Project_Brief_Template

November 26, 2021

1 2D Design Template

2 Overview

The purpose of this project is for you to apply what you have learnt in this course. This includes working with data and visualizing it, create model of linear regression or logistic regression, as well as using metrics to measure the accuracy of your model.

Please find the project handout description in the following link: - DDW-MU-Humanities Handout - DDW-MU-SocialStudies Handout

There are two parts. - Part 1 is related to predicting COVID-19 deaths - Part 2 is open ended and you can find the problem of your interest as long as it is related to COVID-19. The only requirements are the following: - The problem can be modelled either using Linear Regression (or Multiple Linear Regression) or Logistic Regression. This means either you are working with continous numerical data or classification. You are not allowed to use Neural Networks or other Machine Learning models. - You must use Python and Jupyter Notebook

The following tasks are a general guide to help you do your project for Part 2: 1. Find an interesting problem which you can solve either using Linear Regression or Classification 1. Find a dataset to build your model. You can use Kaggle to find your datasets. 1. Use plots to visualize and understand your data. 1. Create training and test data set. 1. Build your model 1. Use metrics to evaluate your model. 1. Improve your model

2.1 Deliverables

You need to submit this Jupyter notebook together with the dataset into Vocareum. Use the template in this notebook to work on this project.

2.2 Rubrics

The rubrics for the scoring can be found in this link.

2.3 Students Submission

Students' Names: - Chua Min Pei | 1005340 - Sim Yu Hui, Kellie | 1004204 - Ryan Kaw Zheng Da | 1005144 - Eunice Kwok Xiu Yi | 1005469 - Ng Zhen An | 1005527

SC02 Group 1

3 Part 1

3.1 1.1 Introduction

For Task 1, we were tasked to build a Multiple Linear Regression model that predicts the number of deaths in various countries due to COVID-19.

After some research, we decided to use the data from the site Our World in Data (OWID) since it updates information on a regular daily basis from various sources. It includes data that fall within the following metrics (as decided by OWID):

Metrics	Source	Updated	Countries
Vaccinations	Official data collated by the Our World in Data team	Daily	218
Tests & positivity	Official data collated by the Our World in Data team	Weekly	139
Hospital & ICU	Official data collated by the Our World in Data team	Weekly	38
Confirmed cases	JHU CSSE COVID-19 Data	Daily	196
Confirmed deaths	JHU CSSE COVID-19 Data	Daily	196

Metrics	Source	Updated	Countries
Reproduction rate	Arroyo-Marioli F, Bullano F, Kucinskas S, Rondón-Moreno C	Daily	185
Policy responses	Oxford COVID-19 Government Response Tracker	Daily	186
Other variables of interest	International organizations (UN, World Bank, OECD, IHME…)	Fixed	241

We will elaborate on the chosen predictor variables later in the report.

3.2 1.2 Import Libraries

```
In [91]: import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         import seaborn as sns
         import itertools
```

3.3 1.3 Import Dataset

To obtain the latest version of the dataset, the link to access the CSV file was obtained from the README of OWID's repository. The line currently commented out refers to the CSV file that was downloaded locally and later uploaded into our team member's personal GitHub repository.

```
In [92]: !git clone https://github.com/kelliesyhh/t3-2d-ddw.git

fatal: destination path 't3-2d-ddw' already exists and is not an empty directory.
```

```
In [93]: # Import dataset
         file_url = 'https://covid.ourworldindata.org/data/owid-covid-data.csv'
```

```
        # file_url = 't3-2d-ddw/task-1/data-241121.csv'
        df = pd.read_csv(file_url)
```

```
In [94]: df_first_world_countries = df[df['human_development_index'] >= 0.8]

         features = ['total_cases', 'people_vaccinated_per_hundred', 'median_age', 'icu_patien
         target = ['total_deaths']
         columns = ['date', 'iso_code'] + features + target

         df_task_1 = df_first_world_countries.loc[:, columns]

         df_task_1['date'] = pd.to_datetime(df['date'])
         mask = (df_task_1['date'] > '2021-1-1') & (df_task_1['date'] <= '2021-6-30')
         df_task_1 = df_task_1.loc[mask]
         df_task_1.dropna(inplace=True)
```

For Task 1, we decided to remove USA from the list of countries to predict on as the population size was significantly bigger for USA, meaning that it would disproportionately affect the accuracy of our model due to the high number of cases and deaths.

```
In [95]: unique_codes = list(set(df_task_1['iso_code'].unique()) - set(["USA"]))

         print("Number of Countries:", len(unique_codes))
         print(unique_codes)

         df_task_1 = df_task_1[df_task_1['iso_code'].isin(unique_codes)]
```

```
Number of Countries: 27
['SRB', 'PRT', 'SVK', 'EST', 'SVN', 'NLD', 'CHE', 'ITA', 'CZE', 'ESP', 'IRL', 'BEL', 'FIN', 'DI
```

```
In [96]: df_task_1.to_csv("2d-task-1-v2.csv")
```

3.4 1.4 Multiple Linear Regression Model

3.4.1 1.4.1 Visualisation and Plots

For visualisation of the data, we made use of Matplotlib and Seaborn that were introduced to us in class and the pre-class material.

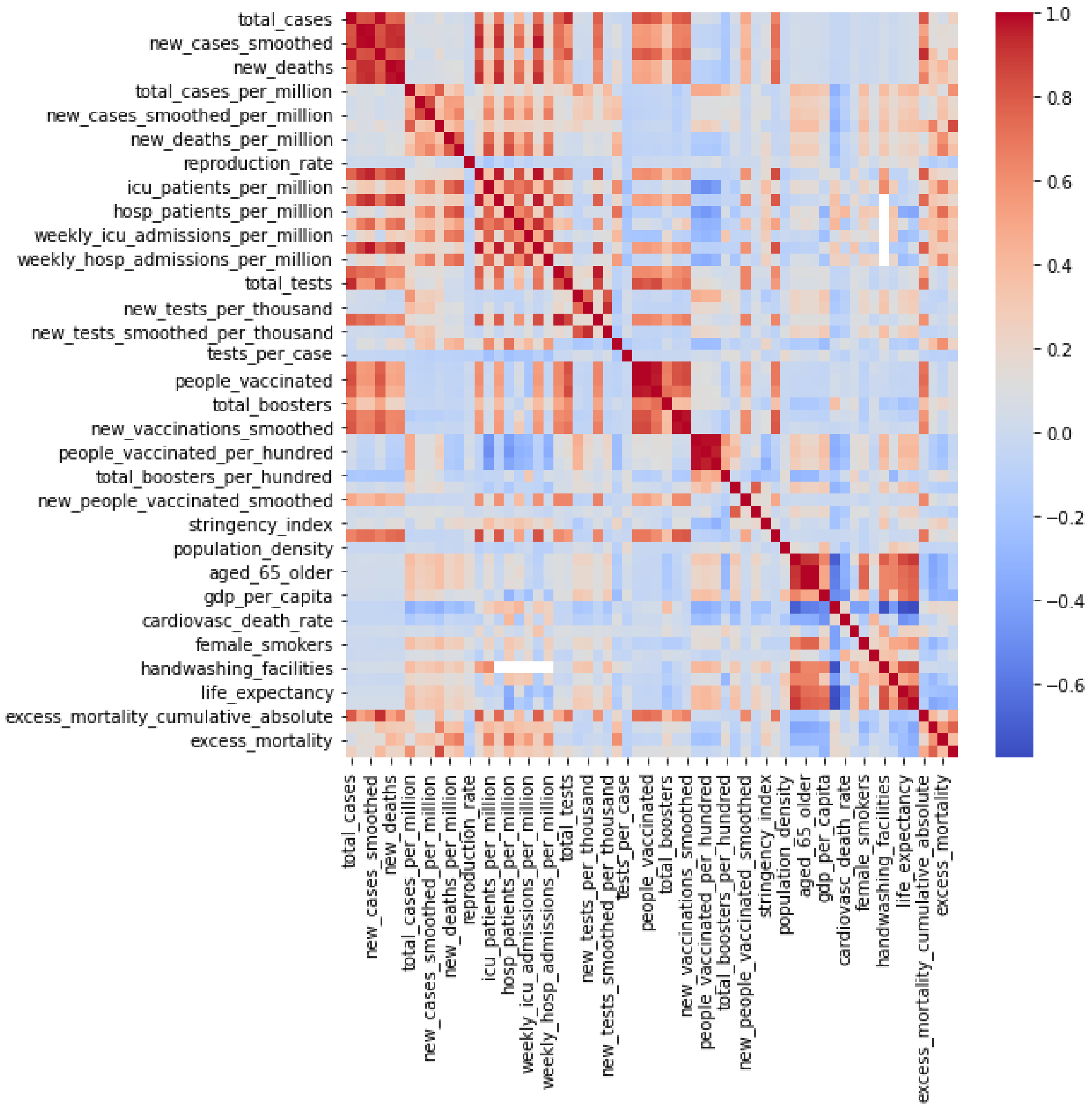
As the code for visualisation was repeated over the course of this report, we decided to insert them as functions for easy usage in the other cells.

For any data that required preprocessing or further DataFrame analysis, we made use of codes from the cohort lessons and pre-class material.

```
In [97]: def determine_correlation(df, column_name):
             """Takes a DataFrame and a column name, return a DataFrame containing the correlati
             pd.set_option('display.max_rows', None)
             correlations = df.corr()
             df_correlation = pd.DataFrame(correlations.loc[:, [column_name]])
```

```
        return df_correlation[[column_name]].sort_values(by=column_name, ascending=False)[1

    def plot_correlation_heatmaps(df, figsize, annot=False):
        """Takes a DataFrame, the figsize of the heatmap to be plotted, and whether the hea
        correlations = df.corr()
        plt.figure(figsize=figsize)
        sns.heatmap(correlations, cmap="coolwarm", annot=annot)
        plt.show()
```

```
In [98]: df_correlation = determine_correlation(df, "total_deaths")
         plot_correlation_heatmaps(df, figsize=(8,8), annot=False)
         df_correlation
```



```
Out[98]:                                             total_deaths
         total_cases                                     0.985001
         excess_mortality_cumulative_absolute            0.924791
         new_cases_smoothed                              0.833336
         new_cases                                       0.820425
         total_tests                                     0.816577
         new_deaths_smoothed                             0.807415
         people_vaccinated                               0.801580
         new_vaccinations_smoothed                       0.792568
         new_deaths                                      0.785400
         total_vaccinations                              0.777244
         new_vaccinations                                0.771464
         icu_patients                                    0.769921
         people_fully_vaccinated                         0.750289
         weekly_hosp_admissions                          0.737262
         hosp_patients                                   0.711575
         new_tests                                       0.695591
         new_tests_smoothed                              0.670885
         population                                      0.660583
         total_boosters                                  0.487783
         new_people_vaccinated_smoothed                  0.480324
         weekly_icu_admissions                           0.408002
         excess_mortality_cumulative_per_million         0.395564
         excess_mortality_cumulative                     0.346920
         total_deaths_per_million                        0.161883
         icu_patients_per_million                        0.159915
         excess_mortality                                0.133797
         total_cases_per_million                         0.104597
         weekly_hosp_admissions_per_million              0.066179
         new_vaccinations_smoothed_per_million           0.052027
         new_deaths_smoothed_per_million                 0.047599
         total_vaccinations_per_hundred                  0.045578
         people_vaccinated_per_hundred                   0.044310
         stringency_index                                0.041968
         handwashing_facilities                          0.040609
         new_people_vaccinated_smoothed_per_hundred      0.039079
         human_development_index                         0.037306
         new_deaths_per_million                          0.035953
         median_age                                      0.032146
         positive_rate                                   0.029646
         hosp_patients_per_million                       0.028394
         total_tests_per_thousand                        0.028327
         new_cases_smoothed_per_million                  0.027670
         aged_65_older                                   0.027497
         aged_70_older                                   0.024743
         diabetes_prevalence                             0.024439
         new_cases_per_million                           0.023086
         life_expectancy                                 0.022771
```

```
gdp_per_capita                                     0.008333
people_fully_vaccinated_per_hundred                0.004510
reproduction_rate                                  0.000631
new_tests_per_thousand                            -0.000498
male_smokers                                      -0.005810
new_tests_smoothed_per_thousand                   -0.005964
female_smokers                                    -0.007083
hospital_beds_per_thousand                        -0.010101
population_density                                -0.019497
weekly_icu_admissions_per_million                 -0.026093
cardiovasc_death_rate                             -0.041108
extreme_poverty                                   -0.041609
tests_per_case                                    -0.046486
total_boosters_per_hundred                        -0.184871
```

Based on our analysis above, we narrowed down to 4 different predictor variables to be used for our Multiple Linear Regression model. We observed that the main 3 categories that have a higher correlation with total deaths are number of cases, vaccinations and number of ICU patients. Thus, these 3 categories of variables were used in our model as predictor variables.

As datasets from different countries were used to train our model, we believe that the background of the countries should also be taken into consideration. Thus, the median age of the population was also selected to be part of our predictor variables.

In this model, we would like to use these features to test and see how they come together to affect the total death.

The predictor variables, the relevant descriptions, and metrics (mentioned in Section 1.1 above) can be seen in the table below.

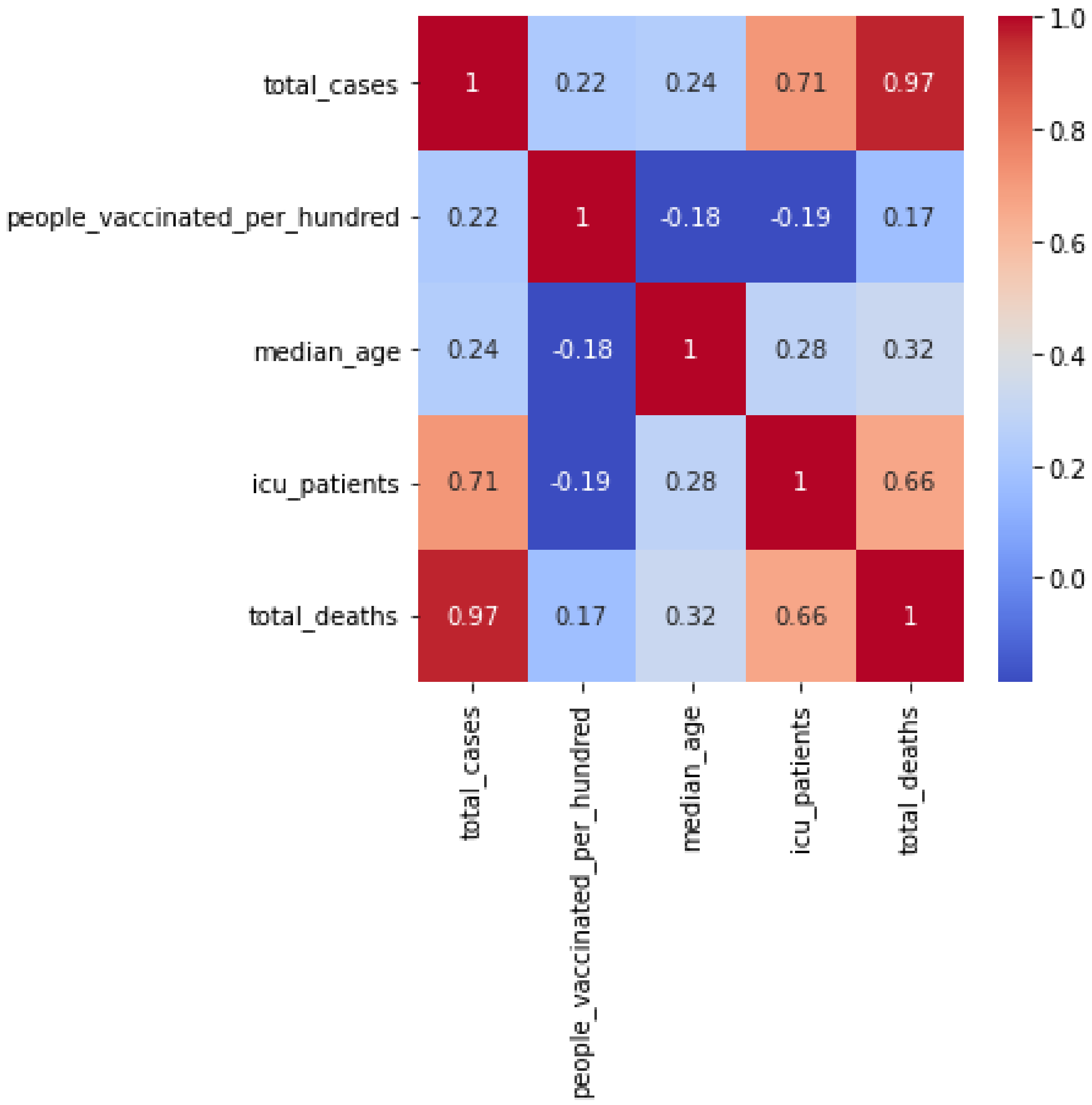
Predictor Variables (X)

