Heart Failure Prediction

Load Packages and Data

```
library(pROC)
library(caret)
library(ggplot2)
library(dplyr)
library(tidyr)
library(corrplot)
library(PRROC)
library(ggbiplot)
library(factoextra)
library(rpart.plot)

# load data
hfp_data <- read.csv("data/heart_failure_clinical_records_dataset.csv")
```

1. Explore Dataset

```
# read structure/summary
str(hfp_data)
```

```
## 'data.frame':    299 obs. of  13 variables:
##  $ age                     : num  75 55 65 50 65 90 75 60 65 80 ...
##  $ anaemia                 : int  0 0 0 1 1 1 1 1 0 1 ...
##  $ creatinine_phosphokinase: int  582 7861 146 111 160 47 246 315 157 123 ...
##  $ diabetes                : int  0 0 0 0 1 0 0 1 0 0 ...
##  $ ejection_fraction       : int  20 38 20 20 20 40 15 60 65 35 ...
##  $ high_blood_pressure     : int  1 0 0 0 0 1 0 0 0 1 ...
##  $ platelets               : num  265000 263358 162000 210000 327000 ...
##  $ serum_creatinine        : num  1.9 1.1 1.3 1.9 2.7 2.1 1.2 1.1 1.5 9.4 ...
##  $ serum_sodium            : int  130 136 129 137 116 132 137 131 138 133 ...
##  $ sex                     : int  1 1 1 1 0 1 1 1 0 1 ...
##  $ smoking                 : int  0 0 1 0 0 1 0 1 0 1 ...
##  $ time                    : int  4 6 7 7 8 8 10 10 10 10 ...
##  $ DEATH_EVENT             : int  1 1 1 1 1 1 1 1 1 1 ...
```

```
summary(hfp_data)
```

```
##       age           anaemia       creatinine_phosphokinase    diabetes
##  Min.   :40.00   Min.   :0.0000   Min.   :  23.0           Min.   :0.0000
##  1st Qu.:51.00   1st Qu.:0.0000   1st Qu.: 116.5           1st Qu.:0.0000
##  Median :60.00   Median :0.0000   Median : 250.0           Median :0.0000
##  Mean   :60.83   Mean   :0.4314   Mean   : 581.8           Mean   :0.4181
##  3rd Qu.:70.00   3rd Qu.:1.0000   3rd Qu.: 582.0           3rd Qu.:1.0000
##  Max.   :95.00   Max.   :1.0000   Max.   :7861.0           Max.   :1.0000
##  ejection_fraction high_blood_pressure   platelets      serum_creatinine
##  Min.   :14.00     Min.   :0.0000      Min.   : 25100   Min.   :0.500
##  1st Qu.:30.00     1st Qu.:0.0000      1st Qu.:212500   1st Qu.:0.900
##  Median :38.00     Median :0.0000      Median :262000   Median :1.100
##  Mean   :38.08     Mean   :0.3512      Mean   :263358   Mean   :1.394
##  3rd Qu.:45.00     3rd Qu.:1.0000      3rd Qu.:303500   3rd Qu.:1.400
##  Max.   :80.00     Max.   :1.0000      Max.   :850000   Max.   :9.400
##   serum_sodium        sex            smoking            time
##  Min.   :113.0   Min.   :0.0000   Min.   :0.0000   Min.   :  4.0
##  1st Qu.:134.0   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 73.0
##  Median :137.0   Median :1.0000   Median :0.0000   Median :115.0
##  Mean   :136.6   Mean   :0.6488   Mean   :0.3211   Mean   :130.3
##  3rd Qu.:140.0   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:203.0
##  Max.   :148.0   Max.   :1.0000   Max.   :1.0000   Max.   :285.0
##   DEATH_EVENT
##  Min.   :0.0000
##  1st Qu.:0.0000
##  Median :0.0000
##  Mean   :0.3211
##  3rd Qu.:1.0000
##  Max.   :1.0000
```

```
head(hfp_data)
```

	a…	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction
	<dbl>	<int>	<int>	<int>	<int>
1	75	0	582	0	20
2	55	0	7861	0	38
3	65	0	146	0	20
4	50	1	111	0	20
5	65	1	160	1	20
6	90	1	47	0	40

6 rows | 1-6 of 14 columns

```
# check missing values
colSums(is.na(hfp_data))
```

```
##                      age                  anaemia creatinine_phosphokinase
##                        0                        0                        0
##                 diabetes        ejection_fraction      high_blood_pressure
##                        0                        0                        0
##                platelets         serum_creatinine             serum_sodium
##                        0                        0                        0
##                      sex                  smoking                     time
##                        0                        0                        0
##              DEATH_EVENT
##                        0
```

```
# distribution
table(hfp_data$DEATH_EVENT)
```

```
##
##   0   1
## 203  96
```

```
prop.table(table(hfp_data$DEATH_EVENT))
```

```
##
##         0         1
## 0.6789298 0.3210702
```

203 (67.8%) of patients survived, while 96 (32.1%) died.

Which variables are related to whether a patient died from heart failure (response variable (`DEATH_EVENT`)?

2. Exploratory Data Analysis

2. 1 Correlation Matrix

```
# correlation matrix
cor_matrix_y <- cor(hfp_data)
cor_matrix_y
```

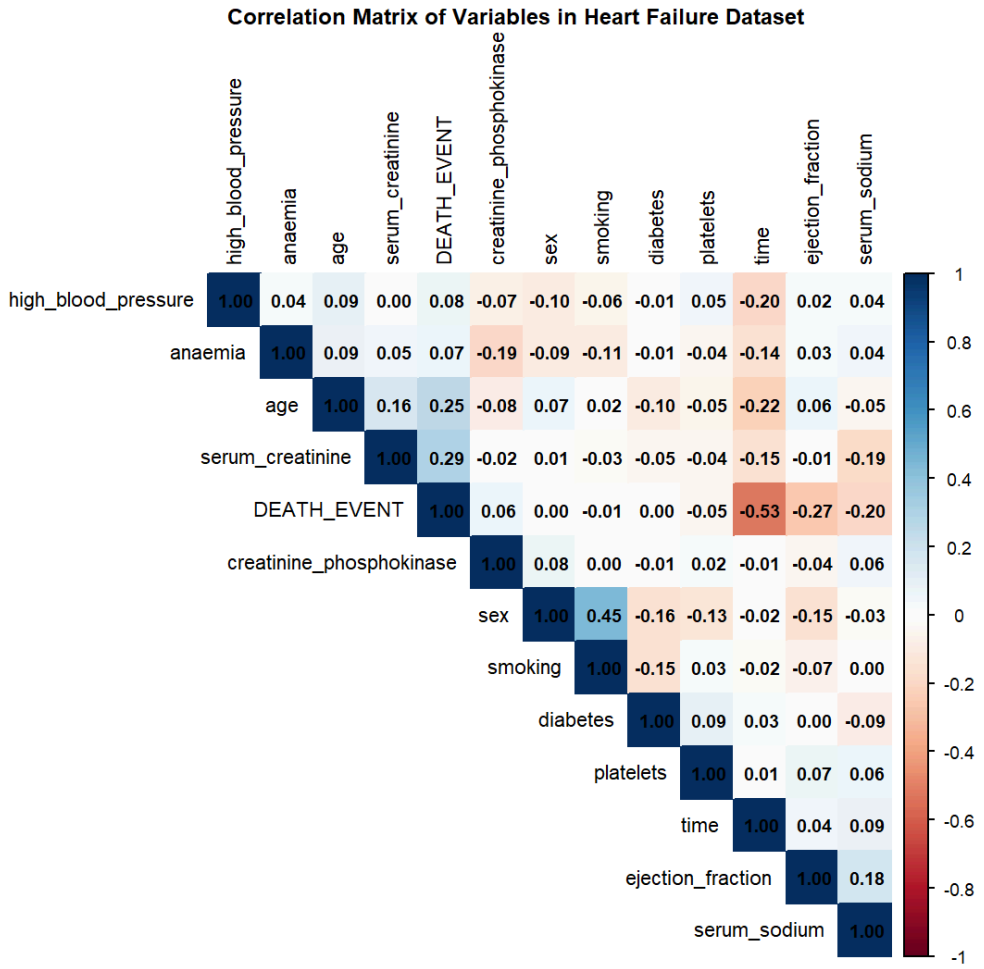
```
##                                age     anaemia creatinine_phosphokinase
## age                     1.00000000  0.08800644             -0.081583900
## anaemia                 0.08800644  1.00000000             -0.190741030
## creatinine_phosphokinase -0.08158390 -0.19074103              1.000000000
## diabetes                -0.10101239 -0.01272905             -0.009638514
## ejection_fraction        0.06009836  0.03155697             -0.044079554
## high_blood_pressure      0.09328868  0.03818200             -0.070589980
## platelets               -0.05235437 -0.04378555              0.024463389
## serum_creatinine         0.15918713  0.05217360             -0.016408480
## serum_sodium            -0.04596584  0.04188161              0.059550156
## sex                      0.06542952 -0.09476896              0.079790629
## smoking                  0.01866787 -0.10728984              0.002421235
## time                    -0.22406842 -0.14141398             -0.009345653
## DEATH_EVENT              0.25372854  0.06627010              0.062728160
##                            diabetes ejection_fraction high_blood_pressure
## age                     -0.101012385        0.06009836         0.093288685
## anaemia                 -0.012729046        0.03155697         0.038182003
## creatinine_phosphokinase -0.009638514       -0.04407955        -0.070589980
## diabetes                 1.000000000       -0.00485031        -0.012732382
## ejection_fraction       -0.004850310        1.00000000         0.024444731
## high_blood_pressure     -0.012732382        0.02444473         1.000000000
## platelets                0.092192828        0.07217747         0.049963481
## serum_creatinine        -0.046975315       -0.01130247        -0.004934525
## serum_sodium            -0.089550619        0.17590228         0.037109470
## sex                     -0.157729504       -0.14838597        -0.104614629
## smoking                 -0.147173413       -0.06731457        -0.055711369
## time                     0.033725509        0.04172924        -0.196439479
## DEATH_EVENT             -0.001942883       -0.26860331         0.079351058
##                           platelets serum_creatinine serum_sodium          sex
## age                     -0.05235437      0.159187133 -0.045965841  0.065429524
## anaemia                 -0.04378555      0.052173604  0.041881610 -0.094768961
## creatinine_phosphokinase  0.02446339     -0.016408480  0.059550156  0.079790629
## diabetes                  0.09219283     -0.046975315 -0.089550619 -0.157729504
## ejection_fraction         0.07217747     -0.011302475  0.175902282 -0.148385965
## high_blood_pressure       0.04996348     -0.004934525  0.037109470 -0.104614629
## platelets                 1.00000000     -0.041198077  0.062124619 -0.125120483
## serum_creatinine         -0.04119808      1.000000000 -0.189095210  0.006969778
## serum_sodium              0.06212462     -0.189095210  1.000000000 -0.027566123
## sex                      -0.12512048      0.006969778 -0.027566123  1.000000000
## smoking                   0.02823445     -0.027414135  0.004813195  0.445891712
## time                      0.01051391     -0.149315418  0.087640000 -0.015608220
## DEATH_EVENT              -0.04913887      0.294277561 -0.195203596 -0.004316376
##                              smoking         time  DEATH_EVENT
## age                      0.018667868 -0.224068420  0.253728543
## anaemia                 -0.107289838 -0.141413982  0.066270098
## creatinine_phosphokinase  0.002421235 -0.009345653  0.062728160
## diabetes                 -0.147173413  0.033725509 -0.001942883
## ejection_fraction        -0.067314567  0.041729235 -0.268603312
## high_blood_pressure      -0.055711369 -0.196439479  0.079351058
## platelets                 0.028234448  0.010513909 -0.049138868
## serum_creatinine         -0.027414135 -0.149315418  0.294277561
## serum_sodium              0.004813195  0.087640000 -0.195203596
```

```
## sex                       0.445891712 -0.015608220 -0.004316376
## smoking                   1.000000000 -0.022838942 -0.012623153
## time                     -0.022838942  1.000000000 -0.526963779
## DEATH_EVENT              -0.012623153 -0.526963779  1.000000000
```

```
# top predictors most correlated with DEATH_EVENT
cor_target <- cor_matrix_y["DEATH_EVENT", ]
top_5 <- sort(abs(cor_target[names(cor_target) != "DEATH_EVENT"]), decreasing = TRUE)[1:5]
top_5
```

```
##              time  serum_creatinine ejection_fraction               age
##         0.5269638         0.2942776         0.2686033         0.2537285
##      serum_sodium
##         0.1952036
```

```
# heatmap
corrplot(cor_matrix_y,
         method = "color",
         type = "upper",         # upper triangle
         order = "hclust",       # group similar variables
         addCoef.col = "black",  # show correlation values
         tl.cex = 0.7,           # axis text
         number.cex = 0.6,       # correlation values
         tl.col = "black",       # axis label color
         cl.cex = 0.6,           # color legend text size
         mar = c(1, 1, 2.5, 1))  # plot margins

title("Correlation Matrix of Variables in Heart Failure Dataset", cex.main = 0.75)
```



From this matrix, we can see that the top 5 predictors most correlated with `DEATH_EVENT` are `time`, `serum_creatinine`, `ejection_fraction`, `age`, and `serum_sodium`.

2.2 Boxplots of Continuous Features

To observe visual differences, we explored all continuous variables grouped by `DEATH_EVENT`.

