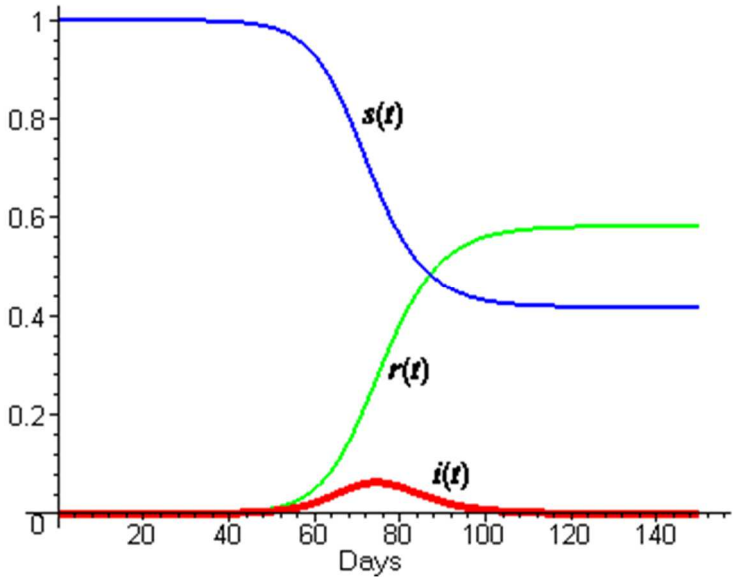
**State-Level Covid-19 Dynamics**

Introduction:

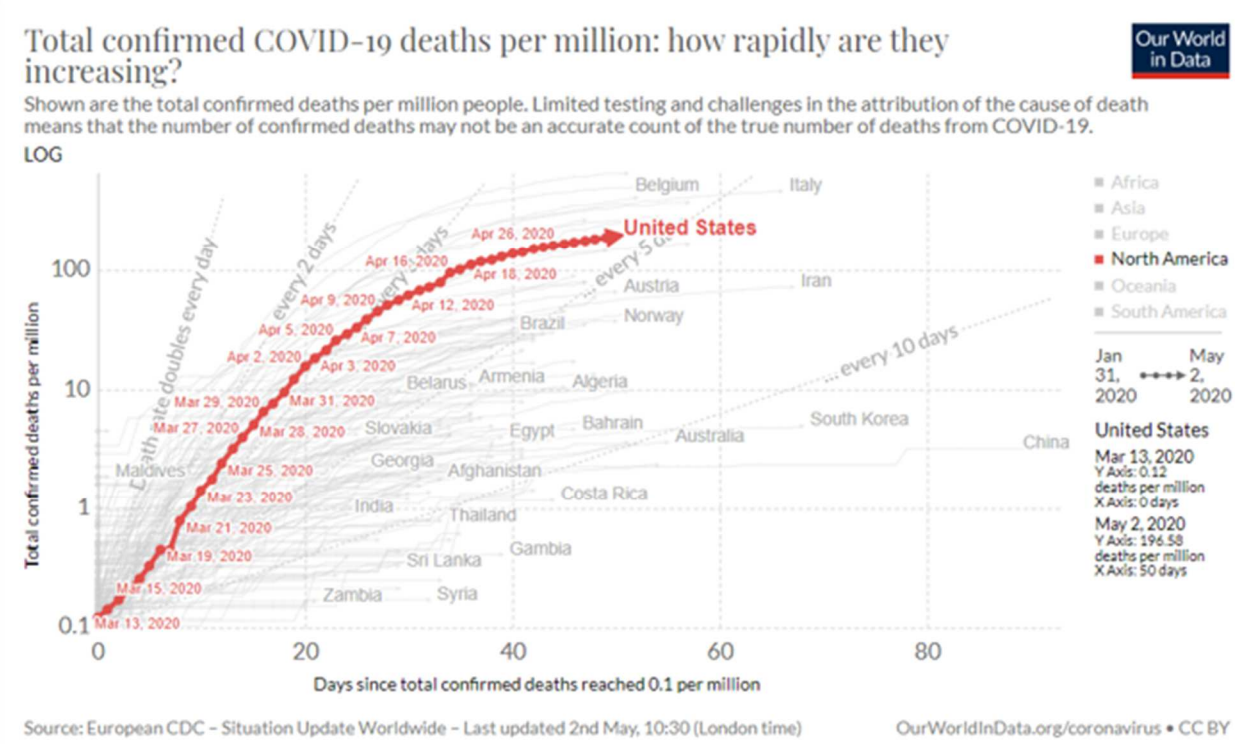
Most parts of the U.S. are now 60 days into the Covid-19 crisis. The ability of understand the dynamics of Covid-19 infection and the consequences of Covid-19 infections is important in order to identify and triage the resources needed to manage this crisis. Given the scarcity of testing for the virus and antibodies to the virus, predictions are difficult to make, but the need is so great that there are many attempts underway to project the course of the Covid-19 crisis (https://www.wired.com/story/the-mathematics-of-predicting-the-course-of-the-coronavirus/).

The traditional model that is applied to model a pandemic is the so-called SIR model, which puts members of a population into three groups: susceptible to infection, infected, and recovered or removed (which is to say, either alive and immune, or dead). Sometimes this model is augmented by including people who are "exposed" but not yet infected. In the face of inadequate testing, this model is certainly confounded in distinguishing those isolated at home who are susceptible from those isolated at home who are infected. Typically, this model yields an "i" curve for those who are infected that peaks and then subsides, as shown below.




(https://www.maa.org/press/periodicals/loci/joma/the-sir-model-for-spread-of-disease-the-differential-equation-model)

The data from around the world thus far for both the number of positive tests and the number of deaths is far more like the "r" (recovering) curve than the "i" (infected) curve, however, as shown below.




([https://ourworldindata.org/grapher/covid-death-days-since-per-million](https://ourworldindata.org/grapher/covid-death-days-since-per-million))

Curves such as the one above for the population normalized death rate consist of a series of points representing values for consecutive days. Such curves are therefore constructed from a time-series of data. If such a curve can be accurately fit to a functional form whose shape is determined by a small number of time-independent parameters, then those parameters serve to "freeze time" because those parameters hold true at least for all of the times in the original data set and hopefully also hold true for some time beyond the times in the original data set. One can then construct a model which shows how other static data for the geography in question relates to the values of the parameters used to fit to the original time-series data set.

Consider looking at the dynamics of Covid-19 infections at the state level. In late March, the New York Times published an article on the geography of Covid-19 in the U.S., inspired by analyses by Jed Kolko, Joe Cortright, and Bill Bishop. These early findings indicated that density was a very important factor in determining how the Corona-19 virus impacts a state. It has become clear, however, that it is not density alone that seems to make cities vulnerable. Consideration must also be given to the kind of density and the way in which it impacts daily work and living. Places can be dense yet still provide accommodations for people to isolate and be socially distant. In essence, there is a sizeable difference between rich dense places, where people can shelter in place, work remotely, and have all of their food and other needs delivered to them, and poor dense places, which push people out onto the streets, into stores and onto crowded transit with one another. In addition, the article also notes that Covid-19 death rates per capita are higher in counties with older populations and larger shares of minorities, and colder, wetter climates. Finally, other factors, such as the mean age of population, and the prevalence of preexisting health conditions, such as smoking, obesity, diabetes and heart disease, impact how

states are being impacted by the Covid-19 virus (https://www.citylab.com/equity/2020/04/coronavirus-spread-map-city-urban-density-suburbs-rural-data/609394/).

This paper attempts to first fit time series data representing aspects of Corona-19 virus infection and disease for each state (https://covidtracking.com/about-data/faq; https://raw.githubusercontent.com/COVID19Tracking/covid-tracking-data/master/data/states_daily_4pm_et.csv) to an accurate functional form for data from March 3, 2020 to April 13, 2020. Data science tools are then applied to additional data for each state to determine which of the features contained in the additional state data affect the value of the time-series fit parameters for each state and to what extent those features contribute to the values of the fit parameters.

Preparing Targets from State COVID-19 Data:

State COVID-19 Data was processed through several steps. First, the data was inspected and columns which were visually found to be redundant were removed:

```
# Drop total, posNeg, and hospitalized columns as they are redundant
# Drop other columns that will not be used
#df_drop = df.drop(columns = [6, 7, 8, 9, 11, 12, 14, 15, 17, 18, 19, 20, 21, 22, 23])
df_drop = df.drop(columns = ['inIcuCurrently', 'inIcuCumulative',
                             'onVentilatorCurrently', 'onVentilatorCumulative',
                             'hash', 'dateChecked', 'hospitalized', 'total',
                             'posNeg', 'fips', 'deathIncrease',
                             'hospitalizedIncrease', 'negativeIncrease',
                             'positiveIncrease', 'totalTestResultsIncrease'])
df_drop.head()
```

Then, several new features were created: percent of all tests positive, percent of positive test hospitalized, percent of positive tests recovered, and percent of positive tests deceased.

```
# Create new features
# Divide positive by totalTestResults to get positive_percent
df_drop["percent_positive"] = ""
df_drop["percent_positive"] = 100*df_drop["positive"]/df_drop["totalTestResults"]
df_drop.head()
```

```
# Divide hospitalized by positive to get hospitalized_percent
import numpy as np
df_drop["hospitalized_percent"] = ""
df_drop["hospitalized_percent"] = np.nanmax(df_drop[['hospitalizedCurrently','hospitalizedCumulative']], axis=1)
df_drop["hospitalized_percent"] = 100*df_drop["hospitalized_percent"]/df_drop["positive"]
df_drop.head()
```

```
# Divide recovered by positive to get recovered_percent
df_drop["recovered_percent"] = ""
df_drop["recovered_percent"] = 100*df_drop["recovered"]/df_drop["positive"]
df_drop.head()
```

```
# Divide death by positive to get death_percent
df_drop["death_percent"] = ""
df_drop["death_percent"] = 100*df_drop["death"]/df_drop["positive"]
df_drop.head()
```

The population of each state in 2019 was obtained from U. S. Census data to create several population normalized features. Normalization of features required several steps due to the structure of the original state COVID-19 data.

```
# Load latest state population data
import io
df_state_pop = pd.read_csv(io.StringIO(uploaded['nst-est2019-01.csv'].decode('utf-8')))
df_state_pop["Population"] = pd.to_numeric(df_state_pop["Population"])
df_state_pop.head()
```

```
# Add column of state populations (population) to df_drop_total_posNeg
# Need to sort rows by state using index numbering from state_list

df_drop["population"] = ""

for i in range(len(df_drop)):
  for index in range(len(df_state_pop)):
    if df_drop.iloc[i, 0] == df_state_pop.iloc[index, 0]:
      df_drop.iloc[i, 13] = df_state_pop.iloc[index, 1]

df_drop[["population"]] = df_drop["population"].apply(pd.to_numeric)

df_drop.head()
```

```
# Normalize positive to state population
df_drop["positive_norm"] = ""
df_drop["positive_norm"] = df_drop["positive"]/df_drop["population"]
df_drop.head()
```

```
# Normalize hospitalized to state population
df_drop["hospitalized_norm"] = ""
df_drop["hospitalized_norm"] = np.nanmax(df_drop[['hospitalizedCurrently','hospitalizedCumulative']], axis=1)
df_drop["hospitalized_norm"] = df_drop["hospitalized_norm"]/df_drop["population"]
df_drop.head()
```

```
# Normalize recovered to state population
df_drop["recovered_norm"] = ""
df_drop["recovered_norm"] = df_drop["recovered"]/df_drop["population"]
df_drop.head()
```

```
# Normalize death to state population
df_drop["death_norm"] = ""
df_drop["death_norm"] = df_drop["death"]/df_drop["population"]
df_drop.head()
```

An examination of the features of the state COVID-19 data reveals that many of the features have sparse data:

```
df_drop.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3097 entries, 2020-04-29 to 2020-01-22
Data columns (total 18 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3097 non-null   object 
 1   positive                3082 non-null   float64
 2   negative                2916 non-null   float64
 3   pending                 654 non-null    float64
 4   hospitalizedCurrently   1037 non-null   float64
 5   hospitalizedCumulative  1112 non-null   float64
 6   recovered               881 non-null    float64
 7   death                   2370 non-null   float64
 8   totalTestResults        3095 non-null   float64
 9   percent_positive        3051 non-null   float64
 10  hospitalized_percent    1678 non-null   float64
 11  recovered_percent       881 non-null    float64
 12  death_percent           2321 non-null   float64
 13  population              2917 non-null   float64
 14  positive_norm           2917 non-null   float64
 15  hospitalized_norm       1639 non-null   float64
 16  recovered_norm          806 non-null    float64
 17  death_norm              2239 non-null   float64
dtypes: float64(17), object(1)
memory usage: 539.7+ KB
```

Features were selected for further examination which have the greatest number of non-null entries: percent_positive, hospitalized_percent, death_percent, positive_norm, hospitalized_norm, and death_norm.

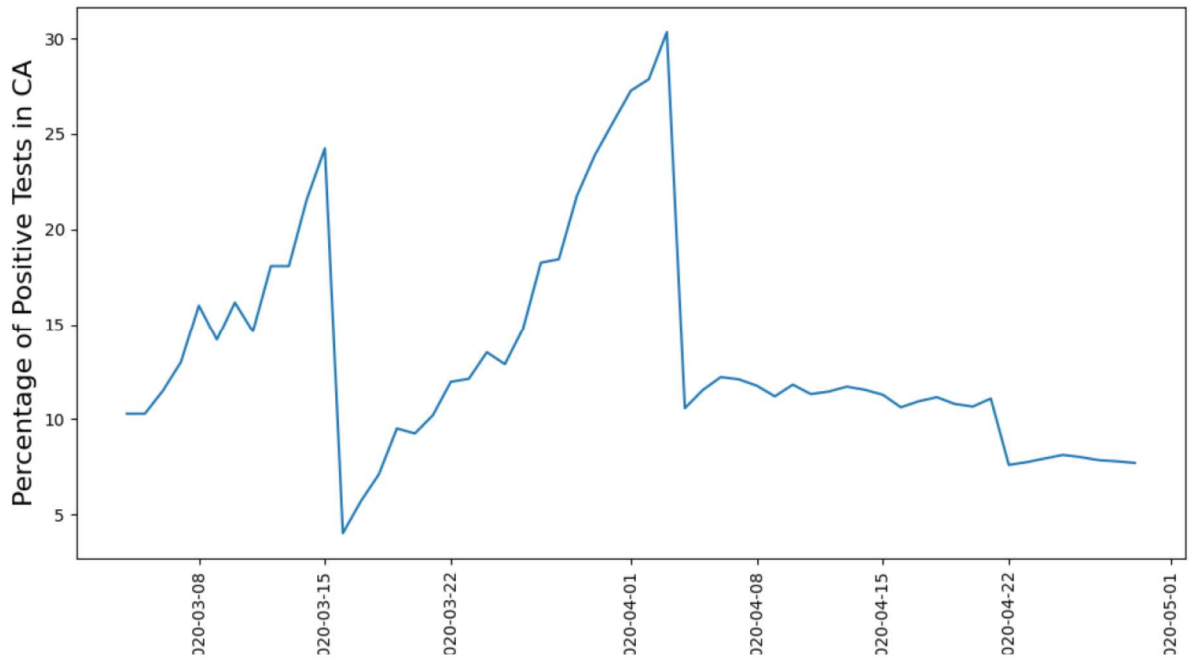
The state COVID-19 dataframe (df.state) was separated into individual dataframes for each state.

```
# Get the unique values of 'state' column
state_list = df.state.unique()
state_list
```

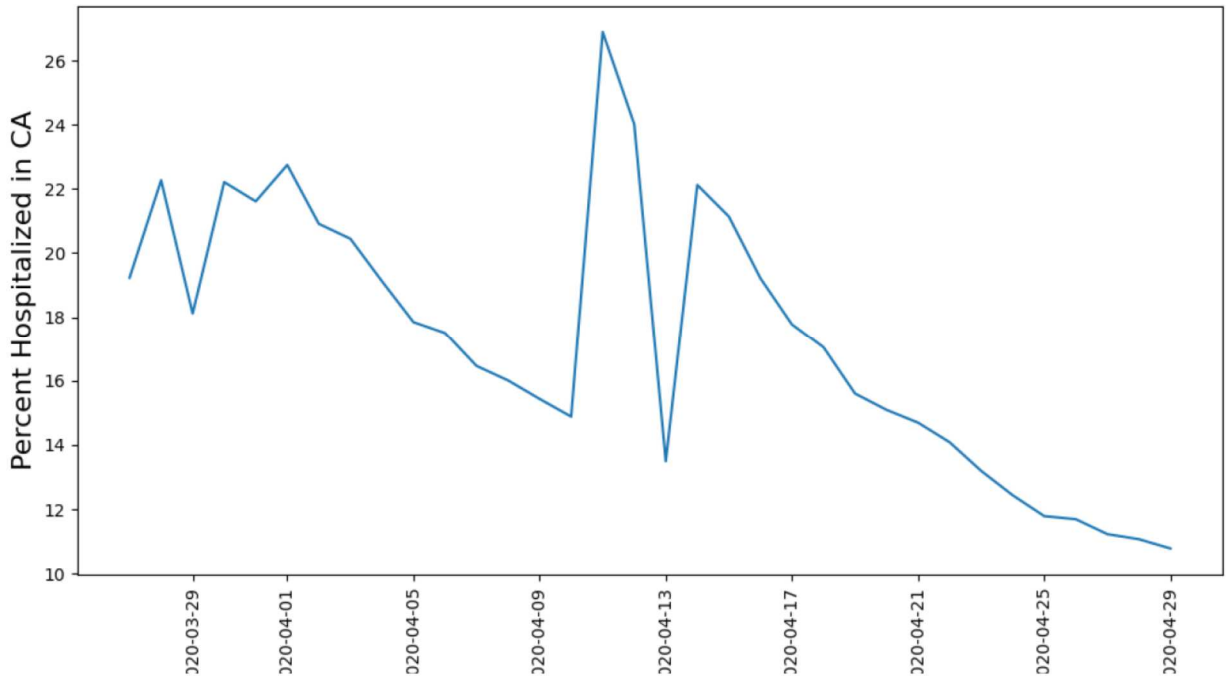
```
#create a data frame dictionary to store the state data frames
df_state_dict = {elem : pd.DataFrame for elem in state_list}

for key in df_state_dict.keys():
    df_state_dict[key] = df_drop[:][df_drop.state == key]
```

