2D_mu_cc07_group2

November 26, 2021

1 2D MU CC07 Group 2

1.1 Acknowledgements

All members contributed equally to all portions of this 2D project.

1.2 Group Members

1. Aditya 1005395
2. Aishwarya 1005762
3. Clement 1004988
4. Sean 1005122
5. Yuliati 1005644

2 Task 1 Text Only

2.1 Variable Selection & Definitions

To predict `new_deaths_smoothed_per_million`, we experimented with the following independent variables. These variables were initially chosen through intellectual conjecture on what variables are most likely to affect `new_deaths_smoothed_per_million`. Subsequently, we elaborated on the steps taken to ensure we picked variables that were the most linearly related to `new_deaths_smoothed_per_million`.

2.1.1 Independent Variables

1. `icu_patients_per_million` = Number of COVID-19 patients in intensive care units (ICUs) on a given day per 1,000,000 people
2. `new_cases_smoothed_per_million` = New confirmed cases of COVID-19 (7-day smoothed) per 1,000,000 people
3. `stringency_index` = Government Response Stringency Index: composite measure based on 9 response indicators including school closures, workplace closures, and travel bans, rescaled to a value from 0 to 100 (100 = strictest response)
4. `median_age` = Median age of the population, UN projection for 2020
5. `extreme_poverty` = Share of the population living in extreme poverty, most recent year available since 2010
6. `people_fully_vaccinated_per_hundred` = Total number of people who received all doses prescribed by the vaccination protocol per 100 people in the total population

7. `people_vaccinated_per_hundred` = Total number of people who received at least one vaccine dose per 100 people in the total population
8. `reproduction_rate` = Real-time estimate of the effective reproduction rate (R) of COVID-19. See https://github.com/crondonm/TrackingR/tree/main/Estimates-Database
9. `human_development_index` = A composite index measuring average achievement in three basic dimensions of human development—a long and healthy life, knowledge and a decent standard of living. Values for 2019, imported from http://hdr.undp.org/en/indicators/137506

2.1.2 Dependent Variable

1. `new_deaths_smoothed_per_million` = New deaths attributed to COVID-19 (7-day smoothed) per 1,000,000 people

2.2 Data pre-processing

2.2.1 Attempt 1 - Forward fill

We attempted to replace missing values using both the forward fill and backward fill techniques. However, this resulted in a much poorer linear correlation between our independent and dependent variables. Section **??**.

2.2.2 Attempt 2 - Dropping missing values

As $> 95\%$ of the data points contain rows that have missing values, we decided to drop the rows with missing values altogether as filling them up through either forward or backward propagation would inevitably skew the data points towards a more unnatural result. This also produced a more favourable linear correlation between our independent and dependent variables which will be shown in our next section.

We processed the data by 1. Extracting out columns containing our independent and dependent variables 2. Dropping rows with missing values

2.3 Univariate Linear Regression

For reliability purposes, a random_state of 100 has been set. `r2` and `adjusted_r2` values have been sorted in ascending order.

	r2	adjusted_r2
`icu_patients_per_million`	0.595933	0.590514
`new_cases_smoothed_per_million`	0.277922	0.268237
`stringency_index`	0.116847	0.105001
`median_age`	0.00456267	-0.00878895
`extreme_poverty`	0.0200249	0.00688066
`people_fully_vaccinated_per_hundred`	0.179166	0.168156
`people_vaccinated_per_hundred`	0.225711	0.215326
`reproduction_rate`	0.044482	0.0316658
`human_development_index`	0.06814	0.0556411

2.3.1 Analysis

Only the number of ICU patients per million had a significant linear correlation with respect to the number of new deaths per million (Smoothed across 7 days).

Vaccination rates, new cases per million (Smoothed across 7 days) and stringency index produced a relatively moderate linear correlation, indicating a moderate degree of correlation between the two variables.

2.4 Multivariate variable selection

Next, we fine-tuned the variables selected based on the following reasons:

2.4.1 1. High linear correlation

We selected the top 3 independent variables based on their `adjusted_r2_score`.

2.4.2 2. Eliminate multicollinearity

Source

Multicollinearity occurs when the independent variables selected are highly correlated with one another. This means that the change in one independent variables results in the change of another, leading to the model fluctuating significantly. Consequently, 1. Finding optimal beta values may be unstable 2. Overfitting may occur 3. Selection of independent variables may be unpredictable

To prevent multicollinearity, we will be dropping independent variables with high linear correlation with each other. Numerical calculations showcasing the high linear correlation between our independent variables may be found Section **??**. Also, for the sake of optimising our multivariate linear regression model, we have chosen to drop the variable with a lower linear correlation with our dependent variable. Hence, the following variables will be dropped: 1. `new_cases_smoothed_per_million` - Strong linear correlation with `icu_patients_per_million` 2. `people_fully_vaccinated_per_hundred` - Strong linear correlation with `people_vaccinated_per_hundred`

2.4.3 3. Removing variables with negligible linear correlation

Similar to 1, we removed variables which Section **??** a linear correlation of < 0.01 with our independent variable.

2.5 Multivariate Linear Regression

To prevent overfitting, we will only be concerned with the `adjusted_r2_score`. For reference, the highest `adjusted_r2_score` through univariate linear regression is 0.592940113924265. Highest value is in **bold**.

	r2	adjusted_r2
icu_patients_per_million, people_vaccinated_per_hundred	0.60807	0.602813
icu_patients_per_million, people_vaccinated_per_hundred, stringency_index	0.608751	**0.603504**

2.5.1 Analysis

While multivariate linear regression did showed an improvement in the `adjusted_r2_score`, **the improvement is arguably slight**.

2.6 What happens if we only have variables with low linear correlation?

For reference, the highest `adjusted_r2_score` amongst the 4 variables listed below, for univariate linear regression, is 0.00835851947117039. Highest value obtained through multivariate linear regression is given in **bold**.

	r2	adjusted_r2
median_age, extreme_poverty	0.0205068	0.00736906
median_age, extreme_poverty, reproduction_rate	0.0634354	0.0508734
median_age, extreme_poverty, reproduction_rate, human_development_index	0.113784	**0.101898**

2.6.1 Analysis

As expected, while we do see an increase in overall `adjusted_r2_score`, the new `adjusted_r2_score` is still low.

2.7 Combining variables with high & low linear correlation

For experimental purposes, we developed a multivariate linear regression model using independent variables with both high and low linear correlation. Highest value obtained is given in **bold**. For reference, the highest `adjusted_r2_score` we have obtained thus far, from a combination of the top three features, is 0.6058531836499368.

	r2	adjusted_r2
icu_patients_per_million, median_age	0.59587	0.59045
Top 2 + Bottom 2 features	0.60694	0.601668
All 7 features	0.615201	**0.610039**

2.7.1 Analysis

The `adjusted_r2_score` of all 7 features trumped that of the top 3 features by only 0.005. In other words, our multivariate linear regression model using just the top three features is **just as good** as combining all seven features. **Since the model with 3 features is far less computationally intensive, it should still be preferentially selected**.

2.8 Feature transformation

In a bid to further improve our multivariate linear regression model, we attempted to square root and square our top three independent variables. For reference, the highest `adjusted_r2_score` we have obtained thus far, from a combination of the top three features, is 0.6058531836499368. Models which performed better have their `adjusted_r2_score` highlighted in **bold**

	r2	adjusted_r2
With icu_patients_per_million_squared	0.618999	**0.613889**
With icu_patients_per_million_square_root	0.598616	0.593232
With people_vaccinated_per_hundred_squared	0.61373	**0.60855**
With people_vaccinated_per_hundred_square_root	0.615667	**0.610512**
With stringency_index_squared	0.614383	**0.609211**
With stringency_index_square_root	0.609623	0.604387

2.9 Combining transformed features

From the experimental results obtained, we developed a model by combining the polynomial features which obtained a higher `adjusted_r2_score`. Highest value is indicated in **bold**

	r2	adjusted_r2
With top 2 transformed features	0.626004	0.620987
With top 3 transformed features	0.632497	0.627568
With top 4 transformed features	0.639255	**0.634417**

2.10 Analysis

By introducing transformed features to our model, the `adjusted_r2_score` had a **marked improvement of more than 0.04**.

3 Conclusion

3.1 Final model

Our team attempted to predict daily deaths due to COVID-19 based on historical data. Independent of time and location, our final model is able to achieve a relatively high linear correlation of 0.6521598639422896. This model is ~0.06 higher than a simple linear correlation model (Highest value obtained was 0.592940113924265) and it consists of the following independent variables: 1. `icu_patients_per_million` 2. `people_vaccinated_per_hundred` 3. `stringency_index` 4. `icu_patients_per_million_squared` 5. `people_vaccinated_per_hundred_square_root` 6. `people_vaccinated_per_hundred_squared` 7. `stringency_index_squared`

3.2 Analysis

These findings show strong justifications that vaccinations may indeed help to curb death rates due to COVID-19. Additionally, it may also prove that implementing safe management measures in response to COVID-19 helps to curb death rates due to COVID-19. Most trivially, our results show that having a higher number of ICU patients leads to a higher death rate, which likely tells us that patients within the ICU are more likely to die as a result of COVID-19.

All that being said, we are aware that correlation does not equate to causation. Hence, further experiments need to be conducted to justify the hypotheses laid out above.

However, regardless of whether our hypotheses are true, we have developed a sufficiently accurate linear regression model to predict death rates due to COVID-19, at least until 10th November 2021.

3.3 Future work

To further test the strength of our model, our model should be tested against future datasets. If the linear regression remains high of > 0.5, it shows that our model is indeed independent of time and is not overfitting the data.

Additionally, our model has the potential to be further improved by introducing non-linear models, introducing other features that we were unable to test due to dataset and time limitations or by further transforming our current features.

Finally, our model can be improved by choosing a data set with a lower proportion of rows with missing values.

Section **??**

4 Import Modules

Section **??**

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import Markdown, display
from tabulate import tabulate
```

5 Functions

The following cell initialises all the functions that we will be using in this notebook.

Section **??**

[2]:
```
def printmd(string):
    '''Used for custom formatting of print statements'''
    display(Markdown(string))

def get_features_target(df, features, target):
    '''
    Retrieves corresponding columns from dataframe
    - df is the dataframe
    - features is an array
    - target is an array
    '''
    return df.loc[:, features], df.loc[:, target]

def prepare_features(df_features):
    '''
```

```
    1. Converts df_features into a 2d numpy array
    2. Adds a column of ones to df_features
    3. Returns df_features
    '''
    ones = np.ones((df_features.shape[0], 1))

    X = np.concatenate((ones, df_features.to_numpy()), axis=1)

    return X

def prepare_target(df_target):
    '''
    Converts df_target into a 2d numpy array and returns it
    '''

    target = df_target.to_numpy()

    return target

def normalise_z(df):
    '''Performs Z-score normalisation on specified dataframe'''
    mean = df.mean()
    std = df.std()
    return (df-mean)/std

def split_data(df_feature, df_target, random_state=None, test_size=0.5):
    '''Splits dataset into train and test set'''
    np.random.seed(random_state)

    n = int(df_feature.shape[0] * test_size)

    test_array = np.random.choice(df_feature.index.values, size=n,␣
↪replace=False)

    df_feature_train = df_feature.iloc[~df_feature.index.isin(test_array)]
    df_feature_test = df_feature.iloc[df_feature.index.isin(test_array)]
    df_target_train = df_target.iloc[~df_target.index.isin(test_array)]
    df_target_test = df_target.iloc[df_target.index.isin(test_array)]

    return df_feature_train, df_feature_test, df_target_train, df_target_test

# Get predicted values, y, based on current beta
def predict(df_features, beta):
    '''Obtains predicted target values via linear regression algorithm'''
    df_features_norm = normalise_z(df_features)

    array_2d = prepare_features(df_features_norm)
```

```
    return np.matmul(array_2d, beta)

# Calculate cost
def compute_cost(X, y, beta):
    '''Computes cost based on current beta values'''
    m = X.shape[0]
    predicted = np.matmul(X, beta)

    J = 1/(2*m) * np.sum((predicted-y)**2)
    return J

# Minimise cost
def gradient_descent(X, y, beta, alpha, num_iters):
    '''Minimises cost via the gradient descent algorithm
    - X = Features
    - y = Target values
    - beta = Parameters
    - alpha = Step size
    - num_iters = Number of iterations'''

    m = X.shape[0]
    J_storage = []

    for _ in range(num_iters):
        predicted = np.matmul(X, beta)
        derivative_error = (1/m) * np.matmul(X.T, predicted-y)

        beta = beta - alpha * derivative_error
        J_storage.append(compute_cost(X, y, beta))


    return beta, J_storage

# Obtain r2_score
def r2_score(y, ypred):
    '''Calculate r2_score using sum of squared residuals (ss_res) and total sum␣
 ↪of squares (ss_tot)'''
    y_ave = np.mean(y)
    ss_res = np.sum((y-ypred)**2)

    ss_tot = np.sum((y-y_ave)**2)

    return 1 - (ss_res/ss_tot)

# Obtain adjusted r2
def adjusted_r2_score(r2, N, p):
```

```
    '''
    - r2 is the value returned from r2_score()
    - N is the total sample size
    - p is the number of independent variables
    '''

    return 1 - ((1-r2)*(N-1)) / (N-p-1)

def get_r2_values(df, features, y, pred):
    '''
    Combines r2_score() and adjusted_r2_score() into one function to
    - Print r2_score and adjusted_r2_score
    - Return r2_score and adjusted_r2_score
    '''
    r2 = r2_score(y, pred)
    adjusted_r2 = adjusted_r2_score(r2, y.shape[0], df.shape[1])
    title = ", ".join(features)

    printmd(f'''Features: **{title}**
1. r2_score is {r2}
2. adjusted_r2_score is {adjusted_r2}''')
    return r2, adjusted_r2

# Obtain MSE
def mean_squared_error(target, pred):
    n = target.shape[0]

    return 1/n * np.sum((target-pred)**2)

def format_df(df):
    '''
    Returns a new dataframe that calculates the mean across a 1-month period␣
 ↪for each country
    '''

    ls = []
    loc_date = []

    # For each location
    for loc in df["location"].unique():
        # For each year
        for y in range(2020, 2022):
            # For each month
            for i in range(1, 13):
                date_filter = (pd.to_datetime(df['date']).dt.month == i) & (pd.
 ↪to_datetime(df['date']).dt.year == y)
                location_filter = df["location"] == loc
```

```
                loc_date.append([loc, f'{i}-2021'])

                mean = df[date_filter].loc[location_filter].iloc[:, 2:].mean()
                ls.append(mean)

    # Convert to df
    month_mean_df = pd.DataFrame(ls, columns=df.columns)
    month_mean_df[["location", "date"]] = loc_date

    # Drop na
    df = month_mean_df.dropna()

    return df

def plot_data(df, y, pred, features, target_col):
    '''Plots scatterplot for each feature in features'''

    for i, f in enumerate(features):
        plot = plt.figure(i)
        plt.scatter(df[f], y)
        plt.scatter(df[f], pred)
        plt.xlabel(f)
        plt.ylabel(target_col[0])
        plt.show()

def lin_reg(df, features, target_col=["new_deaths_smoothed_per_million"],␣
↪random_state=100, show_data_plot=True):
    '''
    Trains dataset according to gradient descent algorithm
    This function is dependent on
    - get_features_target()
    - split_data()
    - normalise_z()
    - prepare_features()
    - prepare_target()
    - get_r2_values()
    - gradient_descent()
    - predict()
    - plot_data()
    '''
    df_features, df_target = get_features_target(df, features, target_col)

    # Split data
    df_features_train, df_features_test, df_target_train, df_target_test =␣
↪split_data(df_features, df_target, random_state=random_state, test_size=0.3)
```

```
    # Normalise features
    df_features_train = normalise_z(df_features_train)

    # Prepare features, target
    X = prepare_features(df_features_train)
    target = prepare_target(df_target_train)

    # Initialise variables for gradient descent
    # Beta: num_features * 1 column
    beta = np.zeros((X.shape[1], 1))
    alpha = 0.01
    iterations = 1000

    # Minimise cost function via gradient descent algorithm
    beta, J_storage = gradient_descent(X, target, beta, alpha, iterations)

    # Plot total cost against number of iterations
    plt.plot(J_storage)

    # Get predicted values
    pred = predict(df_features_test, beta)

    # Prepare target
    target = prepare_target(df_target_test)

    # Calculate and return r2, adjusted_r2
    r2, adjusted_r2 = get_r2_values(df, features, target, pred)

    if show_data_plot:
        # For each feature, plot a scatterplot
        plot_data(df_features_test, target, pred, features, target_col)

    return beta, r2, adjusted_r2
```

6 Read Dataset

Our dataset is taken from Our World In Data which is updated on a daily basis.

Our dataset was **last updated on 10th November 2021**.

Section **??**

[3]:
```
# Read the CSV file
df = pd.read_csv("owid-covid-data.csv")
```

7 Variable Selection & Definitions

To predict `new_deaths_smoothed_per_million`, we experimented with the following independent variables. These variables were initially chosen through intellectual conjecture on what variables are most likely to affect `new_deaths_smoothed_per_million`. Subsequently, we elaborated on the steps taken to ensure we picked variables that were the most linearly related to `new_deaths_smoothed_per_million`.

7.1 Independent Variables

1. `icu_patients_per_million` = Number of COVID-19 patients in intensive care units (ICUs) on a given day per 1,000,000 people
2. `new_cases_smoothed_per_million` = New confirmed cases of COVID-19 (7-day smoothed) per 1,000,000 people
3. `stringency_index` = Government Response Stringency Index: composite measure based on 9 response indicators including school closures, workplace closures, and travel bans, rescaled to a value from 0 to 100 (100 = strictest response)
4. `median_age` = Median age of the population, UN projection for 2020
5. `extreme_poverty` = Share of the population living in extreme poverty, most recent year available since 2010
6. `people_fully_vaccinated_per_hundred` = Total number of people who received all doses prescribed by the vaccination protocol per 100 people in the total population
7. `people_vaccinated_per_hundred` = Total number of people who received at least one vaccine dose per 100 people in the total population
8. `reproduction_rate` = Real-time estimate of the effective reproduction rate (R) of COVID-19. See https://github.com/crondonm/TrackingR/tree/main/Estimates-Database
9. `human_development_index` = A composite index measuring average achievement in three basic dimensions of human development—a long and healthy life, knowledge and a decent standard of living. Values for 2019, imported from http://hdr.undp.org/en/indicators/137506

7.2 Dependent Variable

1. `new_deaths_smoothed_per_million` = New deaths attributed to COVID-19 (7-day smoothed) per 1,000,000 people

Section **??**

8 Data pre-processing

8.1 Attempt 1 - Forward fill

We attempted to replace missing values using both the forward fill and backward fill techniques. However, this resulted in a much poorer linear correlation between our independent and dependent variables. Section **??**.

8.2 Attempt 2 - Dropping missing values

As $> 95\%$ of the data points contain rows that have missing values, we decided to drop the rows with missing values altogether as filling them up through either forward or backward propagation

would inevitably skew the data points towards a more unnatural result. This also produced a more favourable linear correlation between our independent and dependent variables which will be shown in our next section.

We processed the data by 1. Extracting out columns containing our independent and dependent variables 2. Dropping rows with missing values

Section **??**

[4]:
```
useful_cols = ["location", "date", "icu_patients_per_million",␣
 ↪"new_cases_smoothed_per_million", "stringency_index", "median_age",␣
 ↪"extreme_poverty", "people_fully_vaccinated_per_hundred",␣
 ↪"people_vaccinated_per_hundred", "reproduction_rate",␣
 ↪"human_development_index", "new_deaths_smoothed_per_million"]

df_useful = df[useful_cols]

# Drop rows with nan
df_dropna_one = df_useful.dropna()

print(f"Original sample size is {df_useful.shape[0]}. After dropping missing␣
 ↪values, the number of data points is {df_useful.dropna().shape[0]}")
```

