Report

Of

Project Stage 3

Group: 1002_L32_G3

Group members:

500366554 yxia0083

520595273 hliu4477

530649584 ruli0058

530333591 zche9596

Part A:

Initial Section: Domain and Dataset Overview

- **Domain**: This project focuses on the medical field, especially the relationship between mortality and medical density. In the dataset used, medical density usually refers to the number of available medical resources (such as doctors, nurses, pharmacists, etc.) per unit population in the area. We aim to explore how the distribution of medical resources affects mortality in different regions to provide data support for formulating public health policies.
- **Dataset**: The dataset "SDG_goal3_clean" source is a data file sent via email in response to a data request. It contains data from 2000 to 2015. The dataset has 164 rows and 29 columns.
- **Predictive Attribute**: Our predictive attribute(Target variable y) was the maternal mortality ratio, and we wanted to develop a predictive model based on relevant factors such as healthcare worker density and so on.
- **Data Splitting**: To ensure the model's effectiveness, we randomly split the dataset into a training size and a test size in a 70/30 ratio. The training set will be used to train the model, and the test set will be used to evaluate the model's predictive ability.

SID:500366554

Model Selection

I will use the following two models to predict mortality:

Linear regression model: used to analyze the linear relationship between medical density (doctors) and mortality (maternal). The goal of this model is to find the best fit line that minimizes the error between the predicted value and the actual value.

Polynomial Regression: Polynomial regression is an extension of linear regression that captures nonlinear relationships in data by introducing polynomial features. You can raise the input features to a higher power and then perform linear regression to capture nonlinear trends.

Feature variables (X):

Health worker density (number of PHYSICIAN per 10,000 population):

Describes the accessibility and resource allocation of health services in a specific area.

Target variable (y):

Maternal mortality rate:

The number of mothers who died from pregnancy-related causes per 100,000 live births.

Linear regression:

In the linear regression model, I used the sklearn package. I first read the dataset SDG_goal3_clean using pandas. After that, I set the target variable and feature variables.

These feature variables may be closely related to the distribution of medical staff, which helps us understand how medical resources affect mortality.

Model training and evaluation

Data splitting: Split the dataset into a training set (70%) and a test set (30%) for training the model and evaluating model performance.

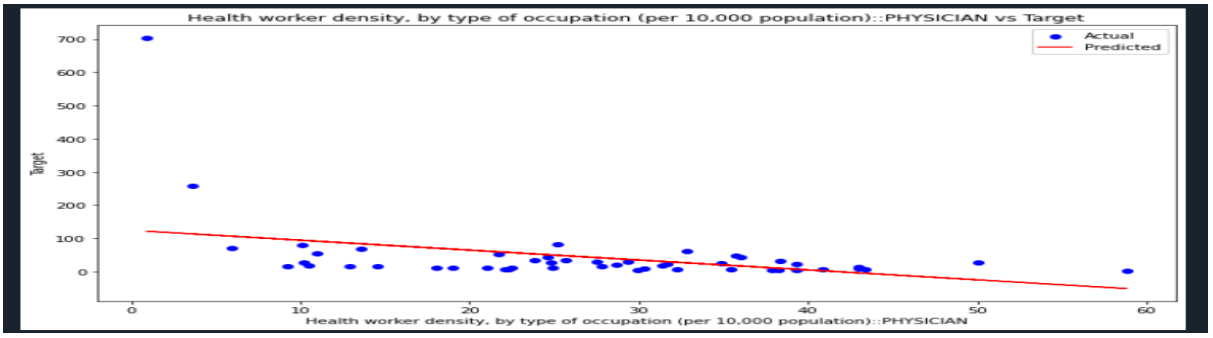
Evaluation metrics:

Mean squared error (MSE): Measures the average squared difference between the predicted and actual values.

Coefficient of determination (R^2 score): Measures the ability of the model to explain the data. The closer the value is to 1, the more effective the model is.

```python
import pandas as pd
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import joblib
import matplotlib.pyplot as plt


data = pd.read_csv('SDG_goal3_clean.csv')

#modle 1 LinearRegression
X = data[['Health worker density, by type of occupation (per 10,000 population)::PHYSICIAN']]
y = data['Maternal mortality ratio']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)
joblib.dump(model, 'linear_regression_model.pkl')
loaded_model = joblib.load('linear_regression_model.pkl')


train_predictions = loaded_model.predict(X_train)
test_predictions = loaded_model.predict(X_test)

train_mse = mean_squared_error(y_train, train_predictions)
test_mse = mean_squared_error(y_test, test_predictions)
r2 = r2_score(y_test, test_predictions)

plt.figure(figsize=(10, 6))
for i, column in enumerate(X.columns):
    plt.subplot(1, len(X.columns), i+1)
    plt.scatter(X_test[column], y_test, color='blue', label='Actual')
    plt.plot(X_test[column], model.predict(X_test), color='red', label='Predicted')
    plt.title(f'{column} vs Target')
    plt.xlabel(column)
    plt.ylabel('Target')
    plt.tight_layout()
plt.legend()
plt.show()

print("Training set mean square error: ", train_mse)
print("Test set mean square error: ", test_mse)
print("R2 score: ", r2)
```

At the same time, I visualize the target. For each feature, draw a scatter plot with the target variable and add a regression line. The actual value of the test set is represented by blue points, and the predicted value is represented by a red line. The relationship between each feature and the target variable is shown in the scatter plot and regression line. The red regression line shows the fit of the model. From the figure, it can be observed that the points are densely distributed and close to the regression line. This shows that the model fits well on the current feature set and there are not many outliers or extreme values in the data.



After model training and evaluation, the mean square error and determination coefficient of the training set and test set are output, which are 16594.73142005619, 8616.830403768248, and 0.18604035279275621, respectively.

The results of the current model show that linear regression has limited predictive power for this data set. Although the scatter plot is close to the regression line, which may indicate that the local data fits better, the overall model still cannot capture more complex patterns.

Possible reasons:

Nonlinear relationship: There may be a nonlinear relationship between the data and the target variable, and the linear regression model cannot capture these complex patterns. Although the scatter plot is close to the regression line on some specific features, the overall performance is still poor.

```
In [9]: runfile('C:/Users/Misooo/Desktop/stage3/未命名1.py', wdir='C:/Users/Misooo/Desktop/stage3')
Training set mean square error:  16594.73142005619
Test set mean square error:  8616.830403768248
R2 score:  0.18604035279275621
```