Select features were then plotted for each state for examination. The percentage features for a state were generally found to give erratic results:

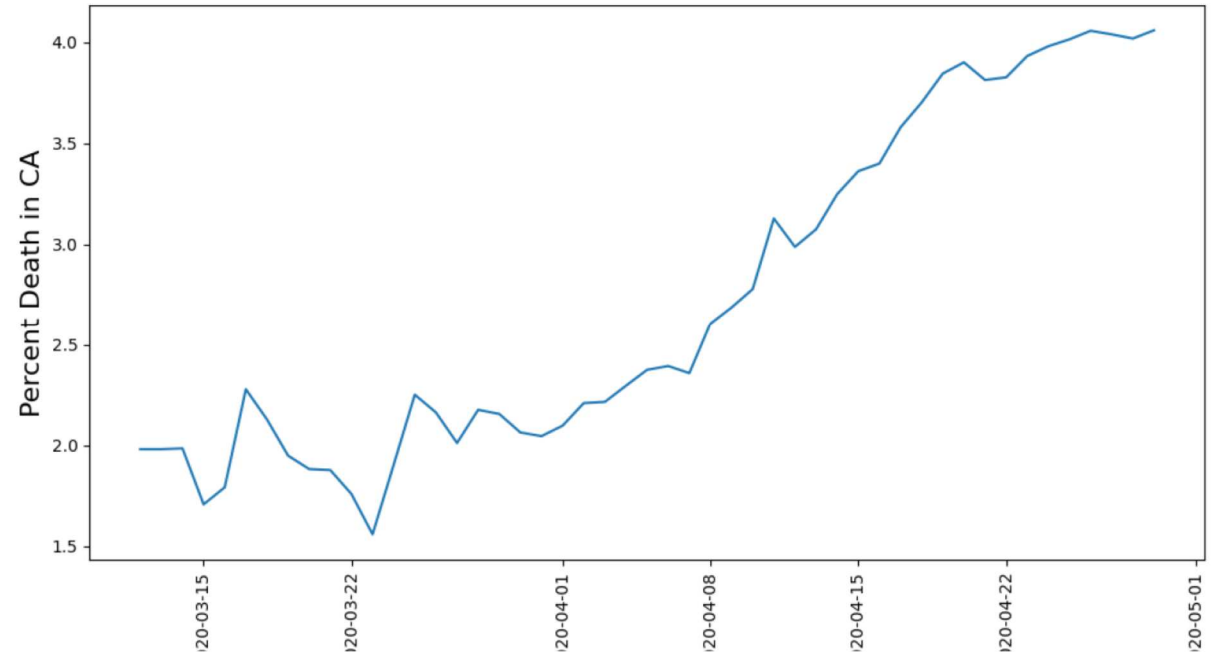
```
fig = plt.figure()
fig, ax = plt.subplots(figsize=(12, 6))
plt.rcParams.update(plt.rcParamsDefault)

plt.plot(df_state_dict['CA'].percent_positive)
plt.xticks(rotation='vertical')

plt.legend(frameon=False)
plt.xlabel('Date', fontsize=18)
plt.ylabel('Percentage of Positive Tests in CA', fontsize=16)
plt.show()
```




```
fig = plt.figure()
fig, ax = plt.subplots(figsize=(12, 6))
plt.rcParams.update(plt.rcParamsDefault)

plt.plot(df_state_dict['CA'].hospitalized_percent)
plt.xticks(rotation='vertical')

plt.legend(frameon=False)
plt.xlabel('Date', fontsize=18)
plt.ylabel('Percent Hospitalized in CA', fontsize=16)
plt.show()
```



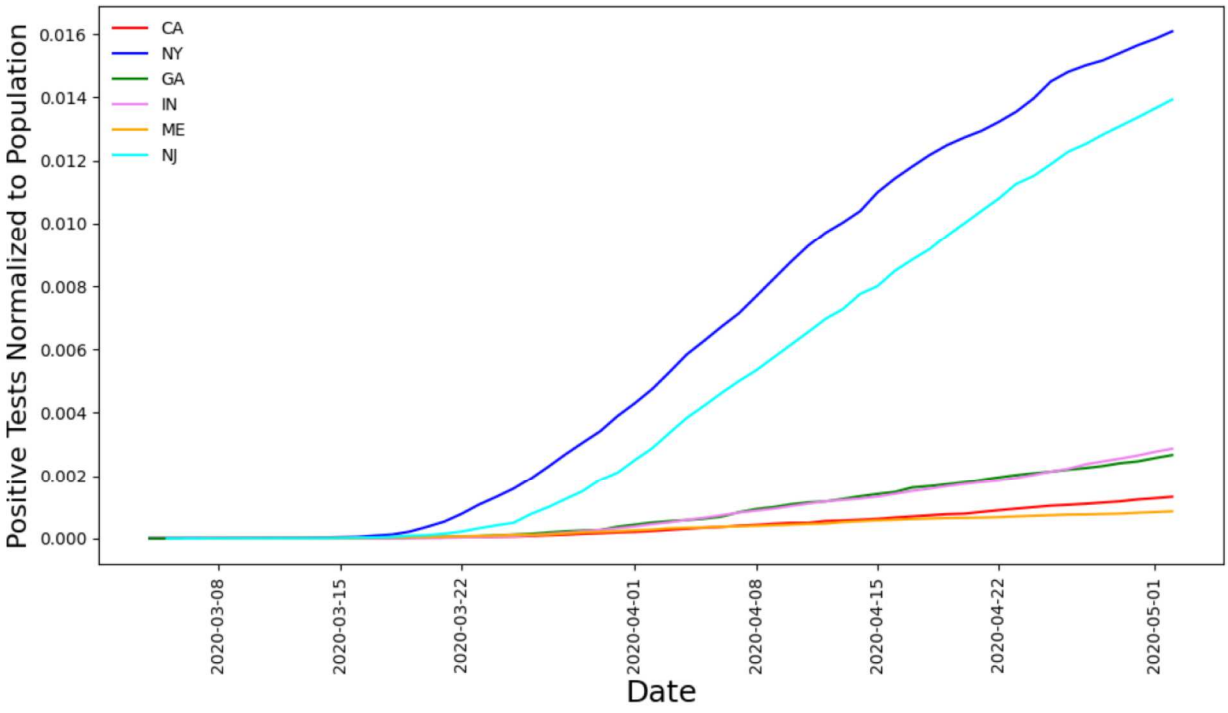

```
fig = plt.figure()
fig, ax = plt.subplots(figsize=(12, 6))
plt.rcParams.update(plt.rcParamsDefault)

plt.plot(df_state_dict['CA'].death_percent)
plt.xticks(rotation='vertical')

plt.legend(frameon=False)
plt.xlabel('Date', fontsize=18)
plt.ylabel('Percent Death in CA', fontsize=16)
plt.show()
```



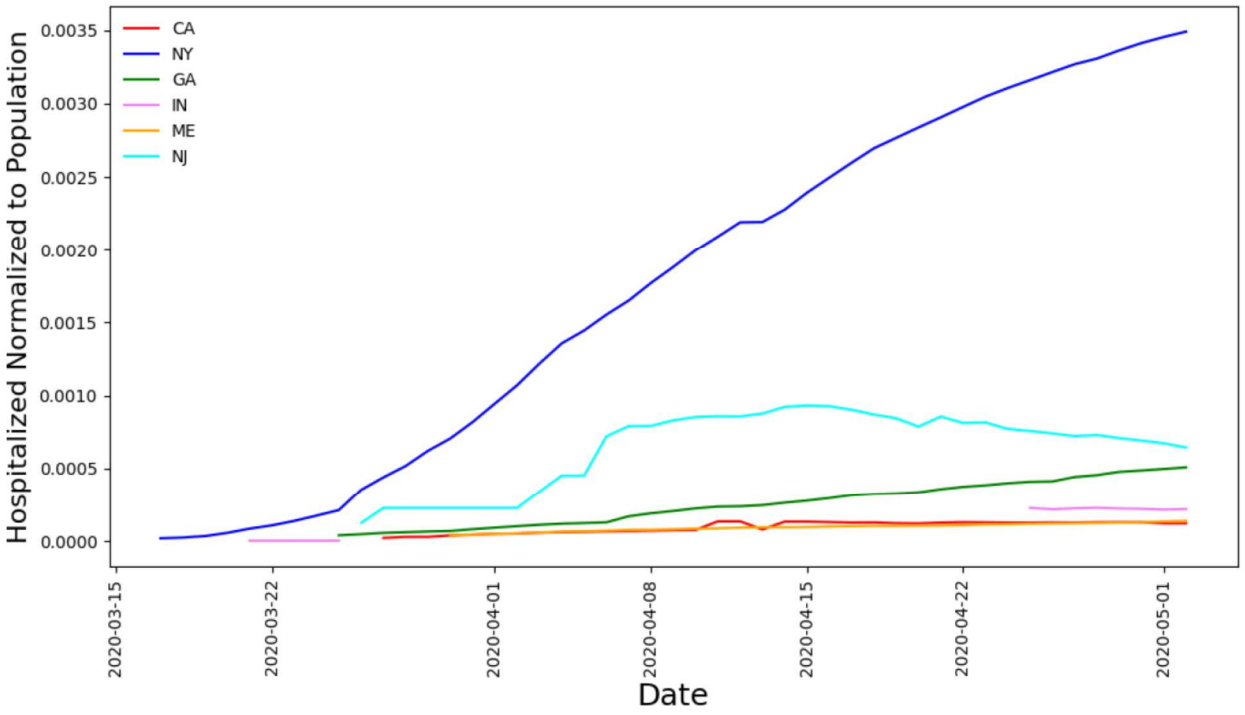
The observed erratic behavior in the percentage features may be due to the differences in testing techniques and reliabilities, both of which were particularly large in the early days of the COVID-19 crisis.

Population normalization allows analogous features to be compared among states. Population normalized features did not exhibit the kind of erratic behavior that was found in the percentage features.

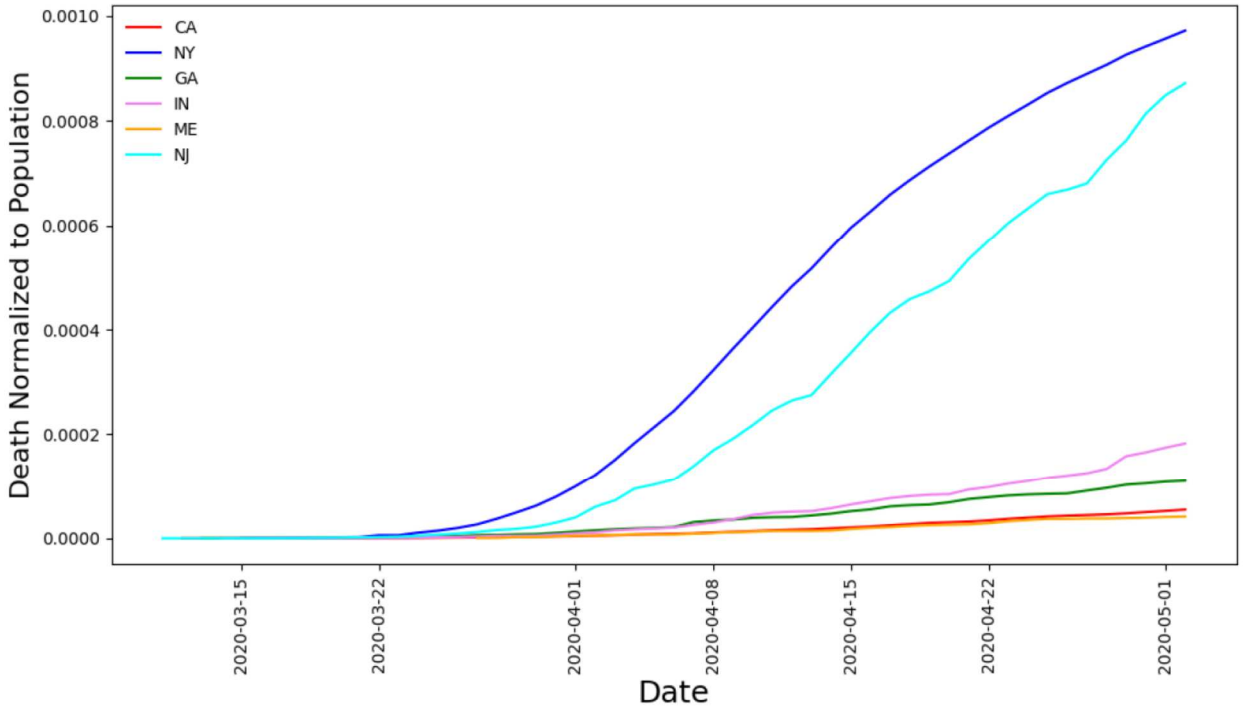
```
fig = plt.figure()
fig, ax = plt.subplots(figsize=(12, 6))
plt.rcParams.update(plt.rcParamsDefault)

plt.plot(df_state_dict['CA'].positive_norm, color="red", label="CA")
plt.plot(df_state_dict['NY'].positive_norm, color="blue", label="NY")
plt.plot(df_state_dict['GA'].positive_norm, color="green", label="GA")
plt.plot(df_state_dict['IN'].positive_norm, color="violet", label="IN")
plt.plot(df_state_dict['ME'].positive_norm, color="orange", label="ME")
plt.plot(df_state_dict['NJ'].positive_norm, color="cyan", label="NJ")
plt.xticks(rotation='vertical')

plt.legend(frameon=False)
plt.xlabel('Date', fontsize=18)
plt.ylabel('Positive Tests Normalized to Population', fontsize=16)
plt.show()
```



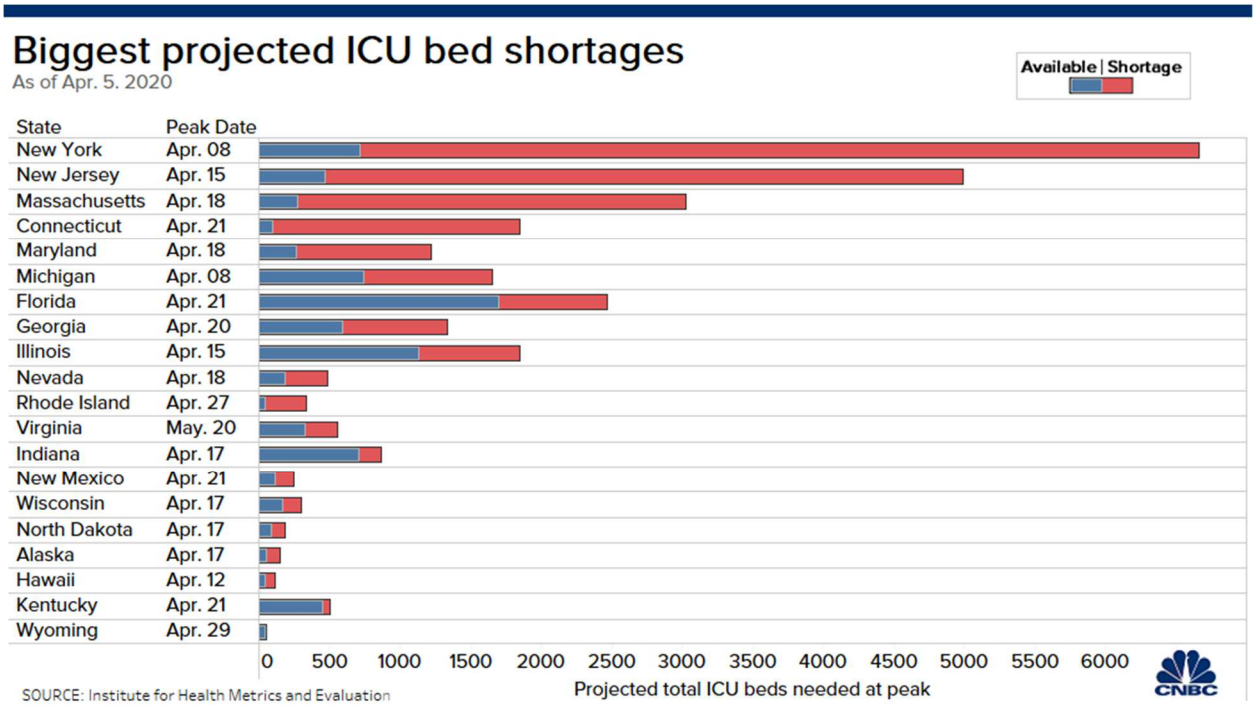
```
fig = plt.figure()
fig, ax = plt.subplots(figsize=(12, 6))
plt.rcParams.update(plt.rcParamsDefault)

plt.plot(df_state_dict['CA'].hospitalized_norm, color="red", label="CA")
plt.plot(df_state_dict['NY'].hospitalized_norm, color="blue", label="NY")
plt.plot(df_state_dict['GA'].hospitalized_norm, color="green", label="GA")
plt.plot(df_state_dict['IN'].hospitalized_norm, color="violet", label="IN")
plt.plot(df_state_dict['ME'].hospitalized_norm, color="orange", label="ME")
plt.plot(df_state_dict['NJ'].hospitalized_norm, color="cyan", label="NJ")
plt.xticks(rotation='vertical')

plt.legend(frameon=False)
plt.xlabel('Date', fontsize=18)
plt.ylabel('Hospitalized Normalized to Population', fontsize=16)
plt.show()
```




```
fig = plt.figure()
fig, ax = plt.subplots(figsize=(12, 6))
plt.rcParams.update(plt.rcParamsDefault)

plt.plot(df_state_dict['CA'].death_norm, color="red", label="CA")
plt.plot(df_state_dict['NY'].death_norm, color="blue", label="NY")
plt.plot(df_state_dict['GA'].death_norm, color="green", label="GA")
plt.plot(df_state_dict['IN'].death_norm, color="violet", label="IN")
plt.plot(df_state_dict['ME'].death_norm, color="orange", label="ME")
plt.plot(df_state_dict['NJ'].death_norm, color="cyan", label="NJ")
plt.xticks(rotation='vertical')

plt.legend(frameon=False)
plt.xlabel('Date', fontsize=18)
plt.ylabel('Death Normalized to Population', fontsize=16)
plt.show()
```




Note that population normalized death curves relate closely to population normalized positive test curves. The normalized hospitalization data has numerous gaps, so that it was not used going forward, although it is noteworthy that, for New Jersey, its normalized hospitalization curve does not track with its normalized positive test curve and its normalized death curve. This may be indicative of New Jersey hospital bed capacity being inadequate to meet hospital bed demand in the state for a sustained period of time (beyond 04/05/2020).