```
Original sample size is 132016. After dropping missing values, the number of
data points is 4539
```

9 Univariate Linear Regression

For reliability purposes, a random_state of 100 has been set. `r2` and `adjusted_r2` values have been sorted in ascending order.

	r2	adjusted_r2
icu_patients_per_million	0.595933	0.590514
new_cases_smoothed_per_million	0.277922	0.268237
stringency_index	0.116847	0.105001
median_age	0.00456267	-0.00878895
extreme_poverty	0.0200249	0.00688066
people_fully_vaccinated_per_hundred	0.179166	0.168156
people_vaccinated_per_hundred	0.225711	0.215326
reproduction_rate	0.044482	0.0316658
human_development_index	0.06814	0.0556411

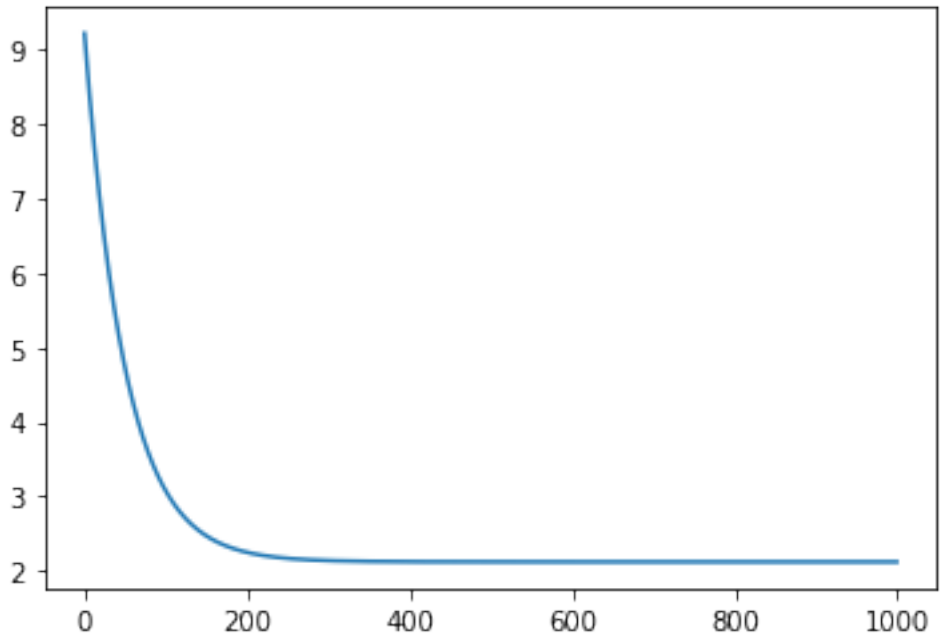
9.1 Analysis

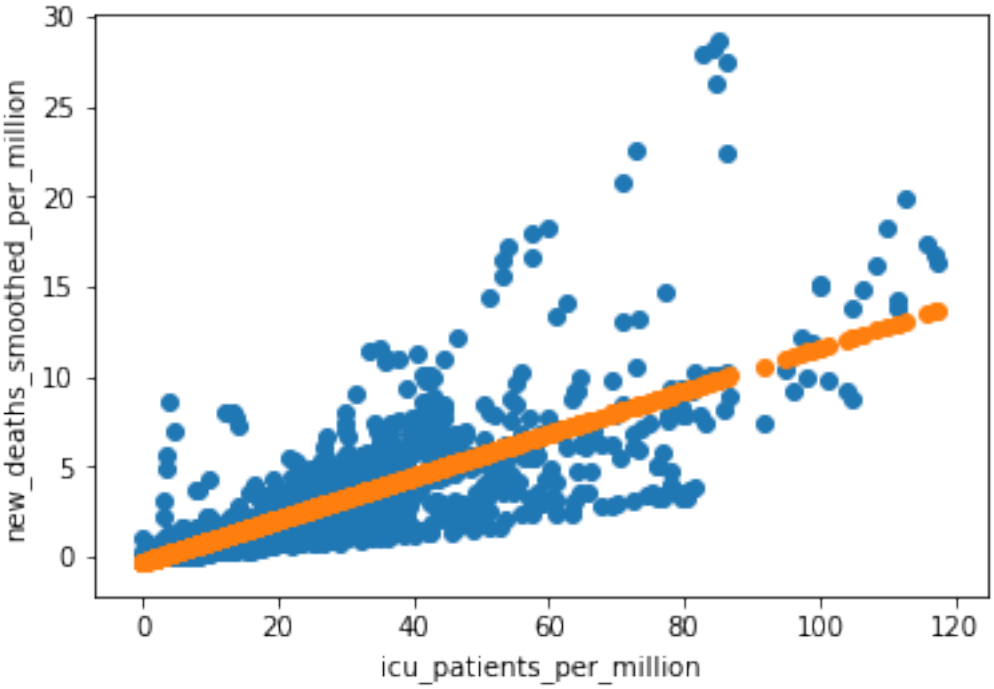
All `adjusted_r2_score`, excluding that of `new_cases_smoothed_per_million` experienced a drastic drop in values. We believe this is likely due to the data set containing far too many rows with missing values. For future improvements, we will be choosing a data set with lesser empty rows.

Section **??**

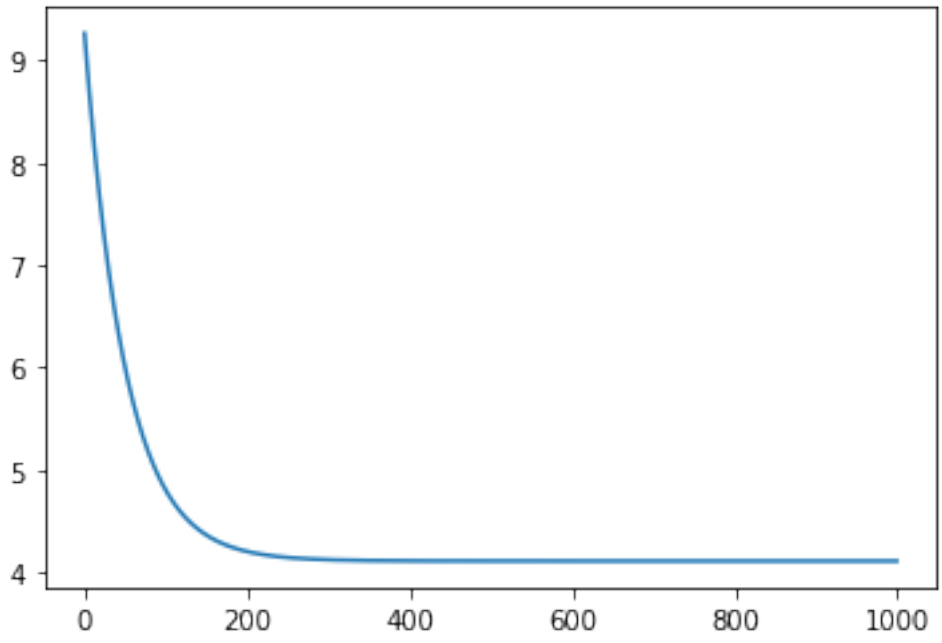
[5]:
```
independent_var = ["icu_patients_per_million",␣
 ↪"new_cases_smoothed_per_million", "stringency_index", "median_age",␣
 ↪"extreme_poverty", "people_fully_vaccinated_per_hundred",␣
 ↪"people_vaccinated_per_hundred", "reproduction_rate",␣
 ↪"human_development_index"]
for i, feature in enumerate(independent_var):
    _, r2, adjusted_r2 = lin_reg(df_dropna_one, [feature], random_state=100)
```

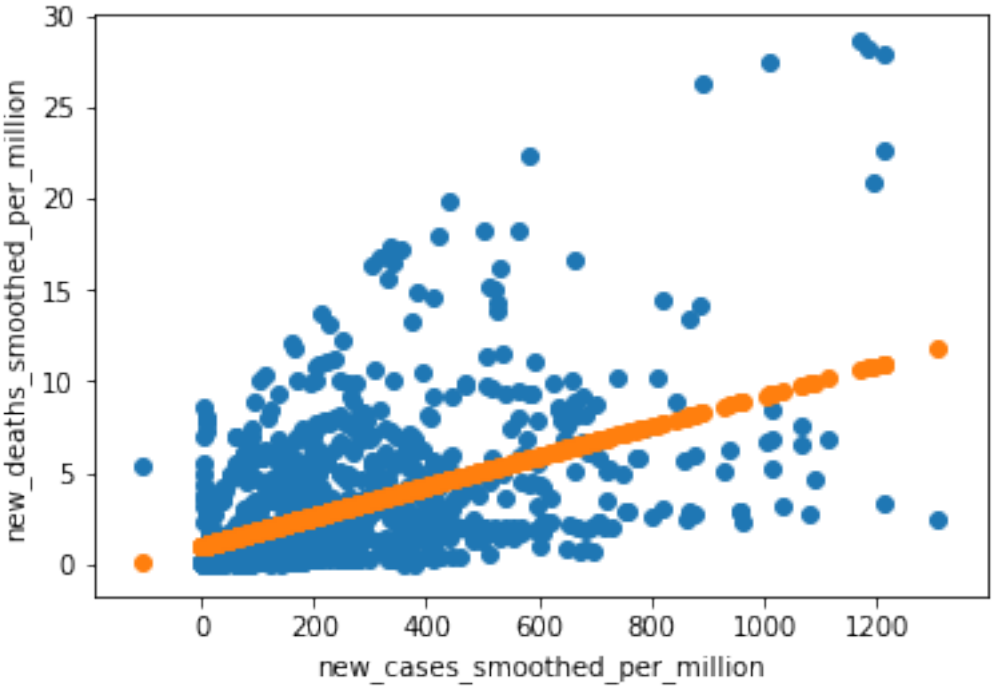
Features: **icu_patients_per_million** 1. r2_score is 0.5959332013218808 2. adjusted_r2_score is 0.592336167505755



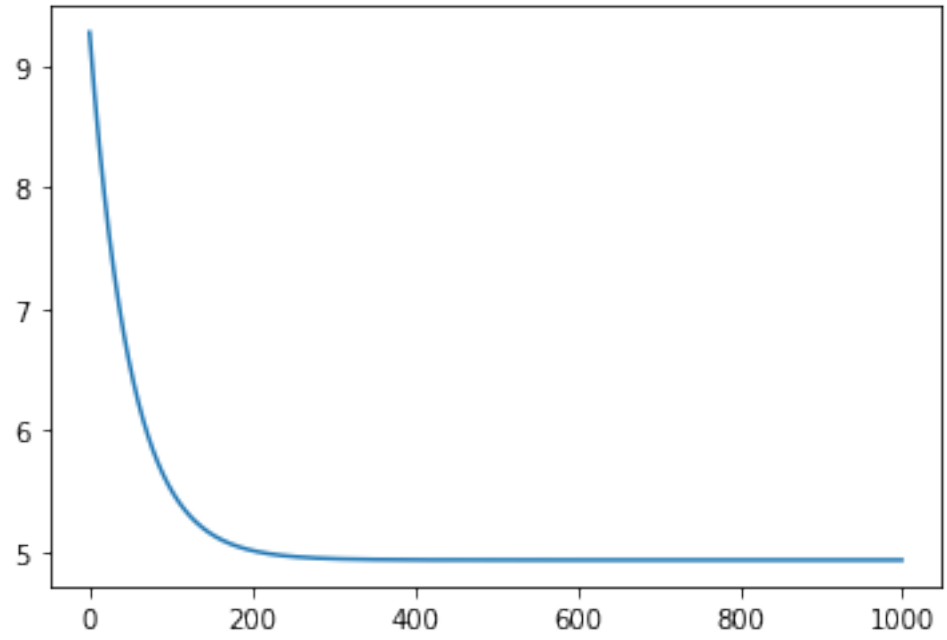


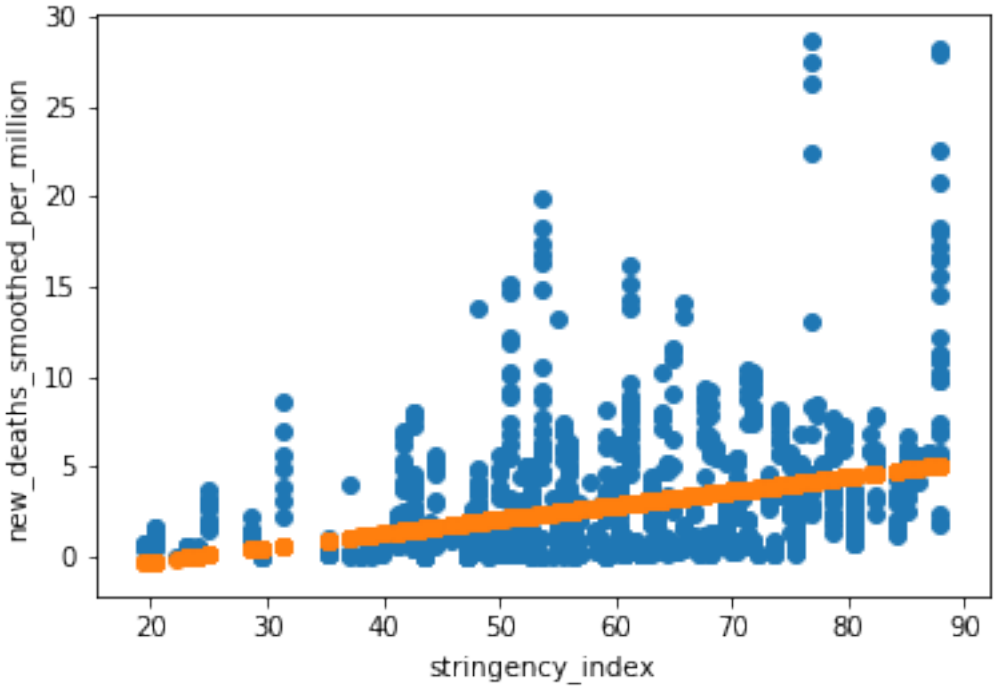
Features: **new_cases_smoothed_per_million** 1. r2_score is 0.277922392580796 2. adjusted_r2_score is 0.2714944020102986



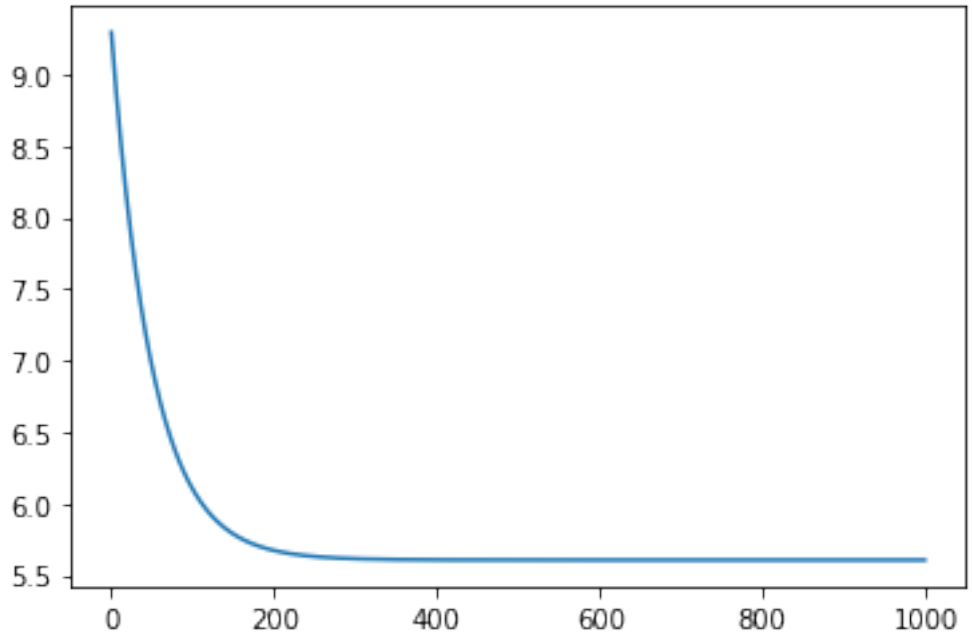


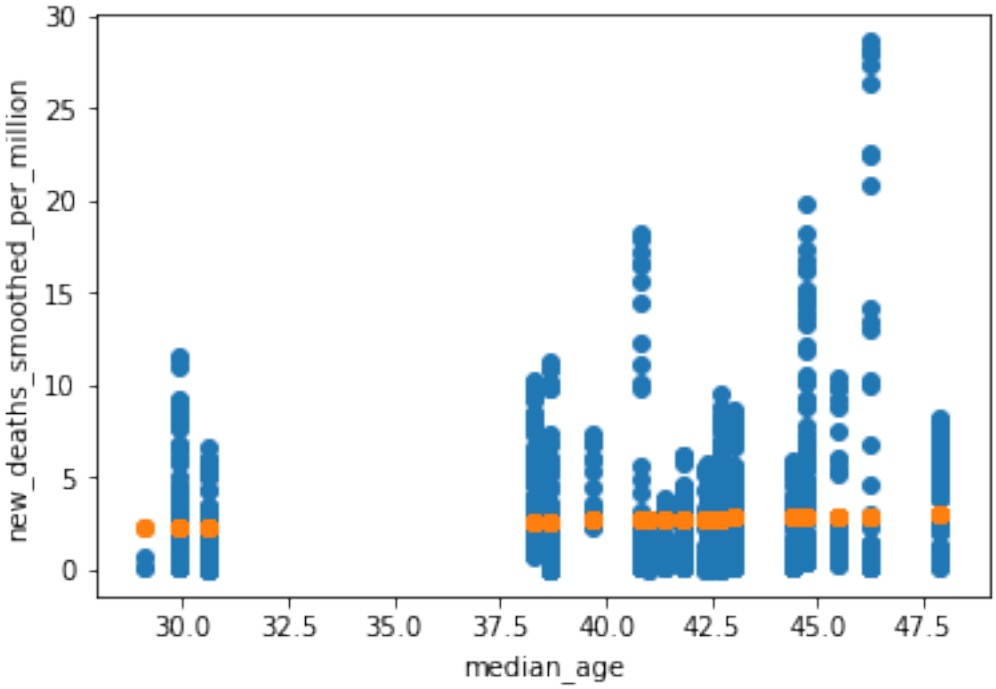
Features: **stringency_index** 1. r2_score is 0.11684654483557266 2. adjusted_r2_score is 0.1089846446412307



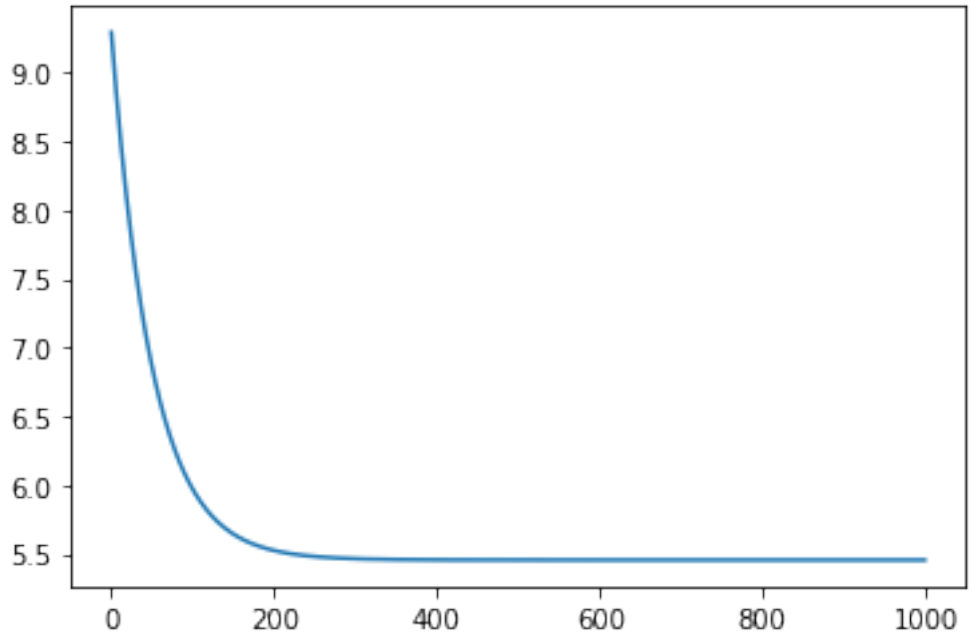


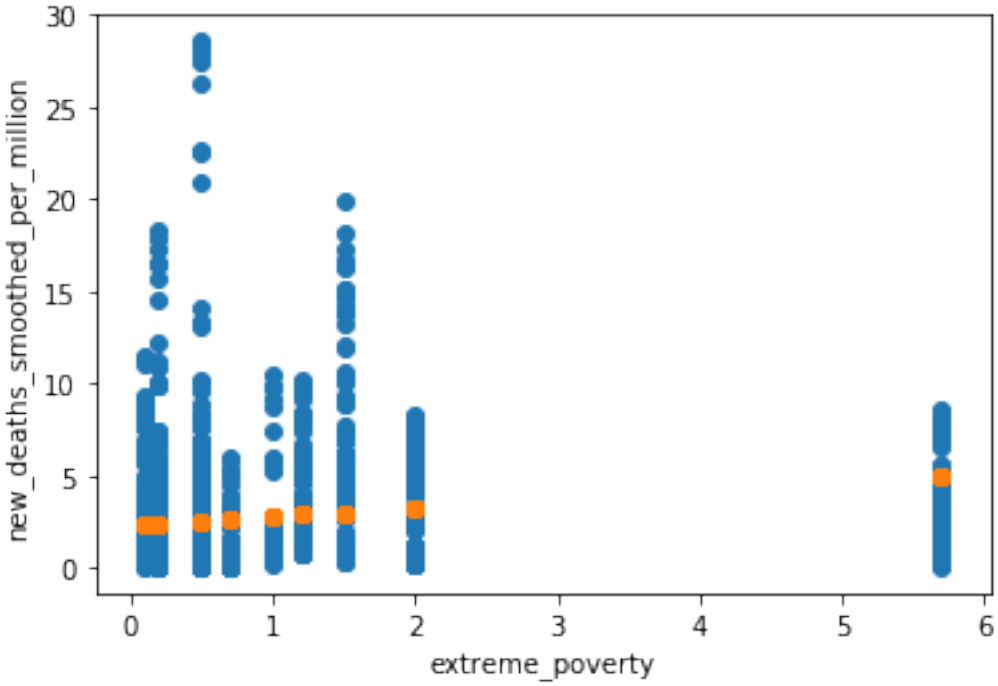
Features: **median_age** 1. r2_score is 0.004562667711661805 2. adjusted_r2_score is -0.00429879221968843



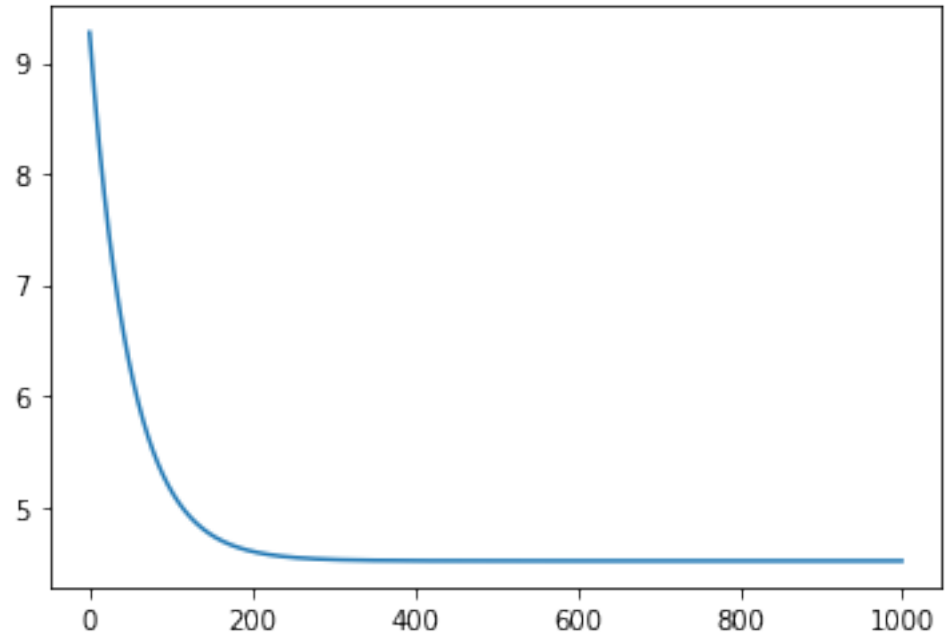


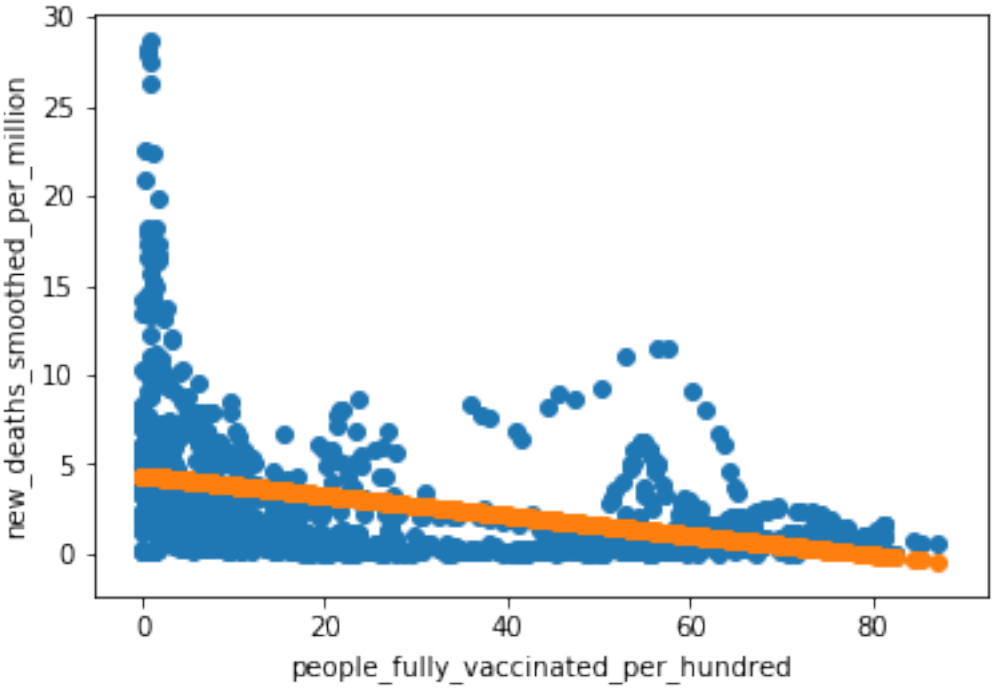
Features: **extreme_poverty** 1. r2_score is 0.020024889830333126 2. adjusted_r2_score is 0.011301075793214377



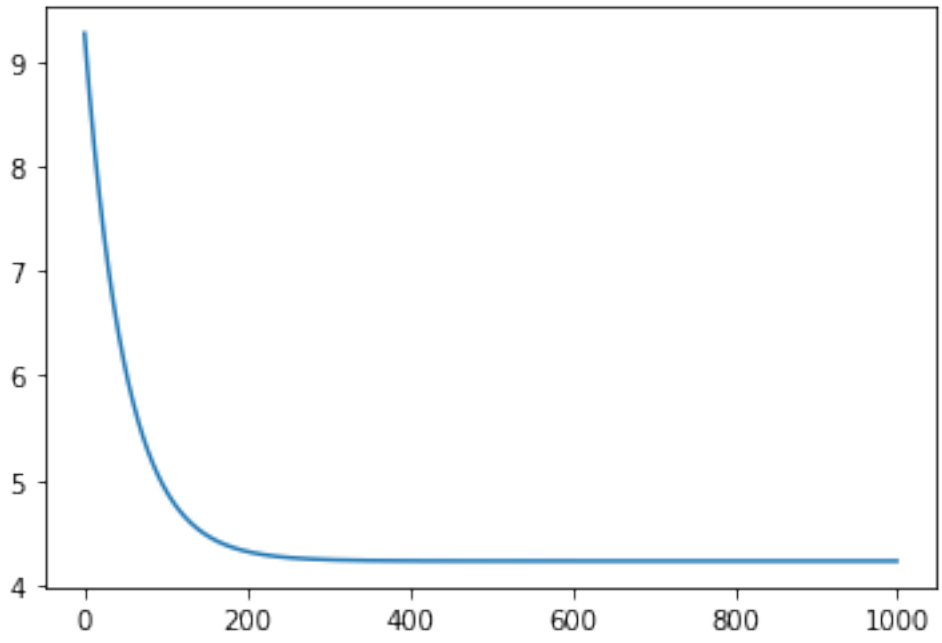


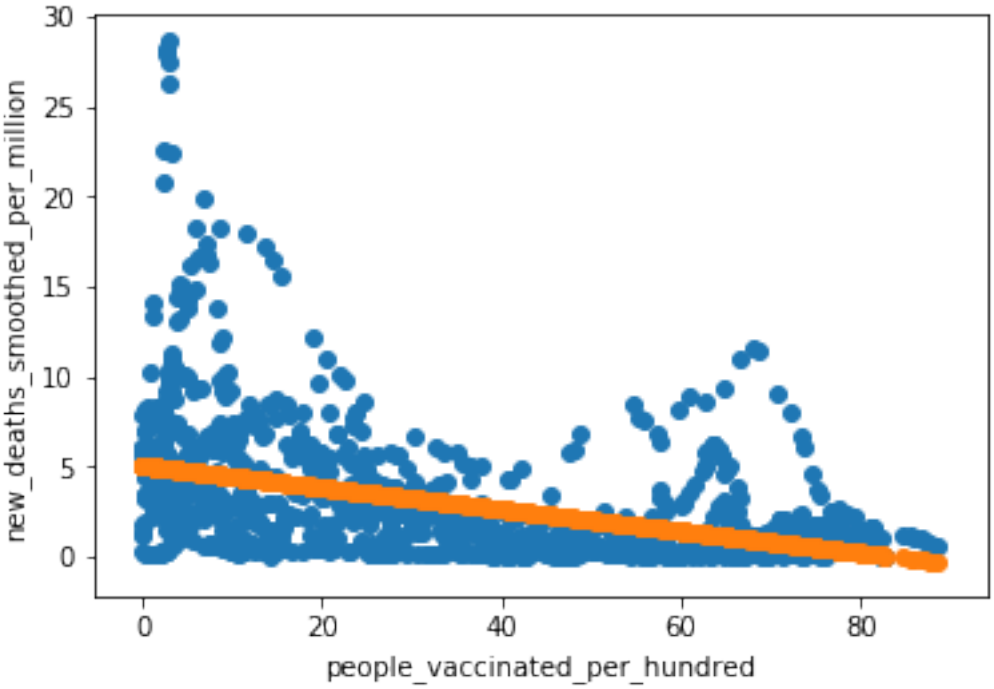
Features: **people_fully_vaccinated_per_hundred** 1. r2_score is 0.17916576746663748 2. adjusted_r2_score is 0.1718586378001684



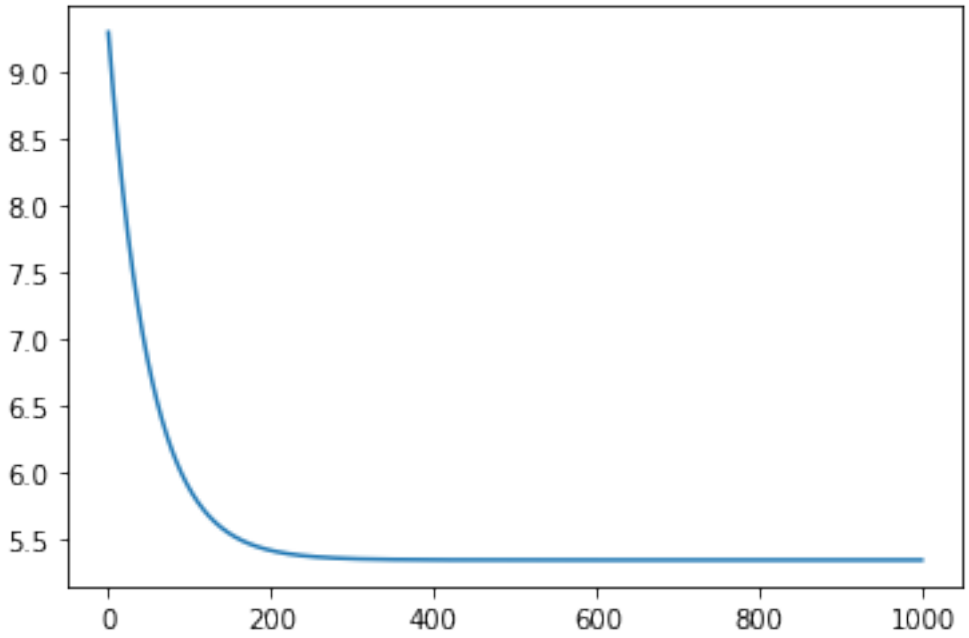


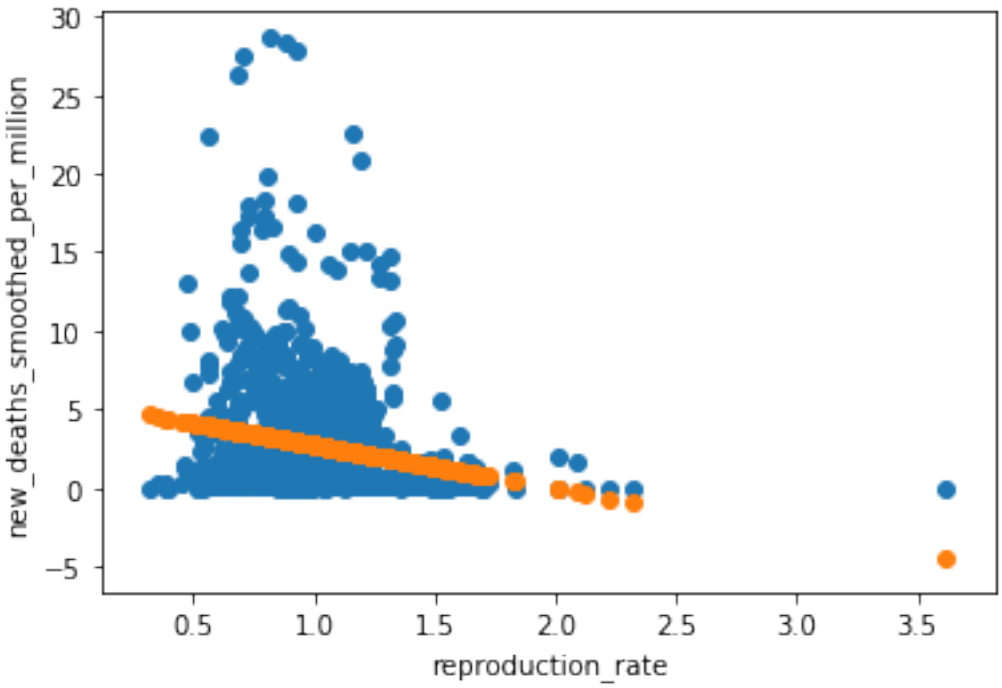
Features: **people_vaccinated_per_hundred** 1. r2_score is 0.2257109391683364 2. adjusted_r2_score is 0.21881815821137796



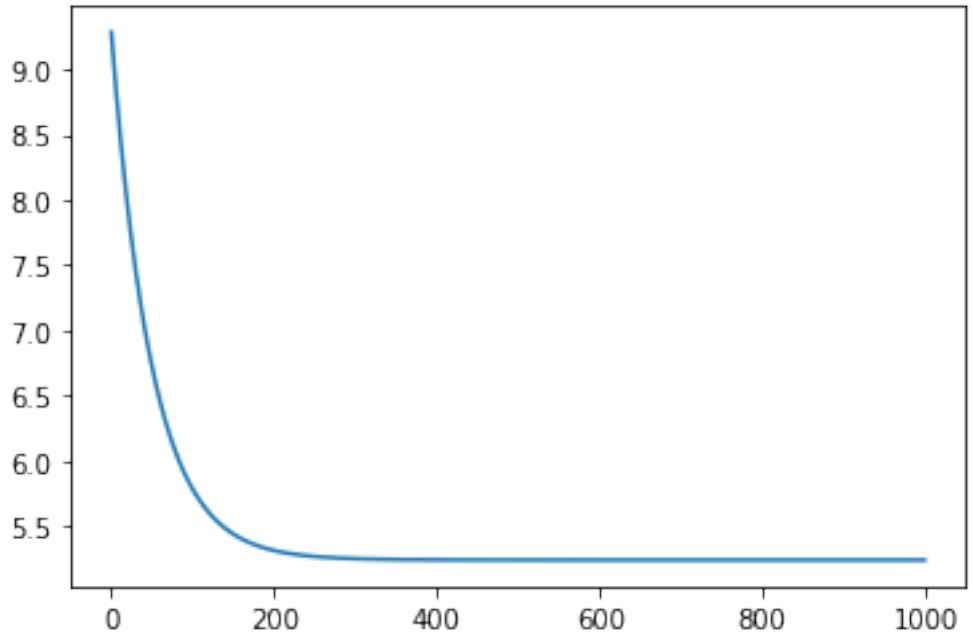


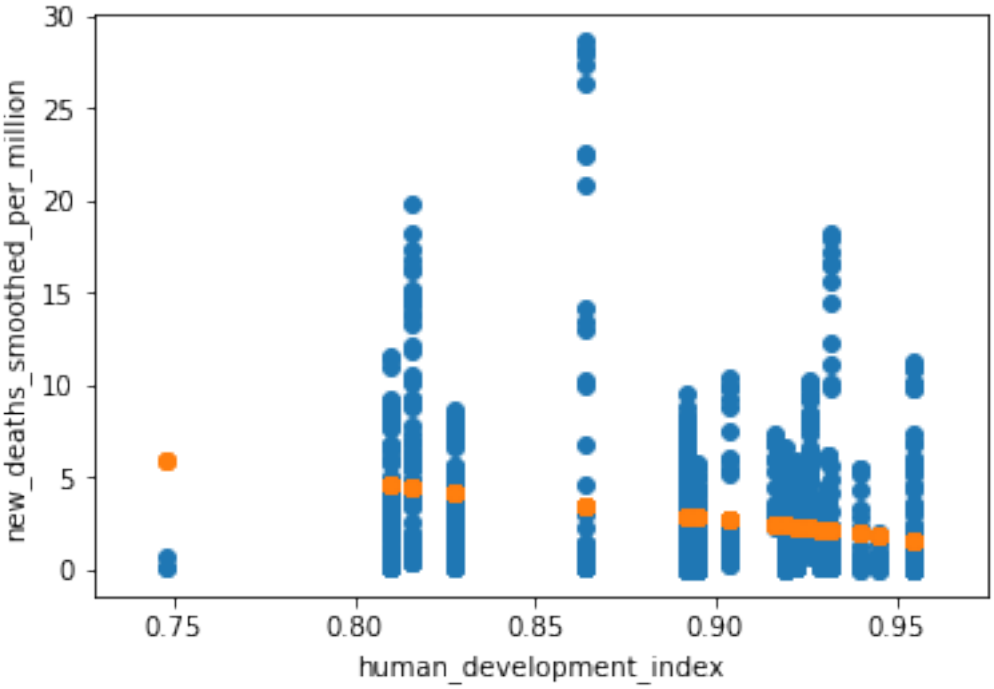
Features: **reproduction_rate** 1. r2_score is 0.04448199021801924 2. adjusted_r2_score is 0.035975895175449746





Features: **human_development_index** 1. r2_score is 0.06813995336217726 2. adjusted_r2_score is 0.0598444633327605





10 Multivariate variable selection

Next, we fine-tuned the variables selected based on the following reasons:

10.1 1. High linear correlation

We selected the top 3 independent variables based on their `adjusted_r2_score`.

10.2 2. Eliminate multicollinearity

Source

Multicollinearity occurs when the independent variables selected are highly correlated with one another. This means that the change in one independent variables results in the change of another, leading to the model fluctuating significantly. Consequently, 1. Finding optimal beta values may be unstable 2. Overfitting may occur 3. Selection of independent variables may be unpredictable

To prevent multicollinearity, we will be dropping independent variables with high linear correlation with each other. Numerical calculations showcasing the high linear correlation between our independent variables may be found Section **??**. Also, for the sake of optimising our multivariate linear regression model, we have chosen to drop the variable with a lower linear correlation with our dependent variable. Hence, the following variables will be dropped: 1. `new_cases_smoothed_per_million` - Strong linear correlation with `icu_patients_per_million` 2. `people_fully_vaccinated_per_hundred` - Strong linear correlation with `people_vaccinated_per_hundred`

10.3 3. Removing variables with negligible linear correlation

Similar to 1, we removed variables which Section **??** a linear correlation of < 0.01 with our independent variable.

Section **??**

11 Multivariate Linear Regression

To prevent overfitting, we will only be concerned with the `adjusted_r2_score`. For reference, the highest `adjusted_r2_score` through univariate linear regression is 0.592940113924265. Highest value is in **bold**.

	r2	adjusted_r2
icu_patients_per_million, people_vaccinated_per_hundred	0.60807	0.602813
icu_patients_per_million, people_vaccinated_per_hundred, stringency_index	0.608751	**0.603504**

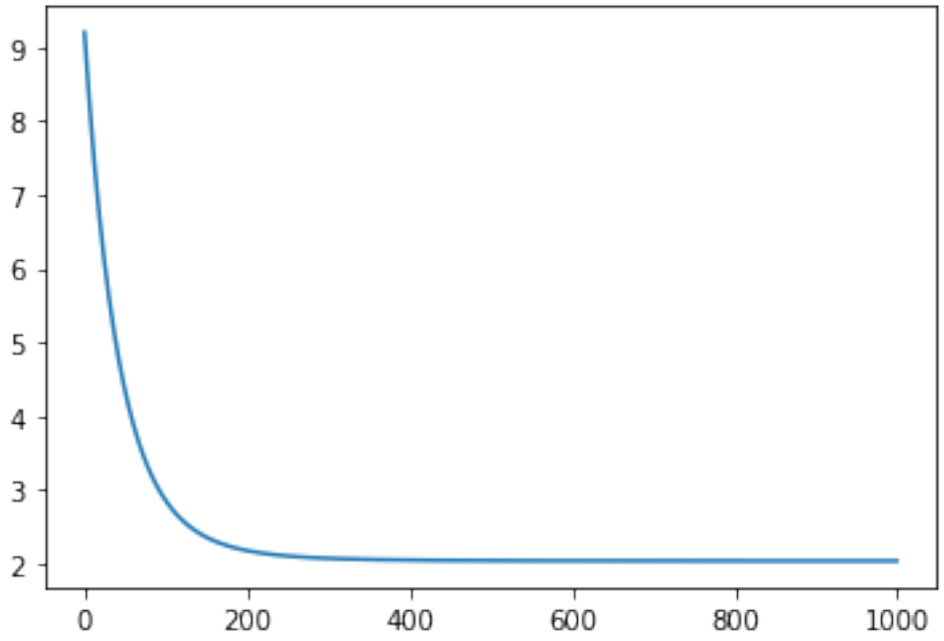
11.1 Analysis

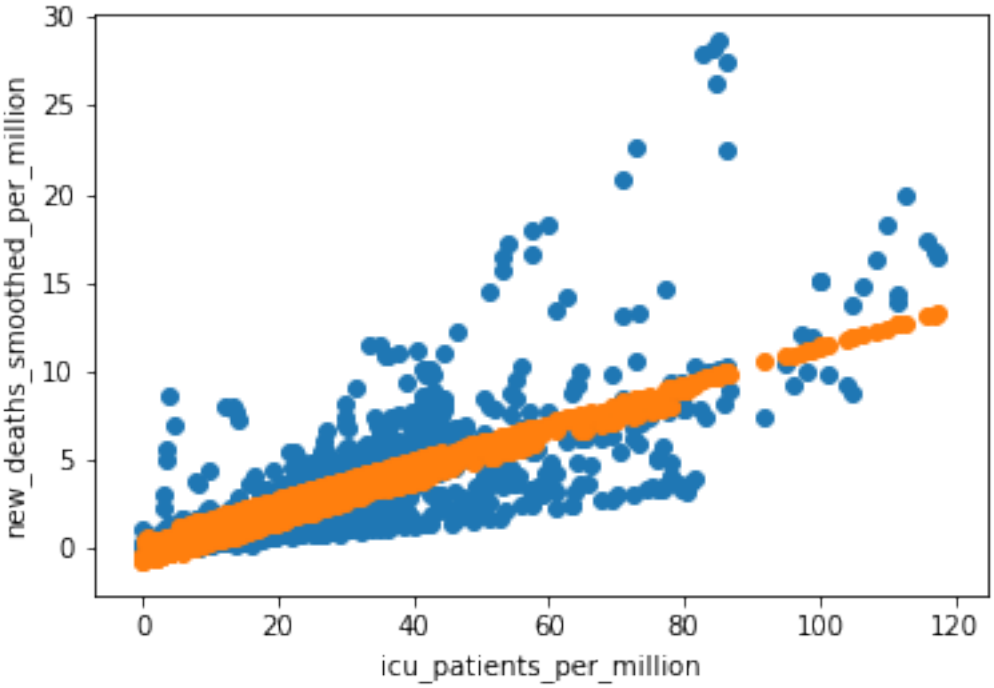
While multivariate linear regression did showed an improvement in the `adjusted_r2_score`, **the improvement is arguably slight**.

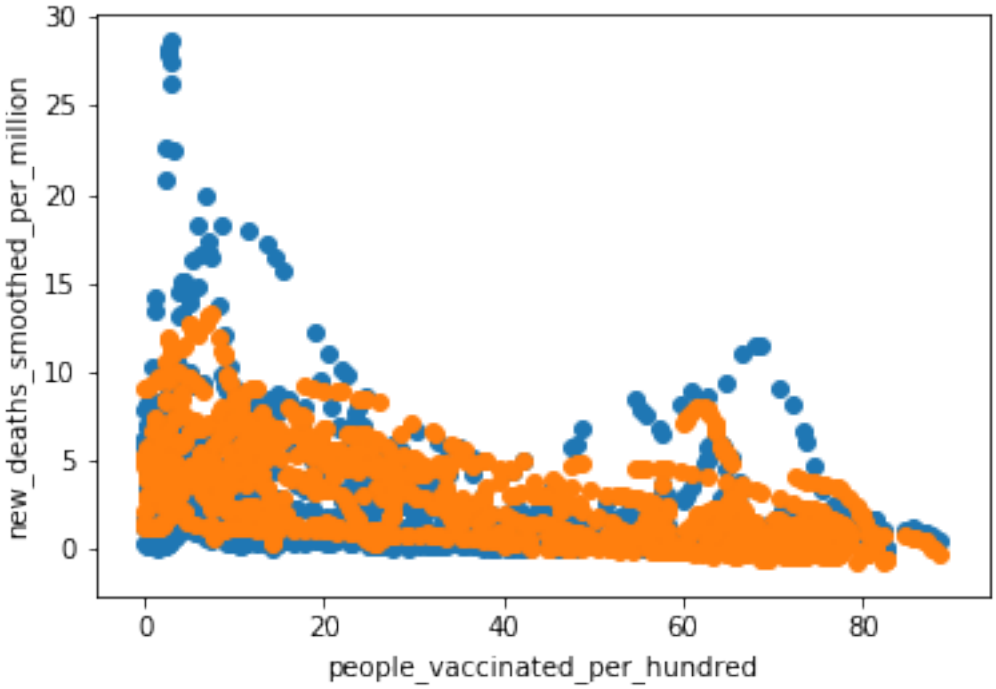
Section **??**

[6]:
```
table = [["", "r2", "adjusted_r2"]]

top_two_features = ["icu_patients_per_million", "people_vaccinated_per_hundred"]
_, r2, adjusted_r2 = lin_reg(df_dropna_one, top_two_features, random_state=100)

table.append([", ".join(top_two_features), r2, adjusted_r2])

top_three_features = ["icu_patients_per_million",␣
↪"people_vaccinated_per_hundred", "stringency_index"]
_, r2, adjusted_r2 = lin_reg(df_dropna_one, top_three_features,␣
↪random_state=100)

table.append( [", ".join(top_three_features), r2, adjusted_r2])
```

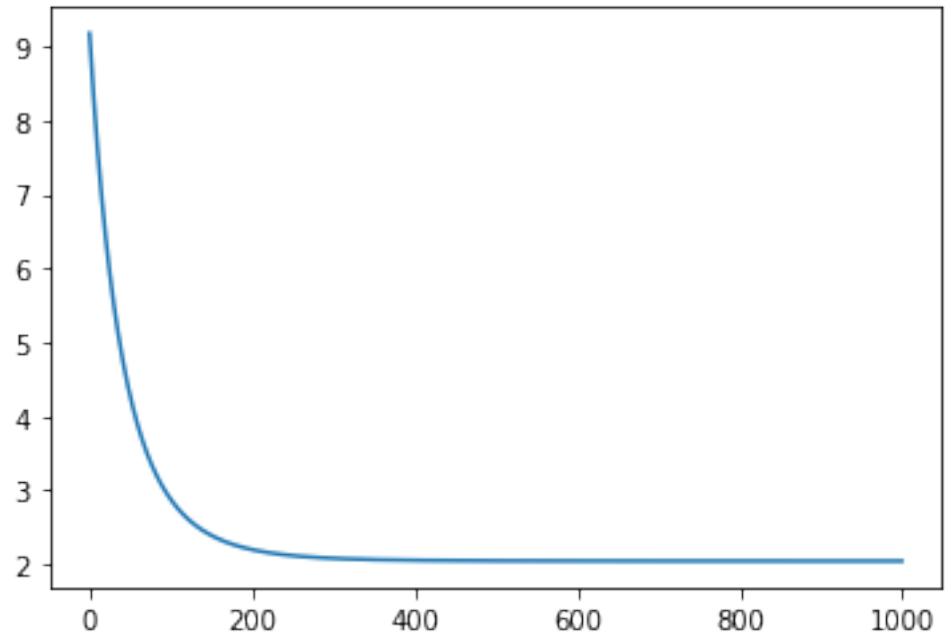
Features: **icu_patients_per_million, people_vaccinated_per_hundred** 1. r2_score is 0.6080696126819294 2. adjusted_r2_score is 0.6045806181360712

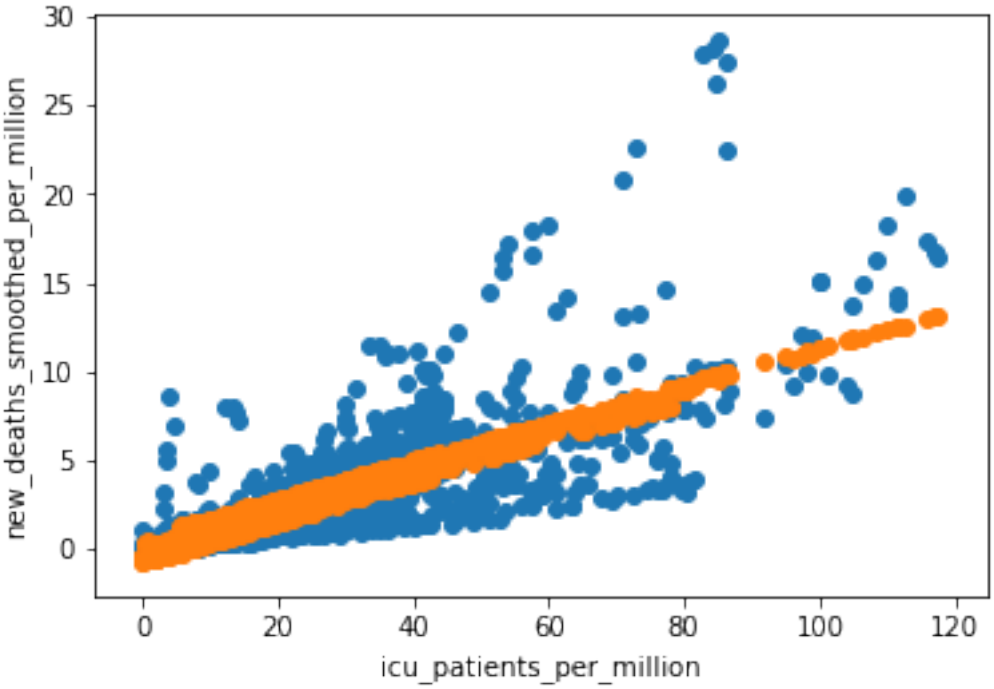

9
8
7
6
5
4
3
2
0
200
400
600
800
1000

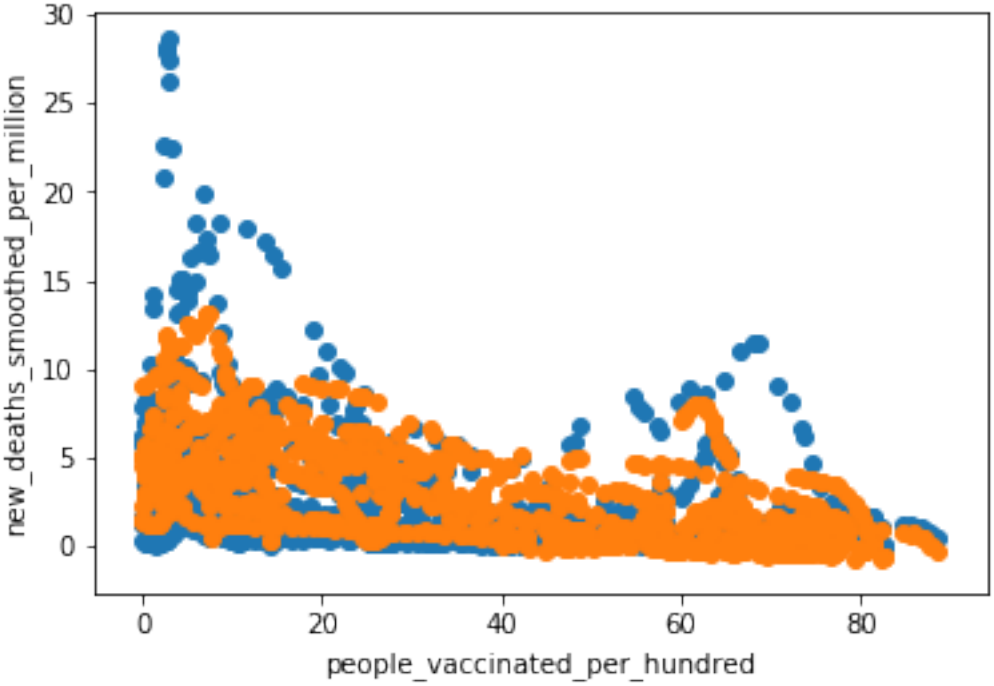

30
25
20
15
10
5
0
new_deaths_smoothed_per_million
0
20
40
60
80
100
120
icu_patients_per_million

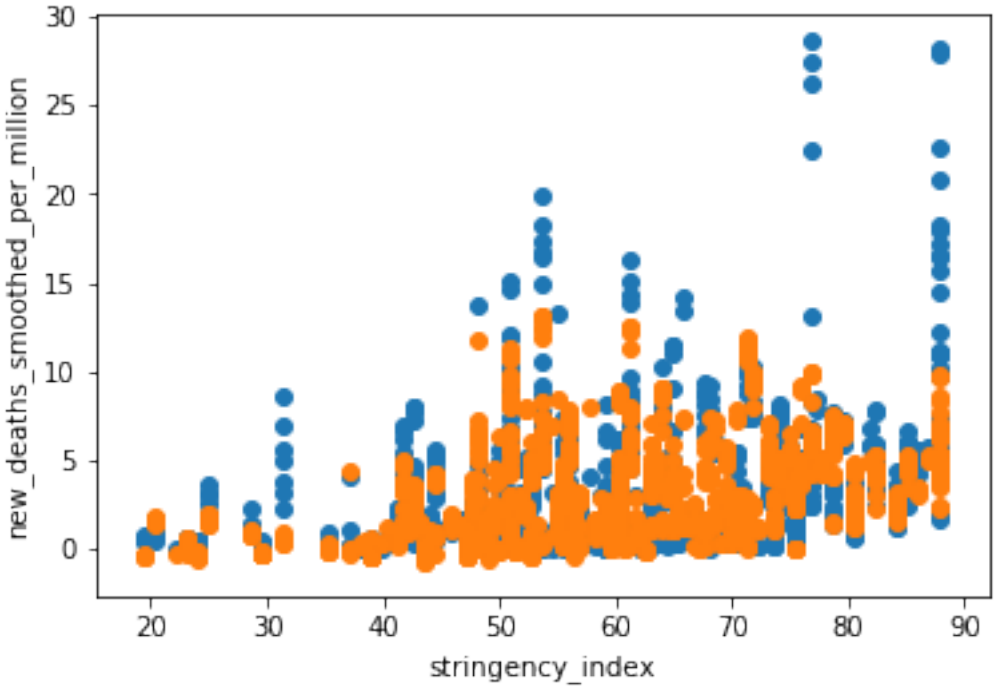


Features: **icu_patients_per_million, people_vaccinated_per_hundred, stringency_index** 1. r2_score is 0.6087513220054519 2. adjusted_r2_score is 0.6052683960885865




new_deaths_smoothed_per_million
icu_patients_per_million
0
5
10
15
20
25
30
0
20
40
60
80
100
120


new_deaths_smoothed_per_million
people_vaccinated_per_hundred
0
5
10
15
20
25
30
0
20
40
60
80



12 What happens if we only have variables with low linear correlation?

For reference, the highest `adjusted_r2_score` amongst the 4 variables listed below, for univariate linear regression, is 0.00835851947117039. Highest value obtained through multivariate linear regression is given in **bold**.

	r2	adjusted_r2
median_age, extreme_poverty	0.0205068	0.00736906
median_age, extreme_poverty, reproduction_rate	0.0634354	0.0508734
median_age, extreme_poverty, reproduction_rate, human_development_index	0.113784	**0.101898**

12.1 Analysis

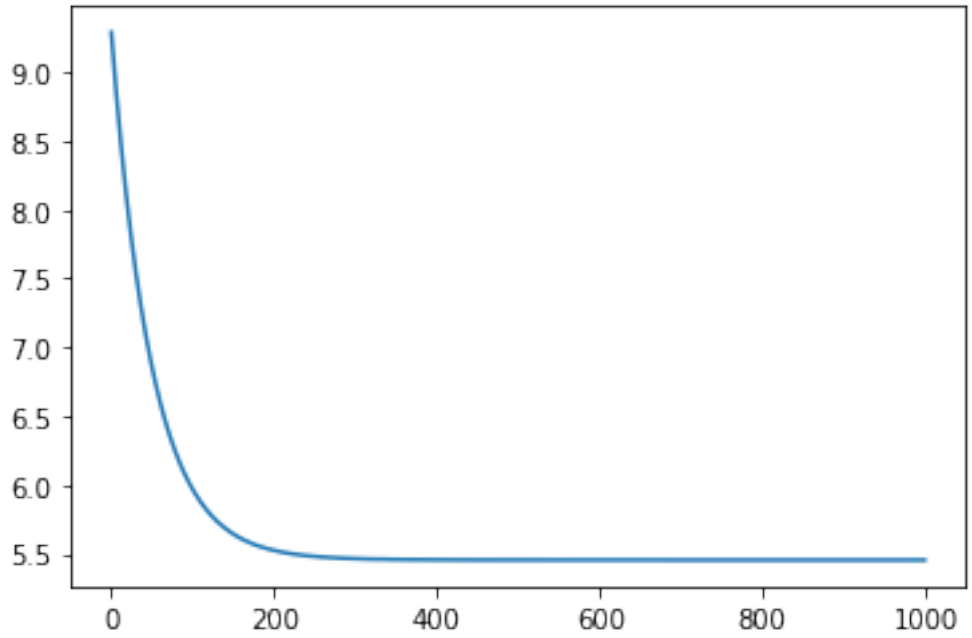
As expected, while we do see an increase in overall `adjusted_r2_score`, the new `adjusted_r2_score` is still low.

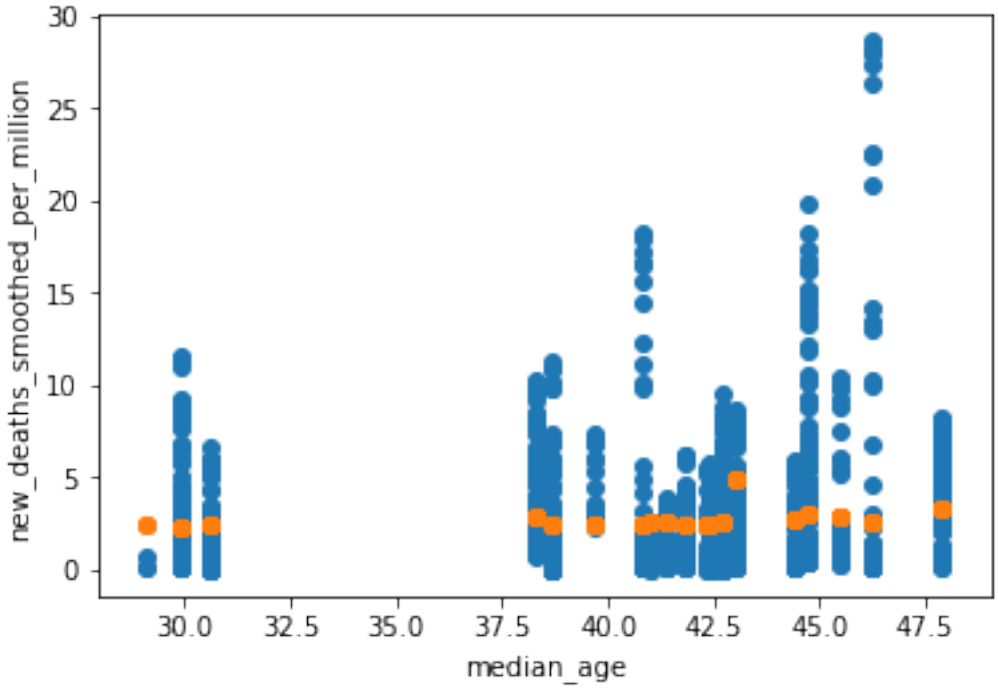
Section **??**

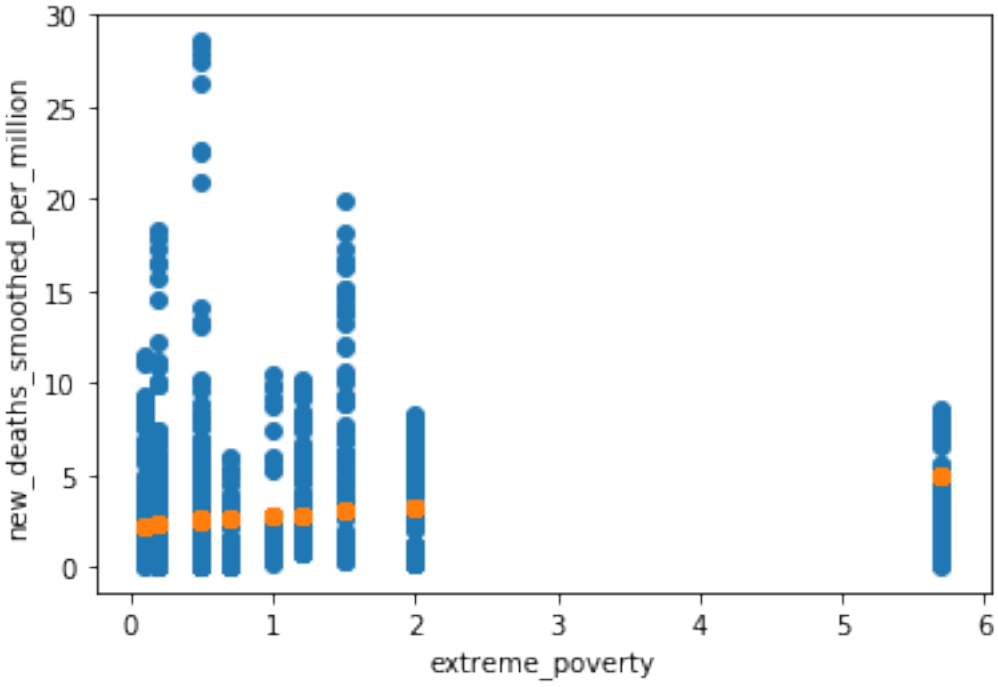
[7]:
```
table = [["", "r2", "adjusted_r2"]]

