SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

01.020 Design Thinking Project III
10.022 Modelling Uncertainty – Task 1
SC02 Team 1

Name	**Student ID**	**Contribution**
Chua Min Pei	1005340	Each team member did an equal amount of work.
Sim Yu Hui, Kellie	1004204	
Ryan Kaw Zheng Da	1005144	
Eunice Kwok Xiu Yi	1005469	
Ng Zhen An	1005527	

1. Task 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) (https://ourworldindata.org/coronavirus) since it updates information on a regular daily basis from various sources. We will elaborate on the chosen predictor variables later in the report.

1.2 Import Libraries

For our code, we made use of Python libraries such as pandas, numpy, matplotlib, seaborn and itertools.

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 (https://github.com/owid/covid-19-data/blob/master/public/data/README.md).

In [44]:

```
# Import dataset
file_url = 'https://covid.ourworldindata.org/data/owid-covid-data.csv'
df = pd.read_csv(file_url)
```

In [45]:

```
df_first_world_countries = df[df['human_development_index'] >= 0.8]
features = ['total_cases', 'people_vaccinated_per_hundred', 'median_age', 'icu_p
atients']
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 [46]:

```
unique_codes = list(set(df_task_1['iso_code'].unique()) - set(["USA"]))
df_task_1 = df_task_1[df_task_1['iso_code'].isin(unique_codes)]
df_task_1.to_csv("2d-task-1-v2.csv")
```

1.4 Multiple Linear Regression Model

1.4.1 Visualisation and Plots

Based on our visualisation, 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. The correlation heatmap can be found in the Appendix 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

1.4.2 Helper Functions and Code for Model

We defined functions that were utilised in the model, taken from the cohort lessons and pre-class material (https://github.com/Data-Driven-World/d2w_notes/blob/master/Multiple_Linear_Regression.ipynb). The code for this can be found in the Appendix Section.

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

1.4.3.1 Accuracy Metrics

As mentioned in the task brief, R^2 is not a good metric to be used for Multiple Linear Regression. As such, we decided to take into account the metric Adjusted R^2, which will show us if the variables used were potentially overfitting our model.

The Adjusted R^2 Score is an improved version of R^2 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].

1.4.3.2 P-Value

P-value is a measure of the probability that an observed difference could have occurred just by random chance [3]. It is used as an alternative to rejection points to provide the smallest level of significance at which the null hypothesis would be rejected [3]. The level of statistical significance is often expressed as a p-value between 0 and 1 [4]. The smaller the p-value, the stronger the evidence that you should reject the null hypothesis. A p-value less than 0.05 is considered statistically significant [4].

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.

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.

In [34]:

```
# 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 [51]:

```
# Graphs obtained from 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)
```

Features	Excel	Python
Total Cases	0.935617957333392	0.9356786678108329
People Vaccinated Per Hundred	0.0284892944313471	0.030134966258012486
Median Age	0.101777147806489	0.10303820661011531
ICU Patients	0.437311576454327	0.44146298467369194
Total Cases + People Vaccinated Per Hundred	0.937011788047751	0.9369666746037052
Total Cases + Median Age	0.94402362989154	0.9440936904332682
Total Cases + ICU Patients	0.936810914732299	0.9367301958324397
People Vaccinated Per Hundred + Median Age	0.156397137118895	0.1596912960332163
People Vaccinated Per Hundred + ICU Patients	0.528784820283488	0.5337845408548232
Median Age + ICU Patients	0.456858670728682	0.4610958775816063
Total Cases + People Vaccinated Per Hundred + Median Age	0.94422952270268	0.9442649153932426
Total Cases + People Vaccinated Per Hundred + ICU Patients	0.946521938647434	0.9403456189112084
Total Cases + Median Age + ICU Patients	0.940815740156368	0.9463952584561669
People Vaccinated Per Hundred + Median Age + ICU Patients	0.563373545710692	0.5679223281875463
Total Cases + People Vaccinated Per Hundred + Median Age + ICU Patients	0.9484801837915	0.9481117427940894

As can be seen above, the Adjusted R^2 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 R^2 value 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 [36]:

```
df_features, df_target = get_features_targets(df_task_1, feature_combis[-1], tar
get)
beta = multiple_linear_regression(df_features, df_target)
print(beta)
```

```
Adjusted R^2 Score:  0.9481117427940894
[[27166.65169094]
 [37827.96947513]
 [-1841.15436338]
 [ 3454.91653116]
 [-3745.21773351]]
```

Analysing the beta coefficient values, we note that the values are quite different from that of Excel. We hypothesise that this is due to the fact that we performed Z-normalization on our features before running the Multiple Linear Regression Model in Python. Normalization is generally required when we are dealing with attributes on a different scale, otherwise, it may lead to a dilution in effectiveness of an important equally important attribute (on lower scale) because of other attribute having values on larger scale [5]. Z-normalization was used in our code to scale the data of an attribute so that it falls in a smaller range (± 3 standard deviations). As a result, the beta coefficients will be greater in the Python code as compared to the one in Excel as the actual data was scaled down in Python.

Coefficient	Excel	Python
Intercept (C / β_0)	-35569.2134759228	27166.65169094
β_1	0.0270689864149344	337827.96947513
β_2	-107.232691163685	-1841.15436338
β_3	850.951553552295	3454.91653116
β_4	-3.21689543088435	-3745.21773351

As a proof of concept, given the beta coefficient values above, we decided to plot the graphs using the equation $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4$ in both Excel and Python. The graphs can be found in the Appendix below.

1.5 Appendix

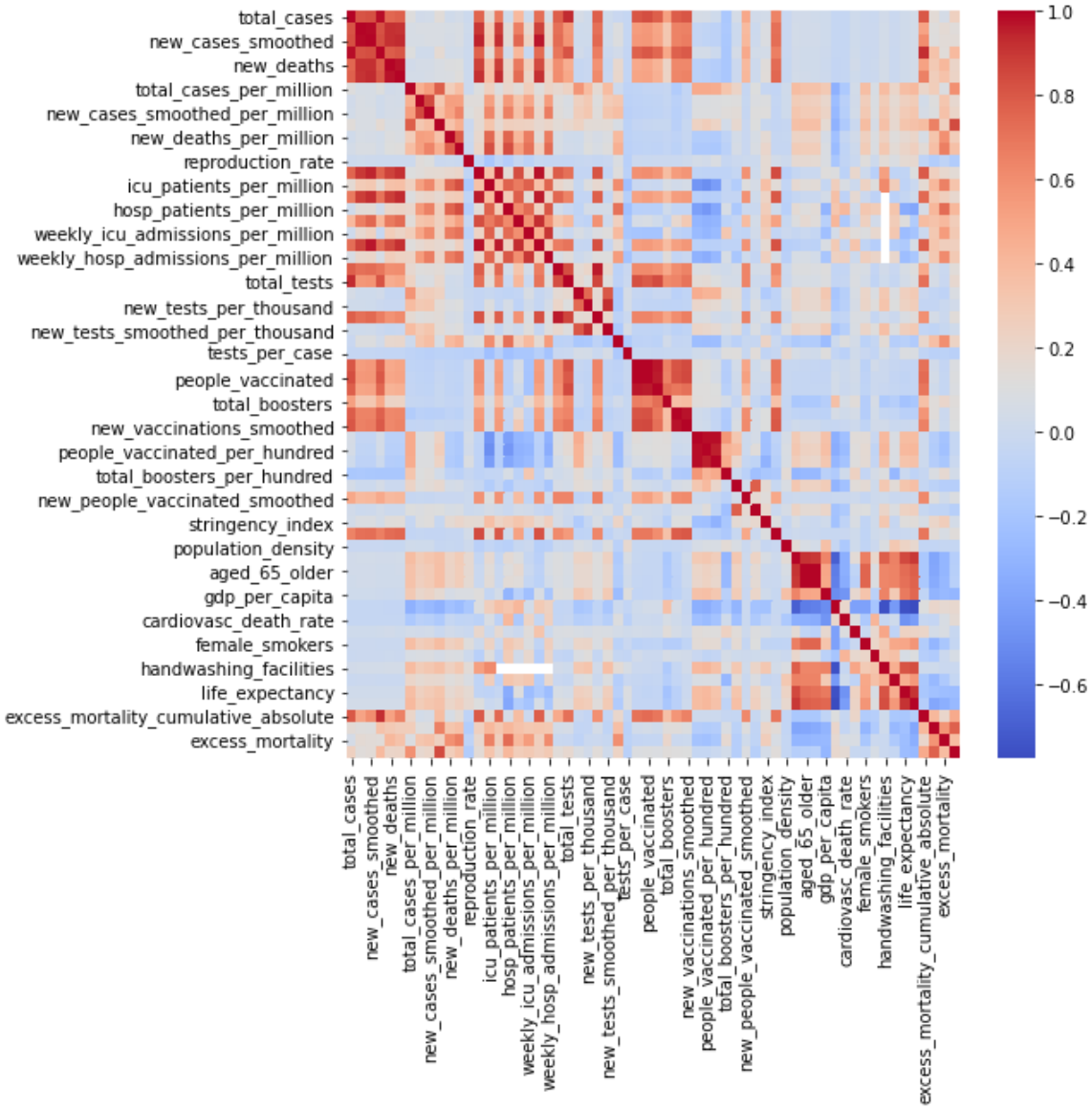
1.5.1 Correlation Heatmaps

In [21]:

```
def determine_correlation(df, column_name):
  """Takes a DataFrame and a column name, return a DataFrame containing the corr
elation score between the column that matches the given name and all other colum
ns."""
  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=Fal
se)[1:]

def plot_correlation_heatmaps(df, figsize, annot=False):
  """Takes a DataFrame, the figsize of the heatmap to be plotted, and whether th
e heatmap should be annotated (default False), returns nothing."""
  correlations = df.corr()
  plt.figure(figsize=figsize)
  sns.heatmap(correlations, cmap="coolwarm", annot=annot)
  plt.show()
```