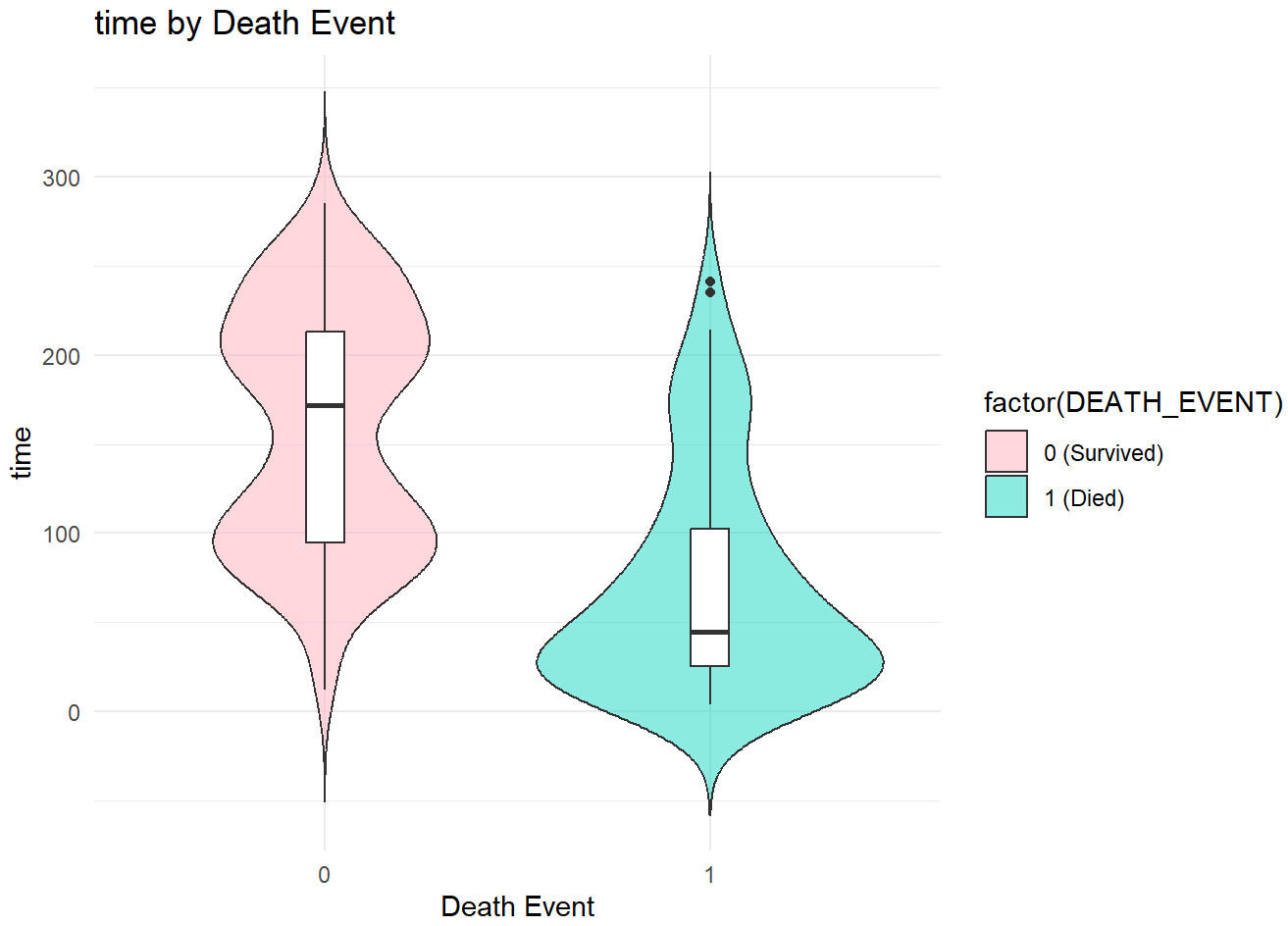
```
# ID binary vars and derive continuous ones
binary_vars <- c("sex", "diabetes", "high_blood_pressure", "smoking", "anaemia")
continuous_vars <- setdiff(names(hfp_data), c(binary_vars, "DEATH_EVENT"))

# include all continuous vars + DEATH_EVENT
box_vars <- hfp_data[, c(continuous_vars, "DEATH_EVENT")]

# pivot longer for faceted plotting
long_box <- pivot_longer(box_vars,
                         cols = -DEATH_EVENT,
                         names_to = "Variable",
                         values_to = "Value")

# plot
ggplot(long_box, aes(x = factor(DEATH_EVENT), y = Value, fill = factor(DEATH_EVENT))) +
  geom_boxplot(alpha = 0.6, outlier.color = "red", outlier.alpha = 0.3) +
  facet_wrap(~ Variable, scales = "free", ncol = 3) +
  labs(title = "Boxplots of All Continuous Variables by Death Event",
       x = "Death Event", y = "Value", fill = "Death Event") +
  scale_fill_manual(values = c("0" = "pink", "1" = "turquoise"),
                    labels = c("0 (Survived)", "1 (Died)")) +
  theme_minimal()
```

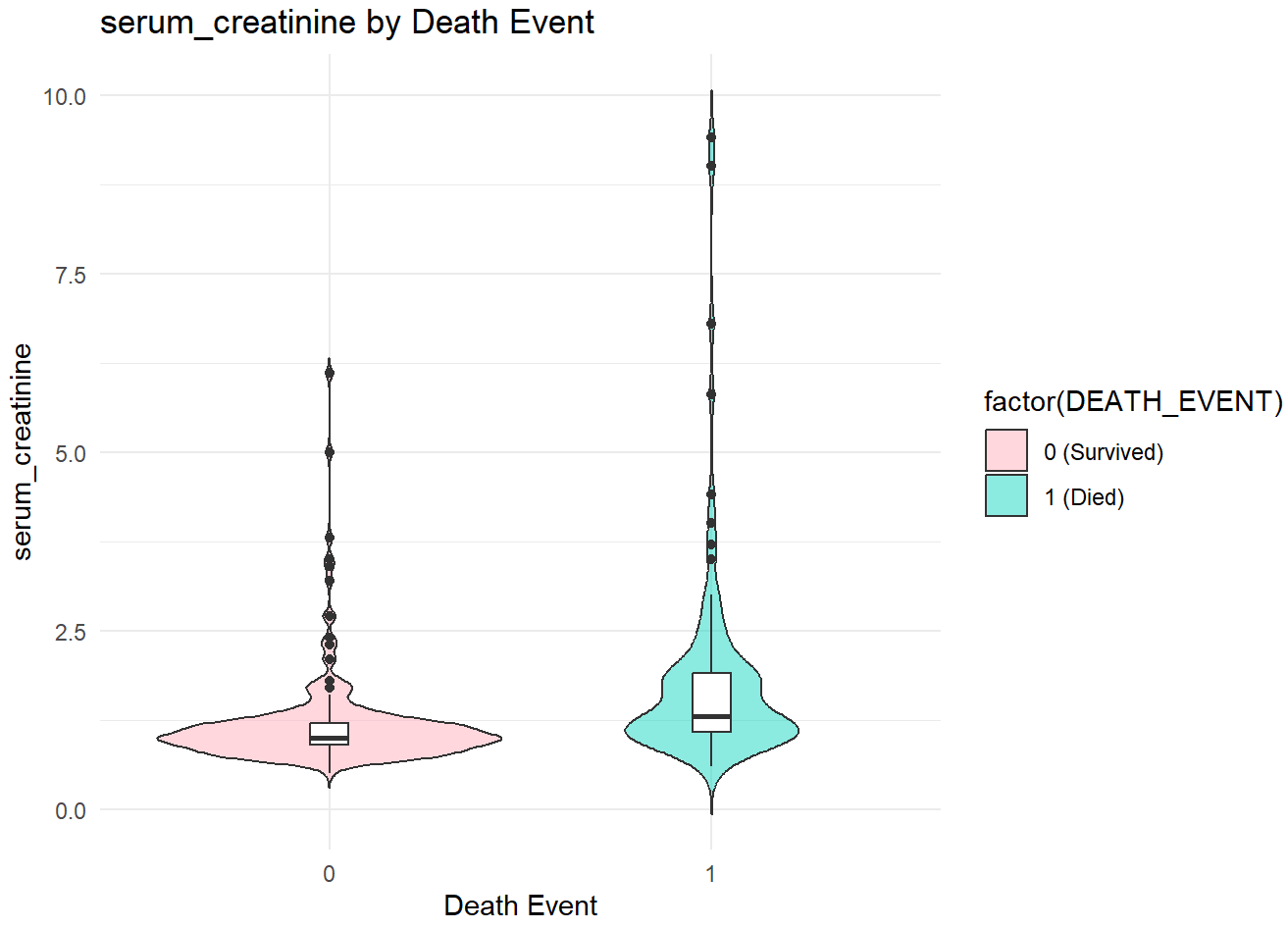
These boxplots align with the top predictors seen in the correlation matrix above.

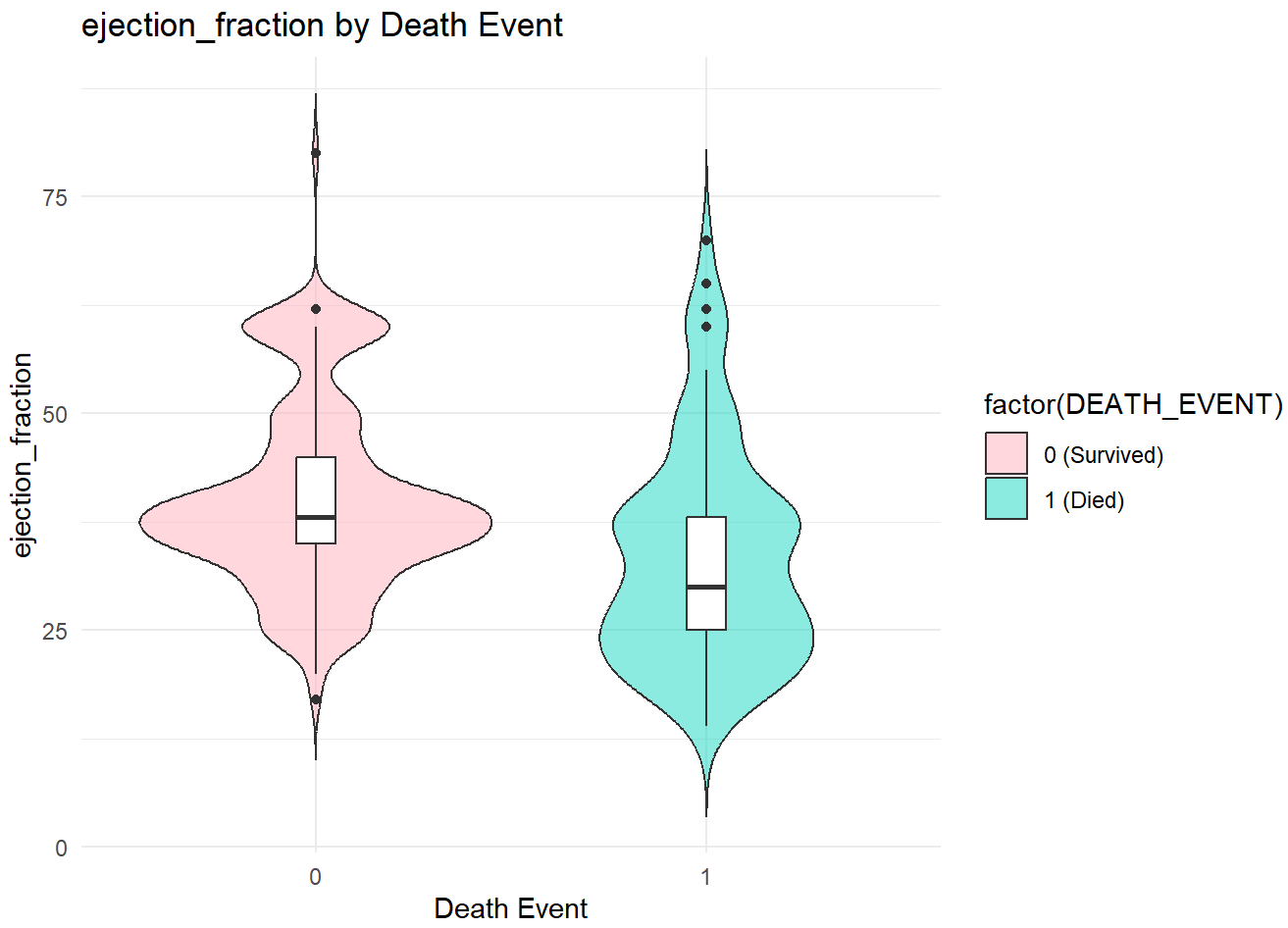
2.3 Violin Plots for Top Predictors

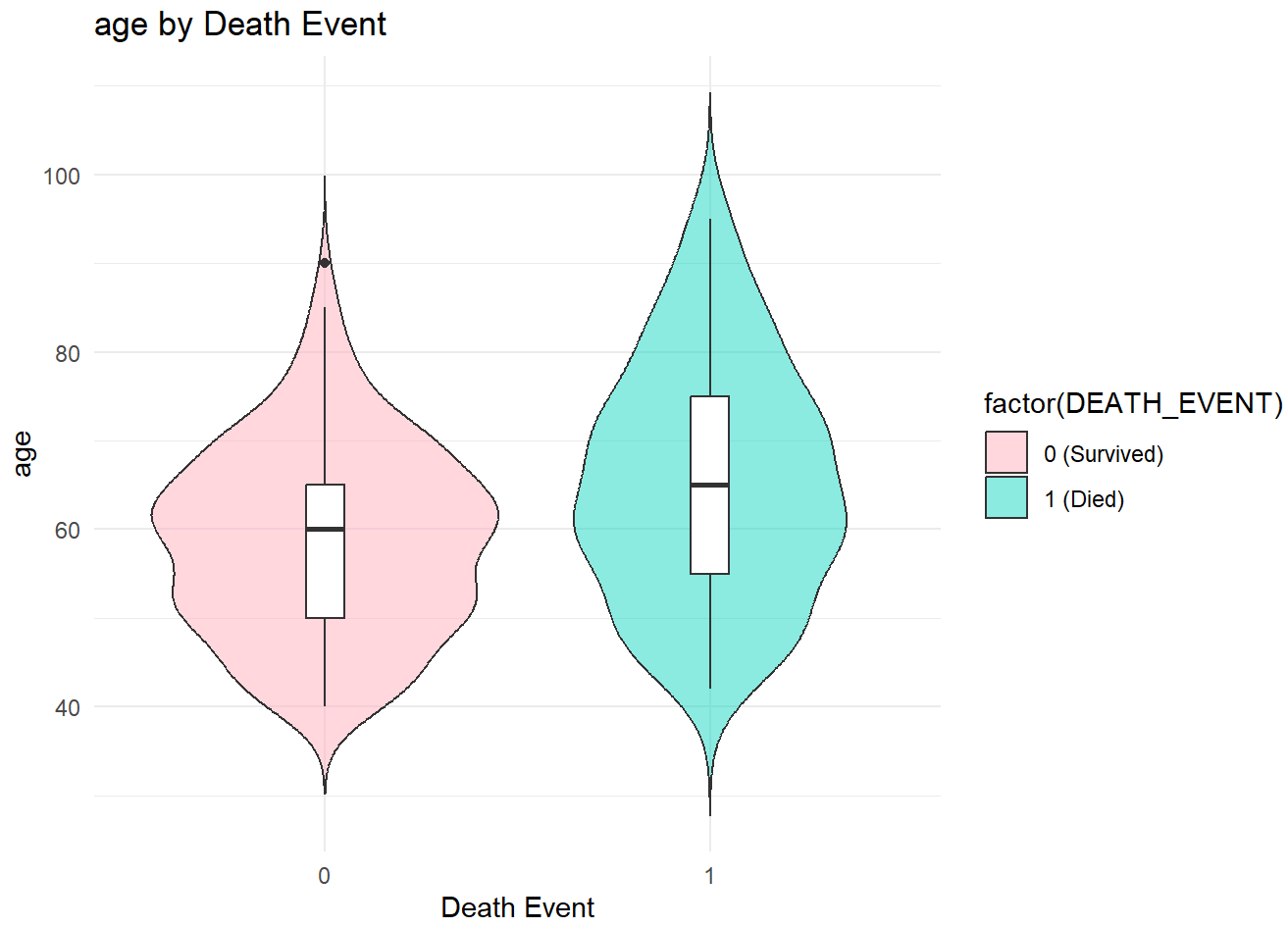
Based on the correlation and boxplots, we've selected the top 5 features to explore more deeply.

