Predicting Heart Failure Survival on Clinical Data using 3 Machine Learning Algorithms

DA 5030

Dauby, Ray

Spring 2024

I. Data Preparation

Business Understanding

The goal of this analysis is to develop a machine learning model that accurately predicts the survival of patients after heart failure based on clinical data. This will be useful in helping healthcare providers make informed decisions about treatment plans, allocate resources more effectively, and provide personalized care for individuals based on their likelihood of survival. By identifying key clinical factors that influence outcomes, the model can assist in early intervention, improve patient management, and enhance overall patient care and outcomes.

Problem to be Solved:

The problem to be solved is predicting whether a patient will survive or not after being diagnosed with heart failure, based on various clinical factors.

Part 1: Identify Data

	a.. <dbl>	anae… <int>	creatinine_phosphokinase <int>	diabetes <int>	ejection_fraction <int>	high_blood_pressure <int>	platele <db
1	75	0	582	0	20	1	26500
2	55	0	7861	0	38	0	26335
3	65	0	146	0	20	0	16200
4	50	1	111	0	20	0	21000
5	65	1	160	1	20	0	32700
6	90	1	47	0	40	1	20400

6 rows | 1-8 of 14 columns

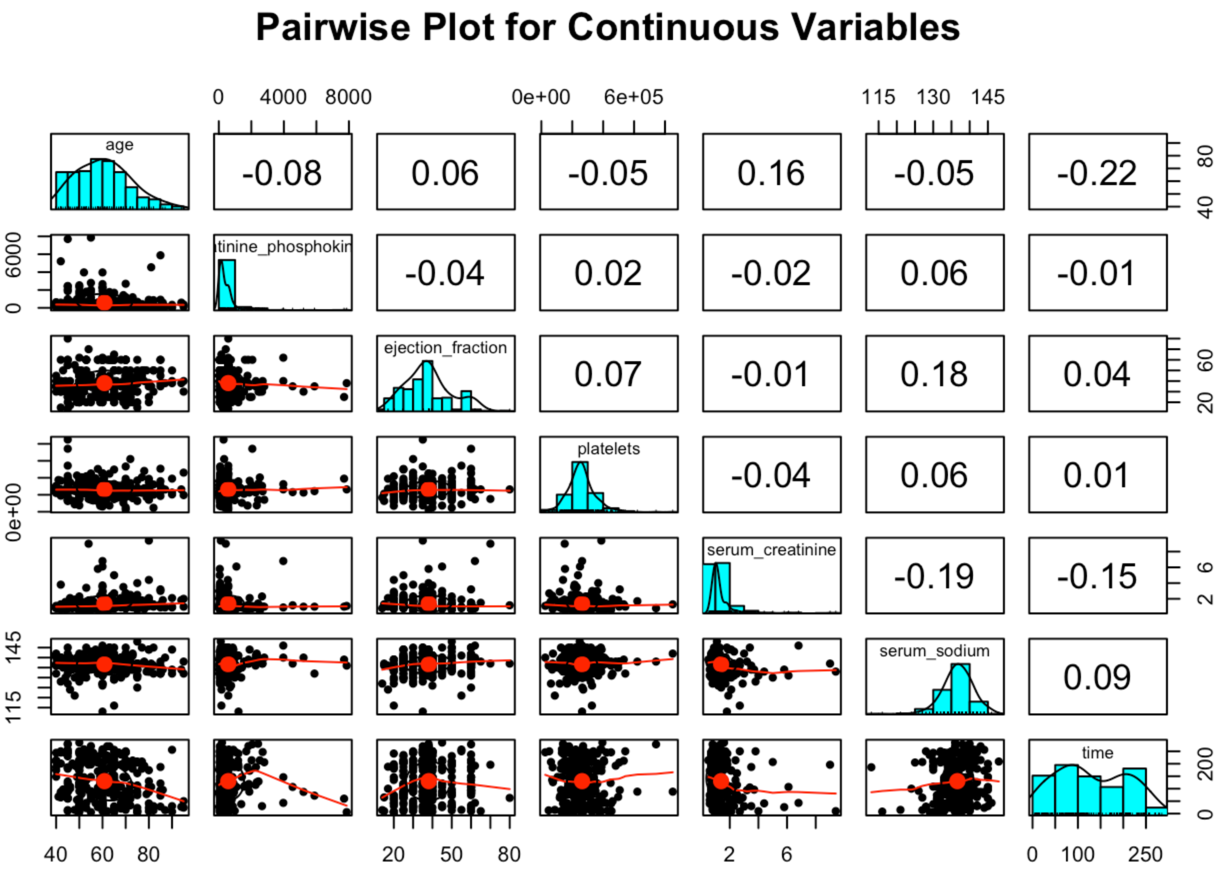
Data Understanding:

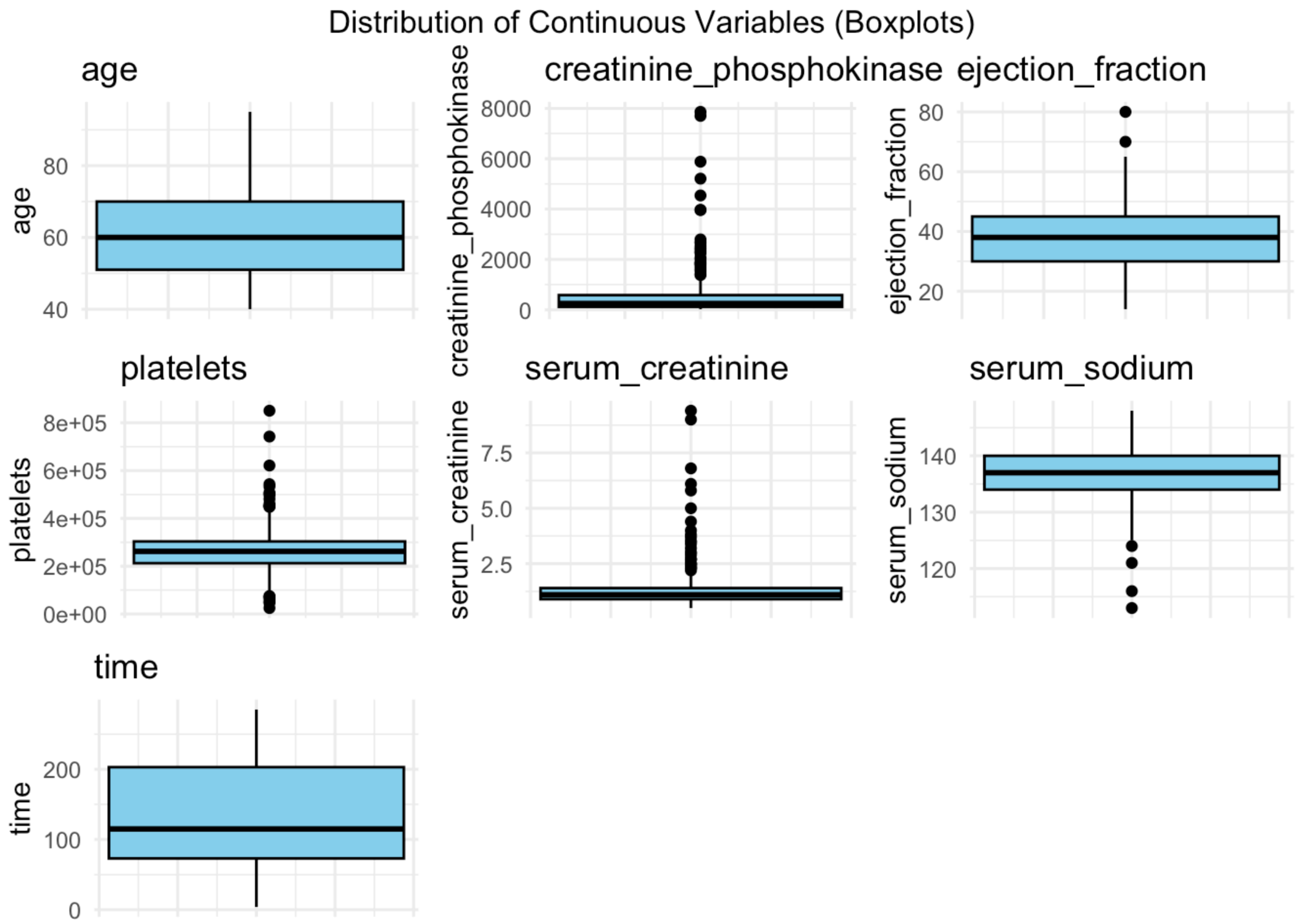
The dataset used in this analysis consists of medical records from 299 heart failure patients, including 105 women and 194 men, aged between 40 and 95 years. The data was collected from the Faisalabad Institute of Cardiology and the Allied Hospital in Faisalabad, Punjab, Pakistan, during April–December 2015. It includes both continuous and binary numeric variables, covering various clinical factors relevant to heart failure outcomes. The dataset has no missing values, and is publicly available from the UCI Machine Learning Repository.

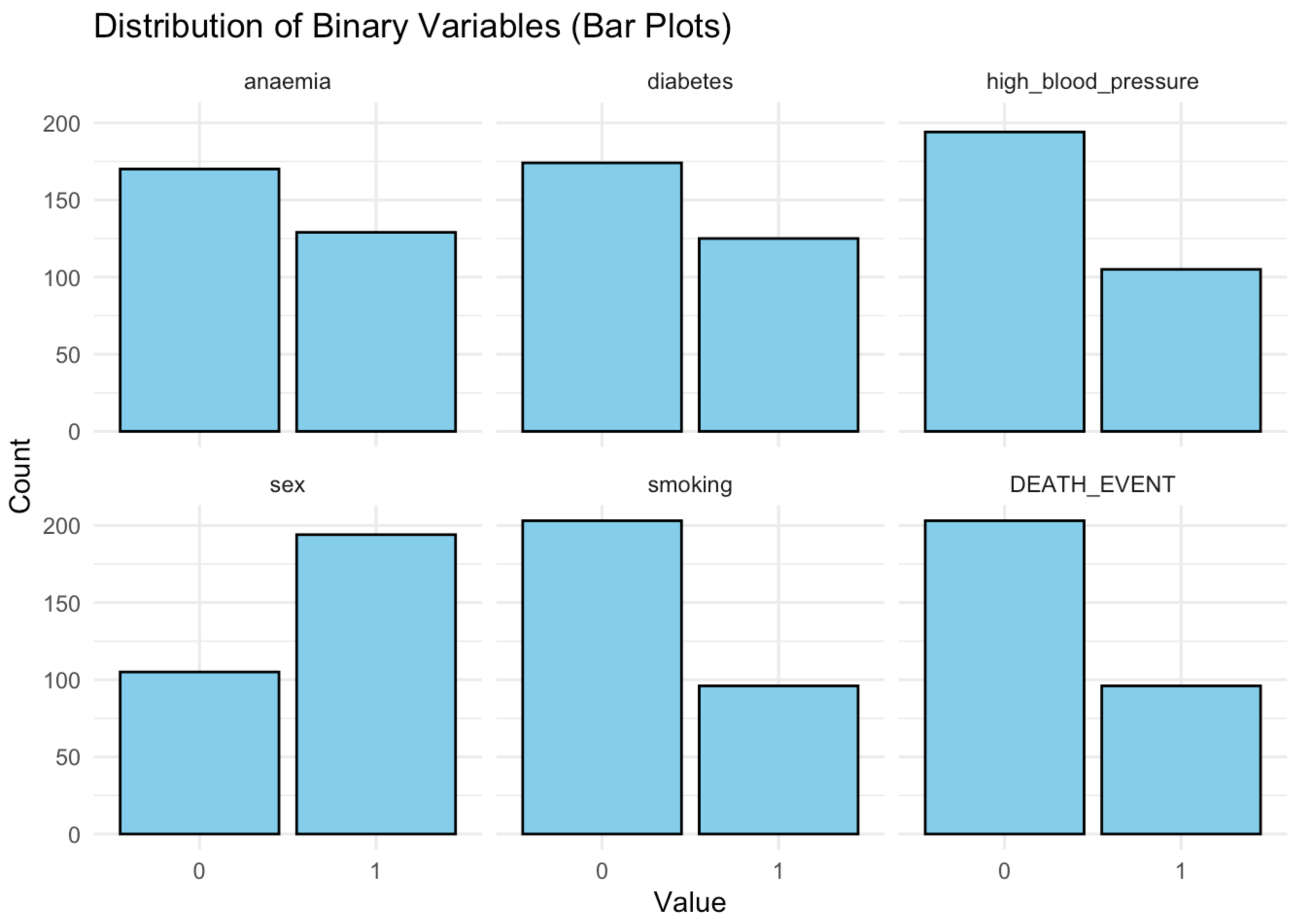
Data Source:

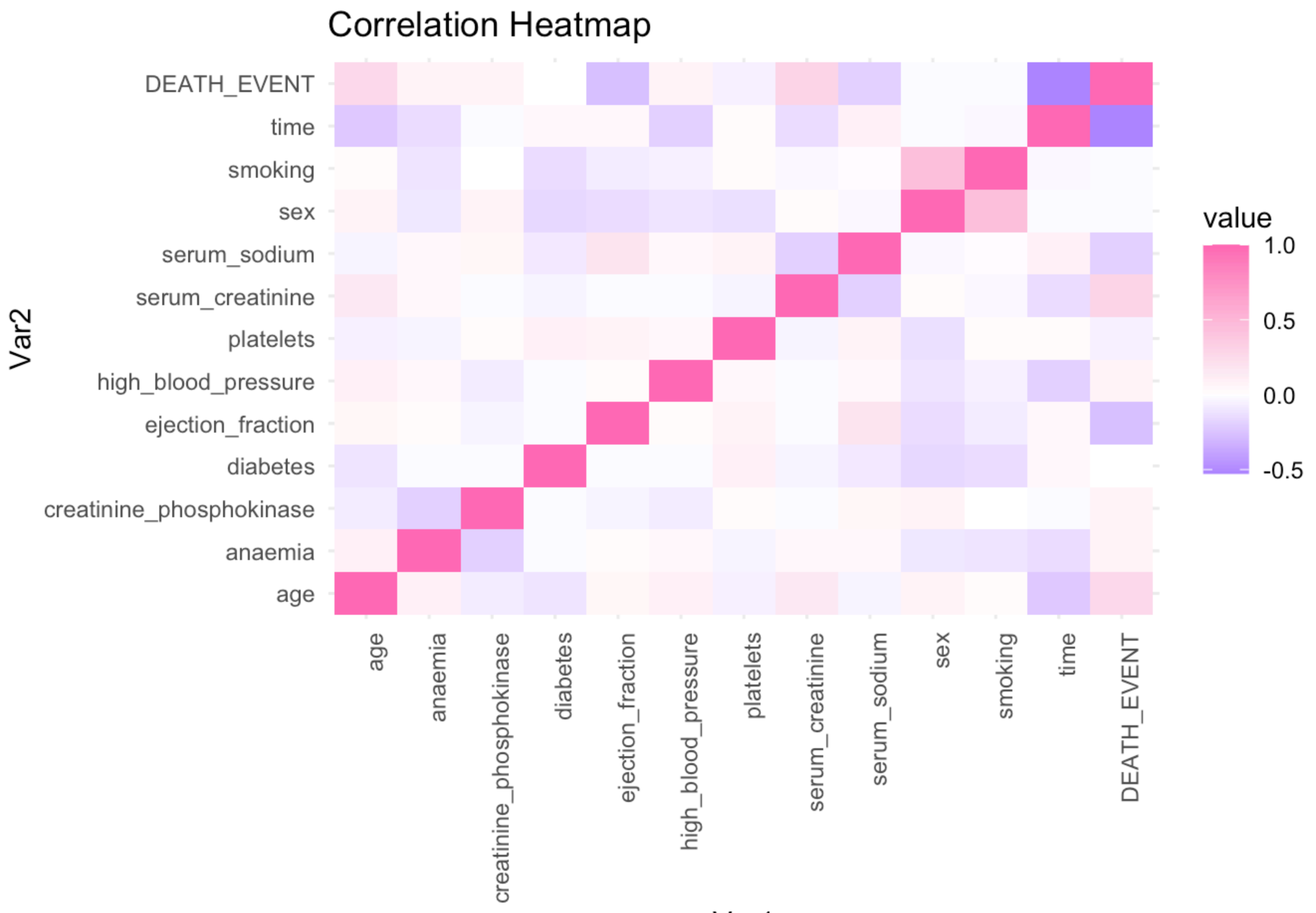
Heart Failure Clinical Records [Dataset]. (2020). UCI Machine Learning Repository. https://doi.org/10.24432/C5Z89R (https://doi.org/10.24432/C5Z89R).

Part 2: Data Exploration Plots









VIF Values for All Variables

	x
age	1.106
anaemia	1.087
creatinine_phosphokinase	1.066
diabetes	1.064
ejection_fraction	1.068
high_blood_pressure	1.068
platelets	1.046
serum_creatinine	1.081
serum_sodium	1.102
sex	1.338
smoking	1.285
time	1.138

Data Exploration:

Outliers Detection

I used boxplots to detect outliers in the continuous variables. These graphs revealed a few potential outliers in variables like creatinine_phosphokinase, serum_creatine, and platelets, where certain values significantly deviated from the rest of the data.

