DATA1002 Project - Stage 3

Group 12

Members: 510615460, 480385312, 510465739, 500522378

[**1.0 Overview**](#_19xacntv597w) **1**

[1.1 Topic](#_yh69rt7vnyja) 1

[1.2 Dataset](#_xlhiw09ifsrr) 1

[1.3 Training and Test Sets](#_berpqvxjlx5b) 4

[**2.0 Individual Sections**](#_jy5xxogit1m5) **5**

[2.1 Section 1 (510615460)](#_awl5tazcslc7) 5

[2.1.1 Description](#_ot7u4co9mmp) 5

[2.1.2 Evaluation](#_rw04tbawlqkk) 6

[2.2 Section 2 (480385312)](#_m96tqrsj2xni) 7

[2.2.1 Description](#_tb73cxa0yl1k) 7

[2.2.2 Evaluation](#_yoclctpvf8ez) 8

[2.3 Section 3 (510465739)](#_idb7kavtsi7m) 10

[2.3.1 Description](#_mdkhj3mkt1hc) 10

[2.3.2 Evaluation](#_ffl443sujwws) 11

[2.4 Section 4 (500522378)](#_6ttkh8wtlpj8) 13

[2.4.1 Description](#_hgny0po7fbqe) 13

[2.4.2 Evaluation](#_nqiicnt4ru6u) 14

[**3.0 Conclusion**](#_sc2vuzkxl5gn) **16**

Part A

# 1.0 Overview

## 1.1 Topic

The aim of this report is to produce predictive models that can be used to predict the number of new deaths from COVID-19 on any given day. This information is relevant to public health officials, as this can help them to predict the outcome of their country’s response to the virus which, in turn, may help a country to determine the best way to alter their response to the pandemic, and potentially advise on how to ‘live with the virus’ with the least number of deaths.

## 1.2 Dataset

The data used in this report is based on the one our group produced in Stage 2 of this project. In order to aid our machine learning approaches, we also chose to add three new columns to represent the cumulative cases, cumulative tests and cumulative vaccinations (the code for this can be found below). The new schema for the data is as follows:

| **Field Name** | **Type** | **Description** |
| --- | --- | --- |
| Date | Date | Date on which the data was collected. Represented in standard ISO format as YYYY-MM-DD. |
| Country | String | The country from which the data was recorded. |
| GDP | Integer | The gross domestic product for the current financial year, measured in USD. |
| Population | Integer | The population of the country. |
| New Cases | Integer | The new cases of COVID-19 recorded for this date. |
| Cumulative Cases | Integer | The total number of cases recorded up to this date for this country. |
| New Deaths | Integer | The new deaths of COVID-19 recorded for this date. |
| Tests | Integer | The number of COVID-19 tests administered for this date. |
| Cumulative Tests | Integer | The total number of tests recorded up to this date for this country. |
| Vaccinations | Integer | The number of COVID-19 vaccinations administered for this date. |
| Cumulative Vaccinations | Integer | The total number of vaccinations recorded up to this date for this country. |
| Vaccines Available | String Array | The names of the brands of COVID-19 vaccines available in the country. |

**Adding Cumulative Columns Code**

​​import csv

data = open("integrated\_data.csv")

new\_data = open("integrated\_data\_v2.csv", "w")

csv\_reader = csv.reader(data)

csv\_writer = csv.writer(new\_data)

cumulative\_vaccinations\_per\_country = {}

cumulative\_cases\_per\_country = {}

cumulative\_tests\_per\_country = {}

first\_line = True

for row in csv\_reader:

if first\_line:

csv\_writer.writerow([row[0], row[1], row[2], row[3], row[4], "Cumulative Cases", row[5], row[6], "Cumulative Tests", row[7], "Cumulative Vaccinations", row[8]])

first\_line = False

else:

country = row[1]

vaccinations = int(row[-2])

cases = int(row[4])

tests = int(row[6])

if country not in cumulative\_vaccinations\_per\_country:

cumulative\_vaccinations\_per\_country[country] = vaccinations

cumulative\_cases\_per\_country[country] = cases

cumulative\_tests\_per\_country[country] = tests

csv\_writer.writerow([row[0], row[1], row[2], row[3], row[4], cumulative\_cases\_per\_country[country], row[5], row[6], cumulative\_tests\_per\_country[country], row[7], cumulative\_vaccinations\_per\_country[country], row[8]])

else:

cumulative\_vaccinations\_per\_country[country] += vaccinations

cumulative\_cases\_per\_country[country] += cases

cumulative\_tests\_per\_country[country] += tests

csv\_writer.writerow([row[0], row[1], row[2], row[3], row[4], cumulative\_cases\_per\_country[country], row[5], row[6],cumulative\_tests\_per\_country[country], row[7], cumulative\_vaccinations\_per\_country[country], row[8]])