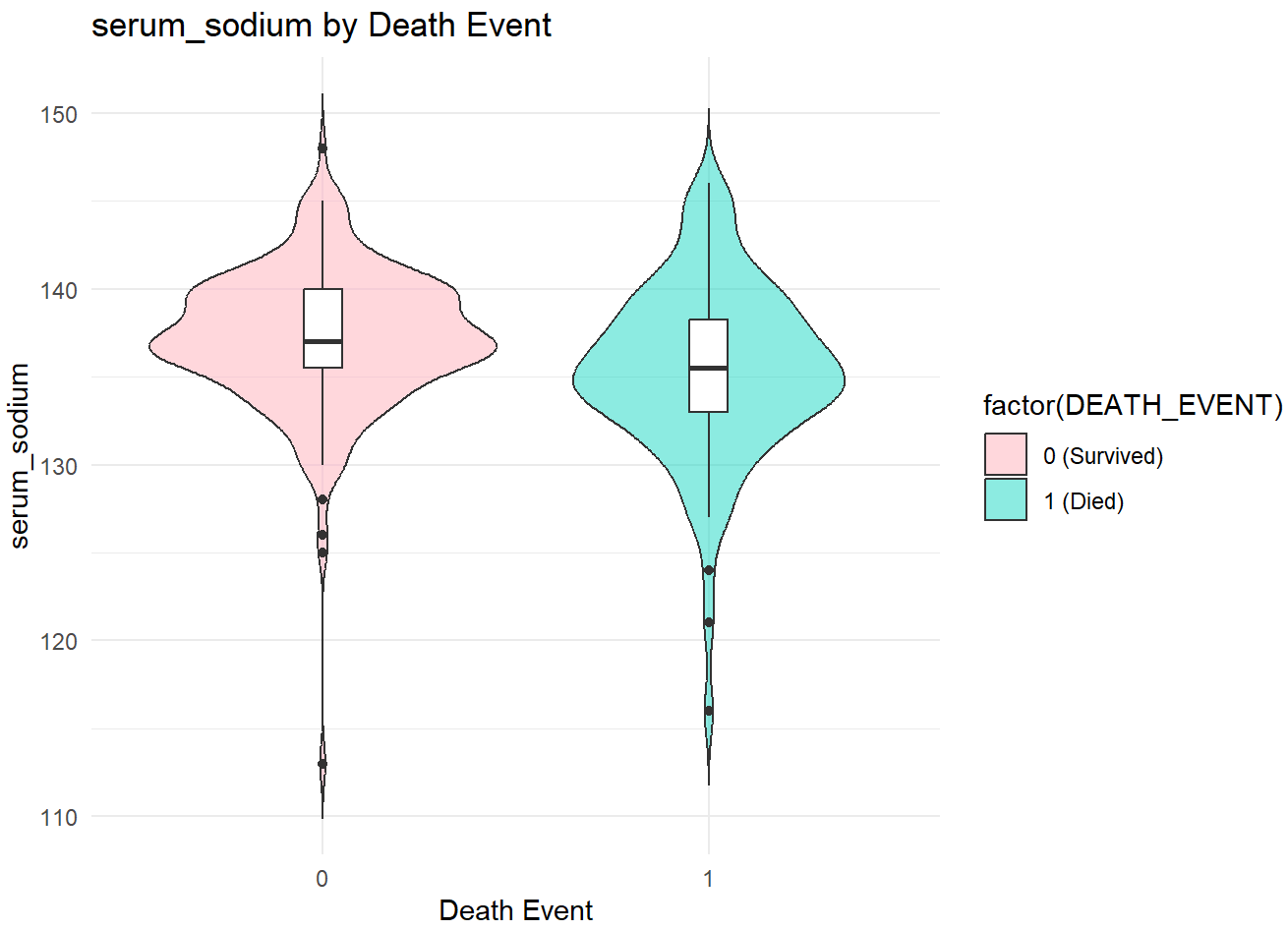
```
top_vars <- c("time", "serum_creatinine", "ejection_fraction", "age", "serum_sodium")

for (var in top_vars) {
  p <- ggplot(hfp_data, aes(x = factor(DEATH_EVENT), y = .data[[var]], fill = factor(DEATH_EVEN
T))) +
    geom_violin(trim = FALSE, alpha = 0.6) +
    geom_boxplot(width = 0.1, fill = "white") +
    scale_fill_manual(values = c("0" = "pink", "1" = "turquoise"),
                      labels = c("0 (Survived)", "1 (Died)")) +
    labs(title = paste(var, "by Death Event"), x = "Death Event", y = var) +
    theme_minimal()
  print(p)
}
```


time by Death Event
300
200
100
0
time
0
1
Death Event
factor(DEATH_EVENT)
0 (Survived)
1 (Died)


serum_creatinine by Death Event
10.0
7.5
5.0
2.5
0.0
serum_creatinine
0
1
Death Event
factor(DEATH_EVENT)
0 (Survived)
1 (Died)


ejection_fraction by Death Event
ejection_fraction
75
50
25
0
0
1
Death Event
factor(DEATH_EVENT)
0 (Survived)
1 (Died)


age by Death Event
age
100
80
60
40
0
1
Death Event
factor(DEATH_EVENT)
0 (Survived)
1 (Died)



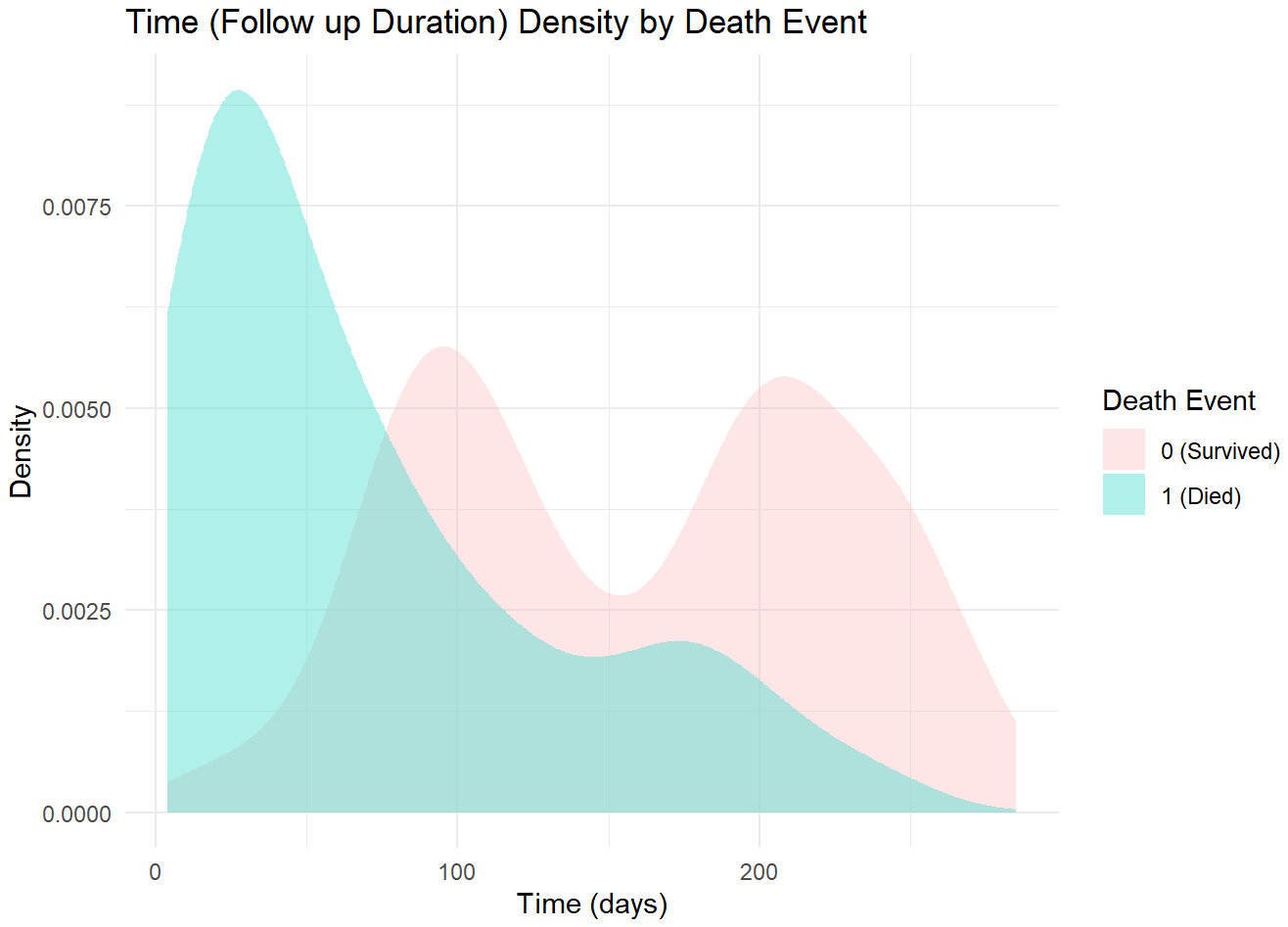
Each of these plots shows the value comparison of patients who survived (`DEATH_EVENT` =0), and died (`DEATH_EVENT` =1)

- `time` : patients who survived had a longer time where they followed up compared to those who died.
- `serum_creatinine` : shows more right-skewed distributions for patients who died, which shows that they had higher levels of serum creatinine, an indicator of poor kidney function.
- `ejection_fraction` : there was more instances of lower ejection fraction (how much blood the heart pumps out each heartbeat) in patients who died.
- `age` : the died group skews older, confirming that age is a risk factor, but still there is some overlap so it shouldn't be used as a sole predictor.
- `serum_sodium` : the distributions are a bit similar, with slightly lower sodium levels in patients who died. Since they are so similar, this variable might be weakly predictive.

2.4 Density Plot for Age

We wanted to explore how the top correlated variable (`time`) is distributed across the response group.

```
ggplot(hfp_data, aes(x = time, fill = factor(DEATH_EVENT))) +
  geom_density(alpha = 0.4, color = NA) +
  labs(
    title = "Time (Follow up Duration) Density by Death Event",
    x = "Time (days)",
    y = "Density",
    fill = "Death Event"
  ) +
  scale_fill_manual(values = c("0" = "pink", "1" = "turquoise"),
                    labels = c("0 (Survived)", "1 (Died)")) +
  theme_minimal()
```



This plot shows the estimated probability density of follow up-time (in days). The blue curve (deaths) is heavily concentrated at lower times (<50 days), and suggests that patients who died had shorter follow up durations. The pink curve shows a “bimodal” distribution (two peaks at ~75 and 200), and indicates a wider range and longer time duration among surviving patients. This feature may contribute more significantly to class separation and therefore reduce error training rate in models like LDA or Logistic Regression.

3. Preprocessing

3.1 Scaling Predictors

```
binary_vars <- c("sex", "diabetes", "high_blood_pressure", "smoking", "anaemia")
continuous_vars <- setdiff(names(hfp_data), c(binary_vars, "DEATH_EVENT"))

scaled_cont <- scale(hfp_data[, continuous_vars])
binary_data <- hfp_data[, binary_vars]

hfp_scaled <- cbind(as.data.frame(scaled_cont), binary_data, DEATH_EVENT = hfp_data$DEATH_EVENT)

head(hfp_scaled)
```

	age <dbl>	creatinine_phosphokinase <dbl>	ejection_fraction <dbl>	platelets <dbl>
1	1.1909487	0.000165451	-1.527997920	1.678834e-02
2	-0.4904571	7.502062717	-0.007064906	7.523048e-09
3	0.3502458	-0.449185725	-1.527997920	-1.036336e+00
4	-0.9108085	-0.485257493	-1.527997920	-5.455595e-01
5	0.3502458	-0.434757017	-1.527997920	6.507077e-01
6	2.4520030	-0.551217299	0.161927651	-6.069065e-01

6 rows | 1-5 of 14 columns

All continuous features were standardized using z-score scaling (mean = 0, sd = 1). This makes sure that the features have equal weight in distance-based methods. The binary features were kept the same to keep their categorical interpretation.

3.2 Train-Test Split

```
hfp_data$DEATH_EVENT <- factor(hfp_data$DEATH_EVENT, levels = c(0, 1), labels = c("Survived", "D
ied"))

set.seed(2025)

train_index <- createDataPartition(hfp_scaled$DEATH_EVENT, p = 0.7, list = FALSE)
train_data <- hfp_scaled[train_index, ]
test_data <- hfp_scaled[-train_index, ]
```

The dataset was split 70/30 using stratified sampling, which makes sure that the data distribution is consistent, keeping the same amount of instances where patients have died for the training and test set.

3.3 K-Fold Cross Validation

```
train_control <- trainControl(method="cv", number=10, classProbs = TRUE, summaryFunction = twoCl
assSummary, savePredictions = TRUE)
```

10-fold cross validation was set up for all models.

