Final Written Report (ADS1002) - Group Project

ADS1002 - Data Challenges 2

Mortality Rates in Intensive Care Units (ICUs)

Prepared by:

Fatimah Ayub (34602291)

Huang Xiao Hui (34518053)

Wesley Kong Zhe Hung (33521743)

Summary:

For this report, all the code written was done during class time that was set aside for project work. The executive summary, background and introduction was written by Fatimah Ayub. While the data preprocessing, manipulation and exploratory data analysis was done by Wesley Kong Zhe Hung. Finally, the summary of the models and analysis of the results was done by Huang Xiao Hui.

Executive Summary:

This report outlines a project aimed at predicting the mortality rate of Intensive Care Unit (ICU) patients using various machine learning algorithms. People who enter the ICU are in dire need of help. Therefore, accurately predicting patient outcomes can enhance treatment plans and improve overall healthcare quality. Mortality rate in our project was solely based on binary: 0 (survival) and 1 (death). By applying machine learning, we can analyze intricate datasets to uncover patterns that help healthcare professionals make informed, patient-specific decisions.

The dataset used was based on a real-life study conducted by PhysioNet Computing in Cardiology Challenge in 2012. This massed 12,000 participants who were monitored for the first 48 hours of ICU admission. We focused on a preprocessed subset of 1,474 patients, containing 37 variables, including key clinical indicators like Glasgow Coma Scale (GCS) and age. As mentioned earlier the data processed split each variable into minimum, maximum and mean. Through the course of the project, we found that the most significant predictors of mortality included Mean NIMAP and other clinically relevant features and discarding the features that were not providing results needed. However, this was dependent on the model that was used because each model had a slightly different top ten features.

column. This was to reduce the number of columns to make it easier for ourselves.

Our analysis began with exploratory data analysis to examine correlations between features and the target variable of in-hospital mortality. We utilized various machine learning algorithms, including Logistic Regression, Lasso Regularization, Ridge Regularization, Decision Trees, and Random Forest Classifiers, to determine their predictive performance. This was primarily based on their accuracy score. Each model was rigorously tested and compared based on accuracy, precision, recall.

Throughout our analysis, we observed that class imbalance significantly impacted model performance, with a higher prevalence of survival cases (171) compared to mortality cases (35). This was conveyed by the confusion matrix created. This imbalance suggests that the models may require further tuning, particularly in adjusting class weights to improve recall rates for the minority class.

Key findings indicated that the best performing algorithm was Random Forest Classifier because it had an accuracy score of 72%, thus capturing the complex patterns in the data. Lass Regression achieved an accuracy score of 71% but at the cost of low recall, hence suggesting the model had difficulties identifying potential at-risk patients. Unlike the other algorithms, Decision Trees had the lowest accuracy score of 58% which had a 10% increase after regularization. However it was not enough to suggest that was a good model as the other accuracies were higher.

In conclusion, while Logistic Regression provided mediocre results at best, the Random Forest Classifier outperformed other models, demonstrating its potential to improve patient care in ICU settings. Future work could refine model parameters and accuracy scores.

Background:

Data provided was from a study conducted in 2012 by PhysioNet Computing in Cardiology Challenge using a total of 12,000 participants. Each participant was monitored for initial 2 days. There were a total of 37 Time-Series variables such as Glasgow Coma Scale (GCS) and Heart Rate as well as other medical factors that can govern patients' mortality outcomes. In addition to the Time-Series variables, there were 7 features which were known as the General Descriptors such as age, Record ID and ICU type. Record ID is an important variable because patients are given IDs based on when they entered the ICU.It is worth noting that the patients with Do Not Resuscitate (DNR) were removed from the sample. Furthermore, predictions were made by using Simplified Acute Physiology Score (SAPS I) and Sequential Organ Failure Assessment (SOFA) scoring systems. These systems are designed to assist with predicting mortality based on some of the variables involved in the study. Nevertheless, there are other systems such as Acute Physiology & Chronic Health Evaluation (APACHE) which can further support the cruciality of such situations. Fortunately, the comma-separated value (CSV) given consisted of 1474 patients who were in the ICU and was preprocessed. Preprocessed data included the minimum, maximum and the mean for each variable. However, there were a total of six sets of data for each variable. Unfortunately, since patients were being monitored the first 48 hours, the study does not disclose how long the patient was in ICU after that time. The preprocessed CSV file offers a rich resource for examining the intricate relationships between these variables and in-hospital mortality.

Introduction:

The aim of our project was to predict the mortality rate utilizing various Machine Learning Algorithms. Accurately predicting patient outcomes in Intensive Care Units (ICUs) is crucial for enhancing treatment plans and improving the overall quality of care provided to the critically ill. Machine learning algorithms are vital because they can analyze the utmost complex of data sets by identifying patterns that may not be obvious to humans. Therefore, using predictive modeling enables healthcare professionals to make informed decisions suited for patients requirements.

Final Written Report (ADS1002) - Group Project

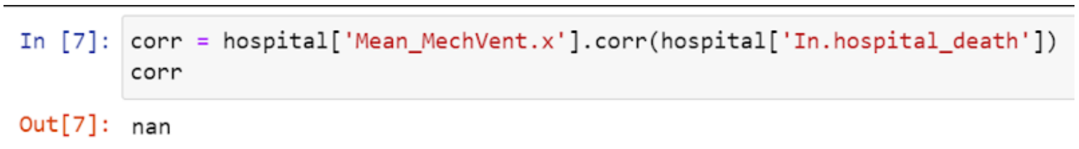
In this project, we utilize a methodical approach that begins with data cleaning, engineering and exploration. Followed by the application of various machine learning algorithms such Lasso regression and Random Forests Classifier. We will determine the performance of these models by using the model with the highest accuracy score. For approval of medicinal drugs, they require a high accuracy score. As a result we want to ensure that our model predictions are highly accurate due its involvement with human life. Therefore, we aim for our accuracy score for each algorithm to be approximately between 95 - 100% as the remaining percentage is due to chance. By doing so, we aim to gain valuable insights on how machine learning can aid in improving patient care in intensive care settings.

Data Quality

As the original dataset given was already preprocessed, we did not make any further changes to the dataset before loading the data into the Jupyter notebook.

We began the data manipulation process by checking for missing values and found none. Then, we removed the maximum and minimum value of 37 variables in the dataset. Since the variables observed are time sensitive, we determined that the mean of the variable would better capture a patient's condition over the first and second 24 hour period. On the other hand, using the maximum and minimum would cause our model to capture outliers which might cause overfitting to occur.