Additionally, we also chose to preprocess the data and encode all qualitative variables using a quantitative representation. The code to do this is as follows:

**Encoding Qualitative Variables Code**

from datetime import date

import pandas as pd

import csv

df = pd.read\_csv("integrated\_data\_v2.csv")

countries\_alphabetically = sorted(df["Country"].unique())

date\_chronologically = list(df["Date"].unique())

csv\_reader = csv.reader(open("integrated\_data\_v2.csv"))

csv\_writer = csv.writer(open("integrated\_data\_v3.csv", "w"))

first\_line = True

for row in csv\_reader:

if first\_line:

csv\_writer.writerow([row[0], "Date Number Indicator",row[1], "Country Numeric Indicator", row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10], row[11], "Number of Vaccines Available"])

first\_line = False

else:

for country in countries\_alphabetically:

number\_of\_vaccines\_available = len(row[11].split(","))

if country == row[1]:

for Date in date\_chronologically:

if Date == row[0]:

csv\_writer.writerow([row[0], date\_chronologically.index(Date),row[1], countries\_alphabetically.index(country), row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10], row[11], number\_of\_vaccines\_available])

## 1.3 Training and Test Sets

To split the data into train and test sets we used the train\_test\_split method from scikit learn, as shown in the following code:

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.1, random\_state=42)

This randomly split our data in training and test sets. The training set contains 90% of the data and the test set contains 10% of the data. We each used random\_state=42 to ensure we all got the same training and test data.

# 2.0 Individual Sections

## 2.1 Section 1 (510615460)

### 2.1.1 Description

This first model is a linear regression model, consisting of an input variable or the independent variable (x) and the single output variable or the dependent variable (y). In the case of this model, there was no preprocessing done to the data that was used and all of the input vectors were used as well.

The linear regression model can be considered the most well-understood algorithm used in predictive analysis. Linear regression models are simple to implement and interpret when it is evident that the relationship between variables x and y are linear, thus, allowing us to find out the cause and effect relationship between the two.

As the model used is a linear regression, no hyper-parameter tuning was done to identify the most optimal values to use. Instead, the model was only trained, tested and evaluated using all input vectors as followed:

**Training Code**

import pandas as pd

from sklearn import linear\_model

from sklearn import metrics

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

df = pd.read\_csv('integrated\_data\_v3.csv')

X = df[['Cumulative Vaccinations']]

y = df['Cumulative Cases']

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.1, random\_state=42)

reg = linear\_model.LinearRegression().fit(x\_train, y\_train)

coef = reg.coef\_

print('Coefficients:', coef)

intercept = reg.intercept\_

print('Intercept:', intercept)

**Training Code Output**

Coefficients: [0.12080543]

Intercept: 335661.4845839665

The training code also identifies the slope and the intercept of the linear regression model. The slope is encoded as the coefficients by using the .coef\_ function and the intercept is found using .intercept\_ function.

### 2.1.2 Evaluation

To evaluate the accuracy of the linear regression model, the mean square error (MSE), r-squared score and the accuracy score of the model were calculated.

**Evaluation Code**

y\_pred = reg.predict(x\_test)

mse = metrics.mean\_squared\_error(y\_test, y\_pred)

print('Mean squared error (MSE):', mse)

# R-squared score: 1 is perfect prediction

print('R-squared score:', metrics.r2\_score(y\_test, y\_pred))

**Evaluation Code Output**

Mean squared error (MSE): 1899179487719.8538

R-squared score: 0.6710300834037559

When the evaluation code above is run, the output shows that the model has a mean square error of 1899179487719.85 (2dp) and a R2 score of 0.67 (2dp). An MSE of 1899179487719.85 is a significantly higher value than of other models with a value of . While the values can be considered adequate, there is always room for improvement.