4. Model Training

We trained the following classifiers:

- Quadratic Discriminant Analysis (QDA)
- Linear Discriminant Analysis (LDA)
- Logistic Regression (LogReg)
- k-Nearest Neighbors (KNN)
- Support Vector Machine (SVM)
- Random Forest (RF)
- Decision Tree (DT)

Since there are binary values, we performed a conversion in the beginning to designate values for patients who died and survived.

```
#Classfication Models
# Convert DEATH_EVENT to a factor for classification since it is numeric (meant for regression)
#test_data$DEATH_EVENT <- as.factor(test_data$DEATH_EVENT)
#train_data$DEATH_EVENT <- as.factor(train_data$DEATH_EVENT)

test_data$DEATH_EVENT <- factor(test_data$DEATH_EVENT, levels = c(0, 1), labels = c("Survived",
"Died"))
train_data$DEATH_EVENT <- factor(train_data$DEATH_EVENT, levels = c(0, 1), labels = c("Survive
d", "Died"))

# QDA Model
set.seed(2025)
qda_model <- train(DEATH_EVENT ~ ., data = train_data, method = "qda", trControl = train_contro
l, metric = "ROC")
print(qda_model)
```

```
## Quadratic Discriminant Analysis
##
## 210 samples
##  12 predictor
##   2 classes: 'Survived', 'Died'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 189, 189, 188, 190, 189, 189, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.7677778  0.8928571  0.447619
```

```
# LDA Model
lda_model <- train(DEATH_EVENT ~ ., data = train_data, method = "lda", trControl = train_contro
l, metric = "ROC")
print(lda_model)
```

```
## Linear Discriminant Analysis
##
## 210 samples
##  12 predictor
##   2 classes: 'Survived', 'Died'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 188, 189, 190, 189, 189, 190, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.8396032  0.8980952  0.6404762
```

```
# Logistic Regression
logistic_regression_model <- train(DEATH_EVENT ~ ., data = train_data, method = "glm", family =
"binomial", trControl = train_control, metric = "ROC")
print (logistic_regression_model)
```

```
## Generalized Linear Model
##
## 210 samples
##  12 predictor
##   2 classes: 'Survived', 'Died'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 189, 189, 190, 188, 189, 189, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.8349206  0.8909524  0.6142857
```

```
# KNN
knn_model <- train(DEATH_EVENT ~ ., data = train_data, method = "knn", trControl = train_contro
l, metric = "ROC")
print (knn_model)
```

```
## k-Nearest Neighbors
##
## 210 samples
##  12 predictor
##   2 classes: 'Survived', 'Died'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 189, 189, 189, 188, 188, 189, ...
## Resampling results across tuning parameters:
##
##   k  ROC        Sens       Spec
##   5  0.7699206  0.9533333  0.3547619
##   7  0.8013889  0.9466667  0.3380952
##   9  0.7920238  0.9533333  0.3880952
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was k = 7.
```

```
# SVM
svm_model <- train(DEATH_EVENT ~ ., data = train_data, method = "svmRadial", trControl = train_c
ontrol, preProcess = c("center", "scale"), metric = "ROC")
print(svm_model)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 210 samples
##  12 predictor
##   2 classes: 'Survived', 'Died'
##
## Pre-processing: centered (12), scaled (12)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 189, 189, 190, 189, 189, 190, ...
## Resampling results across tuning parameters:
##
##   C     ROC        Sens       Spec
##   0.25  0.8511111  0.8709524  0.6095238
##   0.50  0.8535714  0.8842857  0.5928571
##   1.00  0.8499206  0.8771429  0.5595238
##
## Tuning parameter 'sigma' was held constant at a value of 0.05627339
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.05627339 and C = 0.5.
```

```
# Random Forest
rf_model <- train(DEATH_EVENT ~ ., data = train_data, method = "rf", trControl = train_control,
metric = "ROC")
print(rf_model)
```

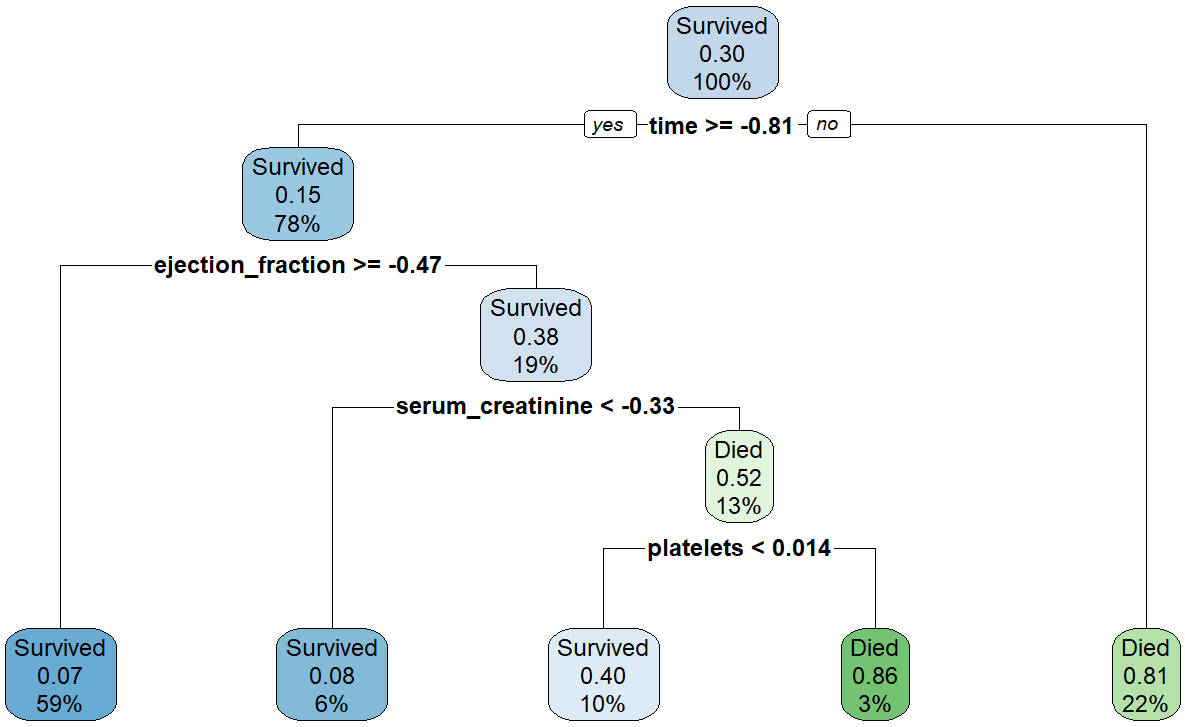
```
## Random Forest
##
## 210 samples
##  12 predictor
##   2 classes: 'Survived', 'Died'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 189, 189, 189, 189, 189, 188, ...
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##    2    0.8975907  0.9461905  0.5761905
##    7    0.8858050  0.8923810  0.6119048
##   12    0.8653288  0.8790476  0.5952381
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
# Decision Tree
dt_model <- train(DEATH_EVENT ~ ., data = train_data, method = "rpart", trControl = train_contro
l, metric = "ROC")
print(dt_model)
```

```
## CART
##
## 210 samples
##  12 predictor
##   2 classes: 'Survived', 'Died'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 190, 188, 189, 189, 189, 188, ...
## Resampling results across tuning parameters:
##
##   cp          ROC        Sens       Spec
##   0.01612903  0.7930952  0.8723810  0.6404762
##   0.02688172  0.7898413  0.8857143  0.6547619
##   0.46774194  0.5919048  0.9457143  0.2380952
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.01612903.
```

```
rpart.plot(dt_model$finalModel, main = "Decision Tree for Heart Failure Prediction")
```

Decision Tree for Heart Failure Prediction




```
dt_pred <- predict(dt_model, newdata = test_data)
confusionMatrix(dt_pred, test_data$DEATH_EVENT)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Survived Died
##   Survived       51    9
##   Died            4   25
##
##                Accuracy : 0.8539
##                  95% CI : (0.7632, 0.9199)
##     No Information Rate : 0.618
##     P-Value [Acc > NIR] : 9.176e-07
##
##                   Kappa : 0.6817
##
##  Mcnemar's Test P-Value : 0.2673
##
##             Sensitivity : 0.9273
##             Specificity : 0.7353
##          Pos Pred Value : 0.8500
##          Neg Pred Value : 0.8621
##              Prevalence : 0.6180
##          Detection Rate : 0.5730
##    Detection Prevalence : 0.6742
##       Balanced Accuracy : 0.8313
##
##        'Positive' Class : Survived
##
```

5. Model Evaluation: F1 and AUC

5.1 Utility Functions

```r
# Function to calculate F1 from confusion matrix
calculate_f1 <- function(cm) {
  precision <- cm$byClass["Pos Pred Value"]
  recall <- cm$byClass["Sensitivity"]

  # Handle division by zero
  if ((precision + recall) == 0) {
    return(NA)
  }
  f1 <- 2 * (precision * recall) / (precision + recall)
  return(round(f1, 4))
}

calculate_auc <- function(actual, predicted_probs, positive_label = "Died") {
  actual <- factor(actual, levels = c("Survived", "Died"))
  predicted_probs <- as.numeric(predicted_probs)
  roc_curve <- roc(response = actual, predictor = predicted_probs, levels = c("Survived", "Die
d"))
  auc_value <- auc(roc_curve)
  return(round(auc_value, 4))
}
```

5.2 Metrics per Model

We generated confusion matrices for each of the models, along with manually calculating F1-score and AUC.

```r
# Logistic Regression
logreg_pred <- predict(logistic_regression_model, newdata = test_data)
logreg_cm <- confusionMatrix(logreg_pred, test_data$DEATH_EVENT, positive = "Died")
logreg_f1 <- calculate_f1(logreg_cm)
print(logreg_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Survived Died
##   Survived       48    9
##   Died            7   25
##
##                Accuracy : 0.8202
##                  95% CI : (0.7245, 0.8936)
##     No Information Rate : 0.618
##     P-Value [Acc > NIR] : 3.019e-05
##
##                   Kappa : 0.6149
##
##  Mcnemar's Test P-Value : 0.8026
##
##             Sensitivity : 0.7353
##             Specificity : 0.8727
##          Pos Pred Value : 0.7812
##          Neg Pred Value : 0.8421
##              Prevalence : 0.3820
##          Detection Rate : 0.2809
##    Detection Prevalence : 0.3596
##       Balanced Accuracy : 0.8040
##
##        'Positive' Class : Died
##
```

```
cat("\nF1: ", logreg_f1, "\n") #Calculating F1
```

```
##
## F1:  0.7576
```

```
logreg_probs <- predict(logistic_regression_model, newdata = test_data, type = "prob")
auc_value <- calculate_auc(test_data$DEATH_EVENT, logreg_probs$Died) #Calculating AUC
cat("AUC:", auc_value, "\n\n\n")
```

```
## AUC: 0.9032
```

```
# LDA
lda_pred <- predict(lda_model, newdata = test_data)
lda_cm <- confusionMatrix(lda_pred, test_data$DEATH_EVENT, positive = "Died")
lda_f1 <- calculate_f1(lda_cm)
print(lda_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Survived Died
##   Survived       48    8
##   Died            7   26
##
##                Accuracy : 0.8315
##                  95% CI : (0.7373, 0.9025)
##     No Information Rate : 0.618
##     P-Value [Acc > NIR] : 1.018e-05
##
##                   Kappa : 0.641
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.7647
##             Specificity : 0.8727
##          Pos Pred Value : 0.7879
##          Neg Pred Value : 0.8571
##              Prevalence : 0.3820
##          Detection Rate : 0.2921
##    Detection Prevalence : 0.3708
##       Balanced Accuracy : 0.8187
##
##        'Positive' Class : Died
##
```

```
cat("\nF1: ", lda_f1, "\n")
```

```
##
## F1:  0.7761
```

```
lda_probs <- predict(lda_model, newdata = test_data, type = "prob")
auc_value <- calculate_auc(test_data$DEATH_EVENT, lda_probs$Died) #Calculating AUC
cat("AUC:", auc_value, "\n\n\n")
```

```
## AUC: 0.9203
```

```
# QDA
qda_pred <- predict(qda_model, newdata = test_data)
qda_cm <- confusionMatrix(qda_pred, test_data$DEATH_EVENT, positive = "Died")
qda_f1 <- calculate_f1(qda_cm)
print(qda_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Survived Died
##   Survived       47   17
##   Died            8   17
##
##                Accuracy : 0.7191
##                  95% CI : (0.6138, 0.8093)
##     No Information Rate : 0.618
##     P-Value [Acc > NIR] : 0.02998
##
##                   Kappa : 0.3734
##
##  Mcnemar's Test P-Value : 0.10960
##
##             Sensitivity : 0.5000
##             Specificity : 0.8545
##          Pos Pred Value : 0.6800
##          Neg Pred Value : 0.7344
##              Prevalence : 0.3820
##          Detection Rate : 0.1910
##    Detection Prevalence : 0.2809
##       Balanced Accuracy : 0.6773
##
##        'Positive' Class : Died
##
```

```
cat("\nF1: ", qda_f1, "\n")
```

```
##
## F1:  0.5763
```

```
qda_probs <- predict(qda_model, newdata = test_data, type = "prob")
auc_value <- calculate_auc(test_data$DEATH_EVENT, qda_probs$Died) #Calculating AUC
cat("AUC:", auc_value, "\n\n\n")
```

```
## AUC: 0.7289
```

```
# KNN
knn_pred <- predict(knn_model, newdata = test_data)
knn_cm <- confusionMatrix(knn_pred, test_data$DEATH_EVENT, positive = "Died")
knn_f1 <- calculate_f1(knn_cm)
print(knn_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Survived Died
##   Survived       48   19
##   Died            7   15
##
##                Accuracy : 0.7079
##                  95% CI : (0.6019, 0.7995)
##     No Information Rate : 0.618
##     P-Value [Acc > NIR] : 0.04905
##
##                   Kappa : 0.3366
##
##  Mcnemar's Test P-Value : 0.03098
##
##             Sensitivity : 0.4412
##             Specificity : 0.8727
##          Pos Pred Value : 0.6818
##          Neg Pred Value : 0.7164
##              Prevalence : 0.3820
##          Detection Rate : 0.1685
##    Detection Prevalence : 0.2472
##       Balanced Accuracy : 0.6570
##
##        'Positive' Class : Died
##
```

```
cat("\nF1: ", knn_f1, "\n")
```

```
##
## F1:  0.5357
```

```
knn_probs <- predict(knn_model, newdata = test_data, type = "prob")
auc_value <- calculate_auc(test_data$DEATH_EVENT, knn_probs$Died) #Calculating AUC
cat("AUC:", auc_value, "\n\n\n")
```

```
## AUC: 0.8142
```

```
# SVM
svm_pred <- predict(svm_model, newdata = test_data)
svm_cm <- confusionMatrix(svm_pred, test_data$DEATH_EVENT, positive = "Died")
svm_f1 <- calculate_f1(svm_cm)
print(svm_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Survived Died
##   Survived       46   10
##   Died            9   24
##
##                Accuracy : 0.7865
##                  95% CI : (0.6869, 0.8663)
##     No Information Rate : 0.618
##     P-Value [Acc > NIR] : 0.000518
##
##                   Kappa : 0.5453
##
##  Mcnemar's Test P-Value : 1.000000
##
##             Sensitivity : 0.7059
##             Specificity : 0.8364
##          Pos Pred Value : 0.7273
##          Neg Pred Value : 0.8214
##              Prevalence : 0.3820
##          Detection Rate : 0.2697
##    Detection Prevalence : 0.3708
##       Balanced Accuracy : 0.7711
##
##        'Positive' Class : Died
##
```

```
cat("\nF1: ", svm_f1, "\n")
```

```
##
## F1:  0.7164
```

```
svm_probs <- predict(svm_model, newdata = test_data, type = "prob")
auc_value <- calculate_auc(test_data$DEATH_EVENT, svm_probs$Died)  #Calculating AUC
cat("AUC:", auc_value, "\n\n\n")
```

```
## AUC: 0.8861
```

```
# Random Forest
rf_pred <- predict(rf_model, newdata = test_data)
rf_cm <- confusionMatrix(rf_pred, test_data$DEATH_EVENT, positive = "Died")
rf_f1 <- calculate_f1(rf_cm)
print(rf_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Survived Died
##   Survived       51   12
##   Died            4   22
##
##                Accuracy : 0.8202
##                  95% CI : (0.7245, 0.8936)
##     No Information Rate : 0.618
##     P-Value [Acc > NIR] : 3.019e-05
##
##                   Kappa : 0.6013
##
##  Mcnemar's Test P-Value : 0.08012
##
##             Sensitivity : 0.6471
##             Specificity : 0.9273
##          Pos Pred Value : 0.8462
##          Neg Pred Value : 0.8095
##              Prevalence : 0.3820
##          Detection Rate : 0.2472
##    Detection Prevalence : 0.2921
##       Balanced Accuracy : 0.7872
##
##        'Positive' Class : Died
##
```

```
cat("\nF1: ", rf_f1, "\n")
```

```
##
## F1:  0.7333
```

```
rf_probs <- predict(rf_model, newdata = test_data, type = "prob")
auc_value <- calculate_auc(test_data$DEATH_EVENT, rf_probs$Died) #Calculating AUC
cat("AUC:", auc_value, "\n\n\n")
```

```
## AUC: 0.9516
```

```
# Decision Tree
dt_pred <- predict(dt_model, newdata = test_data)
dt_cm <- confusionMatrix(dt_pred, test_data$DEATH_EVENT, positive = "Died")
dt_f1 <- calculate_f1(dt_cm)
print(dt_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Survived Died
##   Survived       51    9
##   Died            4   25
##
##                Accuracy : 0.8539
##                  95% CI : (0.7632, 0.9199)
##     No Information Rate : 0.618
##     P-Value [Acc > NIR] : 9.176e-07
##
##                   Kappa : 0.6817
##
##  Mcnemar's Test P-Value : 0.2673
##
##             Sensitivity : 0.7353
##             Specificity : 0.9273
##          Pos Pred Value : 0.8621
##          Neg Pred Value : 0.8500
##              Prevalence : 0.3820
##          Detection Rate : 0.2809
##    Detection Prevalence : 0.3258
##       Balanced Accuracy : 0.8313
##
##        'Positive' Class : Died
##
```

```
cat("\nF1: ", dt_f1, "\n")
```

```
##
## F1:  0.7937
```

```
dt_probs <- predict(dt_model, newdata = test_data, type = "prob")
auc_value <- calculate_auc(test_data$DEATH_EVENT, dt_probs$Died) #Calculating AUC
cat("AUC:", auc_value, "\n\n\n")
```