Variable	Description	Metrics (from OWID)
total_cases	Total confirmed cases of COVID-19	Confirmed cases
people_vaccinated_per_hundred	Total number of people who received at least one vaccine dose per 100 people in the total population	Vaccinations
icu_patients	Number of COVID-19 patients in intensive care units (ICUs) on a given day	Hospital & ICU
median_age	Median age of the population, UN projection for 2020	Others

Predicted Variable (y)

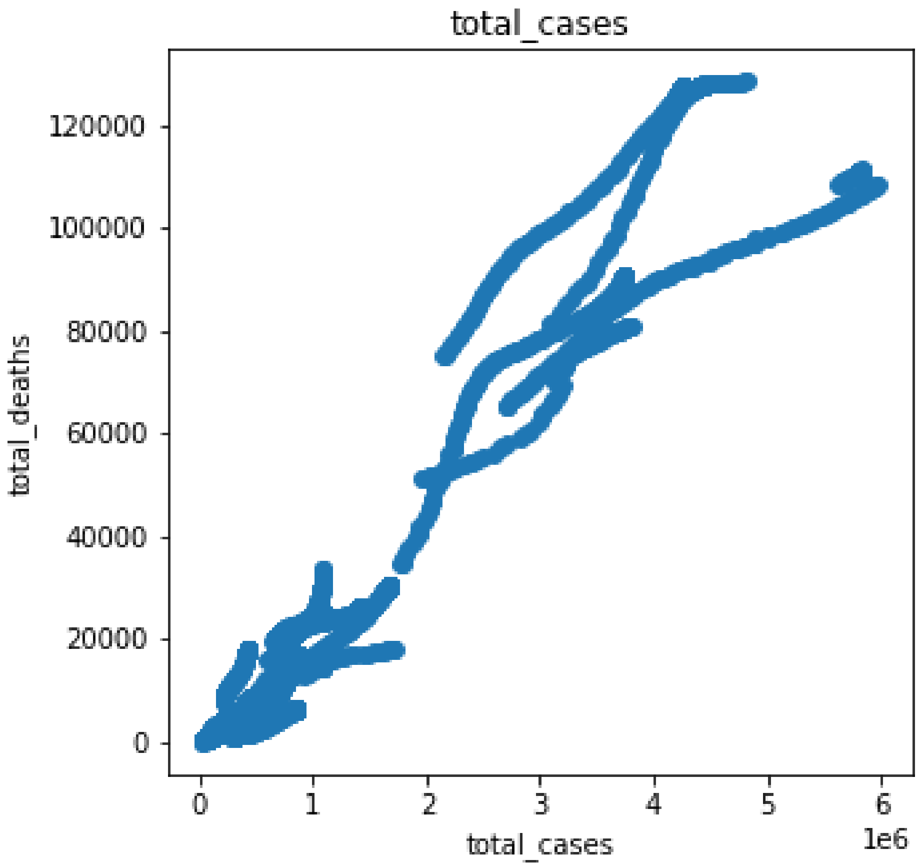
Variable	Description	Category
total_deaths	Total deaths attributed to COVID-19	Confirmed deaths

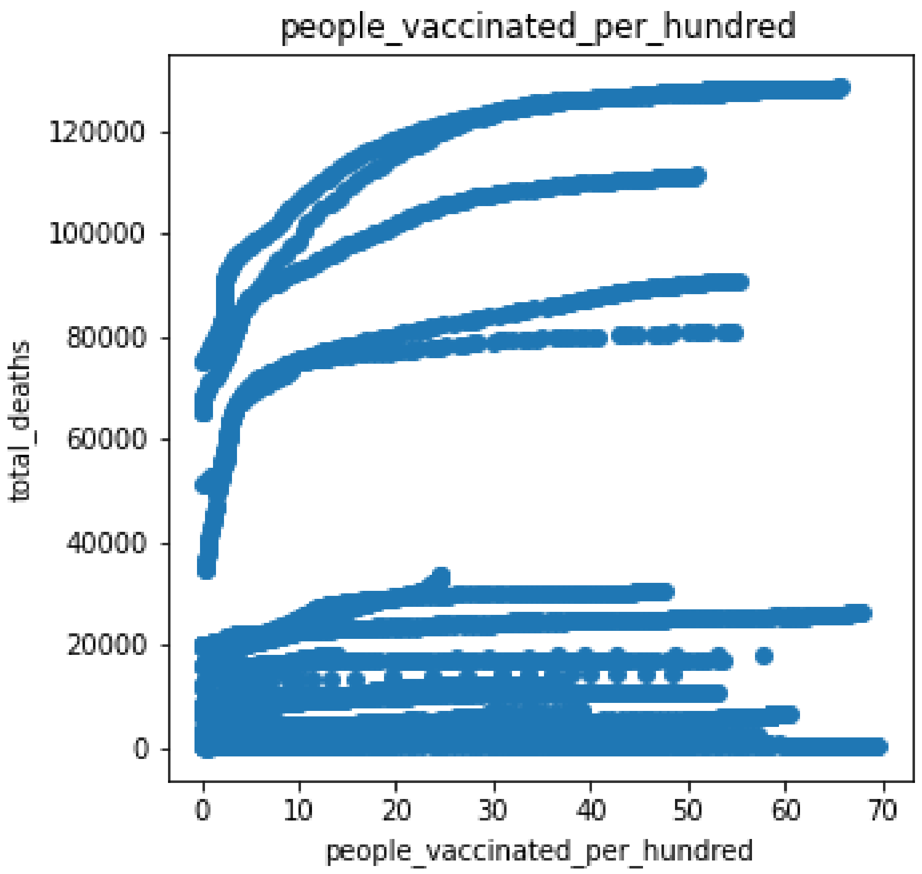
```
In [99]: features = ['total_cases', 'people_vaccinated_per_hundred', 'median_age', 'icu_patien
         target = ['total_deaths']
         columns = features + target

         df_task_1 = df_task_1[columns]
         plot_correlation_heatmaps(df_task_1, figsize=(5,5), annot=True)
```

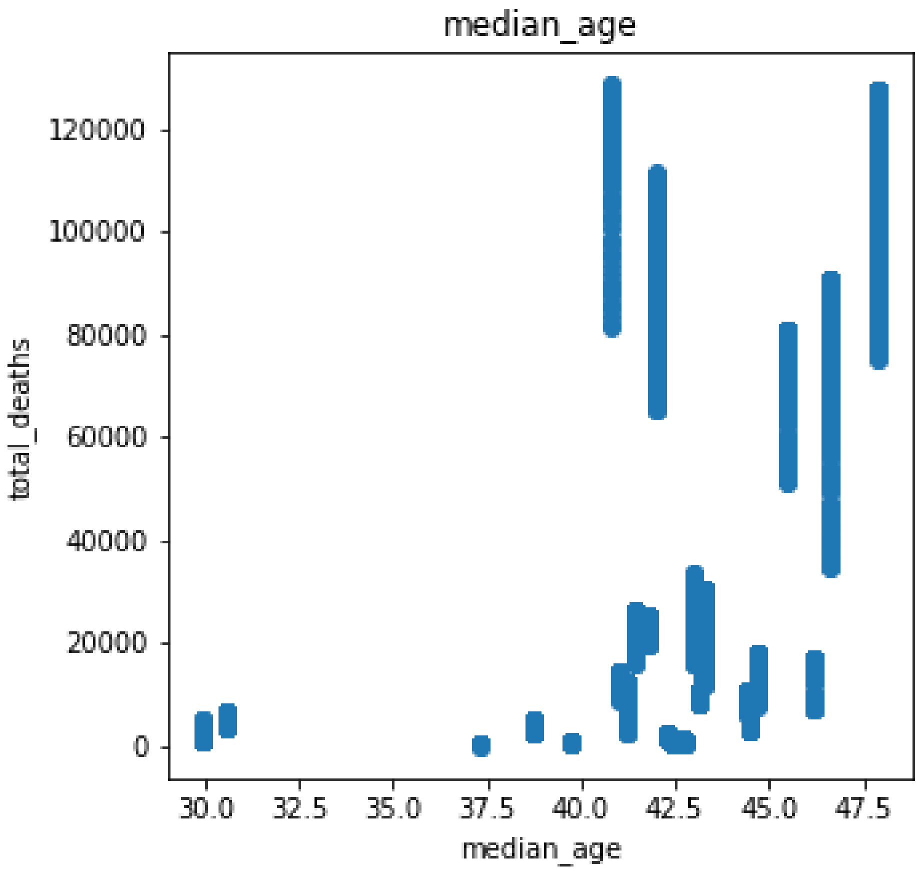


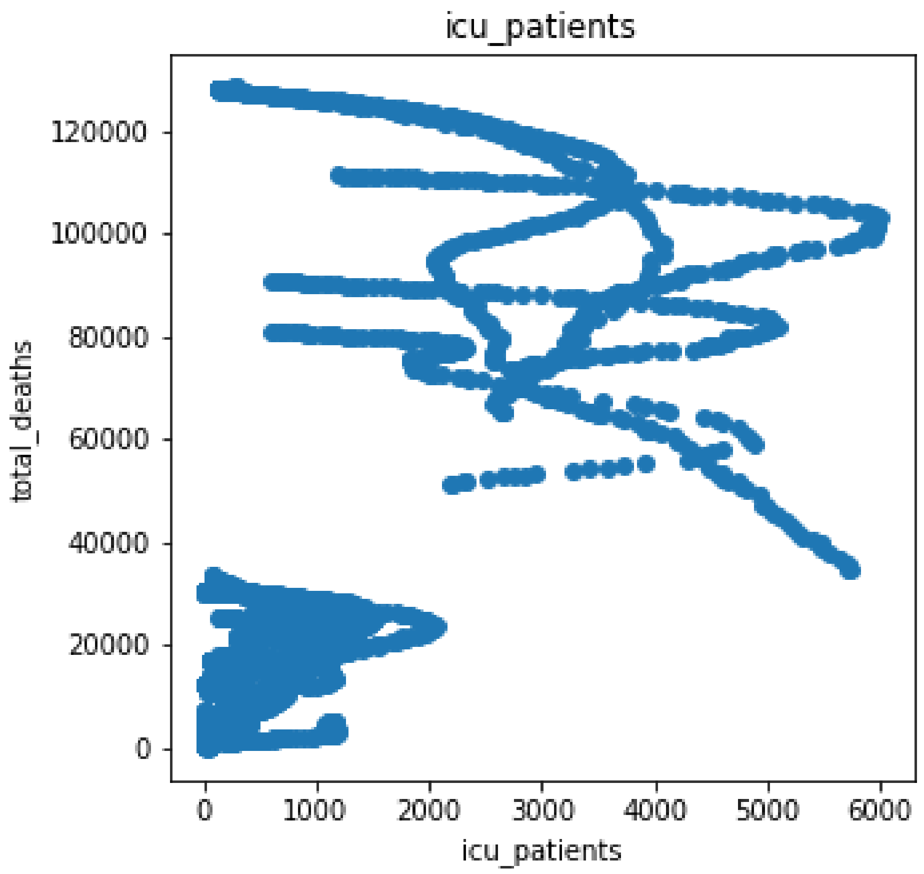
```
In [100]: # Plotting all X variables (total_cases, people_vaccinated_per_hundred,
          # median_age, icu_patients) against y (total_deaths)
```

```
for col in features:
  plt.figure(figsize=(5,5))
  plt.xlabel(col)
  plt.ylabel("total_deaths")
  plt.title(col)
  plt.scatter(df_task_1[col], df_task_1["total_deaths"])
  plt.show()
```




people_vaccinated_per_hundred
total_deaths
120000
100000
80000
60000
40000
20000
0
0
10
20
30
40
50
60
70
people_vaccinated_per_hundred


median_age
total_deaths
120000
100000
80000
60000
40000
20000
0
30.0
32.5
35.0
37.5
40.0
42.5
45.0
47.5
median_age



3.4.2 1.4.2 Helper Functions and Code for Model

In this section, we defined functions that were utilised in the model, taken from the cohort lessons and pre-class material.

```
In [101]: def normalize_z(df):
            """Takes a DataFrame, returns a DataFrame with normalized values using z-score nor
            dfout = (df - df.mean(axis=0)) / df.std(axis=0)
            return dfout

          def normalize_minmax(df):
            """Takes a DataFrame, returns a DataFrame with normalized values using min-max nor
            dfout = (df - df.min(axis=0)) / (df.max(axis=0) - df.min(axis=0))
            return dfout

          def transform_features(df_feature, colname, colname_transformed):
            """Takes a DataFrame, the name of a column to be transformed, and the name for the
            df_feature[colname_transformed] = df[colname].apply(lambda x: x**2)
            return df_feature
```

```
def get_features_targets(df, feature_names, target_names):
  """Takes a DataFrame, a list of columns for the features, and a list of columns fo
  df_feature = df.loc[:, feature_names]
  df_target = df.loc[:, target_names]
  return df_feature, df_target

def prepare_feature(df_feature):
  """Takes a DataFrame containing the features, convert it into a numpy array, chang
  cols = len(df_feature.columns)
  np_feature = df_feature.to_numpy().reshape(-1, cols)
  constants = np.ones(shape=(np_feature.shape[0], 1))
  return np.concatenate((constants, np_feature), axis=1)

def prepare_target(df_target):
  """Takes a DataFrame containing the target, convert it into a numpy array, change
  cols = len(df_target.columns)
  np_target = df_target.to_numpy().reshape(-1, cols)
  return np_target

def predict(df_feature, beta):
  """Takes a DataFrame and an array of beta values, returns the predicted y values a
  df_feature = normalize_z(df_feature)
  np_X = prepare_feature(df_feature)
  return predict_norm(np_X, beta)

def predict_norm(X, beta):
  """Takes a Numpy array and an array of beta values, returns the straight line equa
  y_pred = np.matmul(X, beta)
  return y_pred

def split_data(df_feature, df_target, random_state=None, test_size=0.5):
  """Takes a DataFrame containing the features, a DataFrame containing the target, t
  # indexes = which is the number of rows
  indexes = df_feature.index
  if random_state != None:
      np.random.seed(random_state)

  # k = length / size of the test array
  k = int(test_size * len(indexes))

  test_index = np.random.choice(indexes, k, replace=False)
  train_index = list(set(indexes) - set(test_index))

  df_feature_train = df_feature.loc[train_index, :]
  df_feature_test = df_feature.loc[test_index, :]
  df_target_train = df_target.loc[train_index, :]
  df_target_test = df_target.loc[test_index, :]
```

```
          return df_feature_train, df_feature_test, df_target_train, df_target_test

In [102]: def compute_cost(X, y, beta):
            """Takes a Numpy array containing the features, a Numpy array containing the targe
            J = 0
            m = X.shape[0]
            error = np.matmul(X, beta) - y
            error_sq = np.matmul(error.T, error)
            J = (1/(2*m))*error_sq
            J = J[0][0]
            return J


          def gradient_descent(X, y, beta, alpha, num_iters):
            """Takes a Numpy array containing the features, a Numpy array containing the targe
            m = X.shape[0]
            J_storage = np.zeros((num_iters,1))
            for n in range(num_iters):
              deriv = np.matmul(X.T, (np.matmul(X, beta)-y))
              beta = beta - alpha * (1/m) * deriv
              J_storage[n] = compute_cost(X, y, beta)
            return beta, J_storage

In [103]: def multiple_linear_regression(df_features, df_target):
            # Normalize the features using z normalization
            df_features = normalize_z(df_features)

            # Change the features and the target to numpy array using the prepare functions
            X = prepare_feature(df_features)
            target = prepare_target(df_target)

            iterations = 20000
            alpha = 0.1
            beta = np.zeros((X.shape[1], 1))

            # Call the gradient_descent function
            beta, J_storage = gradient_descent(X, target, beta, alpha, iterations)

            # Call the predict() method
            pred = predict(df_features, beta)

            # Plotting figures for visualisation
            # plt.figure()
            # plt.title("J_storage")
            # plt.plot(J_storage)
            for column in df_features.columns:
              plt.figure()
              plt.xlabel(column)
```

```
    plt.ylabel("total_deaths")
    plt.scatter(df_features[column], target, color='orange')
    plt.scatter(df_features[column], pred, color='blue')
    plt.show()

  # Calculate R^2, Adjusted R^2
  print("R^2 Score: ", r2_score(target, pred))
  print("Adjusted R^2 Score: ", adjusted_r2_score(r2_score(target, pred), X.shape[0]
  return beta
```

3.4.3 1.4.3 Finding the Best Model

In order to find the best model for multiple linear regression, we experimented with all 15 combinations of the 4 features. For our data, X1, X2, X3, X4 would give us the 15 combinations below:

1 feature	2 features	3 features	4 features
X1	X1 + X2	X1 + X2 + X3	X1 + X2 + X3 + X4
X2	X2 + X3	X2 + X3 + X4	
X3	X3 + X4	X3 + X4 + X1	
X4	X4 + X1	X4 + X1 + X2	
X2 + X4			
X1 + X3			

This was done using Excel's Analysis Toolpak and later validated using Python codes. The results for this can be found in this section.