Although the cumulative vaccinations and cumulative cases may seem to have a linear relationship, to increase the accuracy of the model, regularisation techniques and cross validation could have been done before training and testing the model. Additionally, when the evaluation code is run using the train set, the MSE and R2  score were higher in comparison to the values of the test set - with an MSE of 1959608577137.38 (2dp) and R2  score of 0.71 (2dp). The difference in the two values may have been a result of how the train and test sets were sized. In terms of the test set, 10% of the inputted x variable was used which may suggest that the model is under-fitted. Moreover, the use of a linear regression model may have oversimplified the problem as it assumes a linear relationship among the two variables.

## 2.2 Section 2 (480385312)

### 2.2.1 Description

This model is a simple neural network model, consisting of an input layer, output layer and two hidden layers. In this model, all features of the input vector are utilised, and no preprocessing of the data is performed.

In general, neural networks have been shown to be more effective in prediction tasks than other model architectures. While this is a fairly simplistic neural network, the versatility of the model allows for more opportunities for customisation, especially in the number and size of hidden layers. This property is desirable for prediction tasks, as the tweaking of hyperparameters can further increase the accuracy of the model - an option that isn’t offered to a model architecture with fewer hyperparameters (e.g. k-nearest neighbour).

Before producing the model, a grid search was performed to determine the optimal values of the hyperparameters. The specific hyperparameters being investigated were the gradient optimiser (solver), the activation function (activation) and the number/size of the hidden layers (hidden\_layer\_sizes). The code for this is as follows:

**Hyperparameter Tuning Code**

import pickle

from sklearn.neural\_network import MLPRegressor

from sklearn.model\_selection import GridSearchCV

randomSeed = 42

# Load the data from file.

trainX = pickle.load(open("trainX.pkl", 'rb'))

trainY = pickle.load(open("trainY.pkl", 'rb'))

numFeatures = len(trainX[0])

# Set up the parameter grid.

paramGrid = {

"solver": ["adam", "sgd"],

"activation": ["relu", "tanh"],

"hidden\_layer\_sizes": [(numFeatures // 2), (numFeatures // 2, numFeatures // 4)]

}

# Build the base model.

model = MLPRegressor(random\_state=randomSeed)

# Run the grid search.

gridSearch = GridSearchCV(model, param\_grid=paramGrid).fit(trainX, trainY)

print(gridSearch.best\_estimator\_)

With the optimal hyperparameters found, the following code was used to build and train a neural network with these hyperparameters:

**Training Code**

import pickle

from sklearn.neural\_network import MLPRegressor

randomSeed = 42

### Hyperparameters ###

hiddenLayerSizes = (4, 2)

activation = "relu"

optimiser = "adam"

# Load the data from file.

trainX = pickle.load(open("trainX.pkl", 'rb'))

trainY = pickle.load(open("trainY.pkl", 'rb'))

# Build the model.

model = MLPRegressor(activation=activation, solver=optimiser, hidden\_layer\_sizes=hiddenLayerSizes, random\_state=randomSeed)

# Train the model.

model.fit(trainX, trainY)

# Save the model to file.

pickle.dump(model, open("neuralnetwork.pkl", 'wb'))

### 2.2.2 Evaluation

To evaluate the neural network model, I generated predictions for each sample in the test data, then compared these predictions to the actual values for each sample. In this comparison, I calculated a value for the mean squared error (MSE) between the two sets.

**Evaluation Code**

import pickle

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import mean\_squared\_error, r2\_score

# Load the trained model from file.

model = pickle.load(open("neuralnetwork.pkl", 'rb'))

# Load the data from file.

testX = pickle.load(open("testX.pkl", 'rb'))

testY = pickle.load(open("testY.pkl", 'rb'))

# Test the model.

predictedY = model.predict(testX)

# Calculate the MSE and R2 between predicted and actual.

mse = mean\_squared\_error(testY, predictedY)

r2 = r2\_score(testY, predictedY)

print("MSE: {}".format(mse))

print("R2: {}".format(r2))

When the above code is run on the previously-generated model, the R2 score is 0.40 (2dp), and the MSE is 24975.23 (2dp). This equates to an approximate absolute error of 158 deaths per sample.