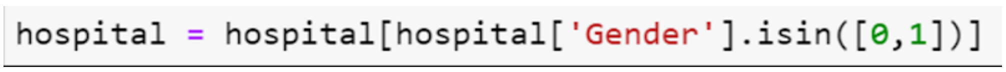
After that, we removed the 'MechVent' column, as every single value in the column was 1. This meant that the correlation between this particular feature and the target variable gave Not A Number (NaN) as seen in the Figure 1 below.



```
In [7]: corr = hospital['Mean_MechVent.x'].corr(hospital['In.hospital_death'])
        corr

Out[7]: nan
```

Figure 1: Code to show that Mean_MechVent.x had no values in the column

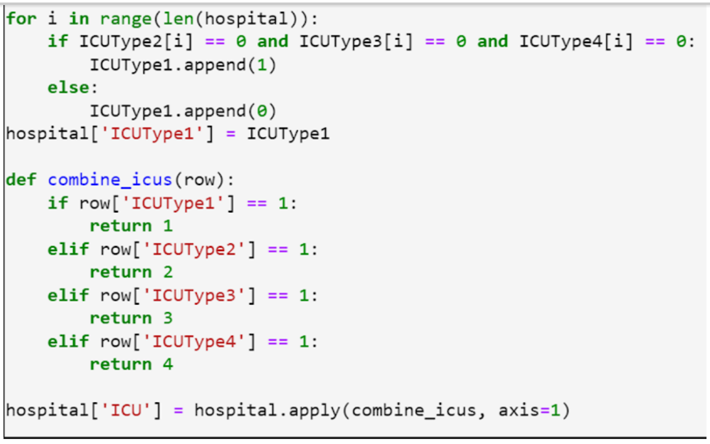
Then, we noticed that there was an error in the 'Gender' column as there were two rows where the values were not 0 or 1. As there was no other way to determine the gender of both of these patients, we decided to remove those two rows. The code written to remove these two rows can be seen in the figure 2 below.



```
hospital = hospital[hospital['Gender'].isin([0,1])]
```

Figure 2: Code removing values which are not 0 or 1 from the Gender column

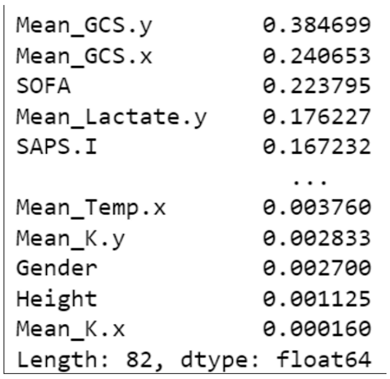
Another error that we noticed was that the dataset given was missing a column for 'ICUType1'. To rectify this issue, we used data engineering to create a new column for 'ICUType1' based on the given data for the other ICU types. We assumed that if in a row, the values of 'ICUType2', 'ICUType3', and 'ICUType4' were all 0, then the patient would belong to 'ICUType1'. After the new column had been created, we decided to create one more column simply named 'ICUType' to easily keep track of the ICU type that the patient was staying in. In this column, a value of 1 to 4 will be recorded depending on the corresponding ICU type that the patient was in. Figure 3 below shows the code written for this step.



```
for i in range(len(hospital)):
    if ICUType2[i] == 0 and ICUType3[i] == 0 and ICUType4[i] == 0:
        ICUType1.append(1)
    else:
        ICUType1.append(0)
hospital['ICUType1'] = ICUType1

def combine_icus(row):
    if row['ICUType1'] == 1:
        return 1
    elif row['ICUType2'] == 1:
        return 2
    elif row['ICUType3'] == 1:
        return 3
    elif row['ICUType4'] == 1:
        return 4

hospital['ICU'] = hospital.apply(combine_icus, axis=1)
```

Figure 3: Code showing how we inserted ICU Type 1 and combining all ICU Types into a single column

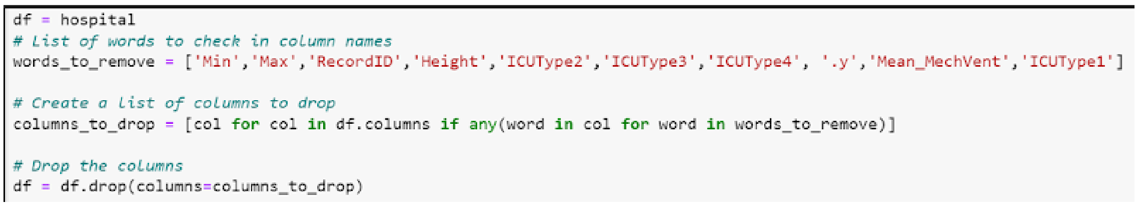
After the initial data cleaning and manipulation, we decided to start on the exploratory data analysis by looking at the correlation between the features and the target variable. Then, arranged the feature's correlation by descending value to get a grasp on the features that were most likely to affect the patients survival rate.



```
Mean_GCS.y        0.384699
Mean_GCS.x        0.240653
SOFA              0.223795
Mean_Lactate.y    0.176227
SAPS.I            0.167232
                    ...
Mean_Temp.x       0.003760
Mean_K.y          0.002833
Gender            0.002700
Height            0.001125
Mean_K.x          0.000160
Length: 82, dtype: float64
```

Figure 4: Table showing correlation between each feature and the target variable

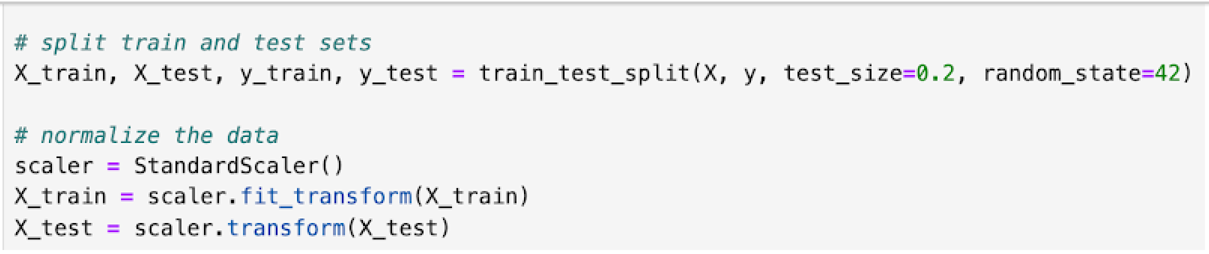
Figure 4 shows all of the features have a correlation of lower than 0.4, this means that the changes in these variables do not explain the changes in the target well. So, we decided not to select features based solely on their correlation with the target variable and instead opted to focus on biologically relevant factors for feature selection. This meant removing 'ICUType' and 'Height' as features as both variables were not biologically relevant to the survival of the patient and had a correlation of lower than 0.4. Furthermore, we decided to only look at the 'x' values of the features as we wanted to avoid overfitting. Another reason is to avoid multicollinearity as most of the 'x' variables were highly correlated with their 'y' counterparts. Thus, to obtain our final dataframe, we used the code shown in Figure 5 to remove the column we deemed unnecessary.



```
df = hospital
# List of words to check in column names
words_to_remove = ['Min','Max','RecordID','Height','ICUType2','ICUType3','ICUType4', '.y','Mean_MechVent','ICUType1']

# Create a list of columns to drop
columns_to_drop = [col for col in df.columns if any(word in col for word in words_to_remove)]

# Drop the columns
df = df.drop(columns=columns_to_drop)
```

Figure 5: Code used to remove key words included in the CSV file.