1.4.3.1 Accuracy Metrics As mentioned in the task brief, R2 is not a good metric to be used for Multiple Linear Regression. As such, we decided to take into account the metric Adjusted R2, which will show us if the variables used were potentially overfitting our model.

The Adjusted R2 Score is an improved version of R2 score that penalizes us for adding an independent variable that does not help in predicting the dependent variable [1] [2]. By taking into account the number of independent variables used for predicting the target variable, we can determine whether adding new variables to the model actually increases the model fit [2].

Sources: 1. https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/ 2. https://www.analyticsvidhya.com/blog/2020/07/difference-between-r-squared-and-adjusted-r-squared/

```
In [104]: def r2_score(target, pred):
            """Takes a Numpy array containing the target and a Numpy array containing the pred
            diff = target - pred
            ssres = np.matmul(diff.T,diff)[0][0]
            target_mean = np.mean(target)
            diff_mean = target - target_mean
            sstot = np.matmul(diff_mean.T, diff_mean)[0][0]
            return 1 - (ssres/sstot)

          def adjusted_r2_score(r2, n, k):
```

```
        """Takes the R^2 score, number of data points, and number of features, returns the
        num = (1 - r2) * (n - 1)
        den = n - k - 1
        return 1 - num/den
```

1.4.3.2 P-Value In statistics, the p-value is the probability of obtaining results at least as extreme as the observed results of a statistical hypothesis test, assuming that the null hypothesis is correct [1]. In other words, a p-value is a measure of the probability that an observed difference could have occurred just by random chance [1]. The p-value is used as an alternative to rejection points to provide the smallest level of significance at which the null hypothesis (which proposes that there is no difference between certain characteristics of a population [2]) would be rejected [1].

The level of statistical significance is often expressed as a p-value between 0 and 1 [3]. The smaller the p-value, the stronger the evidence that you should reject the null hypothesis. A p-value less than 0.05 (typically 0.05) is statistically significant [3].

In our Excel, we obtained p-values for each of our models, which was later used as a method of deciding whether the feature should be used.

Sources: 1. https://www.investopedia.com/terms/p/p-value.asp 2. https://www.investopedia.com/terms/n/null_hypothesis.asp 3. https://www.simplypsychology.org/p-value.html

1.4.3.3 Assumptions For this report, given that we used the total cases, number of people vaccinated, median age and ICU patients to predict the total_deaths, we had to make the assumption that all the variables were independent of each other.

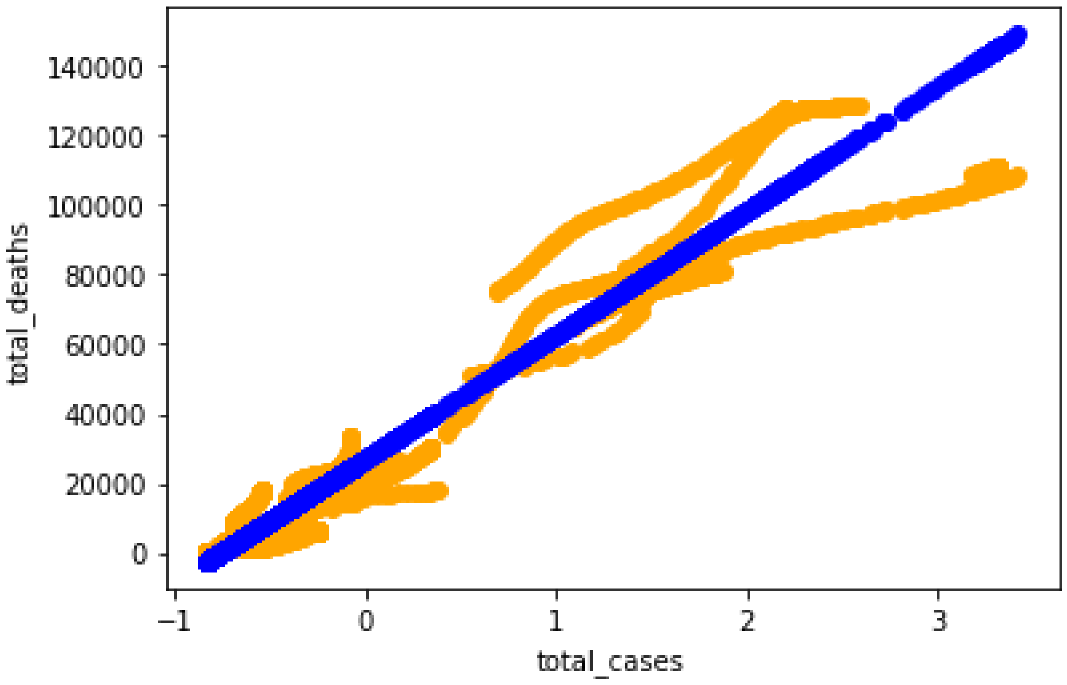
However, in real life, we note that such an ideal situation may not occur. In actuality, the total number of cases in a country could be dependent on the number of people vaccinated or ICU patients and hence it may not be as linear as we expect it to be for the sake of this report.

1.4.3.4 Comparison with Multiple Linear Regression with Python The Linear Regression models were also similarly implemented using Python. Adjusted R2 was used as a metric to determine if the Excel was on the right track.

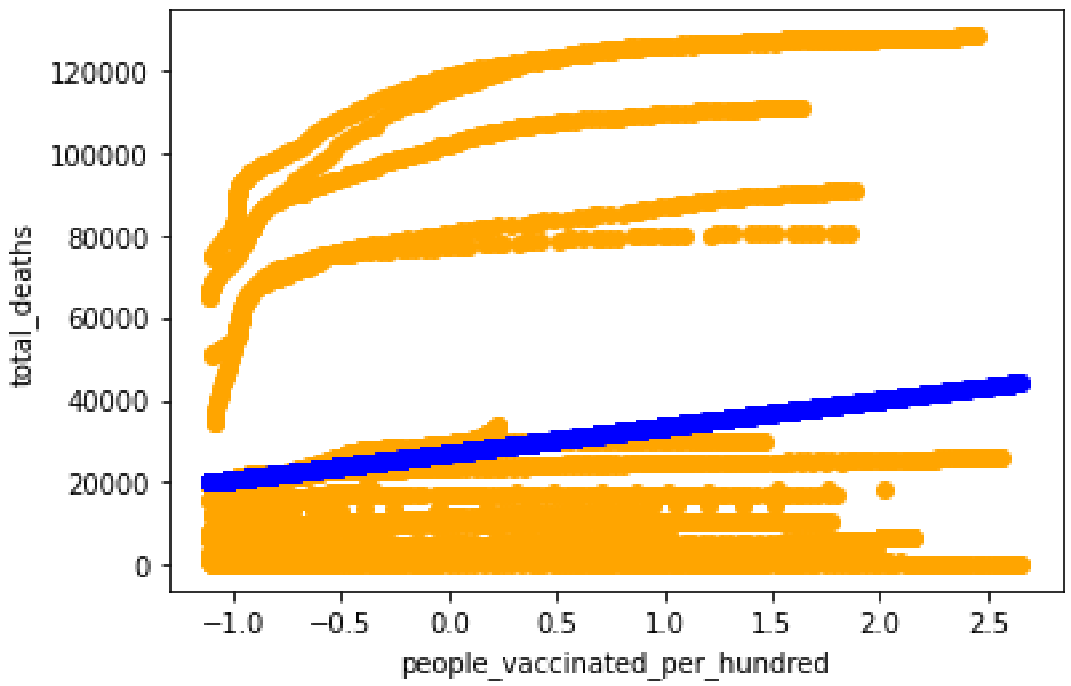
```
In [105]: # Obtaining all 15 combinations of our 4 features
          feature_combis = []
          for x in range(1, len(features)+1):
              for subset in itertools.combinations(features, x):
                feature_combis.append(subset)
```

```
In [106]: # Running Multiple Linear Regression on each combination of features
          for combi in feature_combis:
            print("Feature(s) Used: ", combi)
            df_features, df_target = get_features_targets(df_task_1, combi, target)
            beta = multiple_linear_regression(df_features, df_target)
            print("Beta Coefficient Values: ", beta)
            print("\n")
```

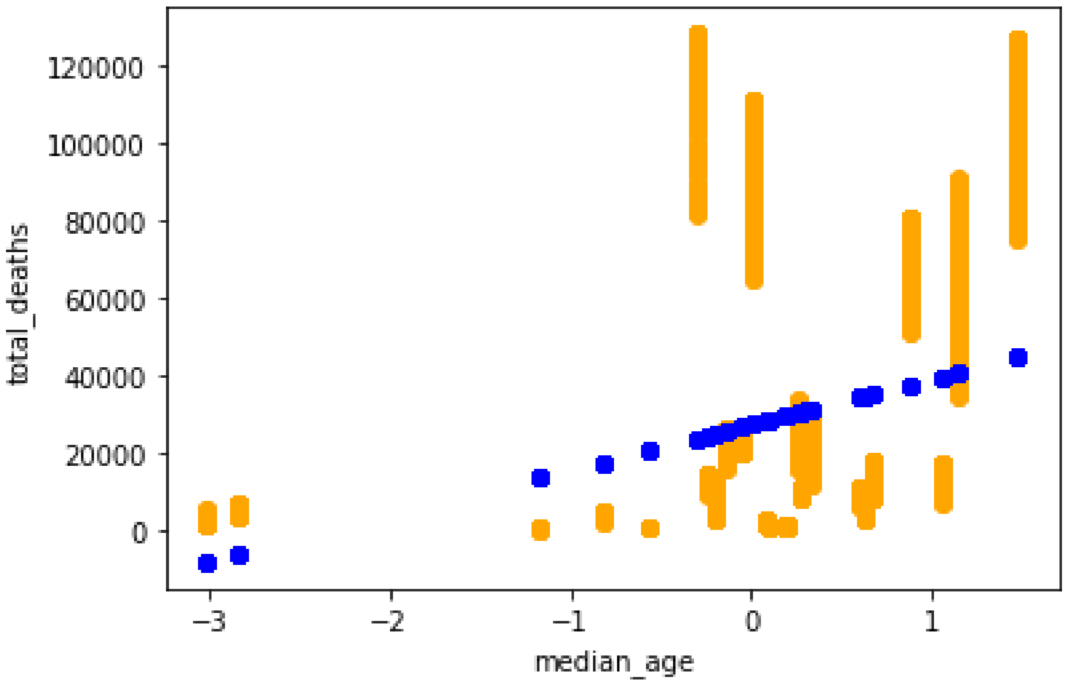
```
Feature(s) Used:  ('total_cases',)
```



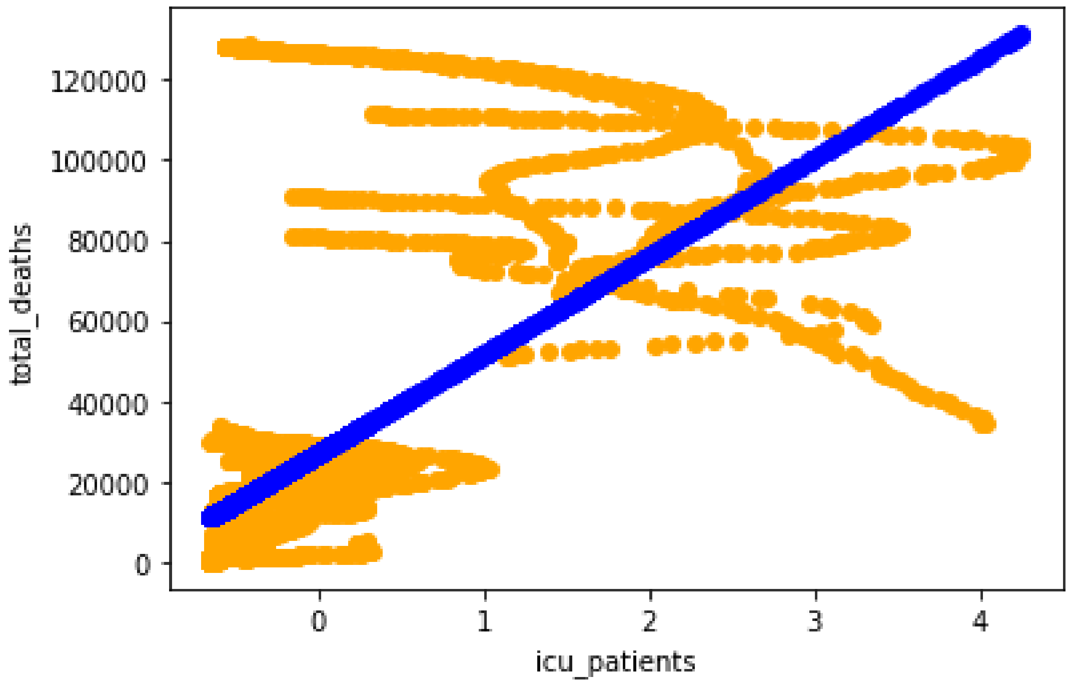
```
R^2 Score:  0.935710431431667
Adjusted R^2 Score:  0.9356786678108329
Beta Coefficient Values:  [[27166.65169094]
 [35596.38213657]]


Feature(s) Used:  ('people_vaccinated_per_hundred',)
```



