Final Progect - Heart Failure

Itai Alcalai 206071110 & Netanel Landesman 315873588 & Gal Levy 208540872

2023-08-07

Data Overview and Abstract:

This document presents a comprehensive analysis of the dataset of 1000 different cases of heart failure received from the NHS (National Health Service). This exploration intends to provide a better understanding of the underlying risk factors of Heart Failure that can offer valuable insights. The R code provided in this document employs a series of packages and libraries designed for data analysis and data manipulation. The primary goal of this document is to convey a detailed understanding of the process, from the raw data to the final results and interpretations.

Step 1: Initiation

In this initial step, we load libraries essential for data manipulation and analysis:

```
# Load the libraries:
library(ggplot2)
library(dplyr)
library(tidyr)
library(corrplot)
library(readr)
library(survival) # for kaplan meier
library(survminer)
library(caret)
library(tidymodels)
library(class) # for KNN
library(rpart) # for Decision Tree
library(randomForest) # for Random Forest
library(e1071) # for SVM
```

Step 2: Load and cleaning the data

After we have downloaded the data from the database, we will load it and perform cleaning operations:

```
# Loading the data:
data <- read.csv("data1.csv")
# Checking whether there are duplicates in the data:
dim(data)[1]
```

```
## [1] 1000
```

```
dim(unique(data))[1]
```

```
## [1] 1000
```

You can see that there are no duplicates in the data, since the number of lines is the same both in the normal version and in the unique version of the data.

```
# Checking if there is Null or NA in the data and if so we will remove it:
any(is.null(data))
```

```
## [1] FALSE
```

```
any(is.na(data))
```

```
## [1] TRUE
```

```
data <- na.omit(data)

# Checking if there are negative values:
sum(rowMeans(data < 0), na.rm = TRUE)
```

```
## [1] 0
```

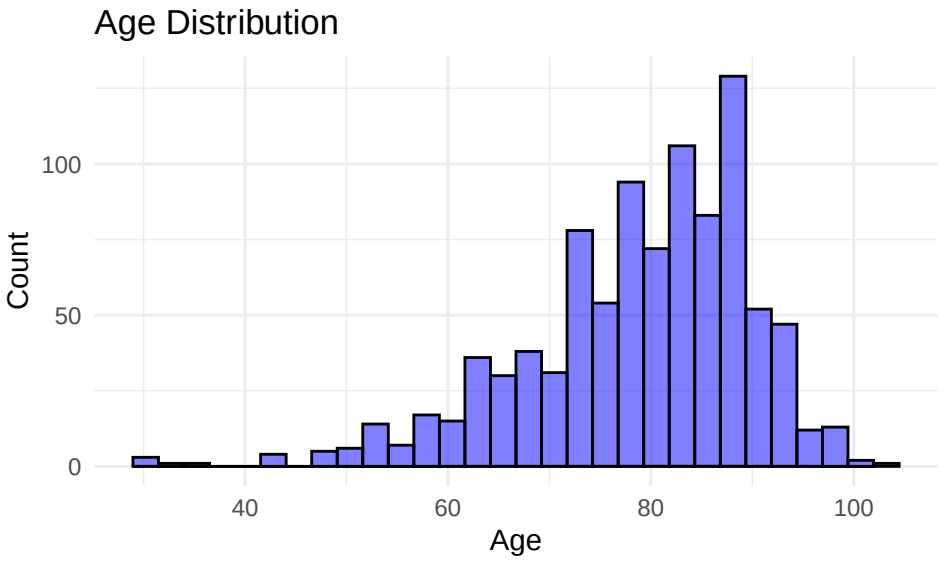
As you can see, we do not have null values or negative values in the data, but we do have NA values so we removed them.

EDA:

Step 3.1: Presentation of an age distribution graph

In this code we are showing the age distribution among the population. We can see that the distribution is more or less normal and the ages 70-90 are the most frequent.

```
# Show the distribution of age
ggplot(data, aes(x = age)) +
  geom_histogram(bins = 30, fill = "blue", alpha = 0.5, color = "black") +
  geom_density(alpha = 0.2, fill = "#FF6666") + # This is for the KDE
  theme_minimal() + labs(title = "Age Distribution", x = "Age", y = "Count")
```