Finally, we defined the feature matrix X by excluding the target variable and 'ICUType' while also setting the target variable, y, to 'In.hospital_death'. We then split the data into raining (80%) and testing (20%) sets using train_test_split(), ensuring that the model is trained on one portion of the data and evaluated on another, unseen portion. Lastly, the data is normalized using StandardScaler(), which standardized the features to have a mean of 0 and a standard deviation of 1, helping the machine learning model perform better, particularly when features have different scales. This process ensures the model learns effectively and generalizes well to new, unseen data. The code written for this process is shown in the figure below.



```
# split train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# normalize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Figure 6: Code used for specifying the variables of X, y and code used for training the Logistic Regression with normalised data. Achieved by splitting into training and testing.

Summary of models and analysis

Logistic regression

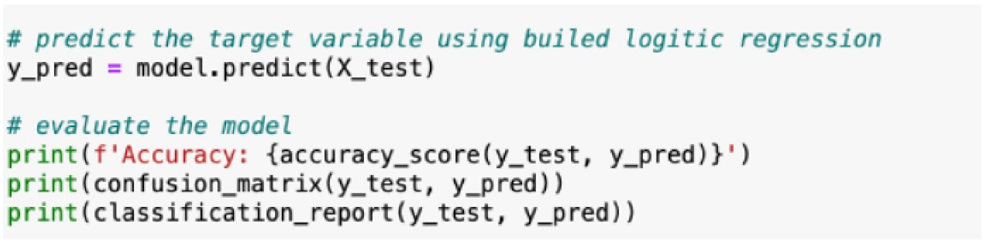
Logistic regression is used when the dependent variable is nominally scaled, allowing to predict a binary outcome. With logistic regression, it estimates the probability of occurrence of categories of the variable. In regard to our project, we wanted to find out which features have an influence on the target variable which was ‘in.hospital_death’. Since the target variable was binary: 0 meant survival and 1 indicated death. We also explored the influence of features such as gender, age and Mean_NIMAP (mean arterial pressure) and how they can affect mortality rate. We asked the question: How likely is mortality to occur if the patient has specific health needs?

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \varepsilon$$

Figure 7: The equation for Linear Regression (Osborned,2015).

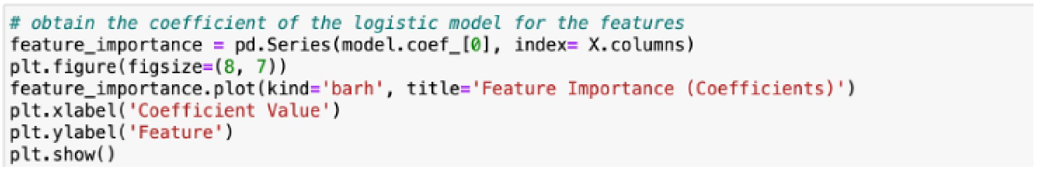
Figure 7 represents Y as the target variable which was the binary value we were trying to find. The β_n refers to the coefficients of the equation corresponding to the features included in the models. Furthermore, the coefficients can decide the direction of the graph.

Unlike linear regression, logistic regression involves sigmoid functions. Thus, the shape of the curve is an ‘S’. However, depending on the coefficients the shape may vary. Thus, the advantage of logistic regression can be that it supports further regularization such as Lasso (L1) and Ridge (L2) regularizations. These regularization methods can help prevent overfitting when managing large data sets such as these.



```
# predict the target variable using builed logitic regression
y_pred = model.predict(X_test)

# evaluate the model
print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

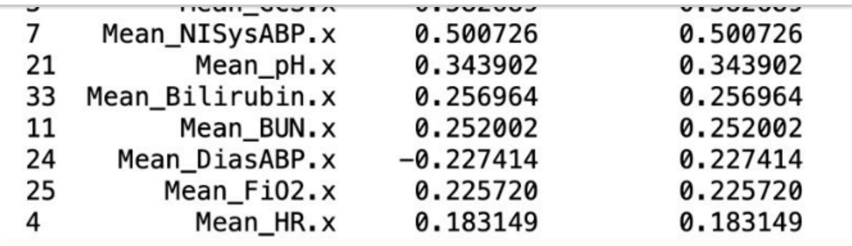
Figure 8: Code used for predicting y test data by printing accuracy score, confusion and classification report.



```
# obtain the coefficient of the logistic model for the features
feature_importance = pd.Series(model.coef_[0], index= X.columns)
plt.figure(figsize=(8, 7))
feature_importance.plot(kind='barh', title='Feature Importance (Coefficients)')
plt.xlabel('Coefficient Value')
plt.ylabel('Feature')
plt.show()
```

Figure 9: Code used to visualize the coefficient of feature in logistic model.

Final Written Report (ADS1002) - Group Project

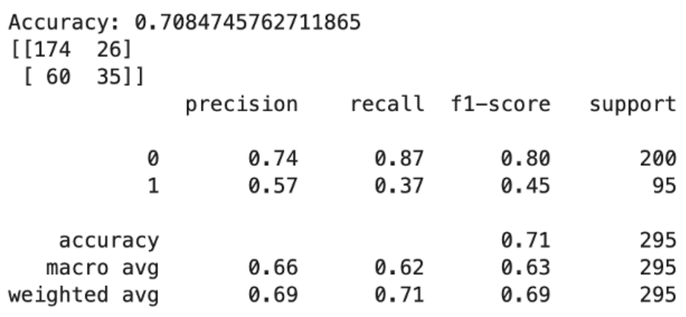


```
7     Mean_NISysABP.x    0.500726     0.500726
21          Mean_pH.x    0.343902     0.343902
33  Mean_Bilirubin.x    0.256964     0.256964
11         Mean_BUN.x    0.252002     0.252002
24     Mean_DiasABP.x   -0.227414     0.227414
25        Mean_FiO2.x    0.225720     0.225720
4           Mean_HR.x    0.183149     0.183149
```

Figure 10: List of the top ten most important variables

Once we determined the test and training scores, we plotted a Confusion Matrix. The confusion matrix provides a detailed breakdown of the true positives (TP), false positives (FP), and false negatives (FN). This was made by the model on the testing data. The model learns a set of coefficients when undergoing training, which describes how each feature affects the probability of death. Features with larger coefficients have a stronger influence on the prediction.