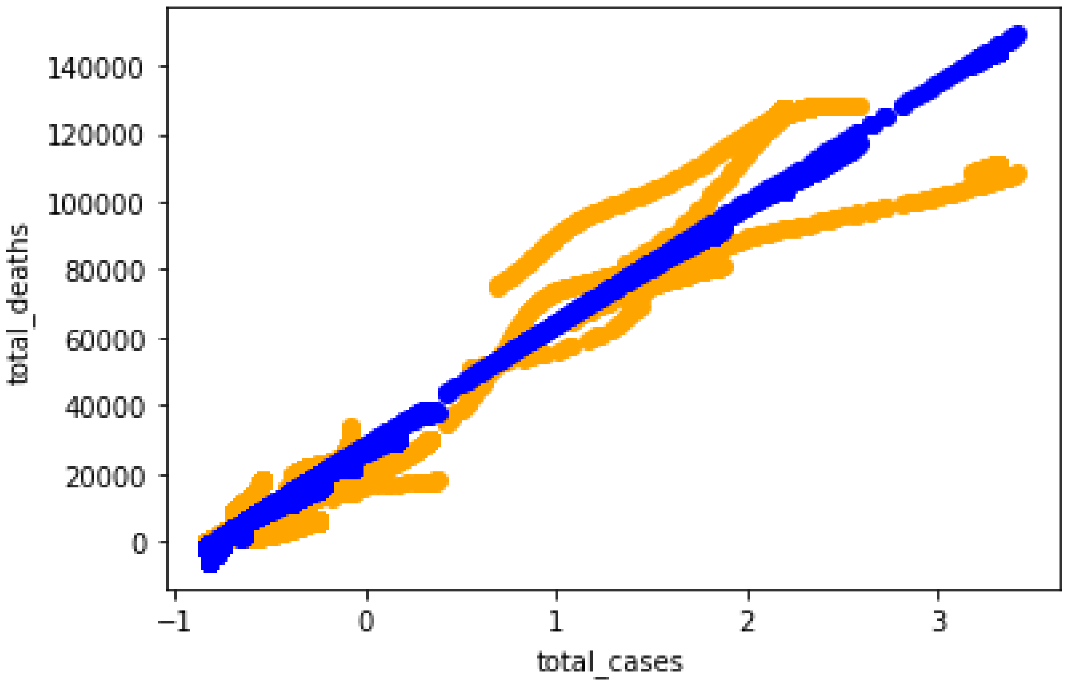
```
R^2 Score:  0.030613911953687234
Adjusted R^2 Score:  0.030134966258012152
Beta Coefficient Values:  [[27166.65169094]
 [ 6438.64587017]]


Feature(s) Used:  ('median_age',)
```

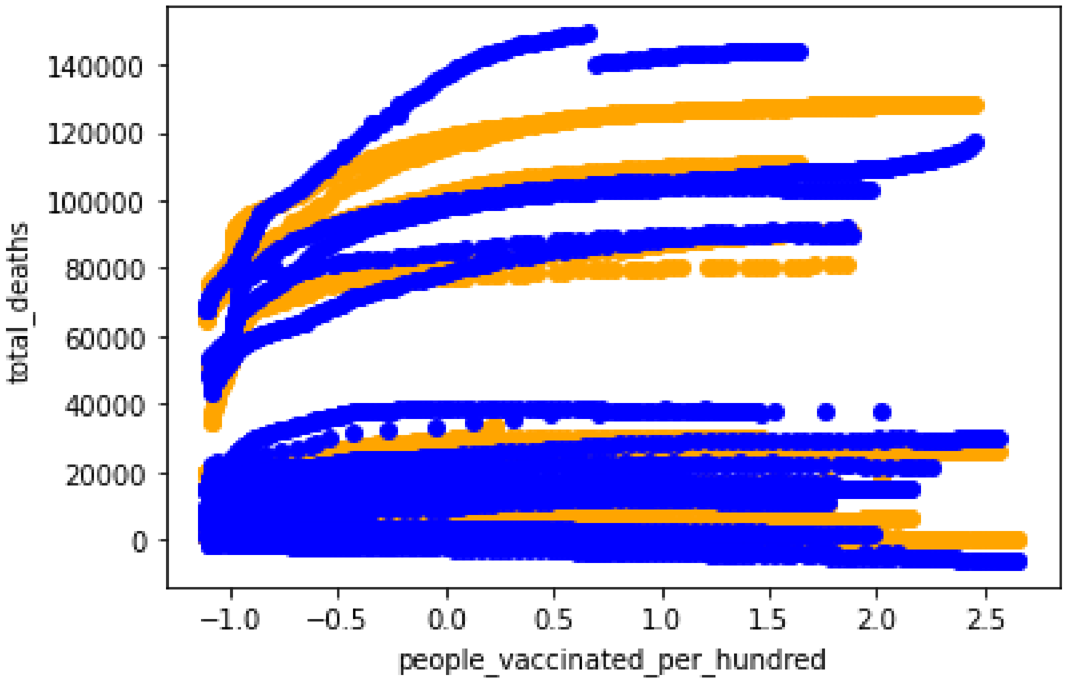


```
R^2 Score:  0.10348115070561614
Adjusted R^2 Score:  0.10303820661011498
Beta Coefficient Values:  [[27166.65169094]
 [11837.65693348]]


Feature(s) Used:  ('icu_patients',)
```



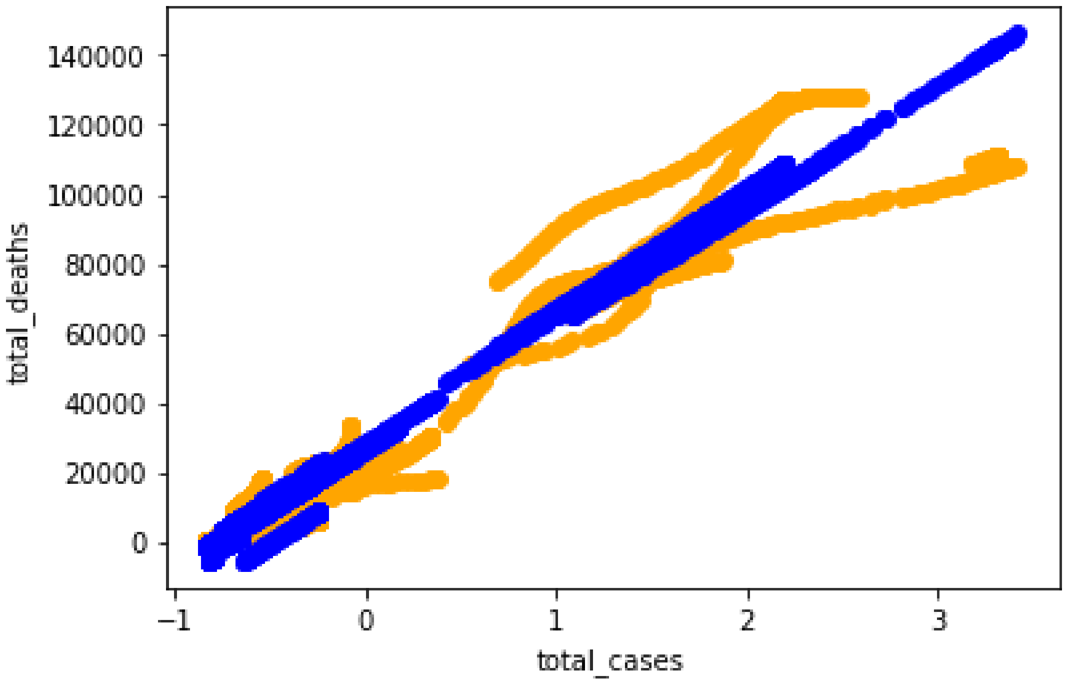
```
R^2 Score:  0.441738805422 00106
Adjusted R^2 Score:  0.4414629846736918
Beta Coefficient Values:  [[27166.65169094]
 [24457.82881978]]


Feature(s) Used:  ('total_cases', 'people_vaccinated_per_hundred')
```

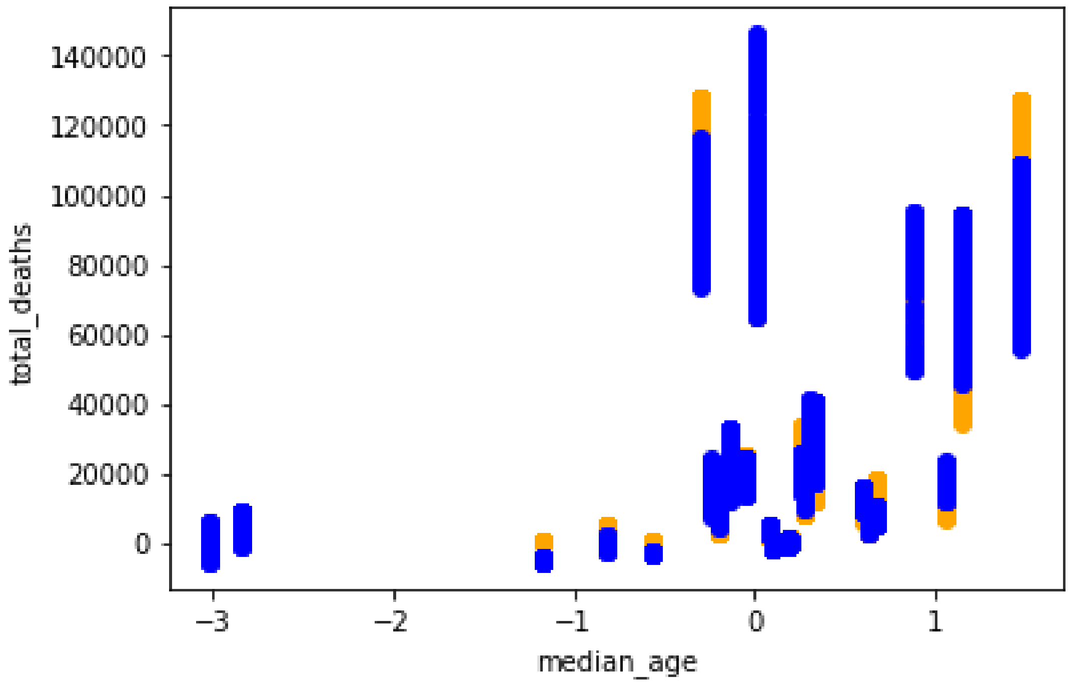




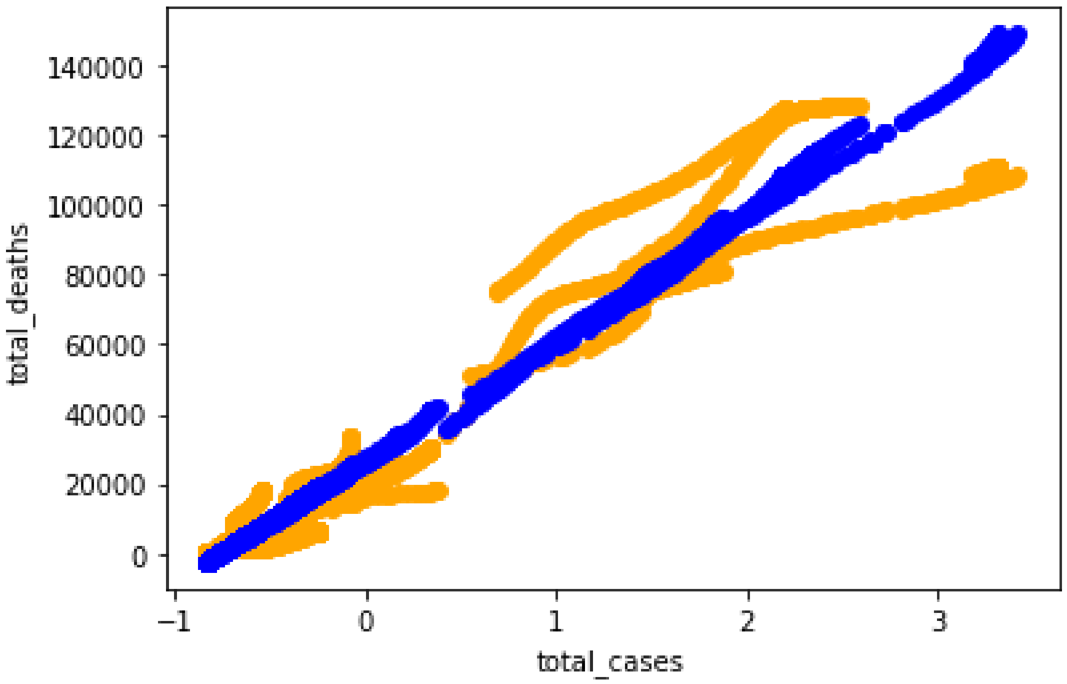
```
R^2 Score:  0.9370133659558506
Adjusted R^2 Score:  0.9369666746037052
```

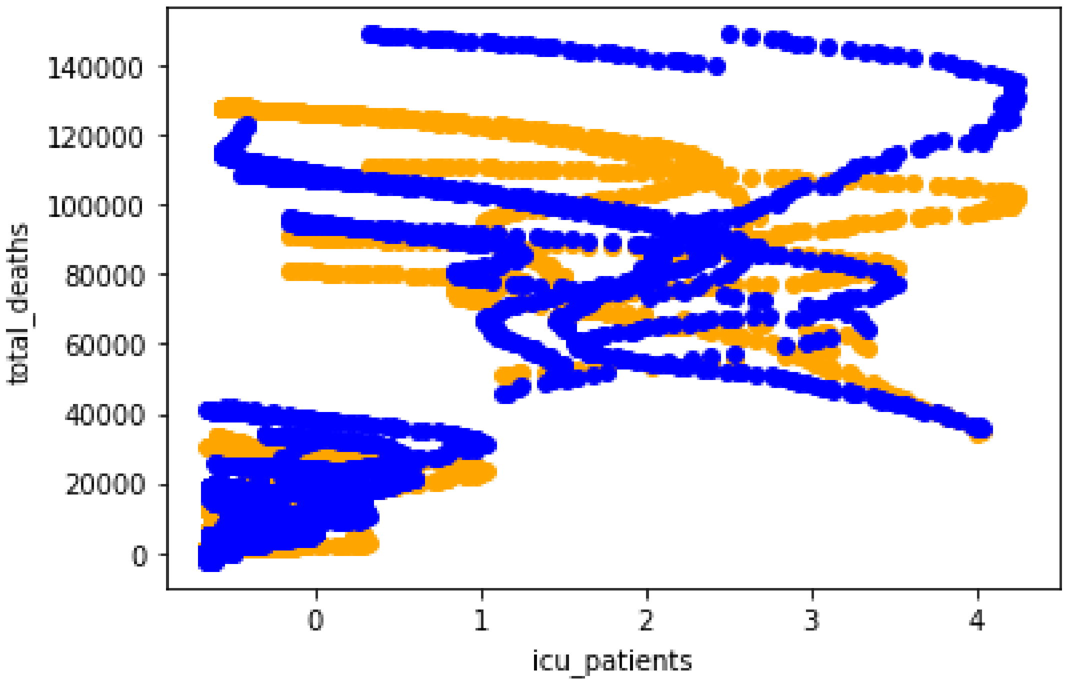
```
Beta Coefficient Values:  [[27166.65169094]
 [35892.0923096 ]
 [-1360.81865527]]


Feature(s) Used:  ('total_cases', 'median_age')
```





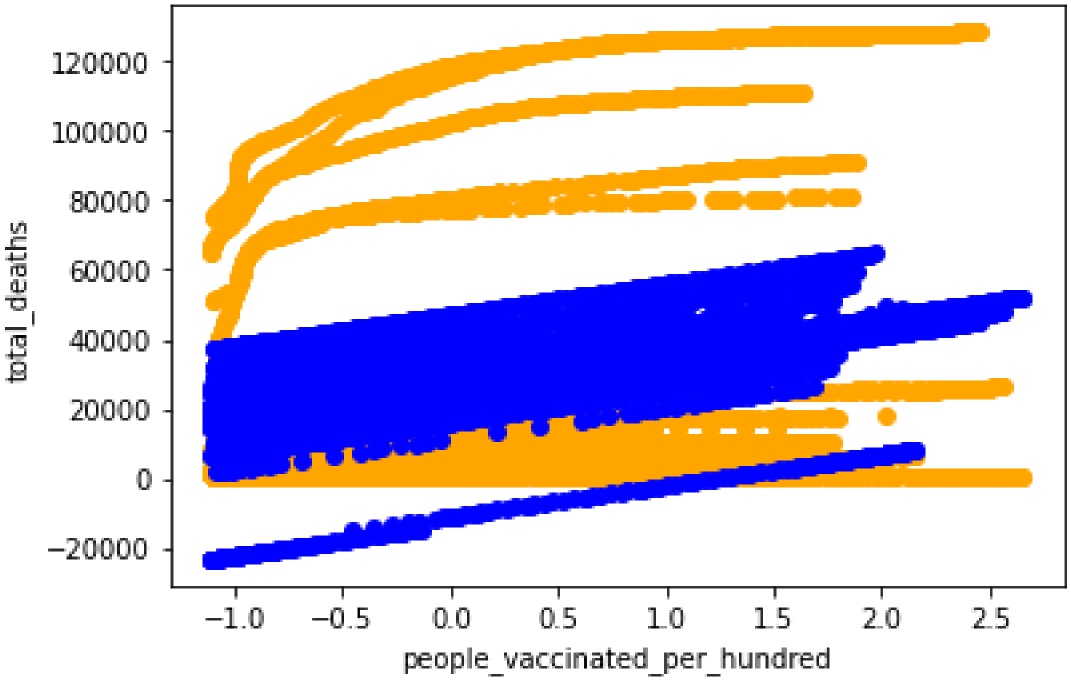
```
R^2 Score:  0.9441351025144288
Adjusted R^2 Score:  0.9440936904332682
Beta Coefficient Values:  [[27166.65169094]
 [34759.68711611]
 [ 3479.71524753]]


Feature(s) Used:  ('total_cases', 'icu_patients')
```