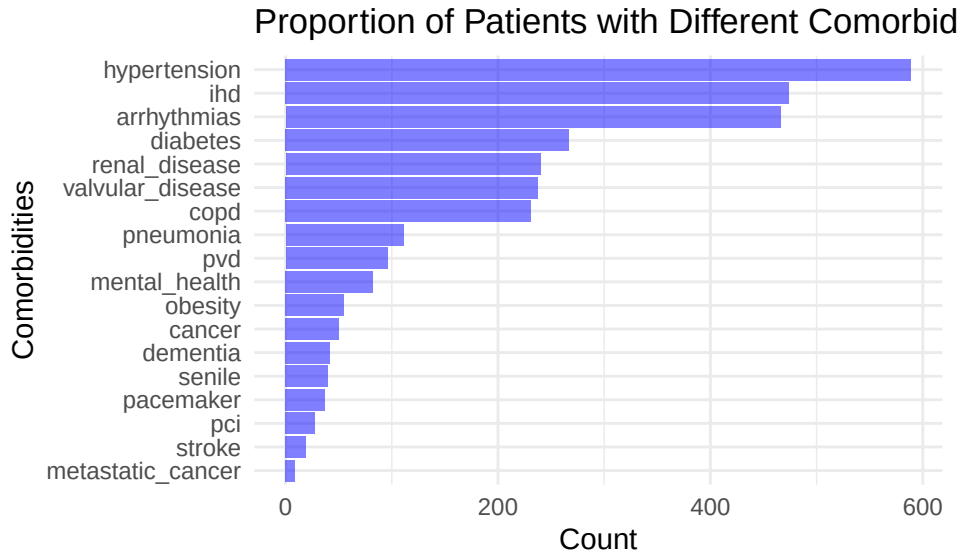


Step 3.2: Fetch Comorbidities graph

In this graph we can see on the y-axis the various comorbidities that the patients came with at the time of the event, and on the x-axis the number of patients with those comorbidities It can be seen that hypertension, arrhythmia and ihd are among the common comorbidities among the population. This does not surprise us since these are major risk factors for heart failure.

```
# Define the list of comorbidities
comorbidities <- c('cancer', 'dementia', 'diabetes', 'hypertension', 'ihd',
                   'mental_health', 'arrhythmias', 'copd', 'obesity', 'pvd',
                   'renal_disease', 'valvular_disease', 'metastatic_cancer',
                   'pacemaker', 'pneumonia', 'pci', 'stroke', 'senile')
# Summarize the data
data_sum <- data %>% select(comorbidities) %>%
  summarise_all(sum) %>%gather(key = "comorbidity", value = "count")
```

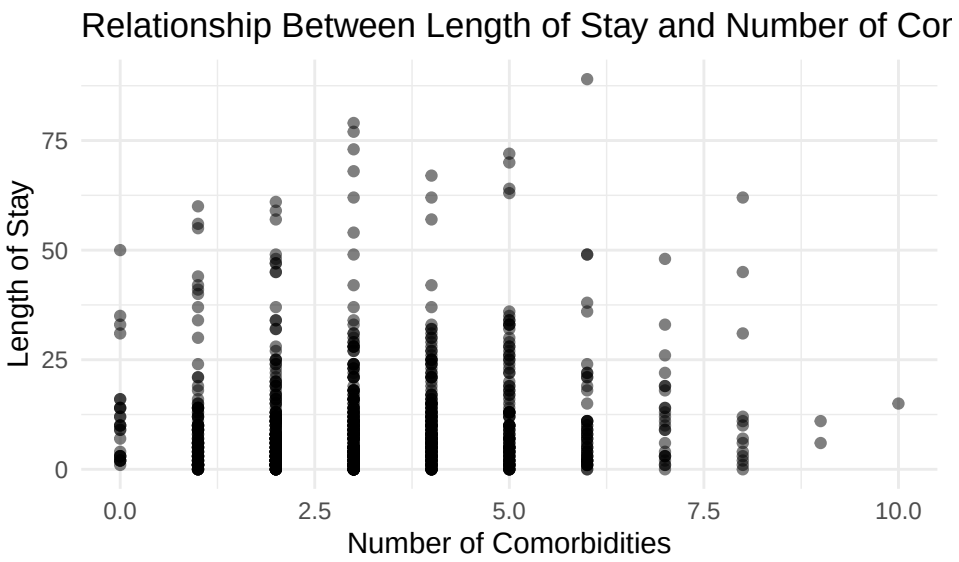
```
# Order data by count
data_sum <- data_sum %>% arrange(count)
# Create the bar plot
ggplot(data_sum, aes(x = count, y = reorder(comorbidity, count))) +
  geom_bar(stat = "identity", fill = "blue", alpha = 0.5) +
  labs(title = "Proportion of Patients with Different Comorbidities", x = "Count", y = "Comorbidities")
  theme_minimal()
```



Step 3.3: Relationship Between Length of Stay and Number of Comorbidities

After we saw the distribution of the comorbidities among the patients, in this graph we attribute the length of stay in the hospital to the comorbidities. It can be seen that the lower the number of comorbidities a person has, the shorter he will stay in the hospital. Also, if a person has many comorbidities, he will also stay in the hospital for a shorter period of time. This can be attributed to the fact that apparently if a person has few comorbidities then he does not need a long hospitalization and alternatively if a person has a lot of comorbidities then he probably died and therefore no longer stays in the hospital. The patients who stay for a long time in the hospital are those who have on the one hand a significant number of comorbidities but on the other hand the doctors manage to help them

```r
# Calculate the sum of comorbidities for each patient
data$comorbidity_count <- rowSums(data[comorbidities])

# Create the scatter plot
ggplot(data, aes(x = comorbidity_count, y = los)) +
  geom_point(alpha = 0.5) +
  labs(title = "Relationship Between Length of Stay and Number of Comorbidities",
       x = "Number of Comorbidities", y = "Length of Stay") + theme_minimal()
```