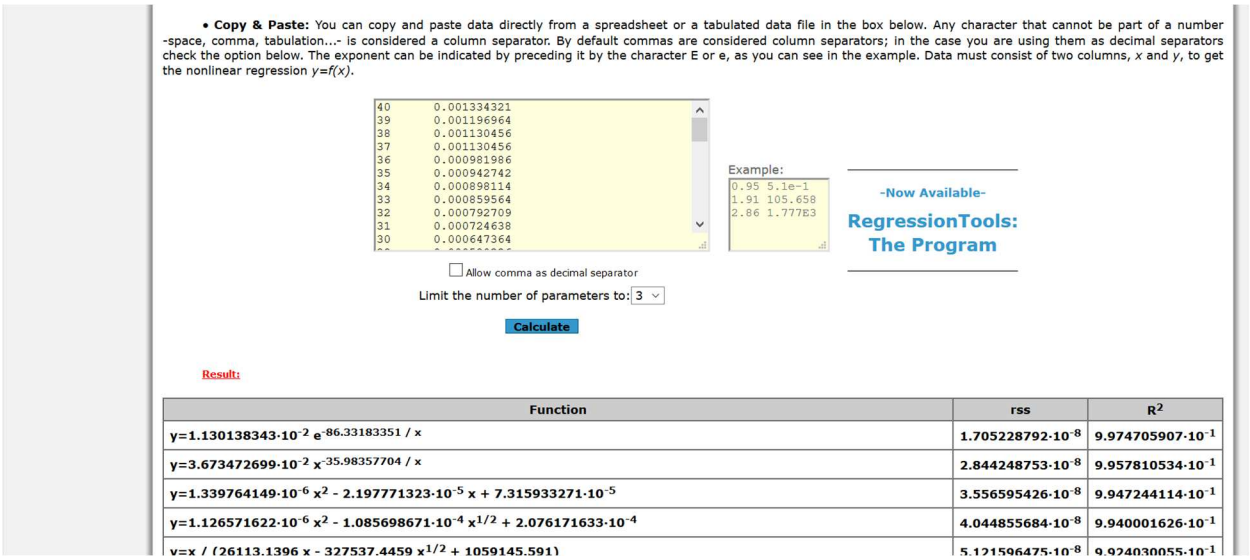

(https://www.cnbc.com/2020/04/06/coronavirus-cases-states-with-biggest-hospital-bed-shortfalls.html)

An attempt was made to fit the population normalized positive test and death curves to a functional form which would allow the use of a relatively small number of parameters and a functional form to represent the time series data underlying those curves for each state. A review of the literature suggested that it should be possible to fit the time series data for the states to a logistic model:

$$Q_t = \frac{a}{1+e^{b-c(t-t_0)}}$$

where $Q_t$ is the cumulative confirmed cases (deaths); a is the predicted maximum of confirmed cases (deaths). b and c are fitting coefficients. t is the number of days since the first case. $t_0$ is the time when the first case occurred (https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&cad=rja&uact=8&ved=2ahUKEwjw2J-C0uPoAhVYnp4KHburC60QFjACegQIBBAB&url=https%3A%2F%2Farxiv.org%2Fpdf%2F2003.05447&usg=AOvVaw0m7OQ7RRxpGKrGPUKHSqxo).

For each data set, the data was systematically fit to 31 different functional forms, including that of a logistic model, at the following website: http://www.xuru.org/rt/NLR.asp#CopyPaste
Although the algorithm was set to use up to three parameters, the algorithm consistently used only two parameters. The site provides both RSS and r2 statistics for each functional form fit. An example of the output using Colorado's population normalized positive test data as input is as follows:



• **Copy & Paste:** You can copy and paste data directly from a spreadsheet or a tabulated data file in the box below. Any character that cannot be part of a number -space, comma, tabulation...- is considered a column separator. By default commas are considered column separators; in the case you are using them as decimal separators check the option below. The exponent can be indicated by preceding it by the character E or e, as you can see in the example. Data must consist of two columns, *x* and *y*, to get the nonlinear regression *y=f(x)*.

40 0.001334321
39 0.001196964
38 0.001130456
37 0.001130456
36 0.000981986
35 0.000942742
34 0.000898114
33 0.000859564
32 0.000792709
31 0.000724638
30 0.000647364

Example:
0.95 5.1e-1
1.91 105.658
2.86 1.777E3

-Now Available-
RegressionTools: The Program

Allow comma as decimal separator

Limit the number of parameters to: 3

Calculate

Result:

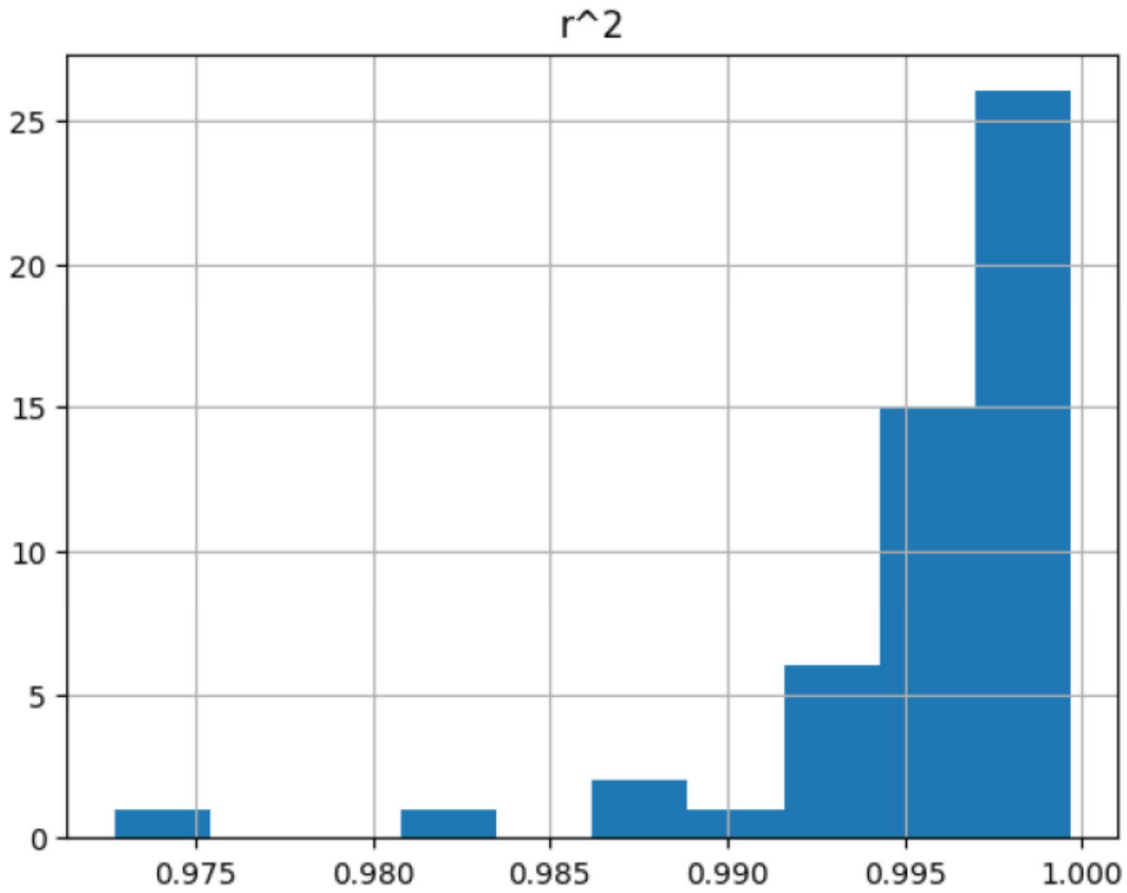
| Function | rss | $R^2$ |
|---|---|---|
| $y=1.130138343\cdot10^{-2}\ e^{-86.33183351\ /\ x}$ | $1.705228792\cdot10^{-8}$ | $9.974705907\cdot10^{-1}$ |
| $y=3.673472699\cdot10^{-2}\ x^{-35.98357704\ /\ x}$ | $2.844248753\cdot10^{-8}$ | $9.957810534\cdot10^{-1}$ |
| $y=1.339764149\cdot10^{-6}\ x^2 - 2.197771323\cdot10^{-5}\ x + 7.315933271\cdot10^{-5}$ | $3.556595426\cdot10^{-8}$ | $9.947244114\cdot10^{-1}$ |
| $y=1.126571622\cdot10^{-6}\ x^2 - 1.085698671\cdot10^{-4}\ x^{1/2} + 2.076171633\cdot10^{-4}$ | $4.044855684\cdot10^{-8}$ | $9.940001626\cdot10^{-1}$ |
| $y=x\ /\ (26113.1396\ x - 327537.4459\ x^{1/2} + 1059145.591)$ | $5.121596475\cdot10^{-8}$ | $9.924030055\cdot10^{-1}$ |

*Population Normalized Positive Test Data*:

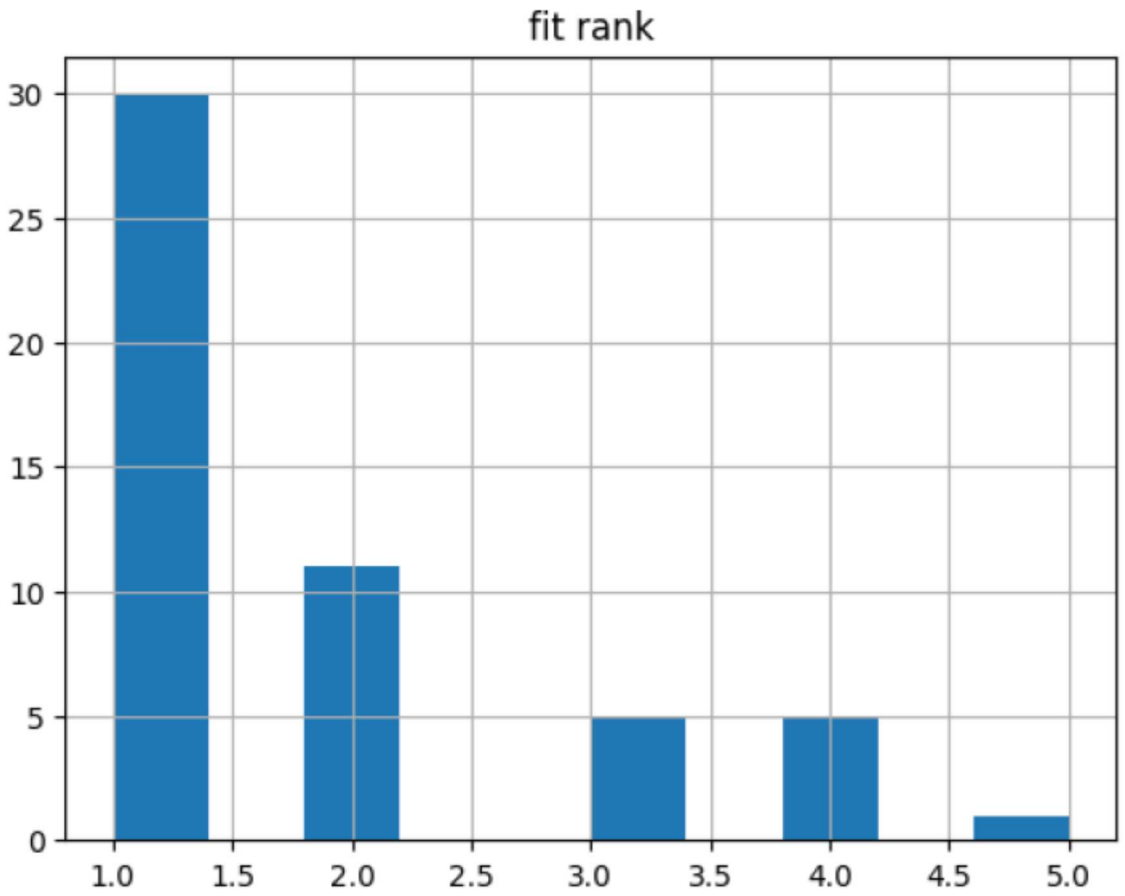
For the vast majority of states, the following functional form provided the best fit:

$$P_t = a*e^{b/(t-t_0)}$$

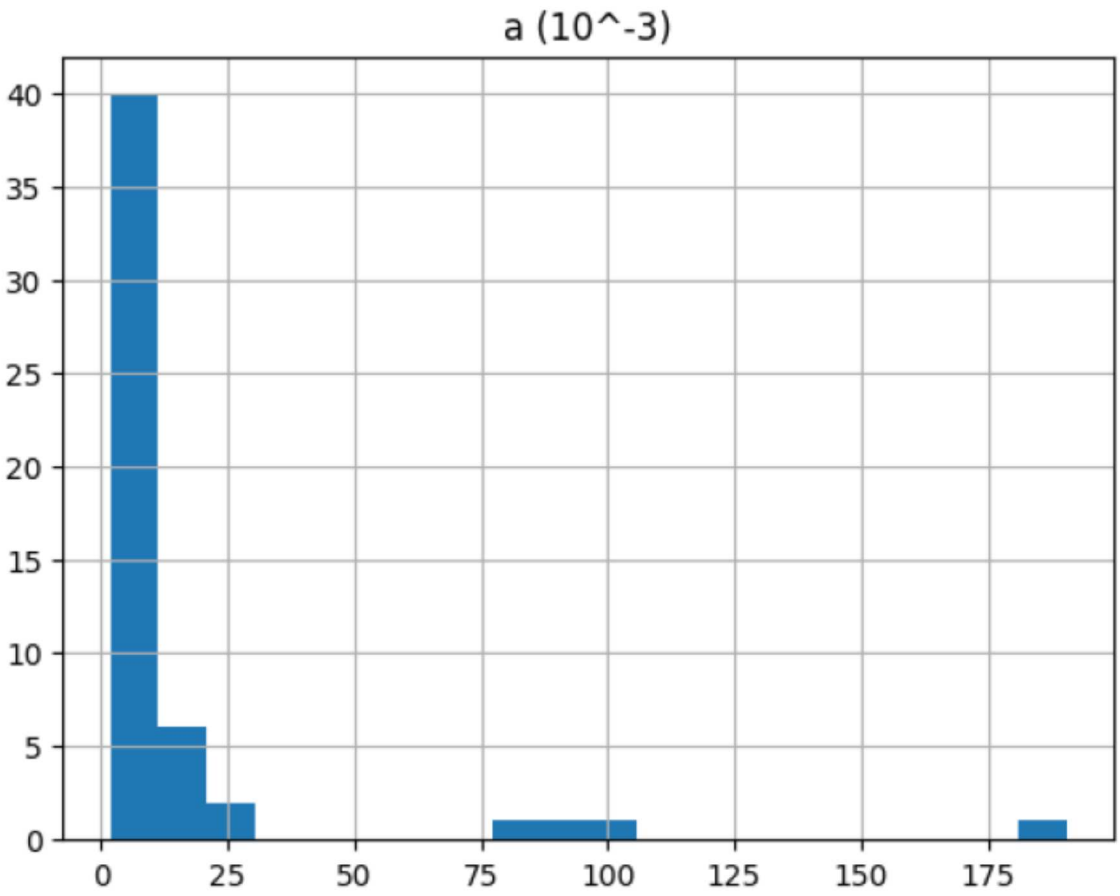
Note that, as time gets very large, the argument of the exponent goes to zero and the value of P goes to a. Thus, a is the maximum value of P. When time goes to zero, the argument of the exponent gets very large and, since b is always negative, the exponent goes to zero so that the value of P goes to zero. The quality of fit is captured in the following histogram that shows the distribution of $r^2$ scores for fitting the state population normalized data sets to the above functional form.