bottom_two_features = ["median_age", "extreme_poverty"]
```

```
_, r2, adjusted_r2 = lin_reg(df_dropna_one, bottom_two_features,␣
 ↪random_state=100)

table.append([", ".join(bottom_two_features), r2, adjusted_r2])

bottom_three_features = ["median_age", "extreme_poverty", "reproduction_rate"]
_, r2, adjusted_r2 = lin_reg(df_dropna_one, bottom_three_features,␣
 ↪random_state=100)

table.append([", ".join(bottom_three_features), r2, adjusted_r2])

bottom_four_features = ["median_age", "extreme_poverty", "reproduction_rate",␣
 ↪"human_development_index"]
_, r2, adjusted_r2 = lin_reg(df_dropna_one, bottom_four_features,␣
 ↪random_state=100)

table.append([", ".join(bottom_four_features), r2, adjusted_r2])
```

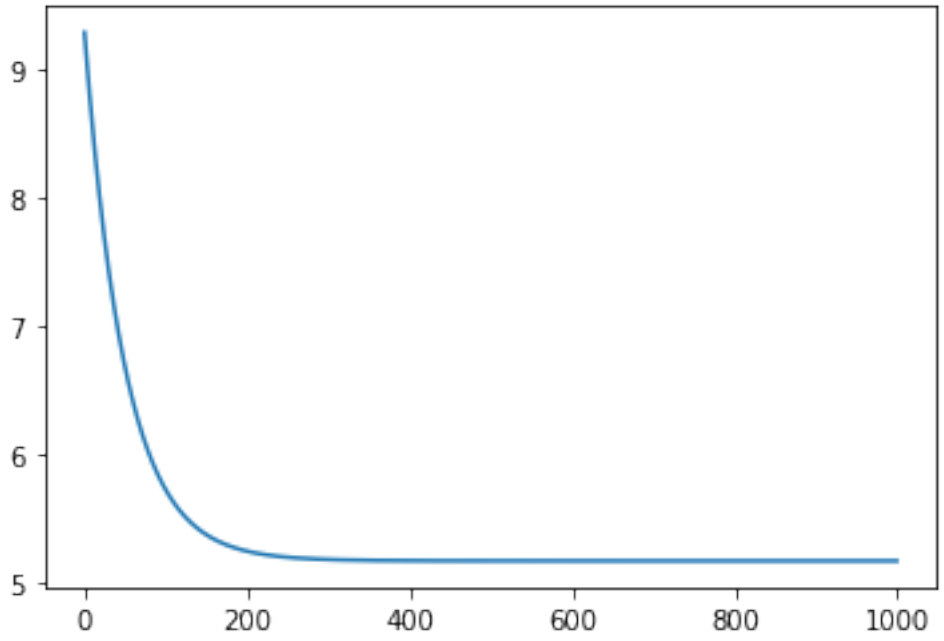
Features: **median_age, extreme_poverty** 1. r2_score is 0.020506817566091406 2. adjusted_r2_score is 0.011787293686857825

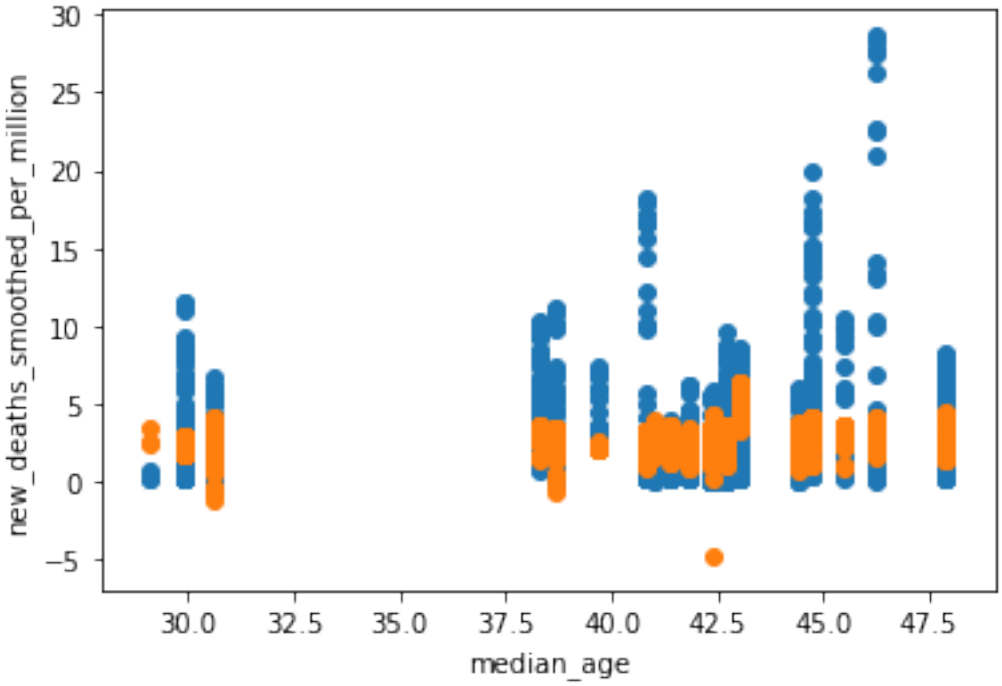


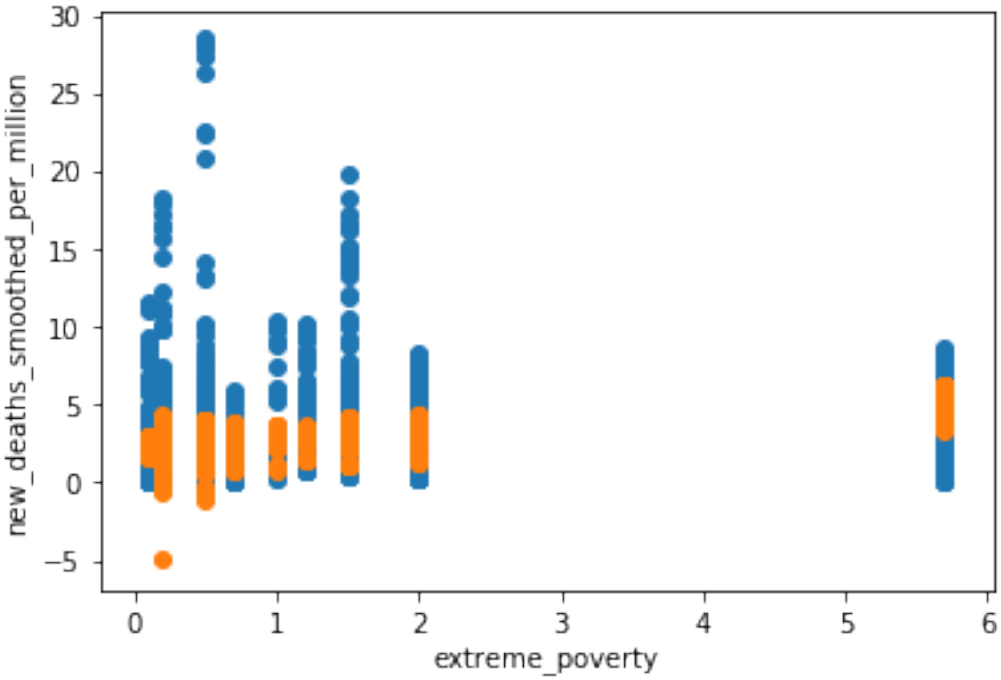


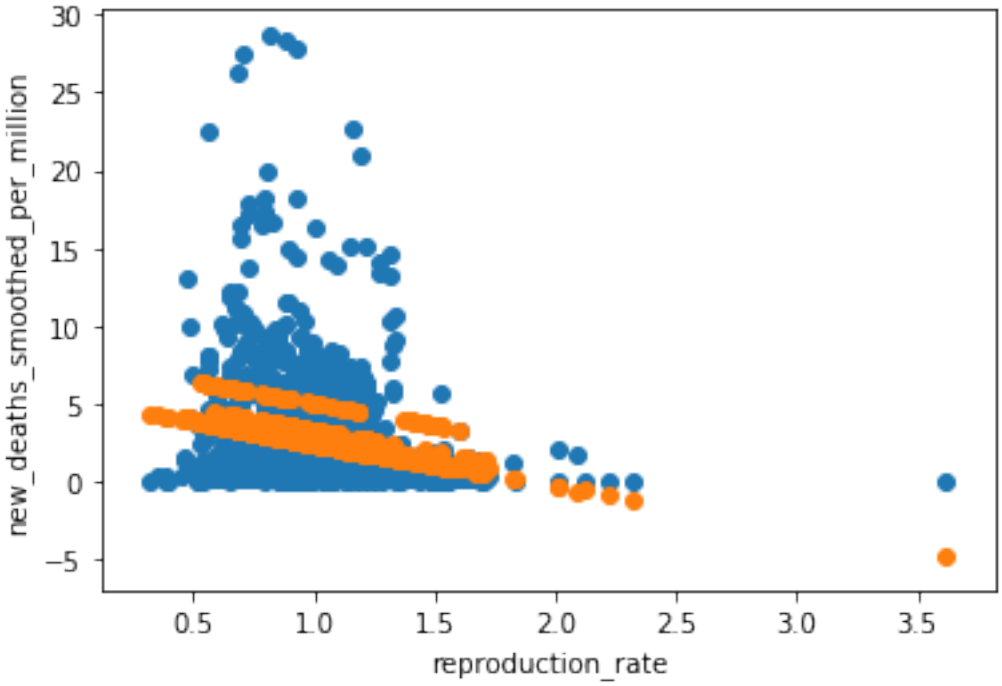


Features: **median_age, extreme_poverty, reproduction_rate** 1. r2_score is 0.06343537009193767 2. adjusted_r2_score is 0.05509799949928429


9
8
7
6
5
0
200
400
600
800
1000

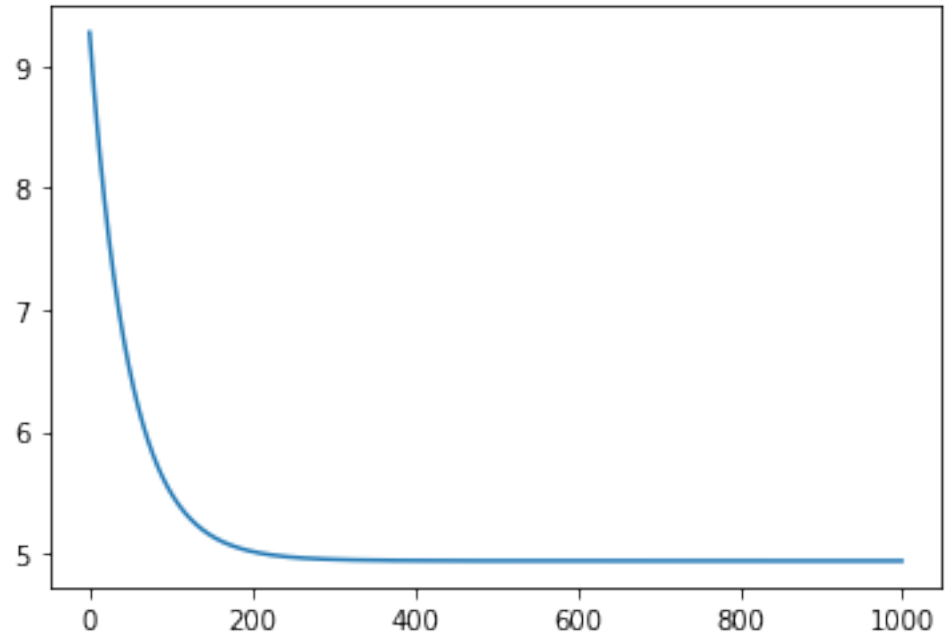

new_deaths_smoothed_per_million
30
25
20
15
10
5
0
−5
30.0
32.5
35.0
37.5
40.0
42.5
45.0
47.5
median_age

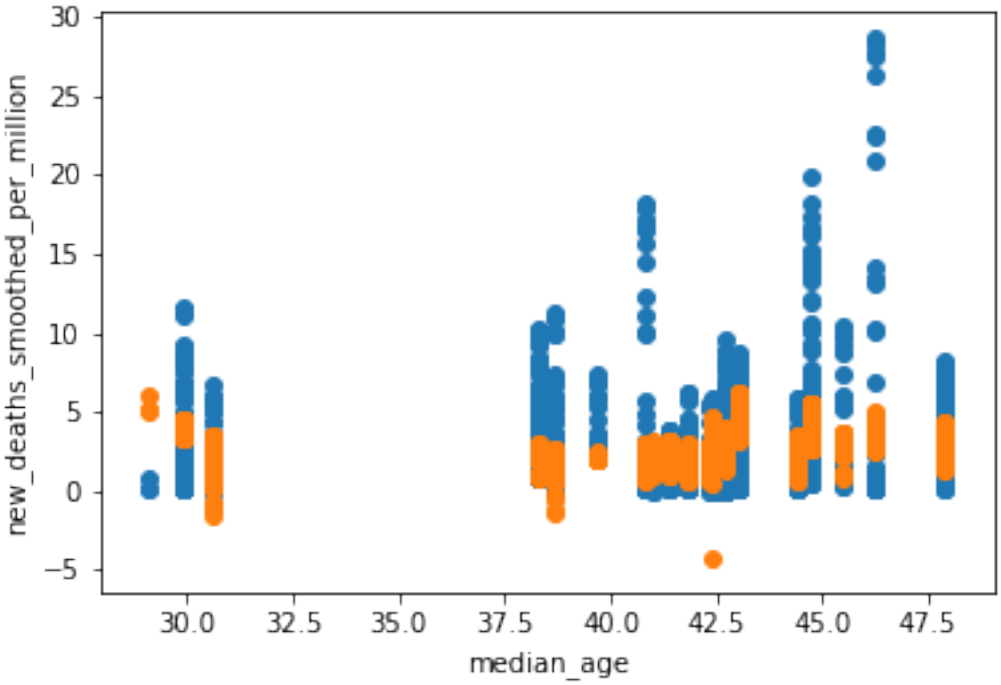


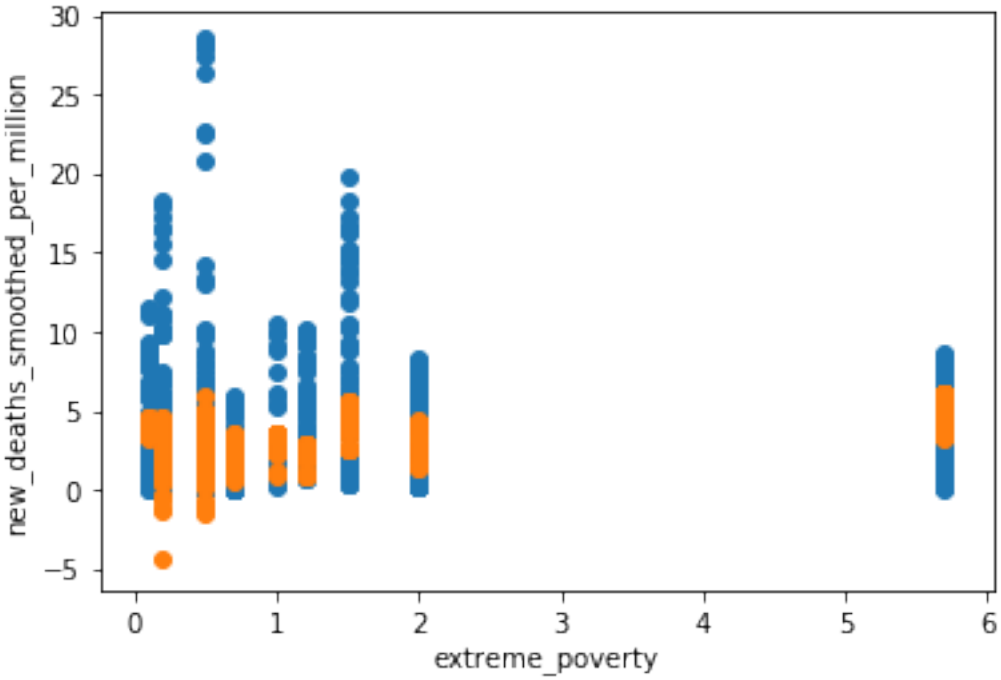


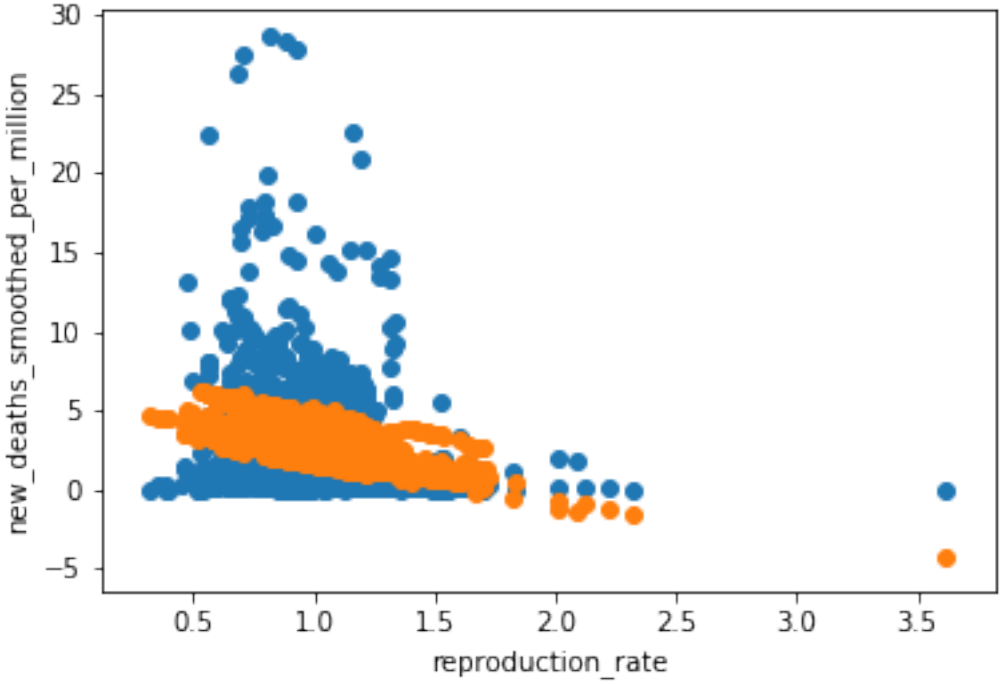
Features: **median_age, extreme_poverty, reproduction_rate, human_development_index** 1. r2_score is 0.11378428257773465 2. adjusted_r2_score is

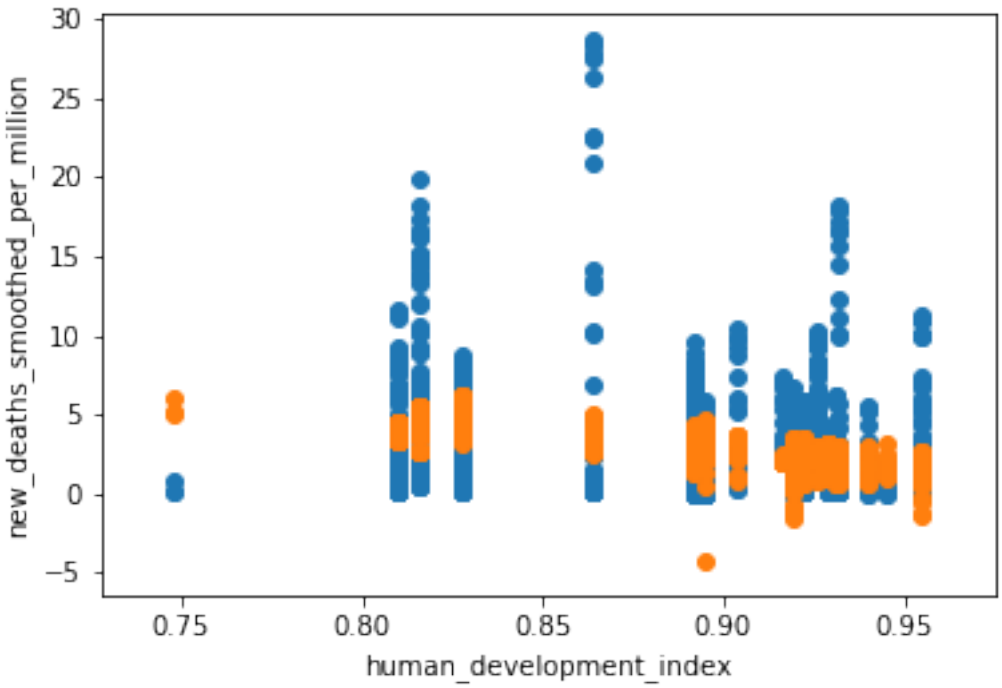
0.10589512188851558






new_deaths_smoothed_per_million
extreme_poverty
30
25
20
15
10
5
0
−5
0
1
2
3
4
5
6


new_deaths_smoothed_per_million
reproduction_rate
30
25
20
15
10
5
0
−5
0.5
1.0
1.5
2.0
2.5
3.0
3.5



13 Combining variables with high & low linear correlation

For experimental purposes, we developed a multivariate linear regression model using independent variables with both high and low linear correlation. Highest value obtained is given in **bold**. For reference, the highest `adjusted_r2_score` we have obtained thus far, from a combination of the top three features, is 0.6058531836499368.

	r2	adjusted_r2
icu_patients_per_million, median_age	0.59587	0.59045
Top 2 + Bottom 2 features	0.60694	0.601668
All 7 features	0.615201	**0.610039**

13.1 Analysis

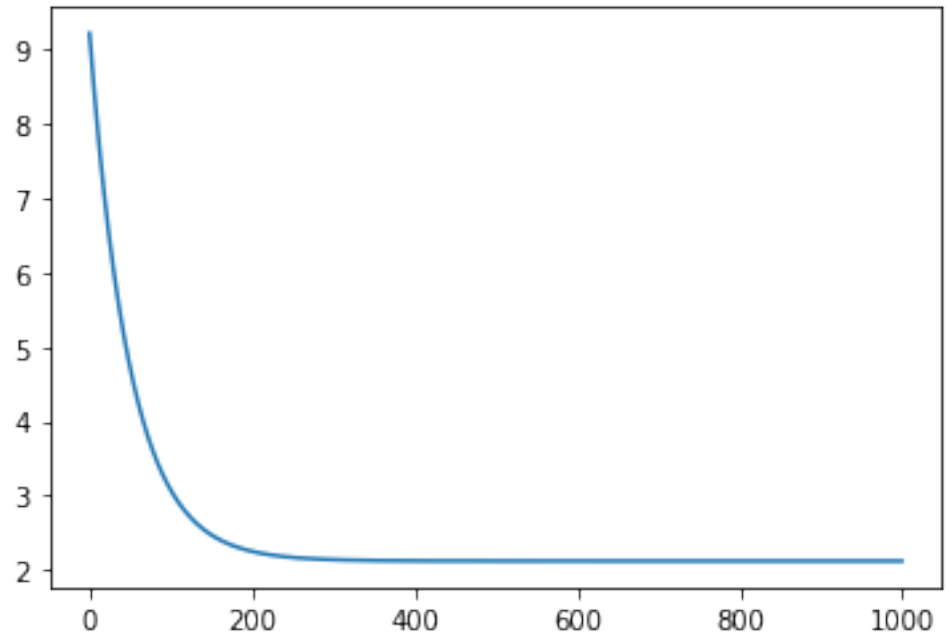
The `adjusted_r2_score` of all 7 features trumped that of the top 3 features by only 0.007. In other words, our multivariate linear regression model using just the top three features is **just as good** as combining all seven features. **Since the model with 3 features is far less computationally intensive, it should still be preferentially selected**.

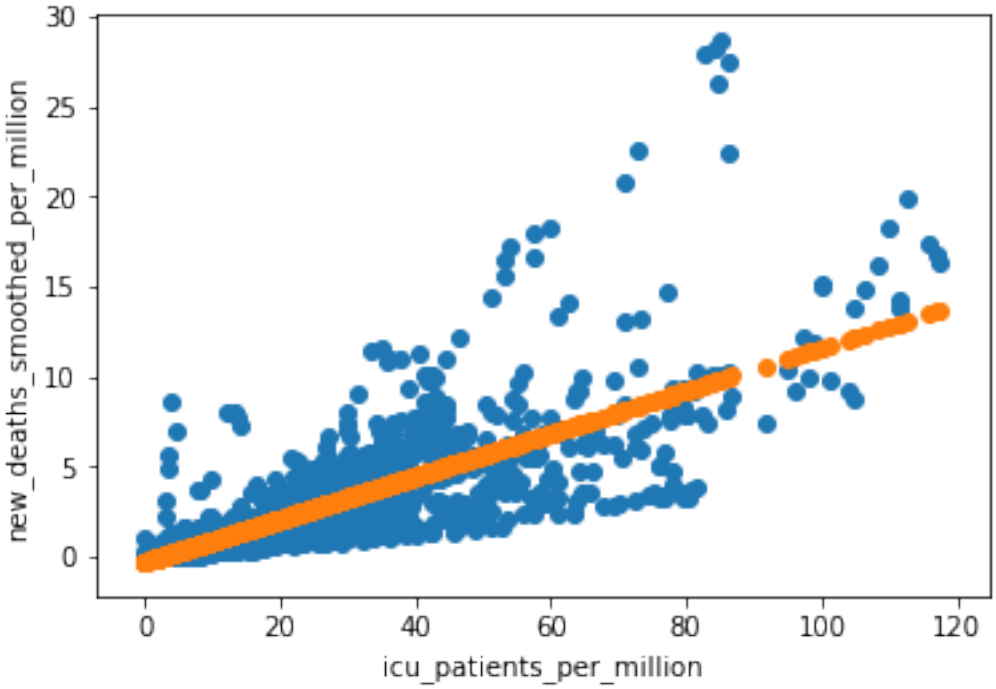
Section **??**

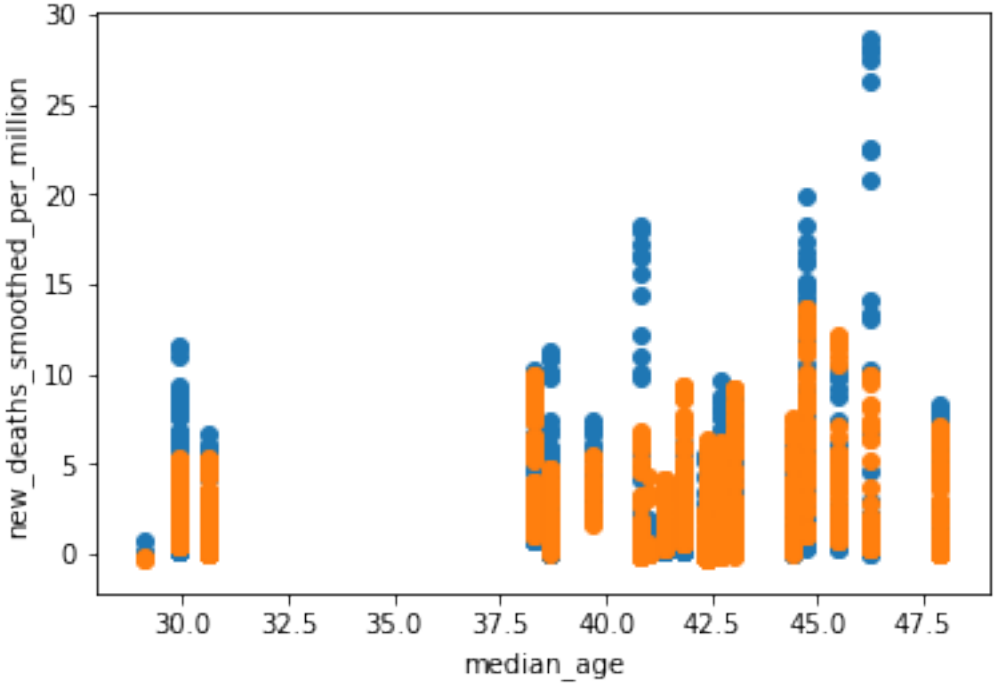
[8]:
```
table = [["", "r2", "adjusted_r2"]]

top_and_bottom_features = ["icu_patients_per_million", "median_age"]
```

```
_, r2, adjusted_r2 = lin_reg(df_dropna_one, top_and_bottom_features,␣
 ↪random_state=100)
table.append([", ".join(top_and_bottom_features), r2, adjusted_r2])


top_two_and_bottom_two_features = ["icu_patients_per_million",␣
 ↪"people_vaccinated_per_hundred", "median_age", "extreme_poverty"]
_, r2, adjusted_r2 = lin_reg(df_dropna_one, top_two_and_bottom_two_features,␣
 ↪random_state=100)
table.append([", ".join(top_two_and_bottom_two_features), r2, adjusted_r2])