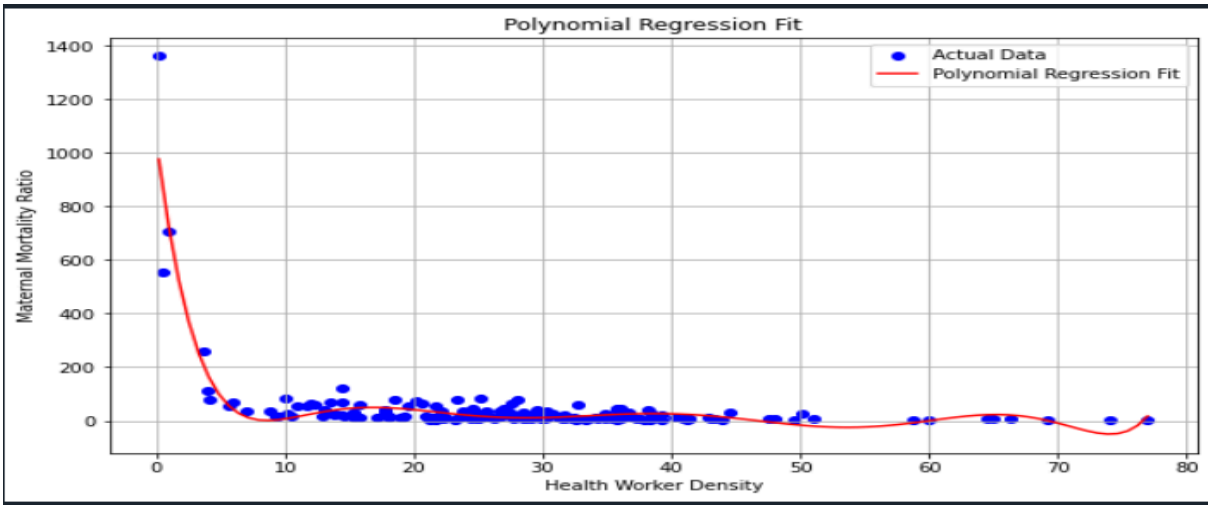
Polynomial Regression:

In the Polynomial Regression model, I used the sklearn packages and the same x, y. Creating a Polynomial Feature Using （degree=8).Because a higher order may result in a better fit of the curve on the training data

```python
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
import numpy as np

poly = PolynomialFeatures(degree=8)
X_poly = poly.fit_transform(X)
model = LinearRegression()
model.fit(X_poly, y)

y_pred = model.predict(poly.transform(X_test))

from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

plt.figure(figsize=(10, 6))


plt.scatter(X, y, color='blue', label='Actual Data')


X_fit = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)
X_fit_poly = poly.transform(X_fit)
y_fit = model.predict(X_fit_poly)

plt.plot(X_fit, y_fit, color='red', label='Polynomial Regression Fit')
plt.title('Polynomial Regression Fit')
plt.xlabel('Health Worker Density')
plt.ylabel('Maternal Mortality Ratio')
plt.legend()
plt.grid()
plt.show()
print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")
```

The blue points in the scatter plot represent the actual data, and the red curve represents the predictions of the polynomial regression model.

By observing the scatter plot I produced below, I can see that the red curve closely follows the blue data points, indicating that the model captures the trend in the data very well.



After training and evaluating the Lasso regression, we get the following output:
Mean Squared Error: 660.7228277317411
R^2 Score: 0.9375870599092779

Analysis of polynomial regression results：

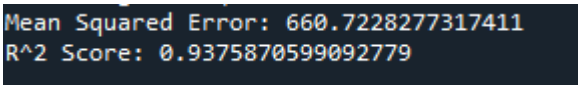
Mean Squared Error: 660.72

The mean squared error is relatively small, indicating that the mean squared error between the model's predictions and the actual values is 660.72, indicating that the model has a low prediction error and performs well.

The R^2 score is 0.9376, indicating that the model can explain about 93.76% of the variance of the target variable. This is a very high score, indicating that the polynomial regression model fits the data very well.

Summary：

The polynomial regression model performs well on this dataset, with very low prediction errors and high explanatory power. This suggests that there may be a complex nonlinear relationship between the density of health workers (physicians) and maternal mortality, which polynomial regression successfully captures.



The polynomial regression model performs well in many aspects, mainly because it not only meets the requirements in terms of prediction effect, but also meets the requirements in terms of fitting ability, model adaptability and interpretability, and model adjustment space. In detail, the core advantage of polynomial regression is that it can capture nonlinear trends in data. Many data sets have nonlinear relationships, and a single linear model often cannot accurately describe such complex relationships. By adding polynomial terms, the model gains greater flexibility to fit these curvilinear trends. At the same time, it is controllable. Since each term (such as linear terms, quadratic terms, cubic terms, etc.) has an independent coefficient, we can understand the role of features of different orders in predicting the target by observing the coefficients of each term. The adaptability of polynomial regression lies in its applicability to a variety of nonlinear tasks. Whether it is a time series problem that requires fitting a curvilinear trend or describing a nonlinear relationship, polynomial regression can be flexibly applied. Polynomial regression performs well in many aspects, especially in terms of nonlinear fitting, model interpretability, controllability, and adaptability. This makes it reasonable to expect that it can not only meet the requirements in terms of prediction accuracy, but also provide balance, adaptability, and interpretability in a range of tasks.

SID：520595273

hliu4477

KNN

I used the knn model and RandomForestRegressor model from the sklearn library to predict maternal mortality based on health care worker density. The core of the knn model is to find the nearest K neighbors through the distance metric, and use the labels of these neighbors to make predictions.

```
In [166]: import pandas as pd
          import numpy as np
          from math import sqrt
          from sklearn import metrics
          from sklearn import neighbors
          from sklearn.ensemble import RandomForestRegressor
          from sklearn.model_selection import train_test_split
          import matplotlib.pyplot as plt
```

```
In [167]: df = pd.read_csv('SDG_goal3_clean.csv')
          X = df[['Health worker density, by type of occupation (per 10,000 population)::NURSEMIDWIFE',
                  'Health worker density, by type of occupation (per 10,000 population)::PHARMACIST']]
          y = df['Maternal mortality ratio']
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=6)
          neigh = neighbors.KNeighborsRegressor(n_neighbors=2).fit(X_train, y_train)
```

First, the pandas library is referenced to process CSV files and calculate the square root of the error with the sqrt square root function in the math module. In sklearn repository import metrics, neighbors, RandomForestRegressor, used to construct the variance, R square, KNN, RF, matplotlib. At the same time pyplot and numpy used to deal with the drawing of various graphics and multidimensional arrays. The CSV file of SDG_goal3_clean.csv is then read using the read_csv function and placed in a variable named df. In the df, we split two characteristics, namely the density of health workers (per 10,000 population) of midwives and pharmacists, and assigned them a value of X. The Maternal Mortality Ratio is the target that the regression model needs to predict and assign it a value of Y.

The train_test_split function was used to divide the datasets X and y into training and test sets, where 70% was used as the training set and 30% of the data was assigned to the test set. random_state ensures that the partitioning of the dataset is repeatable. The KNeighborsRegressor function creates a data model with k 2, and fit(X_train, y_train) applies the model to the training data.