The following histogram indicates that the above functional form was the top choice among the 31 candidate functional forms for fitting the time series of population normalized positive test data for 30 of the 50 states, DC and Puerto Rico (57.7%). This histogram is almost the mirror image of the previous histogram:

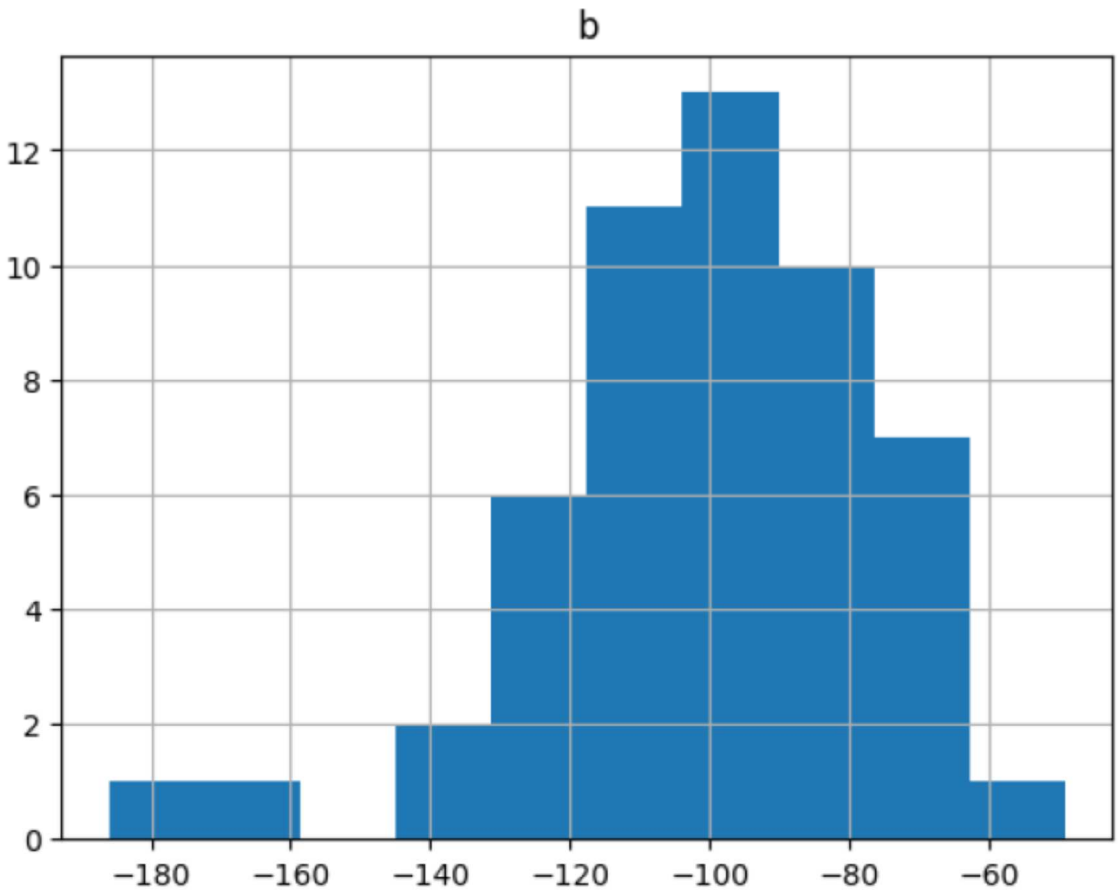



The values for the “a” parameter for the 50 states, DC, and Puerto Rico are distributed as follows:




The high value outliers are NJ (fit rank 1), NY (fit rank 1), RI (fit rank 5), and SD (fit rank 4).

The values for the “b” parameter for the 50 states, DC, and Puerto Rico have a broader distribution:



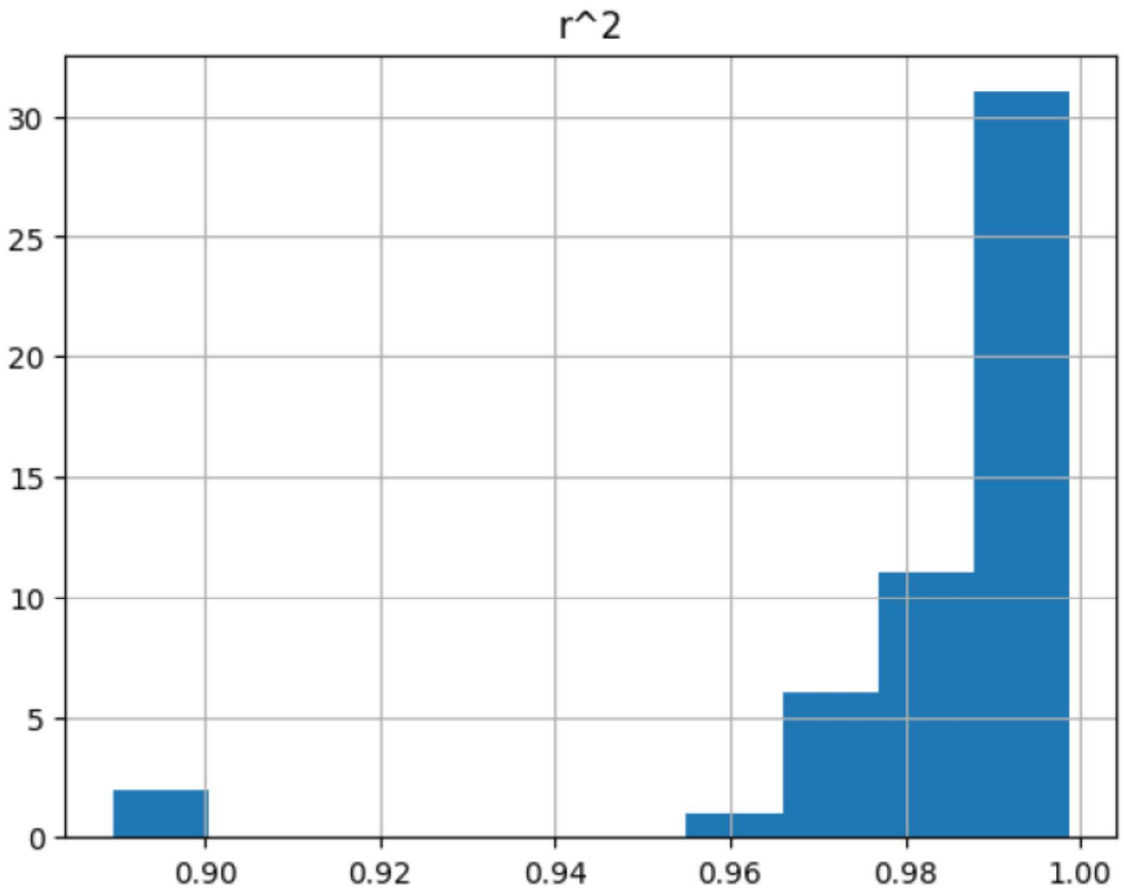

The low value outliers are RI (fit rank 5) and SD (fit rank 4).

*Population Normalized Death Data*:

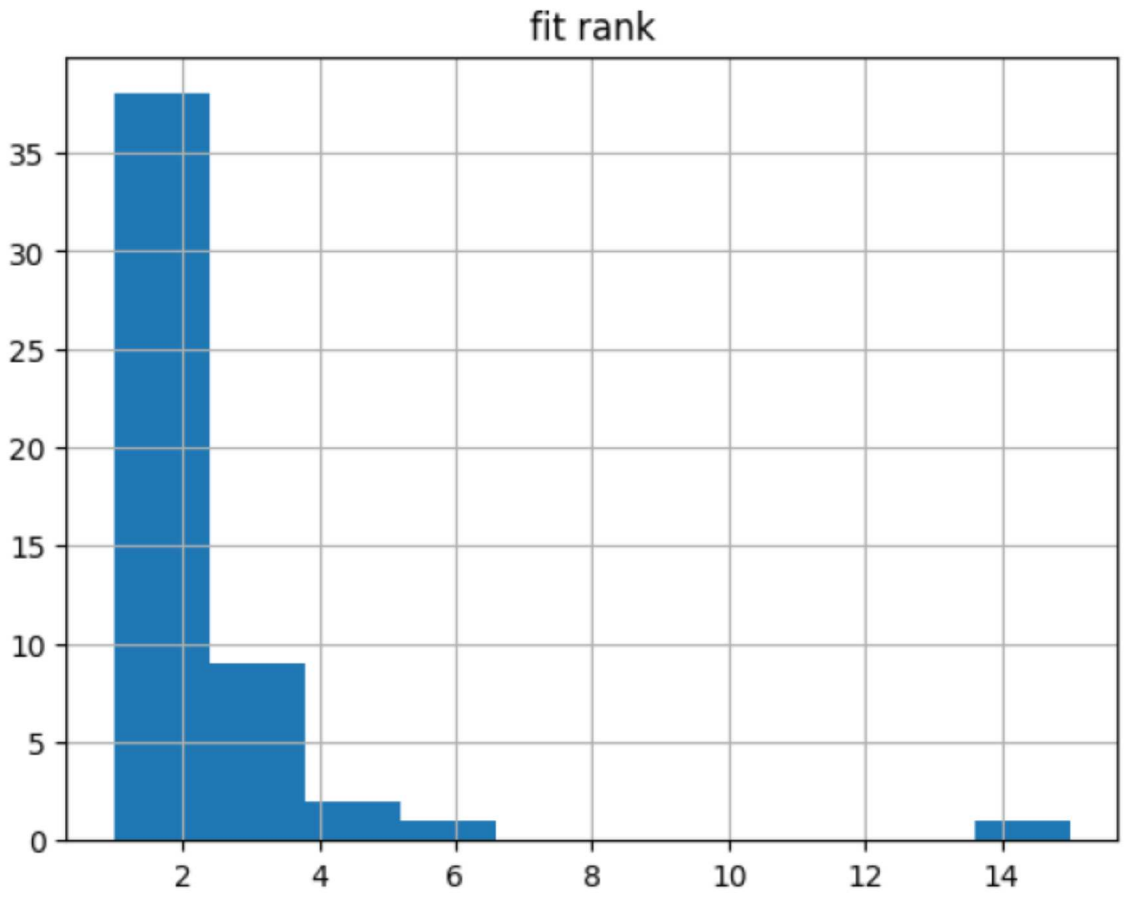
For the vast majority of states, the following functional form provided the best fit:

$$D_t = c*e^{d/(t-t_0)}$$

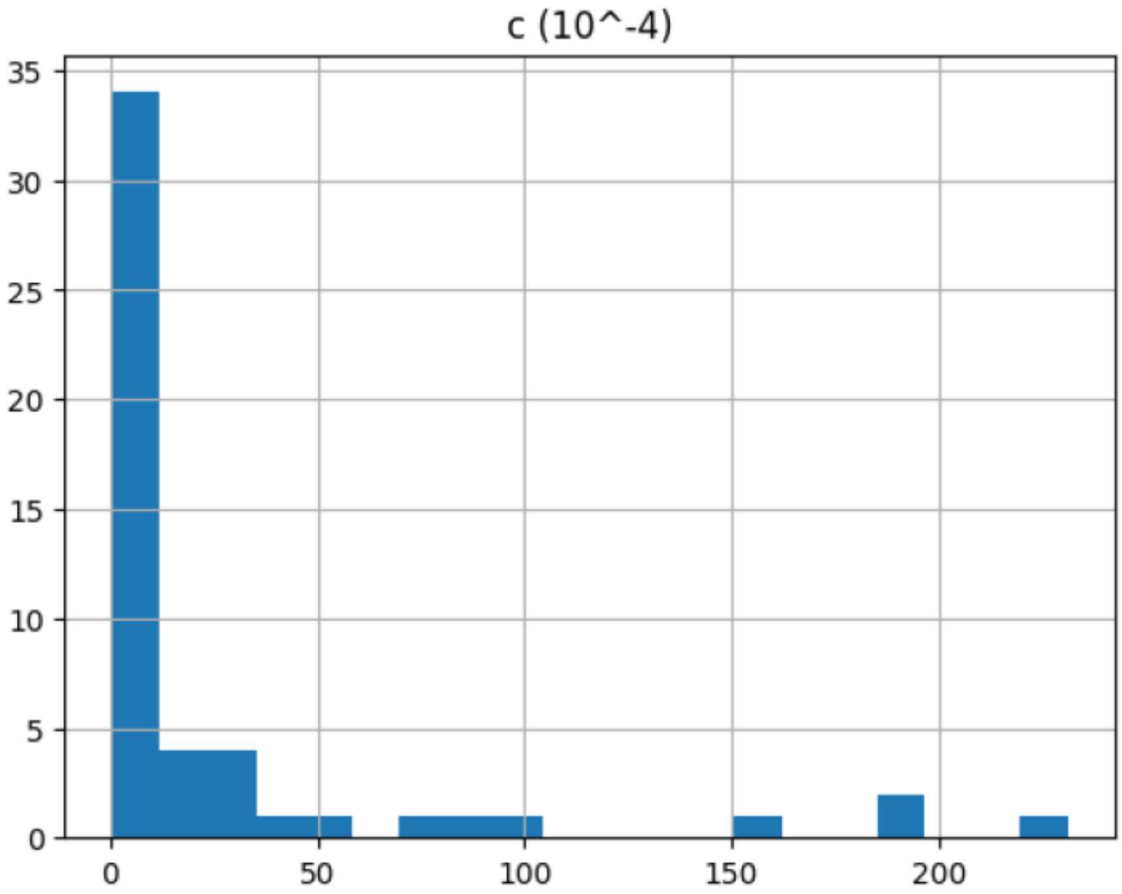
Note that, as time gets very large, the argument of the exponent goes to zero and the value of D goes to c. Thus, c is the maximum value for D. When time goes to zero, the argument of the exponent gets very large and, since d is always negative, the exponent goes to zero so that the value of D goes to zero. The quality of fit is captured in the following histogram that shows the distribution of $r^2$ scores for fitting the state population normalized data sets to the above functional form.




The following histogram indicates that the above functional form was the top choice among the 31 candidate functional forms for fitting the time series of population normalized death data for 38 of the 50 states and DC (74.5%). This histogram is almost the mirror image of the previous histogram:

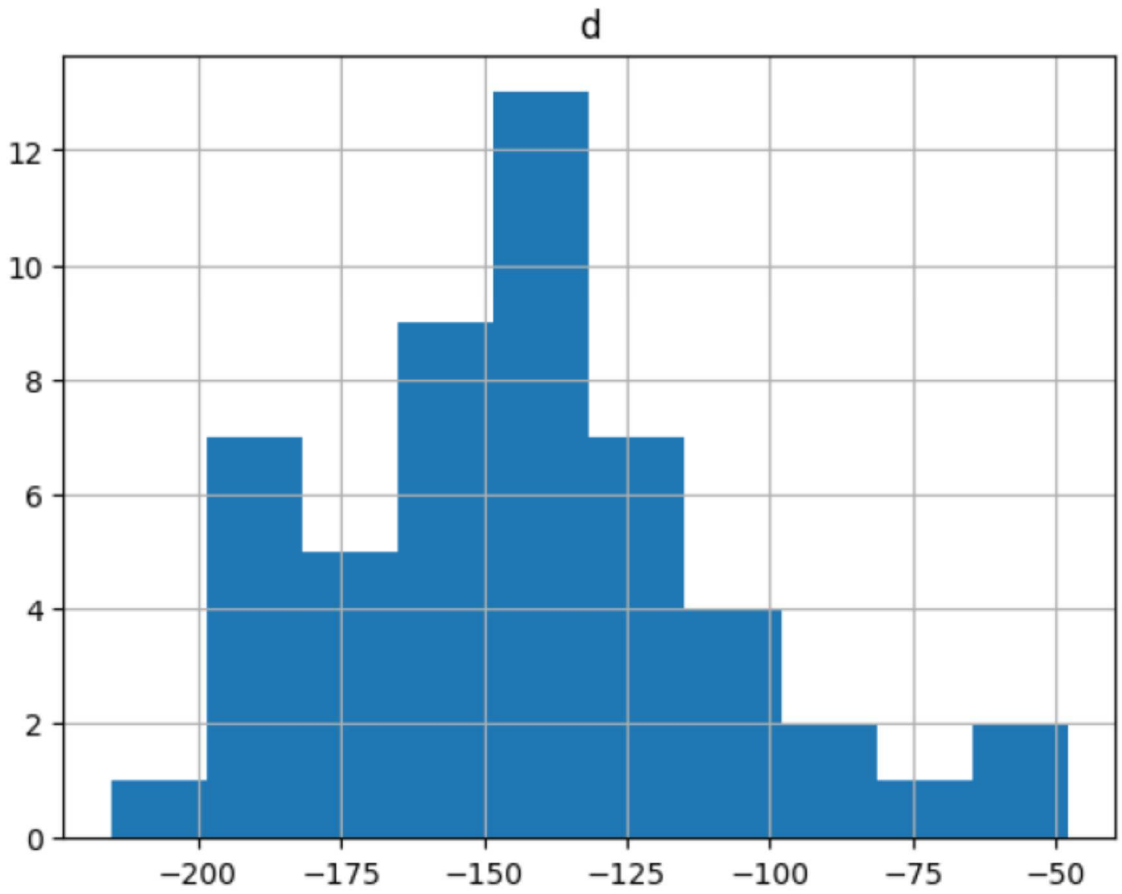



The values for the “c” parameter for the 50 states and DC are distributed as follows:




The high value outliers are NY (fit rank 1), NJ (fit rank 1), CT (fit rank 1), and MA (fit rank 2).

The values for the “d” parameter for the 50 states and DC have a broader distribution:




The high value outliers are WA (fit rank 15) and MT (fit rank 5).

Preparing Features from State Data:

Data reflecting a variety of characteristics of the states was collected from several sources. Static non-health characteristics of the states, such as latitude of center, longitude of center, number of counties, and number of bordering states, were obtained from:

https://www.factmonster.com/explore-all-fifty-us-states
https://www.factmonster.com/us/us-geography/highest-lowest-and-mean-elevations-united-states
https://www.factmonster.com/us/states/land-and-water-area-of-states-2000
https://inkplant.com/code/state-latitudes-longitudes

Health and demographic characteristics of the states were obtained from:

https://worldpopulationreview.com/states/state-densities
https://www.icip.iastate.edu/tables/population/urban-pct-states
https://www.kff.org/other/state-indicator/distribution-by-age/?currentTimeframe=0&sortModel=%7B%22colId%22:%22Location%22,%22sort%22:%22asc%22%7D
https://www.finra.org/rules-guidance/key-topics/covid-19/shelter-in-place

CMS provider-level utilization data (Medicare_Physician_and_Other_Supplier_National_Provider_Identifier__NPI__Aggregate_Report__Calendar_Year_2017.csv) was retrieved from:

https://data.cms.gov/utilization-and-payment/related-data

This data was aggregated to zip-code-level data using code written by Wilton Lam that can be found at:

https://github.com/lamwilton/COVID-19/blob/master/Neural.ipynb

This compilation of data resulted in a collection of 115 features.

