DATA1002 Project Stage 3

Group:
DATA1002-RE-L10-G7

Group members:
SID:490386567/cche4743
SID: 520603578/khar6548

Part A

In this project, the dataset used is the “SDG Goal3 cleaned dataset” (SDG_goal3_clean.csv), taken from the DATA1002 course stream. We aim to predict infant mortality rate from various attributes (region, maternal mortality ratio, neonatal mortality rate, and births attended by personnel).

The data was split into 8:2 ratio, for train and test data respectively. In order to make sure that the split was exactly the same for the respective individual parts, we included a seed in the splitting code.

```
firstline = True
dic = {}

with open("C:\\Users\\Kylie\\Downloads\\SDG_goal3_clean.csv") as file:
   for line in file:
       if firstline:
           firstline = False
       else:
           elements = line.split(",")
           country = elements[0]
           year = elements[1]
           region = elements[2]
           maternal = float(elements[3])
           skilledpersonnel = float(elements[4])
           infant = float(elements[5])
           neonatal = float(elements[11])

           temp = country + " " + year

           dic[temp] = [region, maternal, skilledpersonnel, neonatal, infant]

   df = pd.DataFrame(dic, index=["Region", "Maternal mortality ratio", "Births attended by skilled health personnel (%)", "Neonatal mortality rate", "Infant mortality rate"])
   df = df.transpose()
   df.columns = ["Region", "Maternal mortality ratio", "Births attended by skilled health personnel (%)", "Neonatal mortality rate", "Infant mortality rate"]

#splitting into training and test data
training_data = df.sample(frac=0.8, random_state=24)
test_data = df.drop(training_data.index)
```

490386567 [cche4743]

For this part of the project, a logistic regression model was used to predict Infant mortality rate from SDG dataset.

Python Code:

```python
import pandas as pd
import numpy as np
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Read and pre-process data:
df =
pd.read_csv('/Users/Andy/Library/Containers/com.microsoft.Excel/Data/Desktop/A
ssignments/Data stage 3/SDG_goal3_clean.csv')
df.replace(["Africa","Americas","Asia","Europe","Oceania"],[int(1),int(2),int(
3),int(4),int(5)], inplace=True)

# Select features and target data:
columns = ['Region','Year','Maternal mortality ratio','Neonatal mortality rate
(deaths per 1,000 live births)','Proportion of births attended by skilled
health personnel (%)']
df = df.loc[:, columns]

features = ['Region','Year','Neonatal mortality rate (deaths per 1,000 live
births)','Proportion of births attended by skilled health personnel (%)']

# Splitting data into test of train dataset:
X = df.loc[:, features]
Y = df.loc[:, ['Maternal mortality ratio']]

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state=24,
train_size = .80)

# Prediction:
logreg= LogisticRegression()
logreg.fit(X_train,y_train)

y_pred=logreg.predict(X_test)
print (X_test) #test dataset
print (y_pred) #predicted values
```

Pre-Processing:

In order to train data for the model, each Regions(`"Africa"`,`"Americas"`,`"Asia"`,`"Europe"` & `"Oceania"`) were replaced with a numerical value(`"1"`, `"2"`, `"3"`, `"4"` & `"5"` respectively). This way the data in the column will no longer be a string and the model can process all of the data as integers.

```
# Read and pre-process data:
df =
pd.read_csv('/Users/Andy/Library/Containers/com.microsoft.Excel/Data/Desktop/A
ssignments/Data stage 3/SDG_goal3_clean.csv')
df.replace(["Africa","Americas","Asia","Europe","Oceania"],[int(1),int(2),int(
3),int(4),int(5)], inplace=True)
```

How the model is produced:

Once the dataset is trained and processed, the next step was to select the variables that are going to be used to make a prediction. After picking and splitting feature and target data for the predictive model, a logistic regression body was created by `LogisticRegression()` using scikit-learn. Finally, the prediction is given by `.predict` after the logistic regression body is created.

```
# Select features and target data:
columns = ['Region','Year','Maternal mortality ratio','Neonatal mortality rate
(deaths per 1,000 live births)','Proportion of births attended by skilled
health personnel (%)']
df = df.loc[:, columns]

features = ['Region','Year','Neonatal mortality rate (deaths per 1,000 live
births)','Proportion of births attended by skilled health personnel (%)']

# Splitting data into test of train dataset:
X = df.loc[:, features]
Y = df.loc[:, ['Maternal mortality ratio']]

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state=24,
train_size = .80)

# Prediction:
logreg= LogisticRegression()
logreg.fit(X_train,y_train)

y_pred=logreg.predict(X_test)
print (X_test) #test dataset
print (y_pred) #predicted values
```