all_seven_features = ["icu_patients_per_million", "stringency_index",␣
 ↪"median_age", "extreme_poverty", "people_vaccinated_per_hundred",␣
 ↪"reproduction_rate", "human_development_index"]
_, r2, adjusted_r2 = lin_reg(df_dropna_one, all_seven_features,␣
 ↪random_state=100)
table.append([", ".join(all_seven_features), r2, adjusted_r2])
```

Features: **icu_patients_per_million, median_age** 1. r2_score is 0.5958700863848244 2. adjusted_r2_score is 0.5922724907146597

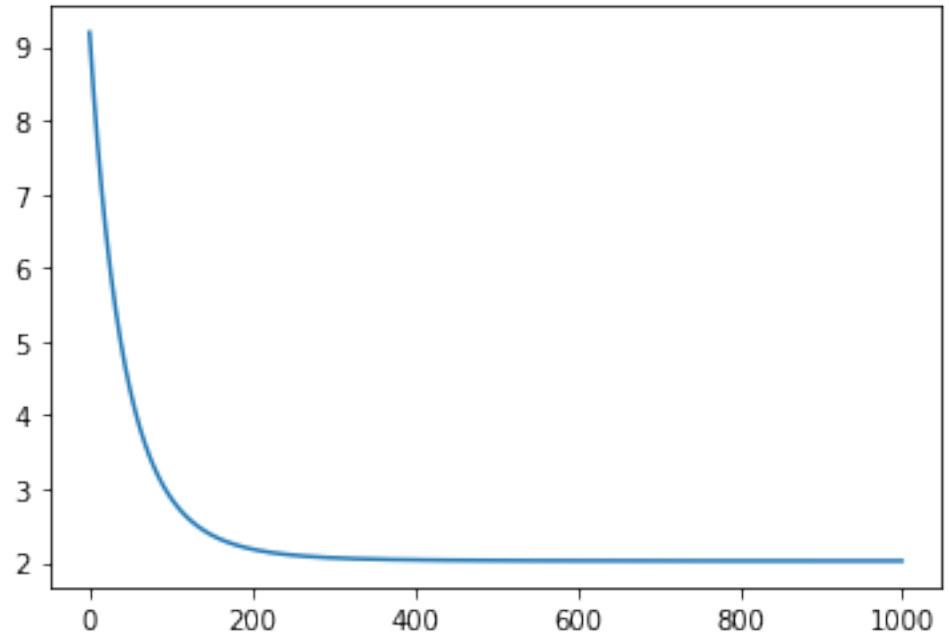


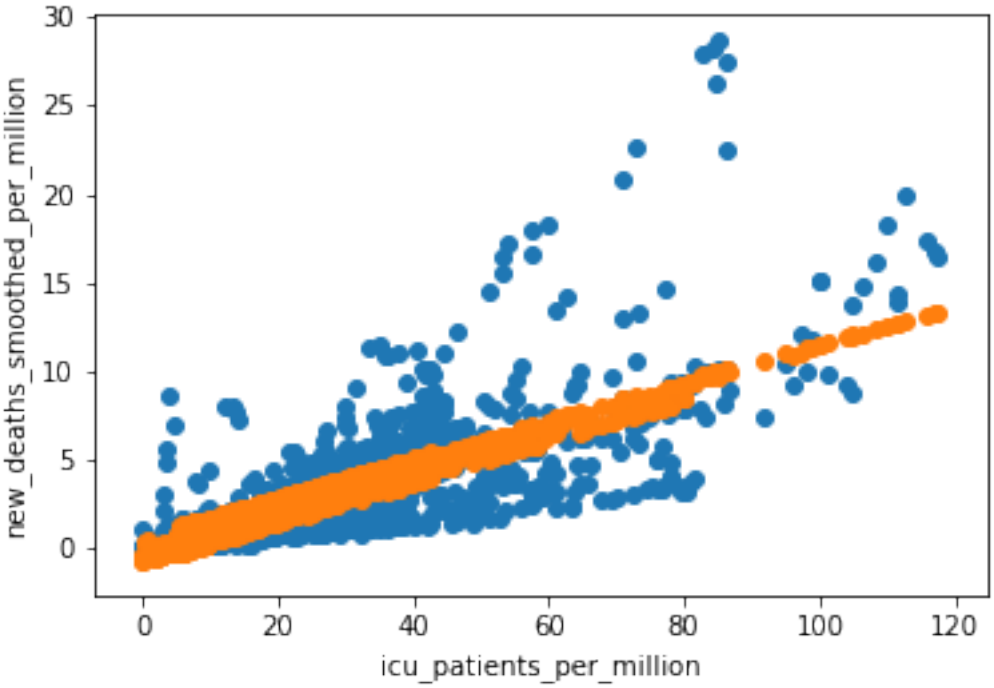


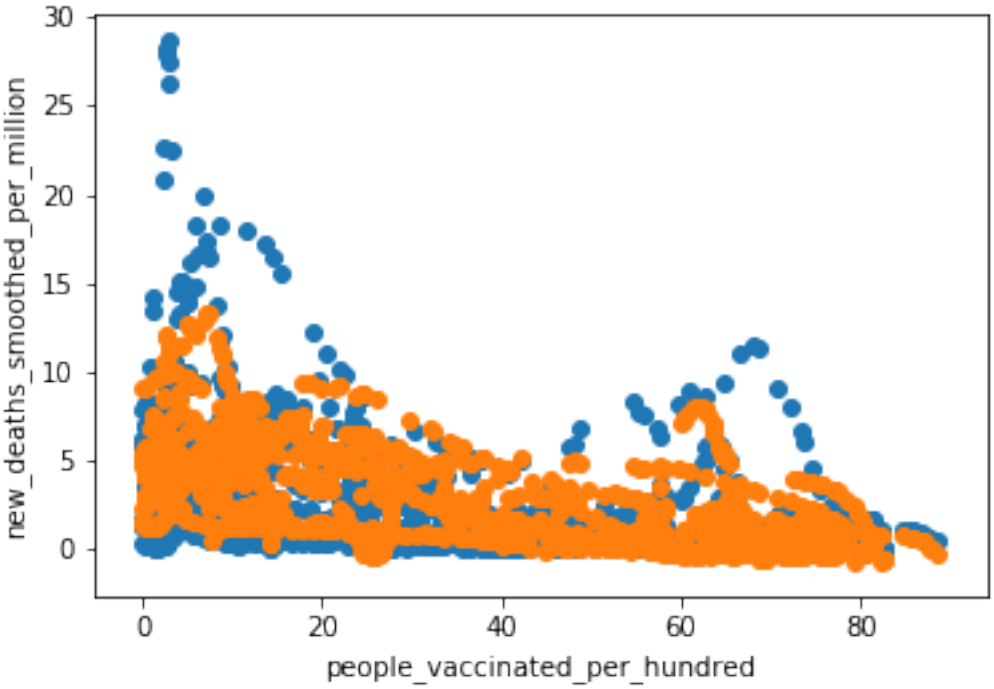


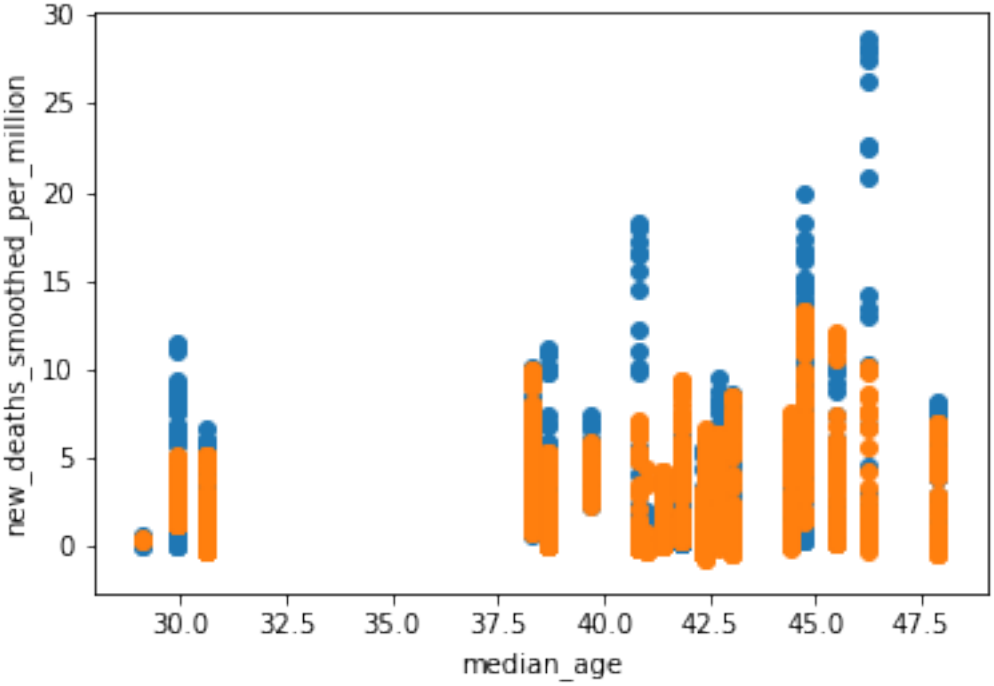
Features: **icu_patients_per_million, people_vaccinated_per_hundred, median_age, extreme_poverty** 1. r2_score is 0.6069398578564481 2. adjusted_r2_score is

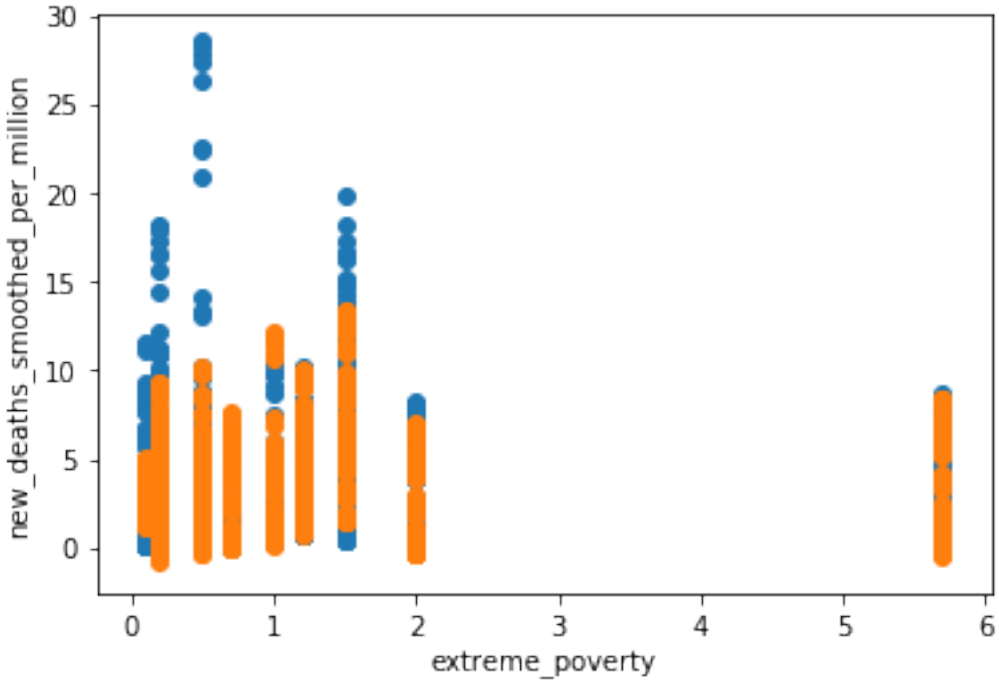
0.6034408061459713



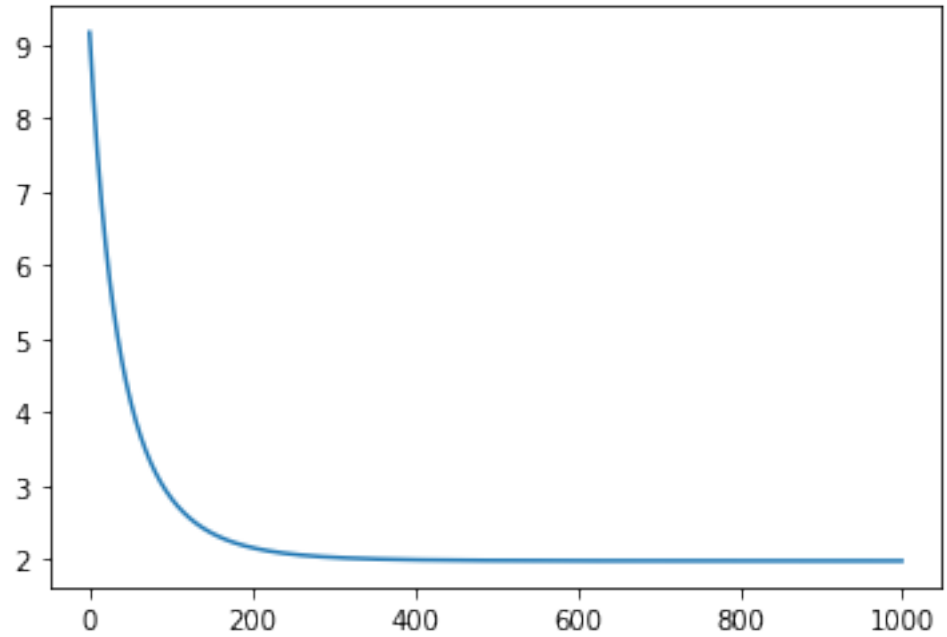


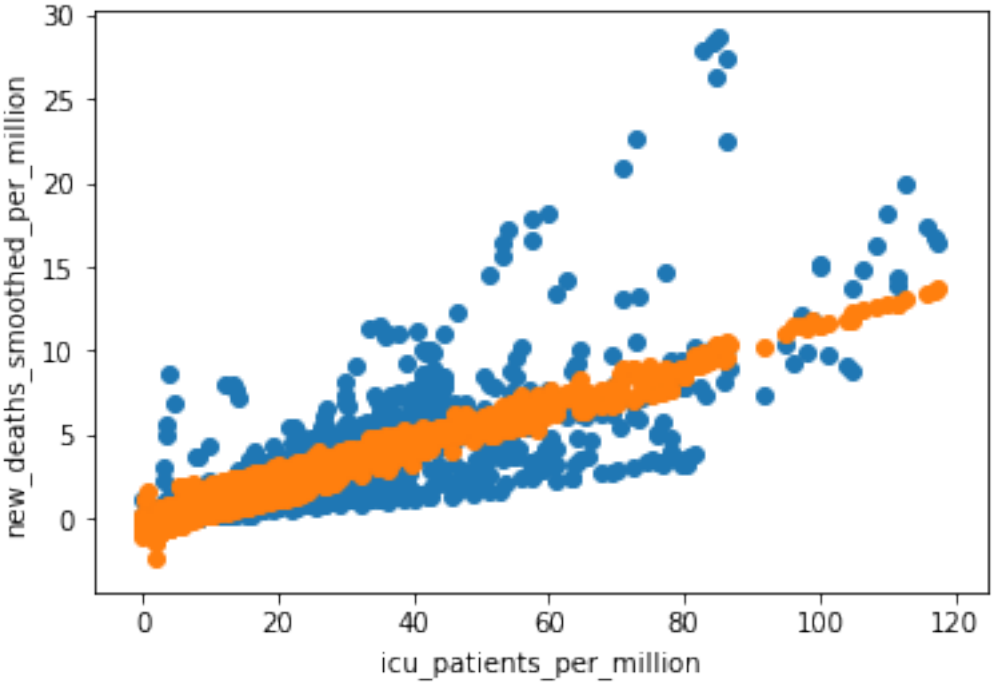

new_deaths_smoothed_per_million
people_vaccinated_per_hundred
0
5
10
15
20
25
30
0
20
40
60
80

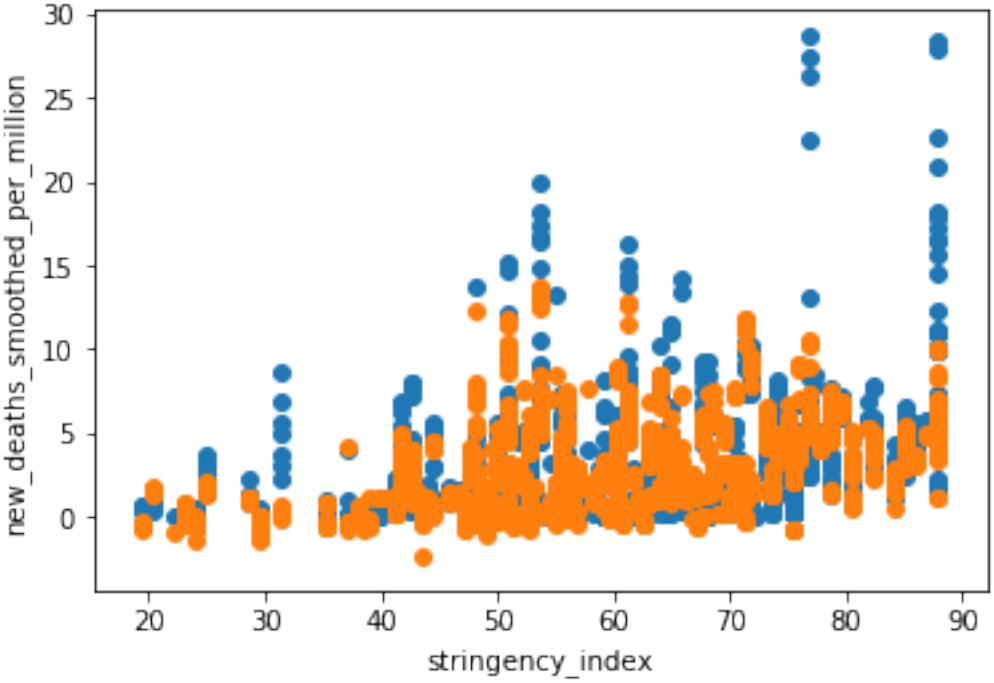

new_deaths_smoothed_per_million
median_age
0
5
10
15
20
25
30
30.0
32.5
35.0
37.5
40.0
42.5
45.0
47.5

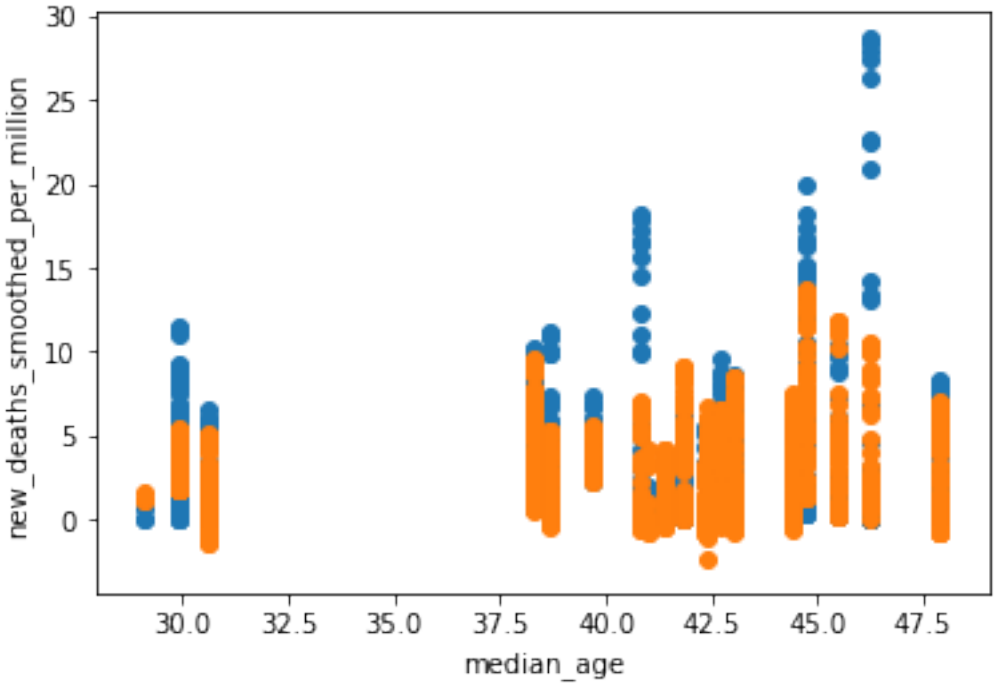


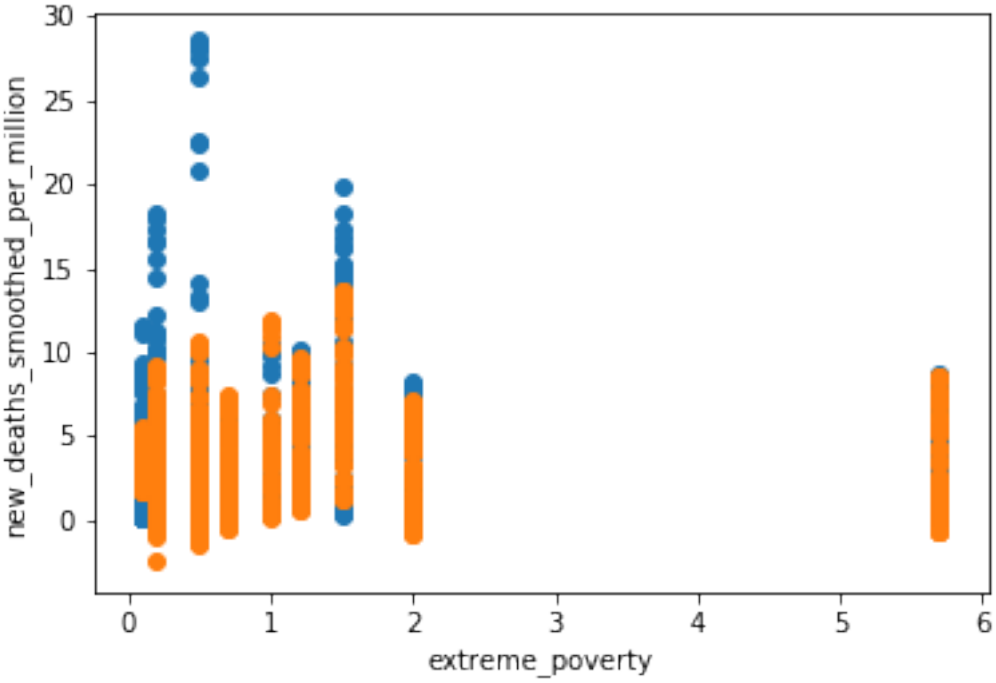
Features: **icu_patients_per_million, stringency_index, median_age, extreme_poverty, people_vaccinated_per_hundred, reproduction_rate, human_development_index** 1. r2_score is 0.615200713926279 2. adjusted_r2_score is 0.611775200993872

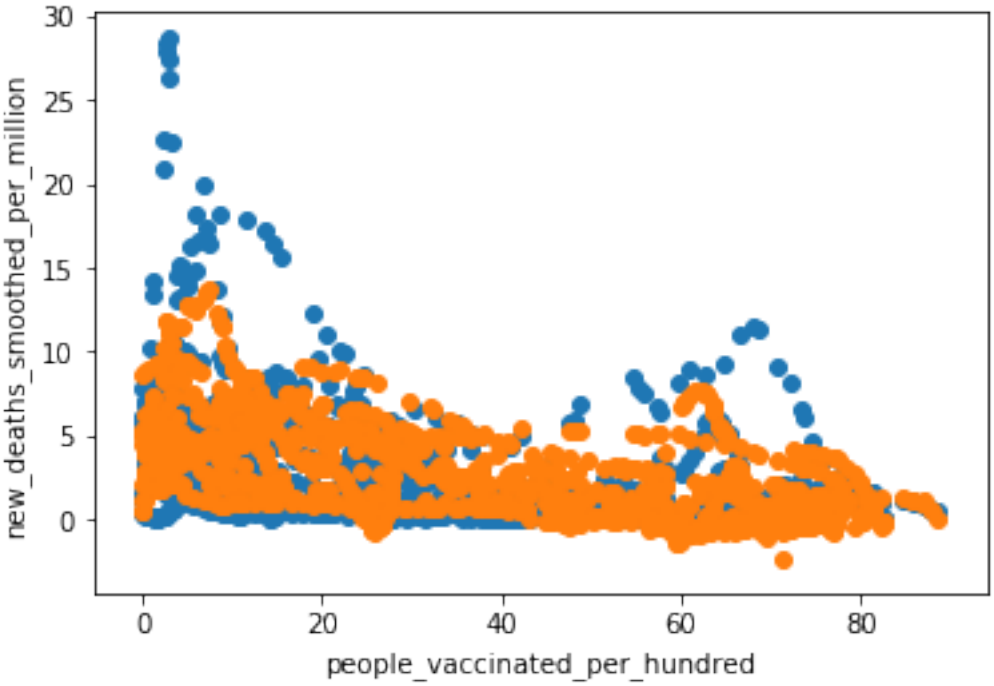


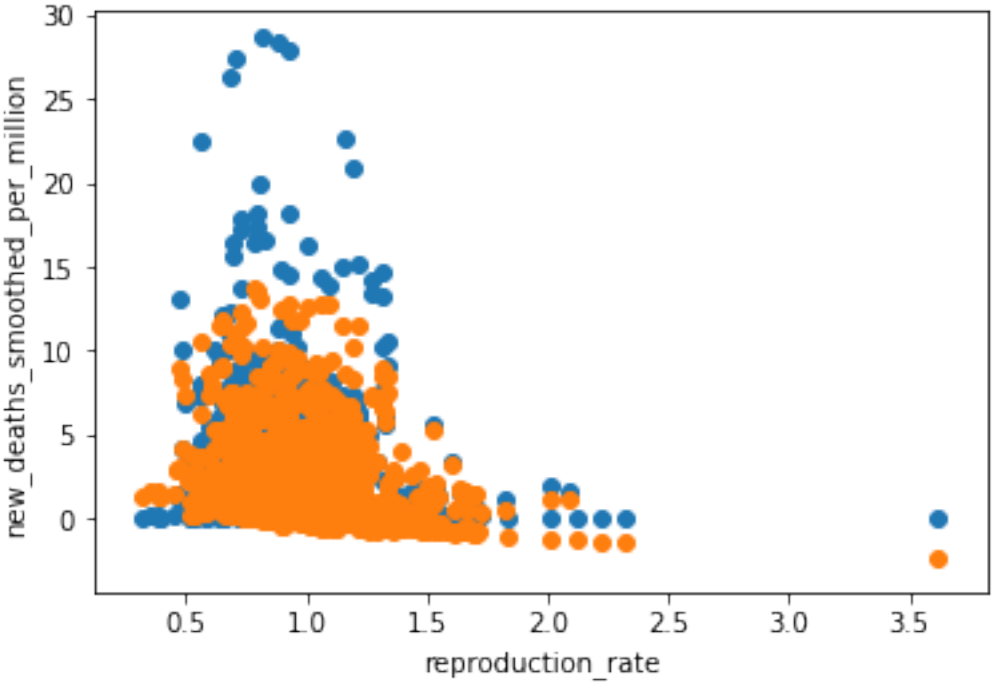

new_deaths_smoothed_per_million
icu_patients_per_million
0
5
10
15
20
25
30
0
20
40
60
80
100
120

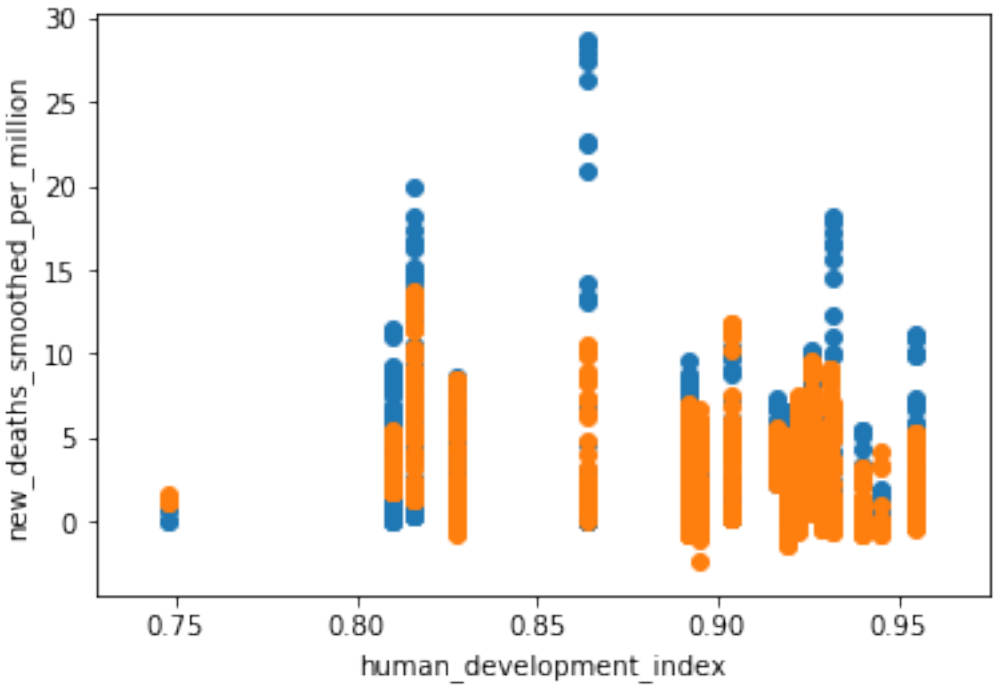

new_deaths_smoothed_per_million
stringency_index
0
5
10
15
20
25
30
20
30
40
50
60
70
80
90


new_deaths_smoothed_per_million
median_age
30
25
20
15
10
5
0
30.0
32.5
35.0
37.5
40.0
42.5
45.0
47.5


new_deaths_smoothed_per_million
extreme_poverty
30
25
20
15
10
5
0
0
1
2
3
4
5
6


new_deaths_smoothed_per_million
30
25
20
15
10
5
0
0
20
40
60
80
people_vaccinated_per_hundred


new_deaths_smoothed_per_million
30
25
20
15
10
5
0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
reproduction_rate



14 Feature transformation

In a bid to further improve our multivariate linear regression model, we attempted to square root and square our top three independent variables. For reference, the highest `adjusted_r2_score` we have obtained thus far, from a combination of the top three features, is 0.6058531836499368. Models which performed better have their `adjusted_r2_score` highlighted in **bold**

	r2	adjusted_r2
With icu_patients_per_million_squared	0.618999	**0.613889**
With icu_patients_per_million_square_root	0.598616	0.593232
With people_vaccinated_per_hundred_squared	0.61373	**0.60855**
With people_vaccinated_per_hundred_square_root	0.615667	**0.610512**
With stringency_index_squared	0.614383	**0.609211**
With stringency_index_square_root	0.609623	0.604387

Section **??**

[9]:
```
df_dropna_one["icu_patients_per_million_squared"] =␣
 ↪df_dropna_one["icu_patients_per_million"] ** 2
df_dropna_one["icu_patients_per_million_square_root"] = np.
 ↪sqrt(df_dropna_one["icu_patients_per_million"])
```

```
df_dropna_one["people_vaccinated_per_hundred_squared"] =␣
 ↪df_dropna_one["people_vaccinated_per_hundred"] ** 2
df_dropna_one["people_vaccinated_per_hundred_square_root"] = np.
 ↪sqrt(df_dropna_one["people_vaccinated_per_hundred"])

df_dropna_one["stringency_index_squared"] = df_dropna_one["stringency_index"]␣
 ↪** 2
df_dropna_one["stringency_index_square_root"] = np.
 ↪sqrt(df_dropna_one["stringency_index"])
```

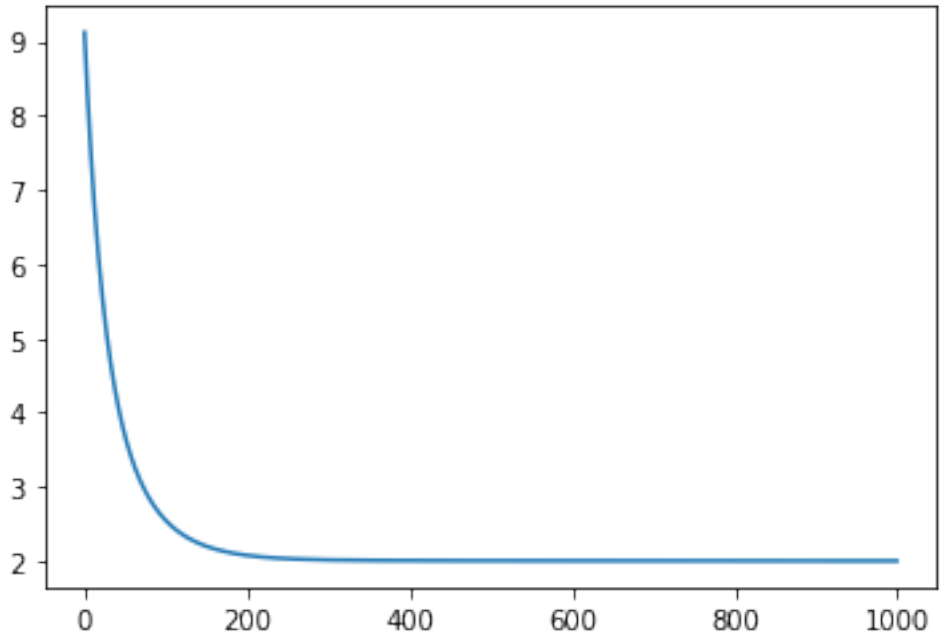
```
/Users/Sean/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
/Users/Sean/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/Users/Sean/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  after removing the cwd from sys.path.
/Users/Sean/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:5:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """
/Users/Sean/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:7:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

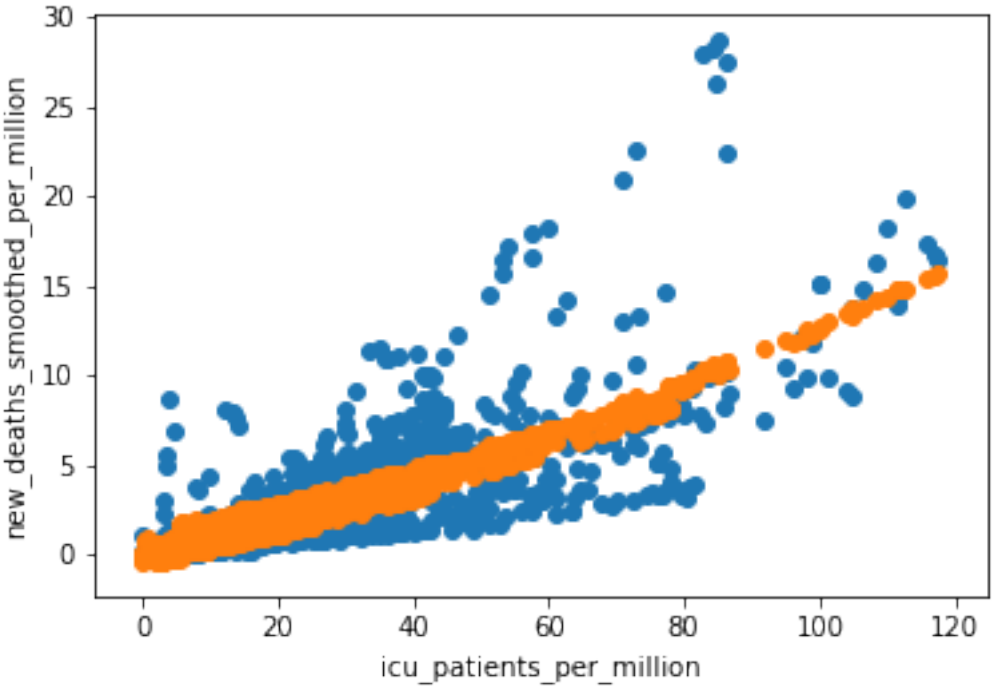
```
See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  import sys
/Users/Sean/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:8:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

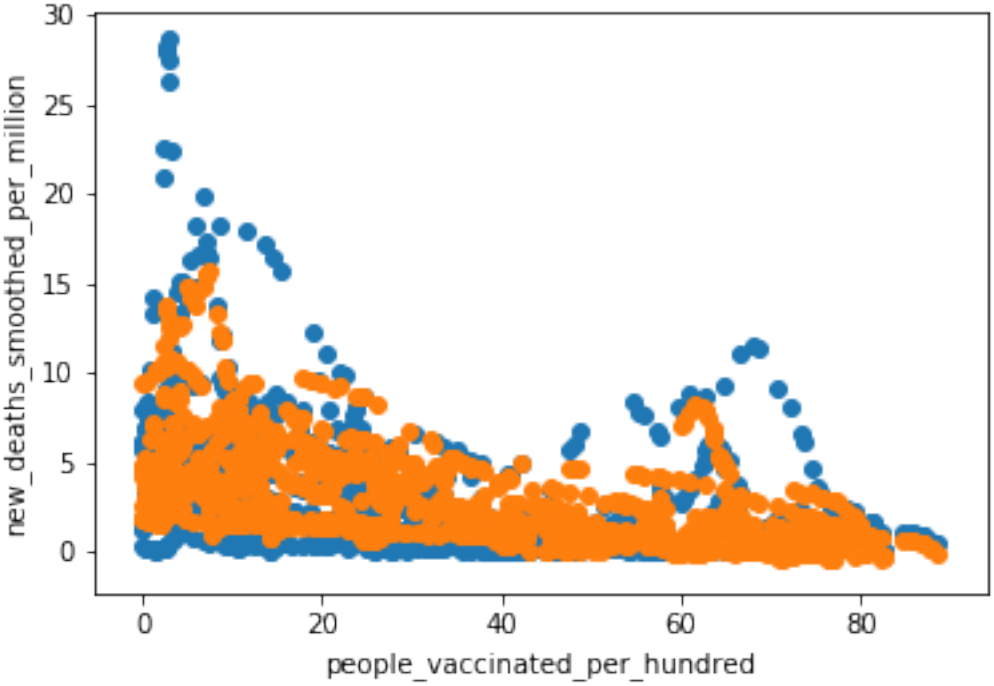
[10]:
```
table = [["", "r2", "adjusted_r2"]]

top_three_with_square = top_three_features +␣
 ↪["icu_patients_per_million_squared"]
_, r2, adjusted_r2 = lin_reg(df_dropna_one, top_three_with_square,␣
 ↪random_state=100,)
table.append([", ".join(top_three_with_square), r2, adjusted_r2])


top_three_with_square_root = top_three_features +␣
 ↪["icu_patients_per_million_square_root"]
_, r2, adjusted_r2 = lin_reg(df_dropna_one, top_three_with_square_root,␣
 ↪random_state=100,)
table.append([", ".join(top_three_with_square_root), r2, adjusted_r2])


top_three_with_square = top_three_features +␣
 ↪["people_vaccinated_per_hundred_squared"]
_, r2, adjusted_r2 = lin_reg(df_dropna_one, top_three_with_square,␣
 ↪random_state=100,)
table.append([", ".join(top_three_with_square), r2, adjusted_r2])


top_three_with_square_root = top_three_features +␣
 ↪["people_vaccinated_per_hundred_square_root"]
_, r2, adjusted_r2 = lin_reg(df_dropna_one, top_three_with_square_root,␣
 ↪random_state=100,)
table.append([", ".join(top_three_with_square_root), r2, adjusted_r2])