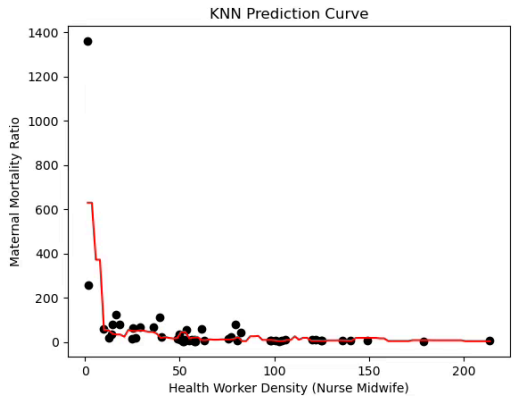
```
y_pred = neigh.predict(X_test)
mse = metrics.mean_squared_error(y_test, y_pred)
print('Root mean squared error (RMSE):', sqrt(mse))
print('R-squared score:', metrics.r2_score(y_test, y_pred))
```

```
Root mean squared error (RMSE): 119.5561092763817
R-squared score: 0.6162047428248281
```

Next, I predict the test set X_test and save the result in y_pred, and calculate the average of the squared difference between the predicted value and the true value by the meters.mean_squared_error function. Using the sqrt(mse) function to calculate the square root of MSE to obtain RMSE, we obtain RMSE of 119.56, which is a large value, indicating that the model prediction is not close to the true value. The r2_score function of scikit-learn is used to

calculate the score of R-squared. The final R square score was close to 0.62, indicating that the model explained only about 62% of the variation in the target variable.

```
']: X_fit = np.linspace(X_test['Health worker density, by type of occupation (per 10,000 population)::NURSEMIDWIFE'].min
                        X_test['Health worker density, by type of occupation (per 10,000 population)::NURSEMIDWIFE'].max
                        100).reshape(-1, 1)
    X_fit_full = np.hstack([X_fit, np.full_like(X_fit, X_test['Health worker density, by type of occupation (per 10,000
    y_fit = neigh.predict(X_fit_full)

    plt.scatter(X_test['Health worker density, by type of occupation (per 10,000 population)::NURSEMIDWIFE'],
                y_test, color='black',)
    plt.plot(X_fit, y_fit, color='red')
    plt.xlabel('Health Worker Density (Nurse Midwife)')
    plt.ylabel('Maternal Mortality Ratio')
    plt.title('KNN Prediction Curve')
    plt.show()
```



Then draw an image containing the actual and predicted values, and use np.linspace() function to generate 100 equally spaced values at the minimum and maximum values of midwife density, because the KNN model requires a two-dimensional array as input.reshape(-1, 1) reshape the generated sequence into a two-dimensional array. The average value of midwife density and pharmacist density was used to create a complete input data matrix using np.full_like(), and the previously trained model was used to predict X_fit_full using the neigh.predict function to obtain the predicted value of maternal mortality rate. Scatter plots of actual values were next plotted with plt.scatter, where the X-axis is midwife density and the Y-axis is actual maternal mortality. The scatter color was set to black and the curve color was set to red. Then set the corresponding axis label, and set the title to "KNN prediction curve" with plt.title().

According to the image, when the density of midwives is close to 0, the maternal mortality rate is significantly higher. With the increase of the density of midwives, the maternal mortality rate drops sharply. When the density of midwives is close to 50, the maternal mortality rate is stable and maintained at a low level.

The black data points show the true maternal mortality rate, and the regions with higher density of midwives are closer to the red predicted line of the KNN model. However, the density of midwives was close to zero, and some of the actual value points were much higher than the predicted curve, indicating that the prediction of the model in this region underestimated the mortality rate.

Advantages of KNN:

The Knn model only stores data at training time, and does not really build the model, which is suitable for those scenarios where predictions are generated quickly. And because knn can handle non-linear data distribution, knn has high flexibility and can be applied to complex scenarios.

Disadvantages of KNN:

From the results, I get a low R-squared value, which can only explain 62% of the variation of the target variable, and the prediction effect is relatively poor. Models do not accurately capture important patterns in the data. In addition, a higher RMSE means that the average error between the predicted value of the model and the actual value is large, indicating that the prediction accuracy is low. At the same time, different k values in the knn model are easily affected by noise, resulting in reduced accuracy. Because the range of feature values is quite different, it is easy to lead to the unbalanced weight of knn on features. In addition, it also brings high computational cost and low computational efficiency.

Random Forest Regressor

```
In [161]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=6)

          model = RandomForestRegressor(n_estimators=100, random_state=6)
          model.fit(X_train, y_train)

Out[161]:       RandomForestRegressor
          RandomForestRegressor(random_state=6)
```

Based on the poor performance of the KNN model, I attempted to construct the relationship between midwife and pharmacist density and maternal mortality using Random Forest Regressor. The model constructs multiple decision trees to make predictions, and finally averages the prediction results of these trees to improve the accuracy and robustness of the model. n_estimators are used to specify 100 decision trees to be included in the random forest. The random_state function specifies a random seed 6 to ensure reproducible results.

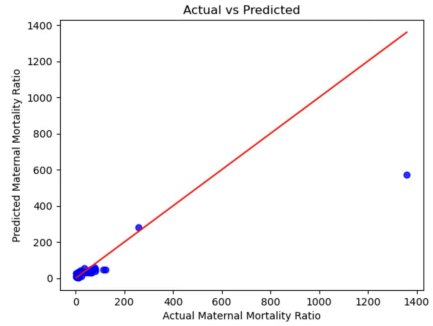
```
y_pred = model.predict(X_test)
mse = metrics.mean_squared_error(y_test, y_pred)
print('Root mean squared error (RMSE):', sqrt(mse))
print('R-squared score:', metrics.r2_score(y_test, y_pred))
```

```
Root mean squared error (RMSE): 114.4896041444063
R-squared score: 0.6480441692275318
```

Firstly, I use the trained Random Forest Regressor model to predict the test set X_test and store it in y_pred. The root mean square error was calculated using mean_squared_error, which measures the mean squared difference between the predicted values of the model and the actual values. Then sqrt(mse)) was used to obtain RMSE=114.49. This value is slightly better than KNN, indicating that the prediction error of the model is small. Using the r2_score() function we get an R squared value close to 0.648, meaning that about 64.8% of the data are correctly predicted.

```
y_pred = model.predict(X_test)
plt.scatter(y_test, y_pred, color='blue', alpha=0.8)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
plt.xlabel('Actual Maternal Mortality Ratio')
plt.ylabel('Predicted Maternal Mortality Ratio')
plt.title('Actual vs Predicted')
plt.show()
```

Next, draw the scatter plot of the actual value and the predicted value, and set the color of the scatter plot to blue. alpha sets the transparency of the points, so that the overlapping points can be clearly displayed. A straight line is drawn based on the y_test values, and this line represents the distribution of data points when, ideally, the predicted and actual values are exactly equal



We can see from the image that most of the blue points are clustered in the lower left corner, indicating that the maternal mortality rate is 0-100, and the model prediction performance is good. However, a blue point is far from the red line when it is close to 1400, indicating that the model prediction error is very large at this time, which also means that the model is not handling outliers well.