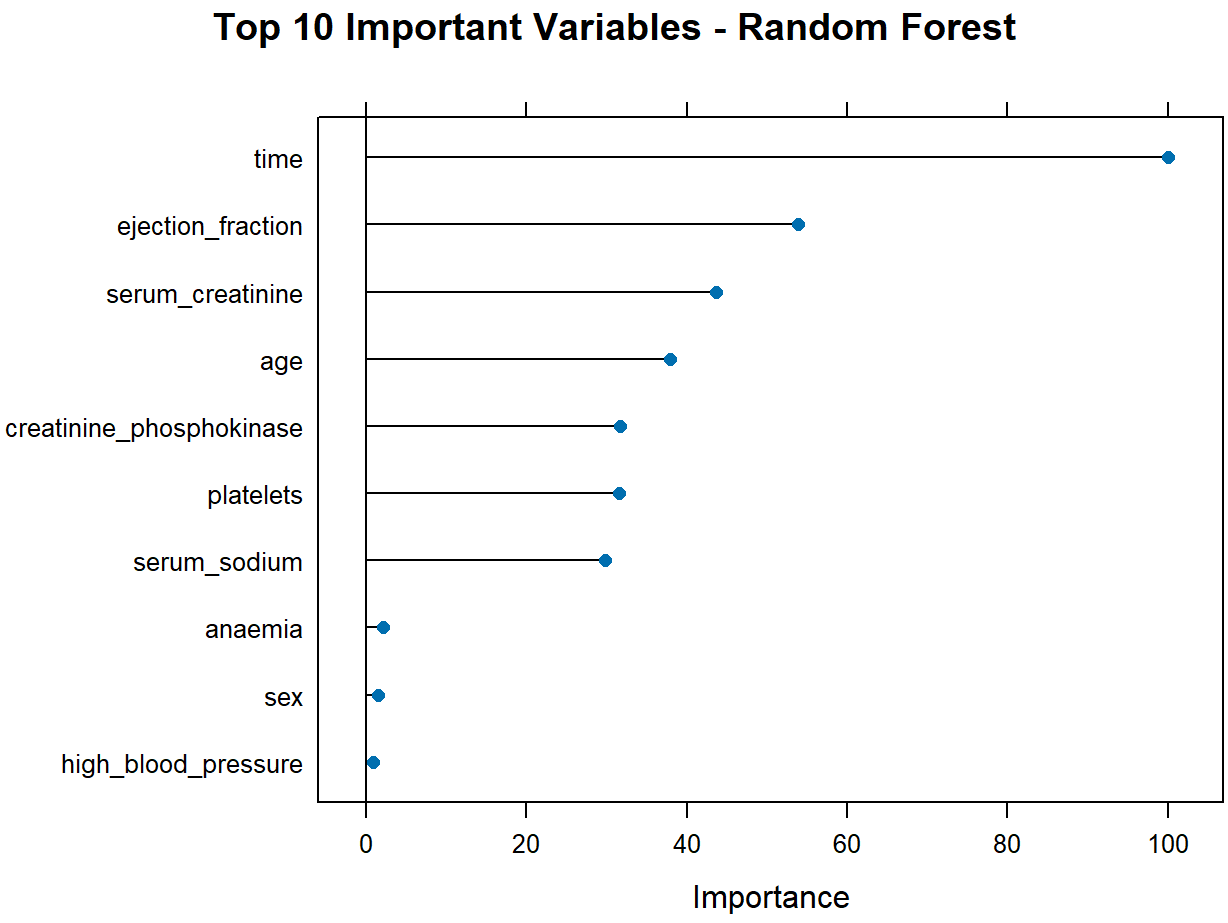
```
## AUC: 0.8639
```

5.3 Variable Importance for Random Forest

```
# variable importance for rf
library(caret)
importance_rf <- varImp(rf_model)
print(importance_rf)
```

```
## rf variable importance
##
##                           Overall
## time                     100.0000
## ejection_fraction         53.9184
## serum_creatinine          43.7095
## age                       37.9317
## creatinine_phosphokinase  31.6159
## platelets                 31.5847
## serum_sodium              29.7679
## anaemia                    2.1262
## sex                        1.4597
## high_blood_pressure        0.9020
## smoking                    0.6079
## diabetes                   0.0000
```

```
# top 10 important variables
plot(importance_rf, top = 10, main = "Top 10 Important Variables - Random Forest")
```



varImp() IDs which predictors had the biggest impact on model performance. In RF, var. importance is calculated based on how much each variable reduces impurity across all trees. The top variables are `time`, `serum_creatinine`, `ejection_fraction`, and `age`, which aligns with domain knowledge-> indicators of heart failure progression.

6. Performance Comparison

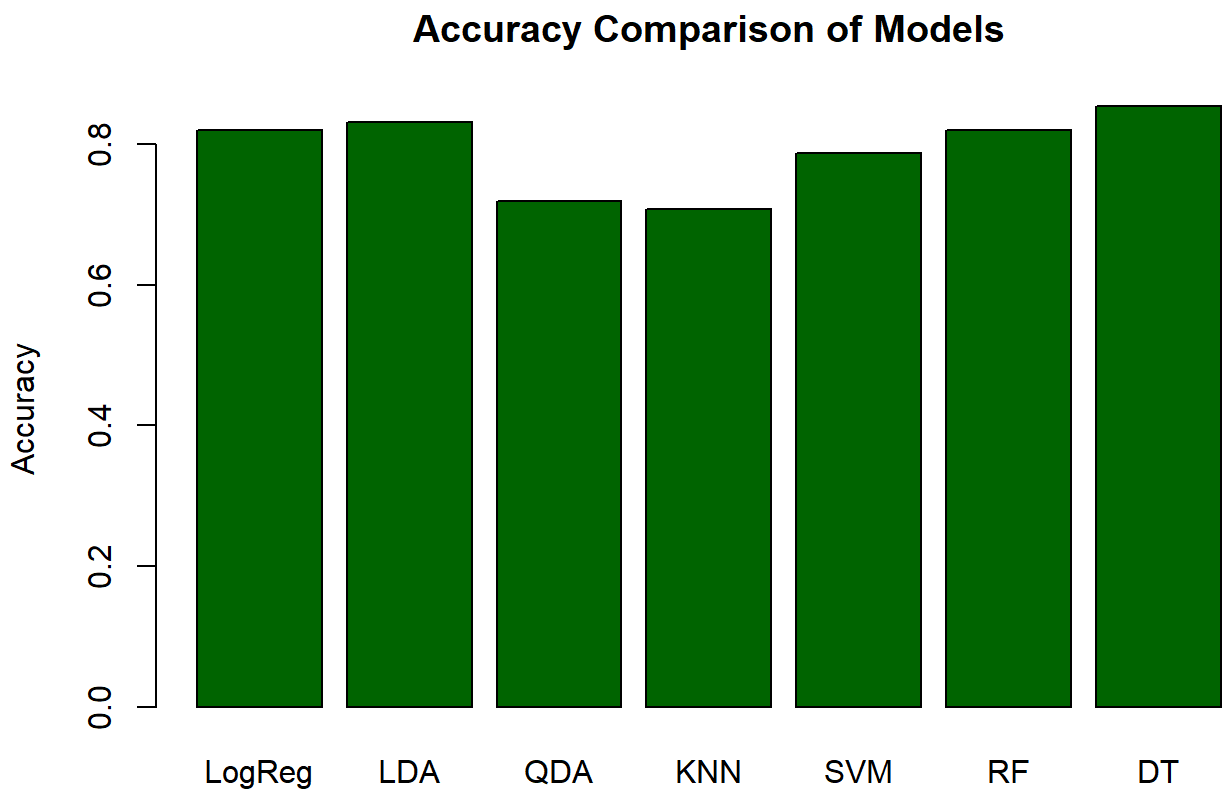
6.1 Summary

```
results <- resamples(list(SVM = svm_model, LogReg = logistic_regression_model, LDA = lda_model,
QDA = qda_model, RF = rf_model, KNN = knn_model))
summary(results)
```