top_three_with_square = top_three_features + ["stringency_index_squared"]
_, r2, adjusted_r2 = lin_reg(df_dropna_one, top_three_with_square,␣
 ↪random_state=100,)
table.append([", ".join(top_three_with_square), r2, adjusted_r2])
```

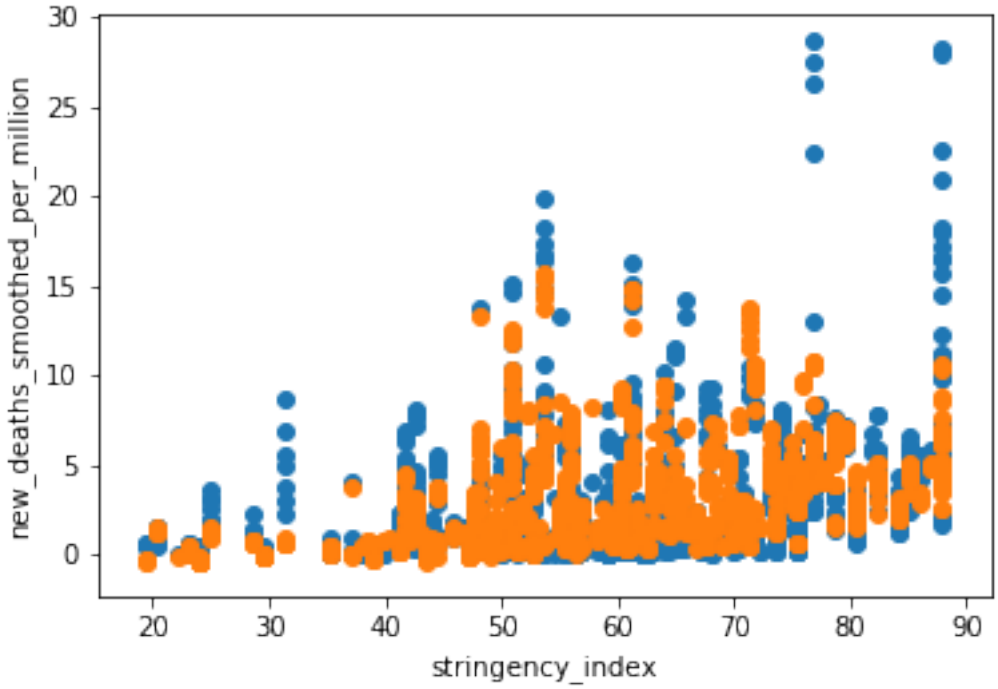
```
top_three_with_square_root = top_three_features +␣
 ↪["stringency_index_square_root"]
_, r2, adjusted_r2 = lin_reg(df_dropna_one, top_three_with_square_root,␣
 ↪random_state=100)
table.append([", ".join(top_three_with_square_root), r2, adjusted_r2])
```

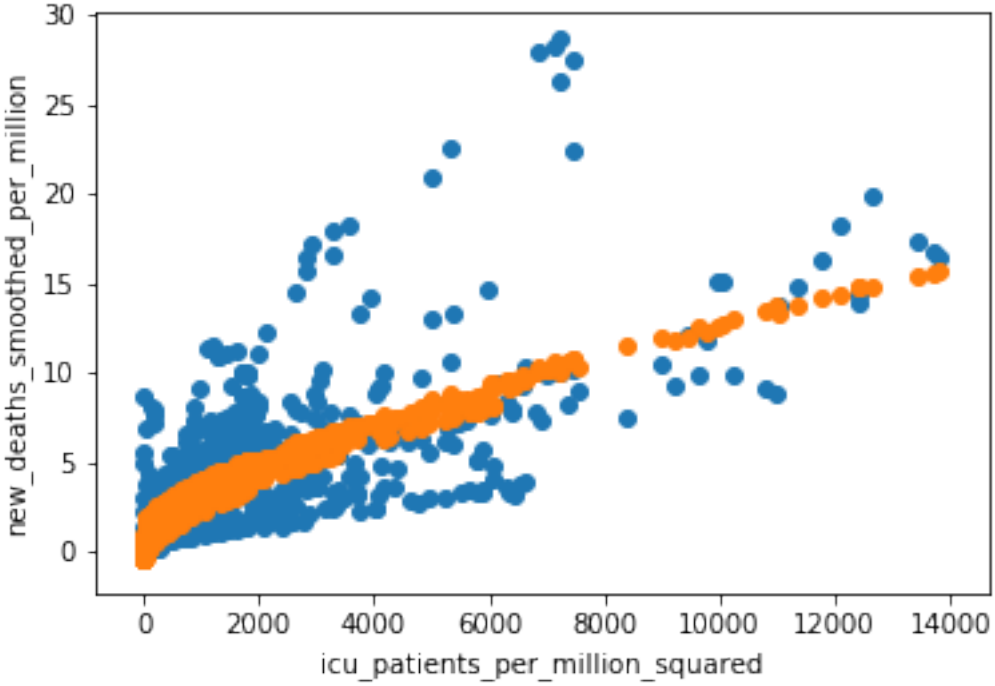
Features: **icu_patients_per_million, people_vaccinated_per_hundred, stringency_index, icu_patients_per_million_squared** 1. r2_score is 0.6189990734542092 2. adjusted_r2_score is 0.6138887778671569




new_deaths_smoothed_per_million
icu_patients_per_million
0
5
10
15
20
25
30
0
20
40
60
80
100
120

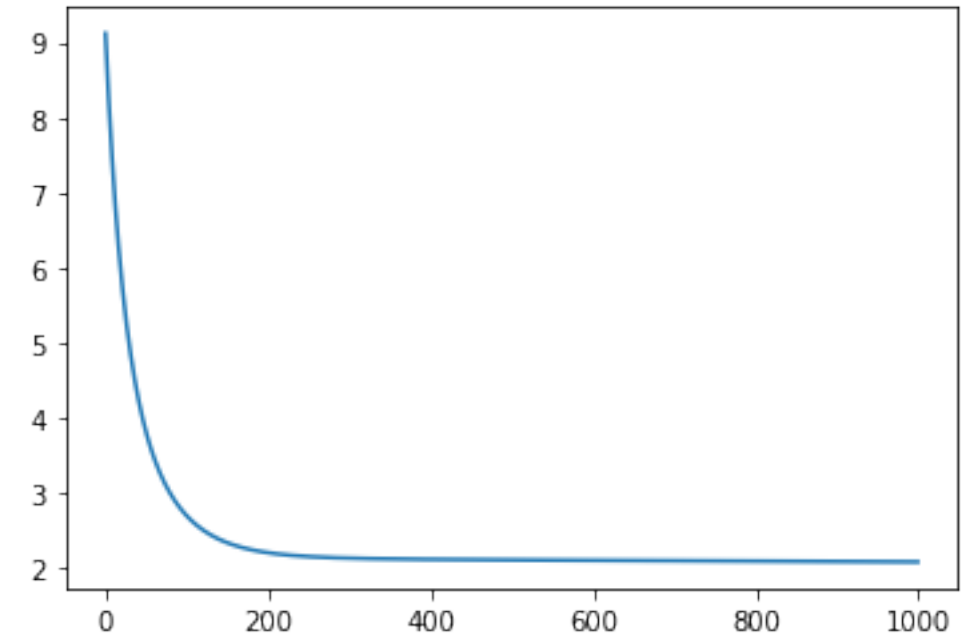

new_deaths_smoothed_per_million
people_vaccinated_per_hundred
0
5
10
15
20
25
30
0
20
40
60
80

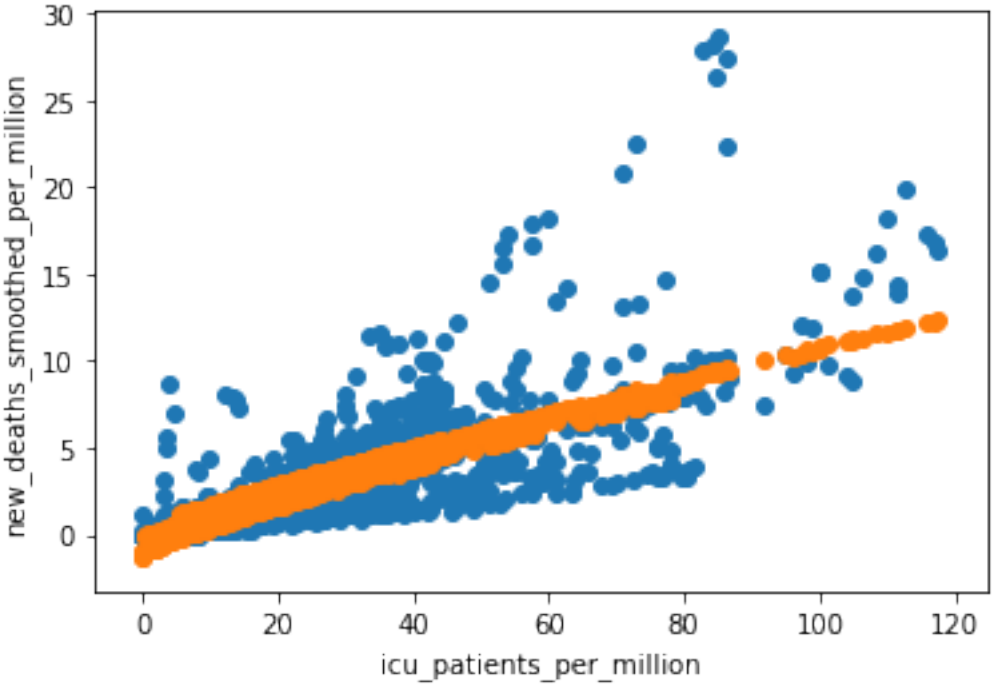


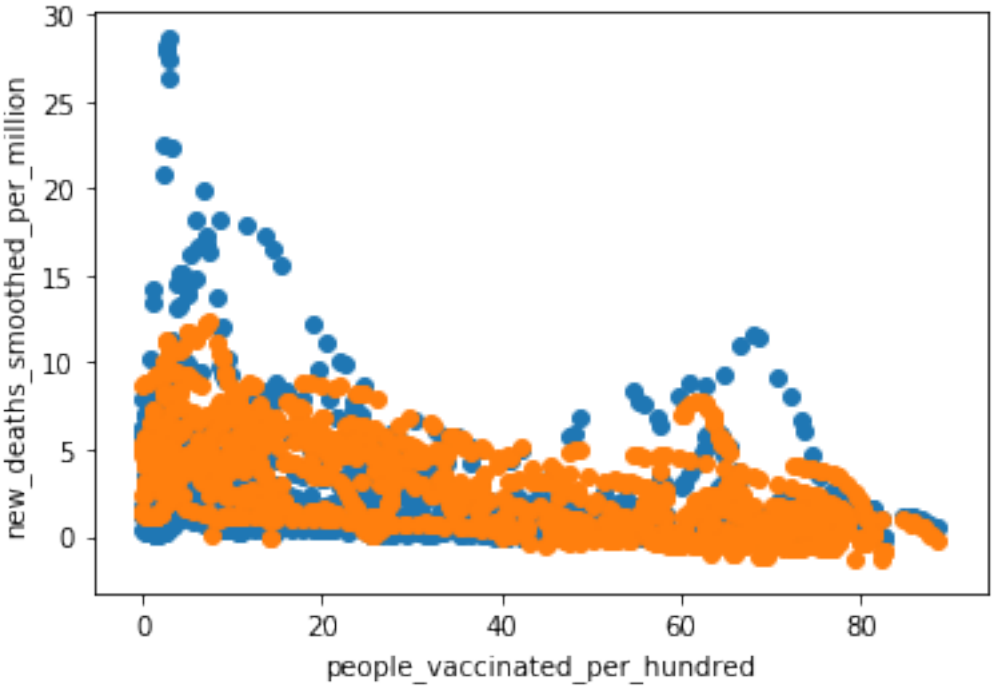


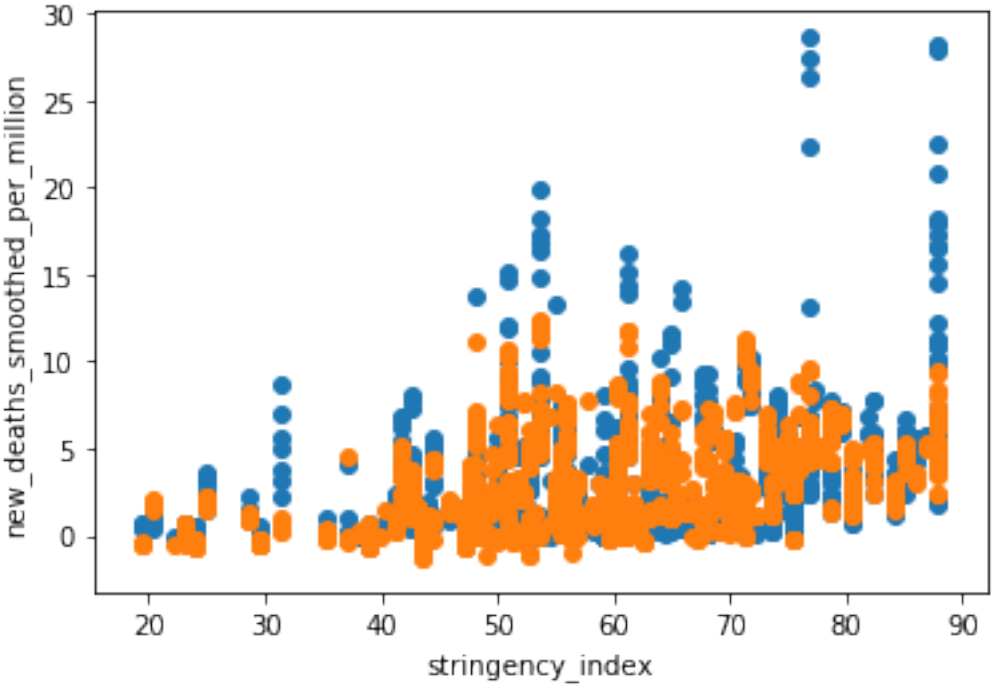
Features: **icu_patients_per_million, people_vaccinated_per_hundred, stringency_index, icu_patients_per_million_square_root** 1. r2_score is 0.5986161134407233

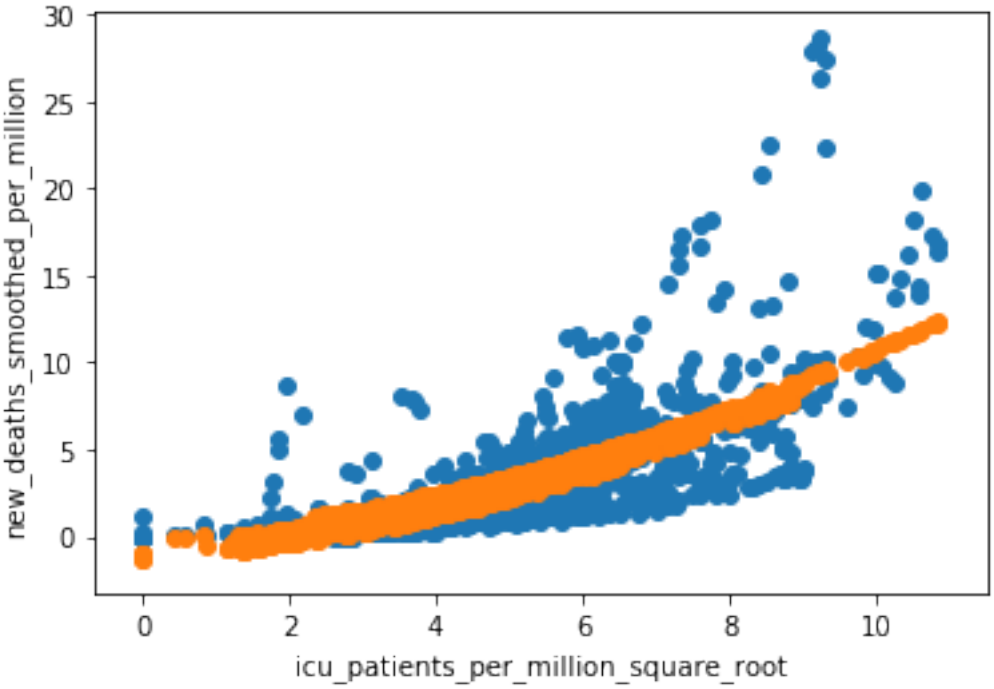
2. adjusted_r2_score is 0.5932324249473798



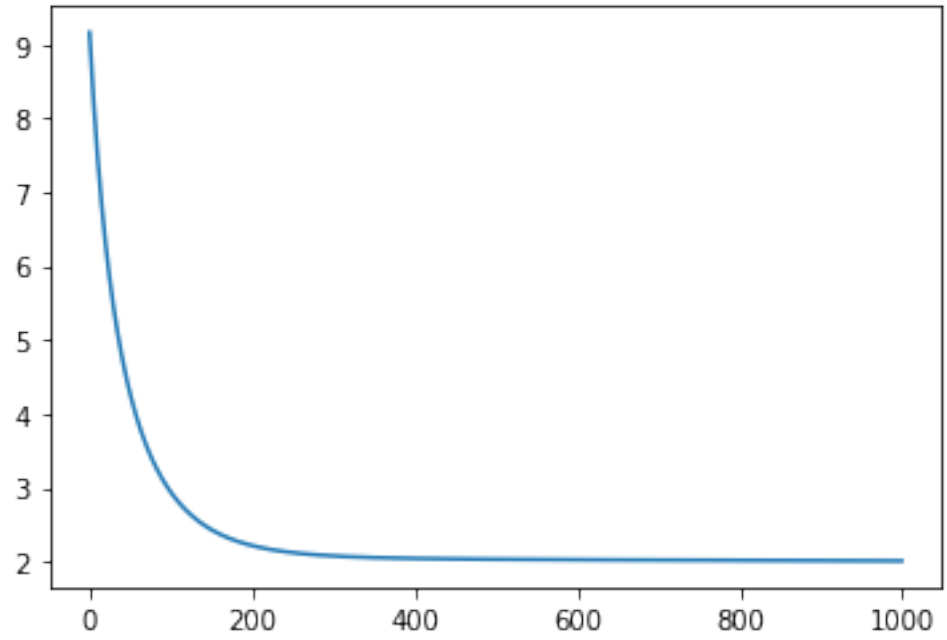


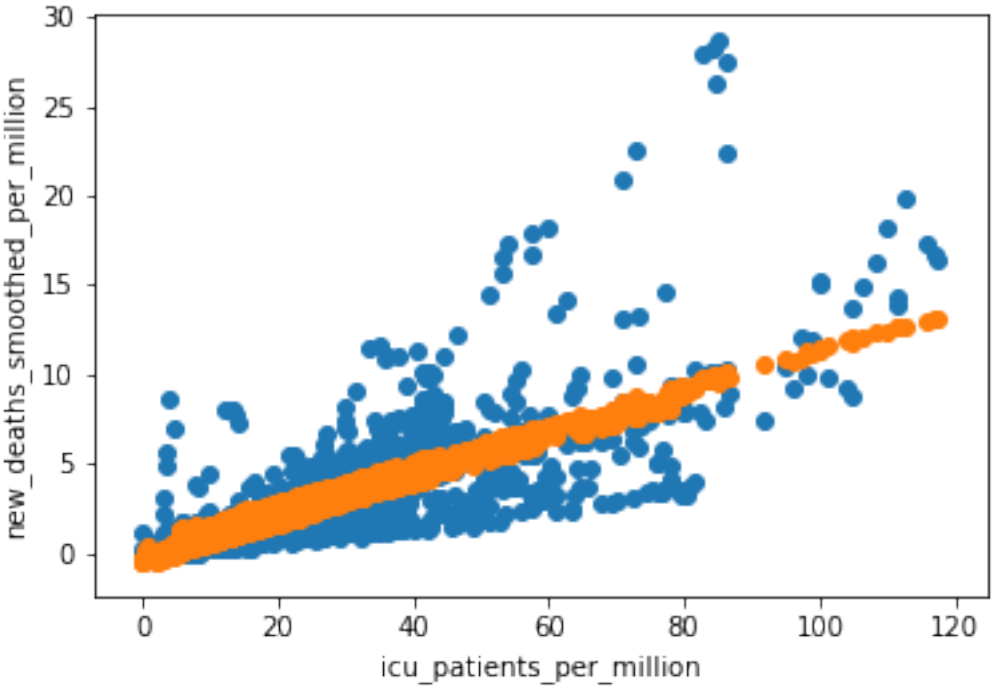

new_deaths_smoothed_per_million
people_vaccinated_per_hundred
0
5
10
15
20
25
30
0
20
40
60
80

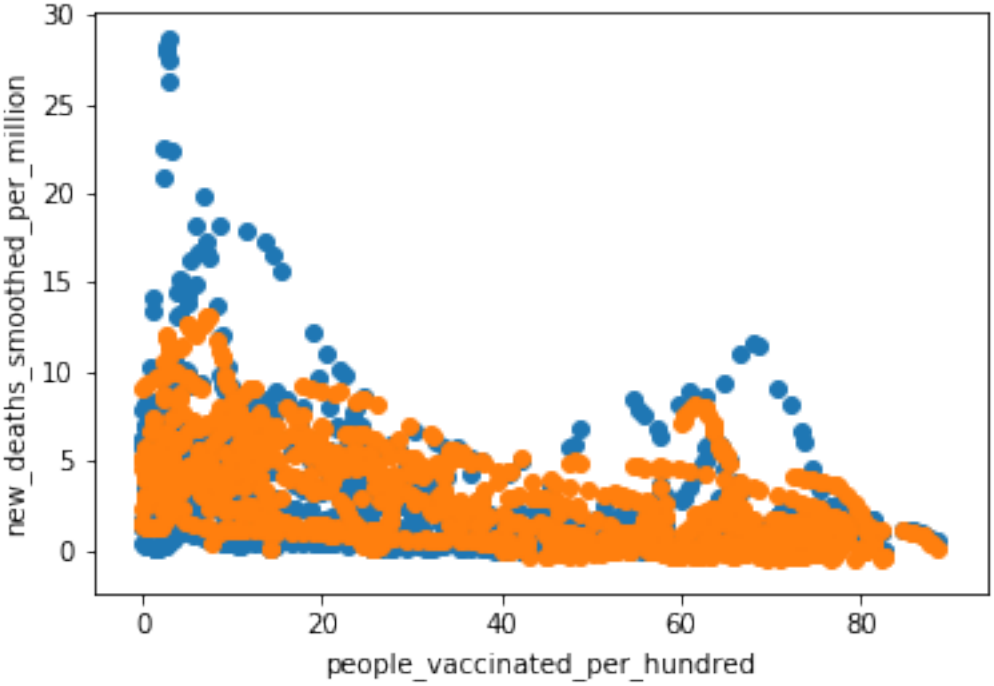

new_deaths_smoothed_per_million
stringency_index
0
5
10
15
20
25
30
20
30
40
50
60
70
80
90

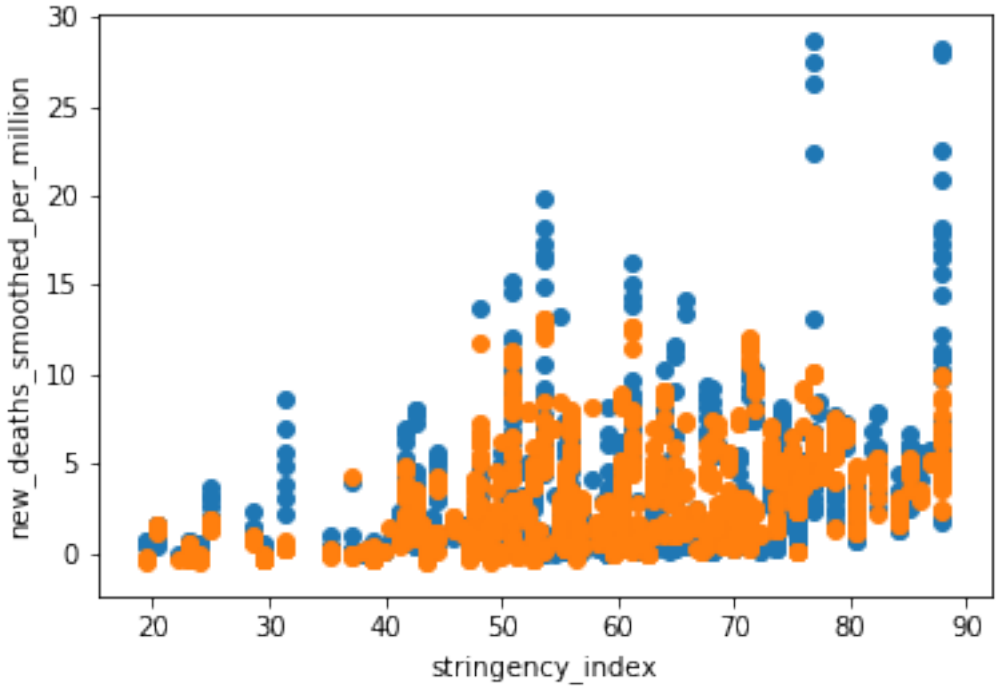


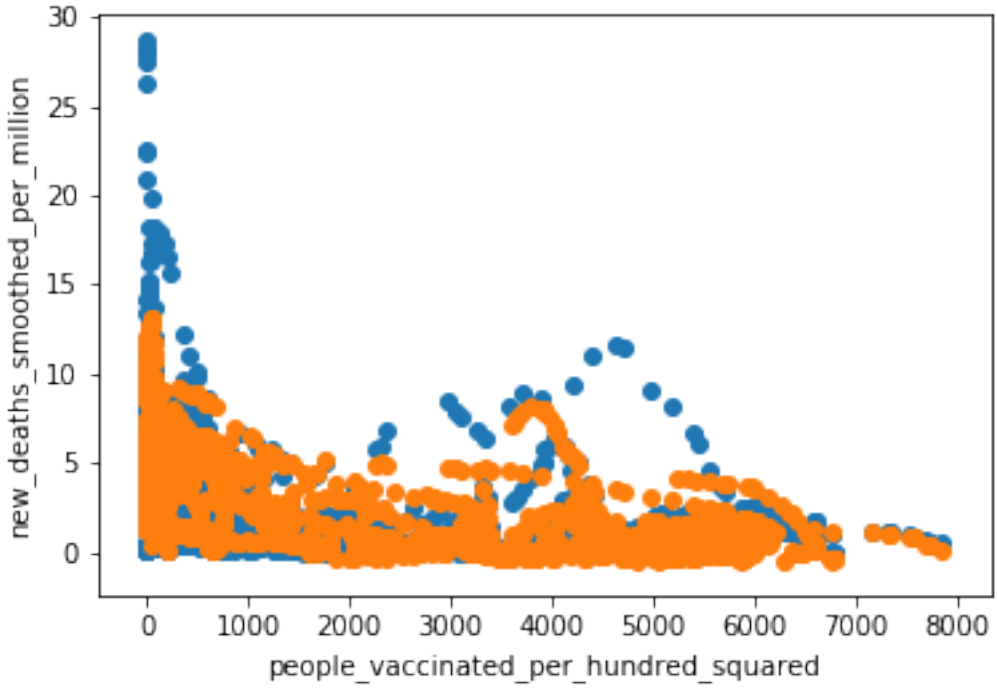
Features: **icu_patients_per_million, people_vaccinated_per_hundred, stringency_index, people_vaccinated_per_hundred_squared** 1. r2_score is 0.6137304829084251 2. adjusted_r2_score is 0.6085495206821596




new_deaths_smoothed_per_million
icu_patients_per_million
0
5
10
15
20
25
30
0
20
40
60
80
100
120

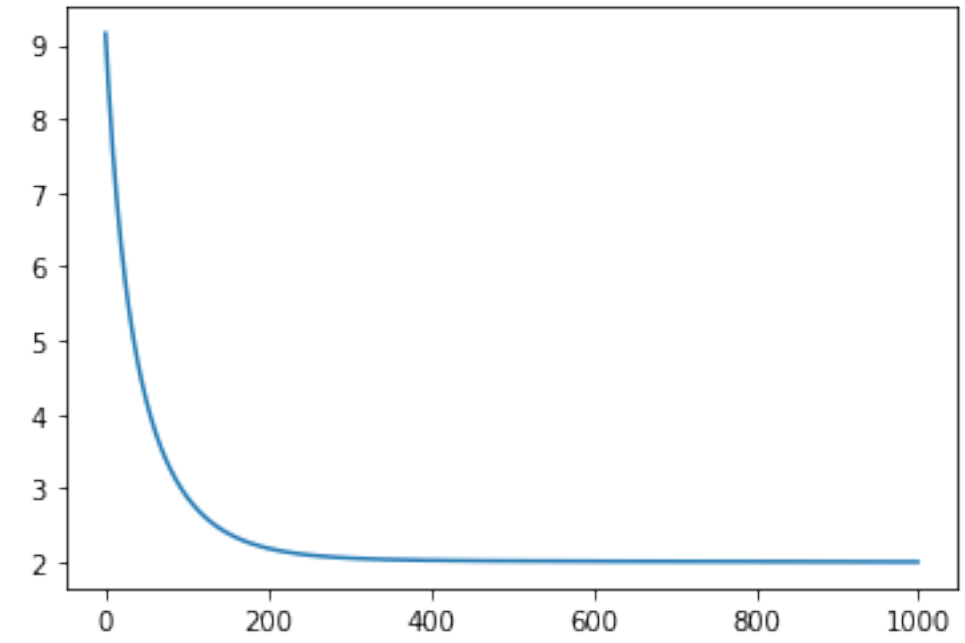

new_deaths_smoothed_per_million
people_vaccinated_per_hundred
0
5
10
15
20
25
30
0
20
40
60
80

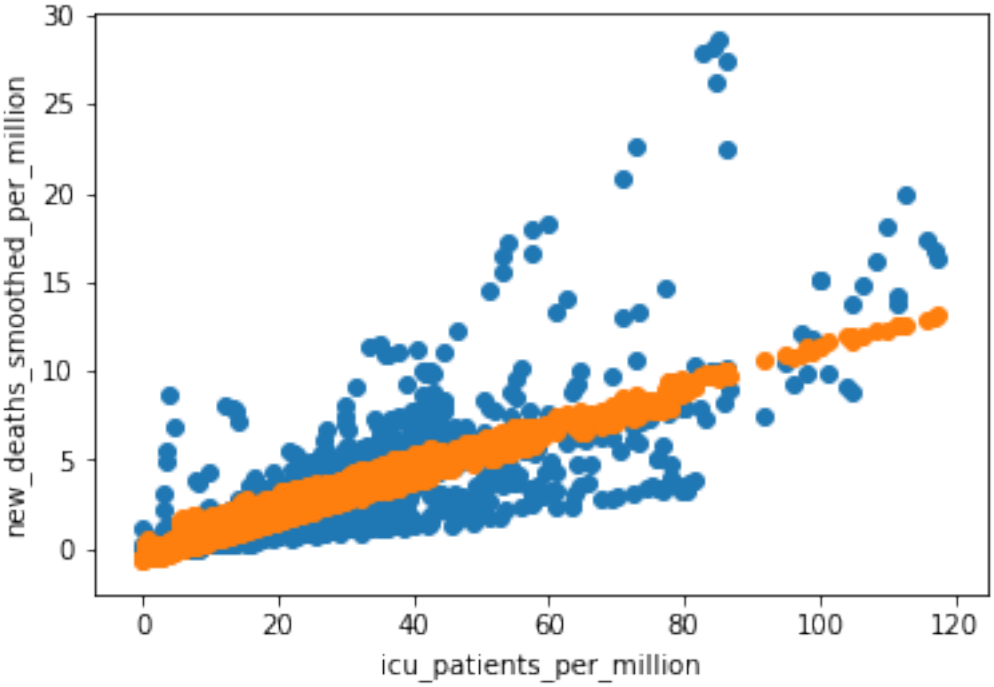


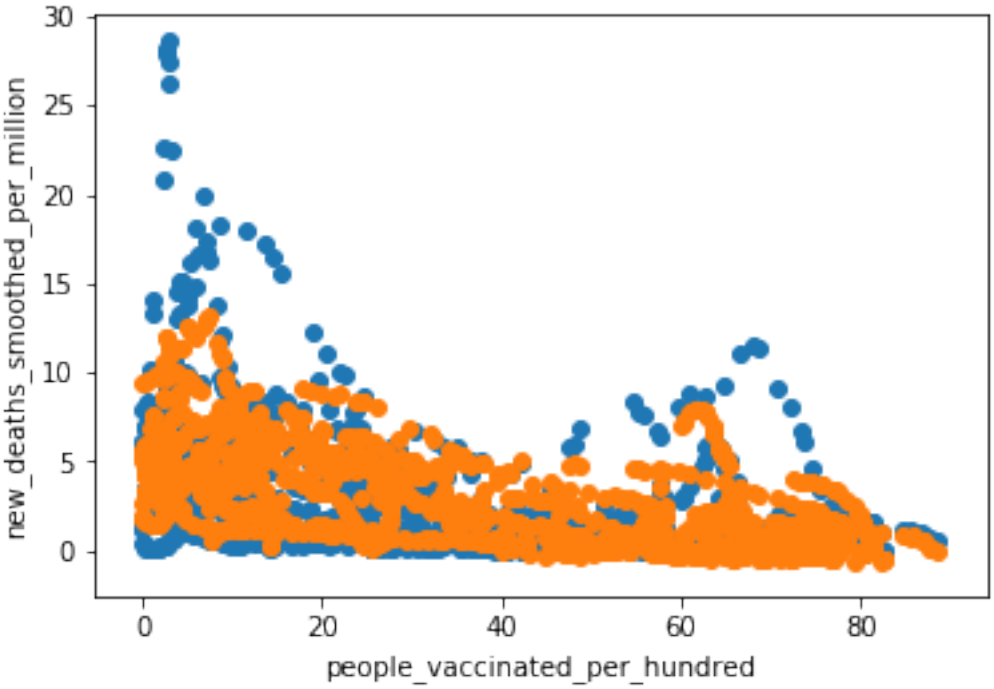


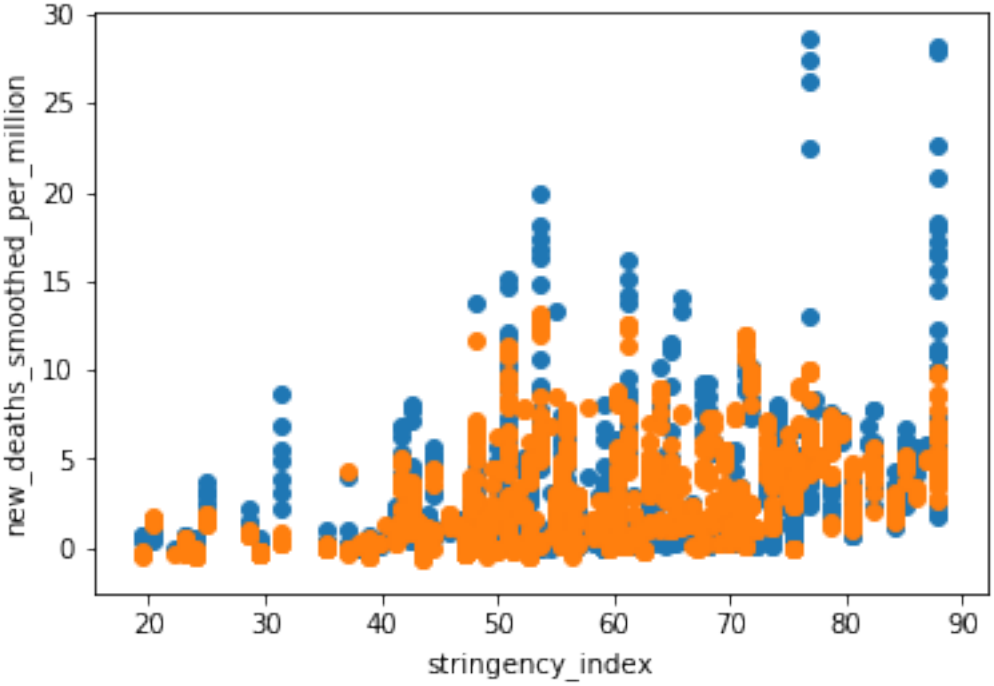
Features: **icu_patients_per_million, people_vaccinated_per_hundred, stringency_index, people_vaccinated_per_hundred_square_root** 1. r2_score is

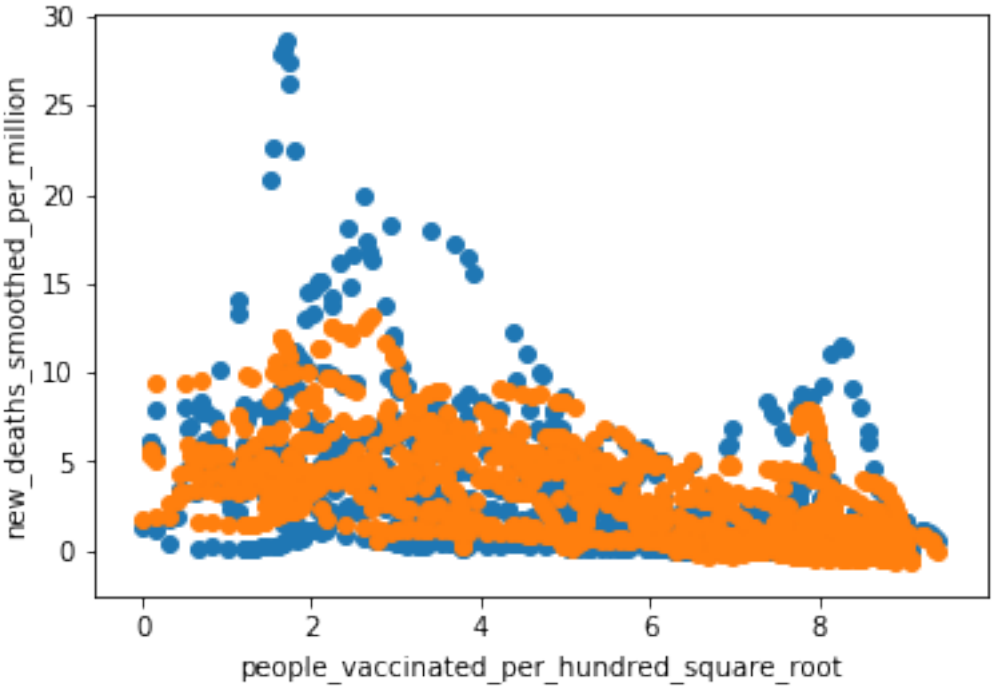
0.6156672803100993 2. adjusted_r2_score is 0.610512295992351



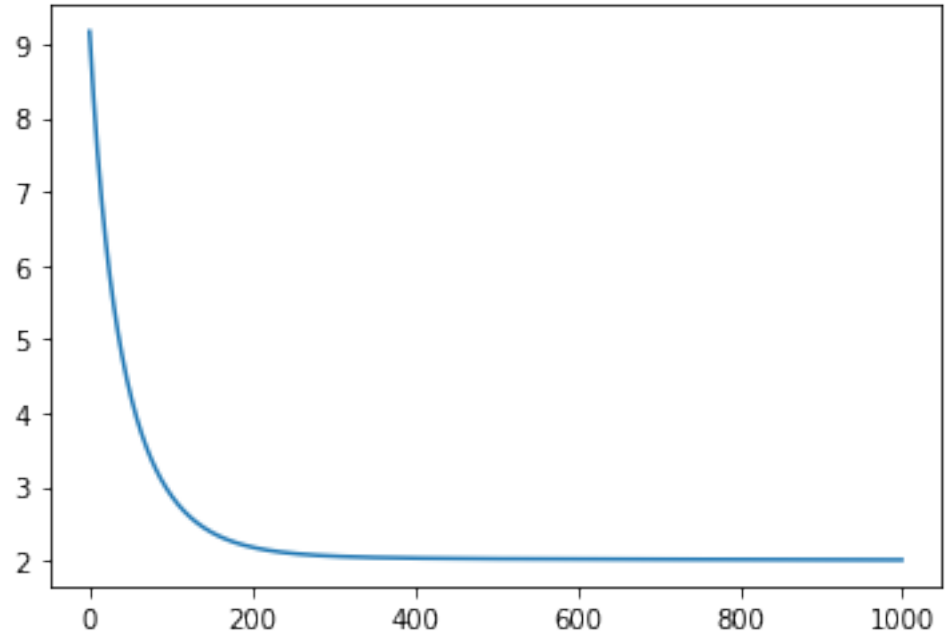


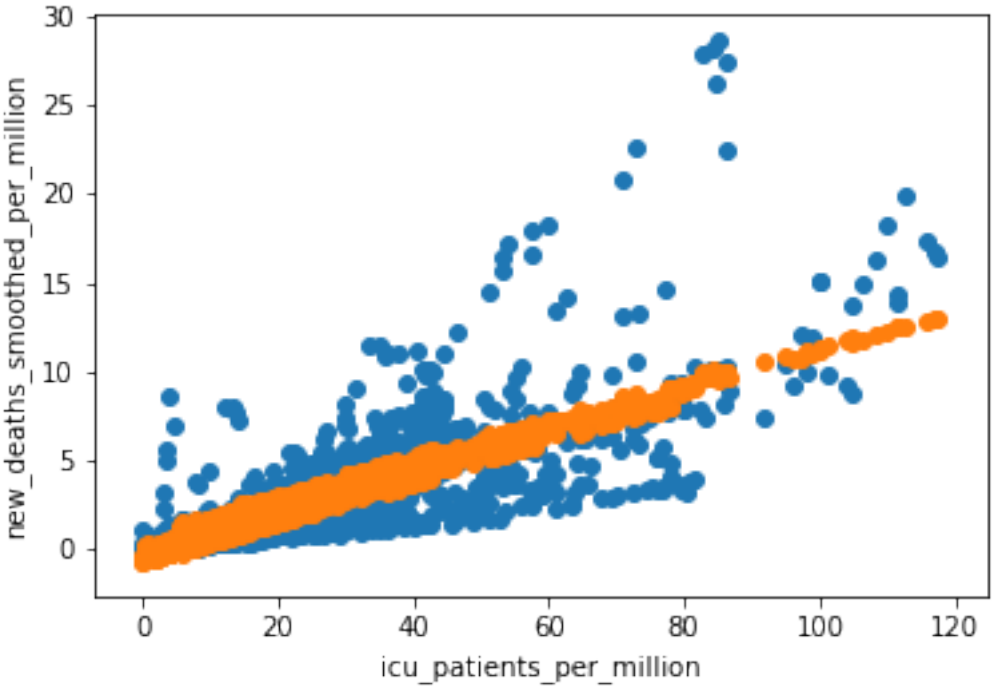

new_deaths_smoothed_per_million
people_vaccinated_per_hundred
0
5
10
15
20
25
30
0
20
40
60
80

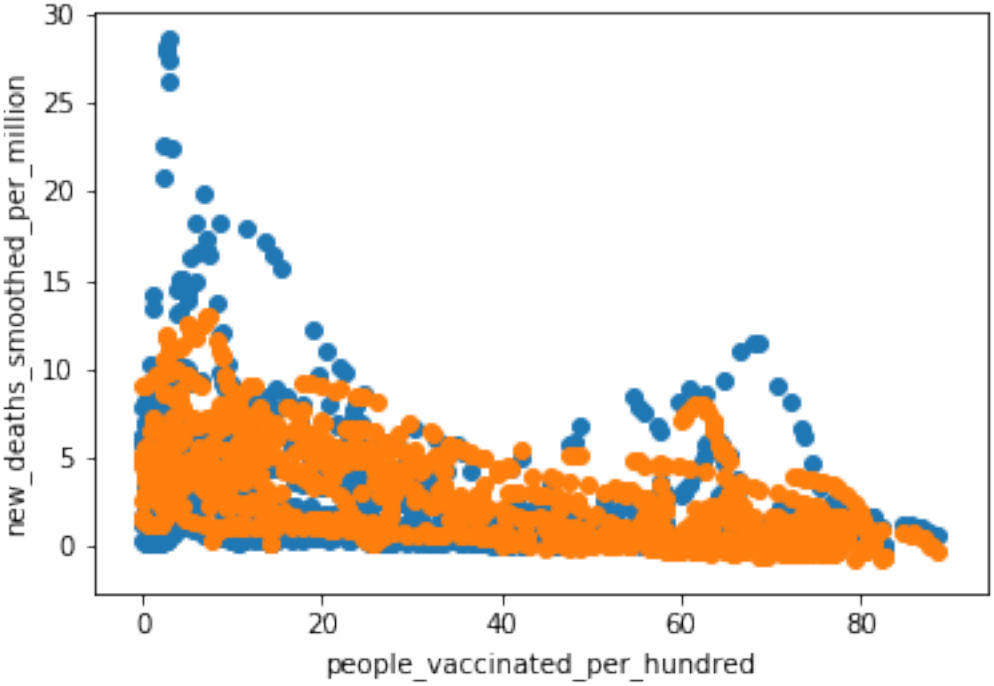

new_deaths_smoothed_per_million
stringency_index
0
5
10
15
20
25
30
20
30
40
50
60
70
80
90

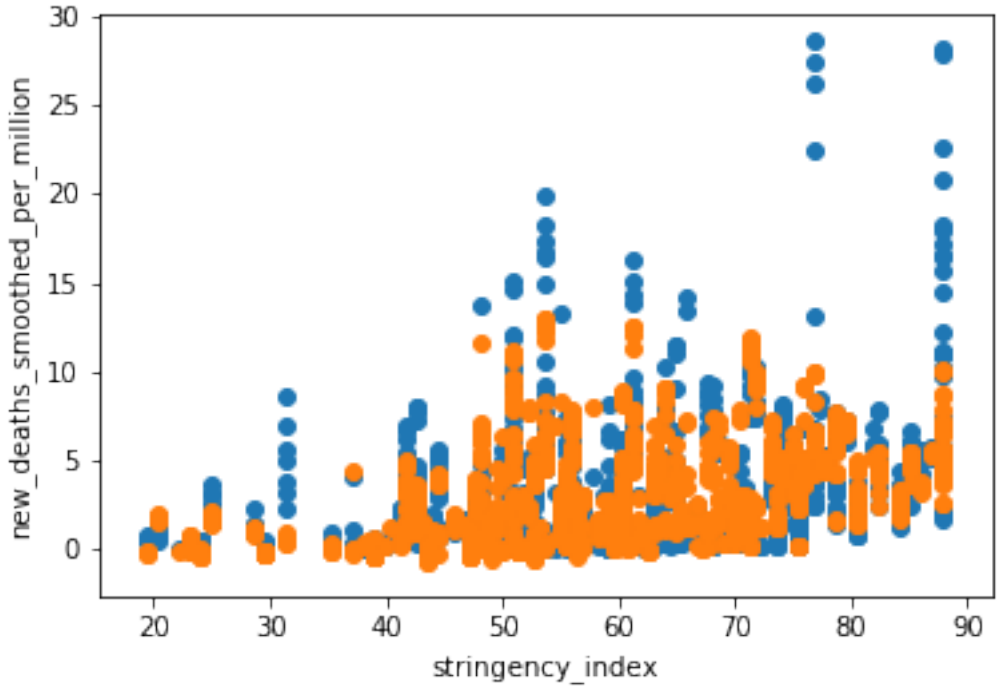


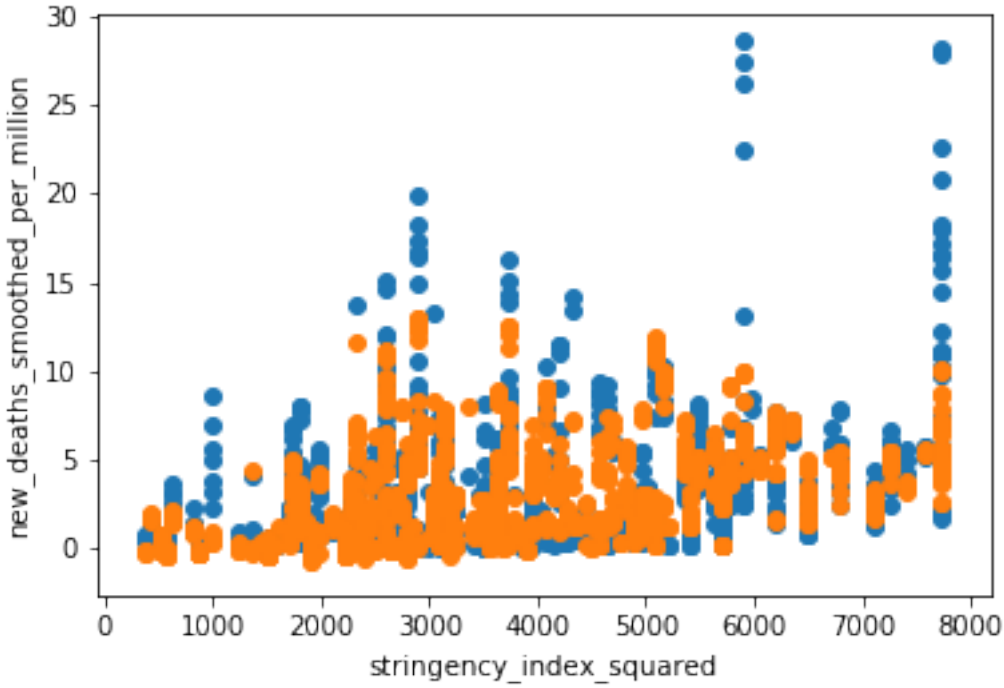
Features: **icu_patients_per_million, people_vaccinated_per_hundred, stringency_index, stringency_index_squared** 1. r2_score is 0.6143834751597625 2. adjusted_r2_score is 0.6092112713988652




new_deaths_smoothed_per_million
icu_patients_per_million
0
5
10
15
20
25
30
0
20
40
60
80
100
120

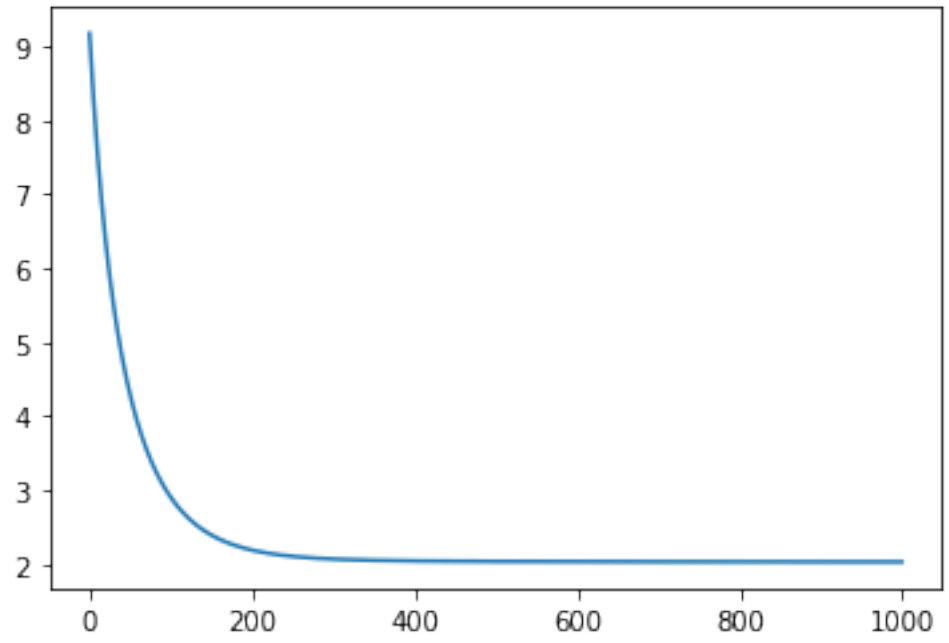

new_deaths_smoothed_per_million
people_vaccinated_per_hundred
0
5
10
15
20
25
30
0
20
40
60
80

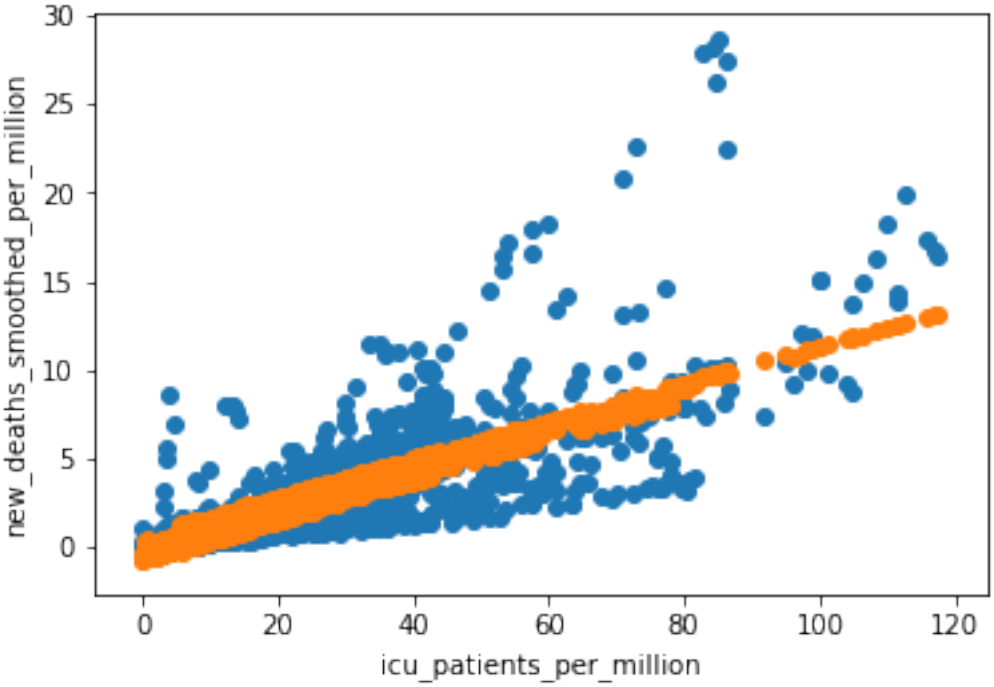


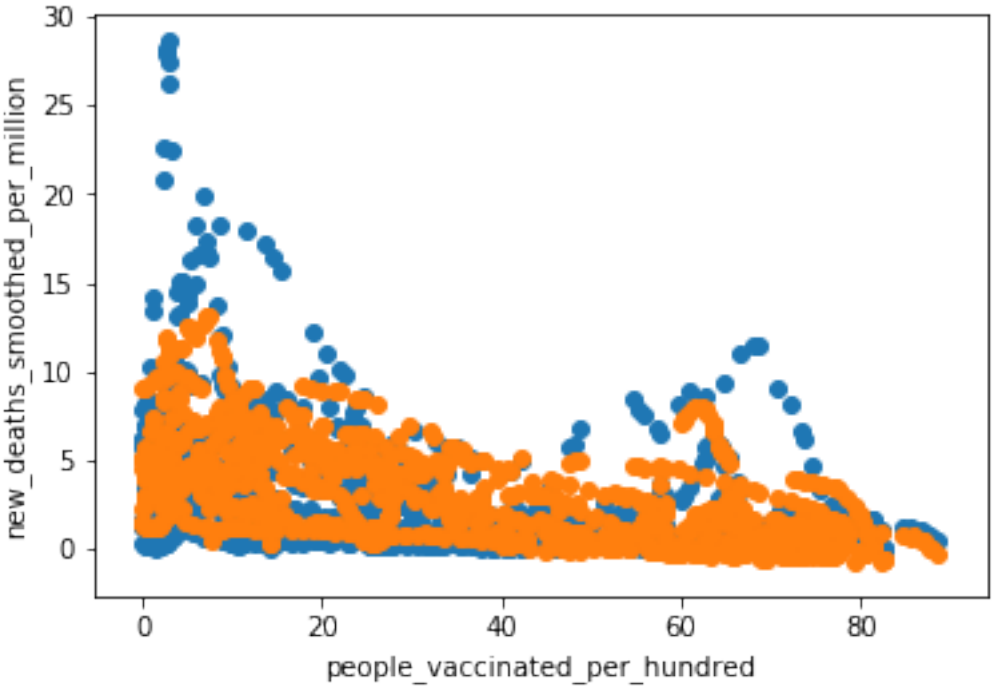


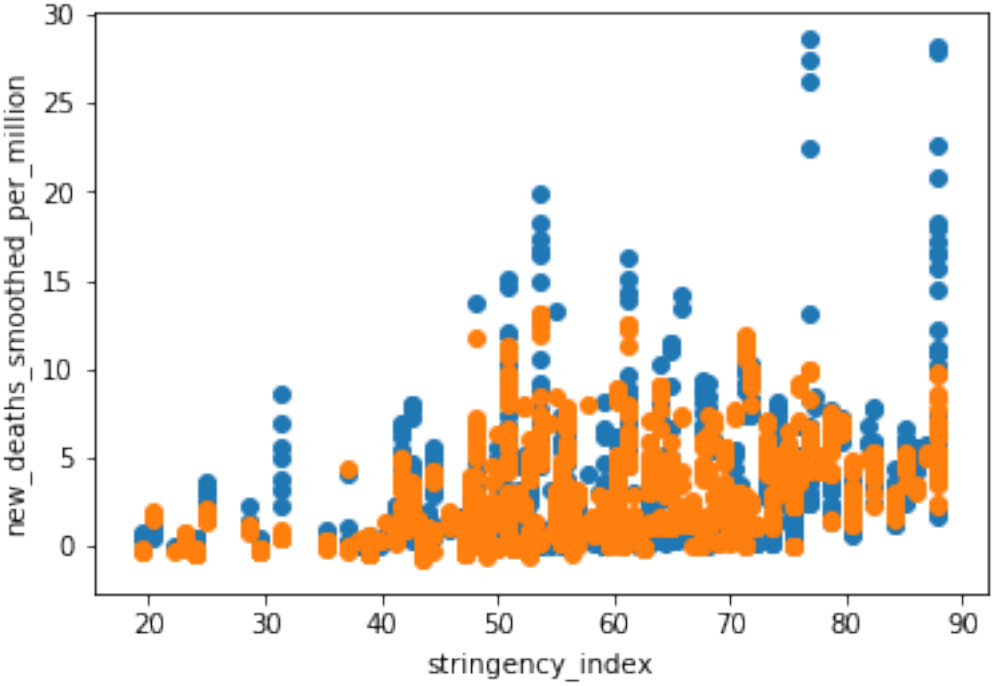
Features: **icu_patients_per_million, people_vaccinated_per_hundred, stringency_index, stringency_index_square_root** 1. r2_score is 0.6096233768431726 2.

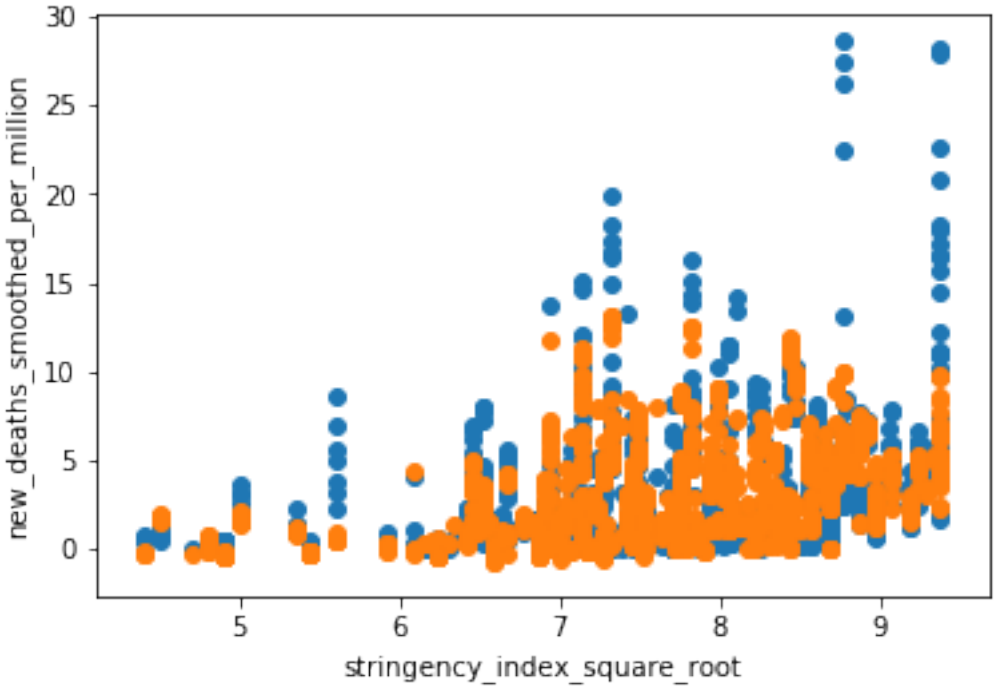
adjusted_r2_score is 0.6043873267561213






new_deaths_smoothed_per_million
people_vaccinated_per_hundred
0
5
10
15
20
25
30
0
20
40
60
80


new_deaths_smoothed_per_million
stringency_index
0
5
10
15
20
25
30
20
30
40
50
60
70
80
90



15 Combining transformed features

From the experimental results obtained, we developed a model by combining the polynomial features which obtained a higher `adjusted_r2_score`. Highest value is indicated in **bold**

	r2	adjusted_r2
With top 2 transformed features	0.626004	0.620987
With top 3 transformed features	0.632497	0.627568
With top 4 transformed features	0.639255	**0.634417**

15.1 Analysis

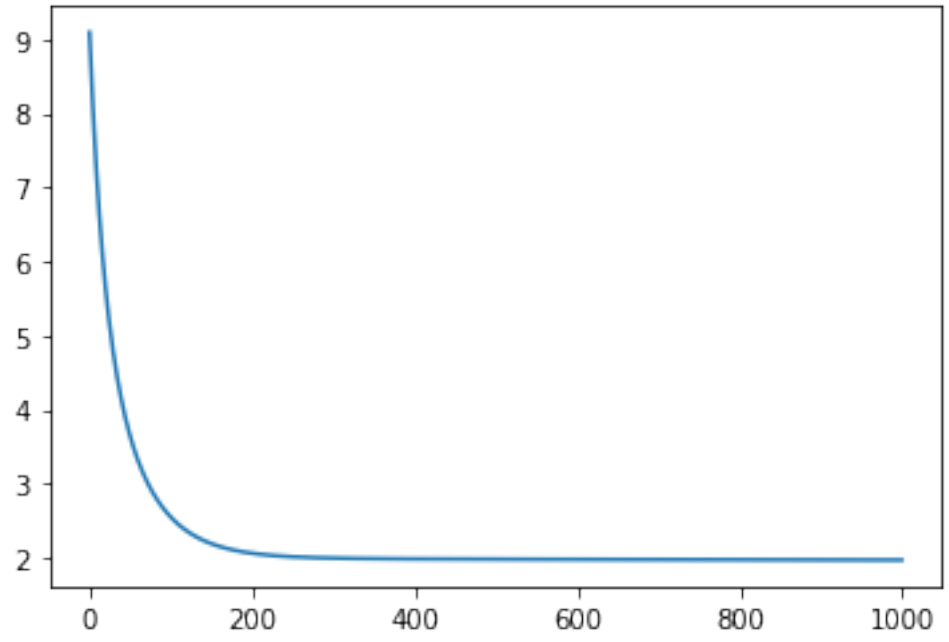
By introducing transformed features to our model, the `adjusted_r2_score` had a **marked improvement of more than 0.04**.

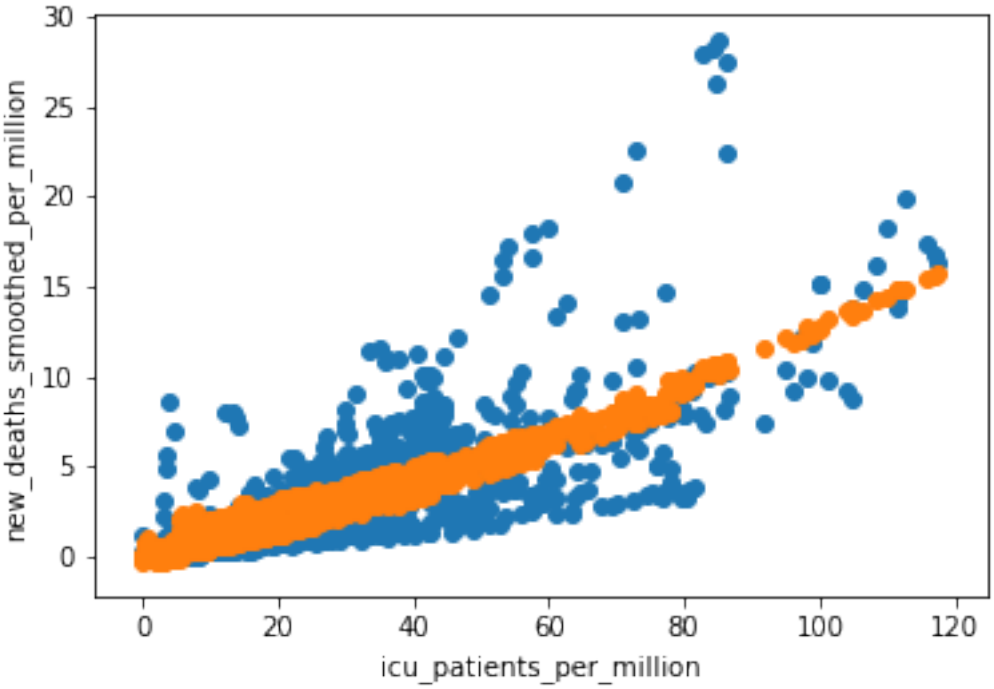
Section **??**

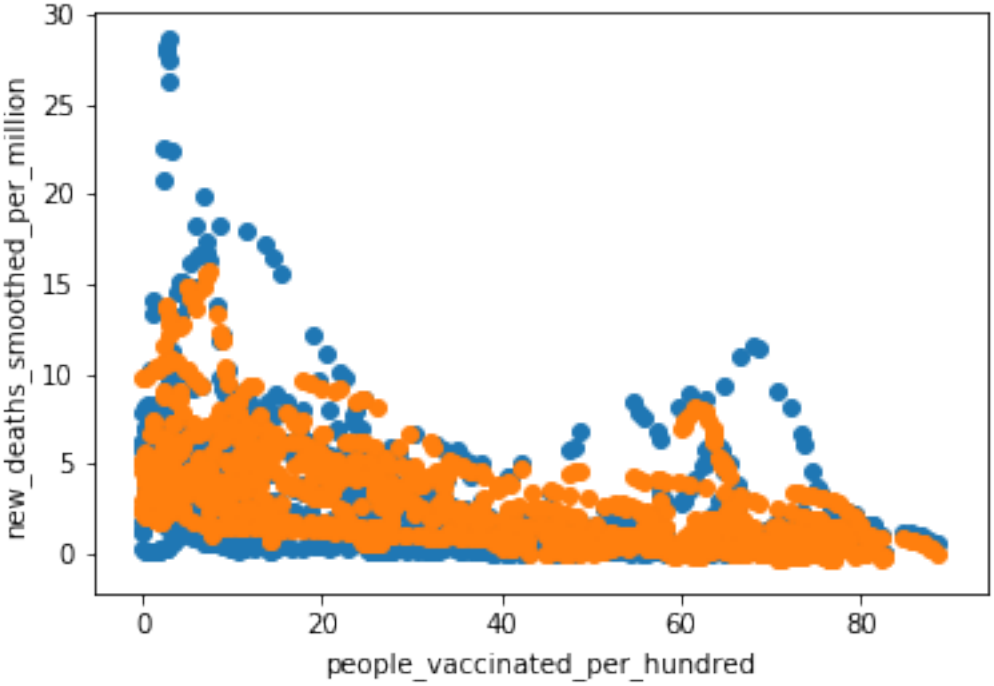
[11]:
```
table = [["", "r2", "adjusted_r2"]]