Result for Logistic Regression:



```
Accuracy: 0.7084745762711865
[[174  26]
 [ 60  35]]
              precision    recall  f1-score   support

           0       0.74      0.87      0.80       200
           1       0.57      0.37      0.45        95

    accuracy                           0.71       295
   macro avg       0.66      0.62      0.63       295
weighted avg       0.69      0.71      0.69       295
```

Figure 11: A Table to show accuracy, recall scores.

Figure 13 shows an accuracy score of approximately 0.708. This means that the model is correctly predicting the outcome 70.8% of the time. Since we are predicting mortality rate which is part of the medical community, we would need a higher accuracy score. Thus, this led us to find the variables which are truly making an impact on the mortality rate.

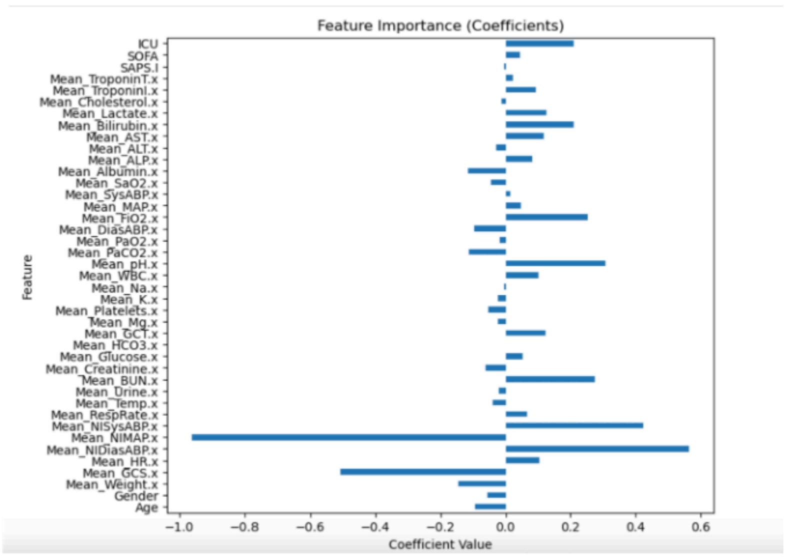


Figure 12: Bar chart to show coefficients of each variable in regard to the target variable.

As we obtained the overall importance for each feature and its relevance to the target. From this bar graph we can easily identify the top ten features based on their coefficients (Fig. 12). The feature importance from our logistic regression model, illustrates how much the log-odds of patient mortality change with a one unit increase in various features. This works when every other feature remains constant. For example, the coefficient for Mean_NIMAP.x is 1.064759. This negative value indicates that as the mean of NIMAP increases, mortality decreases.

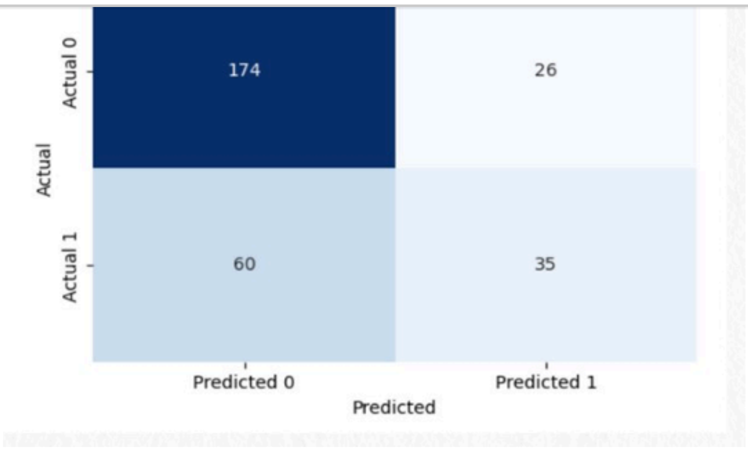
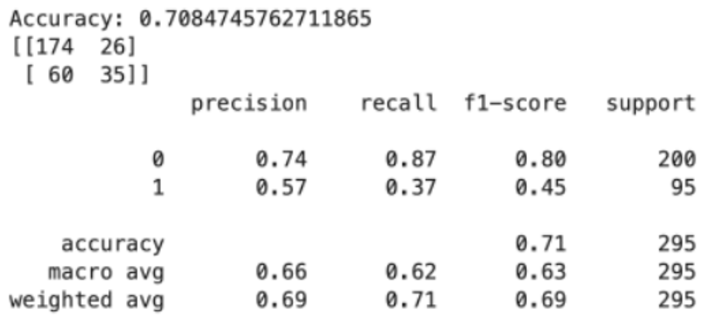


Figure 13: Confusion matrix for the logistic regression

The confusion matrix conveys classification (True Negative and True Positive) and misclassification (False Negative and False Positive). Initially upon observing the matrix the proportion out of 296 in the testing set were 174 TN, 35 TP, 26 FP and 60 FN. The high number of FN highlights the idea of there being low recall because it suggests that the model struggles identifying patients who are at risk of death (Fig. 14)..



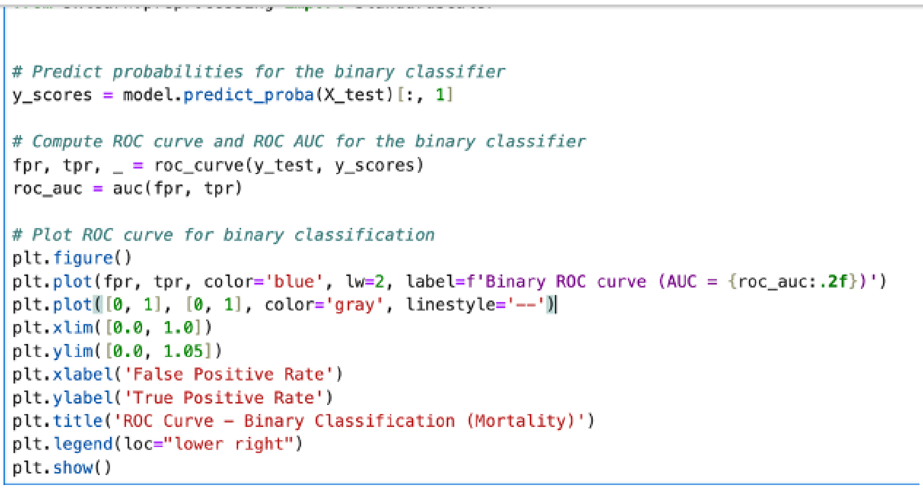
```
Accuracy: 0.7084745762711865
[[174  26]
 [ 60  35]]
              precision    recall  f1-score   support

           0       0.74      0.87      0.80       200
           1       0.57      0.37      0.45        95

    accuracy                           0.71       295
   macro avg       0.66      0.62      0.63       295
weighted avg       0.69      0.71      0.69       295
```

Figure 14: Report when using Logistic Regression

Since the accuracy is around 0.708 it does imply there must be low precision (0.57) and recall (0.37). Therefore, to reduce the false negatives we need to improve the recall. The weighted coefficients of each feature to visualize their importance, whether the specific feature has a positive or negative impact on the outcome and list out the features that most influence the outcomes.