Output:

Test dataset and predicted values of infant mortality rate.

```
     Region  ...  Proportion of births attended by skilled health personnel
(%)
151       4  ...                                               99.9
108       3  ...                                               98.2
14        3  ...                                               99.4
102       4  ...                                              100.0
127       4  ...                                               99.3
153       5  ...                                               99.0
11        4  ...                                               98.5
64        4  ...                                               97.5
121       4  ...                                               99.1
57        4  ...                                               99.8
98        3  ...                                               97.9
111       1  ...                                               99.5
88        4  ...                                               99.8
138       4  ...                                               98.1
69        3  ...                                               99.9
147       4  ...                                               99.2
30        2  ...                                               99.8
49        4  ...                                               98.8
132       3  ...                                               99.9
90        4  ...                                               99.9
146       4  ...                                               99.5
31        2  ...                                               99.7
19        3  ...                                               26.5
116       5  ...                                               96.8
150       4  ...                                               99.9
61        4  ...                                               99.8
59        4  ...                                               98.4
26        2  ...                                               98.6
29        1  ...                                               98.5
58        4  ...                                               99.7
120       4  ...                                               99.1
99        3  ...                                               98.3
149       4  ...                                               99.9

[33 rows x 4 columns]
[  5  19  29   7   7  11   5   5   5   5  29  55   7  32  30   7   8   7
   7   5   7   8 555   6   5   5   5  29  38   5   5  34   5]
```

Evaluation:

```
# Evaluate performance:
from sklearn import metrics
from sklearn.metrics import r2_score
from sklearn.metrics import classification_report

print('Accuracy: ',metrics.accuracy_score(y_test, y_pred))

# R squared:
actual =
[4,67,31,17,4,57,5,9,3,11,82,66,4,35,27,6,20,6,17,2,7,14,258,11,8,4,6,69,70,10
,4,79,10]
predict =
[5,19,29,7,7,11,5,5,5,5,29,55,7,32,30,7,8,7,7,5,7,8,555,6,5,5,5,29,38,5,5,34,5
]
R_square = r2_score(actual, predict)
print('R Squared:', R_square)

# RMSE:
from sklearn.metrics import mean_squared_error
import math

mse = mean_squared_error(actual, predict)
rmse = math.sqrt(mse)
print('RMSE:',rmse)
```

Output:

```
Accuracy:  0.06060606060606061
R Squared: -0.3523925820563054
RMSE: 55.26328168893251
```

After evaluation, accuracy score the program returned a value of `0.06060606060606061` which is only 6% accurate. The R squared value is `-0.3523925820563054`, meaning it is predicting way worse from the actual target values. The RMSE square also shows a similar finding with a high score of `55.26328168893251`, showing a bad fit of the model. Thus, it is clear that this model does not predict the infant mortality rate well in this particular situation.

The reason why the logistic regression model is not a good choice in this situation is that the logistic regression model is most often used for classification and predicting a binary outcome, such as “yes” or “no”. An example of this is a system detecting and filtering email as either a spam or not a spam email. In the case of this report, the Y variable is filled with different integers and thus logistic regression model is not a good choice

520603578 [khar6548]

In this part, the model was made based on the kNN regression. However, we will go through this part later. We will go through the preprocessing since it comes first in the full code.

A. Pre-Processing

The first step in preprocessing was changing the “Region” attribute, before splitting the data into x (inputs) and y (output). The kNN regression cannot process string, so since “Region” was in string, it was coded into numbers 1-5 to make it possible to be processed by kNN.

The second part was rescaling the input attributes to scales of 0 to 1.

```
#making into float and splitting into input & output
training_data =
training_data.replace(["Africa","Americas","Asia","Europe","Oceania"],[int(1),
int(2),int(3),int(4),int(5)])
test_data =
test_data.replace(["Africa","Americas","Asia","Europe","Oceania"],[int(1),int(
2),int(3),int(4),int(5)])

x_train = training_data.drop('Infant mortality rate', axis=1)
y_train = training_data['Infant mortality rate']

x_test = test_data.drop('Infant mortality rate', axis=1)
y_test = test_data['Infant mortality rate']

#scaling to 0-1
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled = scaler.fit_transform(x_train)
x_train_named = pd.DataFrame(x_train_scaled,columns=x_train.columns,
index=x_train.index)
x_train = pd.DataFrame(x_train_scaled)

x_test_scaled = scaler.fit_transform(x_test)
x_test_named =
pd.DataFrame(x_test_scaled,columns=x_test.columns,index=x_test.index)
x_test = pd.DataFrame(x_test_scaled)
```