These values are reasonable, but could definitely be improved. One such method would be to use feature selection on the input vectors to remove non-correlated (or less-correlated) features. While the number of cases and GDP are likely to be highly correlated with the number of deaths, it is possible that features like country and number of tests are less so, since the value from these features is better represented by other features. By including all features regardless of their correlation, extra noise is being introduced into the data, which introduces further confusion into the prediction process.

It is interesting to note that when this evaluation code is instead run using the train set, the R2 score and MSE are much higher, at 0.27 (2dp) and 55624.29 (2dp), respectively This equates to an approximate absolute error of 236 deaths per sample.

The higher R2 score and MSE seen on the training set could be a result of the way in which the data was partitioned. The training and testing sets were randomly partitioned, but it is possible that the sets are unrepresentative in terms of their distribution of countries (or indeed, any input feature) and/or number of deaths.

Additionally, the relative sizes of the train and test partitions may be a factor here, with the test set making up only 10% of the samples. Given the reduced size of the test set, it is possible that this model is underfitting on the data, but it is difficult to say this conclusively.

## 2.3 Section 3 (510465739)

### 2.3.1 Description

This is a k nearest neighbour (kNN) model using all the quantitative variables in our data set to predict the number of new deaths. The kNN model can be used on data with complex trends which makes it ideal for our dataset.

In order to use as many of the variables as possible, the qualitative variables of ‘Date’ and ‘Country’ were encoded into integers, and the ‘Vaccines Available’ variable was used to create a new variable called ‘Number of Vaccines Available’, which is a count of the separate vaccine brands listed in ‘Vaccines Available’. The code for this preprocessing is outlined in [Section 1.2](#_xlhiw09ifsrr).

Before producing the model, GridSearchCV was used to find the ideal hyperparameters, in particular, the ideal value for n\_neighbours and the ideal method for weights. ‘Weights’ refers to the method used to calculate the effect each neighbour will have on the model. There are multiple options for this, the two of which are being considered in this gridsearch being ‘uniform’ and ‘distance’. ‘Uniform’ weights means that each neighbour has the same effect on the model, while ‘distance’ takes into account the distance each neighbour is from the point of interest and adjusts the effect each neighbour has on the model accordingly. GridSearchCV is a more efficient way to tune hyperparameters than manually changing the hyperparameters and re-producing the model. It uses cross-validation to train and test models with the different hyperparameters listed. In the following code, it tests values for ‘n\_neighbour’ between 1 and 50, and the values of ‘uniform’ and ‘distance’ for ‘weights’.

**Hyperparameter Tuning Code**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import GridSearchCV

from sklearn.neighbors import KNeighborsRegressor

# Reading in data frame and creating X and Y variables

df = pd.read\_csv("integrated\_data\_v3.csv")

X = df[["Date Number Indicator",

"Country Numeric Indicator",

"GDP",

"Population",

"New Cases",

"Cumulative Cases",

"Tests",

"Cumulative Tests",

"Vaccinations",

"Cumulative Vaccinations",

"Number of Vaccines Available"]]

Y = df["New Deaths"]

# Splitting data set into train and test sets

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.1, random\_state=42)

# Finding best value for n\_neighbors using GridSearchCV

n\_neighbors = list(range(1,50))

weights = ["uniform", "distance"]

param\_grid = {"n\_neighbors" : n\_neighbors, "weights" : weights}

model = KNeighborsRegressor()

grid\_search = GridSearchCV(model, param\_grid=param\_grid).fit(X\_train, Y\_train)

print(grid\_search.best\_estimator\_)

**Hyperparameter Tuning Output**

KNeighborsRegressor(n\_neighbors=13, weights='distance')

The grid search found that n\_neighbours = 13 and weights = ‘distance’ would produce the most accurate model. Therefore the model was trained using those hyperparameters according to the following code.

**Training Code**

# Splitting data set into train and test sets

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.1, random\_state=42)

# Training model

neigh = neighbors.KNeighborsRegressor(n\_neighbors=13, weights='distance').fit(X\_train, Y\_train)

### 2.3.2 Evaluation

After training the model on the training set, the following code was run to test the model and evaluate its accuracy.