<u>Binary Classification:</u>

Next, we conducted binary classification to solidify the data from the confusion matrix. We created a Receiver Operating Characteristics (ROC) which is a graphical plot to assess the performance of the binary classification. Therefore, the model calculates the Area Under the Curve (AUC).



```
# Predict probabilities for the binary classifier
y_scores = model.predict_proba(X_test)[:, 1]

# Compute ROC curve and ROC AUC for the binary classifier
fpr, tpr, _ = roc_curve(y_test, y_scores)
roc_auc = auc(fpr, tpr)

# Plot ROC curve for binary classification
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label=f'Binary ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Binary Classification (Mortality)')
plt.legend(loc="lower right")
plt.show()
```

Figure 15: Code to show how the ROC was determined.

We explored the True Positive Rate (TPR) or also known as Recall. This specifically looks at the proportion of patients who were correctly classified (Fig. 15). False Positive Rate (FPR) meant the proportion of actual negatives. Therefore, meant patients who survived were wrongly identified as people who had died. From this code, we plotted a line graph also known as the ROC curve. The x-axis was FPR against TPR. The graph then showed the calculated value for AUC.

Results for Binary Classification:

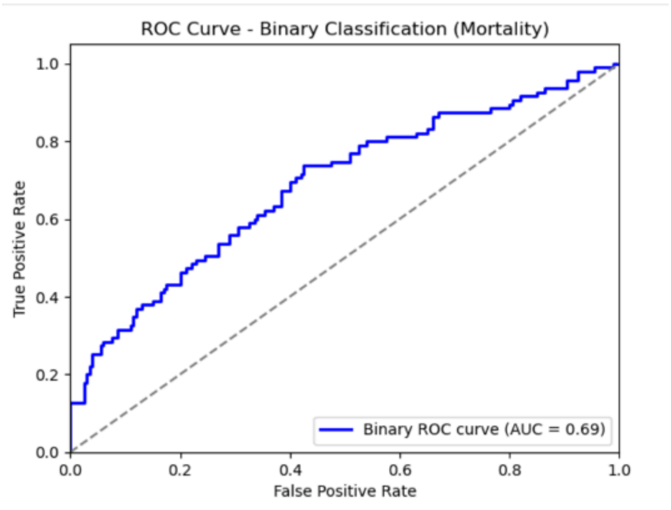
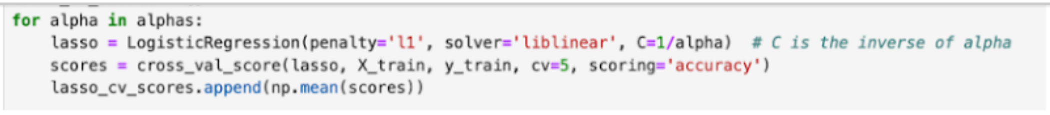


Figure 16: An ROC Curve between FP Rate vs. TP Rate

The dashed diagonal gray line represents the random classifier by probability of 50% of the model being correct (Fig. 16). As one can see the ROC curve (blue line) is outperforming the predicted line. The ROC curve rises slowly with some fluctuations but does not reach the extreme top right corner. Thus, arguing that the model does not achieve a TP rate without some FP. Additionally, the Area Under Curve (AUC) value is 0.69, highlighting how there is a 69% chance that a randomly chosen patient has died. However, this model is not perfect as there is still room for improvement because of the moderately low AUC score of 0.69.

Lasso Regression:

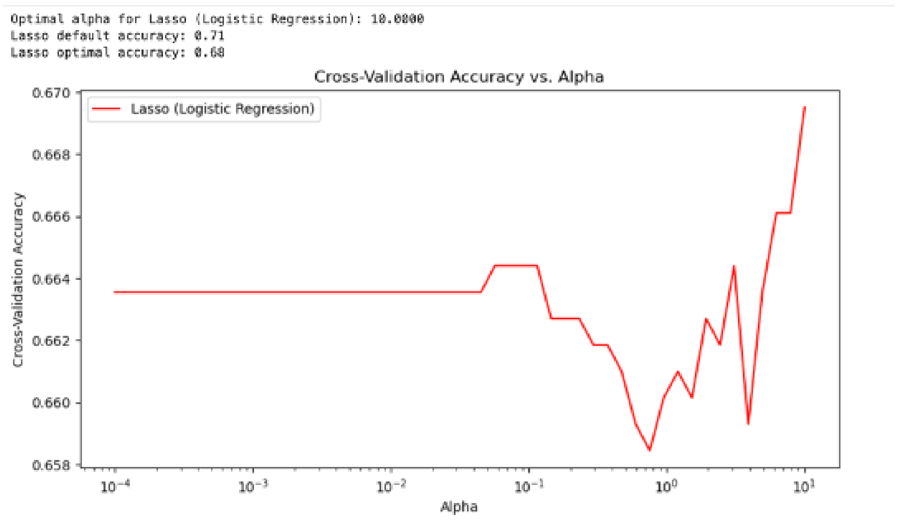


```
for alpha in alphas:
    lasso = LogisticRegression(penalty='l1', solver='liblinear', C=1/alpha)  # C is the inverse of alpha
    scores = cross_val_score(lasso, X_train, y_train, cv=5, scoring='accuracy')
    lasso_cv_scores.append(np.mean(scores))
```

Figure 17: Code to show Cross-Validation of Lasso Regularization

Based on the binary classification we moved on to Lasso regularization. As mentioned earlier lasso can help prevent possible overfitting and is an L1 regularization. Alphas shown in the above code are logarithmically spaced in a range of 0.0001 to 10. This range is used to find an optimal level of regularization. The next line of code represents cross-validation. Cross-validation helps identify the optimum alpha value by evaluating different values to find the value which maximizes performance. Thus, allowing us to find the alpha value with the highest average accuracy. In our project we used a cv value of 5. This means that the regularization uses a 5-fold cross validation. In simpler terms, the data is divided into 5 sections, therefore ensuring each data point is used for both training and validation. As a result of this the model's performance is more reliable because it attempts to avoid overfitting and issues with variance.