top_three_with_top_two_transformed = top_three_features +␣
↪["icu_patients_per_million_squared",␣
↪"people_vaccinated_per_hundred_square_root"]
_, r2, adjusted_r2 = lin_reg(df_dropna_one, top_three_with_top_two_transformed)
```

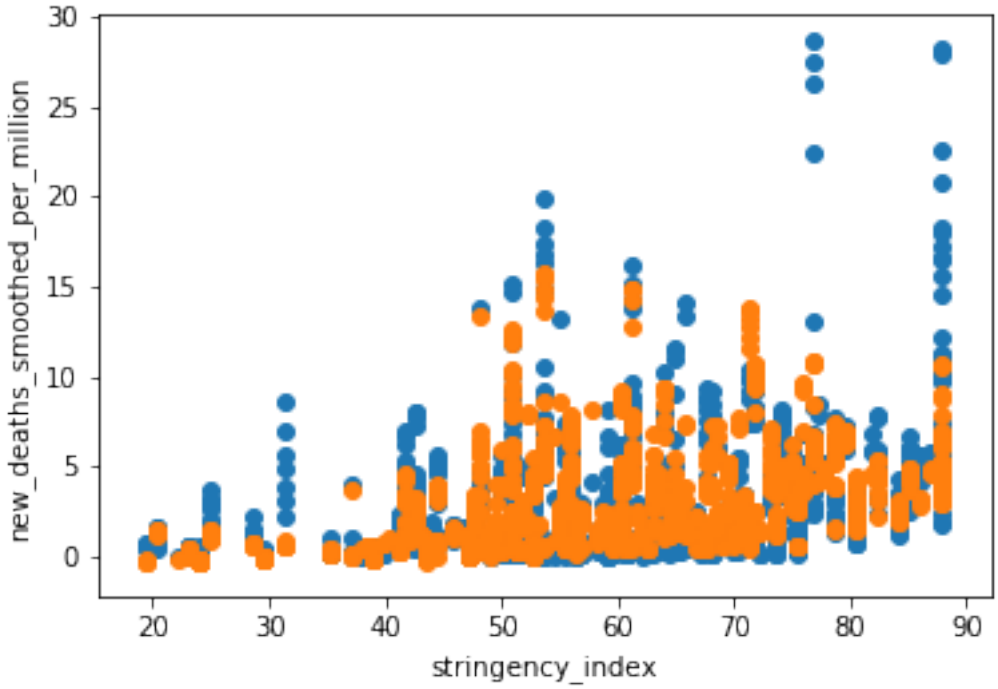
```
table.append([", ".join(top_three_with_top_two_transformed), r2, adjusted_r2])

top_three_with_top_three_transformed = top_three_features +␣
 ↪["icu_patients_per_million_squared",␣
 ↪"people_vaccinated_per_hundred_square_root",␣
 ↪"people_vaccinated_per_hundred_squared"]
_, r2, adjusted_r2 = lin_reg(df_dropna_one,␣
 ↪top_three_with_top_three_transformed)
table.append([", ".join(top_three_with_top_three_transformed), r2, adjusted_r2])