Model Selection: KNN vs Random Forest Regressor

I used two different models for machine learning, aiming to predict maternal mortality. In contrast, Random Forest Regressor model has stronger anti-overfitting ability, and it does not need to normalize or standardize the data, which is more in line with the needs of this study, and its performance and expressivity exceed KNN. Specifically, the RMSE of KNN was 119.56, and the R^2 Score was 0.62. The RMSE of Random Forest Regressor was 114.49, and the R^2 Score was 0.65. A smaller RMSE indicates a smaller error between the predicted value and the actual value, and the closer the R^2 Score value is to 1, the better the model is. In conclusion, Random Forest Regressor performs better than KNN for predicting maternal mortality through health care worker density.

limitation:

If there are outliers in the data set, these outliers can significantly improve the RMSE and thus have a large impact on model evaluation. There may be noise in the data, which may be derived from measurement errors or data input errors, and noise will have a greater impact on the sensitive model.
The limitation of data size, if the dataset size is too small, the model may not be able to learn enough patterns from the data, easily lead to overfitting, and the generalization ability of the model on new data is weak.

SID：530649584

I'd like to use Decision Tree Regression and XGBoost Regression to predict this dataset.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import root_mean_squared_error, r2_score
```

I first established essential libraries and functions for building and evaluating the regression models.

```
df = pd.read_csv("SDG_goal3_clean.csv")
df.head()
```

	Country	Year	Region	Maternal mortality ratio	Proportion of births attended by skilled health personnel (%)
0	Albania	2000	Europe	23	99.1
1	Armenia	2000	Asia	43	96.8
2	Armenia	2005	Asia	35	97.8
3	Armenia	2010	Asia	32	99.5
4	Australia	2000	Oceania	7	99.3

Then, just like my previous work in stage 2, I used pandas to read the dataset as a pandas data frame (df) to get a brief overview of the data.

```
features = [
    'Proportion of births attended by skilled health personnel (%)',
    'Infant mortality rate (deaths per 1,000 live births):::BOTHSEX',
    'Infant mortality rate (deaths per 1,000 live births):::MALE',
    'Infant mortality rate (deaths per 1,000 live births):::FEMALE',
    'Under-five mortality rate, by sex (deaths per 1,000 live births):::BOTHSEX',
    'Under-five mortality rate, by sex (deaths per 1,000 live births):::MALE',
    'Under-five mortality rate, by sex (deaths per 1,000 live births):::FEMALE',
    'Neonatal mortality rate (deaths per 1,000 live births)',
    'Mortality rate attributed to cardiovascular disease, cancer, diabetes or chronic respiratory disease (probability):::BOTHSEX',
    'Mortality rate attributed to cardiovascular disease, cancer, diabetes or chronic respiratory disease (probability):::MALE',
    'Mortality rate attributed to cardiovascular disease, cancer, diabetes or chronic respiratory disease (probability):::FEMALE',
    'Fraction of deaths due to diabetes, among deaths due to cardiovascular disease, cancer, diabetes or chronic respiratory disease',
    'Fraction of deaths due to cancer, among deaths due to cardiovascular disease, cancer, diabetes or chronic respiratory disease',
    'Suicide mortality rate, by sex (deaths per 100,000 population):::BOTHSEX',
    'Suicide mortality rate, by sex (deaths per 100,000 population):::MALE',
    'Suicide mortality rate, by sex (deaths per 100,000 population):::FEMALE',
    'Death rate due to road traffic injuries, by sex (per 100,000 population):::BOTHSEX',
    'Death rate due to road traffic injuries, by sex (per 100,000 population):::MALE',
    'Death rate due to road traffic injuries, by sex (per 100,000 population):::FEMALE',
    'Adolescent birth rate (per 1,000 women aged 15-19 years)',
    'Universal health coverage (UHC) service coverage index',
    'Health worker density, by type of occupation (per 10,000 population)::PHYSICIAN',
    'Health worker density, by type of occupation (per 10,000 population)::NURSEMIDWIFE',
    'Health worker density, by type of occupation (per 10,000 population)::PHARMACIST'
]

target = 'Maternal mortality ratio'

correlation = df[features + [target]].corr()[target].sort_values(ascending=False)
print("Correlation with Maternal Mortality Ratio:\n", correlation)
```

Next, I defined a list of features. I regarded all the variables except our target variable, Maternal Mortality Ratio, as the independent variables and computed their correlation using the .corr() function. I also sorted the values in descending order to let the readers compare them more directly.

```
Correlation with Maternal Mortality Ratio:
 Maternal mortality ratio                                                                                 1.000000
Under-five mortality rate, by sex (deaths per 1,000 live births):::FEMALE                                0.811262
Under-five mortality rate, by sex (deaths per 1,000 live births):::BOTHSEX                               0.794159
Under-five mortality rate, by sex (deaths per 1,000 live births):::MALE                                  0.779171
Infant mortality rate (deaths per 1,000 live births):::FEMALE                                            0.729071
Infant mortality rate (deaths per 1,000 live births):::BOTHSEX                                           0.711988
Infant mortality rate (deaths per 1,000 live births):::MALE                                              0.697461
Adolescent birth rate (per 1,000 women aged 15-19 years)                                                 0.632735
Neonatal mortality rate (deaths per 1,000 live births)                                                   0.589432
Death rate due to road traffic injuries, by sex (per 100,000 population):::FEMALE                        0.528382
Death rate due to road traffic injuries, by sex (per 100,000 population):::BOTHSEX                       0.343119
Death rate due to road traffic injuries, by sex (per 100,000 population):::MALE                          0.276383
Mortality rate attributed to cardiovascular disease, cancer, diabetes or chronic respiratory disease (probability):::FEMALE    0.271695
Fraction of deaths due to diabetes, among deaths due to cardiovascular disease, cancer, diabetes or chronic respiratory disease    0.171974
Mortality rate attributed to cardiovascular disease, cancer, diabetes or chronic respiratory disease (probability):::BOTHSEX    0.164511
Mortality rate attributed to cardiovascular disease, cancer, diabetes or chronic respiratory disease (probability):::MALE      0.073324
Suicide mortality rate, by sex (deaths per 100,000 population):::FEMALE                                 -0.106280
Suicide mortality rate, by sex (deaths per 100,000 population):::MALE                                   -0.113899
Suicide mortality rate, by sex (deaths per 100,000 population):::BOTHSEX                                -0.116024
Fraction of deaths due to cancer, among deaths due to cardiovascular disease, cancer, diabetes or chronic respiratory disease    -0.230950
Health worker density, by type of occupation (per 10,000 population)::PHARMACIST                        -0.280467
Health worker density, by type of occupation (per 10,000 population)::NURSEMIDWIFE                      -0.295435
Health worker density, by type of occupation (per 10,000 population)::PHYSICIAN                         -0.345347
Universal health coverage (UHC) service coverage index                                                   -0.494257
Proportion of births attended by skilled health personnel (%)                                            -0.679962
Name: Maternal mortality ratio, dtype: float64
```

These are the results after I printed the correlations. I selected the highest eight correlations in which the relationship between the target variable and independent variables is at least greater than 0.589.