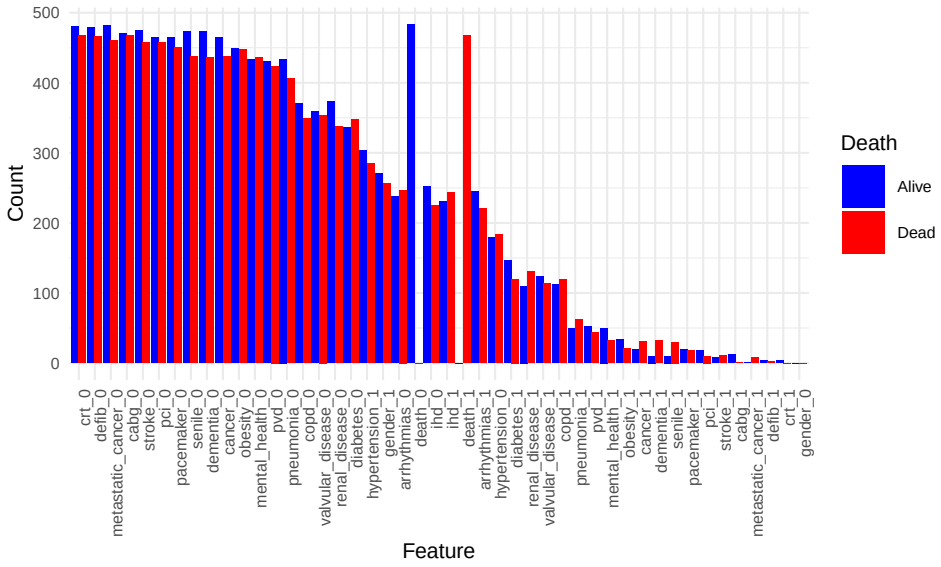
Step 3.4: Comorbidities with relation to death

This R script analyses the dataset, particularly its boolean features, to calculate the counts of 'Alive' and 'Dead' states for each. It creates a dataframe for each state, storing the feature, its count, and the death status. These dataframes are aggregated into a master dataframe, which is then visualized as a dodged bar plot. The bars represent counts of 'Alive' and 'Dead' for each boolean feature, colored differently for distinct visibility.

In fact, in this graph we are looking for Boolean features that strongly influence the aliveness or death of a patient - we will expect to see for such Boolean feature a significant change between their "alive" value (blue) and the "death" value (red)

```r
dfs <- list()
# Finding all boolean features:
boolean_features <- names(data)[sapply(data, function(col) length(unique(col))) == 2]
# For each boolean feature
for (feature in boolean_features) {
  # For each value of the feature (0, 1), generate counts for each survival status
  for (val in c(0, 1)) {
    alive <- sum((data$death == 0) & (data[[feature]] == val))
    dead <- sum((data$death == 1) & (data[[feature]] == val))

    # Convert to data frame for easier plotting
    df_alive <- data.frame(Feature = paste0(feature, "_", val), Count = alive, Death = "Alive")
    df_dead <- data.frame(Feature = paste0(feature, "_", val), Count = dead, Death = "Dead")

    # Append to the list
```

```
    dfs[[paste0(feature, "_", val, "_Alive")]] <- df_alive
    dfs[[paste0(feature, "_", val, "_Dead")]] <- df_dead
  }
}

# Combine all data frames in the list
df <- do.call(rbind, dfs)

# Create the bar plot
ggplot(df, aes(x = reorder(Feature, -Count), y = Count, fill = Death)) +
  geom_bar(stat = "identity", position = "dodge") +
  scale_fill_manual(values = c("blue", "red")) +
  theme_minimal(base_size = 8) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(x = "Feature", y = "Count")
```