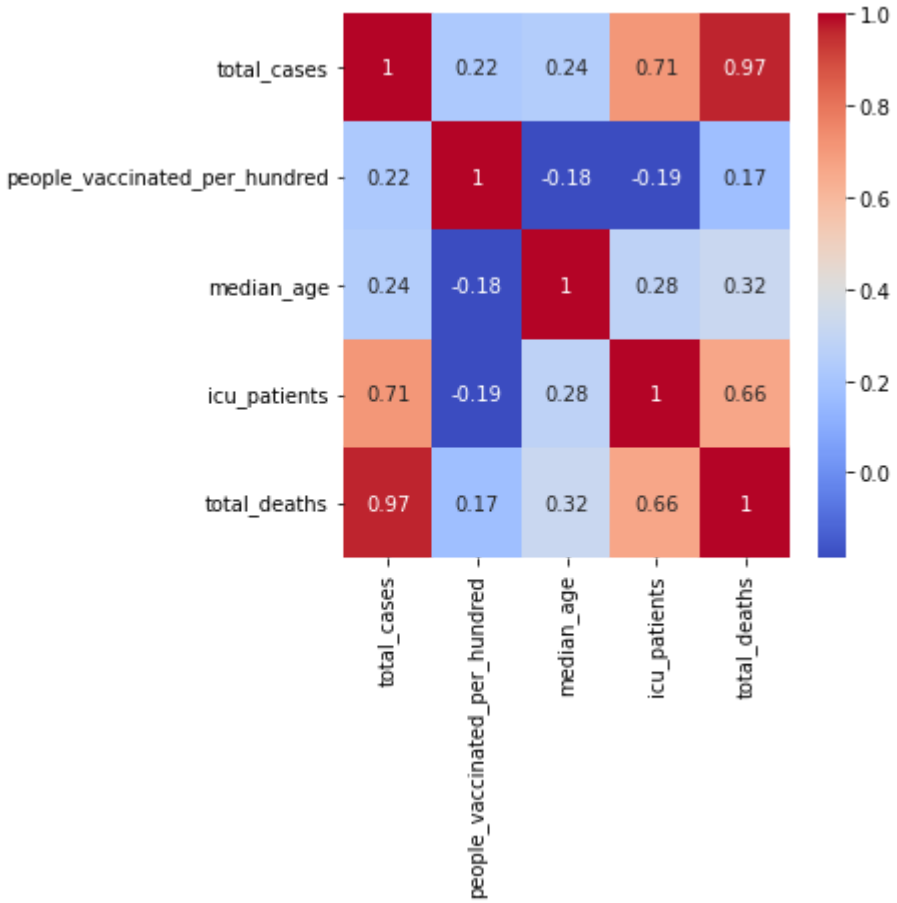
In [22]:

```
df_correlation = determine_correlation(df, "total_deaths")
plot_correlation_heatmaps(df, figsize=(5,5), annot=False)
```



In [54]:

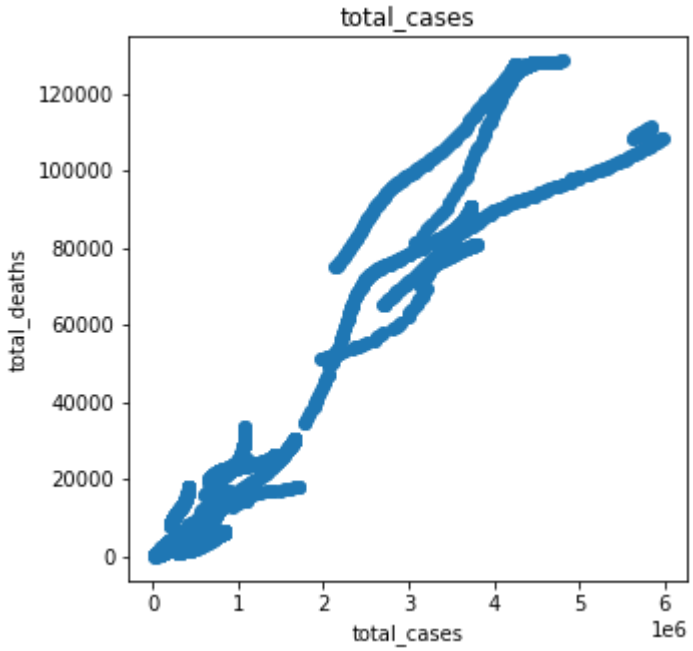
```
features = ['total_cases', 'people_vaccinated_per_hundred', 'median_age', 'icu_p
atients']
target = ['total_deaths']
columns = features + target
df_task_1 = df_task_1[columns]
plot_correlation_heatmaps(df_task_1, figsize=(5,5), annot=True)
```

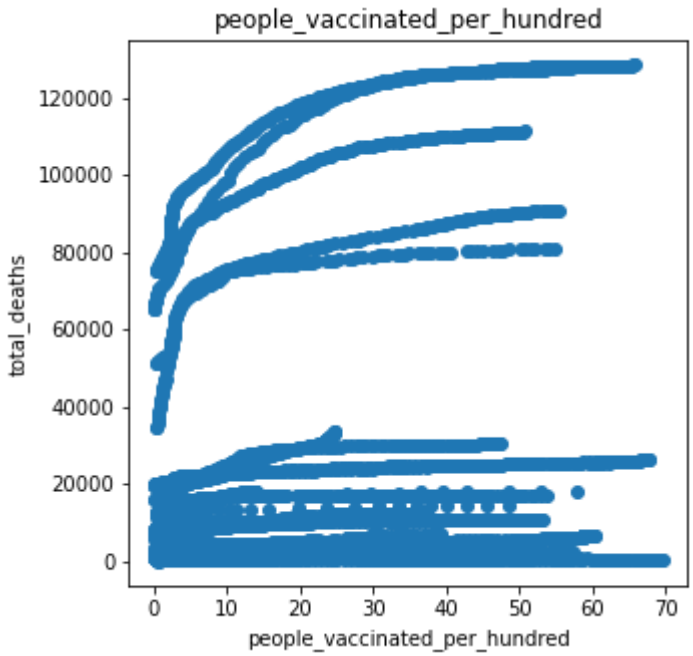


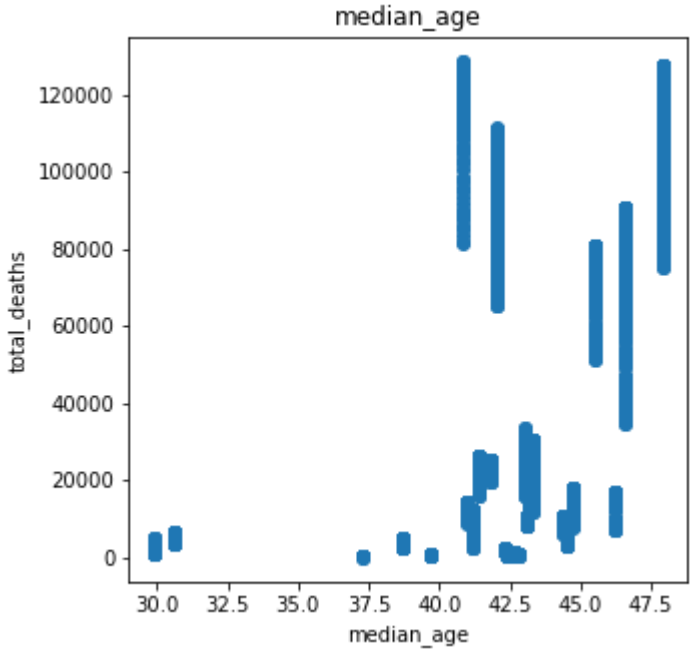
1.5.2 Plotting X against y

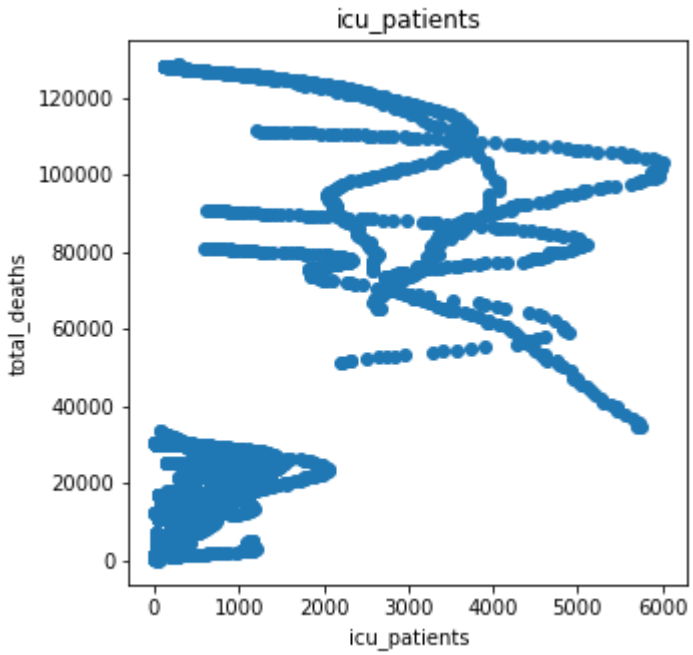
In [24]:

```
# 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()
```


total_cases
120000
100000
80000
60000
40000
20000
0
total_deaths
0
1
2
3
4
5
6
1e6
total_cases