```
X = df
y = df['Maternal mortality ratio']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

I partition my dataset into training and testing sets in these codes before developing a machine-learning model. First, I set X to represent my complete data frame, including all the features. Then, I assign the 'Maternal mortality ratio' column to y as my target variable, representing the outcome I wish to forecast.

Next, I use the train_test_split function to divide the data into training and testing sets, ensuring that 30% is reserved for testing (test_size=0.3) and the rest for training. By specifying random_state=42, I ensure that the data split is consistent each time I execute this code. Finally, I divide the resulting data into four files: X_train, X_test, y_train, and y_test, which represent the features and goals for both training and testing. This allows me to train and assess my model on the training and testing sets.

Measurements:
In my model evaluation, I used two metrics: RMSE and R^2. RMSE, or Root Mean Squared Error, allowed me to directly measure the difference between my model's predictions and the actual values. Since RMSE has the same unit as my target variable, it gave me an intuitive sense of how significant the errors were. For example, if I was predicting mortality rates, the RMSE would also be expressed in mortality rates, making it easy to interpret. I aimed to minimise the RMSE, as a lower value would indicate that my model's predictions were more accurate.
On the other hand, R^2, or the coefficient of determination, helped me understand how well my model explained the variability in the target variable. An R^2 value of 1 would mean that my model perfectly described all the changes in the target variable, while a value close to 0 would suggest that my model had little explanatory power. If R^2 were negative, it would indicate that my model performed worse than simply predicting the mean. By looking at these two metrics together, I could better evaluate how well my model performed regarding both absolute prediction error (RMSE) and its overall explanatory power (R^2).

Decision Tree Regression:

```
from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor(random_state=42)
model.fit(X_train[['Under-five mortality rate, by sex (deaths per 1,000 live births):::FEMALE',
        'Under-five mortality rate, by sex (deaths per 1,000 live births):::BOTHSEX',
        'Under-five mortality rate, by sex (deaths per 1,000 live births):::MALE',
        'Infant mortality rate (deaths per 1,000 live births):::FEMALE',
        'Infant mortality rate (deaths per 1,000 live births):::BOTHSEX',
        'Infant mortality rate (deaths per 1,000 live births):::MALE',
        'Adolescent birth rate (per 1,000 women aged 15-19 years)',
        'Neonatal mortality rate (deaths per 1,000 live births)']], y_train)

y_pred = model.predict(X_test[['Under-five mortality rate, by sex (deaths per 1,000 live births):::FEMALE',
        'Under-five mortality rate, by sex (deaths per 1,000 live births):::BOTHSEX',
        'Under-five mortality rate, by sex (deaths per 1,000 live births):::MALE',
        'Infant mortality rate (deaths per 1,000 live births):::FEMALE',
        'Infant mortality rate (deaths per 1,000 live births):::BOTHSEX',
        'Infant mortality rate (deaths per 1,000 live births):::MALE',
        'Adolescent birth rate (per 1,000 women aged 15-19 years)',
        'Neonatal mortality rate (deaths per 1,000 live births)']])
rmse = root_mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Decision Tree Regression - RMSE: {rmse}, R²: {r2}')
```

I used decision tree regression because it is well-suited for handling complex, non-linear relationships, such as those found in mortality rate data. Decision trees work by recursively splitting the data into smaller subsets and choosing the best split based on how well it separates the data. The predictions come from the average values in the leaf nodes at the end of these splits. I chose this method because of its interpretability—it's easy to understand how the model makes decisions based on different features. However, I also had to be mindful of overfitting, which is a common problem with decision trees. Without controlling the depth of the tree or applying pruning, the model could become too complex and perform poorly on new data. So, I paid close attention to pruning the model to ensure it didn't overfit while still capturing the non-linear relationships in the data. First, I imported the DecisionTreeRegressor from sklearn.tree and initialised it with a fixed random_state=42 to ensure reproducibility. I trained the model using the fit method on selected features from my training dataset (X_train), which included mortality rates and birth statistics, with the target being y_train. After fitting the model, I made predictions on the test set (X_test) using the prediction method for the same features. To evaluate the model's performance, I calculated the Root Mean Squared Error (RMSE) and the R-squared (r2) value, which was stored in rmse and r2. Finally, I printed the results to observe how well the model performed.

```
model = DecisionTreeRegressor(max_depth=5, random_state=42)
model.fit(X_train[['Under-five mortality rate, by sex (deaths per 1,000 live births):::FEMALE',
        'Under-five mortality rate, by sex (deaths per 1,000 live births):::BOTHSEX',
        'Under-five mortality rate, by sex (deaths per 1,000 live births):::MALE',
        'Infant mortality rate (deaths per 1,000 live births):::FEMALE',
        'Infant mortality rate (deaths per 1,000 live births):::BOTHSEX',
        'Infant mortality rate (deaths per 1,000 live births):::MALE',
        'Adolescent birth rate (per 1,000 women aged 15-19 years)',
        'Neonatal mortality rate (deaths per 1,000 live births)']], y_train)

y_pred = model.predict(X_test[['Under-five mortality rate, by sex (deaths per 1,000 live births):::FEMALE',
        'Under-five mortality rate, by sex (deaths per 1,000 live births):::BOTHSEX',
        'Under-five mortality rate, by sex (deaths per 1,000 live births):::MALE',
        'Infant mortality rate (deaths per 1,000 live births):::FEMALE',
        'Infant mortality rate (deaths per 1,000 live births):::BOTHSEX',
        'Infant mortality rate (deaths per 1,000 live births):::MALE',
        'Adolescent birth rate (per 1,000 women aged 15-19 years)',
        'Neonatal mortality rate (deaths per 1,000 live births)']])
rmse = root_mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Pruned Decision Tree Regression - RMSE: {rmse}, R²: {r2}')
```

In the second version of the code, I applied pruning by setting the max_depth of the decision tree to 5, which helped reduce overfitting by controlling the tree's complexity. I followed the same process: fitting the pruned model, making predictions, and evaluating its performance using RMSE and R-squared values. I then printed these results to compare the performance of the pruned decision tree with the original, unpruned model.

```
Decision Tree Regression - RMSE: 79.64192824570408, R²: 0.40084544908684616
Pruned Decision Tree Regression - RMSE: 79.60192710901643, R²: 0.4014471634046738
```

After running both the unpruned and pruned decision tree models, I compared the results. For the unpruned model, the RMSE was 79.64, and the R^2 value was 0.4008. This suggests that while the model could explain about 40% of the variability in the target variable, the error margin in predictions was relatively high. However, when I pruned the decision tree (setting max_depth=5), the RMSE slightly decreased to 79.60, and the R^2 improved marginally to 0.4014. These results indicate that pruning made the model a bit more accurate, with a slight reduction in prediction error and a small improvement in the explanatory power. However, the change between the two models is minimal, implying that reducing the model complexity through pruning did not significantly impact performance. This suggests that the unpruned model was not heavily overfitted, and pruning didn't drastically improve accuracy.