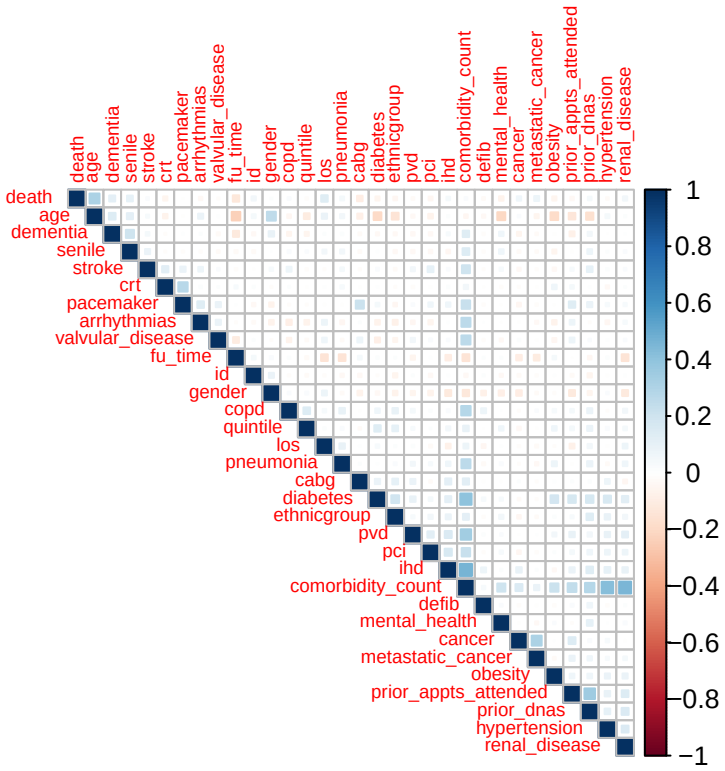


As you can see there is no Boolean feature that directly affects survival or death since all the features (except the death feature of course) received equal values in the quantities between their "death" value and their "alive" value

Step 3.5: Correlation Plot

After we tried to check if there is a correlation between a certain concomitant disease for survival or death and we found none. We wanted to check if there is a connection between different features in the data.

```
# Remove columns where the standard deviation is 0
data <- data[, sapply(data, sd, na.rm = TRUE) != 0]

# Calculate correlation matrix
corr <- cor(data, use = "complete.obs")

# Assuming MS is your data frame or matrix

corrplot(corr, method ='square',type = "upper", order = "hclust",
 tl.cex = 0.6)
```

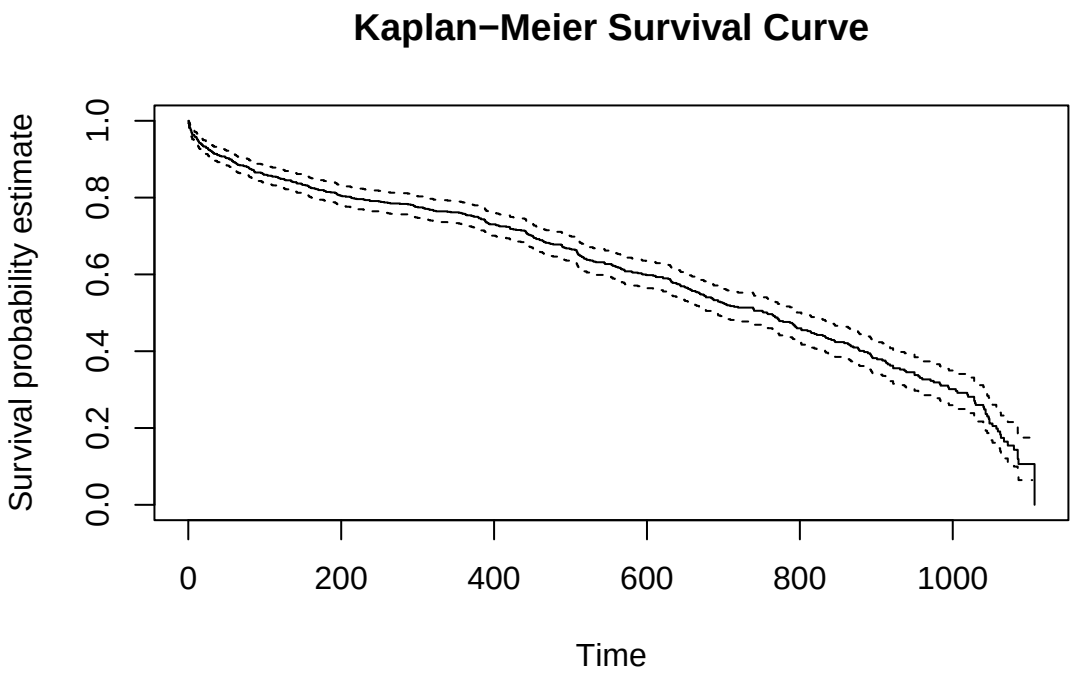


In the correlation output you can see that there is no real correlation between any of the features This can probably be attributed to the fact that there are a large number of features and therefore no features really have a direct correlation to one another. However, it is the totality of the features that actually gives a certain characterization regarding the patient's condition and not a combination of individual features, and we would like to investigate this fact further by ML analyses.