<u>Results for Lasso Regularization:</u>



```
Optimal alpha for Lasso (Logistic Regression): 10.0000
Lasso default accuracy: 0.71
Lasso optimal accuracy: 0.68
```

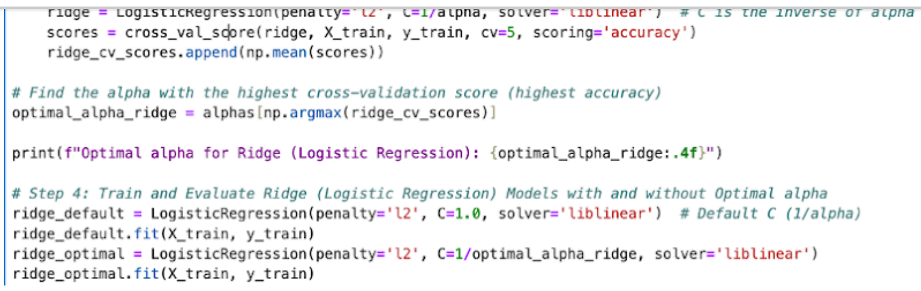
Figure 18: A Graph showing Lasso Regularization

The optimal alpha for Lasso equates to 10 when undergoing cross-validation. This means that the algorithm applies a strength of 10 to reduce the potential overfitting of data. The difference between Lasso default accuracy of 0.71 versus the Lasso optimal accuracy could be due to reducing overfitting more effectively. Thus, indicating that the model has been too simplified.

<u>Ridge Regularization</u>

Since we had used Logistic Regression earlier on in our project, it adds a penalty by making the model more stable. This is achieved by reducing the errors between the prediction and actual values. Hence, fitting the training data and permitting it to work on new sets of data.

Final Written Report (ADS1002) - Group Project



```
    ridge = LogisticRegression(penalty='l2', C=1/alpha, solver='liblinear')  # C is the inverse of alpha
    scores = cross_val_score(ridge, X_train, y_train, cv=5, scoring='accuracy')
    ridge_cv_scores.append(np.mean(scores))

# Find the alpha with the highest cross-validation score (highest accuracy)
optimal_alpha_ridge = alphas[np.argmax(ridge_cv_scores)]

print(f"Optimal alpha for Ridge (Logistic Regression): {optimal_alpha_ridge:.4f}")

# Step 4: Train and Evaluate Ridge (Logistic Regression) Models with and without Optimal alpha
ridge_default = LogisticRegression(penalty='l2', C=1.0, solver='liblinear')  # Default C (1/alpha)
ridge_default.fit(X_train, y_train)
ridge_optimal = LogisticRegression(penalty='l2', C=1/optimal_alpha_ridge, solver='liblinear')
ridge_optimal.fit(X_train, y_train)
```

Figure 19: Code showing the process behind Ridge Regularization.

The alpha in Ridge works slightly differently to Lasso because it controls the strength of the Ridge regularization. The default ridge involves no regularization, thus acting as a baseline value for comparison to the optimal ridge value. Similarly for Lasso, we used a cv value of 5. Indicating that the model is split into 5 parts. 4 parts are for the model to be trained on, whereas the final part is for the testing. Furthermore, it is repeated 5 times, each time it involves a new section.

Result for Ridge Regularization:

```
Optimal alpha for Ridge (Logistic Regression): 0.0001
Ridge default accuracy: 0.71
Ridge optimal accuracy: 0.71
```

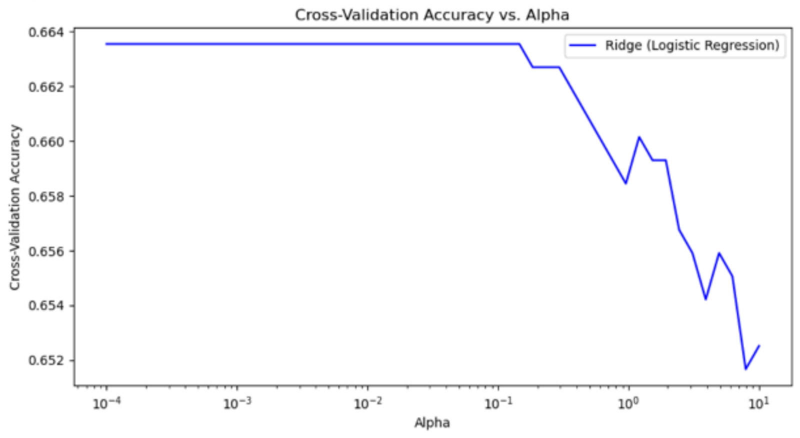
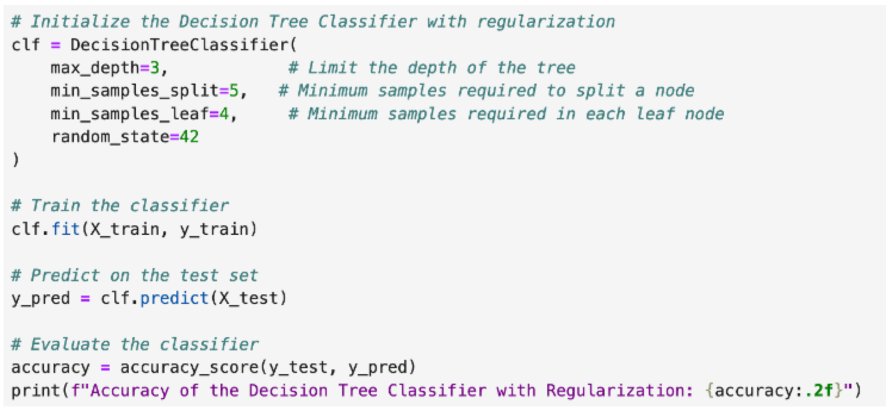


Figure 20: A graph showing Ridge Regularization

For Ridge regularization the optimal alpha of 0.0001 is extremely low, implying that very minimal regularization was needed. Since both the ridge default and optimal accuracy scores are equal, it emphasizes that there is no indication of overfitting or underfitting.