XGBoost Regression:

Therefore, I used another method to get a better result. I chose XGBoost because of its powerful performance and ability to handle complex, multi-dimensional health data. XGBoost works by incrementally building a series of weak learners, typically decision trees, and optimising the overall error through gradient boosting. It is also vital in managing missing values and high-dimensional datasets, which is particularly useful when working with health data that may be incomplete or noisy. When I applied XGBoost to predict maternal mortality rates, I found it well-suited for this data type. The algorithm is designed to handle noise efficiently, and outliers are present in health-related datasets. Moreover, XGBoost's fast training speed allowed me to quickly iterate and fine-tune the model, which is essential when dealing with large and complex data. Overall, XGBoost provided a robust and efficient solution for my prediction task.

```
from xgboost import XGBRegressor

model = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
model.fit(X_train[['Under-five mortality rate, by sex (deaths per 1,000 live births):::FEMALE',
        'Under-five mortality rate, by sex (deaths per 1,000 live births):::BOTHSEX',
        'Under-five mortality rate, by sex (deaths per 1,000 live births):::MALE',
        'Infant mortality rate (deaths per 1,000 live births):::FEMALE',
        'Infant mortality rate (deaths per 1,000 live births):::BOTHSEX',
        'Infant mortality rate (deaths per 1,000 live births):::MALE',
        'Adolescent birth rate (per 1,000 women aged 15-19 years)',
        'Neonatal mortality rate (deaths per 1,000 live births)']], y_train)

y_pred = model.predict(X_test[['Under-five mortality rate, by sex (deaths per 1,000 live births):::FEMALE',
        'Under-five mortality rate, by sex (deaths per 1,000 live births):::BOTHSEX',
        'Under-five mortality rate, by sex (deaths per 1,000 live births):::MALE',
        'Infant mortality rate (deaths per 1,000 live births):::FEMALE',
        'Infant mortality rate (deaths per 1,000 live births):::BOTHSEX',
        'Infant mortality rate (deaths per 1,000 live births):::MALE',
        'Adolescent birth rate (per 1,000 women aged 15-19 years)',
        'Neonatal mortality rate (deaths per 1,000 live births)']])
rmse = root_mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'XGBoost Regression - RMSE: {rmse}, R²: {r2}')
```

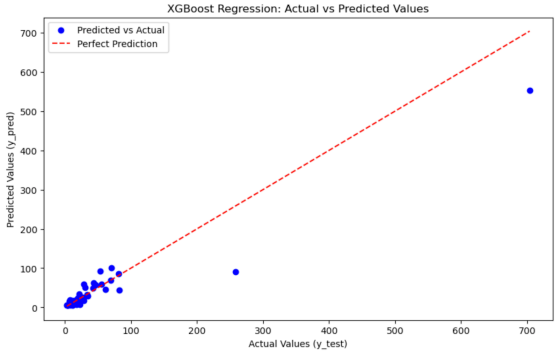
```
XGBoost Regression - RMSE: 34.350455451901944, R²: 0.8885396688313288
```

In this code, I implemented an XGBoost regression model to predict mortality-related features. I began by importing the XGBRegressor from the XGBoost library. I set the model parameters such as n_estimators=100 (which controls the number of boosting rounds) and learning_rate=0.1 to regulate the contribution of each tree. I used random_state=42 to ensure reproducibility. Next, I trained the model

using the fit method on selected features from my training dataset (X_train), which included under-five mortality rates, infant mortality rates, and adolescent birth rates, with the target being y_train. After training the model, I made predictions on the test set (X_test) using the same set of features. To evaluate the performance of my model, I calculated the Root Mean Squared Error (RMSE) and the R-squared (R^2) value, storing them in rmse and r2 respectively. The RMSE was 34.35, and the R^2 was 0.8885, which indicates that the model performed exceptionally well. The low RMSE means the model had a relatively small prediction error, and the high R^2 value (close to 1) shows that the model could explain a significant portion of the variance in the target variable. Overall, the XGBoost regression model provided a more solid predictive performance for this dataset than the Decision Tree regression model.

```
import matplotlib.pyplot as plt
import numpy as np

plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue', label='Predicted vs Actual')

plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--', label='Perfect Prediction')

plt.xlabel('Actual Values (y_test)')
plt.ylabel('Predicted Values (y_pred)')

plt.title('XGBoost Regression: Actual vs Predicted Values')

plt.legend()

plt.show()
```

I created a scatter plot in these codes to compare the actual values (y_test) with the predicted values (y_pred) from my XGBoost regression model. I plotted the actual vs. predicted values as blue points and added a red dashed line to represent perfect predictions. The axes are labelled, and I included a title and a legend for clarity. This plot helps me visually assess how well the projections align with the actual values by showing how close the blue points are to the red line. And here is the plot:



In this plot, I compared the predicted values from my XGBoost regression model with the actual values from the test set. The blue dots represent the predicted values plotted against the actual values (y_test), and the red dashed line represents a "perfect prediction," where the predicted values would exactly match the actual values. Ideally, if the model was perfect, all the blue dots would align with the red line. In this case, most of the blue dots cluster around the line, especially for lower values, indicating that the model performed well predicting those values. However, there are a few outliers, particularly one large blue dot far from the red line, which suggests that the model struggled with some extreme values. Overall, the model's predictions are generally close to the actual values, but there are some instances where it was less accurate.

SID:530333591

Model selection:

1. **logistic regression analysis**
2. **Naive Bayes**

logistic regression analysis:

First, I take SDG_goal3_clean as my dataset, which contains 28 columns and 163 rows. To begin with my model analysis, the first model I choose is logistic regression analysis which is often used to fix questions in category divided, especially in Binary Classification which often contain only 2 answers. And our goal is to predict 2 possible outcomes, like whether success or not, or ill or not.

The dataset I take is column D "Maternal mortality ratio" as dependent variable(Y) and column AB"Health worker density, by type of occupation (per 10,000 population)::PHARMACIST" as Independent variable(X). The dependent variable is the variable we need to predict.

However, we can not directly predict this dependent variable by using logistic regression analysis because this variable is not a category one. So we are trying to create a specific value to define. In this analysis, we try to use the median value, if the value is higher than the median we call it "High Maternal mortality ratio" as category 1 and if not we call it "low Maternal mortality ratio" which is category 0, as shown on code in line 11, 12.