top_three_with_top_four_transformed = top_three_features +␣
 ↪["icu_patients_per_million_squared",␣
 ↪"people_vaccinated_per_hundred_square_root",␣
 ↪"people_vaccinated_per_hundred_squared", "stringency_index_squared"]
beta, r2, adjusted_r2 = lin_reg(df_dropna_one,␣
 ↪top_three_with_top_four_transformed)
table.append([", ".join(top_three_with_top_four_transformed), r2, adjusted_r2])
```

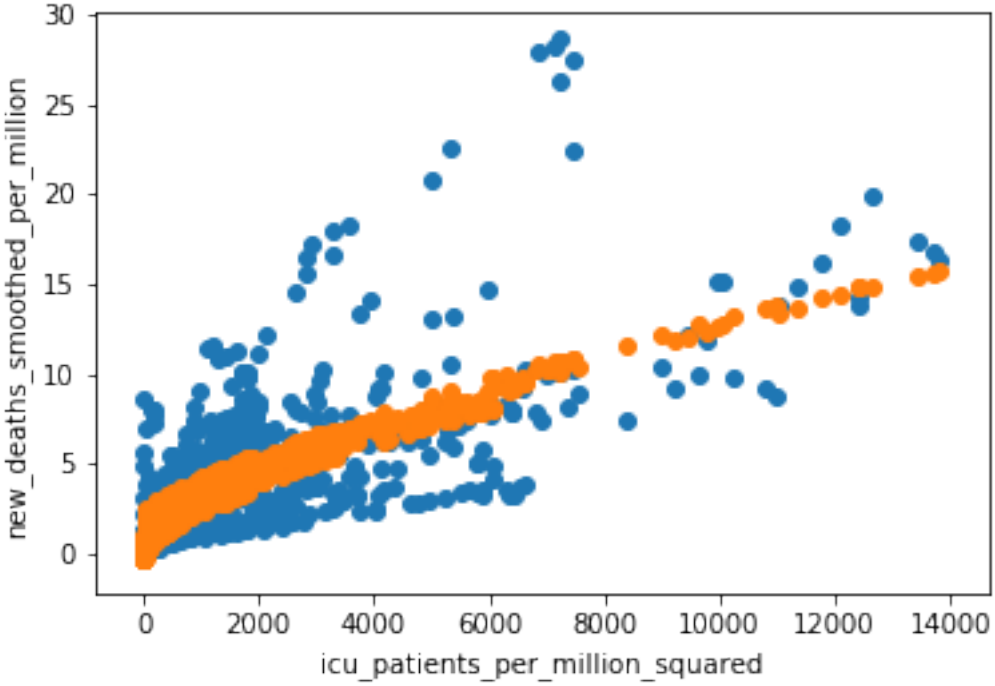
Features: **icu_patients_per_million, people_vaccinated_per_hundred, stringency_index, icu_patients_per_million_squared, people_vaccinated_per_hundred_square_root** 1. r2_score is 0.6260035458604825 2. adjusted_r2_score is 0.6209871999778362

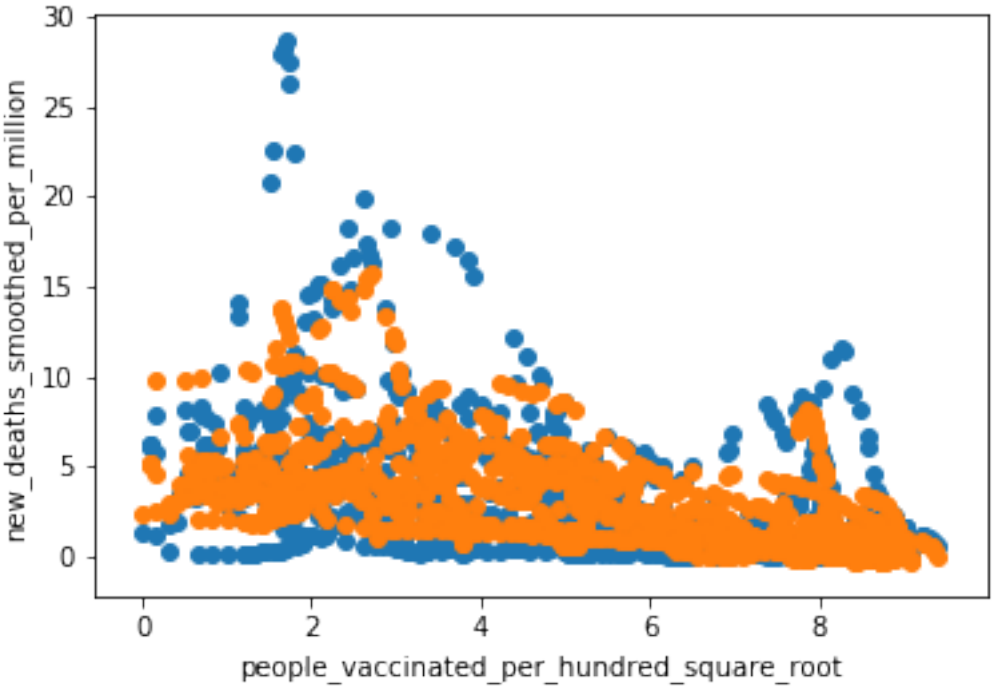



new_deaths_smoothed_per_million
icu_patients_per_million
0
5
10
15
20
25
30
0
20
40
60
80
100
120

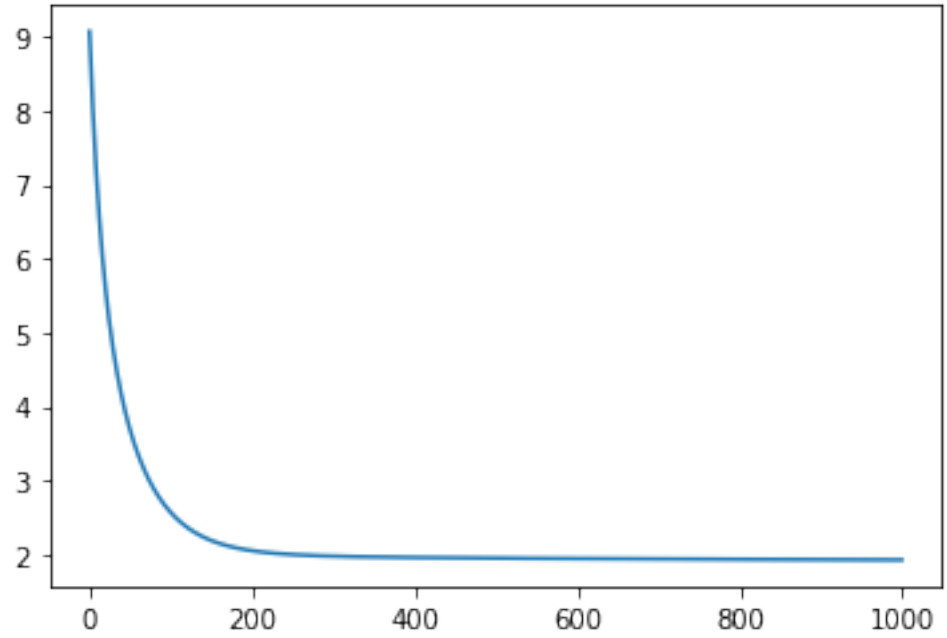

new_deaths_smoothed_per_million
people_vaccinated_per_hundred
0
5
10
15
20
25
30
0
20
40
60
80

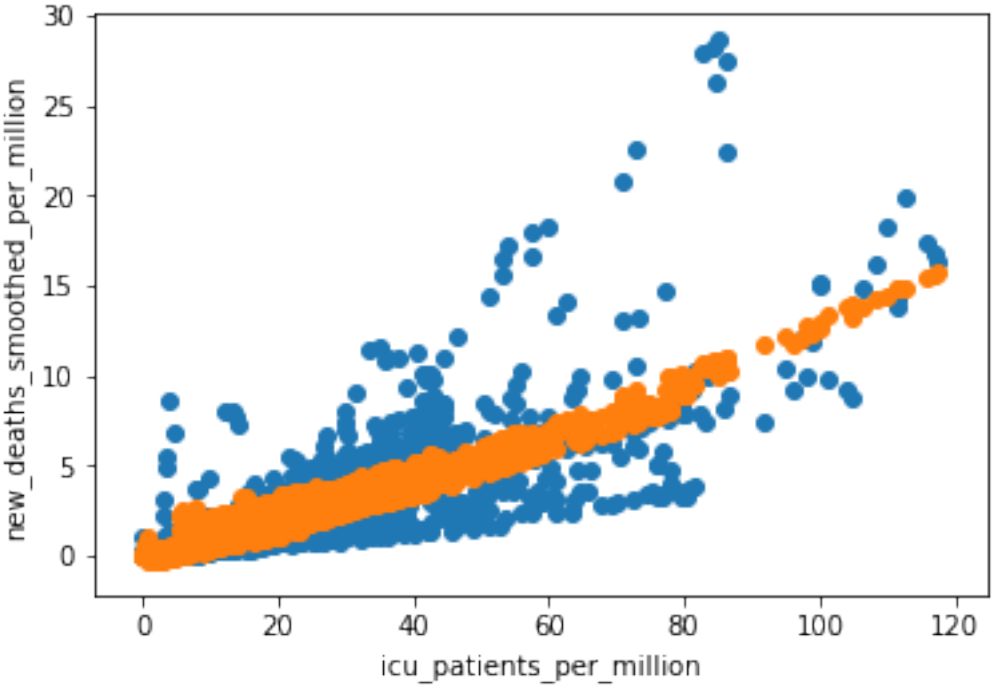

new_deaths_smoothed_per_million
stringency_index
0
5
10
15
20
25
30
20
30
40
50
60
70
80
90

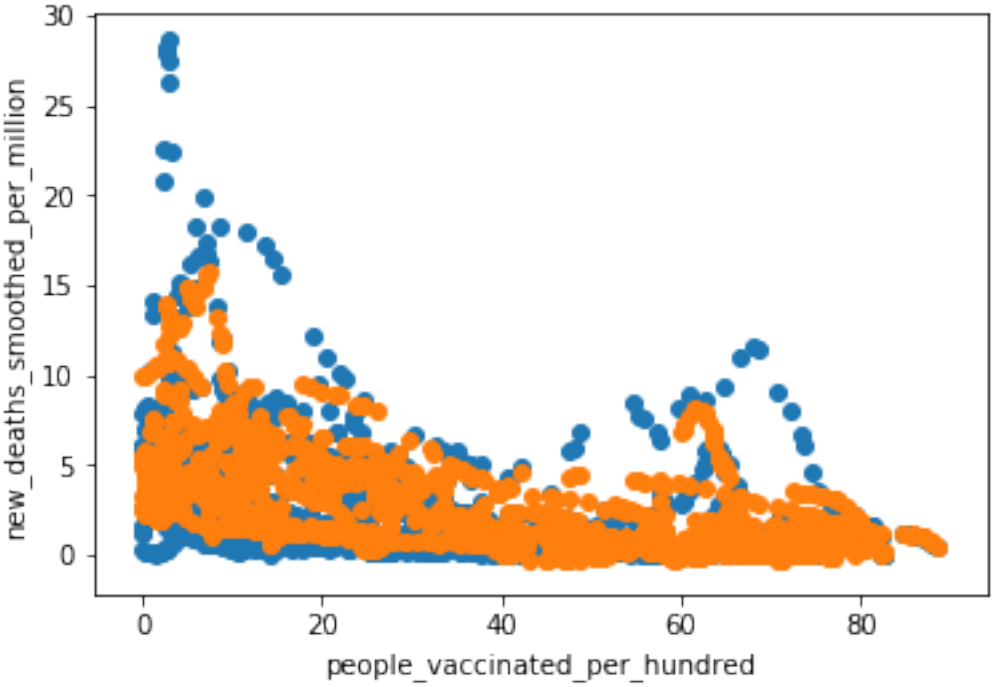

new_deaths_smoothed_per_million
icu_patients_per_million_squared
0
5
10
15
20
25
30
0
2000
4000
6000
8000
10000
12000
14000

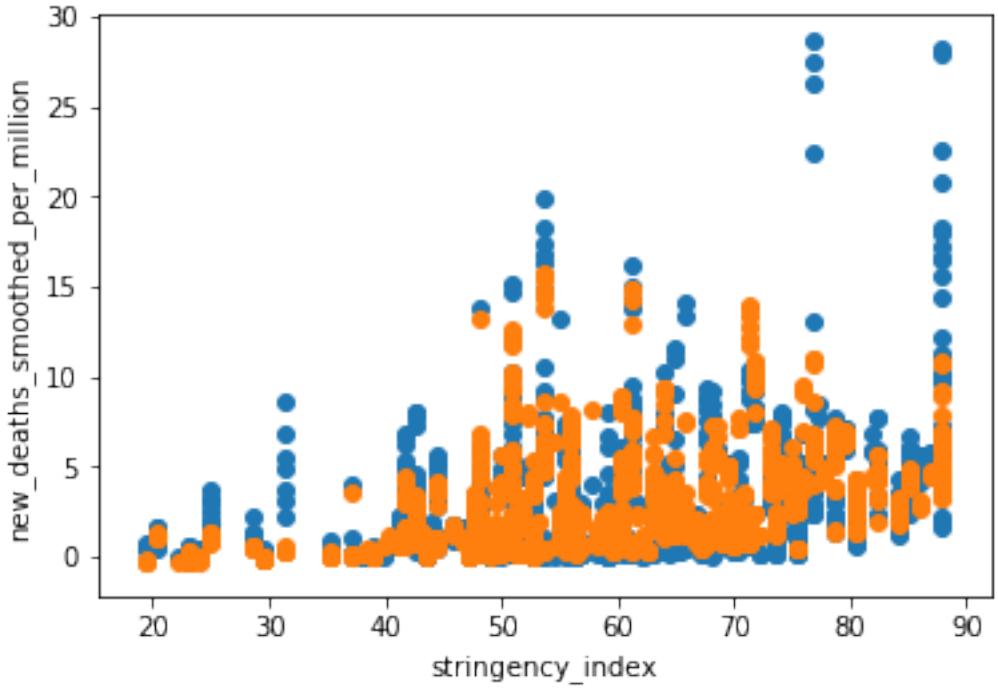


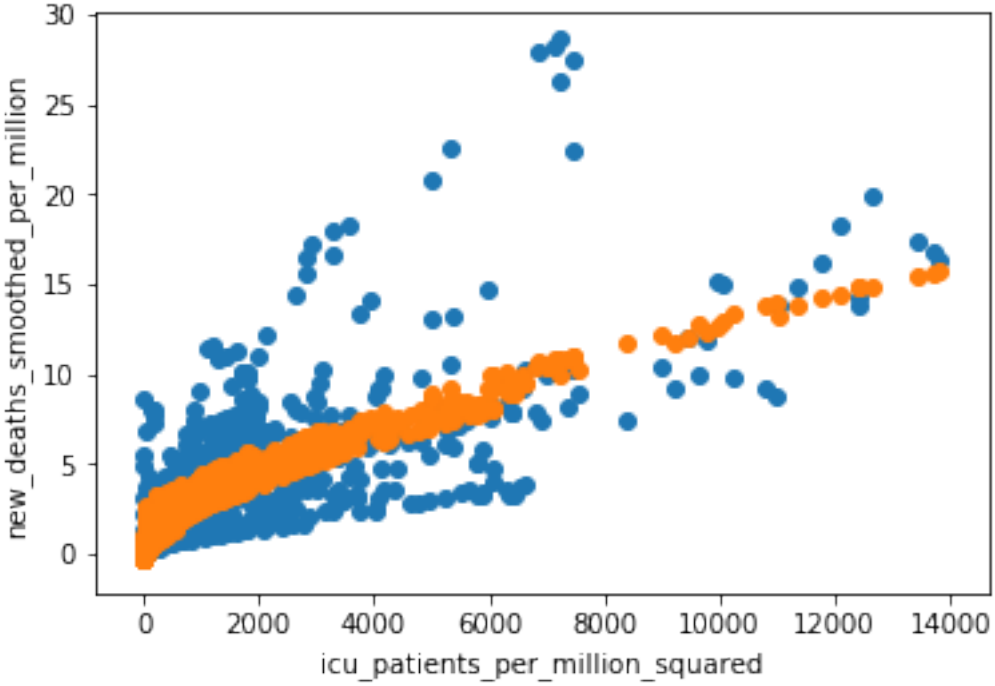
Features: **icu_patients_per_million, people_vaccinated_per_hundred, stringency_index, icu_patients_per_million_squared, people_vaccinated_per_hundred_square_root, people_vaccinated_per_hundred_squared**
1. r2_score is 0.6324973765179638 2. adjusted_r2_score is 0.6275681311955519

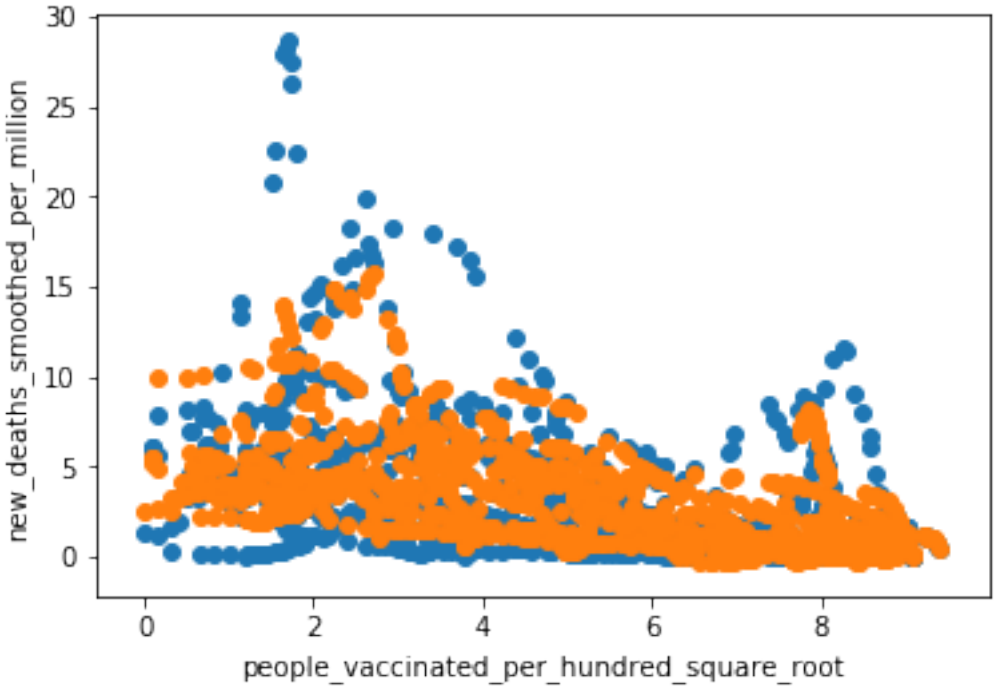


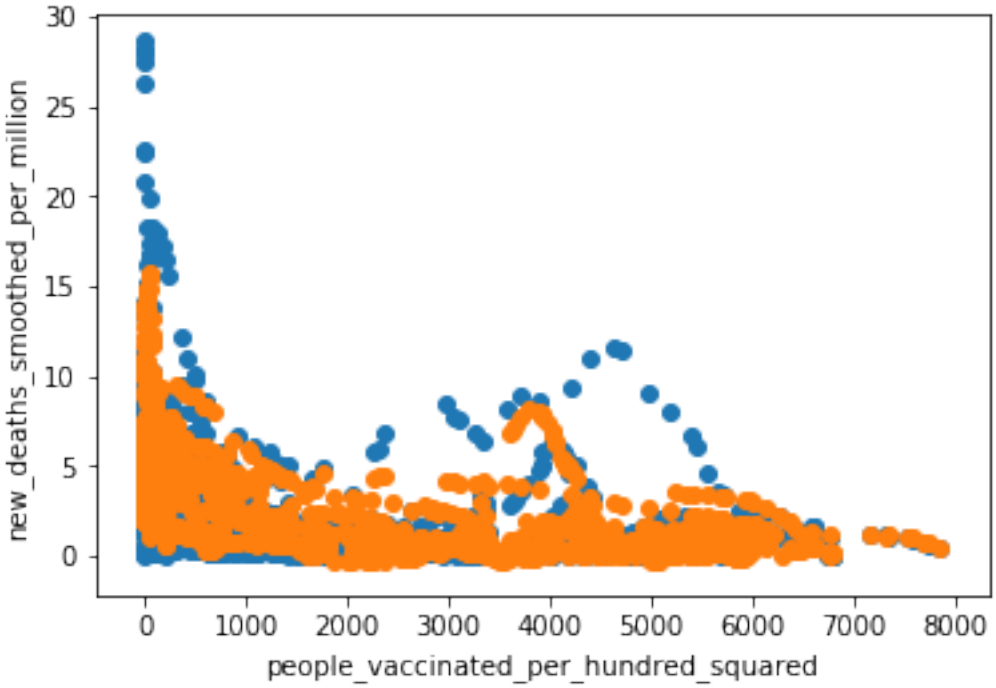

new_deaths_smoothed_per_million
icu_patients_per_million
0
5
10
15
20
25
30
0
20
40
60
80
100
120


new_deaths_smoothed_per_million
people_vaccinated_per_hundred
0
5
10
15
20
25
30
0
20
40
60
80


new_deaths_smoothed_per_million
stringency_index
0
5
10
15
20
25
30
20
30
40
50
60
70
80
90

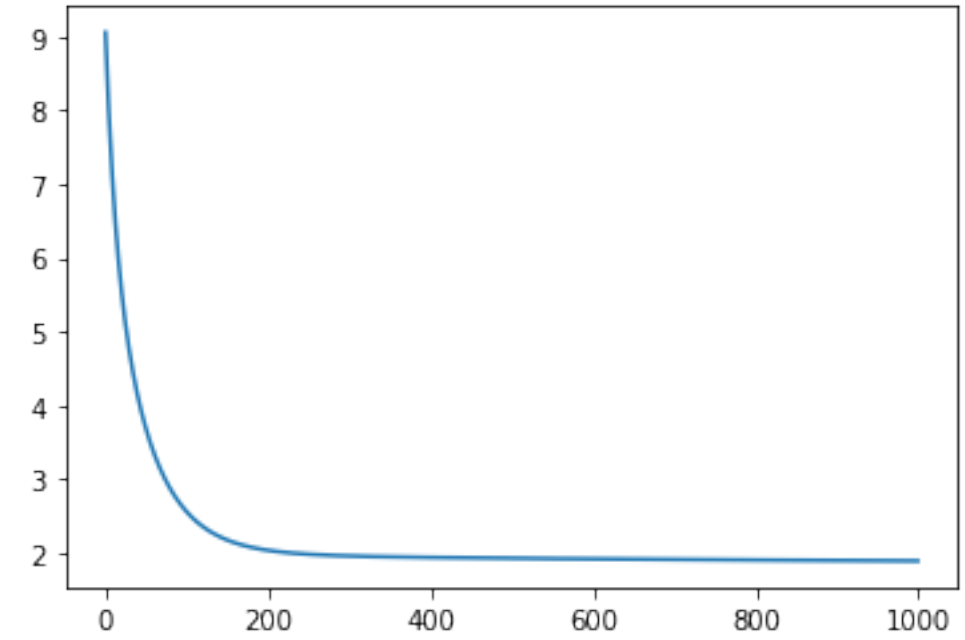

new_deaths_smoothed_per_million
icu_patients_per_million_squared
0
5
10
15
20
25
30
0
2000
4000
6000
8000
10000
12000
14000

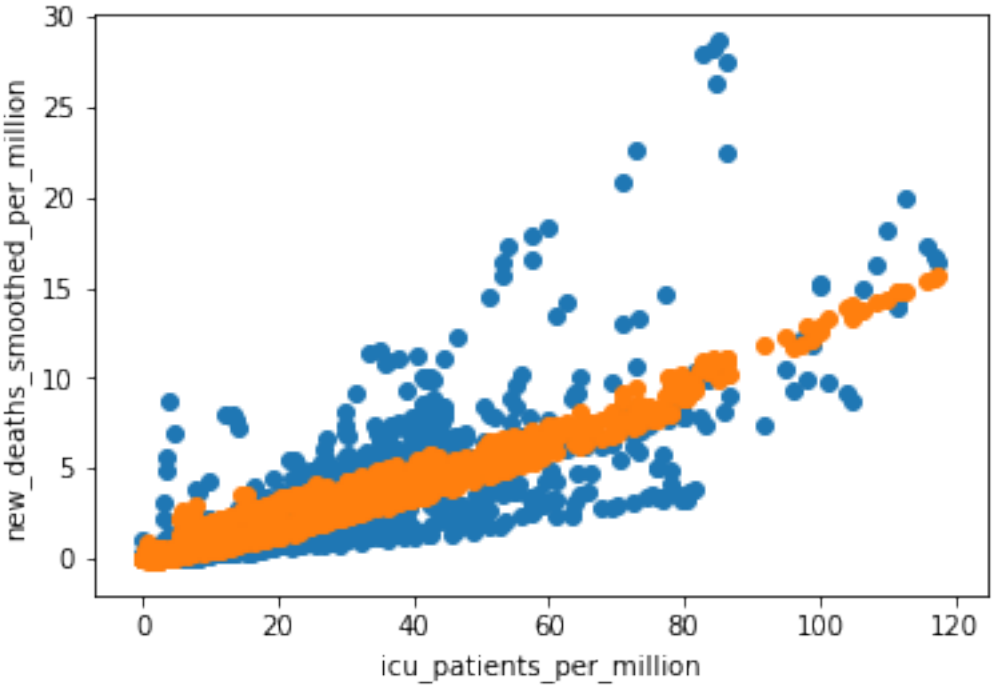


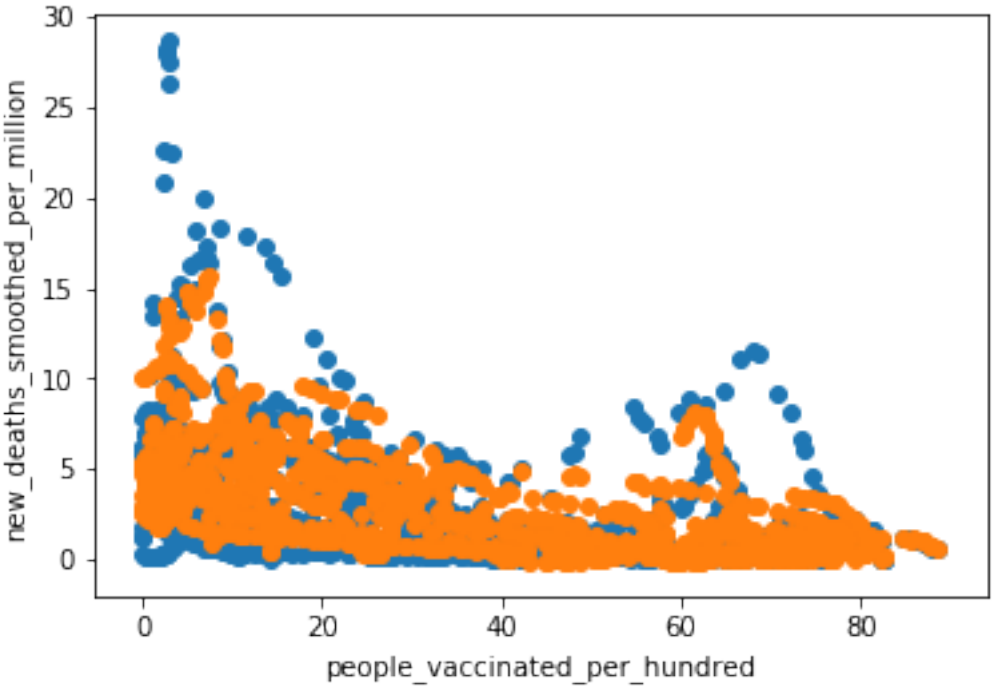


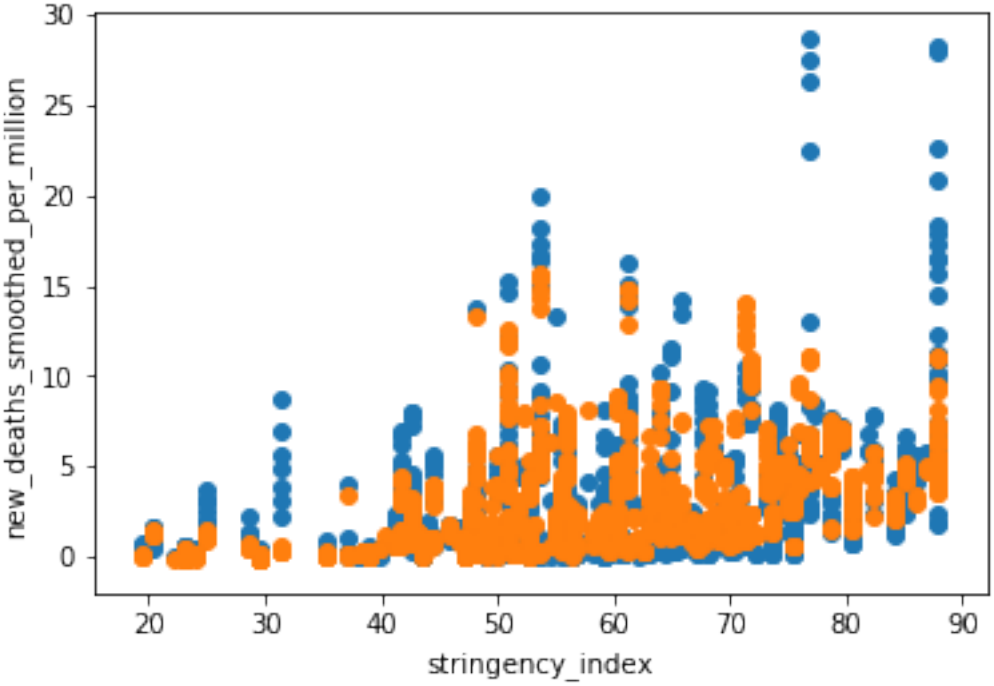
Features: **icu_patients_per_million, people_vaccinated_per_hundred, stringency_index, icu_patients_per_million_squared, peo-**

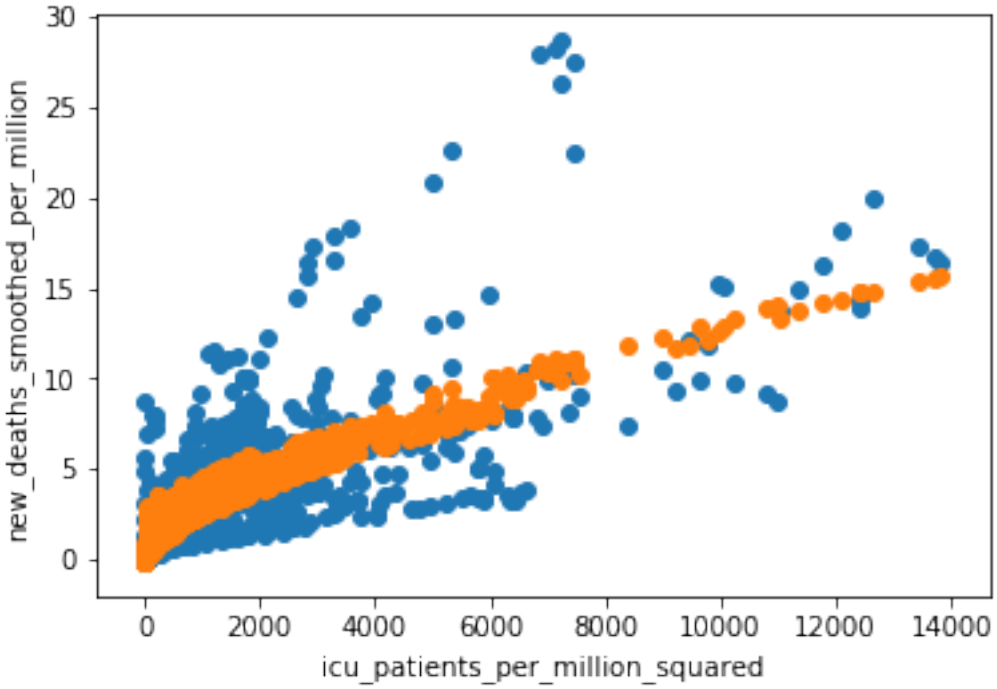
ple__vaccinated__per__hundred__square__root, people__vaccinated__per__hundred__squared, stringency__index__squared 1. r2_score is 0.6392551694095301 2. adjusted_r2_score is 0.634416565124412

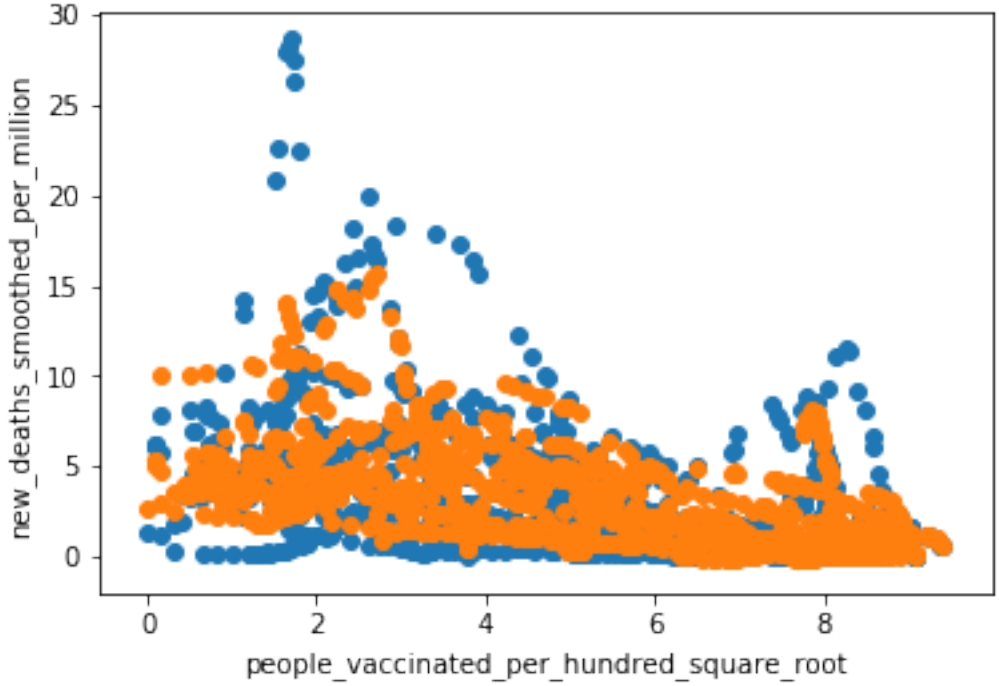


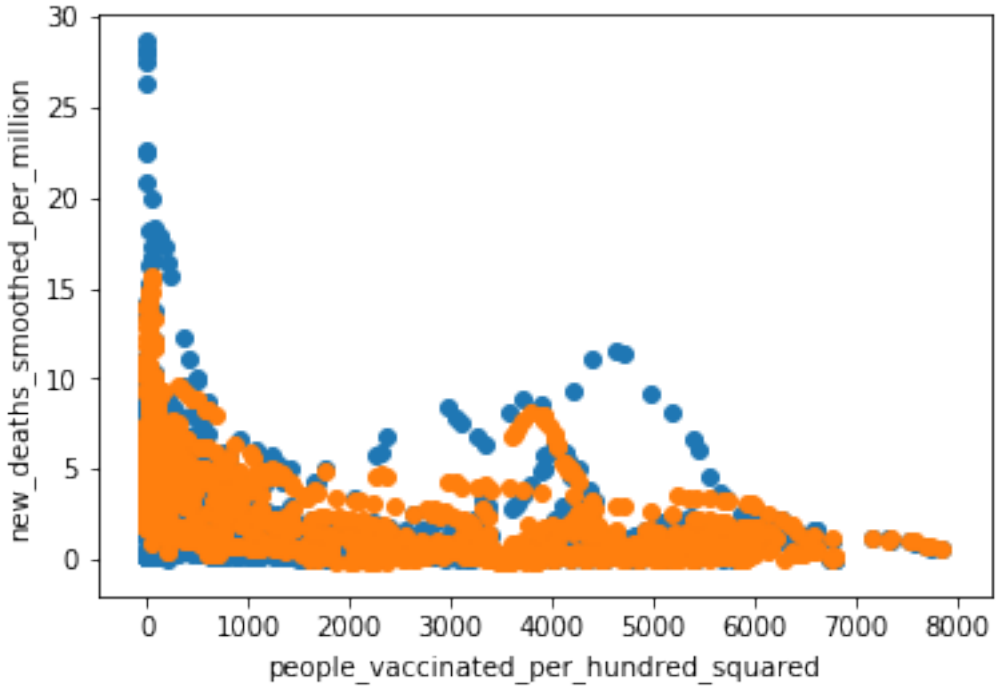


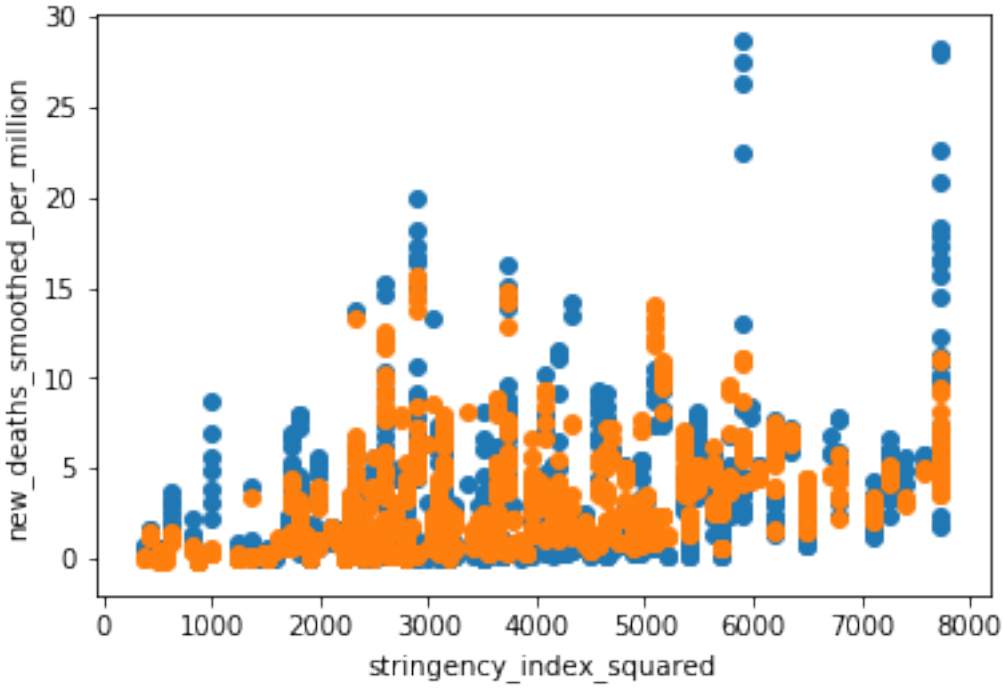

new_deaths_smoothed_per_million
people_vaccinated_per_hundred
0
5
10
15
20
25
30
0
20
40
60
80


new_deaths_smoothed_per_million
stringency_index
0
5
10
15
20
25
30
20
30
40
50
60
70
80
90


new_deaths_smoothed_per_million
icu_patients_per_million_squared
0
5
10
15
20
25
30
0
2000
4000
6000
8000
10000
12000
14000


new_deaths_smoothed_per_million
people_vaccinated_per_hundred_square_root
0
5
10
15
20
25
30
0
2
4
6
8





16 Conclusion

16.1 Final model

Our team attempted to predict daily deaths due to COVID-19 based on historical data. Independent of time and location, our final model is able to achieve a relatively high linear correlation of 0.634417. This model is ~0.04 higher than a simple linear correlation model (Highest value obtained was 0.592940113924265) and it consists of the following independent variables: 1. `icu_patients_per_million` 2. `people_vaccinated_per_hundred` 3. `stringency_index` 4. `icu_patients_per_million_squared` 5. `people_vaccinated_per_hundred_square_root` 6. `people_vaccinated_per_hundred_squared` 7. `stringency_index_squared`

16.2 Analysis

These findings show strong justifications that vaccinations may indeed help to curb death rates due to COVID-19. Additionally, it may also prove that implementing safe management measures in response to COVID-19 helps to curb death rates due to COVID-19. Most trivially, our results show that having a higher number of ICU patients leads to a higher death rate, which likely tells us that patients within the ICU are more likely to die as a result of COVID-19.

All that being said, we are aware that correlation does not equate to causation. Hence, further experiments need to be conducted to justify the hypotheses laid out above.

However, regardless of whether our hypotheses are true, we have developed a sufficiently accurate linear regression model to predict death rates due to COVID-19, at least until 10th November 2021.

16.3 Future work

To further test the strength of our model, our model should be tested against future datasets. If the linear regression remains high of > 0.5, it shows that our model is indeed independent of time and is not overfitting the data.

Additionally, our model has the potential to be further improved by introducing non-linear models, introducing other features that we were unable to test due to dataset and time limitations or by further transforming our current features.

Finally, our model can be improved by choosing a data set with a lower proportion of rows with missing values.

Section **??**

17 Appendix

18 Replacing missing values via forward propagation

18.1 Reproduction rate

1. r2_score is -6.392753582384891e-05
2. adjusted_r2_score is -0.0003204462675572284 (DECREASED from 0.03740407903444909)

18.2 People vaccinated per hundred

1. r2_score is 0.004205067263488016
2. adjusted_r2_score is 0.003949643538885161 (DECREASED from 0.21997546464365736)

18.3 People fully vaccinated per hundred

1. r2_score is 0.006019635166130466
2. adjusted_r2_score is 0.00576467688243032 (DECREASED from 0.1730855138923163)

18.4 ICU patients per million

1. r2_score is 0.006595121327647613
2. adjusted_r2_score is 0.006340310657490966 (DECREASED from 0.592940113924265)

18.5 Stringency index

1. r2_score is 0.01671749289046609
2. adjusted_r2_score is 0.016465278632242786 (DECREASED from 0.1103046673899103)

18.6 Extreme poverty

1. r2_score is 0.019816891055664754
2. adjusted_r2_score is 0.019565471800305323 (INCREASED from 0.012765814940187359)

18.7 Human development index

1. r2_score is 0.039726039188433315
2. adjusted_r2_score is 0.03947972667604127 (DECREASED from 0.06123728635004533)

18.8 Median age

1. r2_score is 0.046472998003899924
2. adjusted_r2_score is 0.04622841610219253 (INCREASED from -0.002810942157140728)

18.9 Analysis

As shown, the univariate linear regression against all but two variables experienced a drop in `adjusted_r2_score`

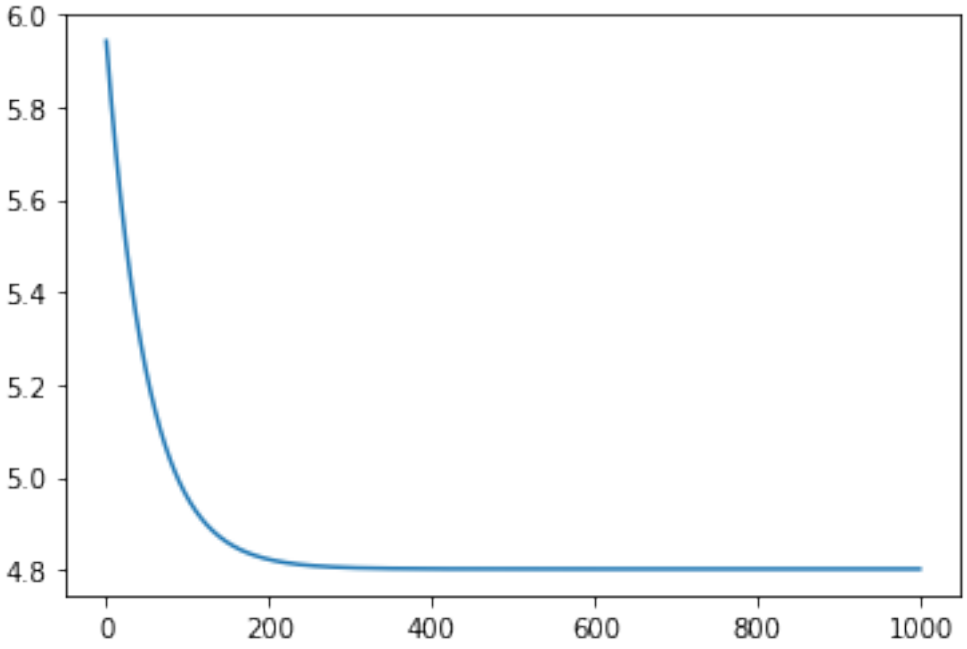
Section **??**

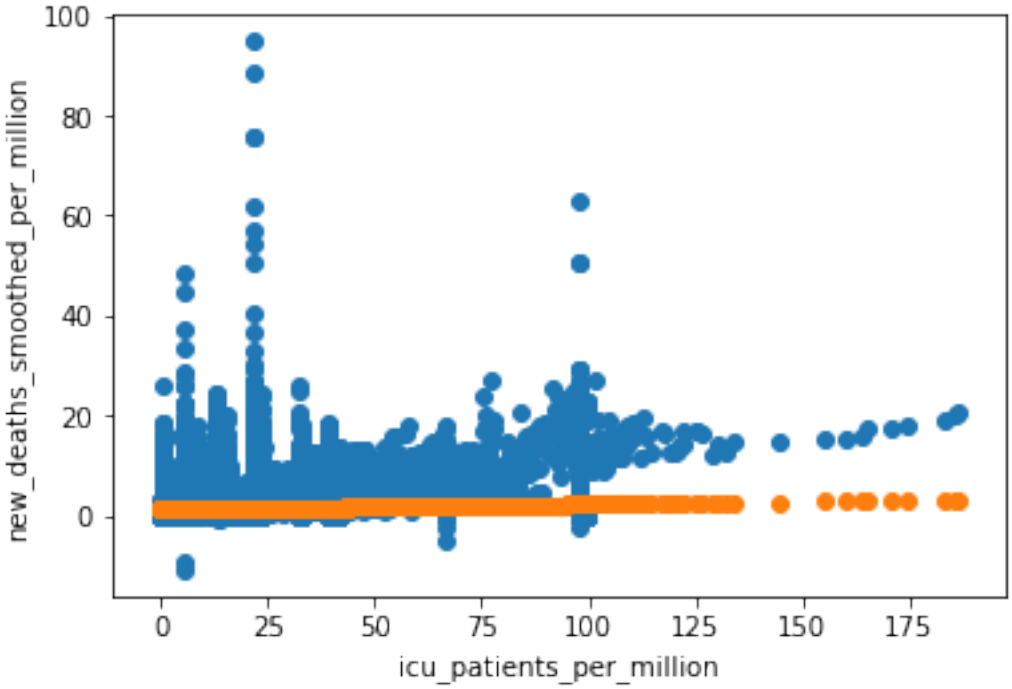
[12]:
```
useful_cols = ["icu_patients_per_million", "new_cases_smoothed_per_million",␣
 ↪"stringency_index", "median_age", "extreme_poverty",␣
 ↪"people_fully_vaccinated_per_hundred", "people_vaccinated_per_hundred",␣
 ↪"reproduction_rate", "human_development_index",␣
 ↪"new_deaths_smoothed_per_million"]

df_useful = df[useful_cols]