Ten additional features were created and added to the data set:

```
# Feature Engineering
# Land Area/Water Area
# df_state_data['State Area Ratio'] = df_state_data['Land Area']/df_state_data['Water Area']
df_state_data['State Area Ratio'] = df_state_data['Land Area'].divide(df_state_data['Water Area'], fill_value=0)

# Elevation Ratio = Highest Elevation/Mean Elevation
# df_state_data['Elevation Ratio'] = df_state_data['Highest Elevation']/df_state_data['Mean Elevation']
df_state_data['Elevation Ratio'] = df_state_data['Highest Elevation'].divide(df_state_data['Mean Elevation'], fill_v

# Capital Area Ratio = Capital Land Area/Capital Water Area
# df_state_data['Capital Area Ratio'] = df_state_data['Captial Land Area']/df_state_data['Capital Water Area']
df_state_data['Captial Land Area'] = df_state_data['Captial Land Area'].astype(float)
df_state_data['Capital Area Ratio'] = df_state_data['Captial Land Area'].divide(df_state_data['Capital Water Area'],

# Boundaries = Number of boarding states + On Coast + Borders Another Country
df_state_data['Boundaries'] = df_state_data['Number of bordering states'] + df_state_data['On Coast'] + df_state_dat

# Latitude Difference to State Capital = Latitude - Capital Latitude
df_state_data['Latitude Difference to State Capital'] = df_state_data['Latitude'] - df_state_data['Capital Latitude'

# Longitude Difference to State Capital = Capital Longitude - Longitude
df_state_data['Longitude Difference to State Capital'] = df_state_data['Capital Longitude'] - df_state_data['Longitu

# Latitude Difference to DC = Latitude - DC Latitude
df_state_data['Latitude Difference to DC'] = df_state_data['Latitude'] - 38.904722

# Longitude Difference to DC = DC Longitude - Longitude
df_state_data['Longitude Difference to DC'] = -77.016389 - df_state_data['Longitude']

# Latitude Difference to US Center = Latitude - Center Latitude
df_state_data['Latitude Difference to Center'] = df_state_data['Latitude'] - 39.833333

# Longitude Different to US Center = Center Longitude - Longitude
df_state_data['Longitude Difference to Center'] = -98.585522 - df_state_data['Longitude']
```

resulting in a total of 125 features.

A correlation matrix for the features was constructed,

```
# Define variables for regression
df_temp1 = df_state_data.drop(df_state_data.index[[3, 12, 27, 42, 50]])
X = df_temp1.drop('State', axis = 1)
```

```
# Look at correlation coefficients
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 1000)
X.corr()
```

revealing that there were a significant number of feature pairs with high correlation coefficients. Therefore, feature pairs with correlation coefficients greater than 0.95 were eliminated:

```
#  Note that there are many highly correlated features which need to be dropped
# Create absolute value correlation matrix
corr_matrix = X.corr().abs()

# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))

# Find index of feature columns with correlation greater than 0.95
to_drop = [column for column in upper.columns if any(upper[column] > 0.95)]

# Drop features by index which were identified as being highly correlated
X = X.drop(X[to_drop], axis=1)
```

Reducing our feature set from 125 features to 38 features.

Applying Regression to Fit State Features to Parameter Targets:

*Population Normalized Positive Test Parameters a and b as Targets*:

The data was separated into train and validate sets using a random 75/25 split.

```
# Train/validate split: random 75/25% train/validate split.
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size = 0.25, random_state = 42)

X_train.shape, y_train.shape, X_val.shape, y_val.shape
```

such that 38 rows were in the train set and 13 rows were in the validate set.

Grid Search Cross Validation was used to find the optimum hyper-parameters for Random Forest Regression with regard to the “a” parameter.

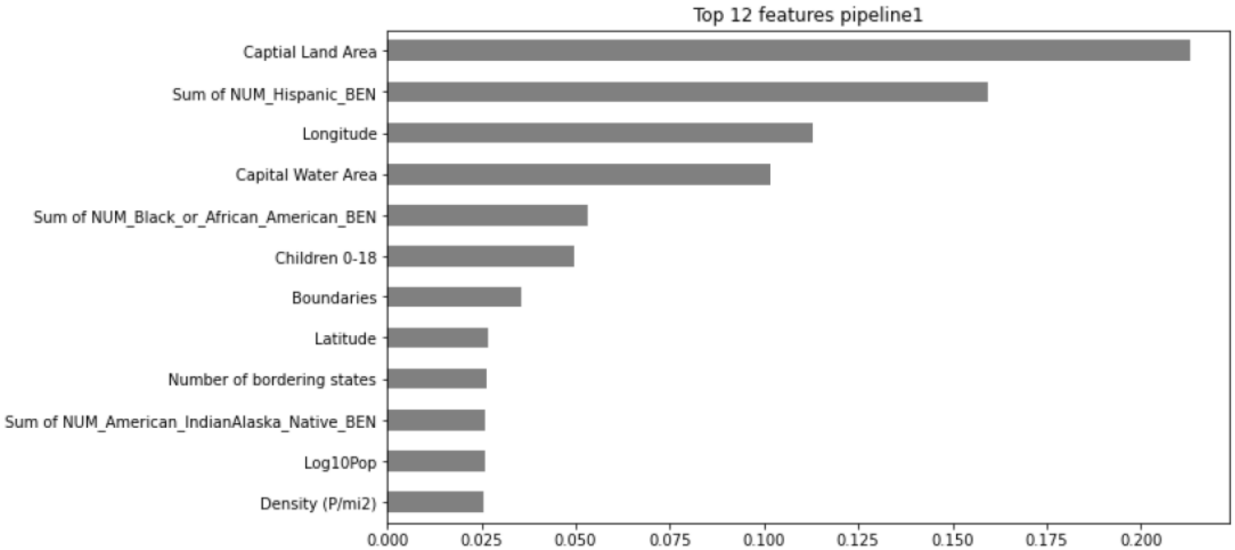
```
# Optimizing Hyperparameters for Random Forest Regressor
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

# Define classifier
forest = RandomForestRegressor(random_state = 1)

# Parameters to fit

max_depth = [2, 3, 4]
n_estimators = [35, 36, 37]
min_samples_split = [1.5, 2, 2.5]
min_samples_leaf = [3.5, 4, 4.5]
max_leaf_nodes = [None]
max_features = ['auto']
ccp_alpha = [0.0, 0.00625, 0.0125]
min_weight_fraction_leaf = [0.0, 0.00625, 0.0125]

hyperF = dict(n_estimators = n_estimators, max_depth = max_depth,
              min_samples_split = min_samples_split,
              min_samples_leaf = min_samples_leaf,
              max_leaf_nodes = max_leaf_nodes,
              max_features = max_features,
              ccp_alpha=ccp_alpha,
              min_weight_fraction_leaf=min_weight_fraction_leaf)

gridF = GridSearchCV(forest, hyperF, cv = 3, verbose = 10,
                     scoring='r2', return_train_score=True,
                     n_jobs = -1)
bestF = gridF.fit(X_train, y_train)

# Output best accuracy and best parameters
print('The score achieved with the best parameters = ', gridF.best_score_, '\n')
print('The parameters are:', gridF.best_params_)
```

The optimal parameters were then used to perform the initial Random Forest Regression for the normed positive test data.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import make_pipeline
import category_encoders as ce
from sklearn.impute import SimpleImputer

pipeline1 = make_pipeline(
    ce.OneHotEncoder(use_cat_names=True),
    SimpleImputer(strategy='mean'),
    RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=3, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=4,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=36, n_jobs=None, oob_score=False,
                      random_state=0, verbose=0, warm_start=False))

pipeline1.fit(X_train, y_train)

# Get the model's training accuracy
print("Training Accurary:  R^2 = ", pipeline1.score(X_train,y_train))

# Get the model's validation accuracy
print('Validation Accuracy:  R^2 = ', pipeline1.score(X_val, y_val))
```

yielding a training accuracy of $r^2 = 0.3014$, a validation accuracy of $r^2 = 0.0694$, and the following feature importances:

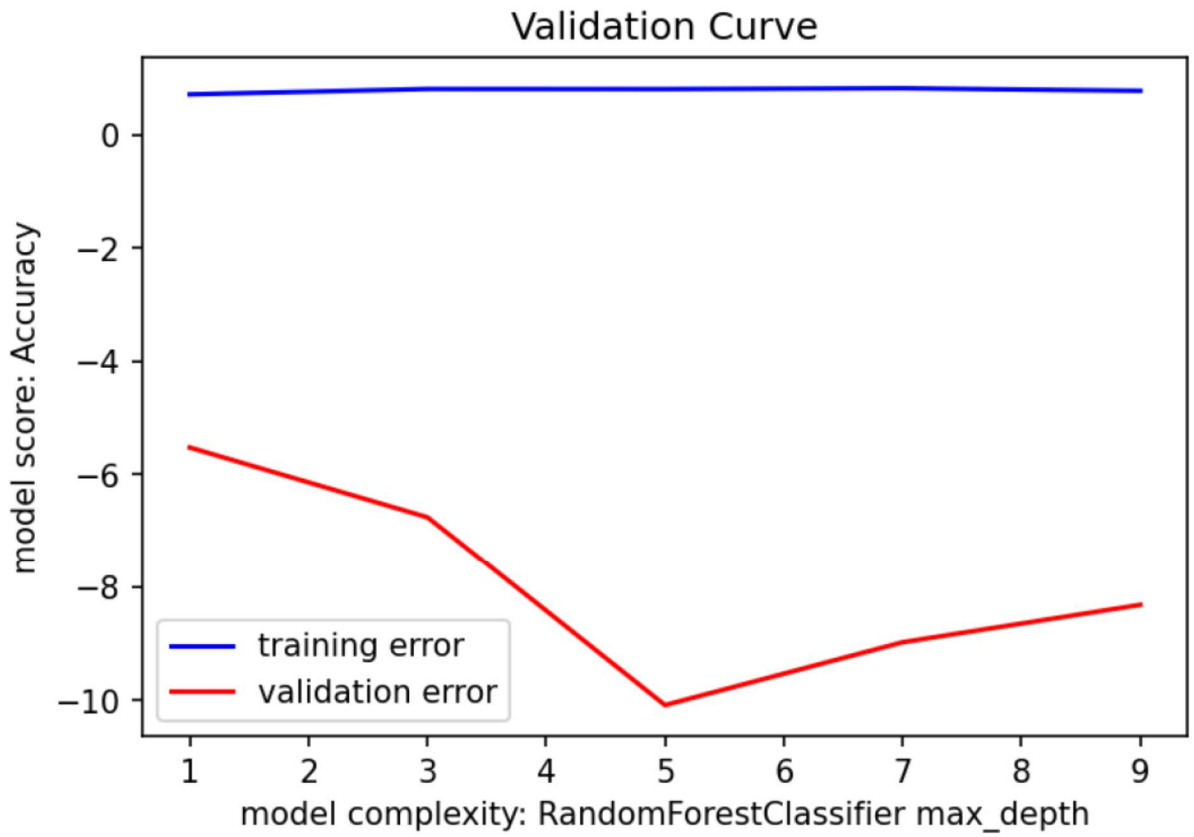
```
# Plot of feature importances from pure Random Forest Regressor
%matplotlib inline
import matplotlib.pyplot as plt
# Get feature importances
encoder = pipeline1.named_steps['onehotencoder']
encoded = encoder.transform(X_train)
rf = pipeline1.named_steps['randomforestregressor']
importances1 = pd.Series(rf.feature_importances_, encoded.columns)
# Plot feature importances
n = 12
plt.figure(figsize=(10,n/2))
plt.title(f'Top {n} features pipeline1')
importances1.sort_values()[-n:].plot.barh(color='grey');
```




Surprisingly, the land area of the capital city is the most important feature with the top four features together carrying the majority of the total feature importance.

The corresponding validation curve was calculated as follows:

```
# Generate validation curves
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import validation_curve
pipeline2 = make_pipeline(
    ce.OrdinalEncoder(),
    SimpleImputer(),
    RandomForestRegressor()
)

depth = range(1, 10, 2)
train_scores, val_scores = validation_curve(
    pipeline2, X_train, y_train,
    param_name='randomforestregressor__max_depth',
    param_range=depth,
    cv=3,
    n_jobs=-1
)

plt.figure(dpi=150)
plt.plot(depth, np.mean(train_scores, axis=1), color='blue', label='training error')
plt.plot(depth, np.mean(val_scores, axis=1), color='red', label='validation error')
plt.title('Validation Curve')
plt.xlabel('model complexity: RandomForestClassifier max_depth')
plt.ylabel('model score: Accuracy')
plt.legend();
```




The resulting validation curve indicates that the regression does not fit the data well as there is not a maximum depth (or level of complexity) which yields a best score. This can be expected given that the number of features is the same as the number of samples (states) in the training data set (https://medium.com/@jennifer.zzz/more-features-than-data-points-in-linear-regression-5bcabba6883e).

Permutation importances were used to further limit the feature set.

```
# Using Eli5 library which does not work with pipelines
transformers = make_pipeline(
    ce.OneHotEncoder(use_cat_names=True),
    SimpleImputer(strategy='most_frequent')
)

X_train_transformed = transformers.fit_transform(X_train)
X_val_transformed = transformers.transform(X_val)

model1 =     RandomForestRegressor(bootstrap=True, ccp_alpha=0, criterion='mse',
                       max_depth=3, max_features='auto', max_leaf_nodes=None,
                       max_samples=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=4,
                       min_samples_split=2, min_weight_fraction_leaf=0,
                       n_estimators=36, n_jobs=None, oob_score=False,
                       random_state=0, verbose=0, warm_start=False)

model1.fit(X_train_transformed, y_train)
```

```
# Get permutation importances
! pip install eli5
from eli5.sklearn import PermutationImportance
import eli5

permuter = PermutationImportance(
    model1,
    scoring='r2',
    n_iter=2,
    random_state=42
)

permuter.fit(X_val_transformed, y_val)
feature_names = X_val.columns.tolist()

eli5.show_weights(
    permuter,
    top=None, # show permutation importances for all features
    feature_names=feature_names
)
```

Of 38 features, 14 permutation importances were found to have weights greater than zero:

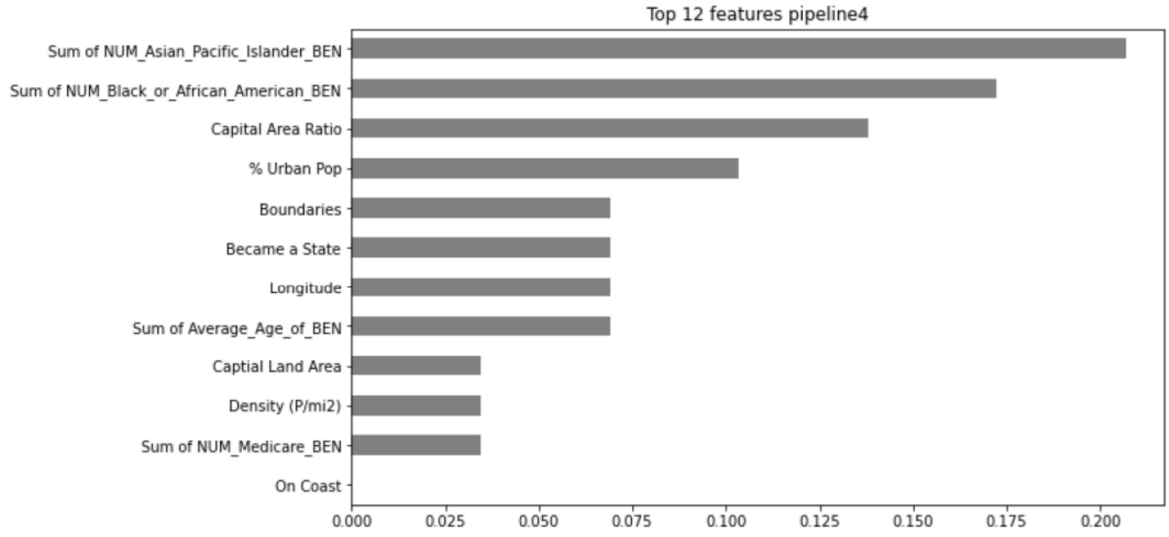
| Weight | Feature |
|---|---|
| 0.1883 ± 0.0854 | Captial Land Area |
| 0.0239 ± 0.2251 | Longitude |
| 0.0091 ± 0.0014 | Density (P/mi2) |
| 0.0063 ± 0.0098 | Sum of NUM_Black_or_African_American_BEN |
| 0.0049 ± 0.0032 | Sum of NUM_Asian_Pacific_Islander_BEN |
| 0.0045 ± 0.0043 | Latitude |
| 0.0044 ± 0.0104 | Became a State |
| 0.0024 ± 0.0001 | Capital Area Ratio |
| 0.0022 ± 0.0019 | Sum of Average_Age_of_BEN |
| 0.0021 ± 0.0014 | Boundaries |
| 0.0015 ± 0.0025 | Sum of NUM_BEN_With_Race_Not_Elsewhere_Classified |
| 0.0007 ± 0.0016 | % Urban Pop |
| 0.0005 ± 0.0008 | On Coast |
| 0.0003 ± 0.0014 | Sum of NUM_Medicare_BEN |

Note that Capital Land Area continues to have the highest feature importance, while other remaining features have shifted in their relative importance. Seven of the 38 features have zero weight and therefore no importance. Seventeen of the 38 features have negative weights. Removing these features served to improve model performance (https://explained.ai/rf-importance/).

Thus 14 features with weights greater than zero were selected for use in additional regressions.

```
# Remove features of 0 importance
zero_importance = 0.0
mask = permuter.feature_importances_ > zero_importance
features1 = X_train.columns[mask]
X_train = X_train[features1]
print('Shape after removing features:', X_train.shape)
```

Applying the Random Forest Regressor to the reduced feature set resulted in the $r^2$ training accuracy dropping from 0.3014 to 0.2698 and the $r^2$ validation accuracy increasing from 0.0694 to 0.1186, indicating a significant increase in the accuracy of the model. The resulting feature importances were plotted:




Now Capital Land Area has slipped way down in importance and some other features that we might have expected to be important, such as features presenting the sum or total number of Medicare/Medicaid beneficiaries in certain ethnic groups, have increased in importance.

To further improve the accuracy of the model, Recursive Feature Elimination was applied. All possible values for number of selected features were tried, leading to the conclusion that the best improvement in validation accuracy could be found when the number of selected features is set to be seven.

