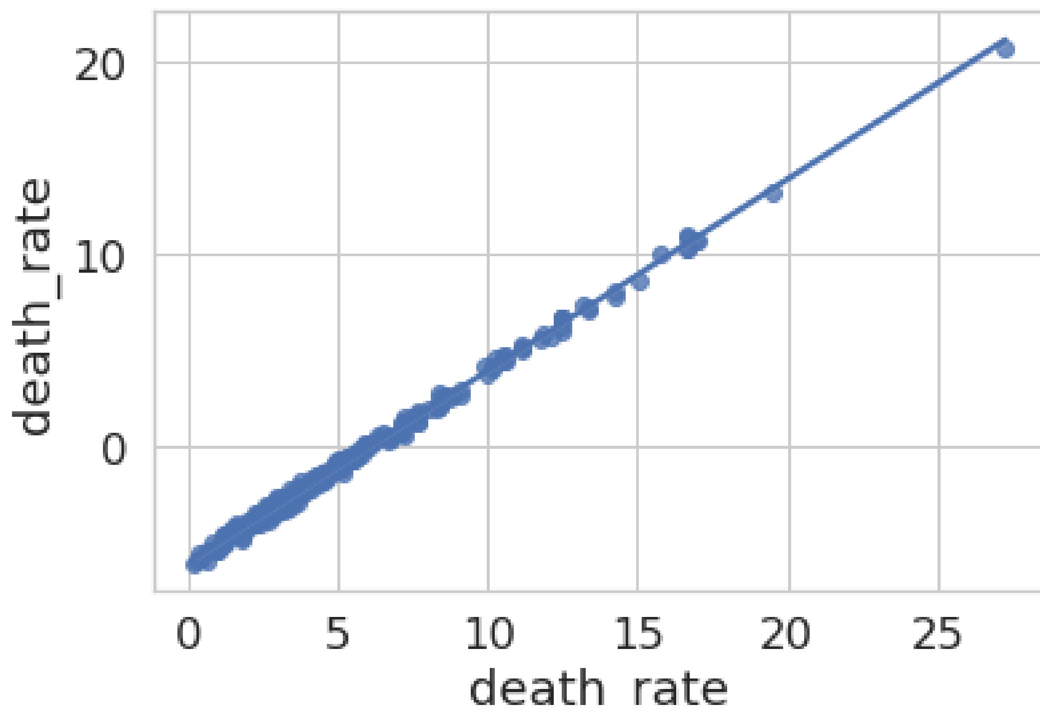
4.632113297626638

[72]: array([4.38060417, 5.2380538 , 4.29117572, 4.94626447, 4.30446833])

[test data below]

[73]:
```
X_test_n = no_cols(test_c).drop([predicting], axis = 1)
y_test = test_c[predicting]

y_predicted = regr0.predict(X_test_n)
residuals = y_test - y_predicted
ax = sns.regplot(y_test, residuals)

test_error = rmse(y_predicted, y_test)
test_rmse.append(test_error)

test_error
```

[73]: 4.274476033190996



1.6.2 Attempt at Standardizing Model

[74]:
```
stda = preprocessing.scale(X_train_h)
stda
```

[74]:
```
array([[ 0.04333002,  0.23610179, -0.10396778, …, -1.06576233,
        -0.57728857, -0.77990472],
       [-0.84730103, -0.25744905, -0.34697191, …, -0.48510175,
```

```
 -0.70850521, -0.3204902 ],
[ 0.65142622,  0.72965263, -0.23897007, …,  1.93044627,
  1.70929051, -0.0588881 ],
…,
[ 0.4296165 ,  0.42412116, -0.99498293, …, -1.17028124,
 -0.26811468,  0.10541863],
[-0.03331847,  0.49462842, -0.37397237, …, -1.0309227 ,
 -0.78611586, -0.25798921],
[ 0.26775567,  0.56513569, -0.72497834, …, -0.61284708,
 -0.15257239, -0.90361803]])
```