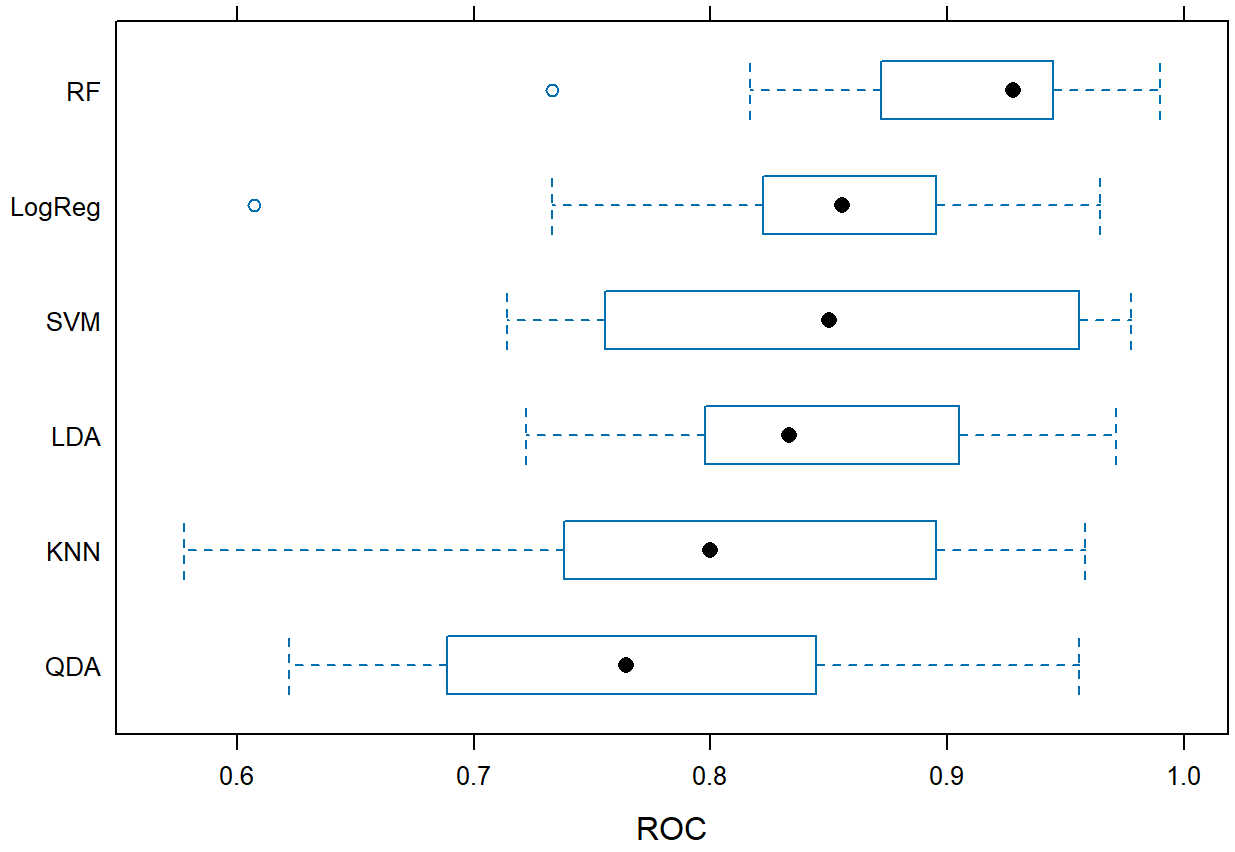
```
##
## Call:
## summary.resamples(object = results)
##
## Models: SVM, LogReg, LDA, QDA, RF, KNN
## Number of resamples: 10
##
## ROC
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## SVM    0.7142857 0.7750000 0.8500000 0.8535714 0.9444444 0.9777778    0
## LogReg 0.6071429 0.8261905 0.8555556 0.8349206 0.8908730 0.9642857    0
## LDA    0.7222222 0.8037698 0.8333333 0.8396032 0.8869048 0.9714286    0
## QDA    0.6222222 0.7000000 0.7642857 0.7677778 0.8285714 0.9555556    0
## RF     0.7333333 0.8729167 0.9277778 0.8975907 0.9416667 0.9897959    0
## KNN    0.5777778 0.7396825 0.8000000 0.8013889 0.8908730 0.9583333    0
##
## Sens
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## SVM    0.7142857 0.8821429 0.9333333 0.8842857 0.9333333 0.9333333    0
## LogReg 0.7857143 0.8595238 0.9000000 0.8909524 0.9333333 1.0000000    0
## LDA    0.8000000 0.8571429 0.8666667 0.8980952 0.9833333 1.0000000    0
## QDA    0.7333333 0.8666667 0.8976190 0.8928571 0.9333333 1.0000000    0
## RF     0.8666667 0.9333333 0.9333333 0.9461905 0.9833333 1.0000000    0
## KNN    0.8000000 0.9333333 0.9666667 0.9466667 1.0000000 1.0000000    0
##
## Spec
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## SVM    0.5000000 0.5000000 0.5833333 0.5928571 0.6666667 0.7142857    0
## LogReg 0.3333333 0.5000000 0.6666667 0.6142857 0.6666667 1.0000000    0
## LDA    0.5000000 0.5416667 0.6666667 0.6404762 0.6666667 0.8571429    0
## QDA    0.1666667 0.3333333 0.3809524 0.4476190 0.6250000 0.8333333    0
## RF     0.3333333 0.4464286 0.5833333 0.5761905 0.6666667 1.0000000    0
## KNN    0.1428571 0.1666667 0.3333333 0.3380952 0.5000000 0.5714286    0
```

6.2 Comparison Plots (Barplot and Boxplot of Model Performance)

```
# Boxplot for Accuracy
## bwplot(results, metric = "Accuracy")
get_accuracy <- function(cm) {
  accuracy <- as.numeric(cm$overall["Accuracy"])
  return(round(accuracy, 4))
}

logreg_acc <- get_accuracy(logreg_cm)
lda_acc <- get_accuracy(lda_cm)
qda_acc <- get_accuracy(qda_cm)
knn_acc <- get_accuracy(knn_cm)
svm_acc <- get_accuracy(svm_cm)
rf_acc <- get_accuracy(rf_cm)
dt_acc <- get_accuracy(dt_cm)

accuracy_df <- data.frame(Model = c("LogReg", "LDA", "QDA", "KNN", "SVM","RF", "DT"),
                          Accuracy = c(logreg_acc, lda_acc, qda_acc, knn_acc, svm_acc, rf_acc, d
t_acc))

# Bar plot for accuracy - used this instead of boxplot because the accuracy were singe points so
boxplot was a line rather than box
barplot(accuracy_df$Accuracy, names.arg = accuracy_df$Model, col = "darkgreen",
        main = "Accuracy Comparison of Models", ylab = "Accuracy")
```

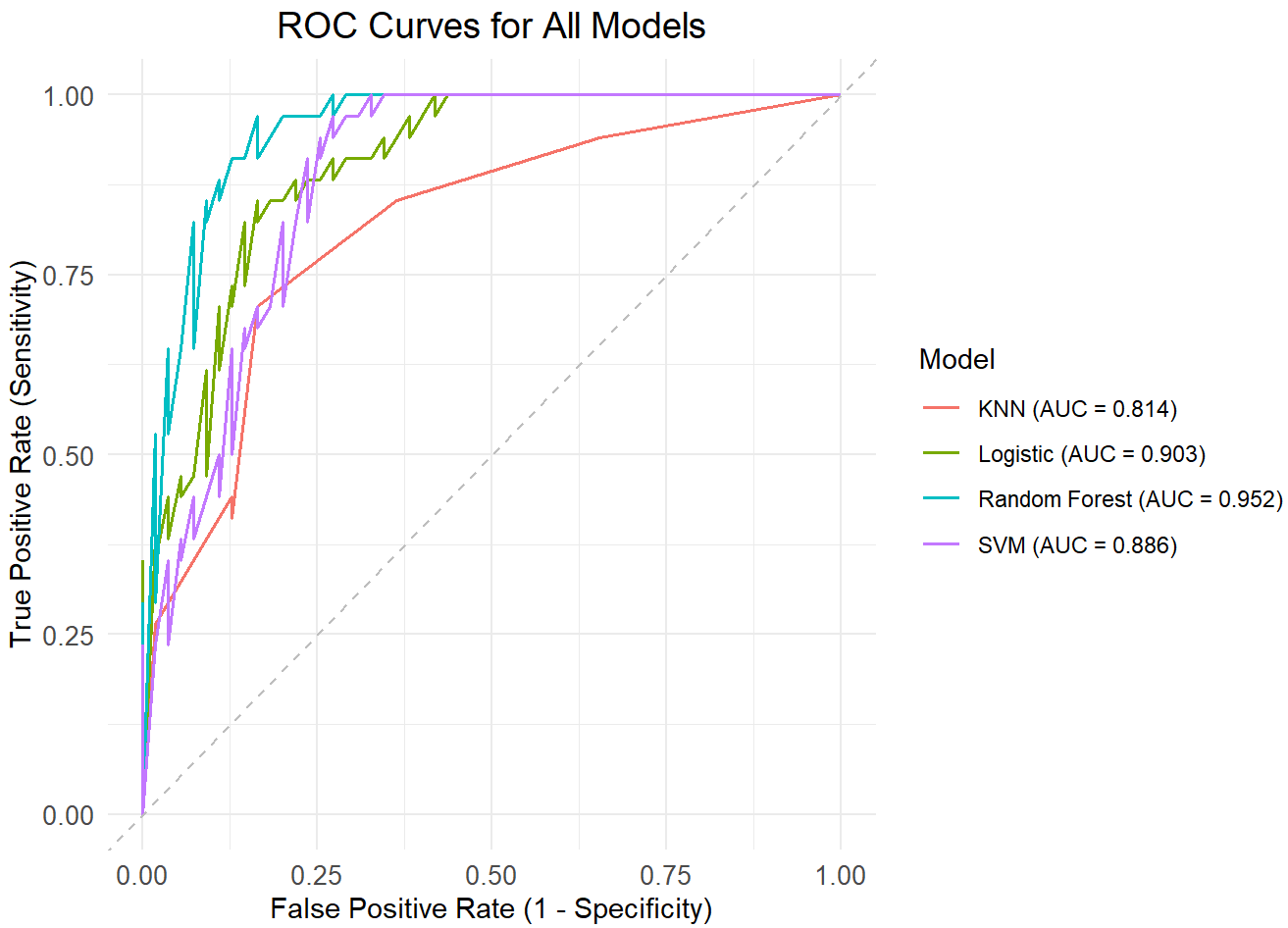


```
# Boxplot for F1 Score
f1_results <- data.frame(Model = c("Logistic Regression", "LDA", "QDA", "KNN", "SVM", "Random Fo
rest", "Decision Tree"),
                         F1_Score = c(logreg_f1, lda_f1, qda_f1, knn_f1, svm_f1, rf_f1, dt_f1))
# boxplot(F1_Score ~ Model, data = f1_results, col = "lightblue",
#         main = "F1 Score Comparison of Models", ylab = "F1 Score")


# Boxplot for AUC (ROC)
bwplot(results, metric = "ROC")
```



6.3 ROC Curve

```r
# function to extract ROC curve data
extract_roc_df <- function(roc_obj, model_name) {
  data.frame(
    FPR = 1 - roc_obj$specificities,
    TPR = roc_obj$sensitivities,
    Model = model_name
  )
}

true_labels <- test_data$DEATH_EVENT

# calculate ROC curves and AUCs
roc_logreg <- roc(true_labels, logreg_probs$Died)
roc_knn    <- roc(true_labels, knn_probs$Died)
roc_svm    <- roc(true_labels, svm_probs$Died)
roc_rf     <- roc(true_labels, rf_probs$Died)

# get AUC values
auc_logreg <- round(auc(roc_logreg), 3)
auc_knn    <- round(auc(roc_knn), 3)
auc_svm    <- round(auc(roc_svm), 3)
auc_rf     <- round(auc(roc_rf), 3)

# combine data
roc_data <- rbind(
  extract_roc_df(roc_logreg, paste0("Logistic (AUC = ", auc_logreg, ")")),
  extract_roc_df(roc_knn, paste0("KNN (AUC = ", auc_knn, ")")),
  extract_roc_df(roc_svm, paste0("SVM (AUC = ", auc_svm, ")")),
  extract_roc_df(roc_rf, paste0("Random Forest (AUC = ", auc_rf, ")"))
)


# plot
ggplot(roc_data, aes(x = FPR, y = TPR, color = Model)) +
  geom_line(size = 0.6) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed", color = "grey") +
  labs(title = "ROC Curves for All Models",
       x = "False Positive Rate (1 - Specificity)",
       y = "True Positive Rate (Sensitivity)") +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 14),
    axis.text = element_text(size = 10),
    axis.title = element_text(size = 11)
  )
```



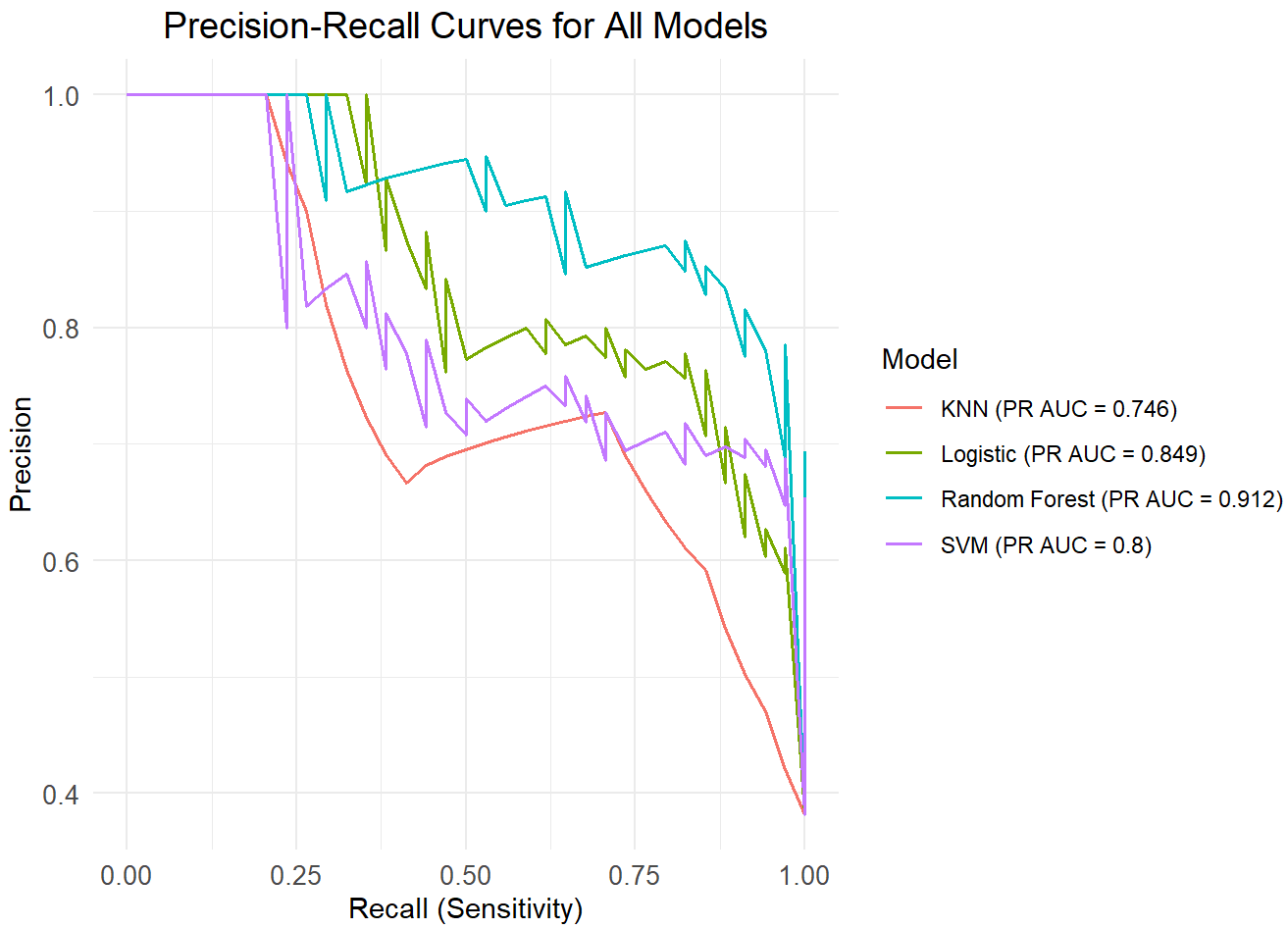
AUC shows probability that random positive class instance (Died) is ranked higher than random negative class (Survived) in predicted probability. It is more informative when class distribution is imbalanced (ex. `hfp_data` 68/32) in comparison to accuracy and F1-score since they are threshold-independent and not affected by skewed data.