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





1.5.3 Helper Functions

In [25]:

```
def normalize_z(df):
  """Takes a DataFrame, returns a DataFrame with normalized values using z-score
normalization."""
  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
normalization."""
  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 transformed column, returns a DataFrame with an additional column for the tr
ansformed data."""
  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 column
s for the target, returns a DataFrame containing the features and a DataFrame co
ntaining the target."""
  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, c
hange it to a column vector, and add a column of '1's in the first column, retur
ns a numpy.array containing the features."""
  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, cha
nge it to a column vector, returns a Numpy array containing the target."""
  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 valu
es after z-normalization and conversion to a Numpy array."""
  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
 equation after standardization and adding of column for constant 1."""
  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 targe
t, the seed used to split randomly, and the fraction used to split randomly (def
```

```
ault 0.5), returns a tuple of 4 DataFrames containing the train and test sets fo
r the features and target DataFrames."""
  # 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 [26]:

```
def compute_cost(X, y, beta):
  """Takes a Numpy array containing the features, a Numpy array containing the t
arget, and beta coefficients at the end of the iteration, returns an array of co
mputed cost function values."""
  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 t
arget, an array of beta values, the learning rate, the number of iterations to p
erform, returns the beta coefficient at the end of the iteration, and an array s
toring the cost value at each iteration."""
  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 [49]:

```
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 function
s
  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)

  # Calculate Adjusted R^2
  print("Adjusted R^2 Score: ", adjusted_r2_score(r2_score(target, pred), X.shap
e[0], X.shape[1]))
  return beta
```

In [33]:

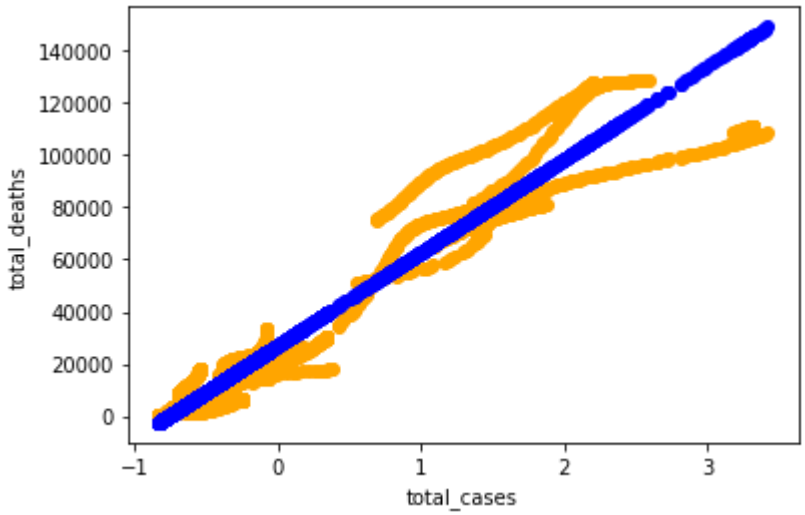
```
def r2_score(target, pred):
  """Takes a Numpy array containing the target and a Numpy array containing the
 predicted values, returns the r2 score."""
  diff = target - pred
  ssres = np.matmul(diff.T,diff)[0][0]
  target_mean = np.mean(target)
  diff_mean = target - target_mean
  sstot = np.matmul(diff_mean.T, 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 adjusted r2 score."""
  num = (1 - r2) * (n - 1)
  den = n - k - 1
  return 1 - num/den
```

1.5.4 Graphs from Multiple Linear Regression

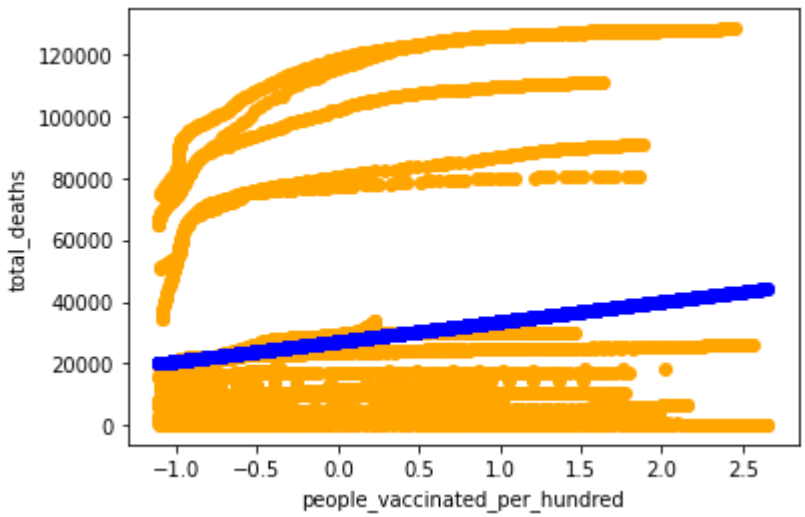
In [123]:

```
# Graphs obtained from 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")
```

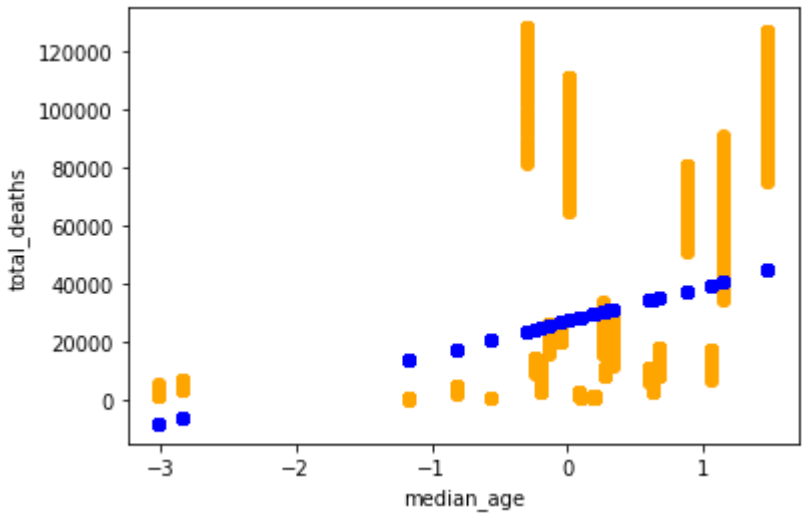
```
Feature(s) Used:  ('total_cases',)
```




```
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',)
```

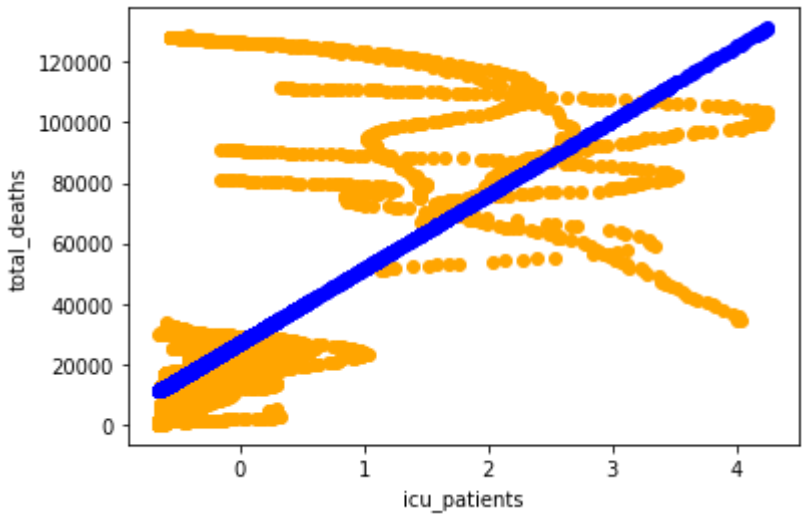



```
R^2 Score:  0.030613911953687567
Adjusted R^2 Score:  0.030134966258012486
Beta Coefficient Values:  [[27166.65169094]
 [ 6438.64587017]]


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



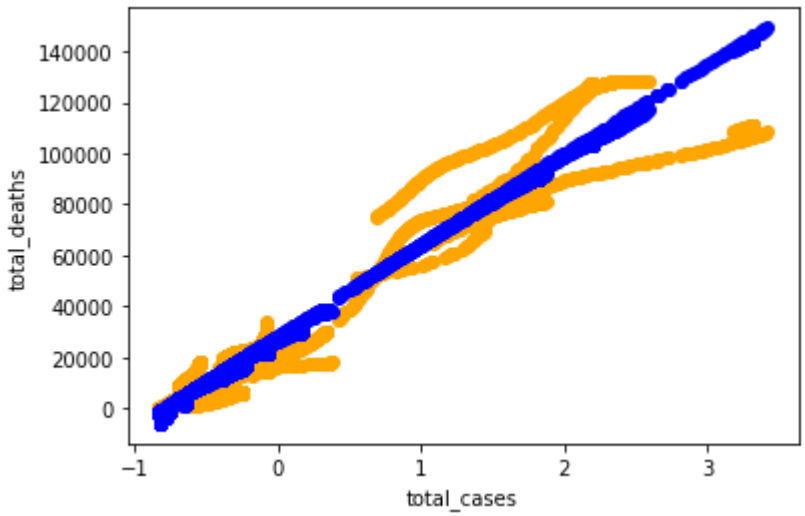