B. How the Model was Produced

Before making the model, the best number for k was calculated manually by code.

```
#checking for k
for K in range(17):
   K = K+1
   model = neighbors.KNeighborsRegressor(n_neighbors = K)
```

```
    model.fit(x_train, y_train)
    pred = model.predict(x_test) #make prediction
    error = sqrt(mean_squared_error(y_test,pred)) #calculate rmse
    if K == 1:
        smallesterror = error
        ksmallesterror = str(K)
    elif smallesterror >= error:
        smallesterror = error
        ksmallesterror = str(K)
print("k value with the smallest RMSE:", str(ksmallesterror))
```

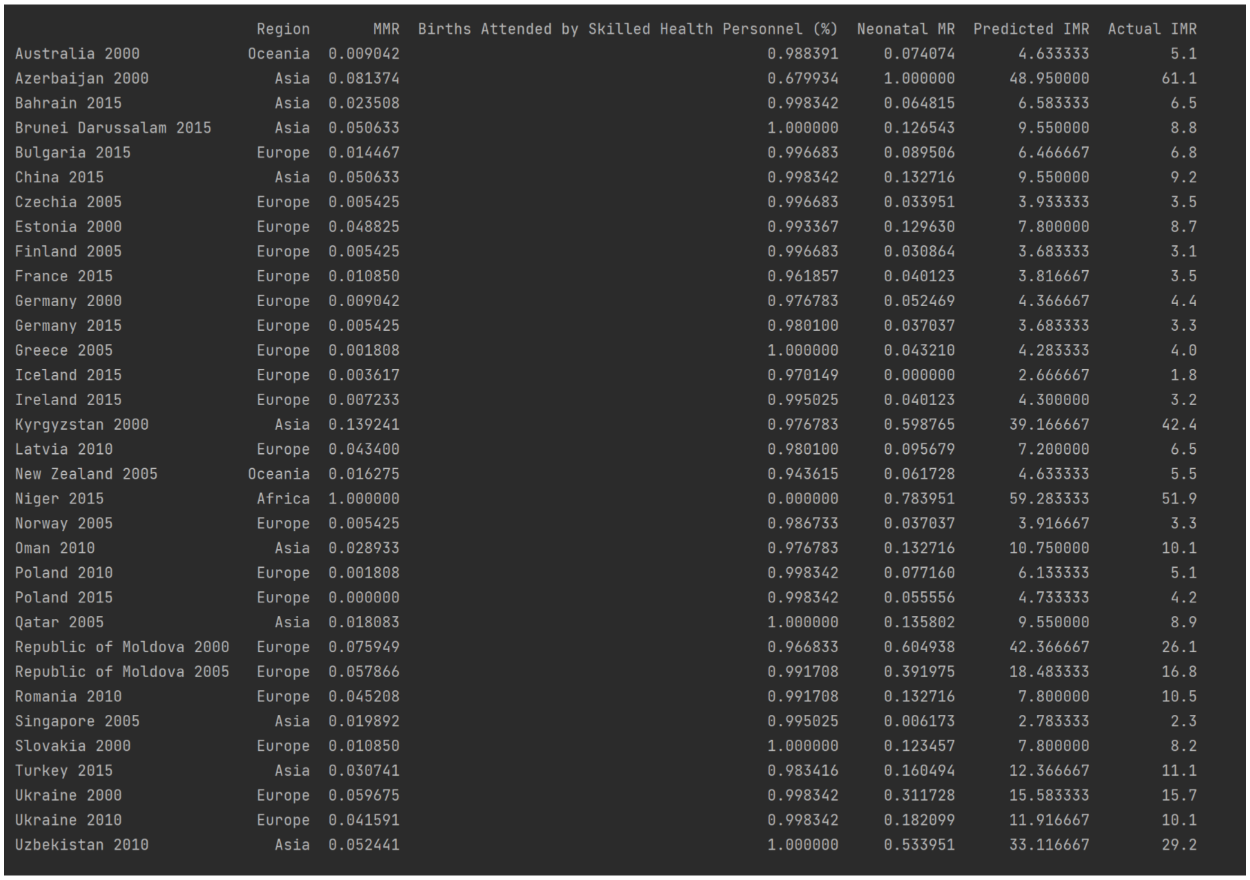
Output:

```
k value with the smallest RMSE: 6
```

The output was just to make sure that the number was not at the end of the given range, because that would mean a possibility of the error still decreasing and we can still get an even smaller error if we increased the range. Since it was 6, the error increased again after hitting 6. This value was then used as the k value for the kNN regression.