- **Random Forest** performs the best (AUC=0.952), and the top-left curve indicates a high sensitive and low false positive rate
- **Logistic Regression** and **Support Vector Machine** perform strongly (AUC=0.903, 0.886 respectively)
- **K-Nearest Neighbour** is least effective out of the 4 (AUC=0.814). This might be because of noise, high variance, “curse of dimensionality”.

6.4 PR Curves

```r
# PR curves
pr_logreg <- pr.curve(scores.class0 = logreg_probs$Died[true_labels == "Died"],
                      scores.class1 = logreg_probs$Died[true_labels == "Survived"],
                      curve = TRUE)

pr_knn <- pr.curve(scores.class0 = knn_probs$Died[true_labels == "Died"],
                   scores.class1 = knn_probs$Died[true_labels == "Survived"],
                   curve = TRUE)

pr_svm <- pr.curve(scores.class0 = svm_probs$Died[true_labels == "Died"],
                   scores.class1 = svm_probs$Died[true_labels == "Survived"],
                   curve = TRUE)

pr_rf <- pr.curve(scores.class0 = rf_probs$Died[true_labels == "Died"],
                  scores.class1 = rf_probs$Died[true_labels == "Survived"],
                  curve = TRUE)

# extract PR curve as data frame
extract_pr_df <- function(pr_obj, model_name) {
  data.frame(
    Recall = pr_obj$curve[, 1],
    Precision = pr_obj$curve[, 2],
    Model = model_name
  )
}

# PR AUCs
auc_pr_logreg <- round(pr_logreg$auc.integral, 3)
auc_pr_knn    <- round(pr_knn$auc.integral, 3)
auc_pr_svm    <- round(pr_svm$auc.integral, 3)
auc_pr_rf     <- round(pr_rf$auc.integral, 3)

# combine to one df for ggplot
pr_data <- rbind(
  extract_pr_df(pr_logreg, paste0("Logistic (PR AUC = ", auc_pr_logreg, ")")),
  extract_pr_df(pr_knn, paste0("KNN (PR AUC = ", auc_pr_knn, ")")),
  extract_pr_df(pr_svm, paste0("SVM (PR AUC = ", auc_pr_svm, ")")),
  extract_pr_df(pr_rf, paste0("Random Forest (PR AUC = ", auc_pr_rf, ")"))
)

# plot
ggplot(pr_data, aes(x = Recall, y = Precision, color = Model)) +
  geom_line(size = 0.6) +
  labs(title = "Precision-Recall Curves for All Models",
       x = "Recall (Sensitivity)",
       y = "Precision",
       color = "Model") +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 14),
    axis.text = element_text(size = 10),
```

```
    axis.title = element_text(size = 11)
  )
```



PR curve focuses on positive class ("Died") by plotting precision vs. recall. More informative than ROC AUC when class distribution is imbalanced, since it directly evaluates model ability to identify true positives.

- **Random Forest** has highest AUC-PR (0.912) which shows its ability to identify actual deaths while also keeping high precision.
- **Logistic Regression** had strong performance at 0.849
- **Support Vector Machine** gives a moderate performance at 0.8
- **K-Nearest Neighbour** performed least well, at 0.746, showing it had more false positives and less effective in ID actual deaths.

7. Hyper-parameter Tuning

7.1 K-Nearest Neighbours (KNN)

```
knn_grid <- expand.grid(k = seq(3, 21, 2))
knn_model <- train(DEATH_EVENT ~ ., data = train_data, method = "knn",
                   trControl = train_control, metric = "ROC",
                   tuneGrid = knn_grid)
print(knn_model)
```

```
## k-Nearest Neighbors
##
## 210 samples
##  12 predictor
##   2 classes: 'Survived', 'Died'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 189, 189, 189, 188, 189, 190, ...
## Resampling results across tuning parameters:
##
##   k   ROC        Sens       Spec
##    3  0.7268821  0.9247619  0.3690476
##    5  0.7342971  0.9390476  0.3190476
##    7  0.7748129  0.9528571  0.2904762
##    9  0.7773469  0.9661905  0.3761905
##   11  0.7862812  0.9590476  0.3547619
##   13  0.8033617  0.9657143  0.3380952
##   15  0.8143537  0.9657143  0.3214286
##   17  0.8136111  0.9657143  0.3047619
##   19  0.8121769  0.9590476  0.3214286
##   21  0.8132370  0.9590476  0.3071429
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was k = 15.
```

7.2 Support Vector Machine (SVM)

```
svm_grid <- expand.grid(C = 2^(-2:2), sigma = 2^(-2:2))
svm_model <- train(DEATH_EVENT ~ ., data = train_data, method = "svmRadial",
                   trControl = train_control, metric = "ROC",
                   preProcess = c("center", "scale"),
                   tuneGrid = svm_grid)
print(svm_model)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 210 samples
##  12 predictor
##   2 classes: 'Survived', 'Died'
##
## Pre-processing: centered (12), scaled (12)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 189, 188, 189, 190, 189, 189, ...
## Resampling results across tuning parameters:
##
##   C     sigma  ROC        Sens       Spec
##   0.25  0.25   0.7745238  0.8980952  0.31904762
##   0.25  0.50   0.6031746  0.9523810  0.03095238
##   0.25  1.00   0.4580952  0.9800000  0.00000000
##   0.25  2.00   0.4731746  0.9733333  0.00000000
##   0.25  4.00   0.5439286  0.9933333  0.00000000
##   0.50  0.25   0.7745238  0.9042857  0.35000000
##   0.50  0.50   0.6876190  0.9457143  0.06190476
##   0.50  1.00   0.3603175  0.9866667  0.00000000
##   0.50  2.00   0.3725397  0.9866667  0.00000000
##   0.50  4.00   0.4302381  0.9861905  0.00000000
##   1.00  0.25   0.7691270  0.9180952  0.31666667
##   1.00  0.50   0.6876190  0.9447619  0.01428571
##   1.00  1.00   0.4857143  0.9866667  0.00000000
##   1.00  2.00   0.4153968  0.9733333  0.01666667
##   1.00  4.00   0.4203571  0.9861905  0.00000000
##   2.00  0.25   0.7711111  0.9190476  0.28571429
##   2.00  0.50   0.6808730  0.9519048  0.01666667
##   2.00  1.00   0.4242857  0.9866667  0.00000000
##   2.00  2.00   0.4139683  0.9795238  0.00000000
##   2.00  4.00   0.4557937  0.9928571  0.00000000
##   4.00  0.25   0.7735714  0.8976190  0.23809524
##   4.00  0.50   0.6282540  0.9590476  0.01428571
##   4.00  1.00   0.4531746  0.9866667  0.00000000
##   4.00  2.00   0.3725397  0.9661905  0.01666667
##   4.00  4.00   0.4557937  0.9861905  0.00000000
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.25 and C = 0.25.
```

7.3 Random Forest (RF)

```
rf_grid <- expand.grid(mtry = 2:6)
rf_model <- train(DEATH_EVENT ~ ., data = train_data, method = "rf",
                  trControl = train_control, metric = "ROC",
                  tuneGrid = rf_grid)
print(rf_model)
```

```
## Random Forest
##
## 210 samples
##  12 predictor
##   2 classes: 'Survived', 'Died'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 189, 189, 188, 188, 189, 189, ...
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##   2     0.8855556  0.9528571  0.5809524
##   3     0.8860317  0.9395238  0.6261905
##   4     0.8883333  0.9052381  0.6261905
##   5     0.8854762  0.9057143  0.6261905
##   6     0.8788889  0.8980952  0.5976190
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 4.
```

7.4 Decision Tree (DT)

```
dt_grid <- expand.grid(cp = seq(0.001, 0.1, by = 0.005))
dt_model <- train(DEATH_EVENT ~ ., data = train_data, method = "rpart",
                  trControl = train_control, metric = "ROC",
                  tuneGrid = dt_grid)
print(dt_model)
```

```
## CART
##
## 210 samples
##  12 predictor
##   2 classes: 'Survived', 'Died'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 189, 189, 189, 189, 189, 188, ...
## Resampling results across tuning parameters:
##
##   cp     ROC        Sens       Spec
##   0.001  0.8065079  0.8790476  0.6119048
##   0.006  0.8065079  0.8790476  0.6119048
##   0.011  0.8065079  0.8790476  0.6119048
##   0.016  0.8065079  0.8790476  0.6119048
##   0.021  0.8048413  0.8857143  0.6619048
##   0.026  0.7987302  0.8857143  0.6452381
##   0.031  0.7857143  0.8923810  0.6285714
##   0.036  0.7593651  0.8990476  0.6285714
##   0.041  0.7593651  0.8990476  0.6285714
##   0.046  0.7593651  0.9057143  0.6285714
##   0.051  0.7593651  0.9057143  0.6285714
##   0.056  0.7638095  0.9323810  0.5952381
##   0.061  0.7638095  0.9323810  0.5952381
##   0.066  0.7638095  0.9323810  0.5952381
##   0.071  0.7638095  0.9323810  0.5952381
##   0.076  0.7638095  0.9323810  0.5952381
##   0.081  0.7638095  0.9323810  0.5952381
##   0.086  0.7638095  0.9323810  0.5952381
##   0.091  0.7638095  0.9323810  0.5952381
##   0.096  0.7638095  0.9323810  0.5952381
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.016.
```

For each model trained, we performed grid search and 10-fold cross validation to optimize `k`, `c` / `sigma`, `mtry`, and `cp` for their respective models. ROC was used to select optimal model using largest value.

8. Principal Component Analysis (PCA)

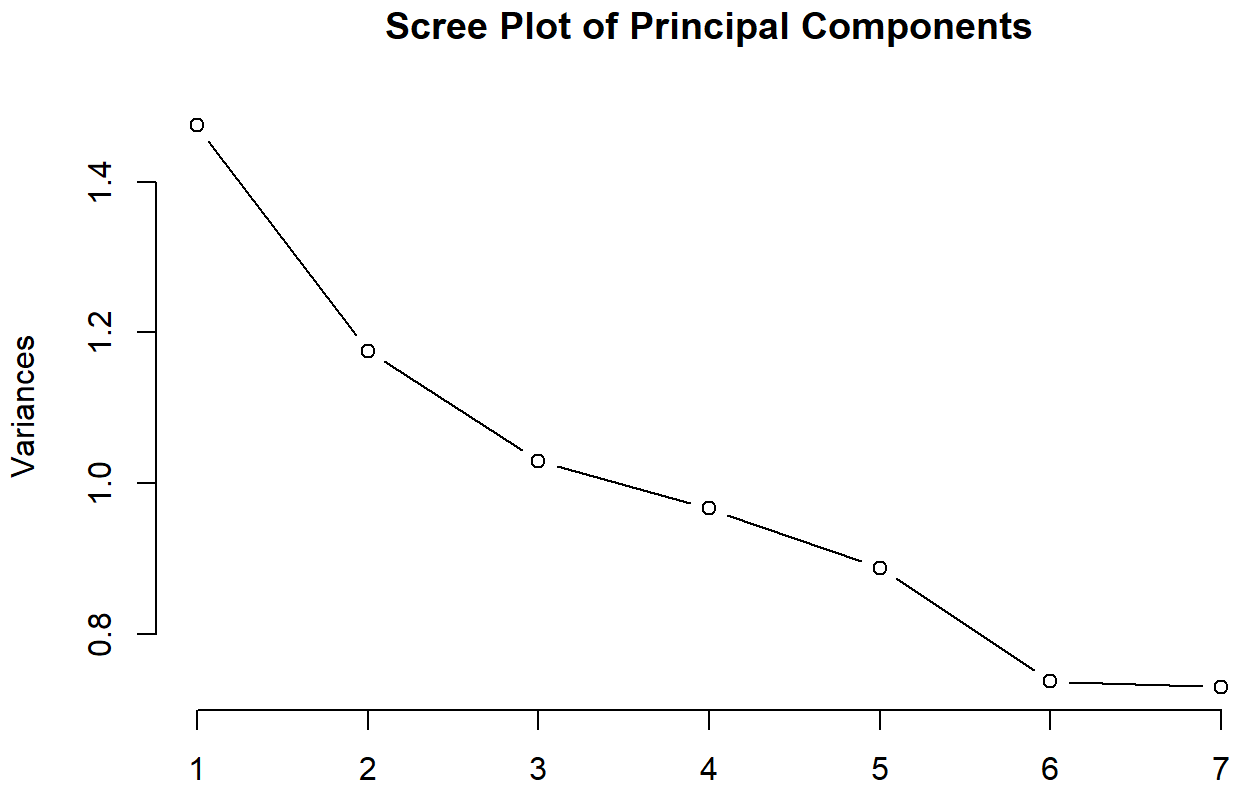
We applied PCA to reduce dimensionality so we can plot and interpret better, as PCA will give us a low-dimensional representation of the data that captures as much variance as possible.

```
# extract only scaled continuous predictors
pca_data <- hfp_scaled[, continuous_vars]
pca_result <- prcomp(pca_data, center = FALSE, scale. = FALSE)
```

8.1 Scree Plot

This plot shows how much variance is explained by each principal component.

```
# scree plot -> proportion of variance explained
screeplot(pca_result, type = "lines", main = "Scree Plot of Principal Components")
```