Step 4: Survival Analysis - Kaplan Meier

The Kaplan-Meier method is used to estimate the survival function from lifetime data. The Surv object is created with 'fu_time' (follow-up time) and 'death' (event status). The survfit function then fits a survival curve to this data. The resulting Kaplan-Meier survival curve, which graphically represents the estimated survival function over time, is then plotted with confidence intervals. The X-axis represents time, and the Y-axis shows the estimated survival probability at each point in time.

```
# Create a Surv object with 'fu_time' and 'death'
surv_obj <- Surv(data$fu_time, data$death)

# Fit the survival curve
fit <- survfit(surv_obj ~ 1)  # "~ 1" implies no covariates, a simple Kaplan-Meier fit

# Plot the survival curve
plot(fit, xlab = "Time", ylab = "Survival probability estimate",
     main = "Kaplan-Meier Survival Curve", conf.int=TRUE)
```



As we can see the follow up time corelates linearly to the survival of an individual. Next we will try to reduce the data and see if we can build a prediction model.

Step 5.0: Initiation of Dimnsional Reduction libraries

In this initial step, we load libraries essential for data manipulation and analysis: The mice library ensure complete data for analysis, scales aids in data visualization, readr makes it easy to import rectangular data, improving efficiency. Rtsne and umap are used for dimensionality reduction, making it possible to visualize high-dimensional data in two or three dimensions, thereby aiding data exploration and interpretation.

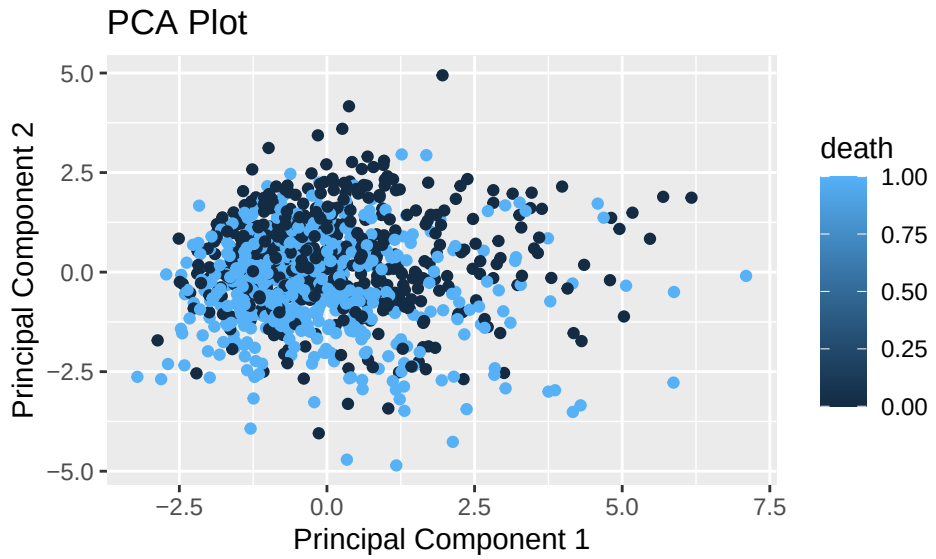
```
library(mice)
library(scales)
library(Rtsne)
library(umap)
```

Step 5.1: Performing PCA

This code reads in a dataset, removes or replaces missing or infinite values, and retains a variable of interest, 'death'. The data is then normalized, and Principal Component Analysis (PCA) is conducted to reduce the dimensionality, capturing the most variance in the first two components. These components are then plotted in a scatterplot, with points colored according to the 'death' variable. The purpose of the code is to simplify the dataset and visualize it in a way that can easily highlight patterns related to 'death'. We can see the based on the best PC1 and PC2 we didn't get a good result and the data can't be represented in 2D. (Same for other PC's we tried).

```
# remove rows with Inf or -Inf values or replace Inf with a specific value
data <- data[!apply(data == Inf | data == -Inf, 1, any),]

# save the 'death' column before scaling and PCA
death <- data$death
```

```r
# remove 'death' column from data before scaling and PCA
data$death <- NULL

# normalize the data
data <- scale(data)

# perform PCA
pca_result <- prcomp(data)

# extract the first two principal components
pca_data <- as.data.frame(pca_result$x[,1:2])

# add 'death' back into the pca_data
pca_data$death <- death

# create a scatter plot of the first two principal components, colored by 'death'
ggplot(data = pca_data, aes(x = PC1, y = PC2, color = death)) +
  geom_point() + xlab("Principal Component 1") + ylab("Principal Component 2") +
  ggtitle("PCA Plot") + scale_color_continuous(name = "death")
```