Then what we need to do is Remove missing value and Define features and target variables, as shown in line 14 to line 17. And then the key part of the whole analysis is to train the model,

```
1  from sklearn.model_selection import train_test_split
2  from sklearn.linear_model import LogisticRegression
3  from sklearn.metrics import classification_report
4  import numpy as np
5  import pandas as pd
6
7  df = pd.read_csv('/Users/kyrie/Documents/悉尼及准备/学校资料/data1002/SDG_goal3_clean.csv')
8
9  target_col = 'Maternal mortality ratio'
10 predictor_col = 'Health worker density, by type of occupation (per 10,000 population)::PHARMACIST'
11 median_value = df[target_col].median()
12 df['target_binary'] = np.where(df[target_col] > median_value, 1, 0)
13
14 df_clean = df.dropna(subset=[predictor_col, 'target_binary'])
15
16 X = df_clean[[predictor_col]]
17 y = df_clean['target_binary']
```

**So first step try to separate the dataset into a training set and a test set. In this dataset, I divided it into 70% and 30%. By using a training set we can train the model to improve its efficiency as we can and at last to test how accurate our model accuracy is we need to use a test model. After divided, we use function "LogisticRegression()
" to train our logistic regression model. These all steps are shown on code line 19 to line 22.**

Last, we should predict and evaluate the model and print the result.

```
18
19  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
20
21  logreg = LogisticRegression()
22  logreg.fit(X_train, y_train)
23
24  y_pred = logreg.predict(X_test)
25  report = classification_report(y_test, y_pred)
26  print(report)
```

Summary of result:

In logistic regression analysis, the result can be simply concluded by precision and recall rate, which means the proportion of predicted positive samples that are actually positive and the proportion of all actual positive samples that model correctly identifies. Precision cares more about the accuracy of positive predictions. Recall cares more about the completeness of capturing all positive case

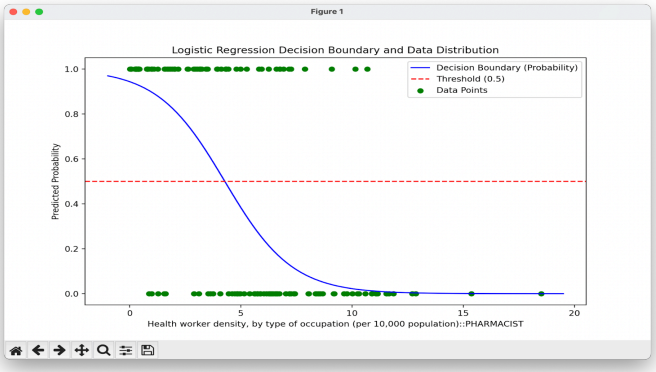
```
              precision    recall  f1-score   support

           0       0.60      0.83      0.70        18
           1       0.88      0.68      0.76        31

    accuracy                           0.73        49
   macro avg       0.74      0.76      0.73        49
weighted avg       0.77      0.73      0.74        49
```

Conclusion:

In precision: Category 0(low death rate):60% Category 1(high death rate):88%
In Recall: Category 0(low death rate):83% Category 1(high death rate): 68%

Overall Accuracy: 73%

The precision of high death rate is 88% is very high, which means the model tends to accurately predict samples it considers to have high mortality, but it may miss some samples that are actually high ratio. The precision of low death rate is 60% which means this model is able to identify most of the low mortality samples, but the precision is relatively not high (60%), indicating that some samples were incorrectly predicted as low mortality. And its overall Accuracy is 73%,which means this model effective.



Visualization:

The blue curve (decision boundary) represents the logistic regression model's predicted probability of "Health worker density, by type of occupation :PHARMACIST." The red dashed line indicates the classification threshold. The green dots show the actual values of each sample in the dataset. From the decision boundary, we can see that samples with lower values tend to be predicted as the positive class, while samples with higher values are predicted as the negative class.It shows the monotonic nature of logistic regression.

Model 2: Naive Bayes

First, I take SDG_goal3_clean as my dataset, which contains 28 columns and 163 rows. To begin with my model analysis, the second model I choose is Naive Bayes, which is typically used for handling discrete features or text data, making it suitable for classification tasks. And our goal is to predict 3 possible outcomes, which I divided into low, middle, high Maternal mortality ratio.

The dataset I take is column D "Maternal mortality ratio" as dependent variable(Y) and column AB"Health worker density, by type of occupation (per 10,000 population)::PHARMACIST" and column E "Proportion of births attended by skilled health personnel (%)" as Independent variables(X). The dependent variable is the variable we need to predict. Then what we need to do is Remove missing values and Define features and target variables.

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, confusion_matrix
import pandas as pd


df = pd.read_csv('/Users/kyrie/Documents/悉尼及准备/学校资料/data1002/SDG_goal3_clean.csv')
target_col = 'Maternal mortality ratio'
feature_cols = [
    'Proportion of births attended by skilled health personnel (%)',
    'Health worker density, by type of occupation (per 10,000 population)::PHARMACIST'
]

df = df.dropna(subset=[target_col] + feature_cols)
```

To predict 3 possible outcomes, we are trying to create a specific value to define. In this analysis, we try to use a low, middle, high Maternal mortality ratio. if the value in the highest third quartile we called it high, so the same with low and middle and then defined it into set 1,2,3, and initialized X, Y, shown on code below.

```
df['target_class'] = pd.qcut(df[target_col], q=3, labels=[1, 2, 3])

X = df[feature_cols]
y = df['target_class']
```

Then, separate the dataset into a training set and a test set. In this dataset, I divided it into 70% and 30%. And initialized Bayes, train model and at last forecast the value.

```
23  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
24
25  nb = GaussianNB()
26
27  nb.fit(X_train, y_train)
28
29
30  y_pred = nb.predict(X_test)
```

At last, we need to print the value and the matrix.

```
33  print("Classification Report:")
34  print(classification_report(y_test, y_pred))
35
36  print("Confusion Matrix:")
37  print(confusion_matrix(y_test, y_pred))
38
```

Summary of result:

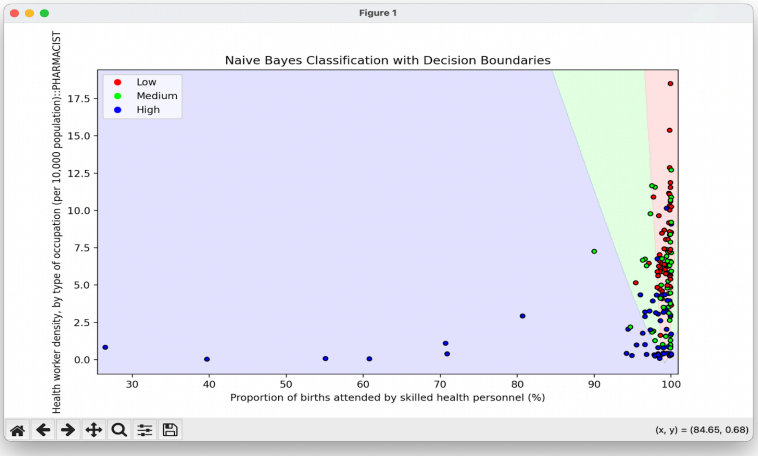
In Naive Bayes, the result can be simply concluded by precision and recall rate, which means the proportion of predicted positive samples that are actually positive and the proportion of all actual positive samples that model correctly identifies. Precision cares more about the accuracy of positive predictions. Recall cares more about the completeness of capturing all positive cases.