Distributions

To assess the distributions of both continuous and binary variables, I created histograms (within a Pairwise Plot) and bar plots. The histograms suggested that several variables, such as creatinine_phosphokinase and serum_creatine had skewed distributions, which may require transformation for better model performance. The binary variables were plotted as bar charts, revealing an imbalance in the target variable, DEATH_EVENT, as well as sex and smoking.

Correlations and Collinearity

I also evaluated the correlation between all of the variables using a Spearman correlation matrix. The results showed moderate correlations between certain variables, such as DEATH_EVENT and time, and sex and smoking. To assess these relationships further, I computed the Variance Inflation Factor (VIF) for each feature, which indicated that multicollinearity is not a significant concern, as all VIF values were below 1.5 and well within an acceptable range.

Part 3: Assess Data Quality (Outliers and Missing Values)

Data Preparation:

Missing Values

The dataset is known to have no missing values, which is ideal for model training and ensures that we do not need to impute any missing data. However, if there were to be missing values, I would consider using the mean for imputing continuous variables and the mode for categorical data.

Imputation

While the dataset has no missing data, a comparison of imputed versus full data could be useful in understanding the impact of missing data imputation on model performance. If data were randomly removed and then imputed, the performance of the model could differ due to the introduction of bias or errors from the imputation process. For example, imputing with the mean or mode may reduce variance in the data, leading to underfitting, or removing outliers could eliminate valuable information that helps the model differentiate between normal and extreme cases.

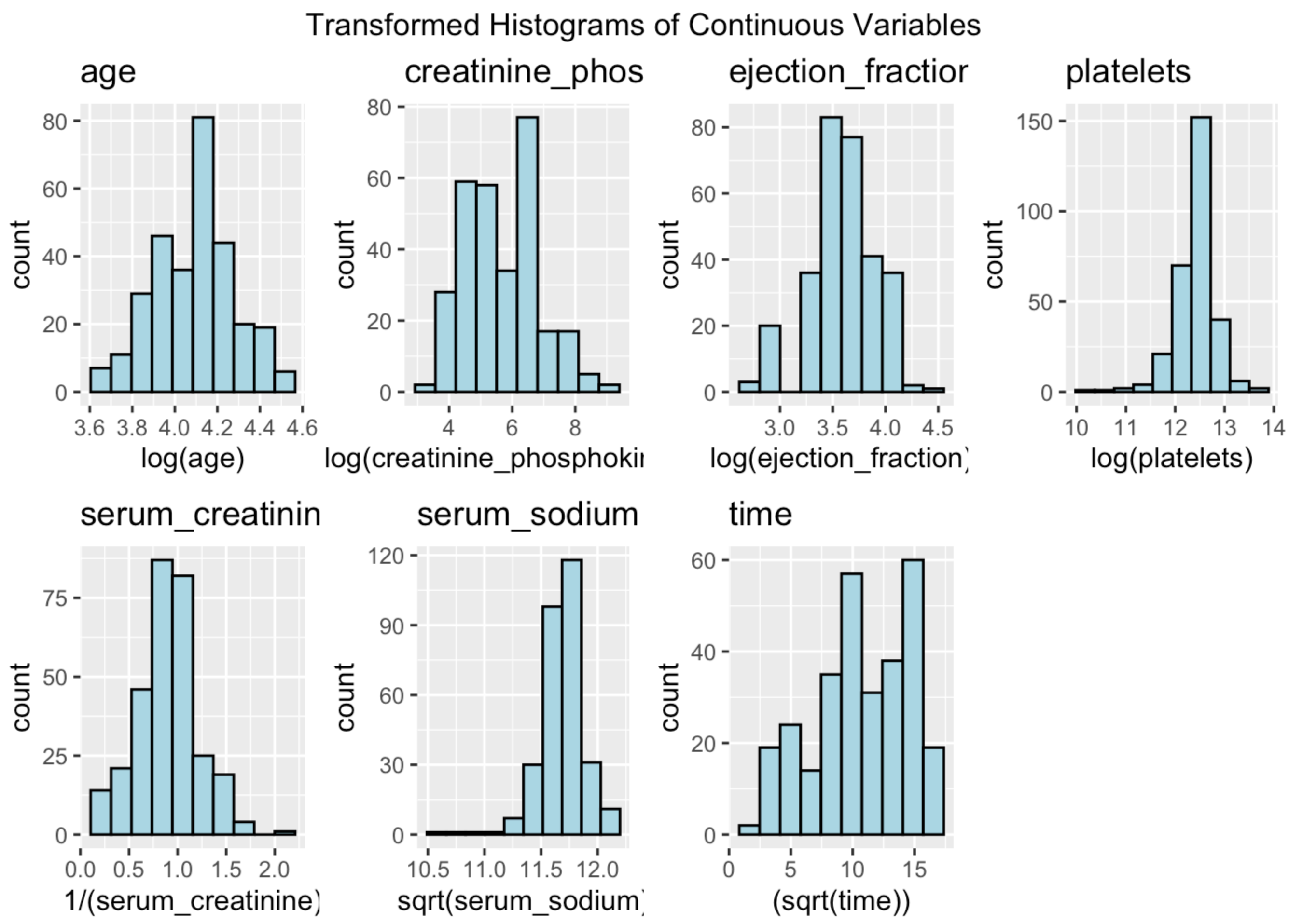
Outliers

I have opted not to remove outliers from the dataset I will use for model development. This decision stems from the fact that the data comes from a clinical context, and outliers could potentially represent extreme but medically relevant cases, such as rare conditions or severe health episodes that significantly affect the outcome. Removing these could skew the results and reduce the generalizability of the model. No cases will be removed from the analysis.