```
R^2 Score:  0.10348115070561648
Adjusted R^2 Score:  0.10303820661011531
Beta Coefficient Values:  [[27166.65169094]
 [11837.65693348]]
```

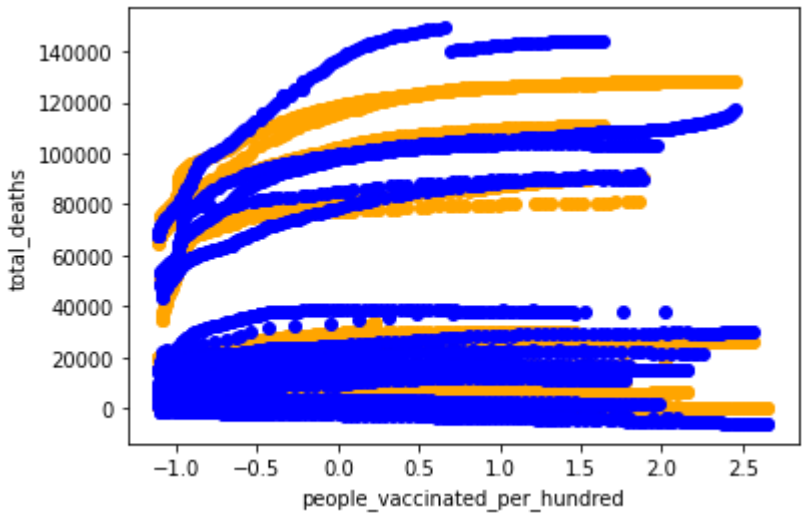
```
Feature(s) Used:  ('icu_patients',)
```




```
R^2 Score:  0.44173880542200117
Adjusted R^2 Score:  0.44146298467369194
Beta Coefficient Values:  [[27166.65169094]
 [24457.82881978]]
```

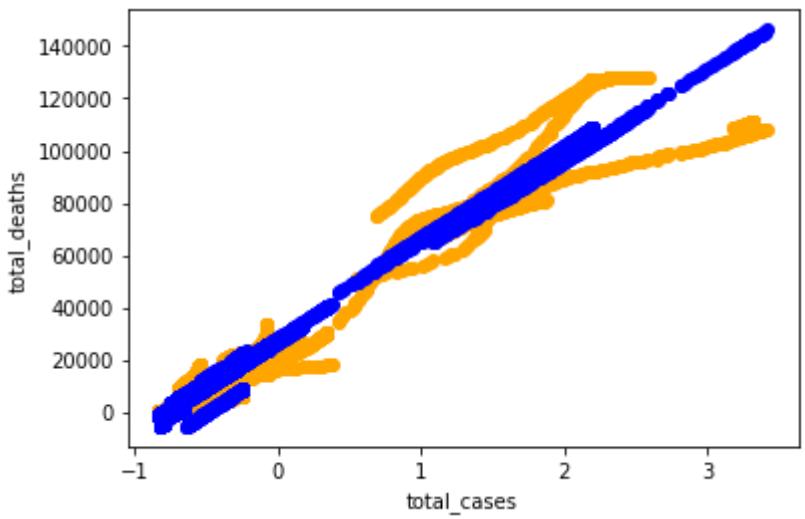
```
Feature(s) Used:  ('total_cases', 'people_vaccinated_per_hundred')
```

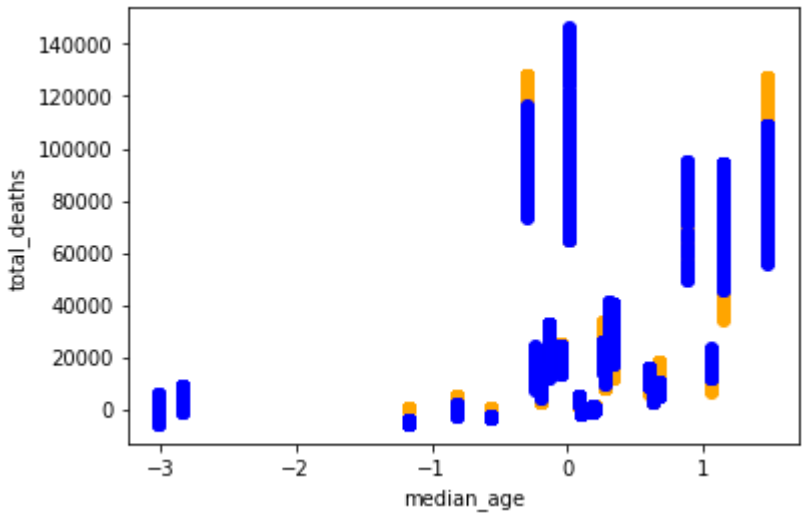





```
R^2 Score:  0.9370133659558506
Adjusted R^2 Score:  0.9369666746037052
Beta Coefficient Values:  [[27166.65169094]
 [35892.0923096 ]
 [-1360.81865527]]
```

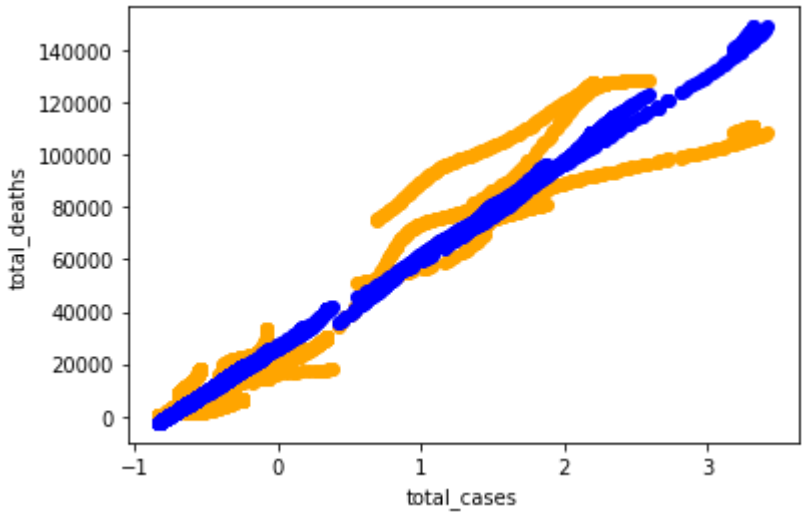
```
Feature(s) Used:  ('total_cases', 'median_age')
```

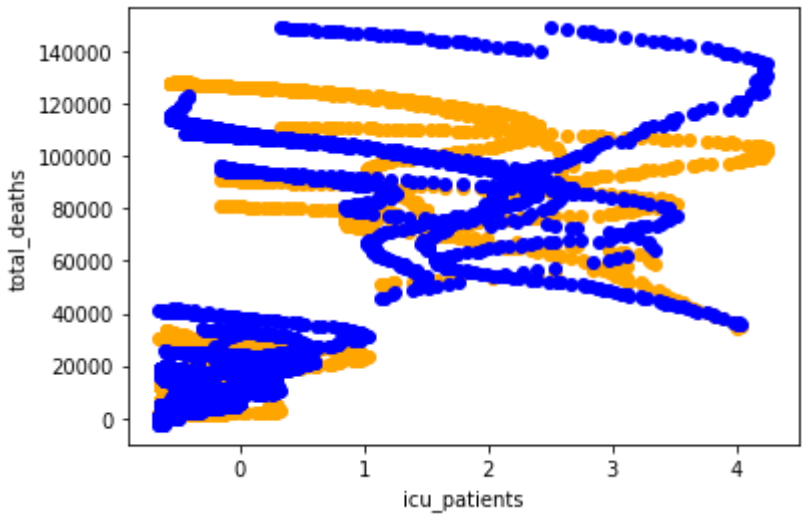




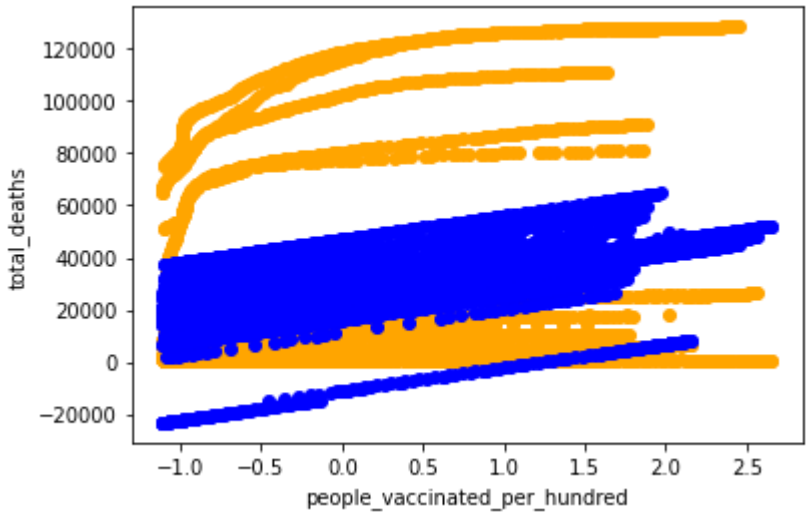