# Forward fill
df_fillna = df_useful.fillna(method="ffill").dropna()

print(f"Number of data points before dropping missing values is {df_useful.
 ↪shape[0]} while the number of data points after dropping missing values is␣
 ↪{df_fillna.shape[0]}")
```

```
independent_var = ["icu_patients_per_million",␣
 ↪"new_cases_smoothed_per_million", "stringency_index", "median_age",␣
 ↪"extreme_poverty", "people_fully_vaccinated_per_hundred",␣
 ↪"people_vaccinated_per_hundred", "reproduction_rate",␣
 ↪"human_development_index"]
for feature in independent_var:
    lin_reg(df_fillna, [feature], random_state=100)
```

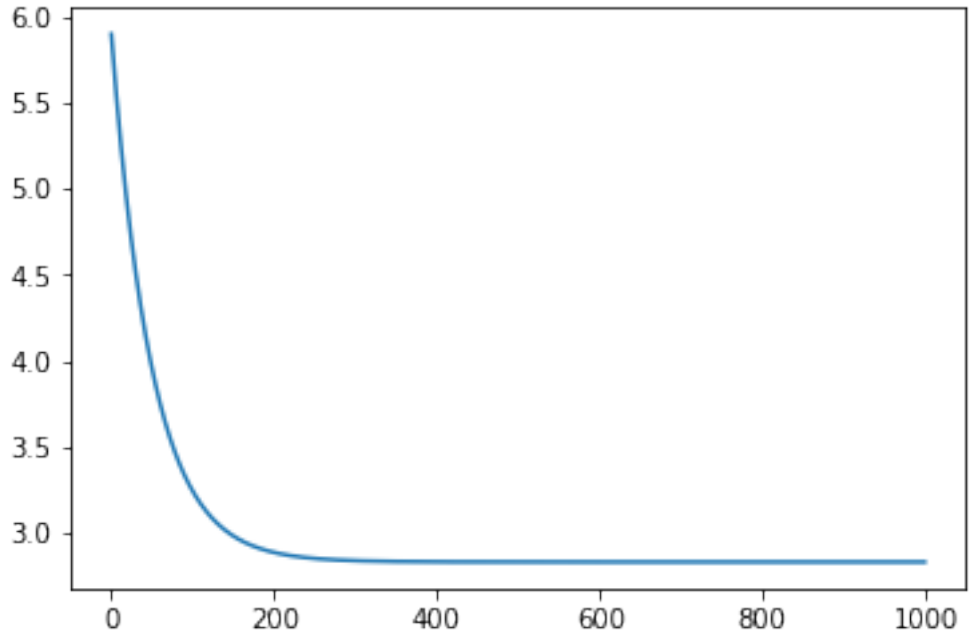
```
Number of data points before dropping missing values is 132016 while the number
of data points after dropping missing values is 129991
```

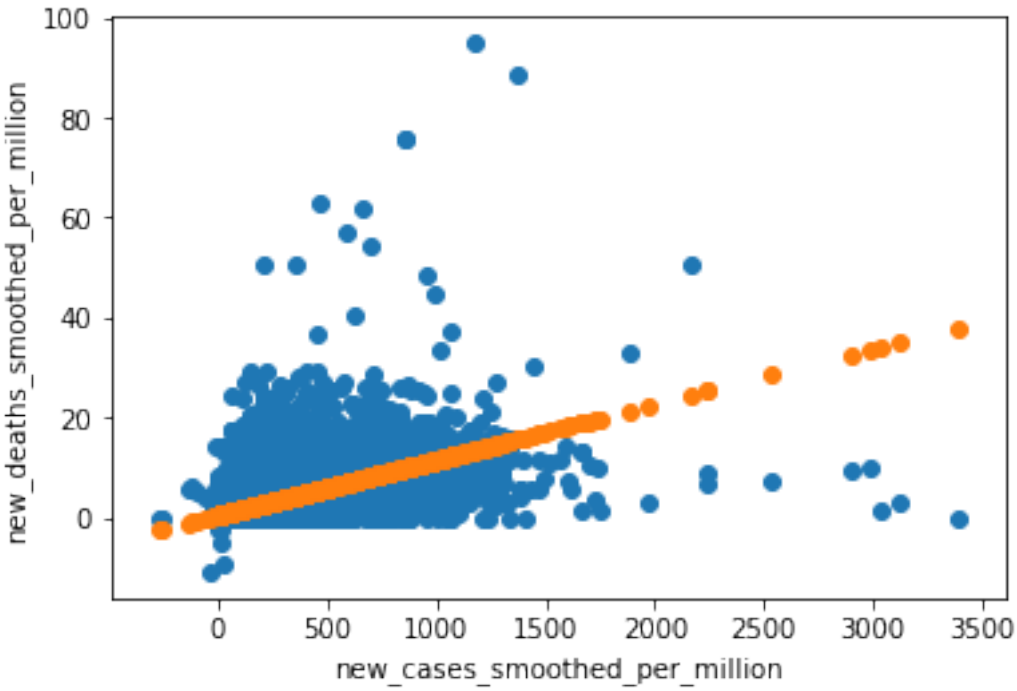
Features: **icu_patients_per_million** 1. r2_score is 0.006595121327647613 2. adjusted_r2_score is 0.006340310657490966



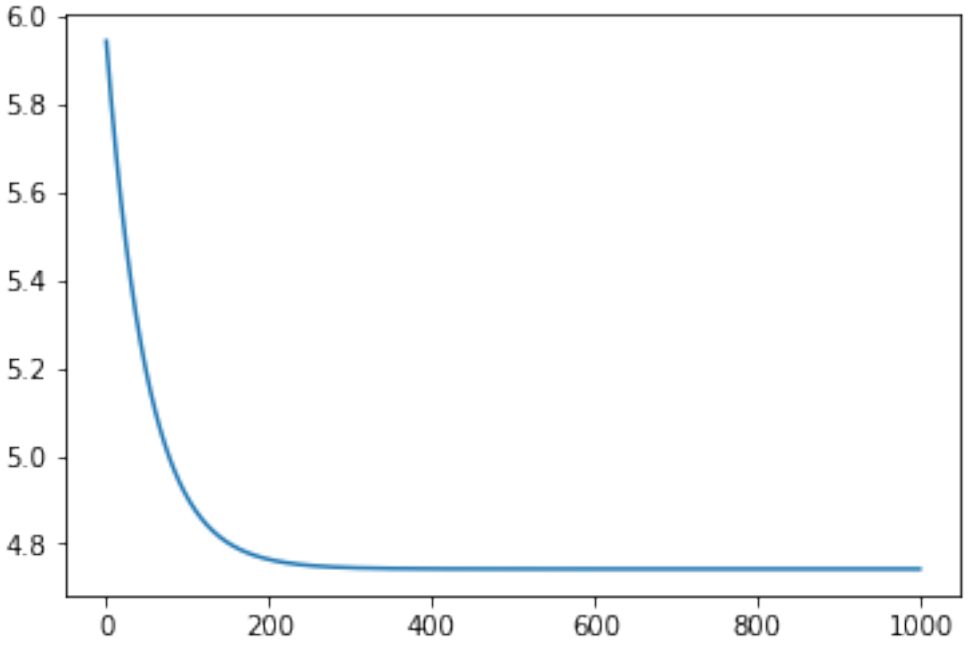


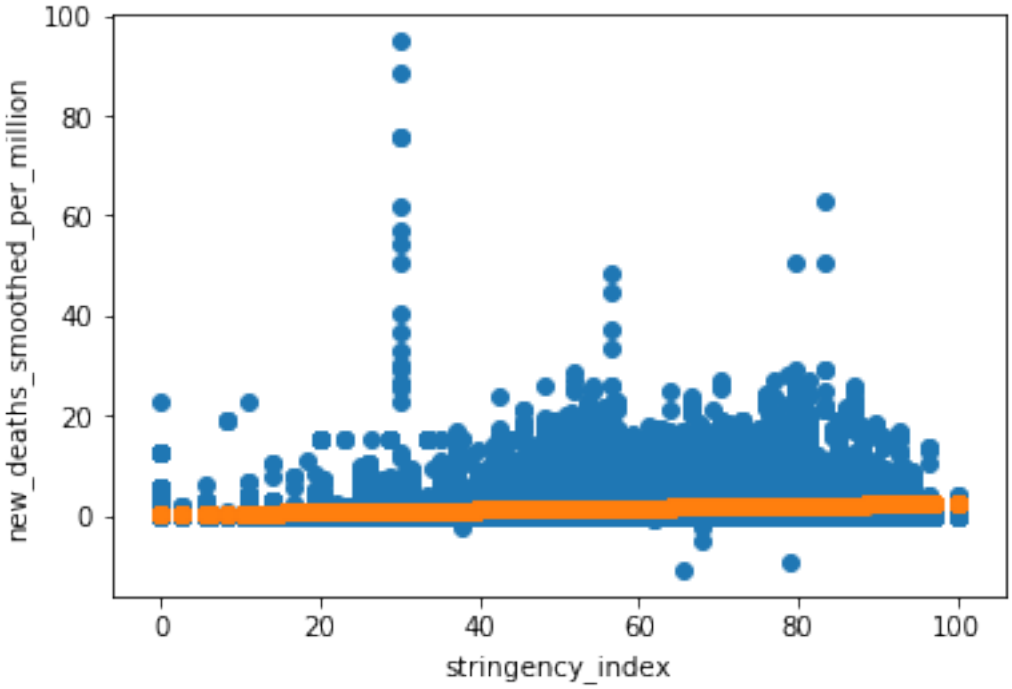
Features: **new_cases_smoothed_per_million** 1. r2_score is 0.4057454742477602 2. adjusted_r2_score is 0.4055930465748129



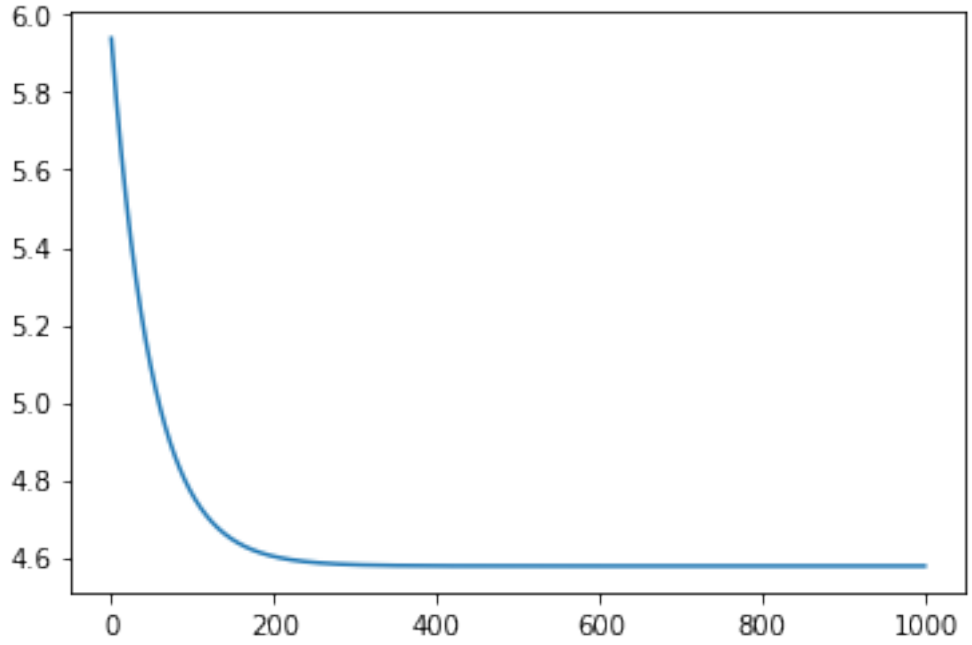


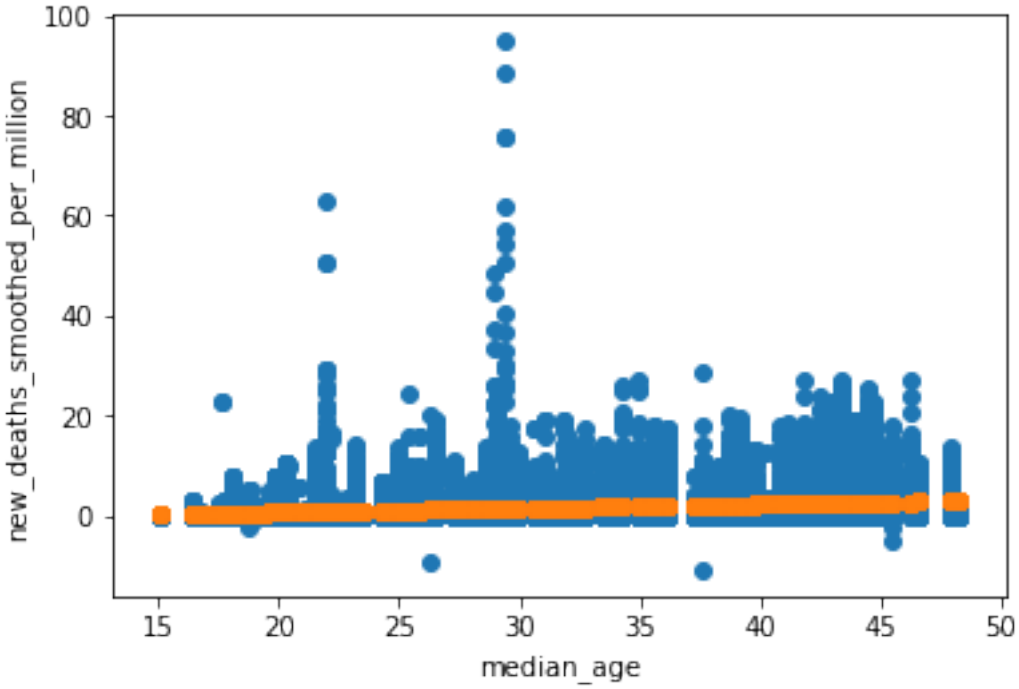
Features: **stringency_index** 1. r2_score is 0.01671749289046609 2. adjusted_r2_score is 0.016465278632242786



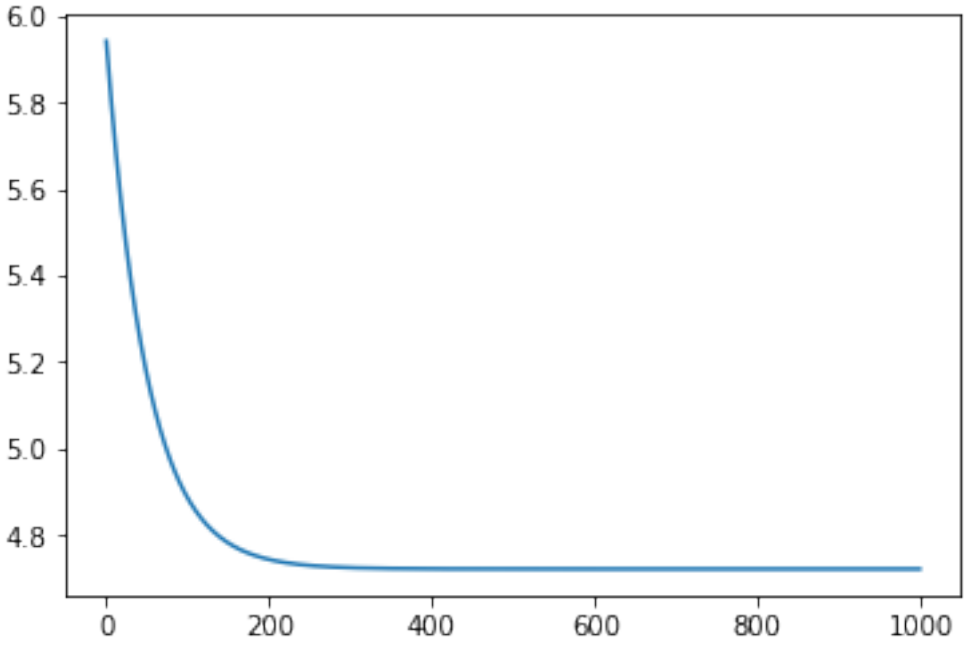


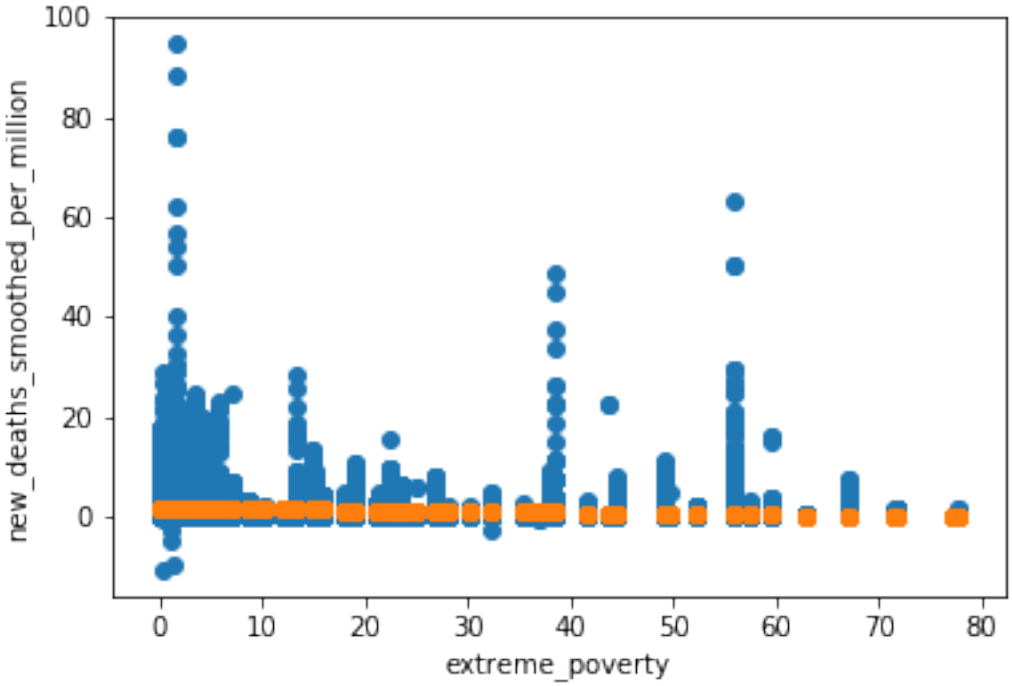
Features: **median_age** 1. r2_score is 0.046472998003899924 2. adjusted_r2_score is 0.04622841610219253



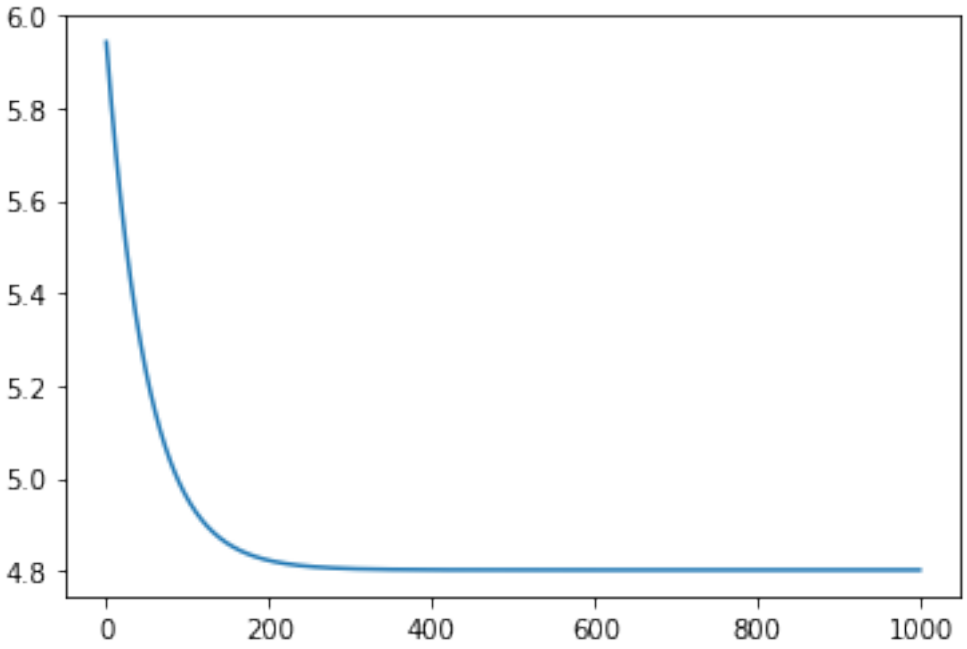


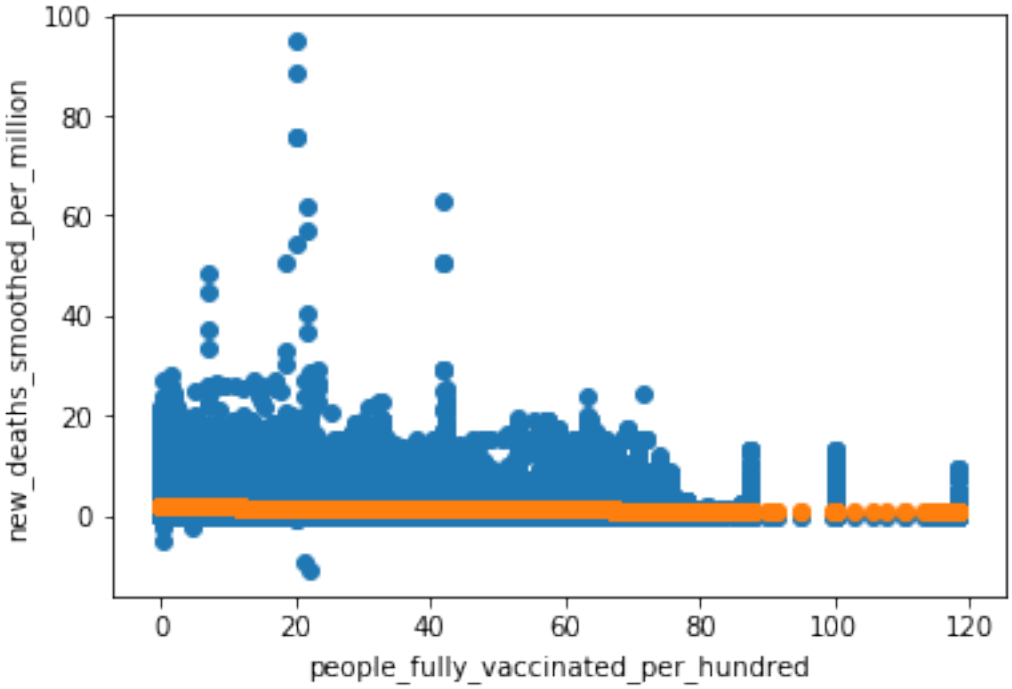
Features: **extreme_poverty** 1. r2_score is 0.019816891055664754 2. adjusted_r2_score is 0.019565471800305323



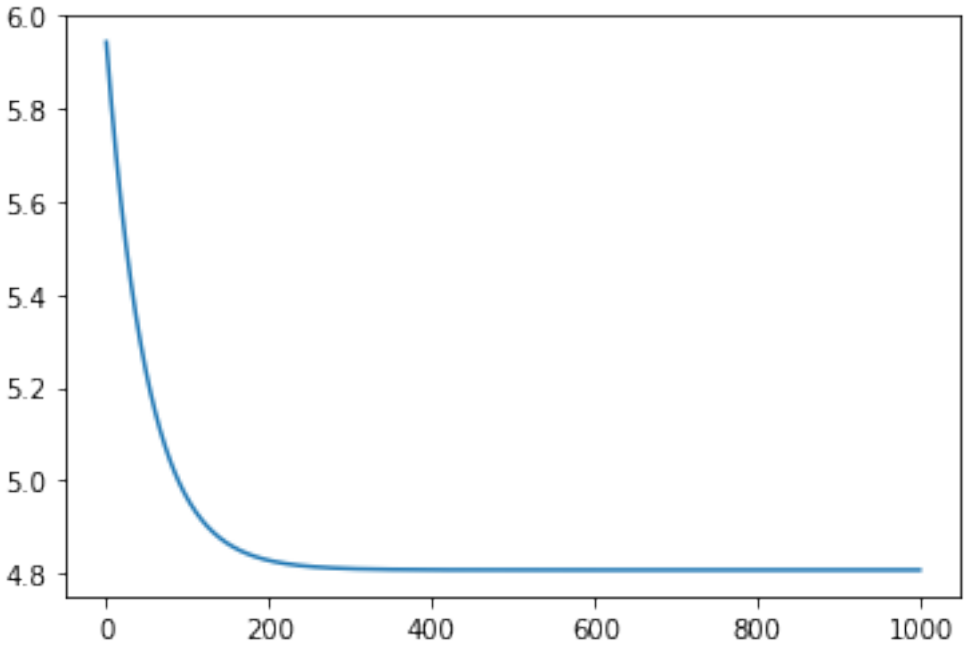


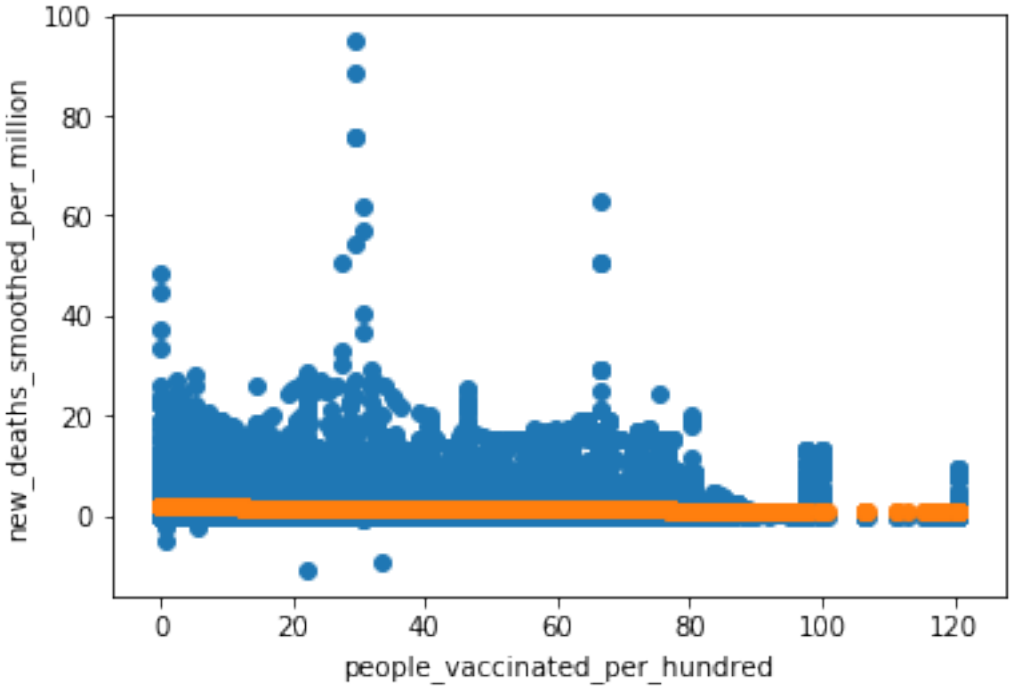
Features: **people_fully_vaccinated_per_hundred** 1. r2_score is 0.006019635166130466 2. adjusted_r2_score is 0.00576467688243032



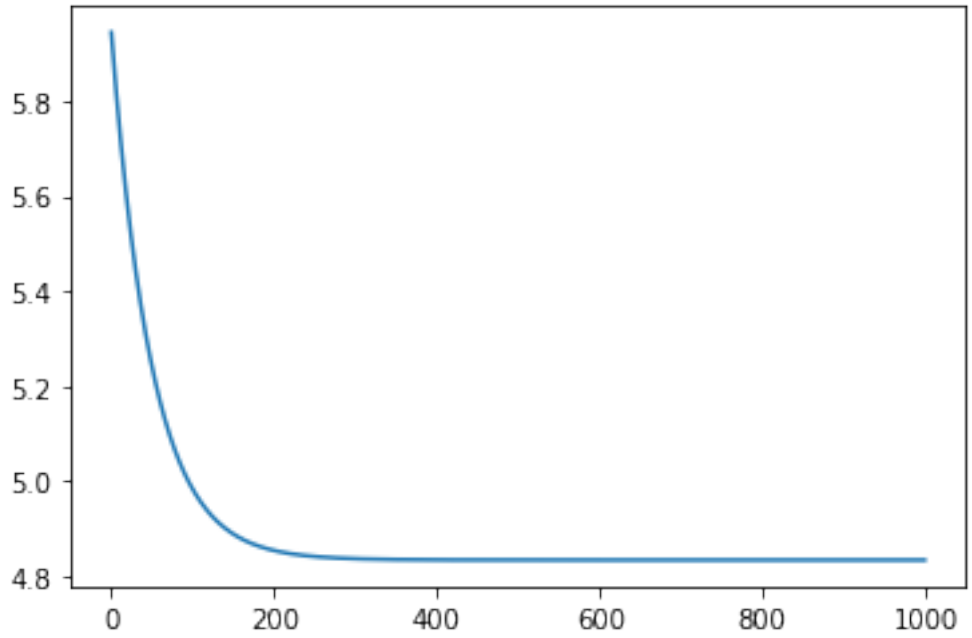


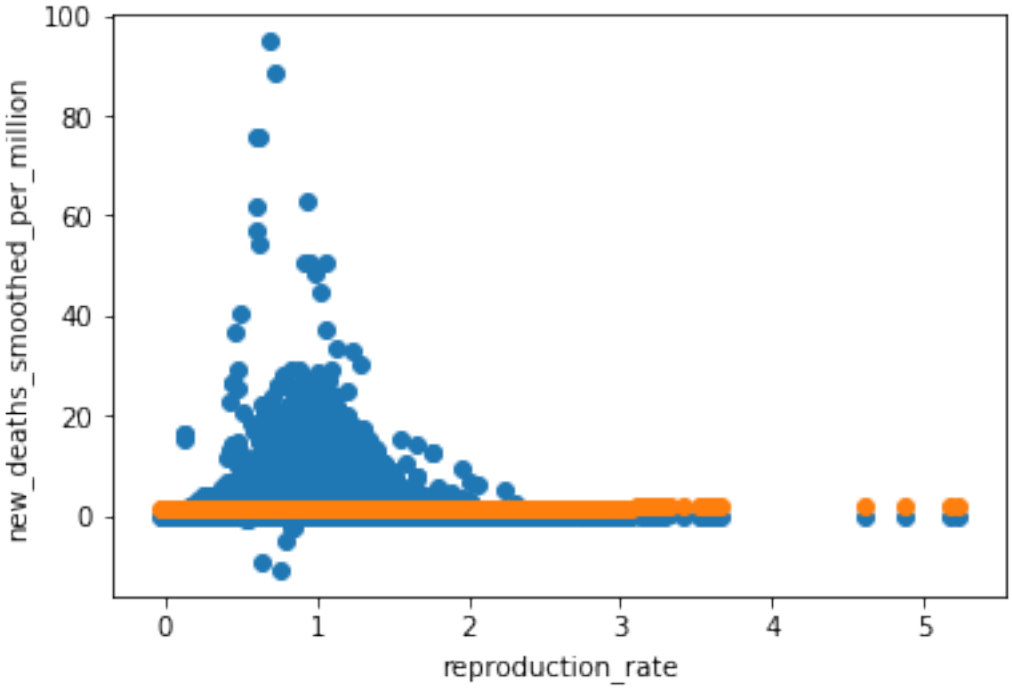
Features: **people_vaccinated_per_hundred** 1. r2_score is 0.004205067263488016 2. adjusted_r2_score is 0.003949643538885161



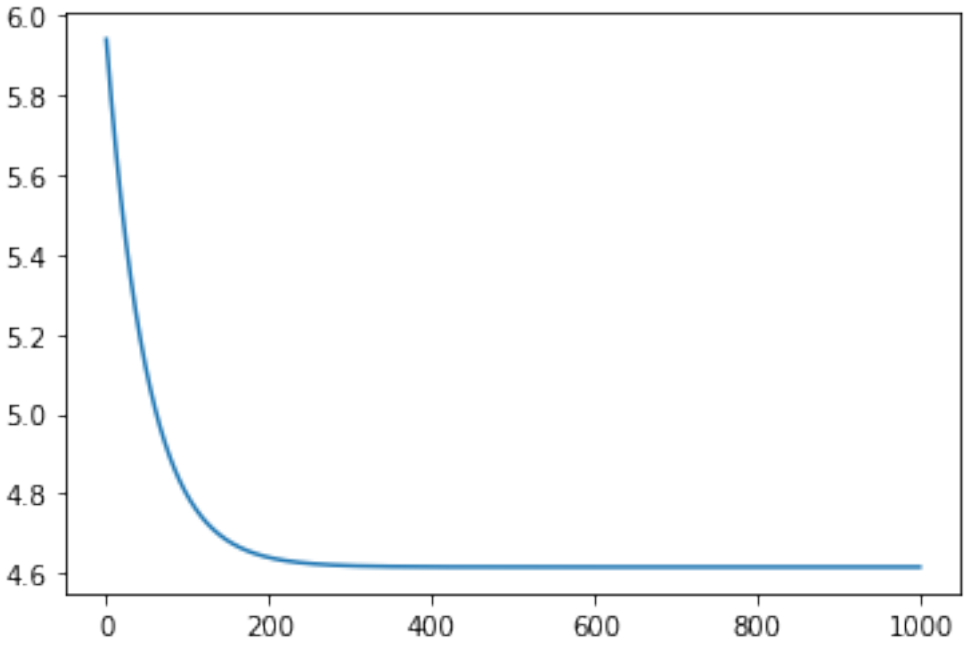


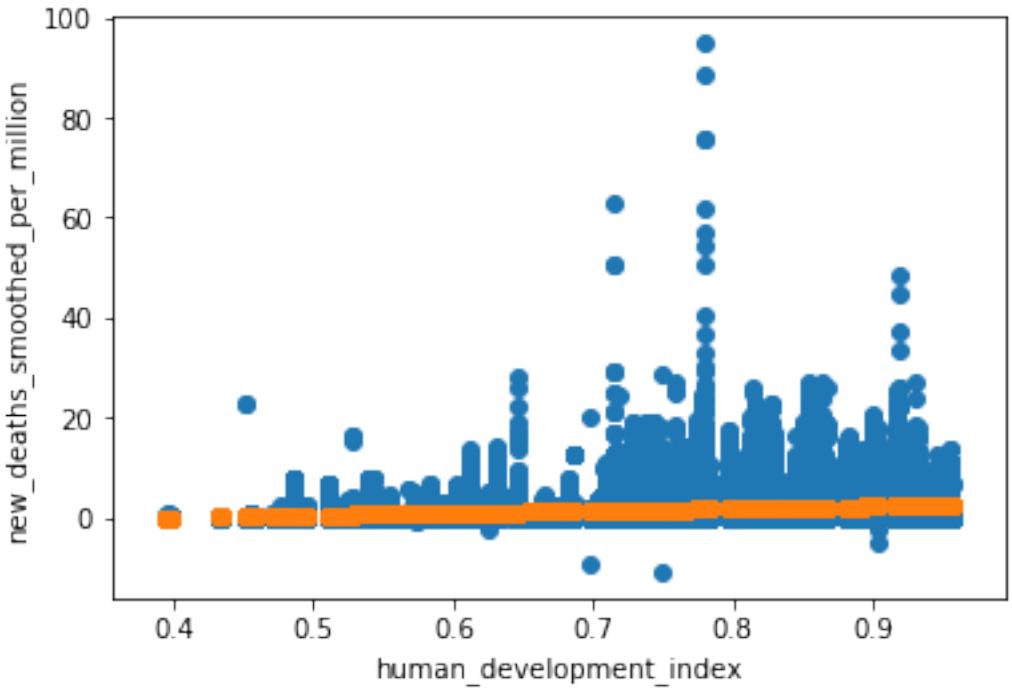
Features: **reproduction_rate** 1. r2_score is -6.392753582384891e-05 2. adjusted_r2_score is -0.0003204462675572284





Features: **human_development_index** 1. r2_score is 0.039726039188433315 2. adjusted_r2_score is 0.03947972667604127





19 Spotting linear correlation between independent variables - Task 1

1. Reorder columns so that correlation values with respect to `new_deaths_smoothed_per_million` is easier to visualise
2. Drop unnamed columns
3. Get correlation between all variables and sort according to our dependent variable, `new_deaths_smoothed_per_million`
4. Plot heatmap

Section **??**

[13]:
```
useful_cols = ["icu_patients_per_million", "new_cases_smoothed_per_million",␣
 ↪"stringency_index", "median_age", "extreme_poverty",␣
 ↪"people_fully_vaccinated_per_hundred", "people_vaccinated_per_hundred",␣
 ↪"reproduction_rate", "human_development_index",␣
 ↪"new_deaths_smoothed_per_million"]

df_appendix_one = df[useful_cols]

# Drop rows with nan
df_appendix_one = df_appendix_one.dropna()

# Get columns
```

```
cols = df_appendix_one.columns.tolist()

# Initialise dependent variable
dep_var = "new_deaths_smoothed_per_million"

# Reorder columns
idx = cols.index(dep_var)
cols = [cols[idx]] + cols[:idx] + cols[idx+1:]
df_appendix_one = df_appendix_one[cols]

# Sort and plot heatmap
corr=df_appendix_one.corr()
corr = corr.sort_values(dep_var, ascending=False)
corr.style.background_gradient(cmap='coolwarm')
```

[13]: <pandas.io.formats.style.Styler at 0x115d56b10>