Part 4: Assess Data Normality

Abnormally Distributed Continuous Variables

x
age
creatinine_phosphokinase
ejection_fraction
platelets
serum_creatinine
serum_sodium
time



Data Cleaning:

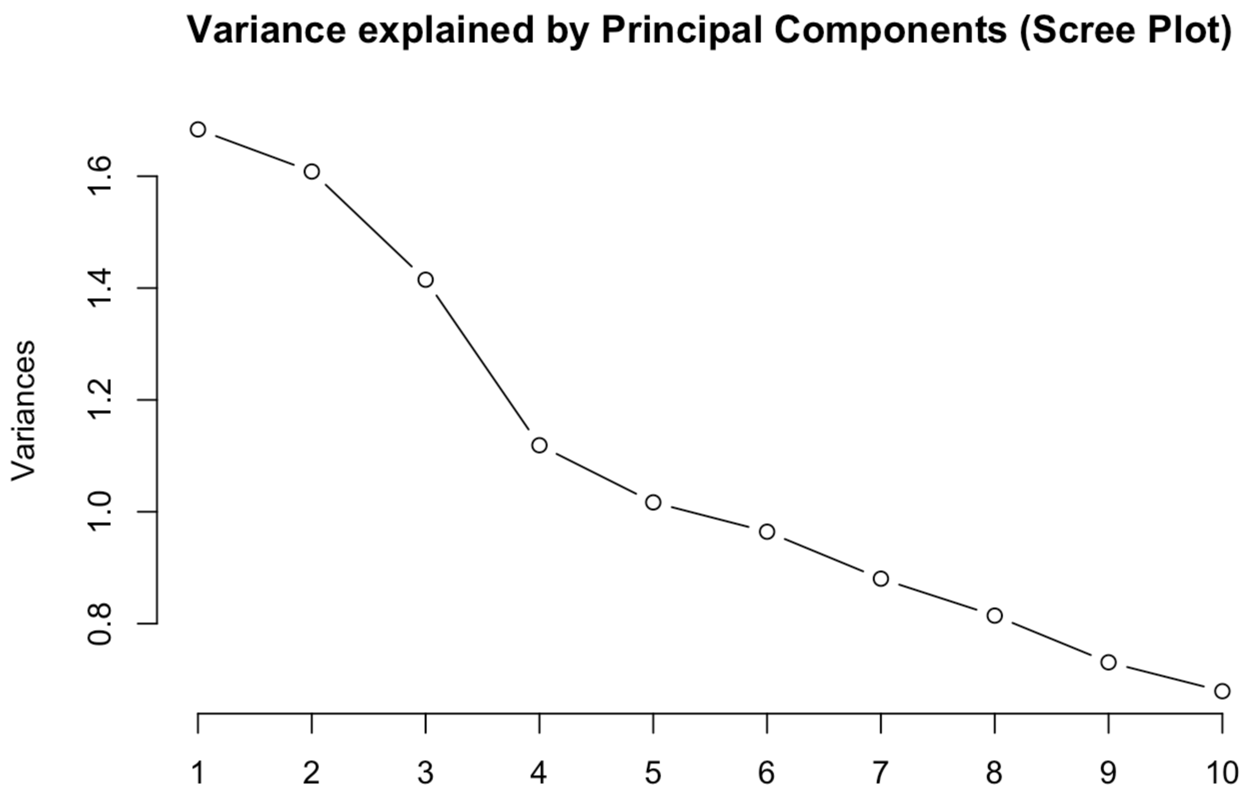
Distribution

To assess normality, I performed the Shapiro-Wilk test on each continuous feature and identified those that did not follow a normal distribution. Based on these findings, I applied appropriate transformations to make the distributions more symmetric and Gaussian-like where needed. Although my selected machine learning algorithms do not require data to follow a Gaussian distribution, I still wanted to explore whether normalizing the features would help with model convergence, particularly for algorithms that are sensitive to feature scaling, such as Support Vector Machines (SVM).

Scaling

For normalization, I used Min-Max scaling, which rescales the data to a range between 0 and 1, ensuring that all features are on the same scale. This is useful when features have different units or scales, which could otherwise dominate the model's learning process. Additionally, the transformed features will help reduce issues arising from extreme values or skewed distributions, allowing the algorithms to perform more effectively. Overall, the goal is to strike a balance between preserving the integrity of the data while optimizing it for model training.

Part 5: Feature Selection and Engineering



Principal Component Analysis:

I applied principal component analysis (PCA) to the dataset to explore the underlying structure of the data and potentially reduce dimensionality. However, the PCA results reveal that the first few principal components do not explain much of the variance in the data, as indicated by the low eigenvalues and variance explained by each component. This suggests that there may be multiple directions of variation in the dataset, or potentially noise, which makes it difficult to capture meaningful patterns in the reduced dimensions. Since there are only 12 features in the dataset, I have decided not to remove any features at this stage, as each feature could potentially contribute valuable information to the model, especially in a healthcare context where all variables may have clinical significance. It does not seem that new derived features will benefit the model.

Data Quality Conclusion:

- **No Missing Data**
- **Outliers Are Present but Justifiable:** There are some extreme values in the data, but because the data is clinical, it is likely that these outliers reflect real and significant health conditions.
- **Non-Normal Distributions:** Several continuous features in the dataset are not normally distributed. This was addressed by applying appropriate transformations.
- **Low Multicollinearity:** The correlation analysis revealed low levels of multicollinearity between features, which suggests that feature interactions are not causing significant issues in model performance.
- **Accurate Clinical Data:** No significant errors were found in the data regarding values outside of valid ranges for clinical variables.

Part 6: Testing and Training sets

Distribution of Target Variable in Whole Dataset

.id	label	variable	value
DEATH_EVENT	DEATH_EVENT	No	203 (67.89%)
DEATH_EVENT	DEATH_EVENT	Yes	96 (32.11%)

Distribution of Target Variable in Training Dataset

.id	label	variable	value
DEATH_EVENT	DEATH_EVENT	No	144 (68.57%)
DEATH_EVENT	DEATH_EVENT	Yes	66 (31.43%)

Distribution of Target Variable in Testing Dataset

.id	label	variable	value
DEATH_EVENT	DEATH_EVENT	No	59 (66.29%)
DEATH_EVENT	DEATH_EVENT	Yes	30 (33.71%)

Division Strategy:

In this analysis, the dataset is split into training and test sets using a 70/30 ratio. The createDataPartition() function from the caret package ensures that the distribution of the binary target variable, DEATH_EVENT, is preserved in both the training and test datasets. By setting the seed for reproducibility, we ensure that the data split can be consistently reproduced across different runs. The splits are not identical in proportion due to the relatively small size of the dataset.