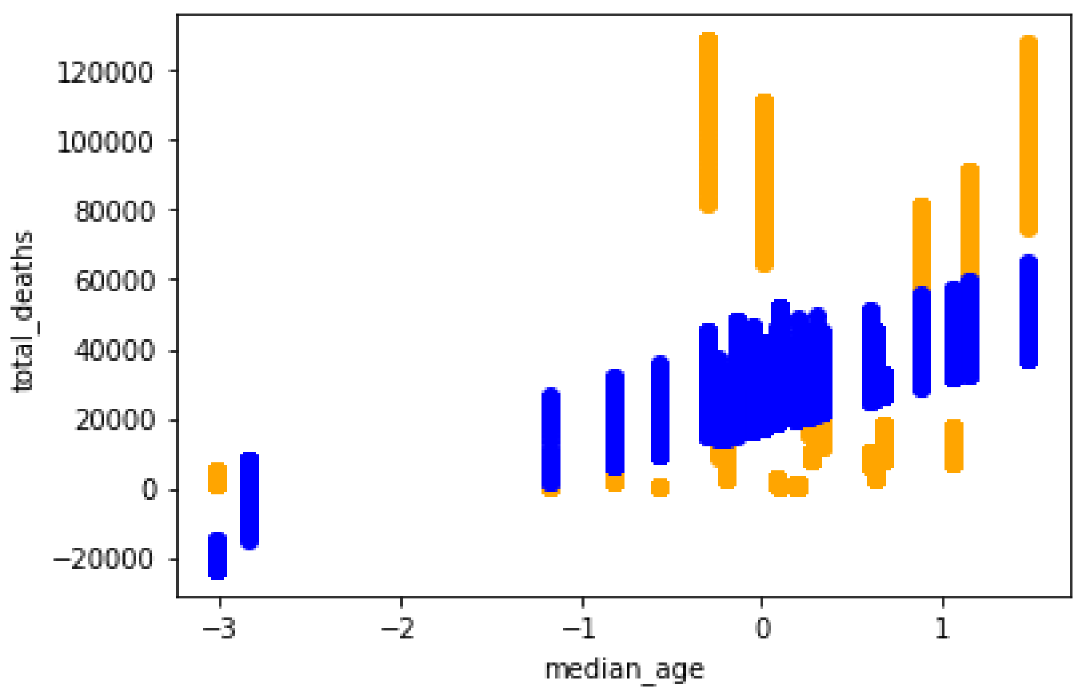




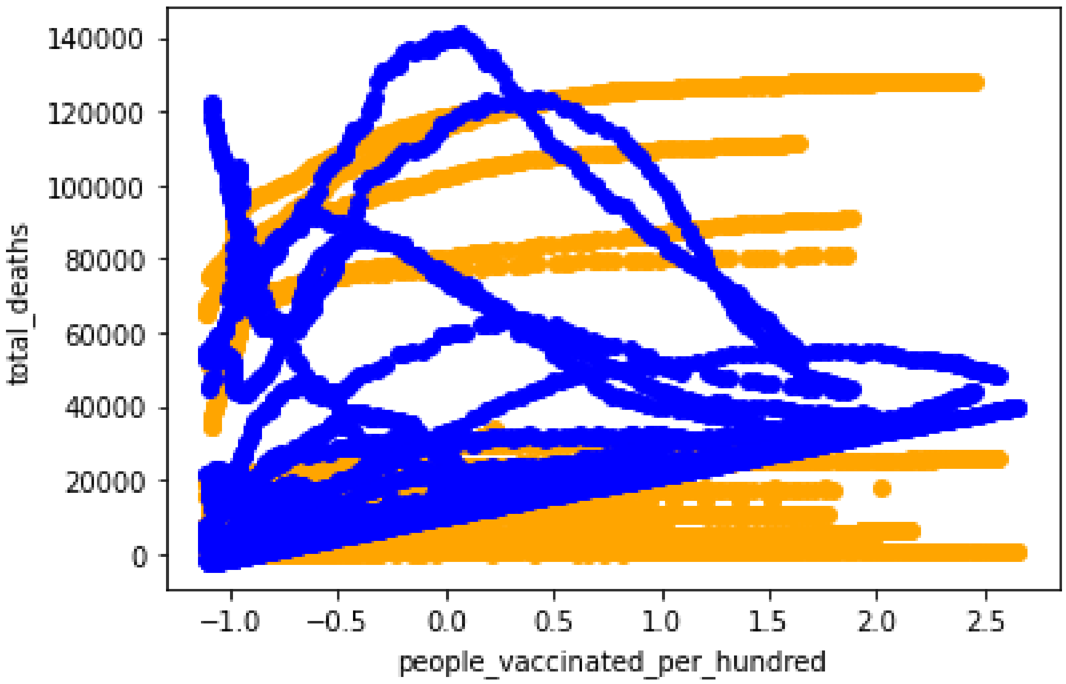
```
R^2 Score:  0.9367770623540452
Adjusted R^2 Score:  0.9367301958324397
```

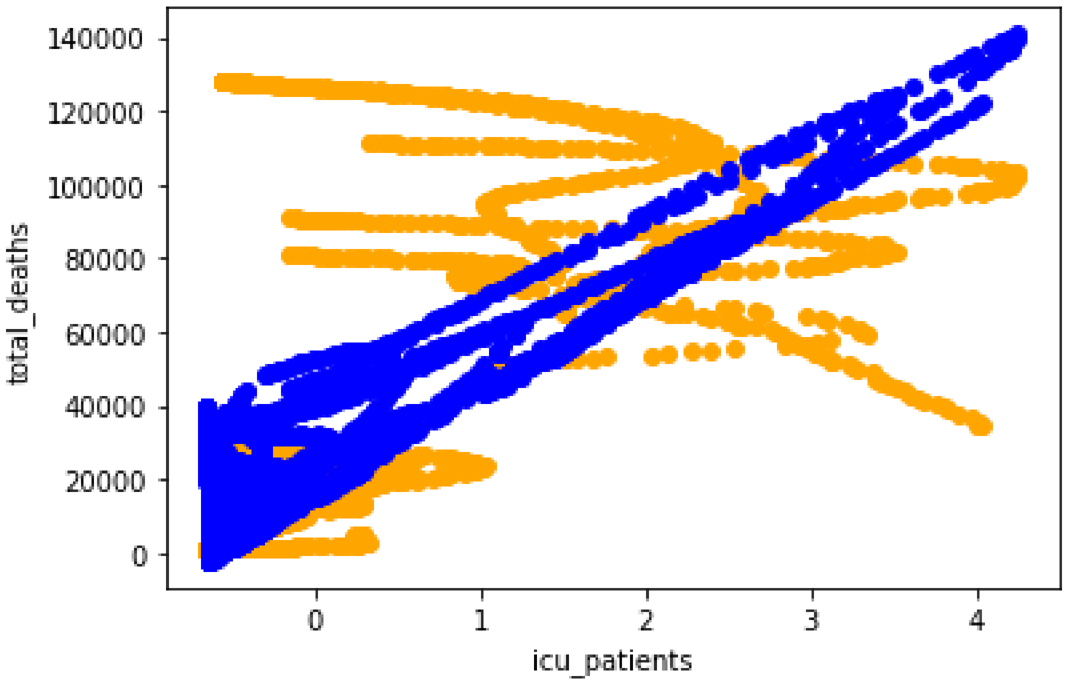
```
Beta Coefficient Values:  [[27166.65169094]
 [36810.98352259]
 [-1708.69726686]]


Feature(s) Used:  ('people_vaccinated_per_hundred', 'median_age')
```





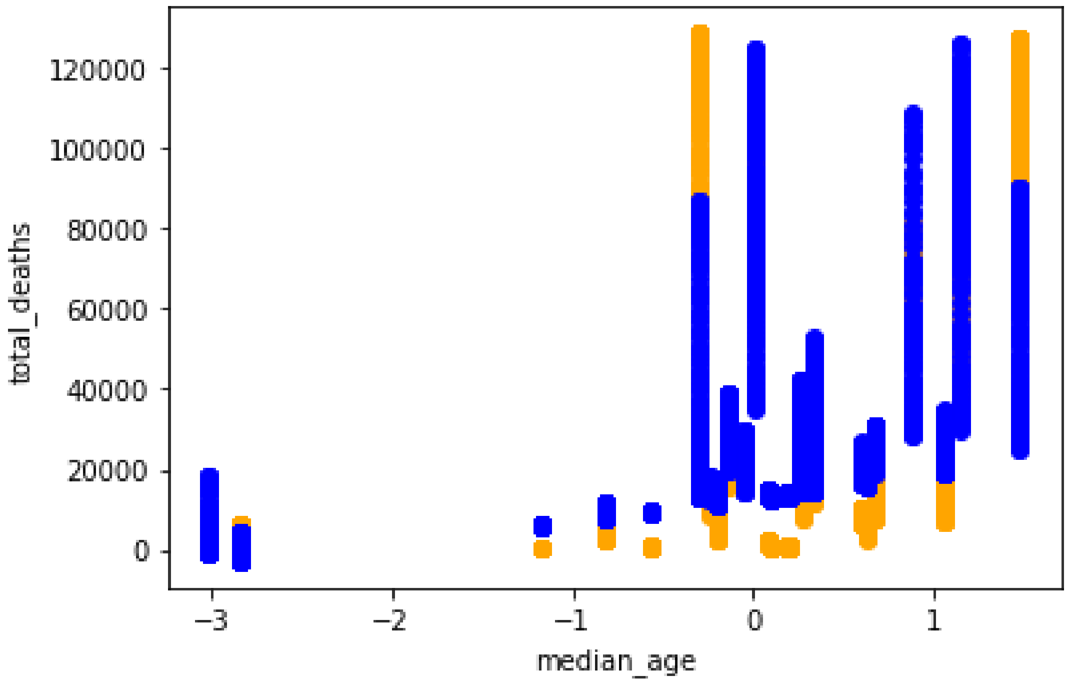
```
R^2 Score:  0.16031374692504432
Adjusted R^2 Score:  0.15969129603321597
Beta Coefficient Values:  [[27166.65169094]
 [ 8925.87390186]
 [13484.0921234 ]]


Feature(s) Used:  ('people_vaccinated_per_hundred', 'icu_patients')
```

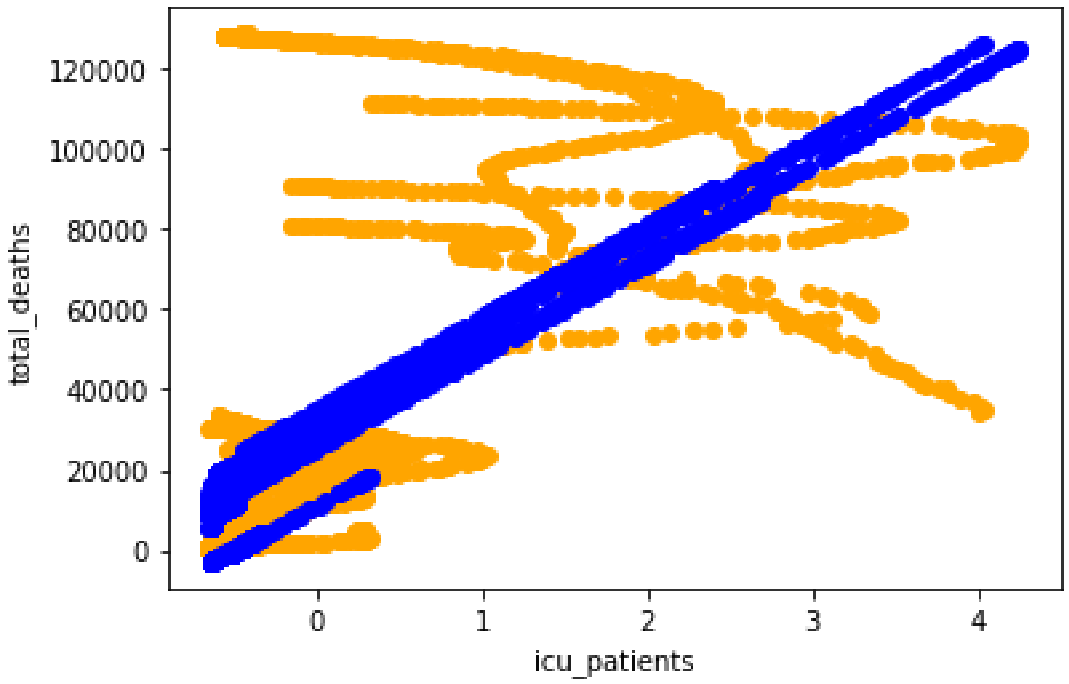




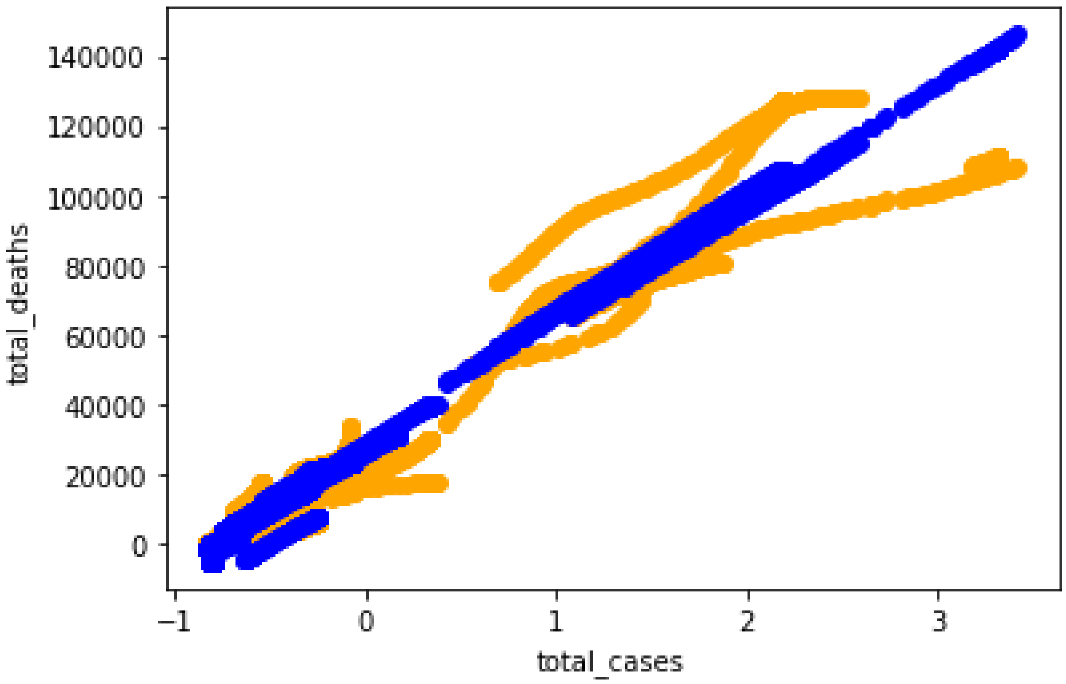
```
R^2 Score:  0.534129885639375
Adjusted R^2 Score:  0.533784540854823
```