```
R^2 Score:  0.9441351025144288
Adjusted R^2 Score:  0.9440936904332682
Beta Coefficient Values:  [[27166.65169094]
 [34759.68711611]
 [ 3479.71524753]]
```

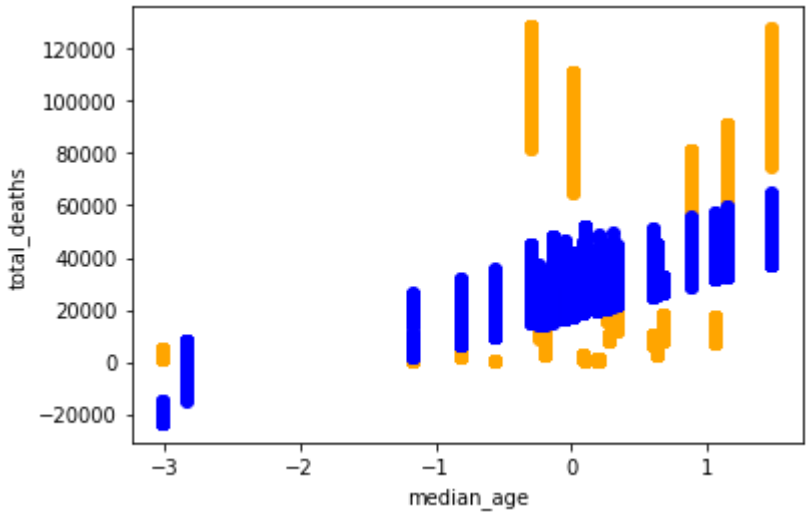
```
Feature(s) Used:  ('total_cases', 'icu_patients')
```



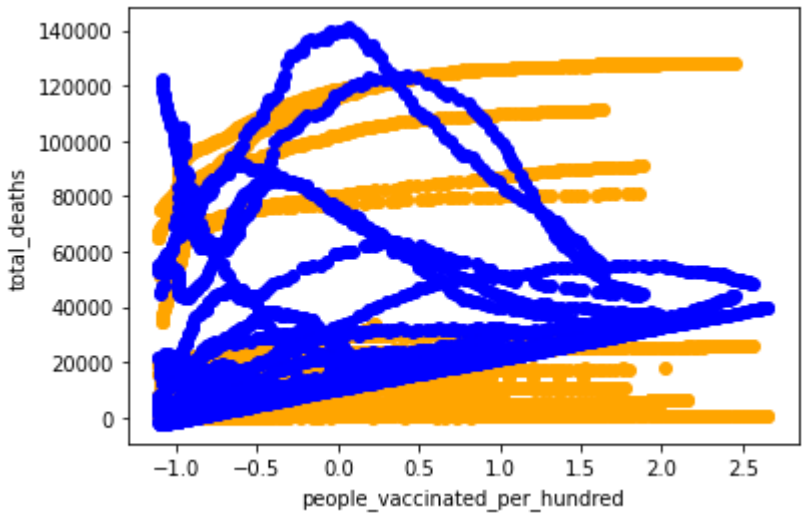


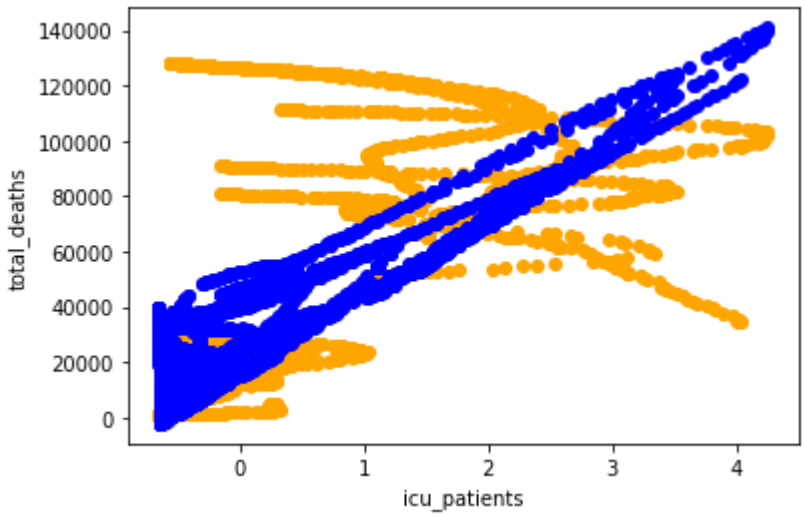
```
R^2 Score:  0.9367770623540452
Adjusted R^2 Score:  0.9367301958324397
Beta Coefficient Values:  [[27166.65169094]
 [36810.98352259]
 [-1708.69726686]]


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



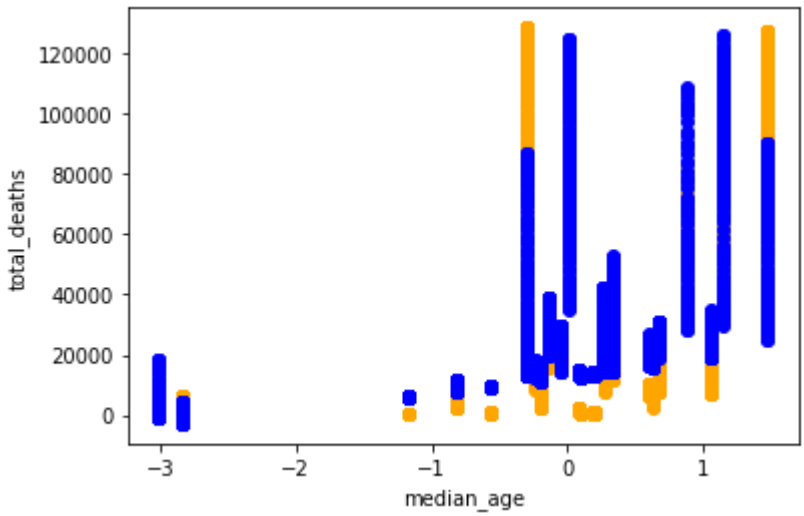


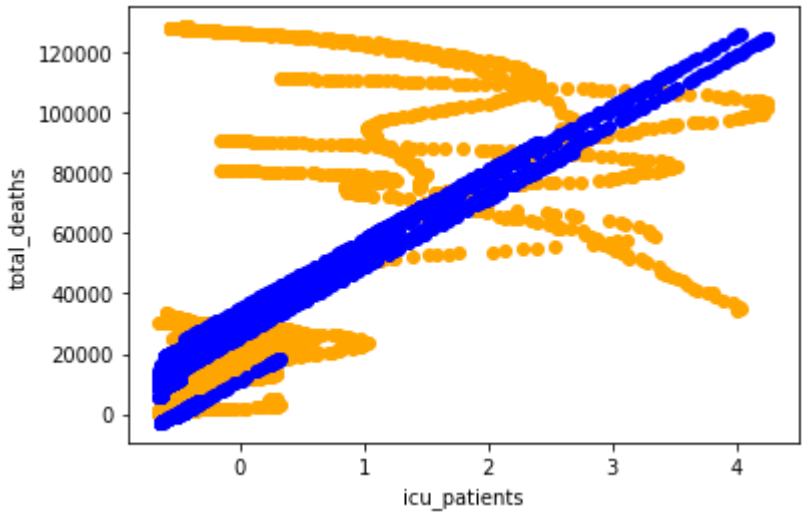

```
R^2 Score:  0.16031374692504352
Adjusted R^2 Score:  0.1596912960332163
Beta Coefficient Values:  [[27166.65169094]
 [ 8925.87390186]
 [13484.0921234 ]]


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



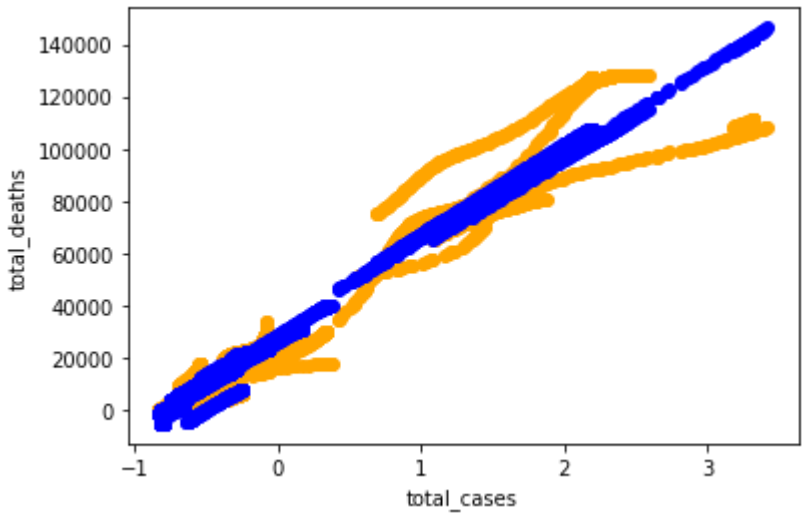


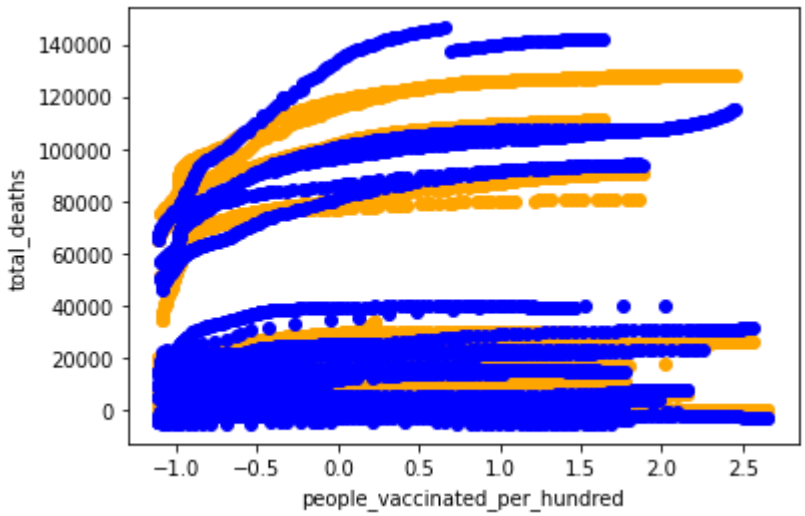