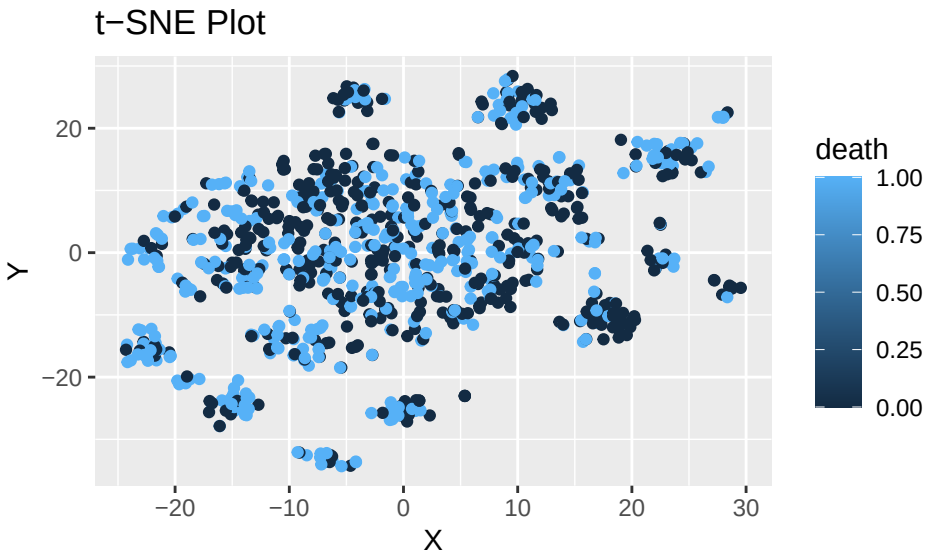
As you can see the data is not linearly separated!

Step 5.2: Performing t-SNE

This script applies t-distributed Stochastic Neighbor Embedding (t-SNE), a technique for high-dimensional data visualization. The t-SNE process is then applied, reducing the data's dimensions while preserving the local structure of the data. The resulting t-SNE values, along with the 'death' data, are plotted on a scatterplot. The main purpose is to visually represent complex data, highlighting potential patterns related to 'death'. Like the PCA, t-SNE results didn't give us meaningful insights about the data distribution in 2D.

```r
# perform t-SNE
set.seed(42)  # for reproducibility
tsne_result <- Rtsne(as.matrix(data), dims = 2, perplexity=30, check_duplicates = FALSE)
```

```r
# create a data frame for plotting
tsne_data <- data.frame(tsne_result$Y)
names(tsne_data) <- c("X", "Y")

# add 'death' back into the tsne_data
tsne_data$death <- death

# create a scatter plot of the t-SNE results, colored by 'death'
ggplot(tsne_data, aes(x = X, y = Y, color = death)) + geom_point() +
  scale_color_continuous(name = "death") + ggtitle("t-SNE Plot")
```



All in all, for the reason that our data consists of many Boolean features and it contains many of them, then it can be said that the data has a high dimension (about 30) and therefore as soon as we reduce the dimensions using the various algorithms we actually see that the data cannot be separated in 2D and therefore the results that do not yield insights were obtained. If the data consisted mostly of continuous features we could get better results

ML algorithms

Our goal is to find a model that well classifies a patient with a heart failure event - will he die or live?

Since, as we have shown, our data is not linearly separated, so we will look for a model that supports such a data type.

Step 6.1: Creating the train and test sets

To ensures that all the features in the data have the same scale we normalized the data:

```r
# min/max normalization function:
norn_fonc <- function(x) { return ((x - min(x)) / (max(x) - min(x))) }
norm_data <- as.data.frame(lapply(data, norn_fonc))
```

For all four algorithms, we chose to divide the data into an 80% training set on which the model will practice and recognize the different classifications. and 20% a test set on which we will test the trained model

We will eventually choose the model that received the highest percentage of accuracy for the test set

```r
# shuffle the data
set.seed(123)
shuffled_data <- norm_data[sample(nrow(norm_data)), ]
split_index <- createDataPartition(shuffled_data$death, p=0.8, list=FALSE)

# split into train/test
train_data <- shuffled_data[split_index, ]
test_data <- shuffled_data[-split_index, ]

X_train <- train_data %>% select(-death)
y_train <- train_data$death

X_test <- test_data %>% select(-death)
y_test <- test_data$death
```

Creation of a data frame that will contain for each model its name and its percentage of accuracy on the test set:

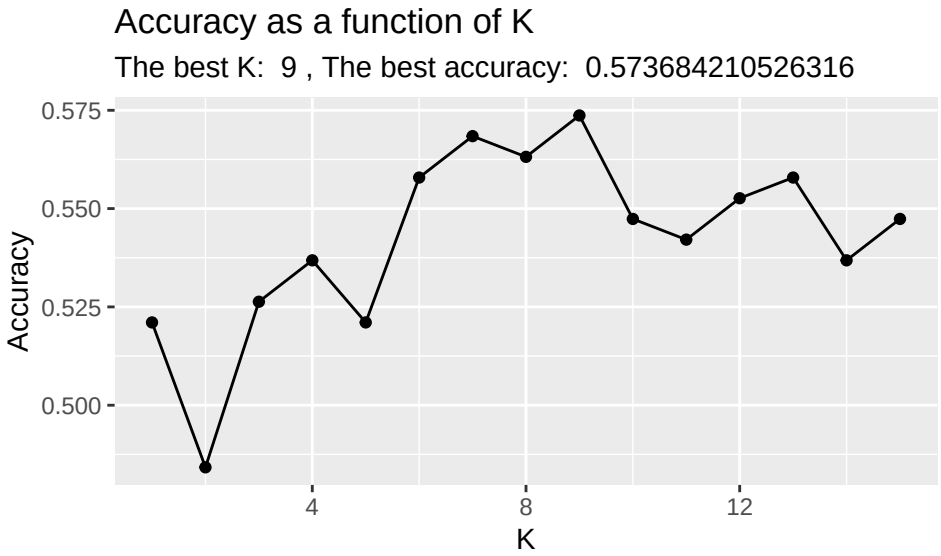
```r
results <- data.frame(model_name = character(), Accuracy = numeric())
```