II. Model Construction

Part 7: Model Selection

Logistic Regression:

Logistic Regression is a fundamental statistical model used for binary classification tasks. It works by modeling the probability of a binary outcome using a logistic function, where the input features are combined linearly, and the output is a value between 0 and 1. This model is appropriate for my dataset because it can easily handle numerical predictors and can provide probabilities for each class.

Random Forest:

Random Forest is an ensemble learning method that builds multiple decision trees and combines their outputs to make a final prediction. Each tree is trained on a random subset of the data with random feature selection, which reduces overfitting and increases robustness. This method is appropriate for my dataset because it can handle complex, non-linear relationships between features and the target variable, without the need for explicit feature engineering. It is also less sensitive to outliers compared to other models. Additionally, Random Forests can rank feature importance, helping to identify which clinical factors are most predictive of patient survival.

Support Vector Machine (SVM):

SVM is a powerful classification algorithm that works by finding the hyperplane that best separates different classes in the feature space. SVM tries to maximize the margin between the closest points (support vectors) of each class, which helps improve generalization. This algorithm is appropriate for my dataset because it works well with high-dimensional data and can handle both linear and non-linear relationships.

Part 8: Logistic Regression Models

Holdout Logistic Regression Model:

The basic logistic regression model was trained using the original dataset without any advanced tuning. Predictions were made on the holdout test set, and performance metrics were computed. This model may underperform because it doesn't leverage techniques like hyperparameter tuning, bagging, or boosting, which can help reduce overfitting and improve generalization.

Logistic Regression Hyperparameter Tuning:

For this model, I used hyperparameter tuning to optimize the logistic regression model using the glmnet method. This model was trained with a grid search for the hyperparameters alpha and lambda. Tuning allows the model to find the best balance between bias and variance, improving its predictive capability.

Best Hyperparameters for Logistic Regression Model

Hyperparameter	Value
Alpha	0.25
Lambda	0.10

Bagged Logistic Regression Model:

In the bagging model, I used bootstrapping to create multiple versions of the logistic regression model, each trained on different random samples of the data. Bagging reduces variance by averaging the predictions from multiple models, which helps improve the robustness of the model and avoid overfitting.

Boosted Logistic Regression Model:

The boosted logistic regression model was created using XGBoost, an implementation of gradient boosting. Boosting aims to reduce bias by focusing on the errors made by the previous models and iteratively correcting them.

Logistic Regression Models Summary Statistics:

Comparison of Logistic Regression Models with Key Metrics

Model	Accuracy	Precision	Recall	F1_Score	TPR	TNR	Holdout_AUC	CV_AUC
Logistic Regression (Original)	0.831	0.867	0.881	0.874	0.881	0.733	0.859	0.863
Logistic Regression (Tuned)	0.753	0.761	0.915	0.831	0.915	0.433	0.869	0.897
Logistic Regression (Bagging)	0.753	0.761	0.915	0.831	0.915	0.433	0.869	0.880
Logistic Regression (Boosting)	0.809	0.828	0.898	0.862	0.898	0.633	0.766	0.898

Logistic Model Evaluation:

A comparison of the four logistic regression models reveals that the Tuned Logistic Regression model is the best overall, achieving the highest CV AUC of 0.897, indicating strong generalization across the training set. It is important to note that for all models in this analysis, the “Positive” case is when the DEATH_EVENT is “No”. While it sacrifices some accuracy (0.753) and True Negative Rate (TNR) (0.433) in favor of a significantly improved recall (0.915), it delivers the highest F1 score (0.831), making it the most balanced and effective at identifying positive cases. The Original Logistic Regression model offers solid performance with a high accuracy (r round(accuracy, 3)) but falls short in TNR and AUC. The Bagging model mirrors the tuned model in recall (0.915) and F1 score (0.831) but has slightly lower CV AUC (0.88), while the Boosted Logistic Regression model, though performing well in terms of accuracy (0.809) and precision (0.828), has the lowest Holdout AUC (0.766), indicating it may not generalize as effectively as the tuned model.

Part 9: Random Forest Models

Original Random Forest Model:

The initial Random Forest model was trained using the original dataset, applying 500 trees and allowing the model to learn from multiple decision trees, each trained on a different subset of the data. This form of ensemble learning reduces overfitting by averaging predictions from multiple models, making the overall predictions more robust and generalized.

Random Forest Hyperparameter Tuning:

For the tuned Random Forest model, hyperparameter tuning was performed to optimize the mtry parameter, which determines the number of features considered for each split in the decision trees. The best mtry value was identified through cross-validation.

Best Hyperparameters for Random Forest

Hyperparameter	Value
Best mtry	2

Bagged Random Forest Model:

While bagging (Bootstrap Aggregating) is a core component of Random Forest, it is not necessary to apply bagging explicitly in this context. Random Forest inherently uses bagging, where each tree in the forest is trained on a random subset of the data. As a result, applying additional bagging methods would be redundant.

Boosted Random Forest Model:

Boosting involves sequentially training models where each new model attempts to correct the errors made by the previous model. However, boosting is not typically applied to Random Forest because it is a parallel model. Unlike boosting algorithms like XGBoost, Random Forest models work in parallel (each tree is built independently), and boosting methods (which rely on the sequential training of models) are not directly compatible with Random Forests. Additionally, the combination of bagging and boosting methods in one model (like trying to “boost” a Random Forest) is unnecessary and could lead to inefficiencies.