```
# Recursive Feature Elimination
from sklearn.feature_selection import RFE, f_regression
from sklearn.model_selection import StratifiedKFold

rfr =     RandomForestRegressor(bootstrap=True, ccp_alpha=0,
                          max_depth=3, max_features='auto', max_leaf_nodes=None,
                          max_samples=None, min_impurity_decrease=0.0,
                          min_impurity_split=None, min_samples_leaf=4,
                          min_samples_split=2, min_weight_fraction_leaf=0,
                          n_estimators=36, n_jobs=None, oob_score=False,
                          random_state=0, verbose=0, warm_start=False)

#Selecting 7 features turns out to give maximum validation accuracy
number_selected_features = 7
rfe = RFE(rfr, n_features_to_select=number_selected_features, verbose =3)
rfe.fit(X_train,y_train)
```

```
rfe_support = rfe.get_support()
rfe_feature = X_train.loc[:,rfe_support].columns.tolist()
print(str(len(rfe_feature)), 'selected features')
```

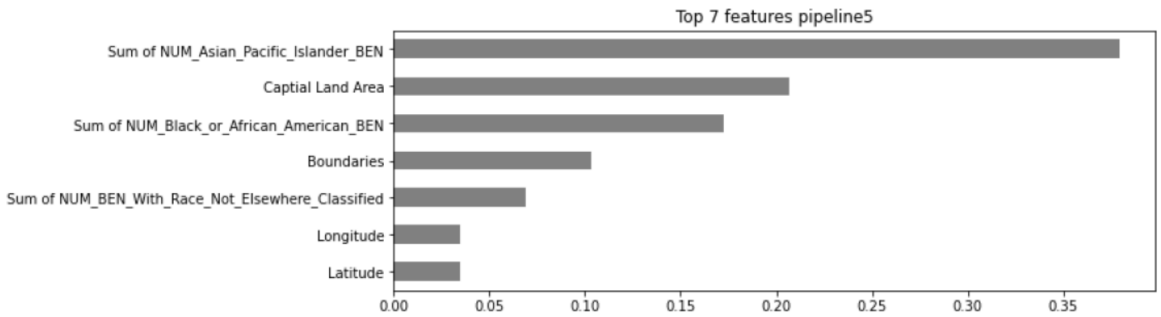
```
from sklearn.metrics import mean_squared_error, r2_score

# Coefficient of determination r2 for the training set
pipeline_score = rfe.score(X_train,y_train)
print("Coefficient of determination r2 for the training set.: ", pipeline_score)

# Coefficient of determination r2 for the validation set
pipeline_score = rfe.score(X_val,y_val)
print("Coefficient of determination r2 for the validation set.: ", pipeline_score)

# The mean squared error
y_pred = rfe.predict(X_val)
print("Mean squared error: %.2f"% mean_squared_error(y_val, y_pred))
```

This indeed improved the performance of the model, with the $r^2$ training accuracy increasing from 0.2698 to 0.2762 and the $r^2$ validation accuracy increasing from 0.1186 to 0.1411, indicating a significant increase in the accuracy of the model. The resulting feature importances were plotted:

```
# Retain only features with highest importance from RFE
X_train_rfe_select = X_train[rfe_feature]
X_val_rfe_select = X_val[rfe_feature]
print('Shape after removing features:', X_train_rfe_select.shape, X_val_rfe_select.shape)
```

```
# Random forest classifier after RFE Feature Selection on Reduced Feature Set

pipeline5 = make_pipeline(
    ce.OneHotEncoder(use_cat_names=True),
    SimpleImputer(strategy = 'most_frequent'),
    RandomForestRegressor(bootstrap=True, ccp_alpha=0,
                          max_depth=3, max_features='auto', max_leaf_nodes=None,
                          max_samples=None, min_impurity_decrease=0.0,
                          min_impurity_split=None, min_samples_leaf=4,
                          min_samples_split=2, min_weight_fraction_leaf=0,
                          n_estimators=36, n_jobs=None, oob_score=False,
                          random_state=0, verbose=0, warm_start=False)
)

# Fit on train, score on val
pipeline5.fit(X_train_rfe_select, y_train);
```

```
pipeline5.fit(X_val_rfe_select, y_val)
# Plot of features
%matplotlib inline
import matplotlib.pyplot as plt

# Get feature importances
encoder = pipeline5.named_steps['onehotencoder']
encoded = encoder.transform(X_val_rfe_select)
rf = pipeline5.named_steps['randomforestregressor']
importances3 = pd.Series(rf.feature_importances_, encoded.columns)

# Plot feature importances
n = number_selected_features
plt.figure(figsize=(10,n/2))
plt.title(f'Top {n} features pipeline5')
importances3.sort_values()[-n:].plot.barh(color='grey');
```



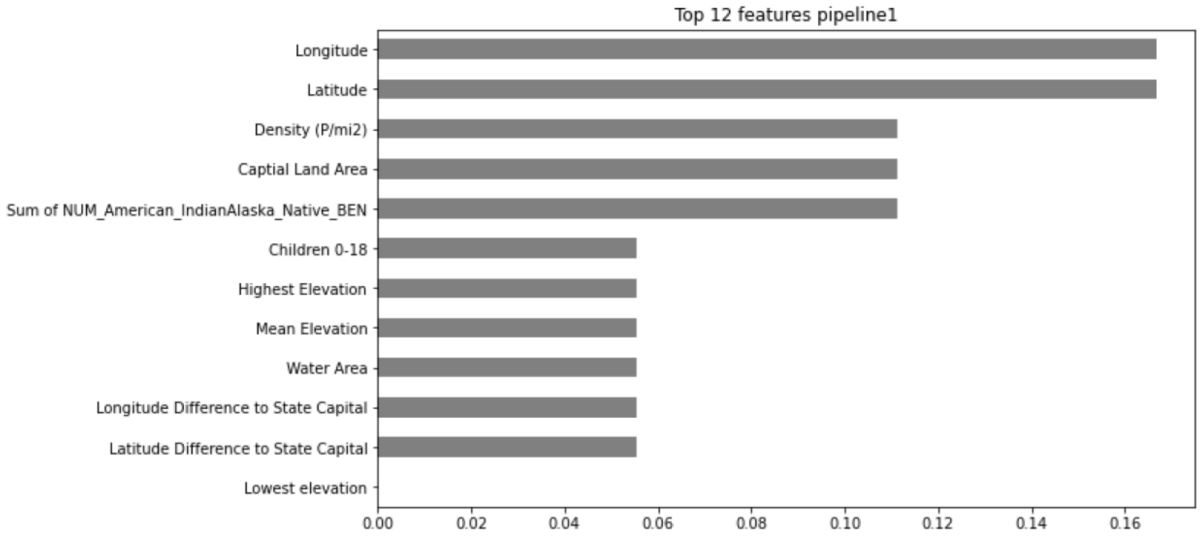
An analogous process to the one that was used for the “a” parameter is used for the “b” parameter, beginning with using Grid Search Cross Validation to find the optimum hyper-parameters for Random Forest Regression with regard to the “b” parameter.

```
# Optimizing Hyperparameters
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

# Define classifier
forest = RandomForestRegressor(random_state = 1)

# Parameters to fit

max_depth = [0.95, 1.0, 1.05]
n_estimators = [16, 18, 20]
min_samples_split = [1.5, 2, 2.5]
min_samples_leaf = [3.5, 4, 4.5]
max_leaf_nodes = [None]
max_features = ['auto']
ccp_alpha = [0.0, 0.05, 0.1]
min_weight_fraction_leaf = [0.0, 0.05, 0.1]

hyperF = dict(n_estimators = n_estimators, max_depth = max_depth,
              min_samples_split = min_samples_split,
              min_samples_leaf = min_samples_leaf,
              max_leaf_nodes = max_leaf_nodes,
              max_features = max_features,
              ccp_alpha=ccp_alpha,
              min_weight_fraction_leaf=min_weight_fraction_leaf)

gridF = GridSearchCV(forest, hyperF, cv = 3, verbose = 10,
                     scoring='r2', return_train_score=True,
                     n_jobs = -1)
bestF = gridF.fit(X_train, y_train)

# Output best accuracy and best parameters
print('The score achieved with the best parameters = ', gridF.best_score_, '\n')
print('The parameters are:', gridF.best_params_)
```

The optimal parameters were then used to perform the initial Random Forest Regression for the normed positive test data.

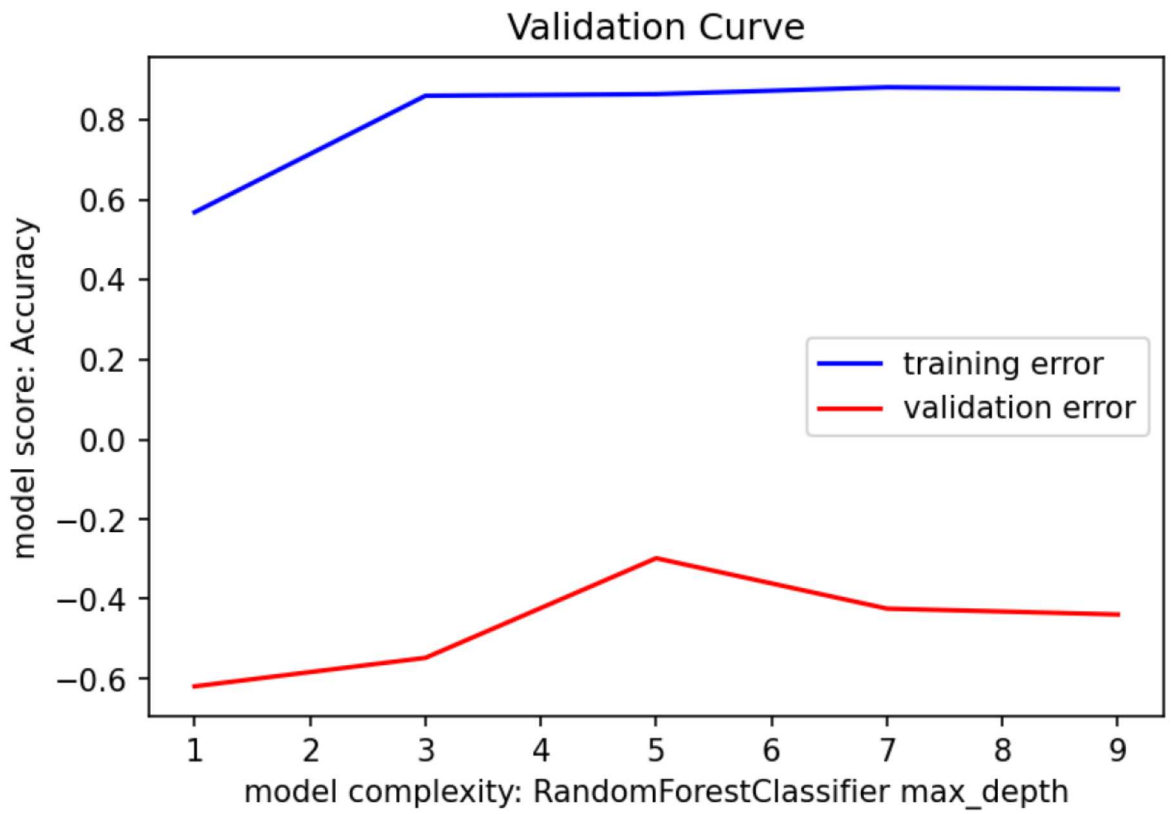
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import make_pipeline
import category_encoders as ce
from sklearn.impute import SimpleImputer

pipeline1 = make_pipeline(
    ce.OneHotEncoder(use_cat_names=True),
    SimpleImputer(strategy='mean'),
    RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=1, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=4,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=18, n_jobs=None, oob_score=False,
                      random_state=0, verbose=0, warm_start=False))

pipeline1.fit(X_train, y_train)

# Get the model's training accuracy
print("Training Accurary:  R^2 = ", pipeline1.score(X_train,y_train))

# Get the model's validation accuracy
print('Validation Accuracy:  R^2 = ', pipeline1.score(X_val, y_val))
```

yielding a training accuracy of r2 = 0.3807, a validation accuracy of r2 = -0.1842, and the following feature importances:




Surprisingly, the latitude and longitude of the state are the most important features with the top five features together carrying about 67% of the total feature importance.

The corresponding validation curve was calculated:



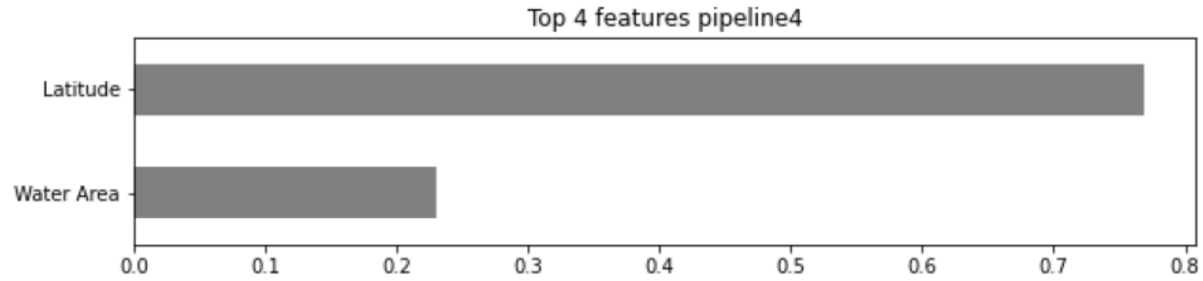

The resulting validation curve indicates that the regression does not fit the data well as there is not a maximum depth (or level of complexity) which yields a best score. Again, this can be expected given that the number of features is the same as the number of samples (states) in the training data set.

As was the case for the “a” parameter, permutation importances were used to further limit the feature set, but here using the optimal hyperparameters that had been selected for the “b” parameter target. Of 38 features, 2 permutation importances were found to have weights greater than zero:

| Weight | Feature |
|---|---|
| 0.1227 ± 0.0096 | Latitude |
| 0.0115 ± 0.0067 | Water Area |

Note that Latitude continues to have the highest feature importance, while Water Area has markedly increased its relative importance.

Applying the Random Forest Regressor to the two features selected by permutation importances resulted in the $r^2$ training accuracy dropping from 0.3807 to 0.2711 and the $r^2$ validation accuracy increasing from -0.1842 to 0.1883, indicating a significant increase in the accuracy of the model. The resulting feature importances were plotted:




Here the importance of Latitude is roughly 3.5 times that of Water Area.

*Population Normalized Death Parameters c and d as Targets*:

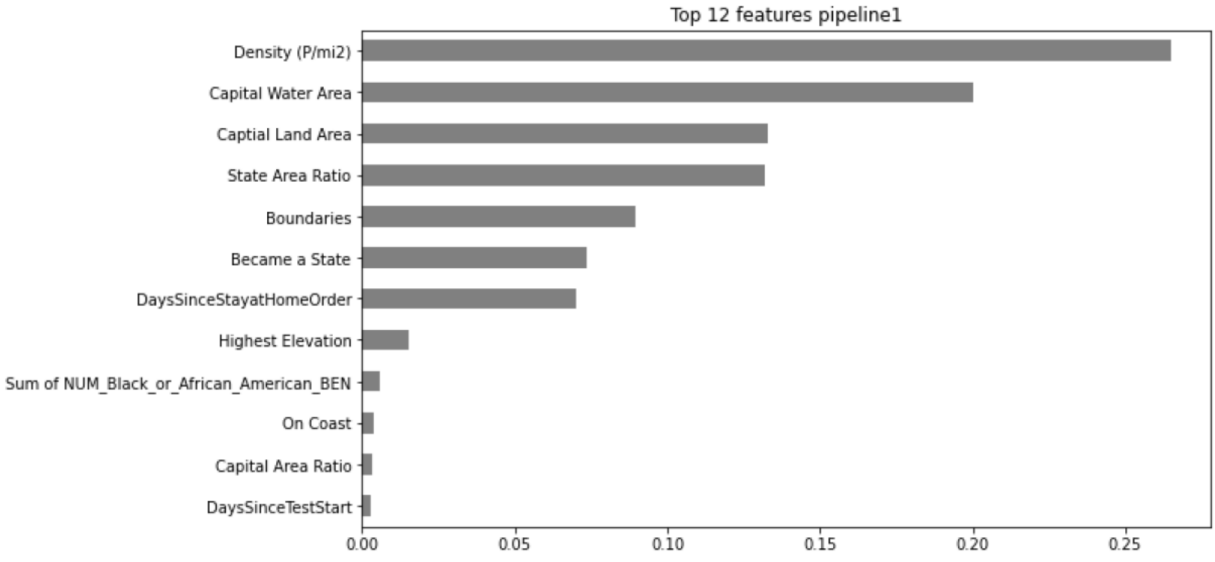
As was the case in working with the population normalized positive test parameters a and b, the data was separated into train and validate sets using a random 75/25 split, resulting in 38 rows in the train set and 13 rows in the validate set. Grid Search Cross Validation was used to find the optimum hyper-parameters for Random Forest Regression with regard to the “c” parameter.

```
# Optimizing Hyperparameters
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

# Define classifier
forest = RandomForestRegressor(random_state = 1)

# Parameters to fit

max_depth = [2, 3, 4]
n_estimators = [13, 14, 15]
min_samples_split = [1.5, 2, 2.5]
min_samples_leaf = [3.5, 4, 4.5]
max_leaf_nodes = [None]
max_features = ['auto']
ccp_alpha = [0.0, 0.00625, 0.0125]
min_weight_fraction_leaf = [0.0, 0.00625, 0.0125]

hyperF = dict(n_estimators = n_estimators, max_depth = max_depth,
              min_samples_split = min_samples_split,
              min_samples_leaf = min_samples_leaf,
              max_leaf_nodes = max_leaf_nodes,
              max_features = max_features,
              ccp_alpha=ccp_alpha,
              min_weight_fraction_leaf=min_weight_fraction_leaf)

gridF = GridSearchCV(forest, hyperF, cv = 3, verbose = 10,
                     scoring='r2', return_train_score=True,
                     n_jobs = -1)
bestF = gridF.fit(X_train, y_train)

# Output best accuracy and best parameters
print('The score achieved with the best parameters = ', gridF.best_score_, '\n')
print('The parameters are:', gridF.best_params_)
```

The optimal parameters were then used to perform the initial Random Forest Regression for the normed death data.