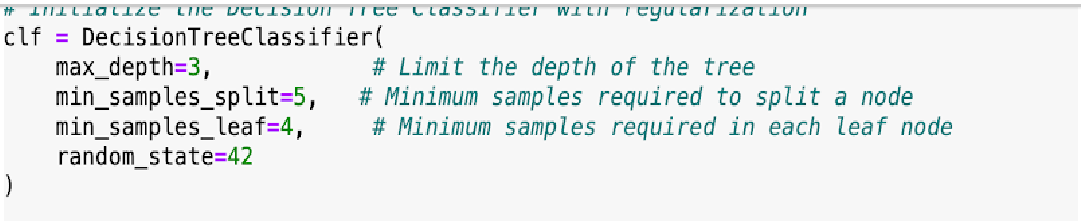
Decision Trees

Decision Trees is a type of supervised learning algorithm. The concept of it works by splitting data into its features until it cannot split any further. The decision trees can be used for both classification and regression. We illustrated the accuracy score of the decision tree before the regularisation,



```
# Initialize the Decision Tree Classifier with regularization
clf = DecisionTreeClassifier(
    max_depth=3,            # Limit the depth of the tree
    min_samples_split=5,   # Minimum samples required to split a node
    min_samples_leaf=4,     # Minimum samples required in each leaf node
    random_state=42
)

# Train the classifier
clf.fit(X_train, y_train)

# Predict on the test set
y_pred = clf.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Decision Tree Classifier with Regularization: {accuracy:.2f}")
```

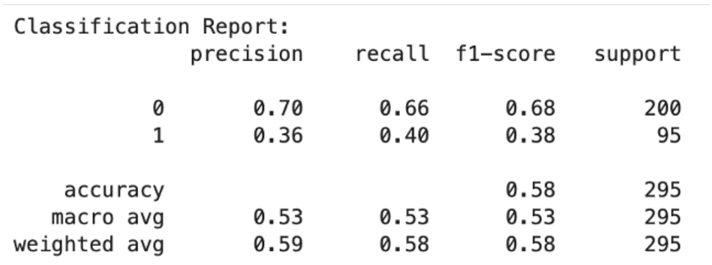
Figure 21: Using Decision Trees with Regularization



```
# Initialize the Decision Tree Classifier with regularization
clf = DecisionTreeClassifier(
    max_depth=3,             # Limit the depth of the tree
    min_samples_split=5,    # Minimum samples required to split a node
    min_samples_leaf=4,      # Minimum samples required in each leaf node
    random_state=42
)
```

Figure 22: Codes for initializing the decision tree with regularization

<u>**Result of Decision Trees:**</u>



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

           0       0.70      0.66      0.68       200
           1       0.36      0.40      0.38        95

    accuracy                           0.58       295
   macro avg       0.53      0.53      0.53       295
weighted avg       0.59      0.58      0.58       295
```

Figure 23: Classification Report when doing initial accuracy score using Decision Tree Classifier

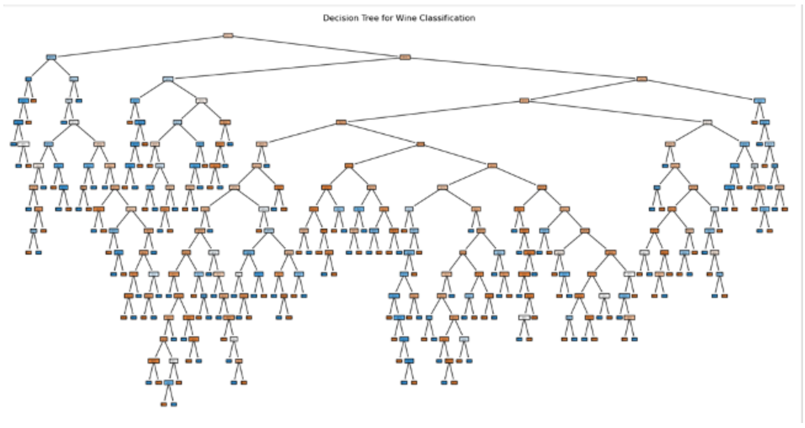
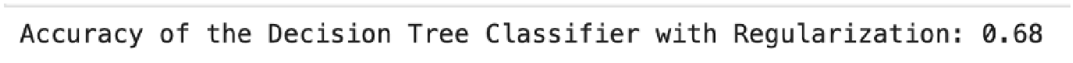


Figure 24: Decision Tree Diagram

The low accuracy score of 0.58 insinuates that there is potential overfitting especially if there are too many branches or nodes (Fig. 24). Another reason for its low accuracy could be because of a class imbalance as in one category there are 200 whilst the other contains 95. Therefore, we reject this model because we believe that the accuracy score can be higher.



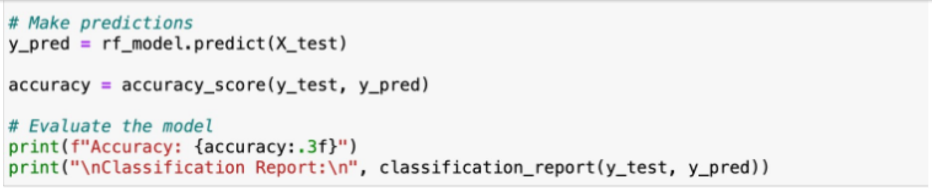
```
Accuracy of the Decision Tree Classifier with Regularization: 0.68
```

Figure 25: Decision Tree Accuracy Score with Regularization

The improvement of accuracy score 0.68 could be due to limiting the maximum depth which prevents there from being excessive branches. As we saw in Figure 22, we had too many leaves which could be another possible reason for hindering the accuracy. The higher accuracy causes the decision tree to avoid overfitting.

<u>**Random Forest Classifier:**</u>

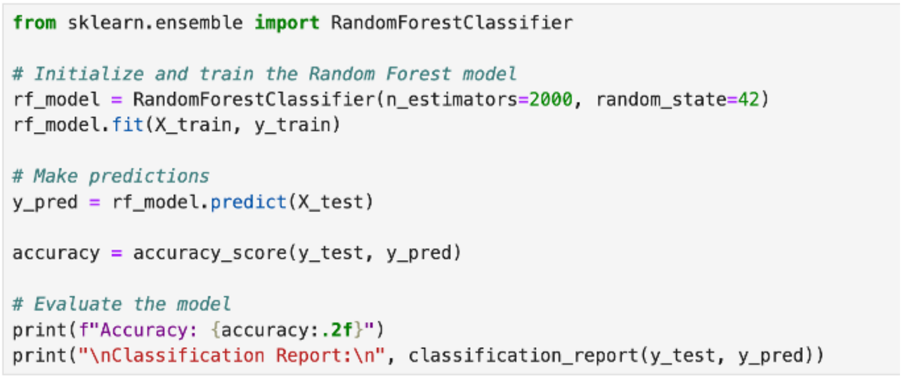
The Random Forest Model creates multiple decision trees and combines their predictions to make it as suitable and robust as possible, by enhancing a model's performance. This algorithm can handle both continuous and categorical variables.