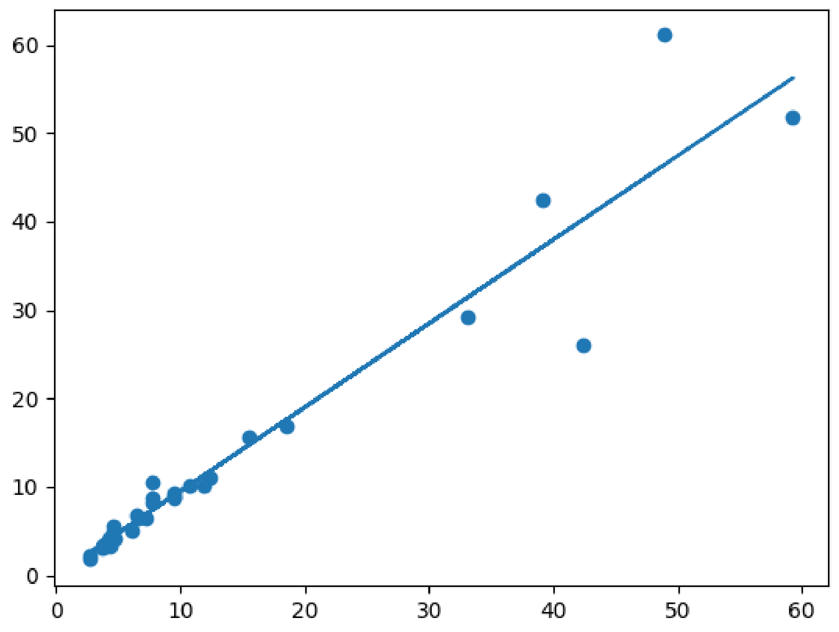
```
#predicting on the test set
knn_model = KNeighborsRegressor(n_neighbors=int(ksmallesterror))
knn_model.fit(x_train,y_train)
predict = knn_model.predict(x_test)

finalmodel = x_test_named.copy()
headers = ["Region", "MMR", "Births Attended by Skilled Health Personnel (%)",
"Neonatal MR"]
finalmodel.columns = headers
finalmodel["Predicted IMR"] = predict
finalmodel["Actual IMR"] = y_test
finalmodel["Region"] =
finalmodel["Region"].replace([float(0),float(0.25),float(0.5),float(0.75),floa
t(1)],["Africa","Americas","Asia","Europe","Oceania"])

pred_actl = x_test_named.drop(["Region", "Maternal mortality ratio", "Births
attended by skilled health personnel (%)", "Neonatal mortality rate"], axis=1)
pred_actl["Predicted infant mortality rate"] = predict
pred_actl["Actual infant mortality rate"] = y_test
print(finalmodel)
print("\n")

plt.scatter(pred_actl["Predicted infant mortality rate"],pred_actl["Actual
infant mortality rate"])
a, b = numpy.polyfit(pred_actl["Predicted infant mortality
rate"],pred_actl["Actual infant mortality rate"],1)
plt.plot(pred_actl["Predicted infant mortality rate"],a*pred_actl["Predicted
infant mortality rate"]+b)
plt.show()
```