**Evaluation Code**

# Testing model

y\_pred = neigh.predict(X\_test)

# Calculate MSE and R2 score

mse = mean\_squared\_error(Y\_test, y\_pred)

r2 = r2\_score(Y\_test, y\_pred)

print("MSE =", mse)

print("R2 =", r2)

**Evaluation Output**

MSE = 4453.821898870453

R2 = 0.8930895412408398

The model has an mean-square error (MSE) of 4453.82 (2 d.p.) which equates to an error of about 68 deaths per sample. It has an R2 value of 0.89. This indicates that the model is quite accurate, especially since the R2 value is close to 1. However, when the training data was tested using the evaluation code it returned a perfect score for both measures (MSE = 0, R2 = 1). This could indicate overfitting to the training data which is potentially decreasing the accuracy of the model on the test data. Therefore, improvements need to be made to combat this.

The use of GridSearchCV likely contributed to the model’s accuracy as it allowed for the optimal hyperparameters to be chosen. Furthermore, the preprocessing of encoding the qualitative variables, in particular ‘Country’ and ‘Date’, into integers is likely to have made the model more accurate as it allowed it to account for differences between countries and changes over time.

## 2.4 Section 4 (500522378)

### 2.4.1 Description

This model is a statistical predictive model that applies the *k*-nearest neighbors(k-NN) algorithm, a non-parametric classification method. In the use of the k-NN regression emphasis, the input consists of the k of the closest training examples in a data set. The output is the property value for the object that I choose. Besides, this value is the average of the values of k nearest neighbors.

Basically, for regression, a useful technique could be to assign weights to the contributions of the neighbors, so the nearer neighbors contribute more to the average than the more distant ones. The parameter selection depends upon the data, a good k can be selected by various heuristic techniques.

Before producing the model, a grid search was performed to determine the best values for n\_neighbors of the hyperparameters. The grid of parameters contains two dictionaries, one for the n\_neigbors and the other is for the weights.

The code for this is as follows:

**Hyperparameter Tuning code**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import GridSearchCV

from sklearn.neighbors import KNeighborsRegressor

df = pd.read\_csv("integrated\_data\_v3.csv")

X = df[["New Cases", "Tests"]]

y = df["New Deaths"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.1, random\_state=42

)

# Finding best value for n\_neighbors using GridSearchCV

n\_neighbors = list(range(1, 50))

param\_grid = {"n\_neighbors": n\_neighbors, "weights": ["uniform", "distance"]}

model = KNeighborsRegressor()

grid\_search = GridSearchCV(model, param\_grid=param\_grid).fit(X\_train, y\_train)

print(grid\_search.best\_estimator\_)

Output:

KNeighborsRegressor(algorithm='auto', leaf\_size=30, metric='minkowski', metric\_params=None, n\_jobs=None, n\_neighbors=10, p=2,

weights='distance')

**Training code**

import pandas as pd

from math import sqrt

from sklearn import metrics

from sklearn import neighbors

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

# Load dataframe and create X and y variables

df = pd.read\_csv("Integrated\_data\_v3.csv")

X = df[["New Cases", "Tests"]]

y = df["New Deaths"]

# Splitting data set into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.1, random\_state=42

)

# Training model

Neigh = neighbors.KNeighborsRegressor(n\_neighbors=10).fit(X\_train, y\_train)

### 2.4.2 Evaluation

To evaluate the model, I generated predictions for each sample in the test data, then compared these predictions to the actual values for each sample. In this comparison, I calculated values for the mean squared error (MSE) and the R-squared score between the two sets to measure the accuracy of the model.

**Evaluation Code**

# Use the model to predict X\_test

y\_pred = neigh.predict(X\_test)

# Mean squared error

mse = metrics.mean\_squared\_error(y\_test, y\_pred)

print("Mean squared error (MSE):", str(mse))

# R-squared score: 1 is perfect prediction

print("R-squared score:", metrics.r2\_score(y\_test, y\_pred))

Output:

Mean squared error (MSE): 11090.77373415765

R-squared score: 0.7337747815615276

As shown, after the code is run, the output of the MSE is 11090.77(2dp), and the R-squared score is 0.73(2dp).

These values are reasonable and logical. Generally, higher *R2* values and smaller *MSE* values reflect better performance in the regression model. The R2 value is quite close to the perfect prediction -- 1, but it definitely could be improved. Also, in this application, this error is pretty large, which makes sense since I didn’t list many variables from the data frame and extracted two variables “New Cases” and “Tests” instead to estimate the “New Deaths”.