```
# Make predictions
y_pred = rf_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

# Evaluate the model
print(f"Accuracy: {accuracy:.3f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Figure 26: Code of the accuracy score when using Random Forest Classifier

Similarly to logistic regression, the dataset is split into 80% training set and 20% testing set. The data split is for training the model to get predicted outcomes.

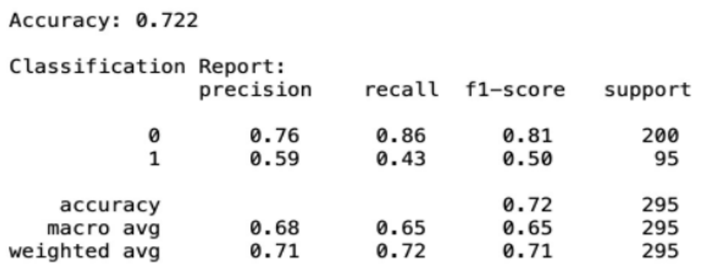


```
from sklearn.ensemble import RandomForestClassifier

# Initialize and train the Random Forest model
rf_model = RandomForestClassifier(n_estimators=2000, random_state=42)
rf_model.fit(X_train, y_train)

# Make predictions
y_pred = rf_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

# Evaluate the model
print(f"Accuracy: {accuracy:.2f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Figure 25: Regularising Random Forest

Normalization of the data is a necessity as it undergoes the process for building multiple decision trees in a process called Bootstrap aggregating (bagging). Bagging is an ensemble learning method. There are many versions of the model which are trained using different random samples. As a result, the predictions are combined to produce a more accurate model. The model is imported by RandomForestClassifier from sklearn packages. Random forests are known for their ability to capture complex patterns in the data and often work well with imbalanced datasets to reduce variance.

<u>Result for Random Forest Classifier</u>



```
Accuracy: 0.722

Classification Report:
               precision    recall  f1-score   support

           0       0.76      0.86      0.81       200
           1       0.59      0.43      0.50        95

    accuracy                           0.72       295
   macro avg       0.68      0.65      0.65       295
weighted avg       0.71      0.72      0.71       295
```

Figure 28: The accuracy and classification report of the model with random forest classifier

The high accuracy score (0.72) when conducting the random forest classifier hints that the model is more confident in predicting survival (0) than death (1). A reason for this could be due to the high number of survival instances in the data set. The low recall (0.43) when classifying death could be due to missing some true positives which are being misclassified.

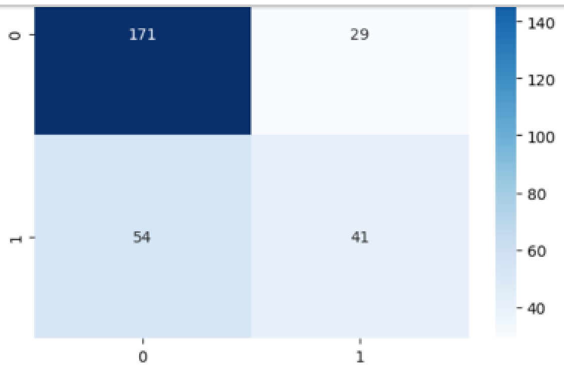


Figure 29: Confusion Matrix on Random Forest Classifier

The confusion matrix illustrates that TP is 41 means the model has rightly predicted these patients' death. The TN value is 171, which means these patients survived.

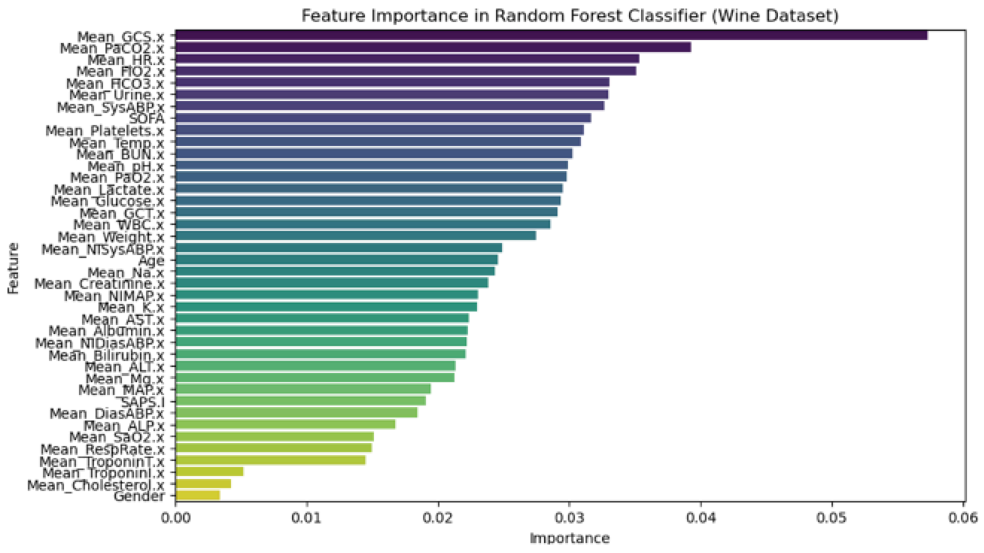


Figure 30: Feature Importance in Random Forest Classifier

Additionally, we also have found that the feature of Mean GCS.X has the highest importance index to our target variable, around 0.058509 when producing the Random Forest Classifiers. From this we can conclude the top 10 features when using this algorithm.

Conclusion:

To conclude, when comparing the three models - logistic regression, random forest, and decision tree - logistic regression stands out for its interpretability. As it directly models the probability of a binary outcome. This allows us to identify the top 10 most significant predictors. However, one limitation of the logistic regression model is its bias towards the majority class (survivors), which in this dataset results in 174 correctly predicted survivors but only around 35 predicted deaths. To address this class imbalance, we can apply the `class_weight` parameter, to adjust the weights inversely proportional to class frequencies to improve the model's ability to predict minority outcomes.

The decision tree model, though easy to interpret, initially showed the lowest accuracy at 0.58. With regularization, this improved to 0.68, but it still lags behind the other models in performance. Therefore, this model was not effective due to its relatively low accuracy score,

On the other hand, the random forest classifier demonstrated the highest accuracy at 0.72, meaning it correctly predicts 72% of the outcomes. By combining the outputs of multiple decision trees, the random forest model captures complex, non-linear patterns that logistic regression cannot. Hence, leading to

Final Written Report (ADS1002) - Group Project

Hopefully, data scientists across the globe can assist the medical communities by using advanced algorithms to predict mortality rate and to provide the best resources to patients who are suffering in the ICU.