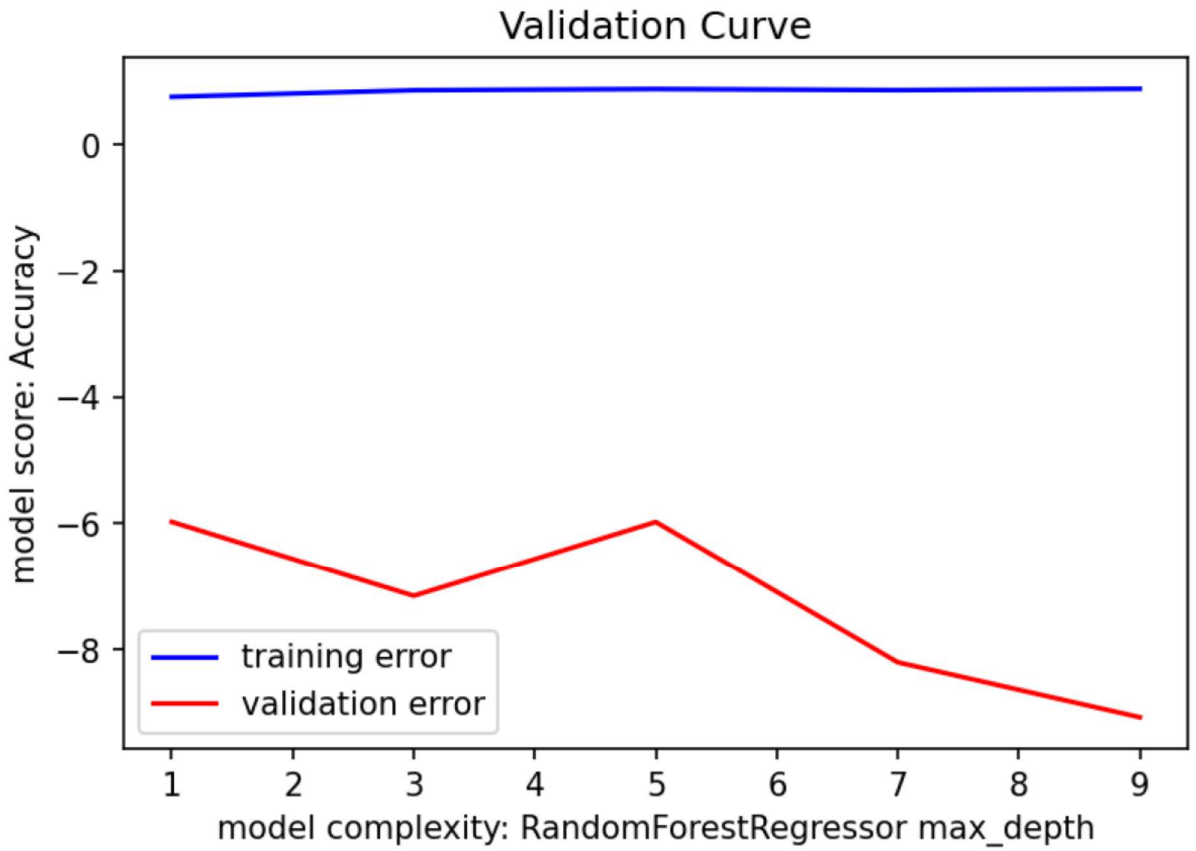
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import make_pipeline
import category_encoders as ce
from sklearn.impute import SimpleImputer

pipeline1 = make_pipeline(
    ce.OneHotEncoder(use_cat_names=True),
    SimpleImputer(strategy='mean'),
    RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=3, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=4,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=14, n_jobs=None, oob_score=False,
                      random_state=0, verbose=0, warm_start=False))

pipeline1.fit(X_train, y_train)

# Get the model's training accuracy
print("Training Accurary:  R^2 = ", pipeline1.score(X_train,y_train))

# Get the model's validation accuracy
ce.OneHotEncoder(use_cat_names=True),
print('Validation Accuracy:  R^2 = ', pipeline1.score(X_val, y_val))
```

yielding a training accuracy of r2 = 0.6583, a validation accuracy of r2 = 0.4334, and the following feature importances:




Not unexpectedly, population density is the most important feature with the top two features making up nearly 50% of the total feature importance.

The corresponding validation curve was calculated:




The resulting validation curve indicates that the regression does not fit the data well as there is not a maximum depth (or level of complexity) which yields a best score. This can be expected given that the number of features is the same as the number of samples (states) in the training data set.

Permutation importances were again used to further limit the feature set. Of 38 features, 11 permutation importances were found to have weights greater than zero:

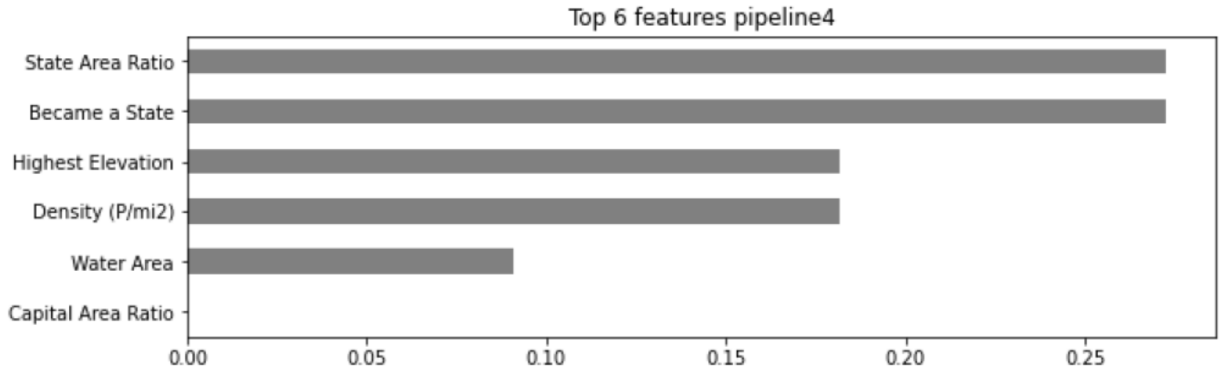
| Weight | Feature |
|---|---|
| 0.4793 ± 0.2232 | Density (P/mi2) |
| 0.1685 ± 0.0511 | Became a State |
| 0.1509 ± 0.0306 | State Area Ratio |
| 0.0270 ± 0.0581 | Highest Elevation |
| 0.0130 ± 0.0060 | Capital Area Ratio |
| 0.0111 ± 0.0073 | Sum of NUM_BEN_With_Race_Not_Elsewhere_Classified |
| 0.0043 ± 0.0137 | Sum of NUM_Hispanic_BEN |
| 0.0022 ± 0.0006 | Capital is the Largest City |
| 0.0020 ± 0.0025 | Water Area |
| 0.0014 ± 0.0039 | DaysSinceStayatHomeOrder |
| 0.0001 ± 0.0032 | On Coast |

Note that Density (P/mi2) continues to have the highest feature importance, while other remaining features have shifted in their relative importance.

Applying the Random Forest Regressor to the 11 features selected by permutation importances resulted in the $r^2$ training accuracy dropping from 0.6583 to 0.6369 and the $r^2$ validation accuracy increasing from 0.4334 to 0.6747, indicating that either (https://stats.stackexchange.com/questions/187335/validation-error-less-than-training-error):

The training set had many 'hard' cases to learn
The validation set had mostly 'easy' cases to predict

The resulting feature importances were plotted:




Here Density (P/mi2) has lost a great deal of importance and the two features with the highest importance do not seem to make sense. Also, only five of the 11 features are shown to have non-zero importances.

In order to improve performance, XG Boosting was attempted after having optimized hyperparameters at n_estimators = 13, max_depth = 3, and learning_rate = 0.25:

```
from xgboost import XGBRegressor
pipeline5 = make_pipeline(
    ce.OrdinalEncoder(),
    XGBRegressor(n_estimators=13,
                 max_depth=3, # try deeper trees because of high cardinality categoricals
                 learning_rate=0.25, # try a higher learning rate
                 random_state=42,
                 n_jobs=-1)
)
pipeline5.fit(X_train, y_train);
```

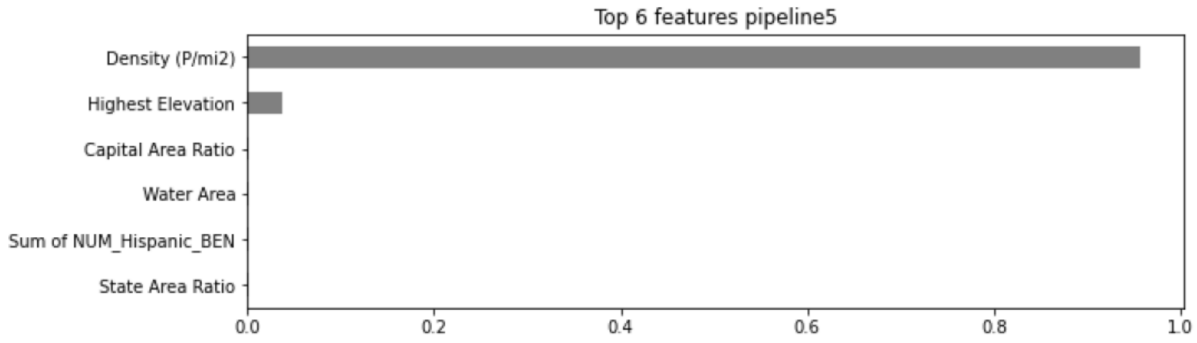
Accuracy scores were calculated:

```
# Coefficient of determination r2 for the training set
pipeline_score = pipeline5.score(X_train,y_train)
print("Coefficient of determination r2 for the training set.: ", pipeline_score)

# Coefficient of determination r2 for the validation set
pipeline_score = pipeline5.score(X_val,y_val)
print("Coefficient of determination r2 for the validation set.: ", pipeline_score)

# The mean squared error
y_pred = pipeline5.predict(X_val)
print("Mean squared error: %.2f"% mean_squared_error(y_val, y_pred))
```

XG Boosting resulted in the $r^2$ training accuracy increasing from 0.6393 to 0.9817 and the $r^2$ validation accuracy decreasing from 0.6747 to 0.5217, providing an improvement over the initial fit, but with validation accuracy less than training accuracy.

Feature importances were calculated and plotted:

```
pipeline5.fit(X_val, y_val)
# Plot of features
%matplotlib inline
import matplotlib.pyplot as plt

# Get feature importances
encoder = pipeline5.named_steps['ordinalencoder']
encoded = encoder.transform(X_val)
rf = pipeline5.named_steps['xgbregressor']
importances3 = pd.Series(rf.feature_importances_, encoded.columns)

# Plot feature importances
n = 6
plt.figure(figsize=(10,n/2))
plt.title(f'Top {n} features pipeline5')
importances3.sort_values()[-n:].plot.barh(color='grey');
```

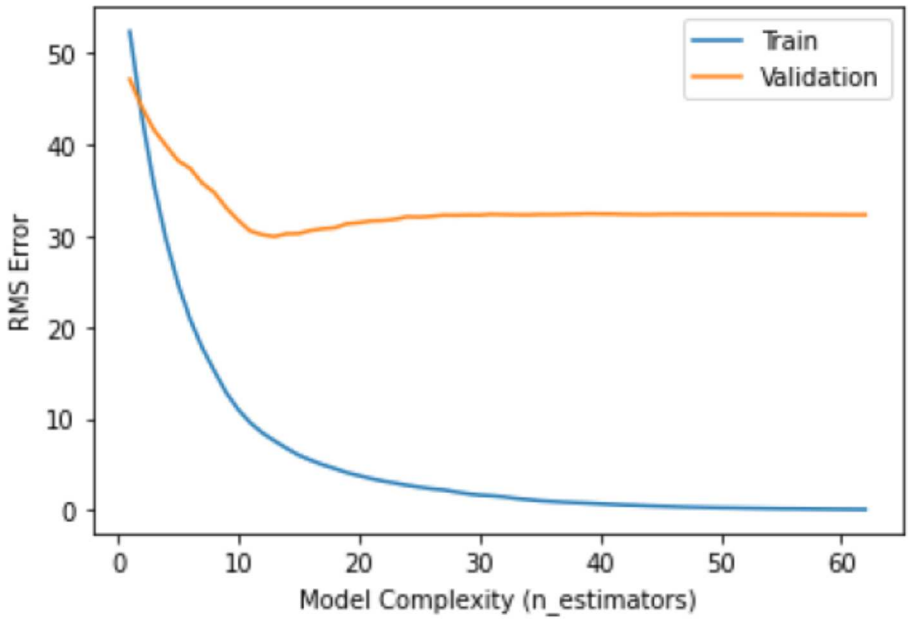



Thus, after the XG Boost regression was performed, only two features were found to have non-zero importances. Density (P/mi2) is again the most important feature and has about 97% of total feature importance.

The choice of n_estimators = 13 was confirmed by computing and plotting classification error versus model complexity:

```
# Gradient boosting using XGboost with 1000 estimators
encoder = ce.OrdinalEncoder()
X_train_encoded = encoder.fit_transform(X_train)
X_val_encoded = encoder.transform(X_val)
X_train.shape, X_val.shape, X_train_encoded.shape, X_val_encoded.shape
```

```
eval_set = [(X_train_encoded, y_train),
            (X_val_encoded, y_val)]

model2 = XGBRegressor(
          n_estimators=1000, # <= 1000 trees, depends on early stopping
          max_depth=3, # try deeper trees because of high cardinality categoricals
          learning_rate=0.25,
          n_jobs=-1)

model2.fit(X_train_encoded, y_train, eval_set=eval_set, eval_metric='rmse',
          early_stopping_rounds=50)
# Plot the results
results = model2.evals_result()
train_error = results['validation_0']['rmse']
val_error = results['validation_1']['rmse']
epoch = range(1, len(train_error)+1)
plt.plot(epoch, train_error, label='Train')
plt.plot(epoch, val_error, label='Validation')
plt.ylabel('RMS Error')
plt.xlabel('Model Complexity (n_estimators)')
# plt.ylim((0.18, 0.22)) # Zoom in
plt.legend();
```




The minimum of RMS error for the validation data set occurs about where n_estimators = 13.

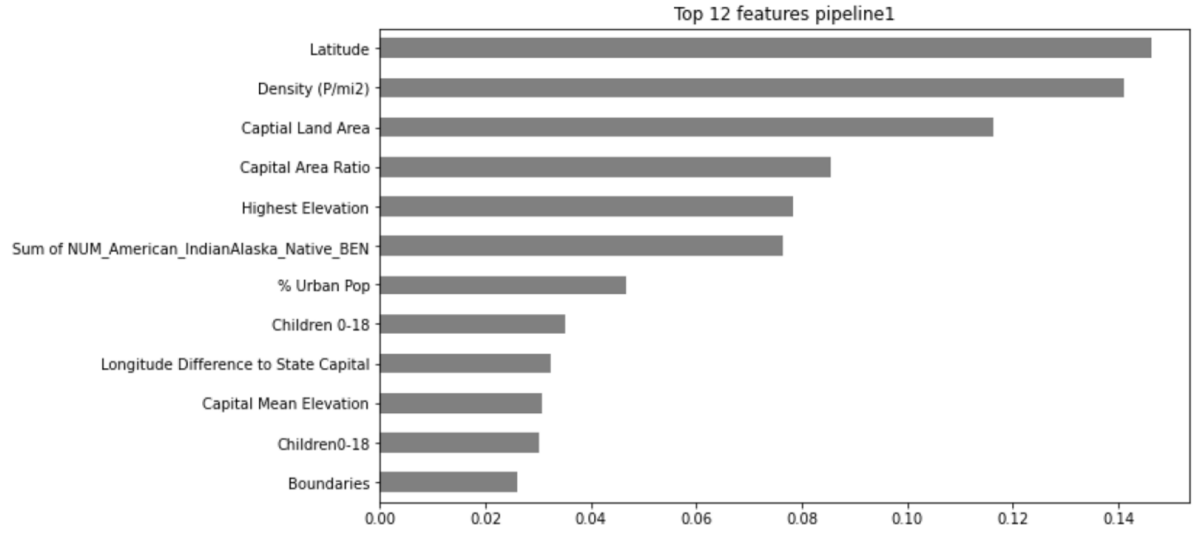
Analogous to how work was done for the “c” parameter, Grid Search Cross Validation was used to find the optimum hyper-parameters for Random Forest Regression with regard to the “d” parameter.