```
R^2 Score:  0.5341298856393752
Adjusted R^2 Score:  0.5337845408548232
Beta Coefficient Values:  [[27166.65169094]
 [11384.21476983]
 [26576.31020694]]


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

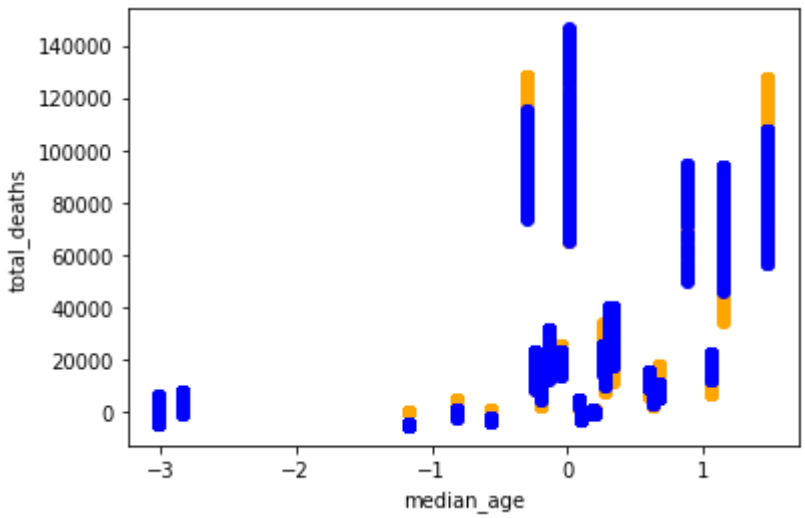




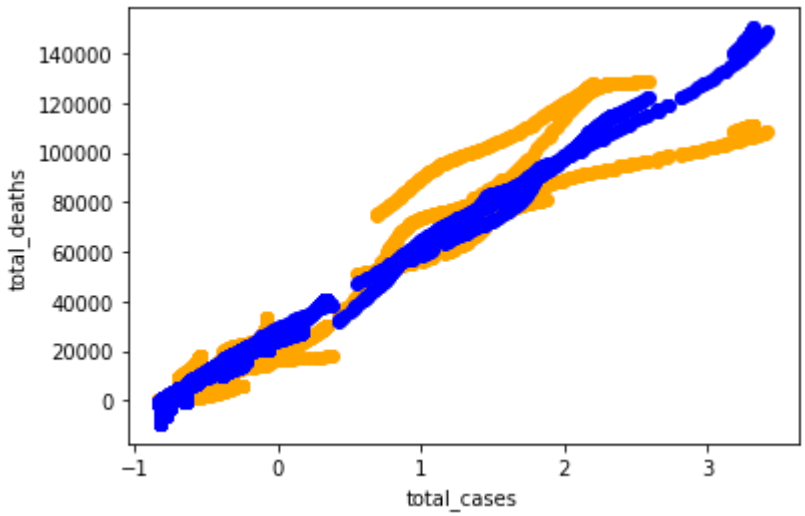

```
R^2 Score:  0.46149506582043476
Adjusted R^2 Score:  0.4610958775816063
Beta Coefficient Values:  [[27166.65169094]
 [ 5389.5766109 ]
 [22943.10708842]]


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

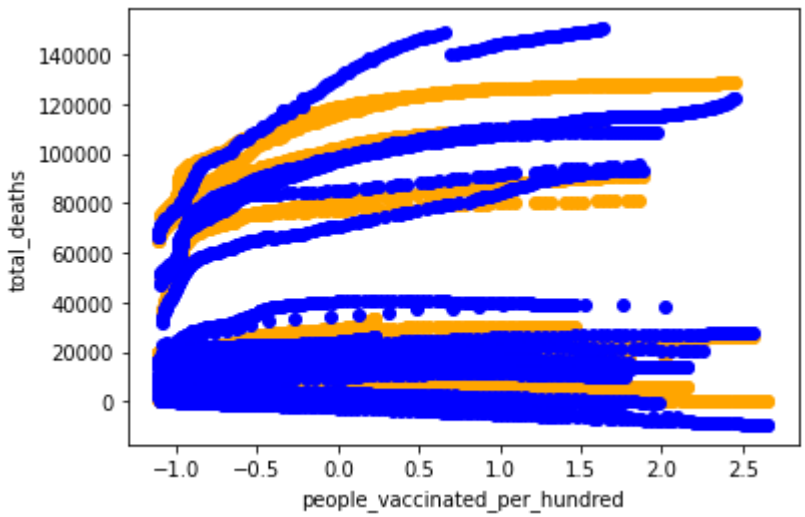

total_deaths
total_cases
140000
120000
100000
80000
60000
40000
20000
0
−1
0
1
2
3

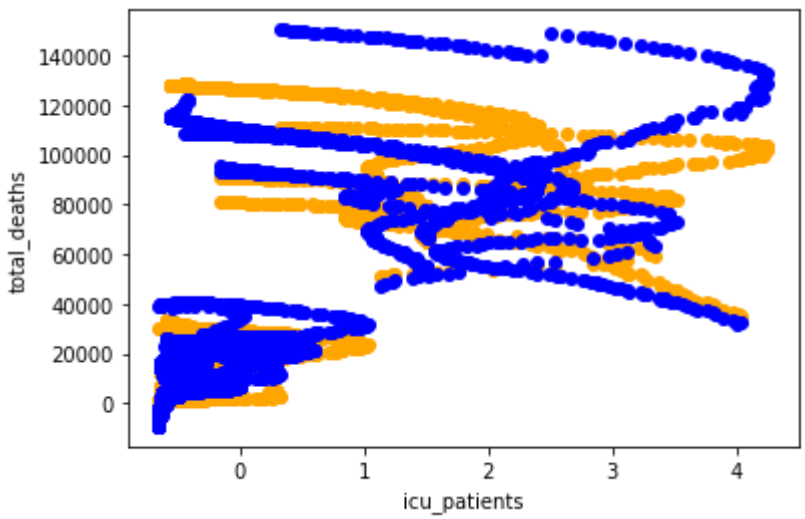

total_deaths
people_vaccinated_per_hundred
140000
120000
100000
80000
60000
40000
20000
0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


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

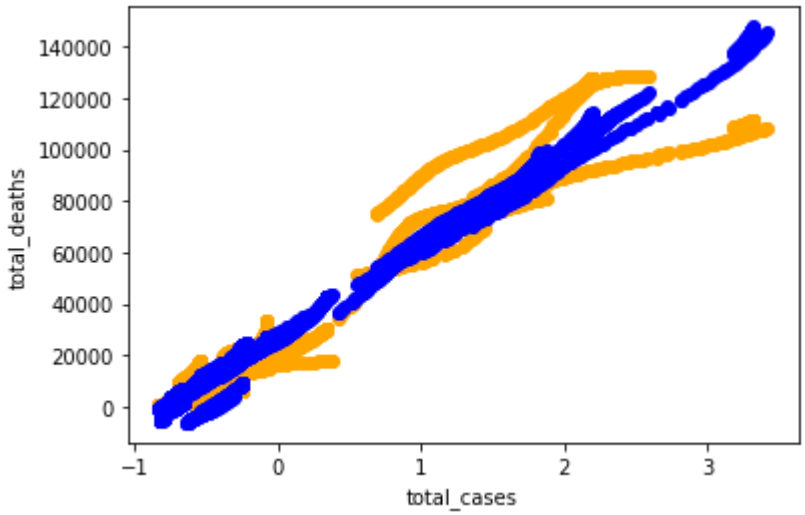
```
R^2 Score:  0.9443199623903851
Adjusted R^2 Score:  0.9442649153932426
Beta Coefficient Values:  [[27166.65169094]
 [34906.69793814]
 [ -529.36486602]
 [ 3346.72182702]]