Random Forest Model Statistics:

Comparison of Random Forest Model Statistics

Model	Accuracy	Precision	Recall	F1_Score	TPR	TNR	Holdout_AUC	CV_AUC
Original Random Forest	0.820	0.864	0.864	0.864	0.864	0.733	0.905	0.898
Tuned Random Forest	0.854	0.871	0.915	0.893	0.915	0.733	0.901	0.905

Feature Importance for Random Forest Model

	Feature	Importance
time	time	29.644
serum_creatinine	serum_creatinine	11.183
age	age	9.809
ejection_fraction	ejection_fraction	8.828
platelets	platelets	8.123
creatinine_phosphokinase	creatinine_phosphokinase	7.340
serum_sodium	serum_sodium	6.276
high_blood_pressure	high_blood_pressure	1.727
diabetes	diabetes	1.416
anaemia	anaemia	1.385
sex	sex	1.340
smoking	smoking	1.136

Random Forest Model Evaluation:

The comparison between the Original and Tuned Random Forest models reveals a significant improvement in performance after hyperparameter tuning. The Tuned Random Forest model, with an optimized mtry value of 2, achieves a higher accuracy (0.854) and F1 score (0.893) compared to the Original model, indicating better overall balance between precision and recall. Notably, the Tuned model excels in recall (0.915) with a significant increase in sensitivity (TPR) to 0.915, making it more effective at identifying positive cases. Despite these gains, the Original Random Forest model still performs well, with an accuracy of 0.82 and a respectable TNR (specificity) of 0.733. In terms of generalization, both models perform well, with the Tuned Random Forest model achieving the highest CV AUC of 0.905, indicating better cross-validated performance. The Original Random Forest model has an AUC of 0.905, reflecting solid performance but not quite matching the Tuned model's enhanced generalization capability.

Part 10: Support Vector Machine Models

Original SVM Model:

This is the baseline SVM model. It uses the linear kernel by default, which assumes that the data can be separated by a straight line (or hyperplane in higher dimensions). The model is trained on the training dataset to predict whether the DEATH_EVENT is “Yes” or “No” based on the features. The cost parameter (C) is set to 1, which strikes a balance between achieving a low error on training data and maintaining the model's ability to generalize to new data.

SVM Hyperparameter Tuning:

The tuned SVM model improves upon the original by optimizing its hyperparameters, specifically the cost parameter (C). In this case, we use the Radial basis function (RBF) kernel (svmRadial), which is a non-linear kernel that allows the model to capture more complex decision boundaries between the classes.

SVM Hyperparameters

Hyperparameter	Value
Best C	2.000
Best Sigma	0.031

Bagged SVM Model:

Bagging (Bootstrap Aggregating) is an ensemble learning technique where multiple models are trained on different subsets of the training data and then combined (usually by averaging or voting) to improve the model's performance. The idea is that by training multiple models on random subsets of the data (with replacement), the model reduces variance and becomes more robust.

In the case of bagging with SVM, we apply the tuned SVM model from the previous step, but we train multiple versions of the model on bootstrapped subsets of the training data. Bagging helps to reduce overfitting by decreasing the model's sensitivity to small fluctuations in the training data. The final prediction is made by aggregating the predictions from all the individual SVM models.

Boosted SVM Model:

Boosting is another ensemble learning technique that combines the predictions of several base models (in this case, SVM models) in a sequential manner. The key idea behind boosting is that each new model focuses on the mistakes (misclassifications) made by the previous models. Boosting corrects errors iteratively, resulting in a more accurate and robust model.

In this case, the boosting algorithm used is XGBoost, a highly efficient and powerful gradient boosting method that combines multiple weak learners (SVMs in this case) into a strong learner. The XGBoost model applies a series of SVMs sequentially, where each new SVM model aims to correct the errors of the previous one.

SVM Model Statistics:

Comparison of SVM Model Statistics

Model	Accuracy	Precision	Recall	F1_Score	TPR	TNR	Holdout_AUC	CV_AUC
Original SVM	0.798	0.815	0.898	0.855	0.898	0.600	0.749	0.854
Tuned SVM	0.798	0.836	0.864	0.850	0.864	0.667	0.845	0.644
Bagged SVM	0.809	0.850	0.864	0.857	0.864	0.700	0.845	0.878
Boosted SVM	0.809	0.862	0.847	0.855	0.847	0.733	0.790	0.910

SVM Model Evaluation:

The comparison of the four SVM models reveals notable differences in performance. The Original SVM model, with an accuracy of 0.798 and a high recall of 0.898 (indicating a strong True Positive Rate, TPR), was effective at identifying positive cases, but it had lower specificity of 0.6, meaning it struggled with correctly identifying negative cases. Its CV AUC of 0.854 suggests solid generalization. The Tuned SVM showed slight improvements in precision (0.836), but had a slightly lower recall of 0.864 and a CV AUC of 0.644, indicating potential overfitting during tuning. The Bagged SVM model outperformed the others with an accuracy of 0.809, high precision of 0.85, recall of 0.864, and specificity of 0.7, pointing to a more balanced performance across all metrics. It also had a CV AUC of 0.878, reflecting better cross-validation performance. The Boosted SVM model achieved similar accuracy and precision (0.809 and 0.862) but excelled in specificity (0.733), showing it was more effective at correctly identifying negative cases. With the highest CV AUC of 0.91, it demonstrated the best generalization, making it the best-performing model overall despite a slightly lower holdout AUC of 0.79.

Part 11: Ensemble Model

In this ensemble model, three of the best-performing models — Boosted Logistic Regression (XGBoost), Tuned Random Forest, and Boosted SVM — were combined to improve predictive performance by leveraging the strengths of different algorithms. The ensemble approach averages the predicted probabilities of the positive class ("Yes") from each model and then makes the final prediction by applying a threshold of 0.5 to this average.