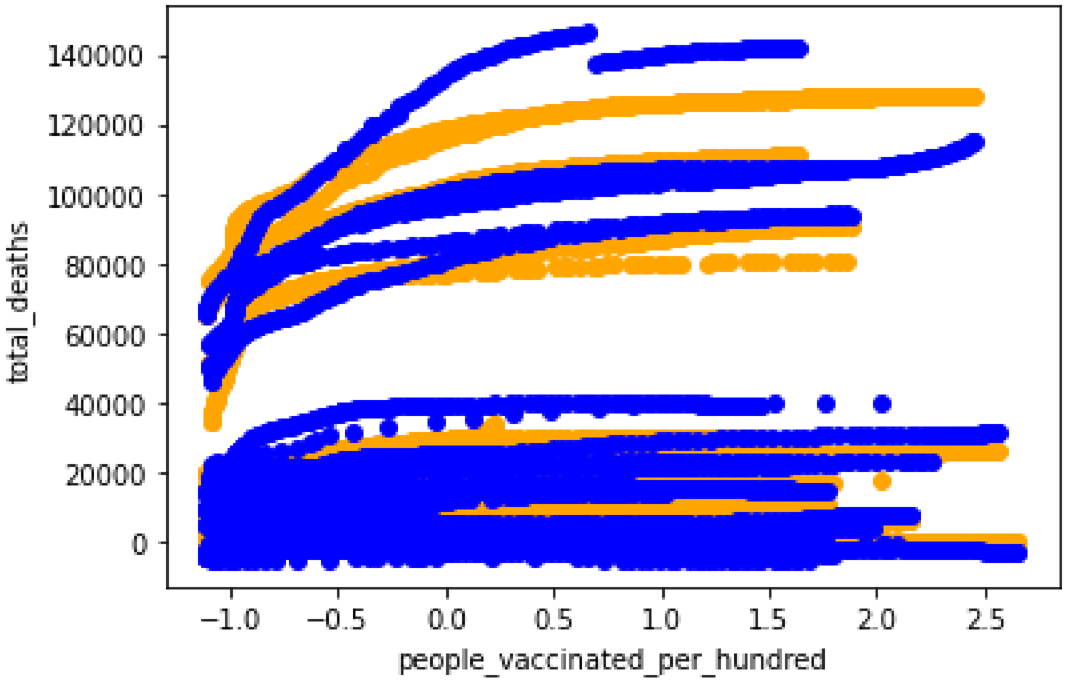
```
Beta Coefficient Values:  [[27166.65169094]
 [11384.21476983]
 [26576.31020694]]


Feature(s) Used:  ('median_age', 'icu_patients')
```

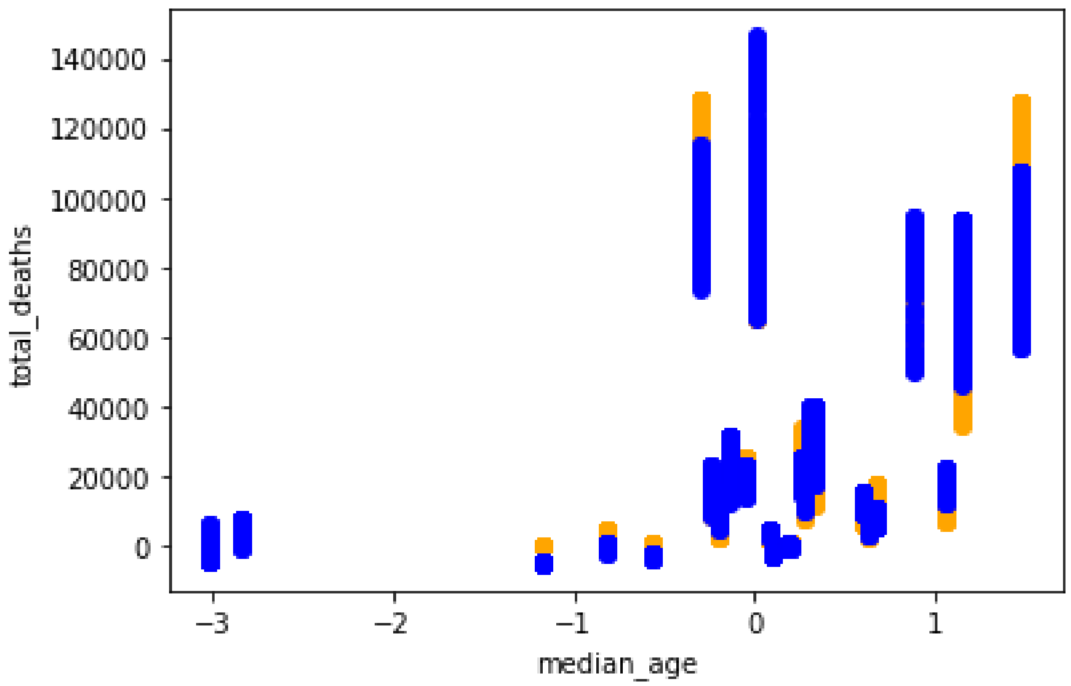




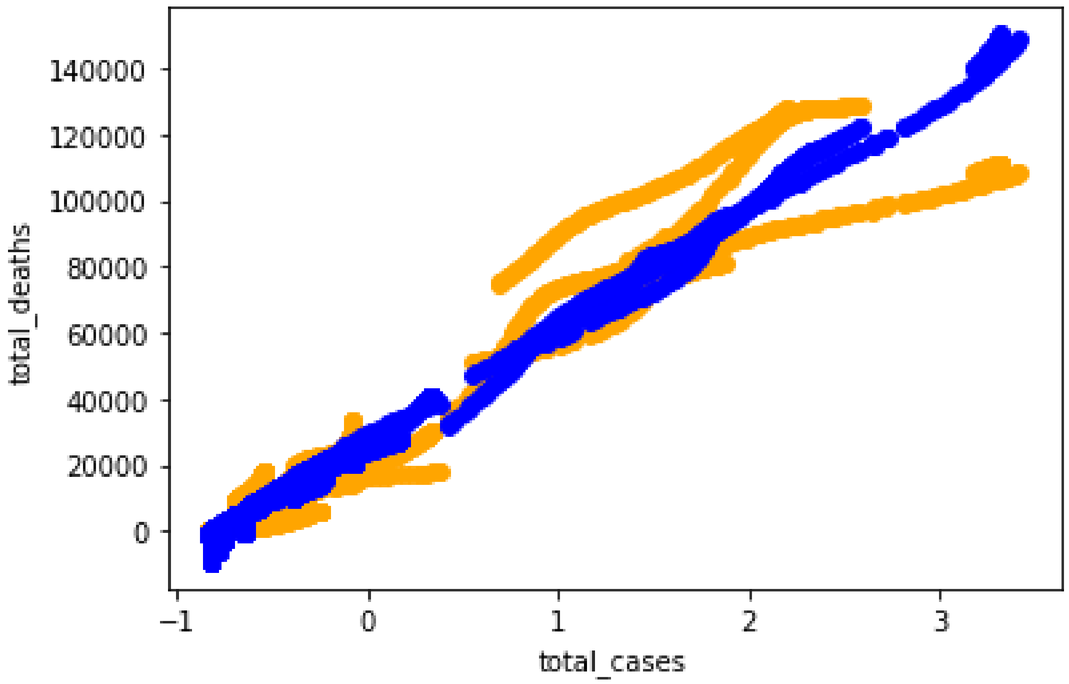
```
R^2 Score:  0.46149506582043454
Adjusted R^2 Score:  0.4610958775816061
Beta Coefficient Values:  [[27166.65169094]
 [ 5389.5766109 ]
 [22943.10708842]]


Feature(s) Used:  ('total_cases', 'people_vaccinated_per_hundred', 'median_age')
```

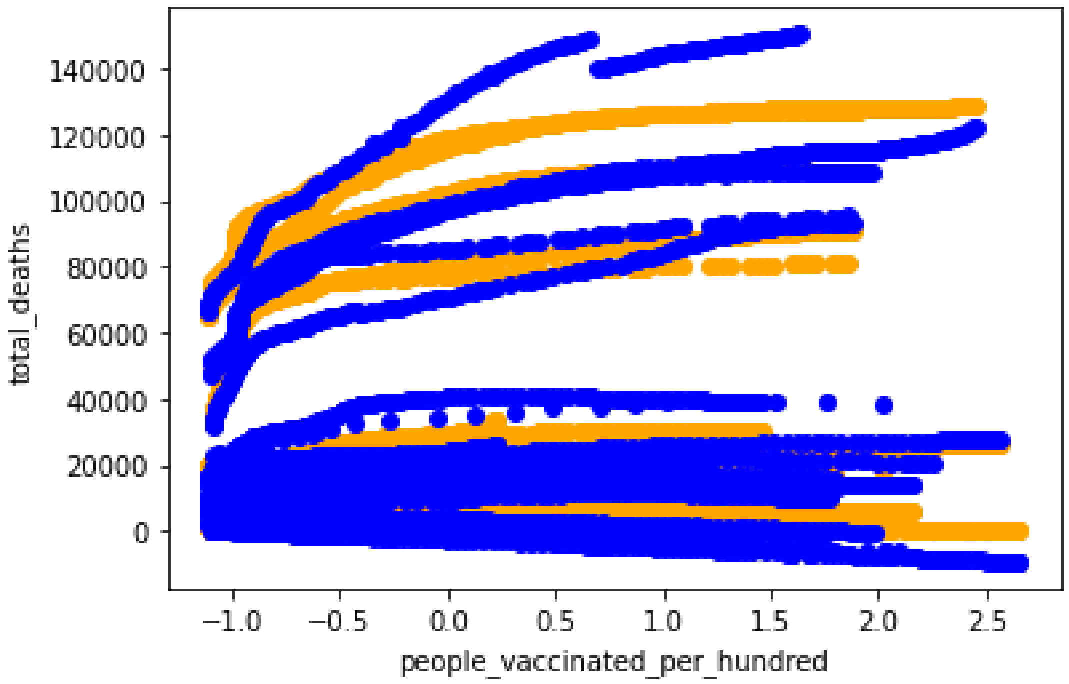

140000
120000
100000
80000
60000
40000
20000
0
total_deaths
−1
0
1
2
3
total_cases

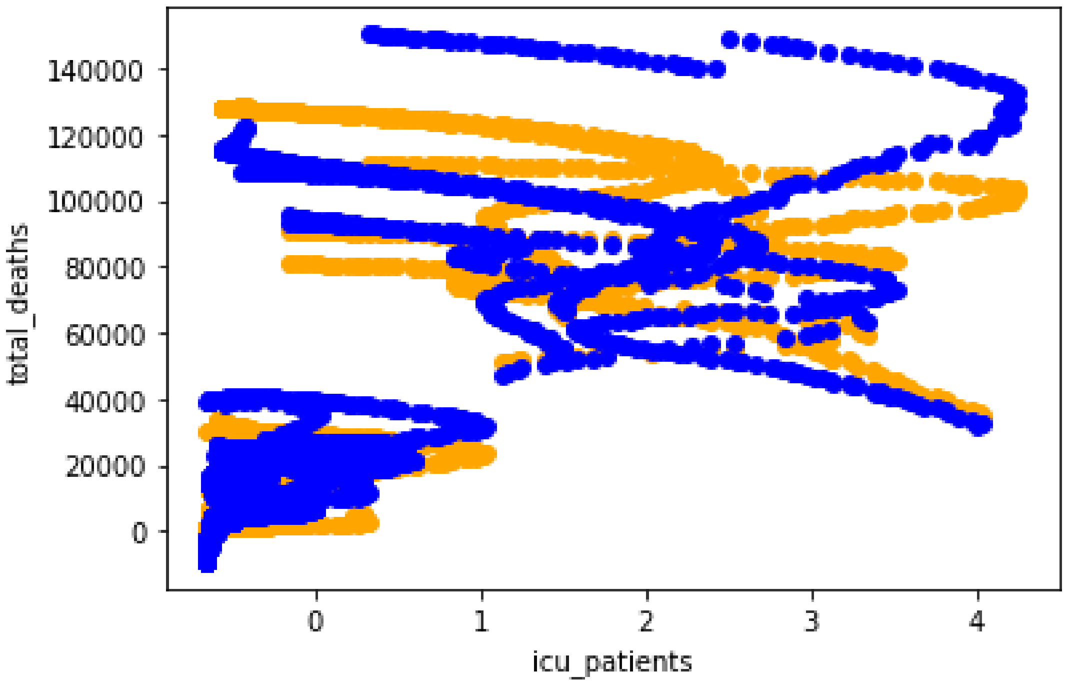

140000
120000
100000
80000
60000
40000
20000
0
total_deaths
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
people_vaccinated_per_hundred



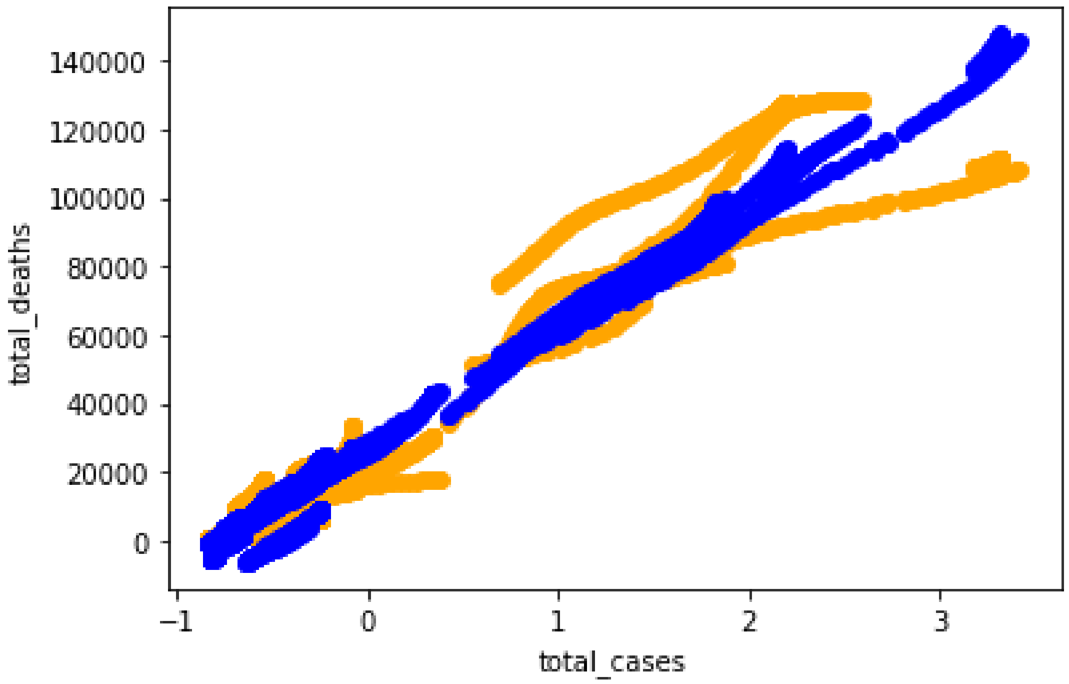
```
R^2 Score:  0.9443199623903851
Adjusted R^2 Score:  0.9442649153932426
Beta Coefficient Values:  [[27166.65169094]
 [34906.69793814]
 [ -529.36486602]
 [ 3346.72182702]]


Feature(s) Used:  ('total_cases', 'people_vaccinated_per_hundred', 'icu_patients')
```