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

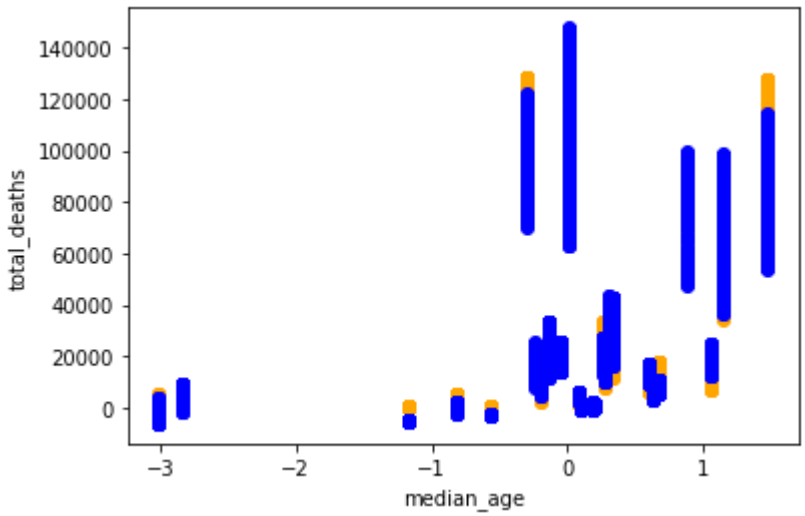


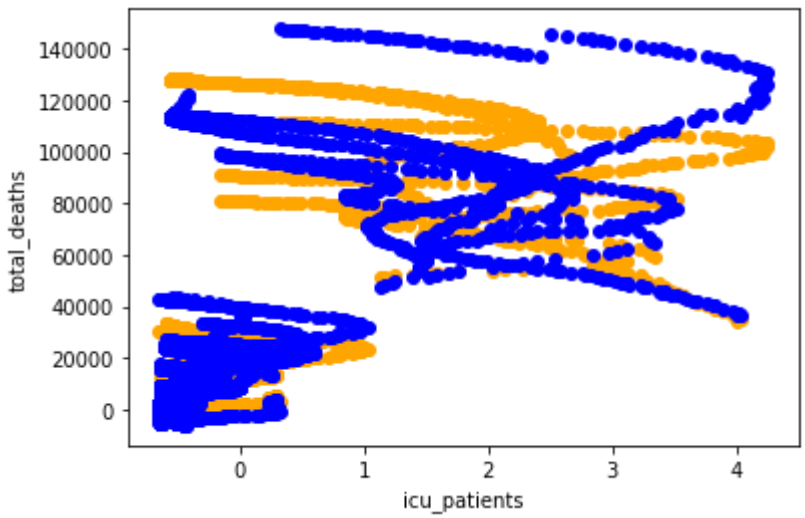




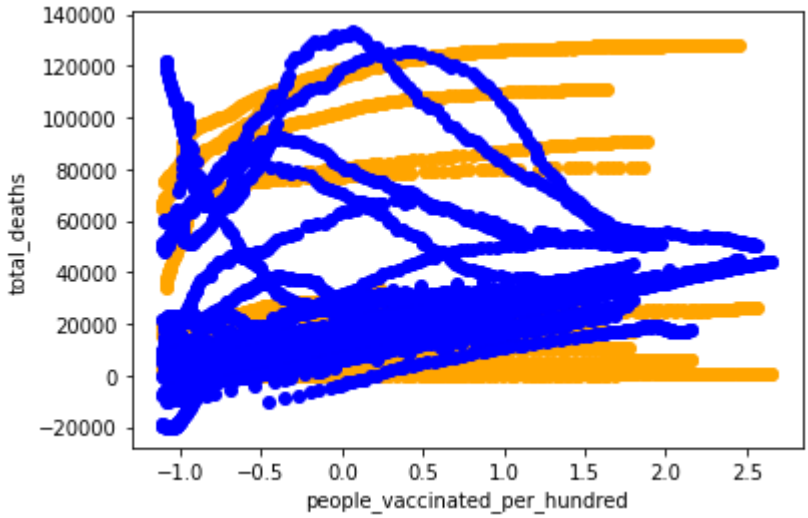
```
R^2 Score:  0.9404045368184566
Adjusted R^2 Score:  0.9403456189112084
Beta Coefficient Values:  [[27166.65169094]
 [38658.82986792]
 [-2614.99607721]
 [-3508.83363267]]


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

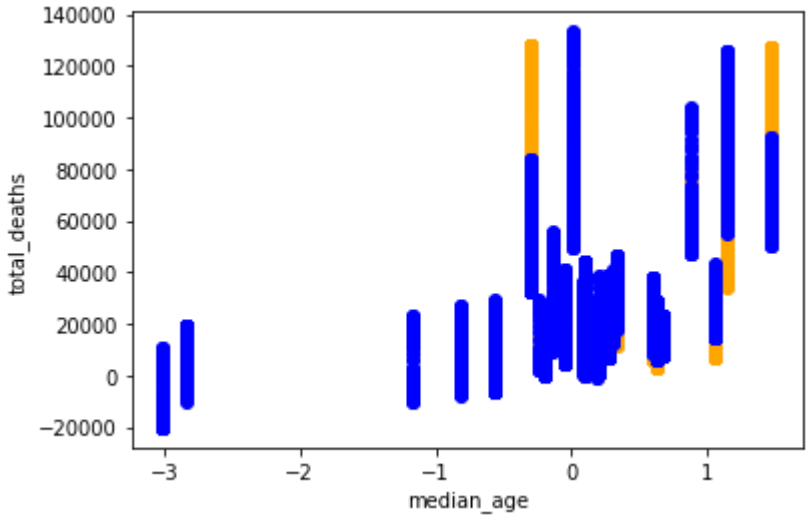


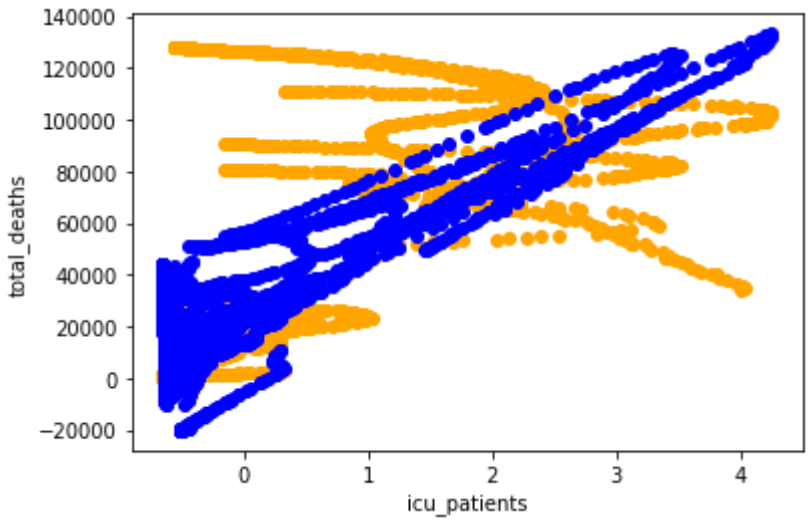




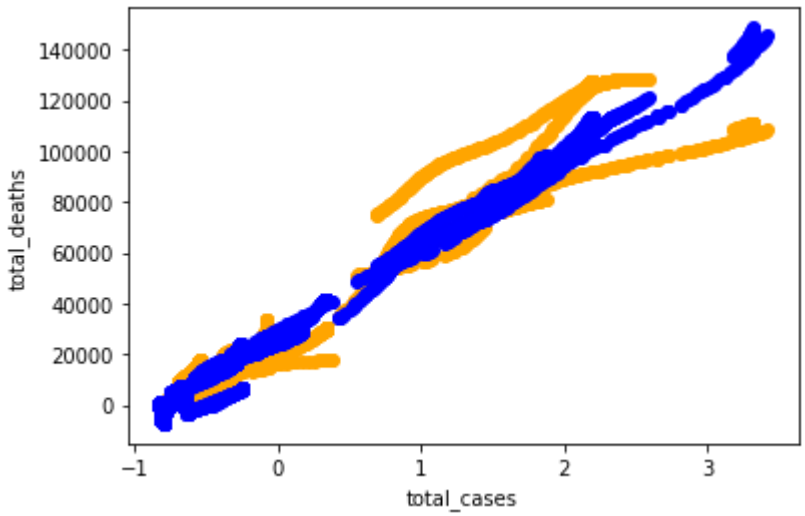
```
R^2 Score:  0.9464482014107781
Adjusted R^2 Score:  0.9463952584561669
Beta Coefficient Values:  [[27166.65169094]
 [36500.4070933 ]
 [ 3777.72965452]
 [-2549.64650923]]