Output:



```
                          Region       MMR  Births Attended by Skilled Health Personnel (%)  Neonatal MR  Predicted IMR  Actual IMR
Australia 2000           Oceania  0.009042                                        0.988391     0.074074       4.633333         5.1
Azerbaijan 2000             Asia  0.081374                                        0.679934     1.000000      48.950000        61.1
Bahrain 2015                Asia  0.023508                                        0.998342     0.064815       6.583333         6.5
Brunei Darussalam 2015      Asia  0.050633                                        1.000000     0.126543       9.550000         8.8
Bulgaria 2015             Europe  0.014467                                        0.996683     0.089506       6.466667         6.8
China 2015                  Asia  0.050633                                        0.998342     0.132716       9.550000         9.2
Czechia 2005              Europe  0.005425                                        0.996683     0.033951       3.933333         3.5
Estonia 2000              Europe  0.048825                                        0.993367     0.129630       7.800000         8.7
Finland 2005              Europe  0.005425                                        0.996683     0.030864       3.683333         3.1
France 2015               Europe  0.010850                                        0.961857     0.040123       3.816667         3.5
Germany 2000              Europe  0.009042                                        0.976783     0.052469       4.366667         4.4
Germany 2015              Europe  0.005425                                        0.980100     0.037037       3.683333         3.3
Greece 2005               Europe  0.001808                                        1.000000     0.043210       4.283333         4.0
Iceland 2015              Europe  0.003617                                        0.970149     0.000000       2.666667         1.8
Ireland 2015              Europe  0.007233                                        0.995025     0.040123       4.300000         3.2
Kyrgyzstan 2000             Asia  0.139241                                        0.976783     0.598765      39.166667        42.4
Latvia 2010               Europe  0.043400                                        0.980100     0.095679       7.200000         6.5
New Zealand 2005         Oceania  0.016275                                        0.943615     0.061728       4.633333         5.5
Niger 2015                Africa  1.000000                                        0.000000     0.783951      59.283333        51.9
Norway 2005               Europe  0.005425                                        0.986733     0.037037       3.916667         3.3
Oman 2010                   Asia  0.028933                                        0.976783     0.132716      10.750000        10.1
Poland 2010               Europe  0.001808                                        0.998342     0.077160       6.133333         5.1
Poland 2015               Europe  0.000000                                        0.998342     0.055556       4.733333         4.2
Qatar 2005                  Asia  0.018083                                        1.000000     0.135802       9.550000         8.9
Republic of Moldova 2000  Europe  0.075949                                        0.966833     0.604938      42.366667        26.1
Republic of Moldova 2005  Europe  0.057866                                        0.991708     0.391975      18.483333        16.8
Romania 2010              Europe  0.045208                                        0.991708     0.132716       7.800000        10.5
Singapore 2005              Asia  0.019892                                        0.995025     0.006173       2.783333         2.3
Slovakia 2000             Europe  0.010850                                        1.000000     0.123457       7.800000         8.2
Turkey 2015                 Asia  0.030741                                        0.983416     0.160494      12.366667        11.1
Ukraine 2000              Europe  0.059675                                        0.998342     0.311728      15.583333        15.7
Ukraine 2010              Europe  0.041591                                        0.998342     0.182099      11.916667        10.1
Uzbekistan 2010             Asia  0.052441                                        1.000000     0.533951      33.116667        29.2
```

<u>NOTES (column names):</u>
MMR: Maternal Mortality Ratio (per 100,000 live births)
Neonatal MR: Neonatal Mortality Rate (per 1,000 live births)
IMR: Infant Mortality Rate (per 1,000 live births)



This figure was created in the matplotlib part of the code. This scatter plot shows the predicted IMR (x-axis) against the actual IMR (y-axis). It also shows the regression line of the function.

C. Evaluation

```
#RMSE
from sklearn.metrics import mean_squared_error
from math import sqrt
mse = mean_squared_error(pred_actl["Actual infant mortality
rate"],pred_actl["Predicted infant mortality rate"])
rmse = sqrt(mse)
print("RMSE: "+ str(rmse))

#R2
from sklearn.metrics import r2_score
r2score = r2_score(pred_actl["Predicted infant mortality
rate"],pred_actl["Actual infant mortality rate"])
print("R2 Score: "+ str(r2score))
```

Output:

```
RMSE: 3.955962341471168
R2 Score: 0.9238064821355301
```

Since the predicted attribute is a regression (IMR), the evaluation process consists of calculating the RMSE and the R^2 score.

The RMSE is the average distance between the predicted IMR and the actual value of IMR. This does not exactly suitable for direct comparison unless it is being used to compare different regressions of the same train and test data. This project involves doing exactly that, which makes this measure applicable.

The R^2 score grades how well the data fits the regression. It usually ranges from 0 to 1, but can be negative if the model does not match the data. R^2 of 1 is a perfect model. The model with R^2 of 0.924 is quite good. Since the R^2 is a grade that is already in a standard range of values, it can be compared with other models that are not necessarily from the same data split or even the same data.

Part B

Two different methods were used in making the predictive models for the project: the logistic regression method and the kNN method. Before making the predictive models, the dataset was divided into training data and test data, in an 8:2 ratio. In order to make sure the same exact split of data was used, we split the data using seed ("random_state=24") in the code. The training data was used to make the regression, and test data was then used to see how accurate the regression is.