Part B

# 3.0 Conclusion

**Model Comparison and Evaluation**

Linear regression is a model that evaluates the relationship between an independent variable and dependent variable, and represents this relationship through a gradient-intercept line on a 2D plane. Linear regression models are relatively simple to interpret and implement when compared to other predictive methods. However, since a linear model assumes a linear relationship between the input and output variables, it is not appropriate for all datasets. In Section 1, a linear model was produced to identify the relationship between two variables in the dataset. While we got a reasonable R2 value, the MSE values for both the train and test sets were extremely high, which suggests that the data is not linear - in other words, a linear model is not an appropriate model for this dataset. Thus, we investigated other models to apply to the dataset.

Neural networks are a model consisting of interconnected nodes organised into layers. These nodes (and the connections between them) attempt to simulate neurons in the brain, with connections representing meaningful aspects with the data. Because of their layered nature, neural networks allow for more customisation and flexibility than linear regression models. However, the introduction of these layers comes with the tradeoff of adding more complexity to the model, making it harder to understand. The neural network in Section 2 had a better MSE than the linear model in Section 1 but a worse R2 score. This may be due to the lack of feature selection within this model, or potentially indicate improvements that could be made to the hidden layers of the model. In order to attempt to avoid the design challenges of improving the neural network, we then began to investigate a K Nearest Neighbour model, with the intention of returning to improve the neural network model if K Nearest Neighbour was not appropriate.

In a K Nearest Neighbour (kNN) model, samples are plotted on an n-dimensional plane based on the values of each of their features. For new samples, a prediction is made by considering all samples nearby to the new sample based on a distance measure, and using an aggregation method to actually make the prediction, based on the labels from the nearby samples. kNN is a more simple approach than neural networks, but allows for more complex dataset trends than linear regression. In our investigation, we found that this model performed the best on our dataset. In Section 3, we tried producing a kNN model using all variables in the dataset. This produced an accurate model - however, a major limitation with kNN models is the amount of memory it uses and the long amount of training time it requires, since it must consider all previous samples. Since our dataset is relatively small, this was not a big issue for us, but the addition of new data in the future could make it worse. Therefore, in Section 4 we decided to try reducing the features of the model to reduce the amount of memory and time needed, but this also reduced the accuracy of the model. Since we currently are not experiencing issues with memory and time, we have deemed that this feature reduction is not worth the decreased accuracy. We believe that future work could involve a deeper exploration into the dataset - particularly into which features most closely correlate with the number of deaths for a sample. This could further improve the kNN model, and could further lead into the development of a more accurate model architecture.

**Limitations**

The measures of accuracy used, mean-squared error (MSE) and R2, allows us to easily and effectively compare each of our models. However, there is one limitation with these measures that could decrease the certainty of our comparisons. These values are measures of the accuracy of the model as a whole, however, since our data is time based, if the accuracy of any of our models changed over time (for example, got less accurate with newer data), this would not be obvious from these measures. Therefore, it is difficult for us to determine which of our models, if any, will generalise well for future data.

The dataset contains mostly quantitative values which allow for simple and accurate predictions. However, some variables, namely ‘Date’, ‘Country’ and ‘Vaccines Available’ are qualitative, so we made the decision to encode them using a quantitative representation. For ‘Vaccines Available’ this was simply a matter of producing a new variable ‘Number of Vaccines Available’ which is a count of the number of vaccines listed in ‘Vaccine Available’. For ‘Date’ and ‘Country’ however, this required encoding them into integers. This encode may have potential impacts on the predictive models, since integers have a meaningful order. For ‘Date’ this is not a large issue, as the dates also have a meaningful order corresponding to the encoding. For ‘Country’ however, this may have introduced an order that in context has no semantic meaning, since the integer representation increments alphabetically. Therefore, this could decrease the accuracy of our models as they attempt to look for a pattern in the data that does actually exist.

Our aim was to produce models that could predict the amount of new deaths from COVID-19 based on other COVID-19 statistics on any given day. We believe that we have succeeded in that aim to some extent, however, there are certainly improvements and future exploration that could be made. Having a predictive model for new deaths could greatly help governments find the best ways to move out of the COVID-19 pandemic and begin living with the virus with the lowest number of deaths possible.