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

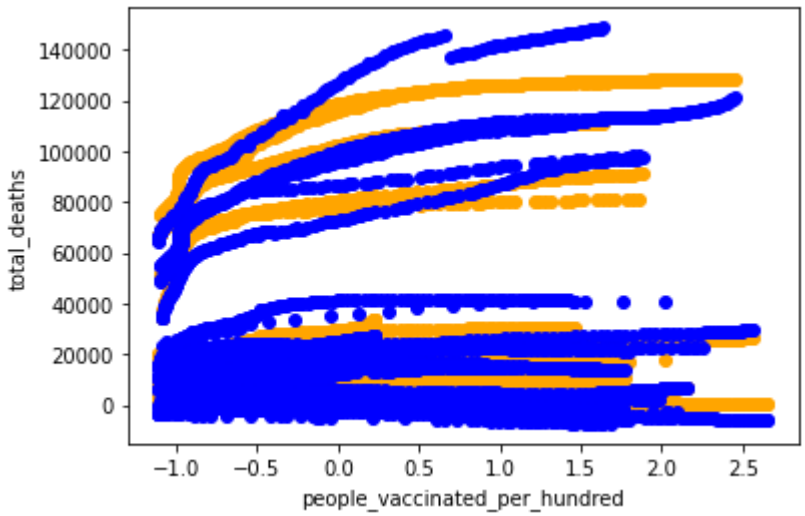


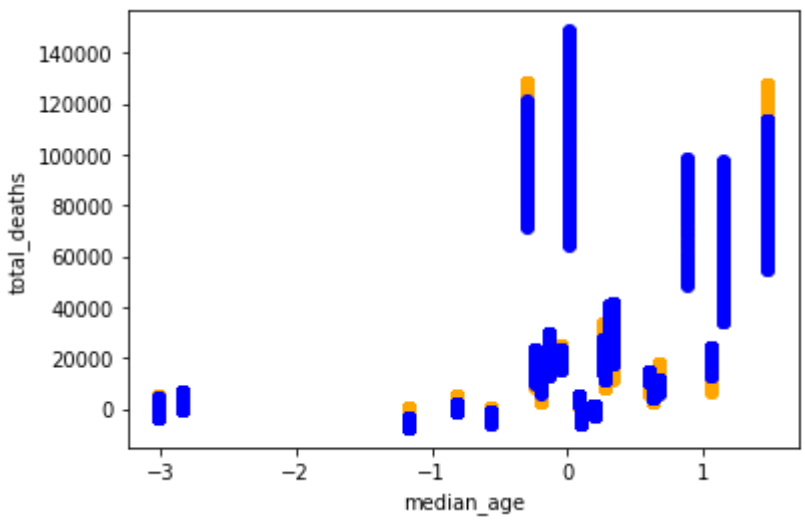


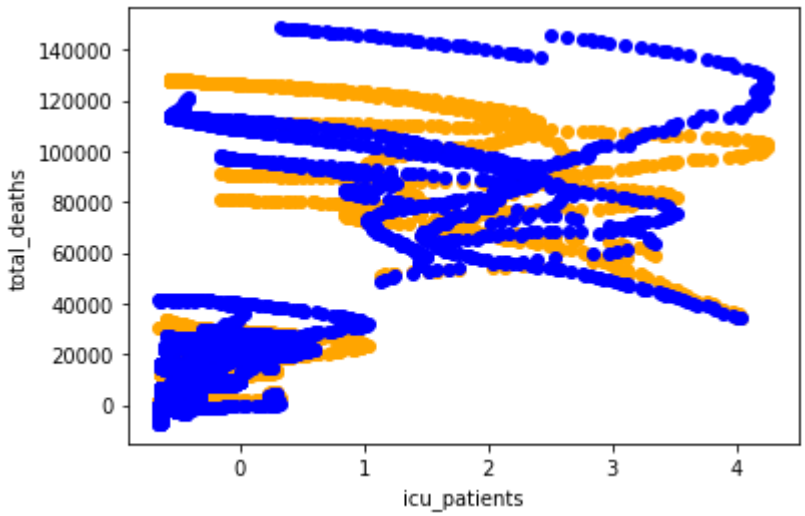


```
R^2 Score:  0.5683490715671141
Adjusted R^2 Score:  0.5679223281875463
Beta Coefficient Values:  [[27166.65169094]
 [12364.92373893]
 [ 7163.82710585]
 [24745.44107178]]


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









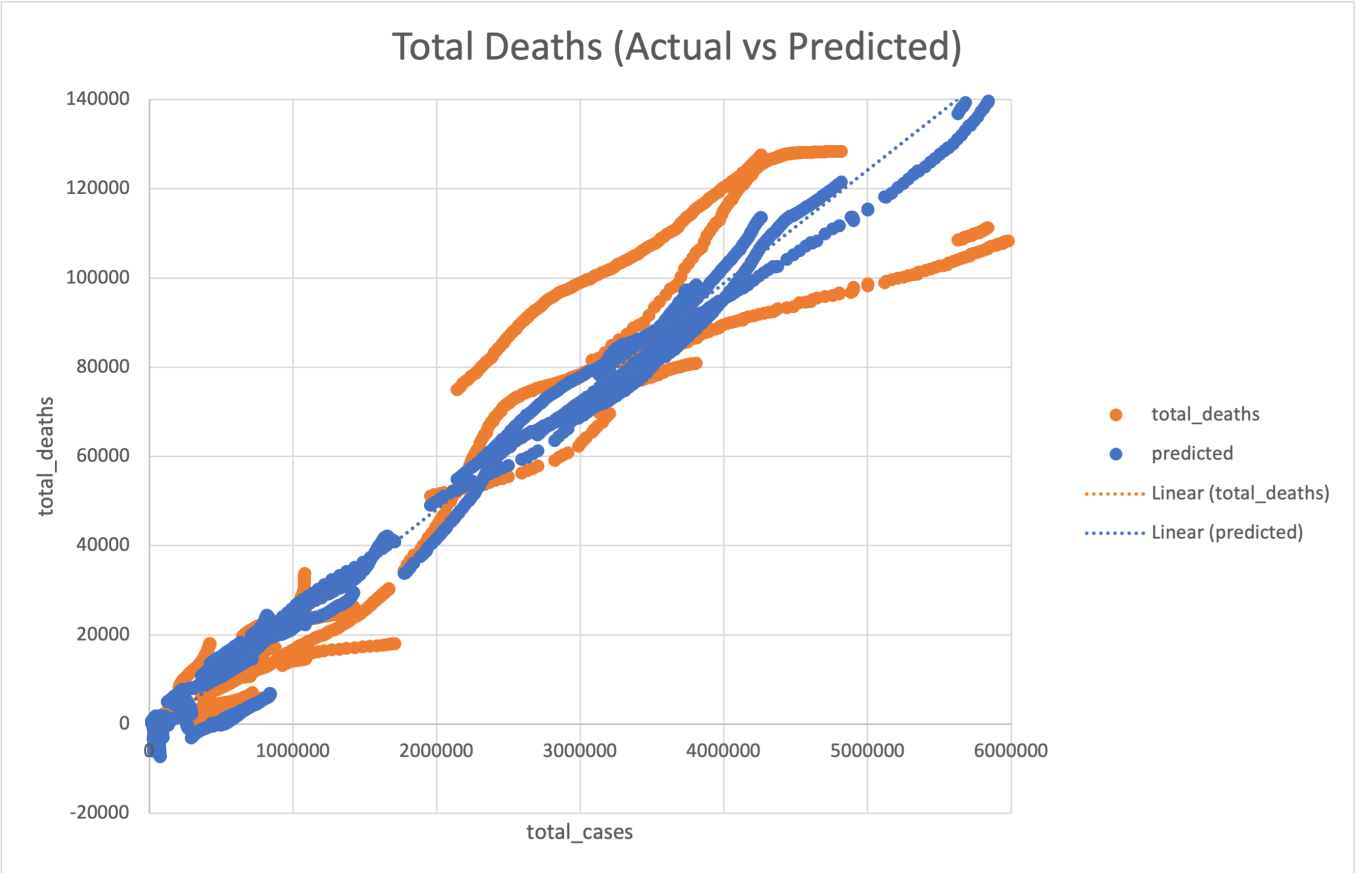

```
R^2 Score:  0.9481758023708868
Adjusted R^2 Score:  0.9481117427940894
Beta Coefficient Values:  [[27166.65169094]
 [37827.96947513]
 [-1841.15436338]
 [ 3454.91653116]
 [-3745.21773351]]
```

1.5.5 Proof of Concept (Excel vs Python)

For the sake of brevity and simplicity, we have only plotted the graphs for the variable X_1, representing the total cases. This is due to the fact that it had the highest Adjusted R^2 value amongst all single variables when we ran the Multiple Linear Regression analysis earlier. We hence deduced that it was the main factor affecting our model, and decided to plot the graph for this variable against the total deaths (actual dataset vs predicted).

The graphs below have different values on the x-axis because of Z-normalization as mentioned above (normalized for Python but not for Excel).

Graph plotted using Excel:



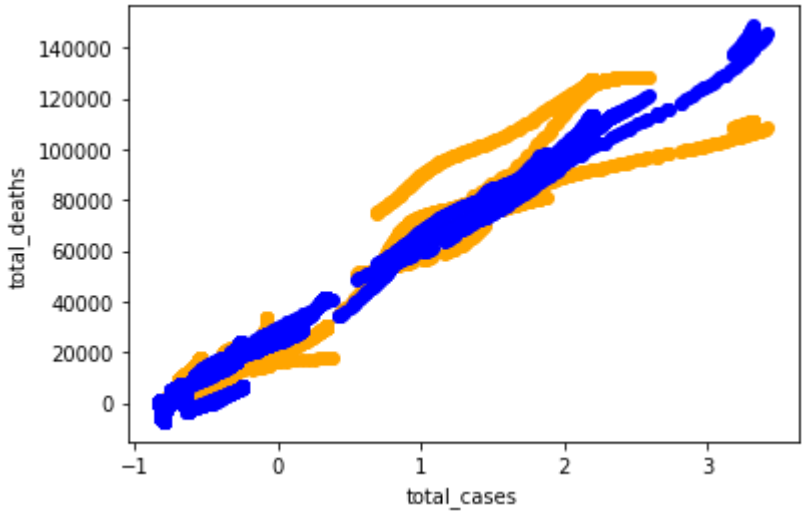
Graph plotted using Python:

In [125]:

```
df_features = normalize_z(df_features)
feature = prepare_feature(df_features)
ytrue = prepare_target(df_target)
pred = beta[0] + beta[1] * df_features['total_cases'] + beta[2] * df_features['p
eople_vaccinated_per_hundred'] + beta[3] * df_features['median_age'] + beta[4] *
df_features['icu_patients']
```

In [126]:

```
plt.figure()
plt.xlabel('total_cases')
plt.ylabel("total_deaths")
plt.scatter(df_features['total_cases'], ytrue, color='orange')
plt.scatter(df_features['total_cases'], pred, color='blue')
plt.show()
```



1.6 Sources

Sources:

1. https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/ (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/ (https://www.analyticsvidhya.com/blog/2020/07/difference-between-r-squared-and-adjusted-r-squared/)
3. https://www.investopedia.com/terms/p/p-value.asp (https://www.investopedia.com/terms/p/p-value.asp)
4. https://www.simplypsychology.org/p-value.html (https://www.simplypsychology.org/p-value.html)
5. https://www.geeksforgeeks.org/data-normalization-in-data-mining/ (https://www.geeksforgeeks.org/data-normalization-in-data-mining/)