```
# Optimizing Hyperparameters
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

# Define classifier
forest = RandomForestRegressor(random_state = 1)

# Parameters to fit

max_depth = [2, 3, 4]
n_estimators = [28, 29, 30]
min_samples_split = [1.5, 2, 2.5]
min_samples_leaf = [3.5, 4, 4.5]
max_leaf_nodes = [None]
max_features = ['auto']
ccp_alpha = [0.0, 0.00625, 0.0125]
min_weight_fraction_leaf = [0.0, 0.00625, 0.0125]

hyperF = dict(n_estimators = n_estimators, max_depth = max_depth,
              min_samples_split = min_samples_split,
              min_samples_leaf = min_samples_leaf,
              max_leaf_nodes = max_leaf_nodes,
              max_features = max_features,
              ccp_alpha=ccp_alpha,
              min_weight_fraction_leaf=min_weight_fraction_leaf)

gridF = GridSearchCV(forest, hyperF, cv = 3, verbose = 10,
                     scoring='r2', return_train_score=True,
                     n_jobs = -1)
bestF = gridF.fit(X_train, y_train)

# Output best accuracy and best parameters
print('The score achieved with the best parameters = ', gridF.best_score_, '\n')
print('The parameters are:', gridF.best_params_)
```

The optimal parameters were then used to perform the initial Random Forest Regression for the normed death data.

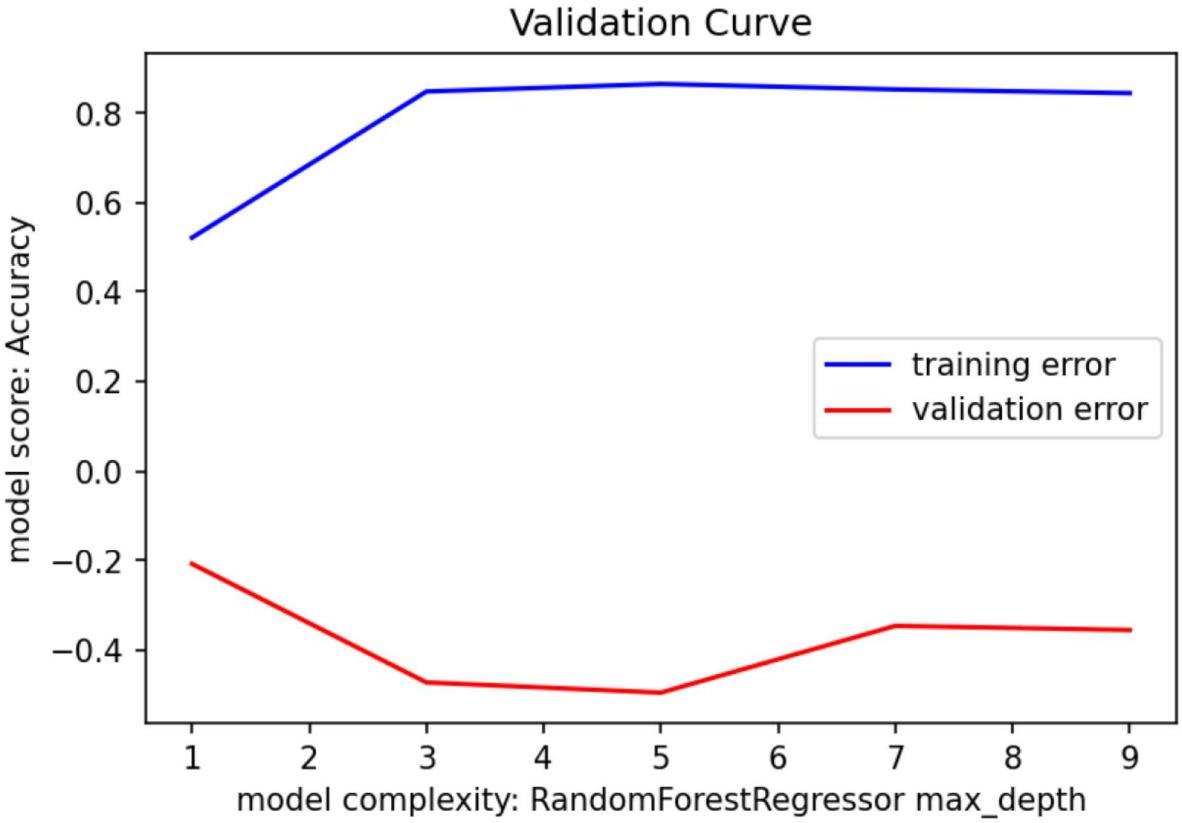
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import make_pipeline
import category_encoders as ce
from sklearn.impute import SimpleImputer

pipeline1 = make_pipeline(
    ce.OneHotEncoder(use_cat_names=True),
    SimpleImputer(strategy='mean'),
    RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=3, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=4,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=29, n_jobs=None, oob_score=False,
                      random_state=0, verbose=0, warm_start=False))

pipeline1.fit(X_train, y_train)

# Get the model's training accuracy
print("Training Accurary:  R^2 = ", pipeline1.score(X_train,y_train))

# Get the model's validation accuracy
print('Validation Accuracy:  R^2 = ', pipeline1.score(X_val, y_val))
```

yielding a training accuracy of $r^2 = 0.6195$, a validation accuracy of $r^2 = 0.4828$, and the following feature importances:




Not unexpectedly, population density is the second most important feature with the top four features making up over 50% of the total feature importance.

The corresponding validation curve was calculated:




The resulting validation curve indicates that the regression does not fit the data well as there is not a maximum depth (or level of complexity) which yields a best score. This can be expected given that the number of features is the same as the number of samples (states) in the training data set.

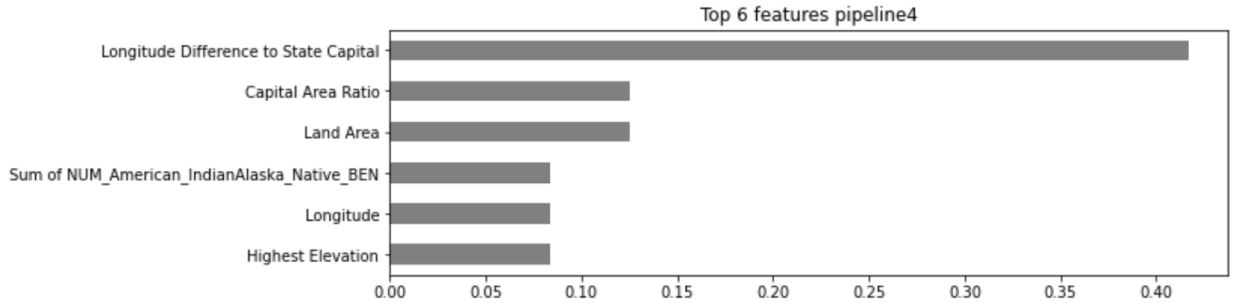
Permutation importances were again used to further limit the feature set. Of 38 features, 22 permutation importances were found to have weights greater than zero:

| Weight | Feature |
|---|---|
| 0.1736 ± 0.1030 | Capital Area Ratio |
| 0.1545 ± 0.2241 | Density (P/mi2) |
| 0.0665 ± 0.0669 | Captial Land Area |
| 0.0546 ± 0.0414 | Latitude |
| 0.0284 ± 0.0096 | % Urban Pop |
| 0.0134 ± 0.0002 | Longitude Difference to State Capital |
| 0.0128 ± 0.0150 | Became a State |
| 0.0094 ± 0.0061 | DaysSinceStayatHomeOrder |
| 0.0072 ± 0.0004 | Latitude Difference to State Capital |
| 0.0068 ± 0.0193 | Sum of NUM_American_IndianAlaska_Native_BEN |
| 0.0060 ± 0.0121 | Number of bordering states |
| 0.0056 ± 0.0070 | Boundaries |
| 0.0046 ± 0.0216 | Capital Mean Elevation |
| 0.0034 ± 0.0172 | Longitude |
| 0.0027 ± 0.0078 | Highest Elevation |
| 0.0022 ± 0.0043 | Land Area |
| 0.0021 ± 0.0068 | Sum of Average_Age_of_BEN |
| 0.0020 ± 0.0000 | Sum of NUM_Black_or_African_American_BEN |
| 0.0013 ± 0.0088 | DaysSinceTestStart |
| 0.0013 ± 0.0031 | State Area Ratio |
| 0.0010 ± 0.0050 | On Coast |
| 0.0009 ± 0.0055 | Capital Water Area |

Note that Capital Area Ratio and Latitude have exchanged their feature importance ranking, while Density (P/mi2) and Capital Land Area their relative importances in second and third place respectively.

Applying the Random Forest Regressor to the 22 features selected by permutation importances resulted in the $r^2$ training accuracy increasing from 0.6195 to 0.6266 and the $r^2$ validation accuracy increasing from 0.4828 to 0.4898.

The resulting feature importances were plotted:



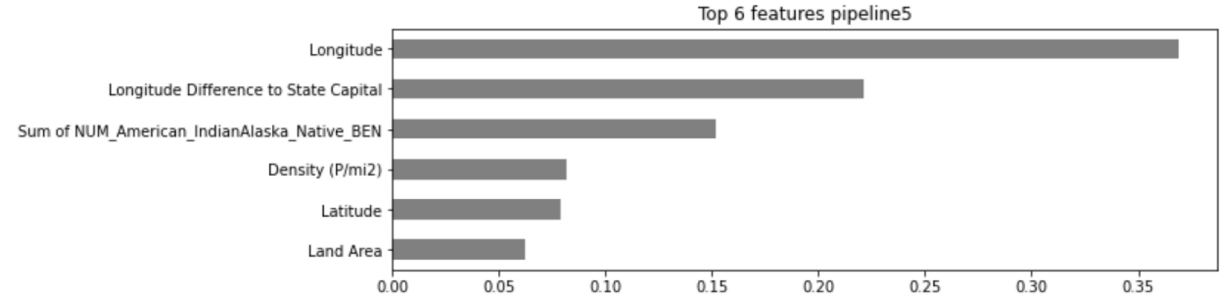

Here Density (P/mi2) has lost a great deal of importance and the three features with the highest importance do not seem to make sense.

In order to improve performance, XG Boosting was attempted after having optimized hyperparameters at n_estimators = 15, max_depth = 1, and learning_rate = 0.41:

```
pipeline5 = make_pipeline(
    ce.OrdinalEncoder(),
    XGBRegressor(n_estimators=15,
                 max_depth=1,
                 learning_rate=0.41, # try a higher learning rate
                 random_state=42,
                 n_jobs=-1)
)
pipeline5.fit(X_train, y_train);
```

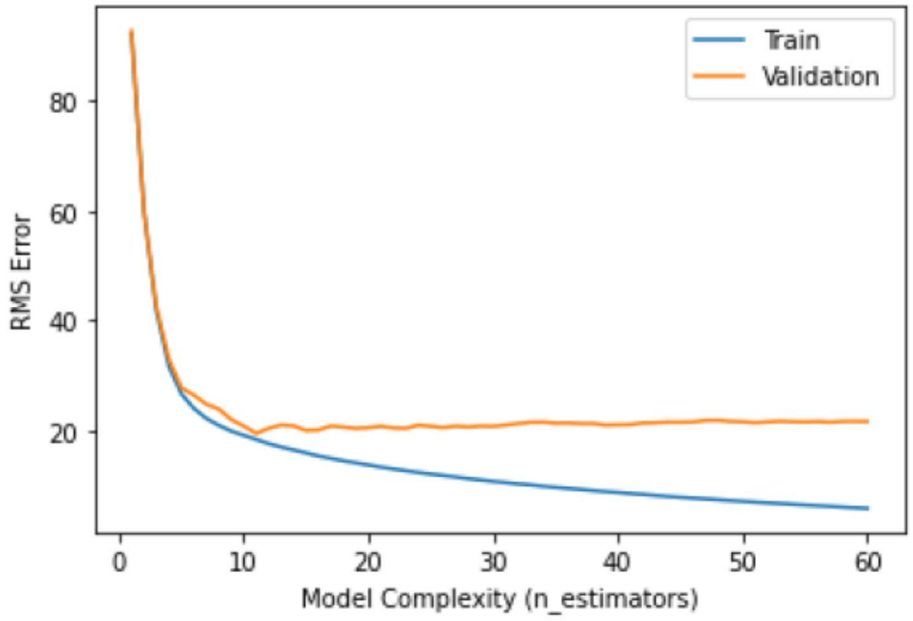
Accuracy scores were calculated in the same way as was done for the “c” parameter. XG Boosting resulted in the $r^2$ training accuracy increasing from 0.6266 to 0.7839 and the $r^2$ validation accuracy decreasing from 0.4898 to 0.6058, providing the best fit.

Feature importances were calculated and plotted:




The XG Boost Regression resulted in feature importances dropping off more rapidly. Longitude is now the most important feature, with about 37% of the total of feature importances. Density (P/mi2) is now the fourth most important feature.

The choice of n_estimators = 15 was confirmed by computing and plotting classification error versus model complexity using the same methodology as was used for the "c" parameter, except with modified hyperparameters (n_estimators = 15, max_depth = 1, and learning_rate = 0.41).




The minimum of RMS error for the validation data set occurs about where n_estimators = 15.

Summary:

The results of the regression analysis above are summarized in the following table.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Population Normalized Positive Tests** | | | | | | | | | | | | |
| | | | **a** | | | **b** | | | | | **fit rank** | **r^2** |
| Distribution | Mean | | 16.215254 | | | -100.951881 | | | | | 1.769231 | 0.995682 |
| | SD | | 31.801661 | | | 25.545128 | | | | | 1.09572 | 0.004749 |
| | | | | | | | | | | | | |
| Best Fit r^2 | Training | | 0.2762 | | | 0.2711 | | | | | | |
| | Validation | | 0.1411 | | | 0.1883 | | | | | | |
| | | | | | | | | | | | | |
| Ranked Feature Importances | | | Sum of NUM_Asian_Pacific_Islander_BEN | 0.38 | | Latitude | 0.78 | | | | | |
| | | | Capital Land Area | 0.21 | | Water Area | 0.22 | | | | | |
| | | | Sum of NUM_Black_or_African_American_BEN | 0.18 | | | | | | | | |
| | | | Boundaries | 0.11 | | | | | | | | |
| | | | Sum of NUM_BEN_With_Race_Not_Elsewhere_Classified | 0.07 | | | | | | | | |
| | | | | | | | | | | | | |
| **Population Normalized Death** | | | | | | | | | | | | |
| | | | **c** | | | **d** | | | | | **fit rank** | **r^2** |
| Distribution | Mean | | 28.922502 | | | -142.879078 | | | | | 2.098039 | 0.984584 |
| | SD | | 53.235594 | | | 33.811201 | | | | | 2.156431 | 0.021142 |
| | | | | | | | | | | | | |
| Best Fit r^2 | Training | | 0.9817 | | | 0.7839 | | | | | | |
| | Validation | | 0.5217 | | | 0.6058 | | | | | | |
| | | | | | | | | | | | | |
| Ranked Feature Importances | | | Density (P/mi2) | 0.98 | | Longitude | 0.38 | | | | | |
| | | | Highest Elevation | 0.02 | | Longitude Difference to State Capital | 0.22 | | | | | |
| | | | | | | Sum of NUM_American_IndianAlaska_Native_BEN | 0.15 | | | | | |
| | | | | | | Density (P/mi2) | 0.08 | | | | | |
| | | | | | | Latitude | 0.07 | | | | | |
| | | | | | | Land Area | 0.06 | | | | | |

The fit rank and $r^2$ values indicate that the fit of both the population normed positive test time series for the states and the population normed death time series for the states were well represented by the functional forms: a*exp(b/t) and c*exp(d/t) respectively.

The distribution of values for the parameters “a” and “c” is relatively tighter than the distribution of values for the parameters for “c” and “d.” Since “a” and “c” represent the maximum values for the population normed positive test results and the population normed deaths respectively, the states ultimately achieve nearly same maximum for their population normed positive tests, but the rates at which they get there (influenced largely by the values of “b”) are quite different and the states ultimately achieve nearly the same maximum for their population normed deaths, but the rates at which they get there (influenced largely by the values of “d”) are also quite different.

It is noteworthy that the best regression validation scores for the “a” and “b” parameter targets are comparable, as are the best regression validation scores for the “c” and “d” parameter targets. Thus, parameter targets from the same time series representation yield consistent regression performances against the same feature set. Also, the best training and validation scores for regression on the population normed positive test parameter targets are much worse then the best training and validation scores for regression on the population normed death parameter targets against the same feature set. This could be due to the diversity of test types that yield positive test results as well as the uneven rate at which testing occurs. The metric of death is therefore much less ambiguous than the metric of a positive test.

Although the regression started with 38 uncorrelated features, the regressions for the “b” and “c” targets ultimately had only two important features and the regressions for “a” and “d” ultimately had only five and six features respectively.

The “a” parameter target, which determines the maximum values of the state population normed positive test curves, has important features which are related to the number of people in a state in certain ethnic groups that are generally known to have compromised access to healthcare services. In total, these features are responsible for 63% of total feature importances. The area of land for the capital of a state and number of boundaries a state has with other states account for the remainder of feature importances for the “a” parameter target. Perhaps the number of boundaries has impact through facilitating spread from states with higher Covid-19 infection prevalence to states with lower Covid-19 infection prevalence. The impact of capital land area is much harder to explain.

The “b” parameter target, which determines the rate of increase in population normed positive tests, has only two important features. Latitude accounts for nearly 80% of the total feature importances and could represent impact based upon variations in climate and/or UV light intensity. Water area accounts for about 20% of total feature importances and could represent the propensity for people to congregate around bodies of water and thereby violate social distancing.

The “c” parameter target, which determines the maximum values of the state population normed death curves, has population density responsible for 98% of feature importances, consistent with the impact of higher virus spread in areas with greater population density on the number of Covid-19 related deaths.

The “d” parameter target, which determines the rate of increase in population normed deaths, has only six important features. The greatest importance is attached to longitude which may reflect the higher rise in Covid-19 infection related deaths in states on the coasts. The longitude difference between the center of a state and the center of the state capital may be important because state capitals are generally population centers and, as the difference grows larger, the closer the state capital is to at least

one of the state boundaries. The number of American Indians or Alsaka natives, representing a known vulnerable population, as well as population density, are also important features influencing the rate at which deaths due to Corona-19 infection increase.

Summary:

This article demonstrates the feasibility and the merits of:

- Fitting the time series data related to both population normed positive tests and population normed deaths to the functional form: a*exp(b/t), thus capturing those two sets of data in a way that is "frozen in time."
- Making use of the resulting four parameters (a, b, c, and d) as targets for regression to identify which characteristics or features of the states are most important in determining the values of those four parameters and hence the shape of both the population normed positive test and population normed death curves.

The quality of results in this article were limited by:

1) Positive test numbers were derived from a diversity of testing methodologies
2) Medicare/Medicaid data-based features were not population normed
3) The dataset was small because it was based upon states, of which there are only 50 plus DC
4) A multitude of important factors which have been discovered during the course of writing this paper which were not included in features such as number of meat processing plants in a state

In order to be timely against the backdrop of a rapidly evolving Covid-19 pandemic, it was decided to publish this work at this stage. Next steps for this work could and arguably should be to:

1) Re-run the analysis done in this article to include data beyond April 13, 2020, particularly to test the robustness of fitting time series data to the a*exp(b/t) (or some other consistent) functional form
2) Repeat the analysis done in this article using population normed Medicare/Medicaid data-based features
3) Apply the approach used in this article to county-level data, providing an opportunity both to further test the robustness of fitting time series data to the a*exp(b/t) (or some other consistent) functional form and to perform regression on a significantly larger number of samples.

Credits:

Special thanks to the following individuals who provided much of the data for features as well as input toward the final version of this article:

James Macion
Wilton Lam
Stephen Kwok