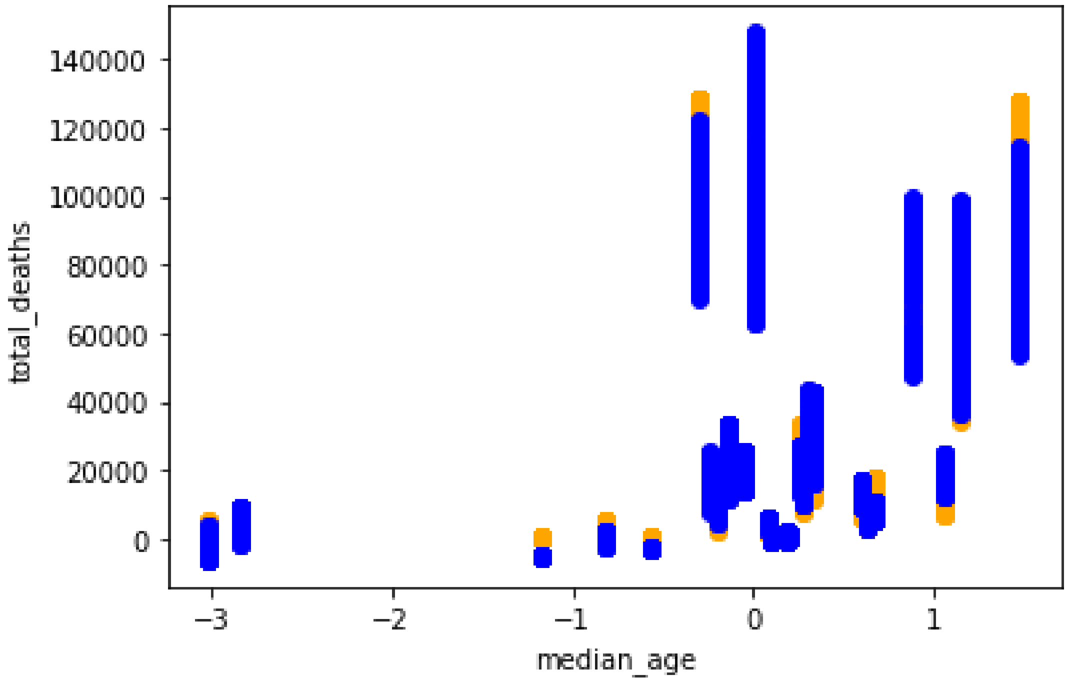

140000
120000
100000
80000
60000
40000
20000
0
total_deaths
−1
0
1
2
3
total_cases

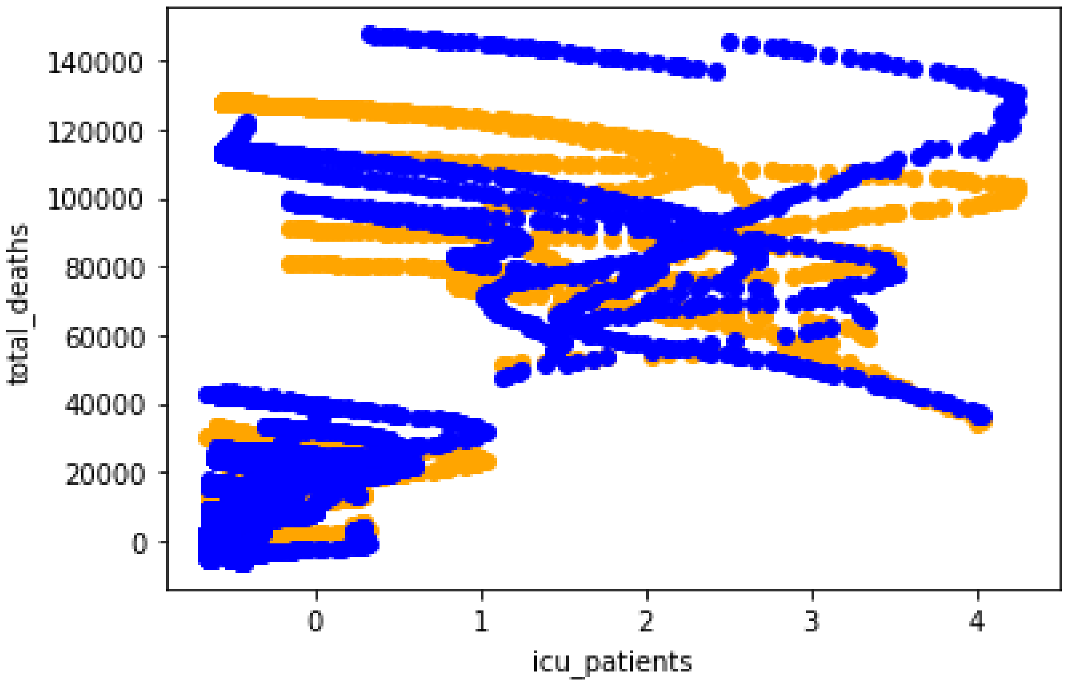

140000
120000
100000
80000
60000
40000
20000
0
total_deaths
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
people_vaccinated_per_hundred



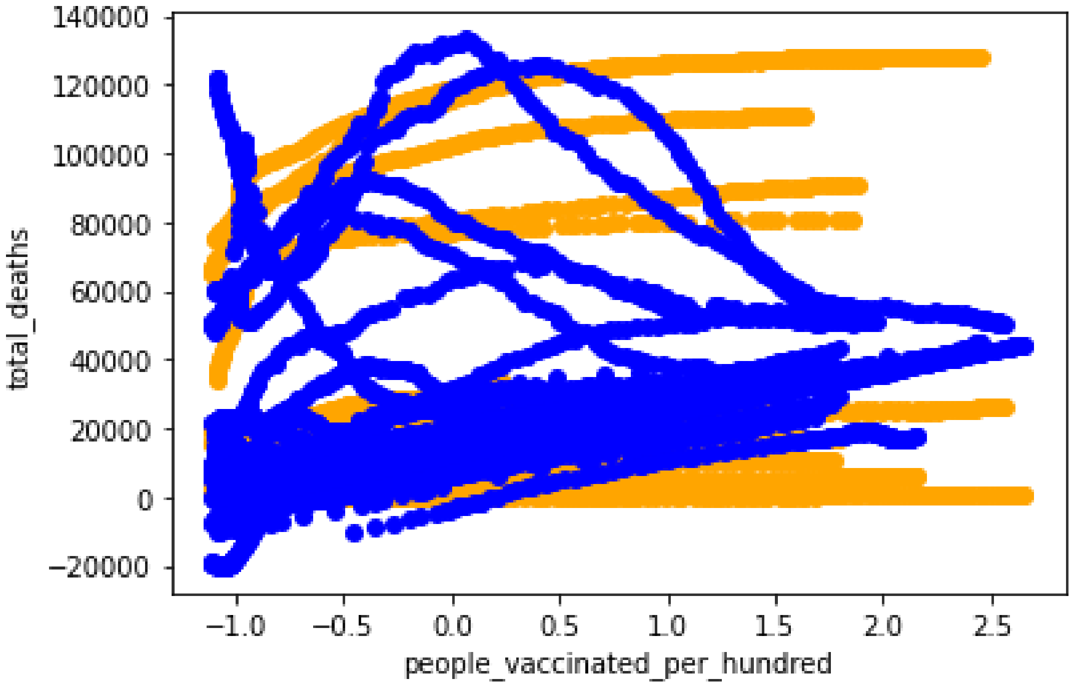
```
R^2 Score:  0.9404045368184565
Adjusted R^2 Score:  0.9403456189112083
Beta Coefficient Values:  [[27166.65169094]
 [38658.82986792]
 [-2614.99607721]
 [-3508.83363267]]


Feature(s) Used:  ('total_cases', 'median_age', 'icu_patients')
```

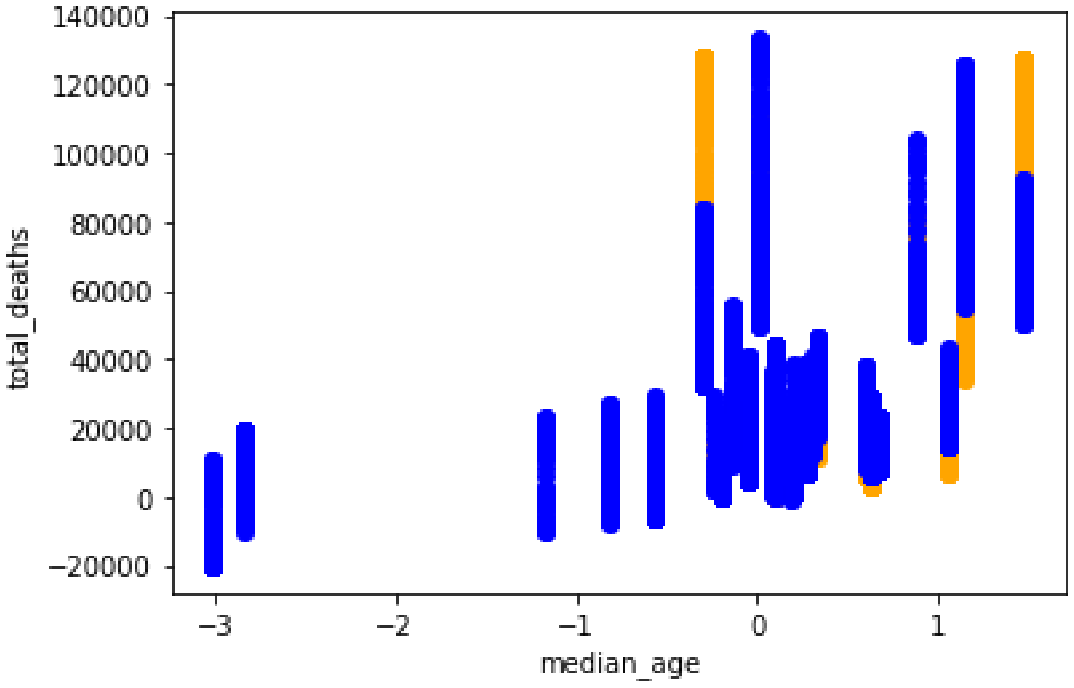

140000
120000
100000
80000
60000
40000
20000
0
total_deaths
−1
0
1
2
3
total_cases

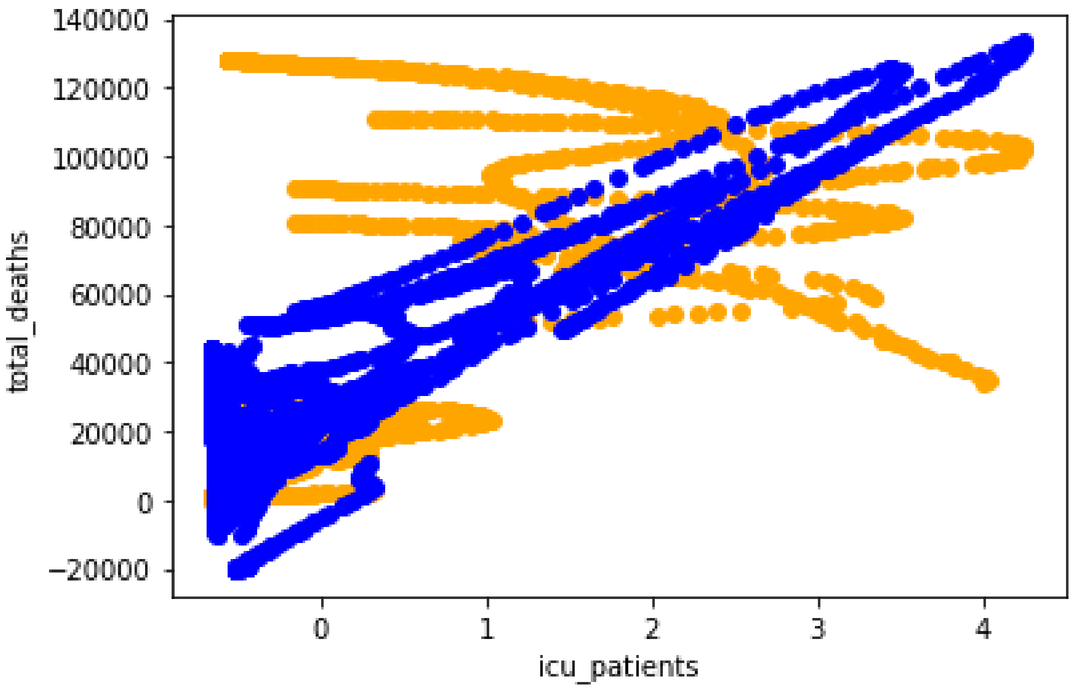

140000
120000
100000
80000
60000
40000
20000
0
total_deaths
−3
−2
−1
0
1
median_age



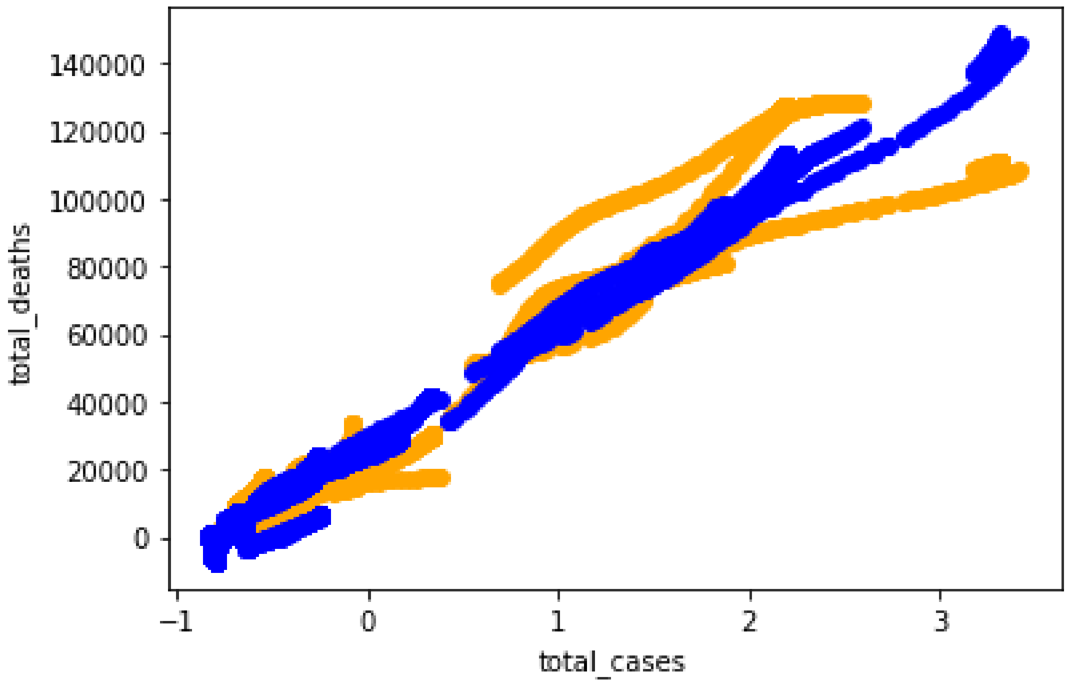
```
R^2 Score:  0.9464482014107781
Adjusted R^2 Score:  0.9463952584561669
Beta Coefficient Values:  [[27166.65169094]
 [36500.4070933 ]
 [ 3777.72965452]
 [-2549.64650923]]


Feature(s) Used:  ('people_vaccinated_per_hundred', 'median_age', 'icu_patients')
```

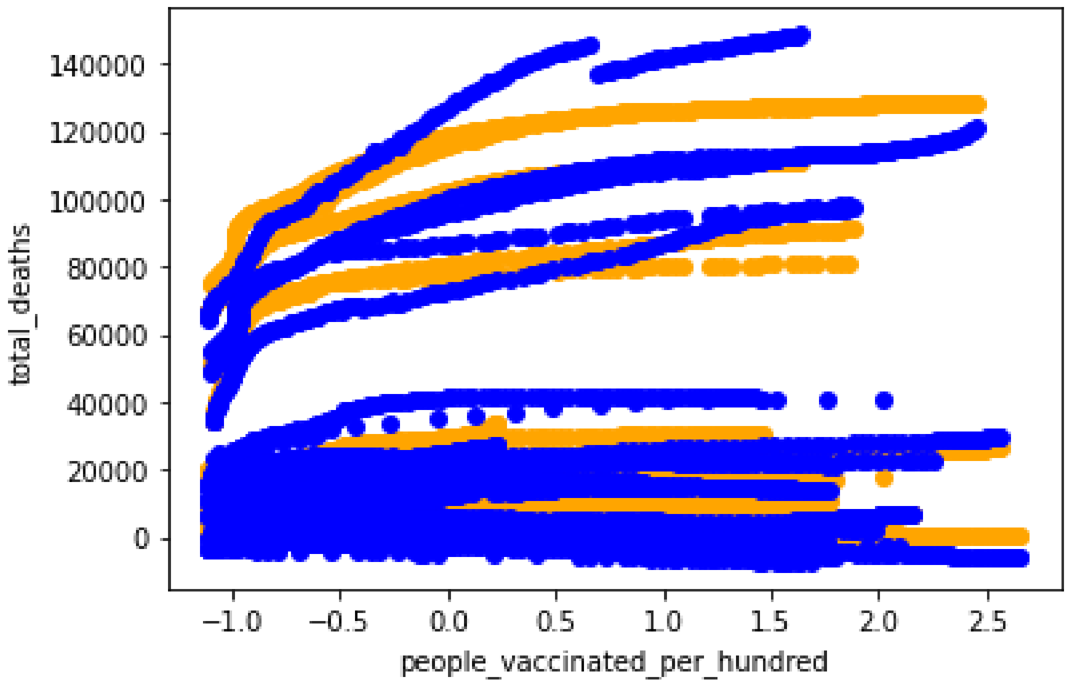

140000
120000
100000
80000
60000
40000
20000
0
−20000
total_deaths
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
people_vaccinated_per_hundred