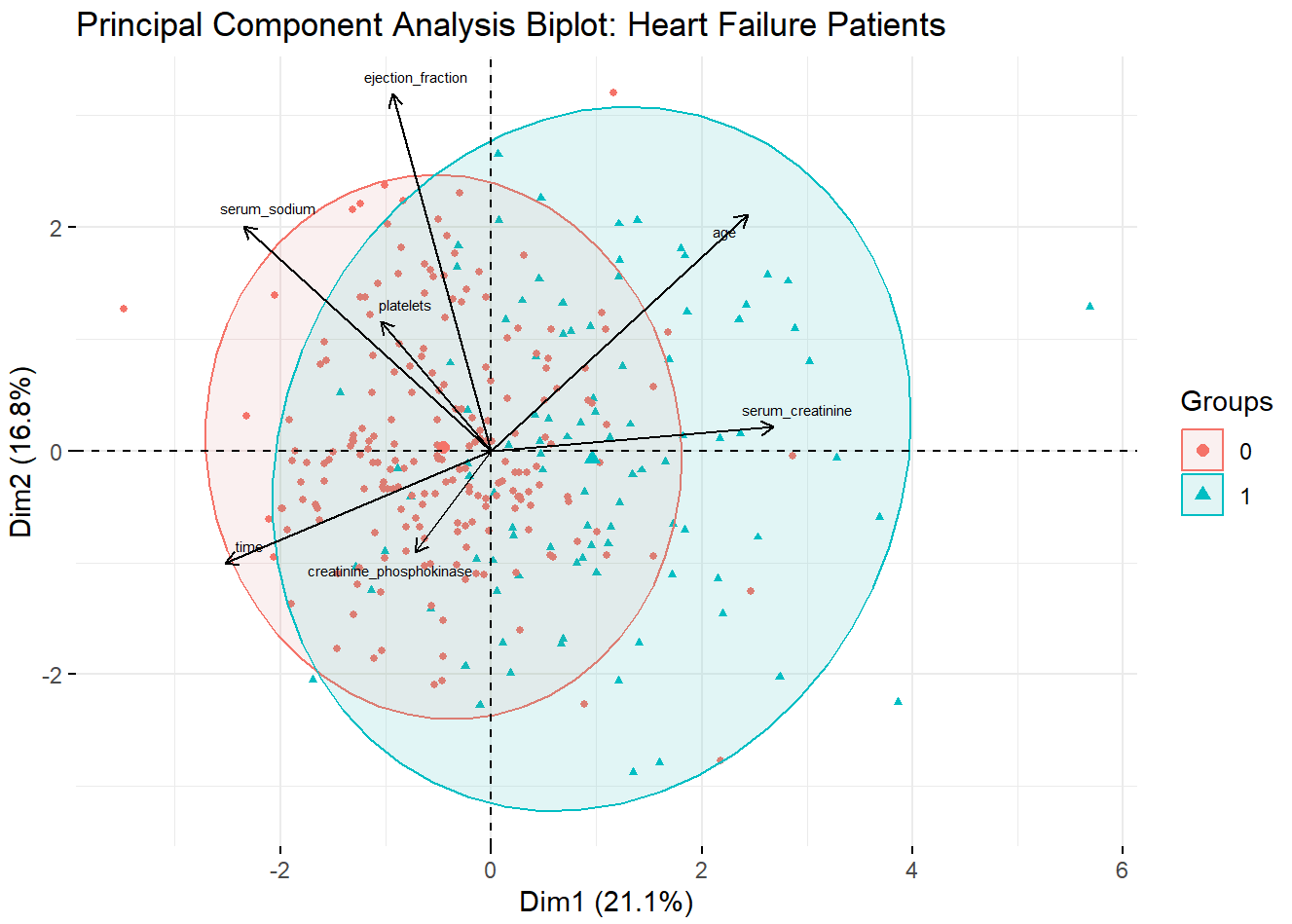


scree plot: shows amt of variance explained by each PC, each point represents 1 PC and y-axis shows how much variance PC explains. We used it to determine how many PCs show meaningful dimensionality reduction - PC1 and PC2 explain most variance.

8.2 PCA Biplot

This plot shows patient scores on the first two PCs and the loadings of each variable.

```
# biplot
fviz_pca_biplot(pca_result,
                label = "var",
                habillage = hfp_scaled$DEATH_EVENT,
                addEllipses = TRUE,
                col.var = "black",
                col.ind = "gray",
                pointsize = 1,
                repel = TRUE,
                labelsize = 2,
                title = "Principal Component Analysis Biplot: Heart Failure Patients")
```



This biplot visualizes both PCs (Dim1, Dim2) for each patient, and variable loadings. A positive loading shows that the variable increases the PC score, and a negative loading shows that the variable decreases the PC score. Each arrow represents a variable and coneys:

- direction: how variable contributes to PCs
- length: how strongly a variable influences PCA (longer=stronger)
- angle between arrows:
    - same direction = **positive correlation** between variables
    - opposite direction = **negative correlation** between variables
    - perpendicular = uncorrelated

This helps us visually see the clustering between survived (0) and died (1).

We can see that variables like `serum_creatinine` and `age` arrows are **long and point right**, therefore they strongly contribute to PC1.

`ejection_fraction` and `serum_sodium` point towards the **upper-left** quadrant of the plot, and it has moderate positive loading on PC2, but negative on PC1.

`time` points **bottom-left,** meaning it loads negatively on PC1 and PC2.

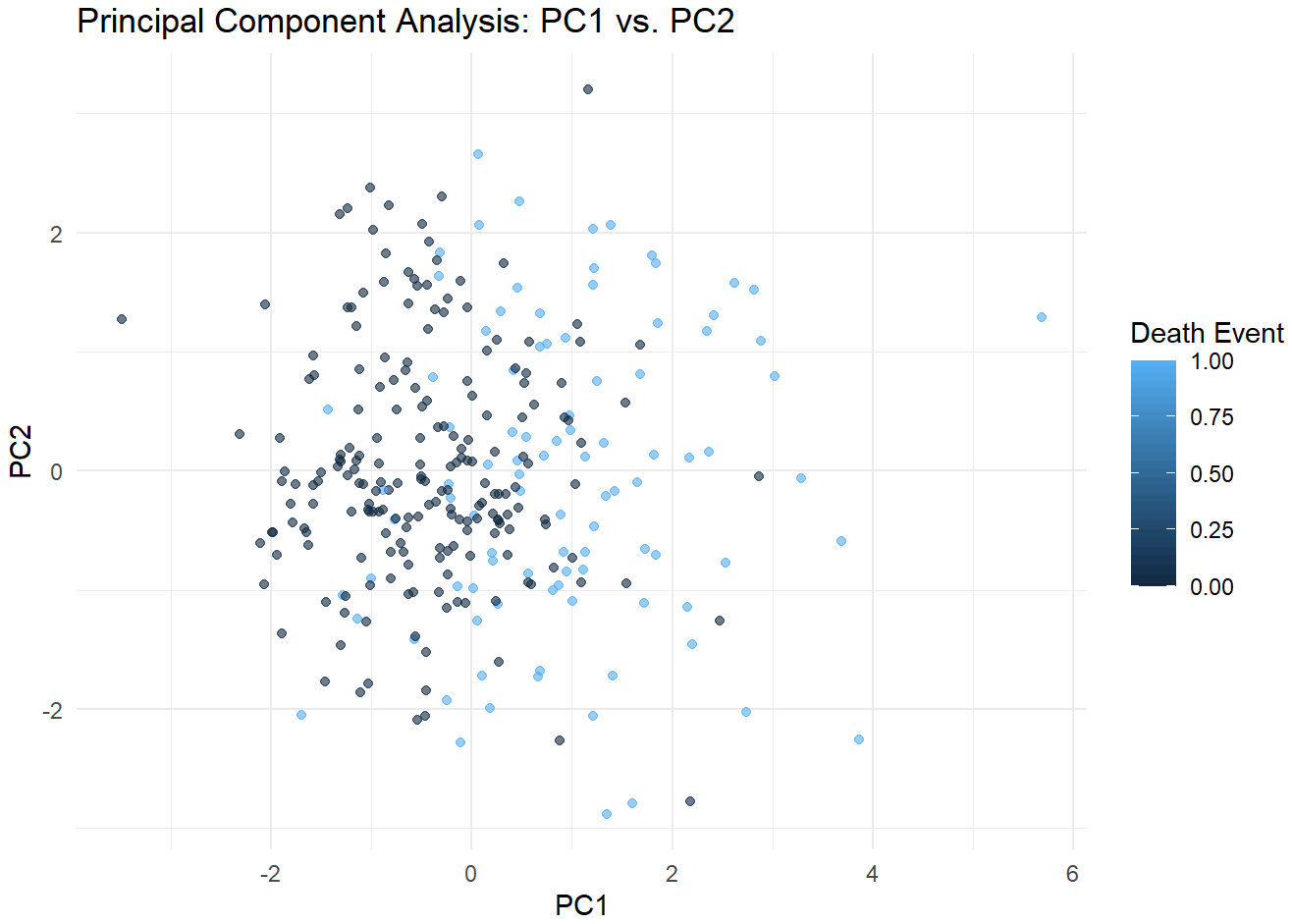
`creatinine_phosphokinase` points **downward**, and it contributes mostly negatively to PC2

`platlets` points **left**, indicating a negative PC1 loading

8.3 PC1 vs. PC2 Scatter-plot

This plot visualizes patients in the PCA space, colored by survival outcome.

```
# PCA dataframe
pca_df <- as.data.frame(pca_result$x)
pca_df$DEATH_EVENT <- hfp_scaled$DEATH_EVENT

# PC1 vs. PC2
ggplot(pca_df, aes(x = PC1, y = PC2, color = DEATH_EVENT)) +
  geom_point(alpha = 0.6) +
  labs(title = "Principal Component Analysis: PC1 vs. PC2", x = "PC1", y = "PC2", color = "Death
Event") +
  theme_minimal()
```



The scatter-plot explores how individual points lie in PCA space. It shows the spatial grouping of `DEATH_EVENT` classes, but it is not perfectly separable. It also shows that PCA doesn't fully separate classes but shows patterns instead.

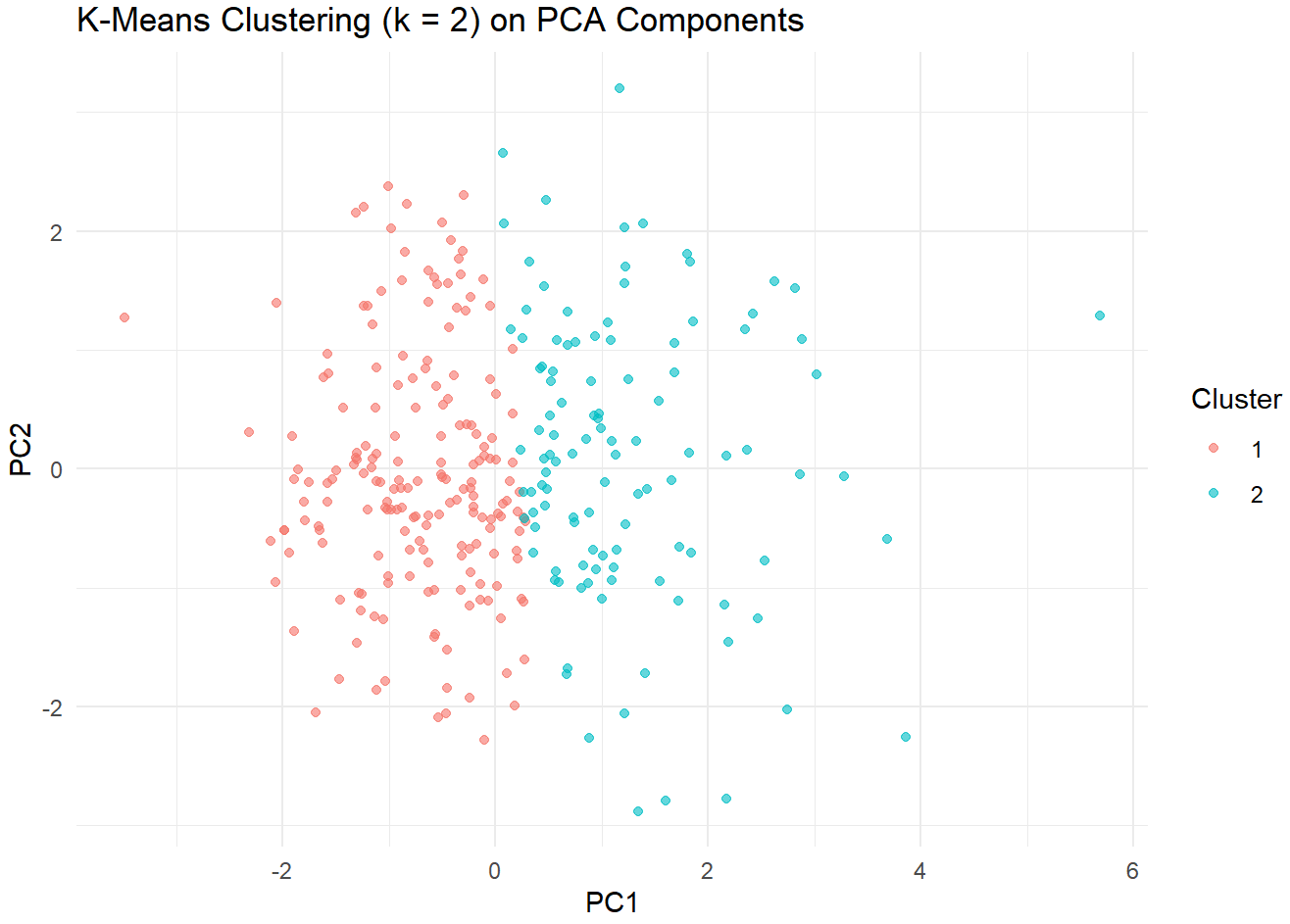
9. K-Means Clustering

We applied K-Means (k=2) to identify groups in PCA reduced space, this will explore **unsupervised** grouping.

```r
set.seed(2025)

# top 2 PCs for visualization, full data for clustering
kmeans_result <- kmeans(pca_data, centers = 2, nstart = 25)

# cluster labels to PCA df
pca_df$Cluster <- as.factor(kmeans_result$cluster)

# confusion matrix to check alignment with DEATH_EVENT
table(Cluster = pca_df$Cluster, DEATH_EVENT = pca_df$DEATH_EVENT)
```

```
##         DEATH_EVENT
## Cluster   0   1
##       1 162  26
##       2  41  70
```

```r
# plot clusters
ggplot(pca_df, aes(x = PC1, y = PC2, color = Cluster)) +
  geom_point(alpha = 0.6) +
  labs(title = "K-Means Clustering (k = 2) on PCA Components") +
  theme_minimal()
```



We used k-means (k=2) on first 2 PC to see the unsupervised grouping of patients. Compared to the other scatter plot above which used `DEATH_EVENT`, K-Means formed the groups based only on patterns in the data and no knowledge of survival outcomes. The plot shows two distinct clusters, which indicates the dataset has a natural structure, and the patients are grouped based on similar features (ex. `age`, `serum_creatinine`, etc) which had a strong impact on patterns found by PCA (plots above).