Step 6.2: KNN

The performance of KNN varies significantly depending on the number of neighbors (K) therefore we ran the algorithm with a different number of neighbors in order to see with which K we will get the best performance on the test set

```r
# Initialize an empty data frame to store KNN results
KNN_df <- data.frame(K = numeric(), Accuracy = numeric())
best_acc <- 0
best_k <- 0

# Loop through different values of K (from 1 to 15)
for (k in 1:15) {
  predicted_labels <- knn(train = X_train, test = X_test, cl = y_train, k = k)
  result_df <- data.frame(Predicted = predicted_labels, Observed = y_test)

  # Calculate accuracy
  correct_predictions <- sum(result_df$Predicted == result_df$Observed)
  total_instances <- nrow(result_df)
  accuracy <- correct_predictions / total_instances

  # Add the results to the KNN results data frame
  KNN_df <- rbind(KNN_df, data.frame(K = k, Accuracy = accuracy))

  # Update the best K and accuracy if the current accuracy is higher
  if (accuracy > best_acc) {
    best_acc <- accuracy
    best_k <- k
  }
}
```

```r
ggplot(KNN_df, aes(K, Accuracy)) + geom_line() + geom_point() +
labs(x = "K", y = "Accuracy", subtitle = paste("The best K: ", as.character(best_k),
", The best accuracy: ", as.character(best_acc))) + ggtitle("Accuracy as a function of K")
```




```
# Finally, we will add the name of the model and its accuracy on the test set:
results <- rbind(results, data.frame(model_name = "KNN", Accuracy = best_acc))
```

As you can see the performance of the algorithm on our data is not good, and for K=9 we will get the best performance. We will save this data in the results df and continue to the next model

Step 6.3: Decision Tree

Decision Trees are a type of flowchart-like structure in which each internal node represents a feature(or attribute), each branch represents a decision rule, and each leaf node represents an outcome. They are a popular machine learning algorithm because they are interpretable, meaning one can see the logical path the algorithm takes to arrive at a decision. Let's dive into our data and see how well a Decision Tree model performs:

```
# Initialize the Decision Tree classifier
dtree <- rpart(y_train ~ ., data = data.frame(cbind(y_train, X_train)), method = "class")

# Predict the classes for the train set
y_train_pred <- predict(dtree, newdata = data.frame(cbind(y_train, X_train)), type = "class")

train_accuracy <- sum(y_train_pred == y_train) / length(y_train)
print(paste('Train Accuracy:', train_accuracy))
```

```
## [1] "Train Accuracy: 0.654247812588202"
```

```
# Predict the classes for the test set
y_pred <- predict(dtree, newdata = data.frame(cbind(y_test, X_test)), type = "class")

test_accuracy <- sum(y_pred == y_test) / length(y_test)
print(paste('Test Accuracy:', test_accuracy))
```

```
## [1] "Test Accuracy: 0.657508468197215"
```

```r
# Finally, we will add the name of the model and its accuracy on the test set:
results <- rbind(results, data.frame(model_name = "Decision Tree", Accuracy = test_accuracy))
```

From the results, our Decision Tree model achieved a training accuracy of approximately 65.42% and a test accuracy of approximately 65.75%. This indicates that our model performs similarly on both the training and test data, which is a good sign as it suggests the model is not overfitting.