Averaging Ensemble Model:

This ensemble combines the predictions of the three best-performing models, and averages them to create a final prediction. This method leverages the diverse strengths of the three models, helping to mitigate the weaknesses of any single model and improving overall prediction accuracy.

Bagged Trees Ensemble Model:

The Bagged Trees Model uses bagging with decision trees to combine multiple trees trained on different subsets of the data. Bagging helps reduce variance and prevent overfitting, as it averages the predictions of several decision trees, each trained on a different bootstrap sample (randomly sampled data with replacement). In this case, the model uses bagging as the meta-model to combine the predictions from the individual models (XGBoost, Random Forest, and Boosted SVM). This model is effective because bagging reduces the overfitting tendency of decision trees and improves model generalization by averaging out errors from individual models.

Ensemble Models Statistics:

Comparison of Averaging Model and Meta-Model Performance

Model	Accuracy	Precision	Recall	F1_Score	TPR	TNR	Holdout_AUC	CV_AUC
Manual Averaging Model	0.831	0.867	0.881	0.874	0.881	0.733	0.894	0.900
Bagged Trees Meta-Model	0.731	0.857	0.706	0.774	0.706	0.778	0.859	0.822

Making Example Prediction with Meta Ensemble Model:

```
# Define new data point
new_data_point <- data.frame(
  age = 65,
  anaemia = 0,
  creatinine_phosphokinase = 400,
  diabetes = 1,
  ejection_fraction = 30,
  high_blood_pressure = 1,
  platelets = 250000,
  serum_creatinine = 1.2,
  serum_sodium = 135,
  sex = 1,
  smoking = 0,
  time = 10
)

# Display the predictions
cat("Final Predicted Class:", final_prediction_meta, "\n")
```

```
## Final Predicted Class: 2
```

```
print(final_prob_meta)
```

```
##     No  Yes
## 1 0.32 0.68
```

Ensemble Models Evaluation:

The Averaging Ensemble Model, which combines the predictions from XGBoost, Tuned Random Forest, and Boosted SVM, achieved an accuracy of 0.831 and demonstrated a high recall of 0.881, indicating its strong ability to correctly identify positive cases. The model also had an impressive F1 score of 0.874, reflecting a balanced performance between precision and recall. The Holdout AUC for the averaging model was 0.894, while the CV AUC was slightly higher at 0.9, highlighting the model's strong generalization. In comparison, the Bagged Trees Meta-Model, which uses bagging with decision trees to combine model predictions, showed a lower accuracy of 0.731 and recall of 0.706. However, it had a higher precision of 0.857 and an F1 score of 0.774. The Holdout AUC for the Bagged Trees Meta-Model was 0.859 and the CV AUC was 0.822, which demonstrated solid performance, though slightly lower than the Averaging Model. The results suggest that the Averaging Ensemble Model outperforms the Bagged Trees Meta-Model in overall accuracy and recall, while the Meta-Model shows more stability in terms of precision.

Part 12: Try Models on Different Datasets

Random Forest Model on Original vs. Imputed Datasets:

Comparison of Random Forest Model Performance (Original vs Imputed Data)

Model	Accuracy	Precision	Recall	F1_Score	TPR	TNR	AUC_Holdout	AUC_CV
Random Forest (Original Data)	0.820	0.864	0.864	0.864	0.864	0.733	0.905	0.898
Random Forest (Imputed Data)	0.216	0.400	0.091	0.148	0.091	0.591	0.840	0.811

SVM Model on Original vs. Normalized Dataset:

Comparison of SVM Model Performance (Original vs Normalized Data)

Model	Accuracy	Precision	Recall	F1_Score	TPR	TNR	AUC_Holdout	AUC_CV
SVM (Original Data)	0.798	0.815	0.898	0.855	0.898	0.600	0.749	0.854
SVM (Normalized Data)	0.807	0.831	0.900	0.864	0.900	0.607	0.754	0.856

In the comparison of SVM models on original versus normalized data, the model trained on normalized data (which involved techniques like Gaussian normalization and min-max scaling) outperformed the original data model in several key performance metrics. The accuracy of the normalized SVM model was 0.807, a notable improvement over the original model's 0.798. Additionally, the precision increased to 0.831 from 0.815, and the recall saw a significant boost to 0.9 compared to 0.898. This improvement in recall indicates that the normalized data model was better at identifying positive cases. The F1 score also improved, rising to 0.864 from 0.855, reflecting better balance between precision and recall. Furthermore, the Holdout AUC for the normalized data model was 0.754, slightly better than the original model's 0.749. However, the TNR (True Negative Rate) decreased from 0.6 to 0.607, suggesting a trade-off where the model becomes more sensitive but less specific. These results suggest that normalization, which standardizes the features, can significantly enhance the model's ability to capture the underlying patterns in the data, particularly for SVM.

When comparing Random Forest models on the original versus imputed data, the performance diverged sharply. The accuracy of the Random Forest model on the original data was 0.82, whereas on the imputed data, it dropped drastically to 0.216. Similarly, the precision and recall were also lower for the imputed model, indicating that imputing missing data (via median for continuous variables and mode for categorical variables) introduced significant noise, leading to poorer performance. The F1 score of 0.148 for the imputed data was substantially lower than the original model's score of 0.864, and the AUC dropped from 0.905 to 0.84.

Part 13: Compare All Models

Best Model for Predicting the Target Variable:

After considering all of the models I developed, the Averaging Ensemble Model stands out as the best overall choice for predicting patient survival. This ensemble method effectively combines the strengths of XGBoost, Tuned Random Forest, and Boosted SVM, achieving a high balance between precision and recall, strong generalization (as evidenced by its CV AUC of 0.9), and a robust F1 score of 0.874. Although the individual models showed strong performance, particularly the Tuned Random Forest and Boosted SVM, the ensemble approach harnesses the complementary strengths of these models to enhance prediction stability and accuracy, making it the optimal choice for this task.