Logistic predictive model is based on logistic regression. It is an algorithm best used to find the binary probability of an event(0/1, True/False, Success/Failure etc.). The logistic predictive model predicts by categorising data into classes and studying the linear relationship from the given data. It calculates probability of outcomes using the Sigmoid function, which converts the numerical result into a binary expression of probability of 1 and 0.

The kNN is short for the k-Nearest Neighbors regression method. It gives a regression based on considering different attributes in training and taking the closest k objects in making the predictions. In counting for k, RMSE was counted first for different k values, to make sure that the regression can be made as accurate as possible. The k with the smallest RMSE was taken.

Weighted kNN would be able to give a more accurate regression since it gives more weight to objects that are closer than those that are further away. However, for this project, the standard kNN method (not weighted) was used.

At the end of both methods, different measures were taken in order to compare them and evaluate which one was more suitable in this case. RMSE is the root-mean-square error, which is the average distance from the predictions to the actual values. Since there is no standard range for the value of the MRSE, it is only suitable to evaluate different regression methods for the same datasets (and the same split for the training and test data).

The logistic regression had a very high RMSE of 55.26. The kNN regression, on the other hand, had an RMSE of 3.956. Since the RMSE measures the distance between the predicted and actual values, the lower the RMSE score the better. The kNN regression had a lower RMSE, which means the logistic model was a very bad fit and the kNN model was more suitable since it is overall closer to the actual values.

The second measure used is the R^2. R^2 is the coefficient of determination, which grades how well the regression fits the actual values. It usually goes from 0 to 1. However, it can become negative if the regression truly does not fit the dataset. An R^2 of 1 is a perfect regression, where the predictions are all accurate. Since it grades the regression in a more standardized range, this measure can be used to compare different regressions. They do not necessarily have to come from the same dataset.

The logistic regression had a poor R^2 of -0.3523925820563054, while the kNN regression produced an R^2 of 0.924. The kNN regression had a higher R^2 value, which means that the predictions were more accurate.

Overall, it can be concluded that the kNN model is much better at predicting the Infant Mortality Rate of the SDG dataset than the logistic model. Even though logistic regression is extremely effective for calculating probabilities and helping estimate the probability of an event happening, it is not particularly suited for this type of data. The logistic regression model is best suited for classification and predicting binary variables instead of numbers. The poor evaluation score of logistic regression from the report shows this.

Based on the evaluation above, the pros and cons of each regression method can be summarized as such:

Logistic Regression	
Pros	Cons
- Easy to implement and train - It is very fast at classifying unknown records. - Good accuracy for many simple data sets and it performs well when the dataset is linearly separable.	- If the number of observations is lesser than the number of features, Logistic Regression should not be used, otherwise, it may lead to overfitting. - Non-linear problems can’t be solved with logistic regression because it has a linear decision surface. Linearly separable data is rarely found in real-world scenarios.

kNN Regression	
Pros	Cons
- Seeing the trend of the attributes is not necessary, since it just takes the closest k objects - Quite accurate in predicting	- Could be more accurate if it was weighted kNN - Took a long time to run

The regression could perhaps be made better or more accurate if we had done more research or dataset that includes attributes that truly are factors of IMR. Our regression right now mainly relies on assumptions such as “places with lower maternal mortality rate would have lower infant mortality rate.” It may be true, but it does not exactly show a causal relationship between the attributes. The methods used also do not show which attribute has the most similarity to the attribute that is being predicted. It could possibly be more accurate if the results are weighted based on which attributes have the most similar changes to the one being predicted.

Two evaluation measures were also used since the measures have their own pros and cons. The limitation of RMSE comes from the fact that it is not in a standardized range. It is not exactly versatile, since the value does not exactly help in comparing regressions for different datasets. The value can also look meaningless until it is compared with another RMSE. It can only be used to compare different methods of regression that are done on the same exact dataset with the same exact split between training data and test data. RMSE shows the literal value, so it does a better job of giving an illustration of the literal deviation of the predictions from the actual values.

R^2 is more versatile since it is in a standardized range, so the score can be used to compare how "well" the regression fit is between different models from different datasets. However, this does not give an image of the size of the actual deviation. Due to this, it is best to include both measures. How "good" or "bad" a regression is also affected by the standard based on context, so we still cannot grade that based on R^2 alone.

They are both similar in the sense that they calculate the scores based on the size of the deviations. This means that they are sensitive to outliers, which is great for data where large deviations are undesirable, but not exactly suitable where outliers do not really matter. The measures also do not help in detecting internal problems with the data (e.g. bias).