```
Classification Report:
              precision    recall  f1-score   support

           1       0.32      0.82      0.46        11
           2       0.40      0.19      0.26        21
           3       0.73      0.47      0.57        17

    accuracy                           0.43        49
   macro avg       0.48      0.49      0.43        49
weighted avg       0.50      0.43      0.41        49

Confusion Matrix:
[[ 9  2  0]
 [14  4  3]
 [ 5  4  8]]
```

Conclusion and visualization:

In precision: Category 1(low Maternal mortality ratio):32% Category 2(middle Maternal mortality ratio):40% Category Category 3(high Maternal mortality ratio):73%

In Recall: Category 1(low Maternal mortality ratio):82% Category 2(middle Maternal mortality ratio):19% Category Category 3(high Maternal mortality ratio):47% The accuracy was increased to 48%. Since the prediction accuracy of high maternal mortality was increased to 0.73 and low Maternal mortality ratio in recall is *2%, both of them are very high, which shows that the model is more effective in the identification of low maternal mortality.



Here is the visualization graph: The Naive Bayes classifier divided the feature space into three distinct regions, corresponding to the target variable's low (red), medium (green), and high (blue). Most samples are concentrated in the higher value area of the features, which means that, in most countries or regions, a high proportion of births are attended by skilled health personnel.

Part B: Advantages and disadvantages of each model

Linear Regression Model:

On the one hand, the strengths include the regression line, which made it simple to visualise the relationship it tries to capture. This model was quick to set up and computationally light, which was a good starting point for my analysis. On the other hand, weaknesses include a low R square of 0.186 and a high MSE, which limit the accuracy of my model and explain only a tiny portion of the variance. The model struggles with the dataset's complexity, as shown by its poor fit to the data points. Although linear regression was a good first approach, we recognise it needs more flexibility to capture more complex patterns in this dataset.

Polynomial Regression Model:

On the one hand, the polynomial regression model's strengths include its ability to capture complex nonlinear relationships, as indicated by the low mean squared error (660.72) and a high R square score of 0.9376. This suggests the model performs very well, explaining over 93% of the variance in the target variable. It also offers flexibility in fitting curvilinear trends, making it adaptable to a wide range of nonlinear data.

On the other hand, polynomial regression can sometimes lead to overfitting if the model becomes too complex with higher-degree terms. While it fits the current dataset well, it may need to be more generalisable to unseen data, and care must be taken to balance model complexity. So, overall, polynomial regression is a powerful tool for capturing nonlinear relationships, but it requires careful tuning to avoid overfitting and maintain generalizability across different datasets.

KNN Model:

On the one hand, the strengths of the KNN model include its ability to find the nearest neighbours through the distance metric, providing a simple and intuitive approach to predictions. The method is also easy to interpret, as it relies on actual data points to make predictions rather than building a complex model. On the other hand, weaknesses include the high RMSE value of 119.56, indicating that the predictions could be more accurate, and the model only explains about 62% of the variance in the target variable (R square = 0.62). This suggests that the model needs help with larger datasets or more complex patterns, especially in cases where the relationship between features and the target needs to be captured better by nearest neighbours. While KNN provides an interpretable and straightforward way to make predictions, it may need to improve with this dataset due to its limited predictive accuracy and difficulty handling complex relationships.

Random Forest Regressor Model:

On the one hand, the strengths of the Random Forest Regressor include its ability to average predictions from multiple decision trees, improving both accuracy and robustness. With 100 decision trees, the model achieved an RMSE of 114.49, which is an improvement over the KNN model, indicating a lower prediction error. Additionally, the R square value of 0.648 means that the model could explain about 64.8% of the variance in the data, showing reasonable predictive power. On the other hand, while the RMSE value is better than that of KNN, it is still relatively high, suggesting that the model has room for improvement. While decent, The R square value is not outstanding, so the model could still struggle with capturing more complex relationships in the data. Overall, the Random Forest Regressor offers improved performance over KNN, providing better accuracy through its ensemble approach, but it still leaves room for further enhancements in predictive accuracy.

Logistic Regression Model:

On the one hand, the logistic regression model's strengths include a high precision of 88% for the high mortality rate category, indicating that the model can accurately identify

most cases where the mortality rate is high. Additionally, the model achieves an overall accuracy of 73%, making it reasonably effective in classifying the data. On the other hand, weaknesses include a lower precision of 60% for the low mortality rate category, meaning that the model needs to identify some samples with low mortality and misclassify them correctly. Furthermore, the recall for the high-mortality category is only 68%, suggesting that the model needs to catch up on some high-mortality cases. Overall, while the logistic regression model effectively identifies high-mortality cases with great precision, it has limitations in accurately classifying low-mortality cases and ensuring that all high-mortality cases are captured.

Naive Bayes Model:

On the one hand, the Naive Bayes model's strengths include its ability to identify high maternal mortality cases with a precision of 73% and a recall of 47%. Additionally, the recall for low maternal mortality is 82%, which shows that the model effectively captures most of these cases. The overall accuracy is 48%, which is reasonable for this type of classification. On the other hand, weaknesses include the low precision for the low and middle maternal mortality categories, which are 32% and 40%, respectively. This means the model needs help to classify many of these cases accurately. The recall for middle maternal mortality is relatively low at 19%, indicating that the model fails to capture many actual middle mortality cases. So, overall, while the Naive Bayes model performs well in identifying high and low maternal mortality cases, it has significant weaknesses in its precision and recall for middle mortality cases, limiting its overall effectiveness.

Decision Tree Regression Model:

On the one hand, the strengths of the decision tree regression model include its ability to handle complex, nonlinear relationships, such as those found in mortality data. The model is highly interpretable, making it easy to understand how decisions are made based on different features. Pruning the model by limiting the tree's depth also helps reduce the risk of overfitting while maintaining the model's ability to capture nonlinear trends. On the other hand, weaknesses include the fact that both the pruned and unpruned models had relatively similar results. For the unpruned model, the RMSE was 79.64, and the R square was 0.4008, while the pruned model slightly improved with an RMSE of 79.60 and an R square of 0.4014. This minimal change suggests that pruning did not significantly enhance the model's accuracy, indicating that the original model was not heavily overfitted. So, overall, while the decision tree regression provides good interpretability and handles nonlinearity, the pruning process did not drastically improve performance, implying that further adjustments may be needed for better results.

XGBoost Model:

On the one hand, the strengths of the XGBoost regression model include its exceptional performance, as evidenced by the low RMSE of 34.35 and the high R square value of 0.8885. This indicates that the model accurately captures the variance in the data and delivers solid predictions. XGBoost's ability to handle boosting rounds (with 100 estimators) ensures that the model refines its predictions iteratively, improving accuracy. On the other hand, a potential weakness is the complexity of tuning hyperparameters, such as the learning rate (set to 0.1), which requires careful calibration. If not tuned properly, the model can underfit or overfit the data, requiring additional experimentation to find the right balance. So, overall, the XGBoost model provided superior performance compared to the Decision Tree regression model, effectively capturing complex patterns in the dataset. However, careful tuning of hyperparameters is required to maximise its effectiveness.

In conclusion, the best-performing model is based on polynomial regression, and the second place is based on XGBoost.