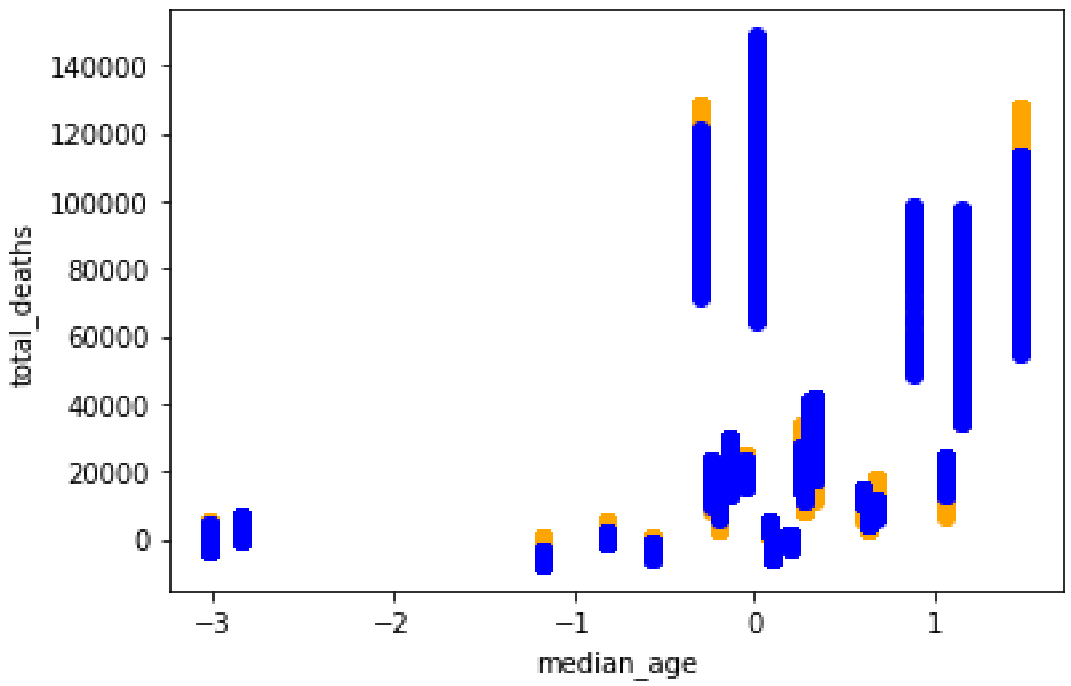

140000
120000
100000
80000
60000
40000
20000
0
−20000
total_deaths
−3
−2
−1
0
1
median_age

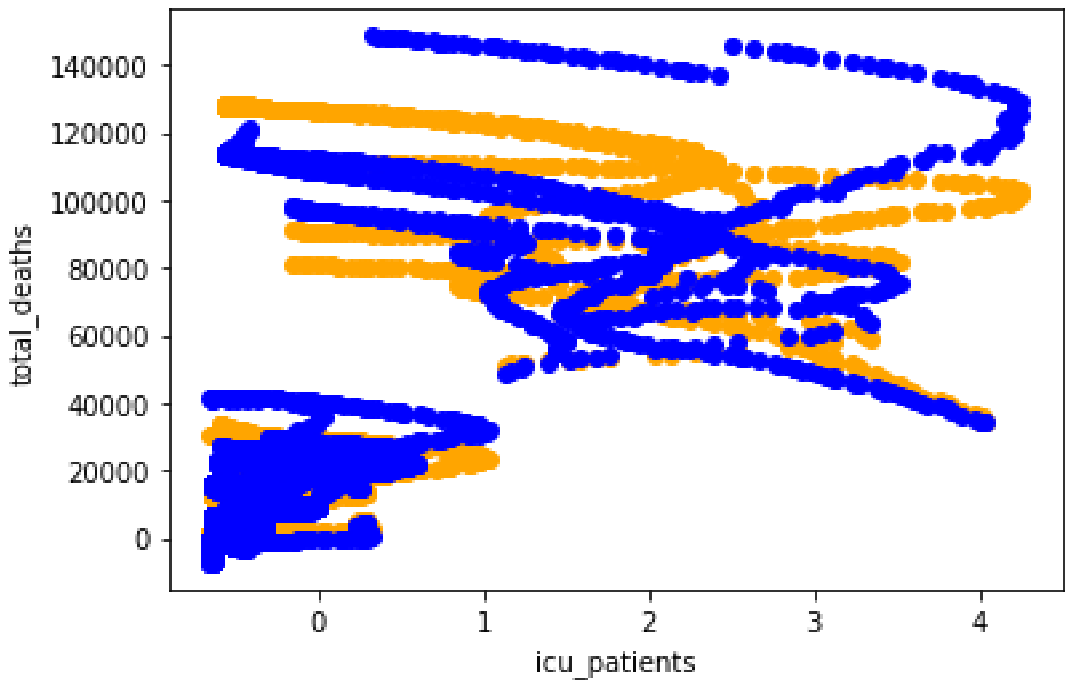


```
R^2 Score:  0.5683490715671139
Adjusted R^2 Score:  0.5679223281875461
Beta Coefficient Values:  [[27166.65169094]
 [12364.92373893]
 [ 7163.82710585]
 [24745.44107178]]


Feature(s) Used:  ('total_cases', 'people_vaccinated_per_hundred', 'median_age', 'icu_patients
```


total_deaths
total_cases
140000
120000
100000
80000
60000
40000
20000
0
−1
0
1
2
3


total_deaths
people_vaccinated_per_hundred
140000
120000
100000
80000
60000
40000
20000
0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5





```
R^2 Score:  0.9481758023708868
Adjusted R^2 Score:  0.9481117427940894
```

```
Beta Coefficient Values:  [[27166.65169094]
 [37827.96947513]
 [-1841.15436338]
 [ 3454.91653116]
 [-3745.21773351]]
```

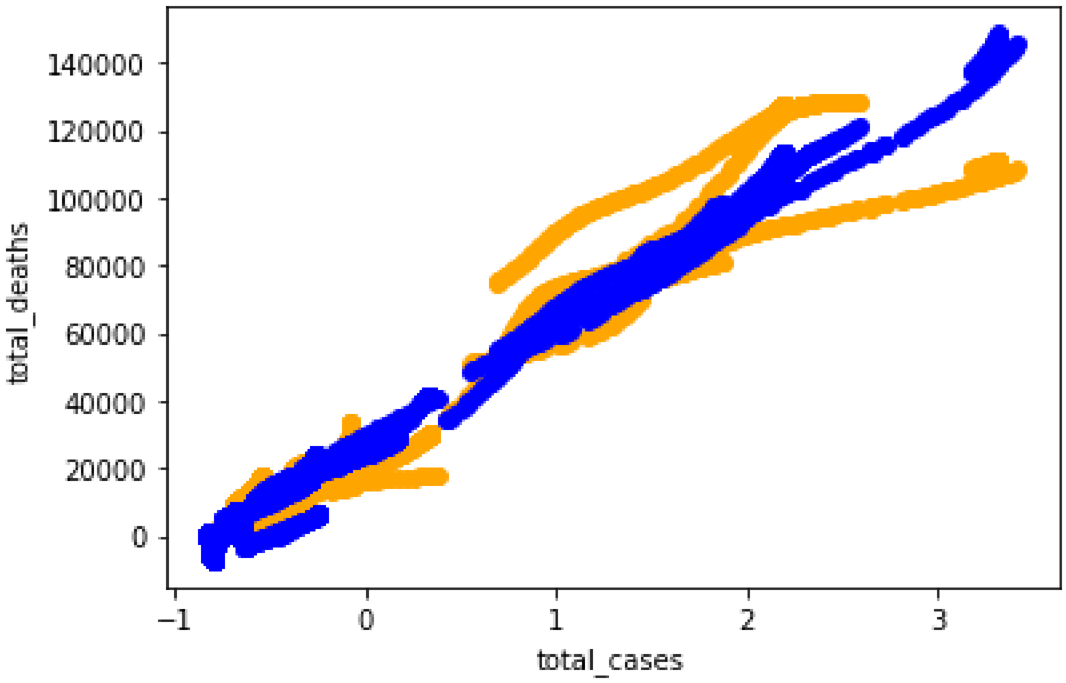
Features	Excel	Python
Total Cases	0.935617957[illegible]	[illegible]2786678108329
People Vaccinated Per Hundred	0.028489294[illegible]	[illegible]4966258012486
Median Age	0.101777147[illegible]	[illegible]3820661011531
ICU Patients	0.437311576[illegible]	[illegible]6298467369194
Total Cases + People Vaccinated Per Hundred	0.937011788[illegible]	[illegible]1666746037052
Total Cases + Median Age	0.944023629[illegible]	[illegible]0936904332682
Total Cases + ICU Patients	0.936810914[illegible]	[illegible]301958324397
People Vaccinated Per Hundred + Median Age	0.156397137[illegible]	[illegible]912960332163
People Vaccinated Per Hundred + ICU Patients	0.528784820[illegible]	[illegible]845408548232

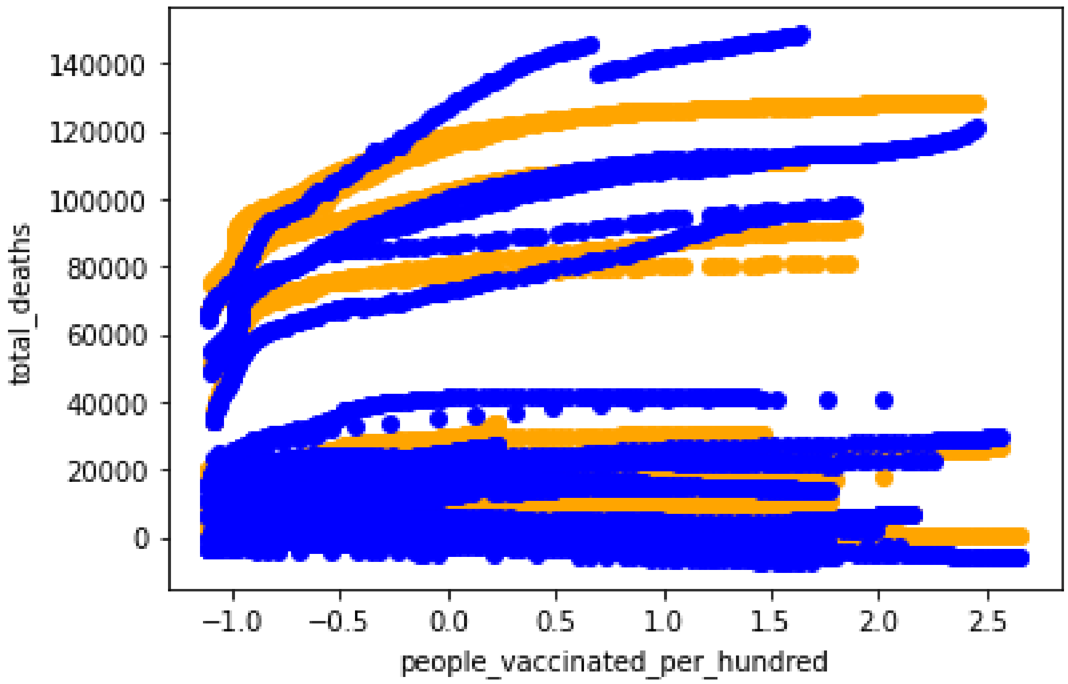
Features	Excel	Python
Median Age + ICU Patients	0.4568586707246802958775816063	
Total Cases + People Vaccinated Per Hundred + Median Age	0.944229522709482649153932426	
Total Cases + People Vaccinated Per Hundred + ICU Patients	0.946521938649403456189112084	
Total Cases + Median Age + ICU Patients	0.940815740159463952584561669	
People Vaccinated Per Hundred + Median Age + ICU Patients	0.563373545715679223281875463	
Total Cases + People Vaccinated Per Hundred + Median Age + ICU Patients	0.94848018379481117427940894	

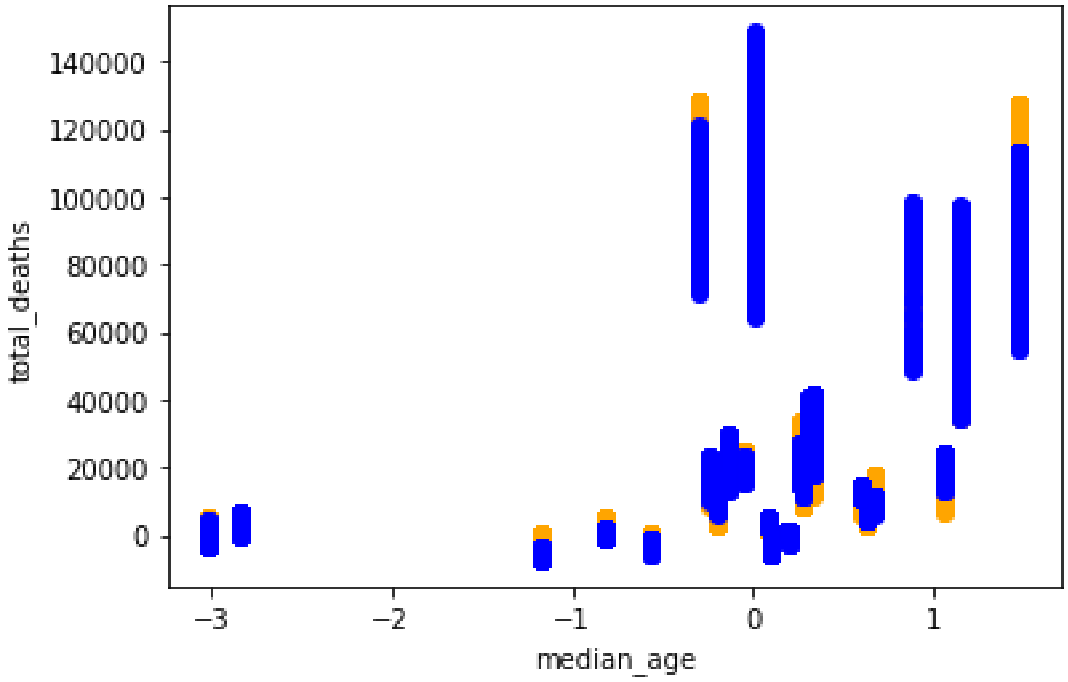
As can be seen above, the Adjusted R2 values for Excel and Python are quite similar to each other, and the same for 3 significant figures minimally. This goes to show that our model is very accurate.

Upon further analysis, we note that the best Adjusted R2value comes from the last regression model with all 4 features being implemented. As such, we decided to isolate it and compare the beta coefficient values.

```
In [107]: df_features, df_target = get_features_targets(df_task_1, feature_combis[-1], target)
          beta = multiple_linear_regression(df_features, df_target)
          print(beta)
```




140000
120000
100000
80000
60000
40000
20000
0
total_deaths
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
people_vaccinated_per_hundred


140000
120000
100000
80000
60000
40000
20000
0
total_deaths
−3
−2
−1
0
1
median_age