Step 6.4: Random Forest

Random Forest is an ensemble learning method that operates by constructing multiple decision trees at training time and outputting the class that is the mode of the classes for classification tasks. It can correct for decision trees' habit of overfitting to their training set. Random Forest offers a boost in accuracy over single decision trees due to the way it aggregates the results of many trees.

```r
# Create a Random Forest model specification with 1000 trees
rf <- rand_forest(trees = 1000) %>% set_engine("ranger") %>% set_mode("classification")

# Train the Random Forest model on the training data
rf_training <- rf %>% fit(as.factor(death) ~ ., data = train_data)

# Make predictions on the training data and bind prediction results
train_pred <- predict(rf_training, train_data) %>%
  bind_cols(predict(rf_training, train_data, type = "prob")) %>%
  bind_cols(train_data %>% select(death))

# Convert 'death' column to a factor
train_pred$death <- as.factor(train_pred$death)

# Calculate accuracy for training set predictions
rf_accuracy <- train_pred %>%
  accuracy(truth = death, .pred_class)

print(paste('Train Accuracy:', rf_accuracy[3]))
```

```
## [1] "Train Accuracy: 0.980242732147897"
```

```r
# Make predictions on the test data and bind prediction results
test_pred <- predict(rf_training, test_data) %>%
  bind_cols(predict(rf_training, test_data, type = "prob")) %>%
  bind_cols(test_data %>% select(death))

# Convert 'death' column to a factor
test_pred$death <- as.factor(test_pred$death)

# Calculate accuracy for test set predictions
rf_accuracy <- test_pred %>%
  accuracy(truth = death, .pred_class)

print(paste('Testing Accuracy:', rf_accuracy[3]))
```

```
## [1] "Testing Accuracy: 0.710575837410613"
```

```r
# Finally, we will add the name of the model and its accuracy on the test set:
results <- rbind(results, data.frame(model_name = "RF", Accuracy = as.numeric(rf_accuracy[3])))
```

From our analysis, the Random Forest model achieves an impressive training accuracy of approximately 97.96%. However, there is a noticeable difference between the training and testing accuracies, with the testing accuracy being around 70.61%. This suggests that while the model is learning the training data exceptionally well, it might be overfitting, which results in reduced performance on unseen data.

Step 6.5: SVM

Finally, we applied an SVM, Support Vector Machines (SVM) are supervised machine learning algorithms which can be used for both classification or regression challenges. They perform classification by finding the hyperplane that best divides a dataset into classes. SVMs are known for their ability to handle high dimensional data and their flexibility in modeling both linear and non-linear relationships.

```r
set.seed(42)

# Initialize the SVM classifier
svm <- svm(formula = y_train ~ ., data = X_train, type = 'C-classification', kernel = 'radial')

# Predict the classes for the train set
y_train_pred <- predict(svm, train_data)

train_accuracy <- sum(y_train_pred == y_train) / length(y_train)

cat("Train Accuracy:", train_accuracy, "\n")
```

```
## Train Accuracy: 0.7216107
```

```r
# Predict the classes for the test set
y_pred <- predict(svm, test_data)

test_accuracy <- sum(y_pred == y_test) / length(y_test)

cat("Test Accuracy:", test_accuracy, "\n")
```

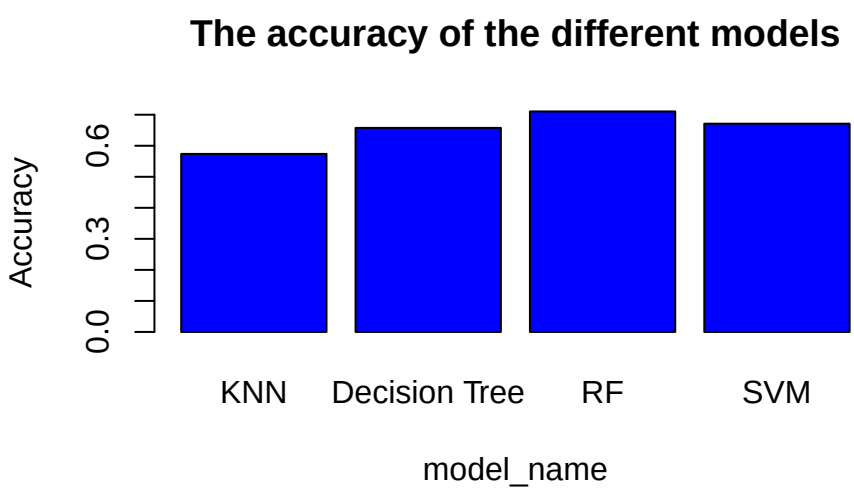
```
## Test Accuracy: 0.6710576
```

```r
# Finally, we will add the name of the model and its accuracy on the test set:
results <- rbind(results, data.frame(model_name = "SVM", Accuracy = test_accuracy))
```

The SVM model yields a training accuracy of approximately 72.16% and a testing accuracy of about 67.11%. The small discrepancy between training and testing accuracies suggests that the model generalizes reasonably well to unseen data.

Comparing the algorithms:

```r
# Create a bar plot:
barplot(results$Accuracy, names.arg = results$model_name, xlab = "model_name", ylab = "Accuracy",
        main = "The accuracy of the different models", col = "blue")
```



Our analysis incorporated four prominent machine learning algorithms to evaluate their performance on the given dataset: Decision Trees, Random Forest , Support Vector Machines (SVM), and K-Nearest Neighbors (KNN). From our evaluations, Random Forest achieved the highest training accuracy, but it also demonstrated a notable disparity between training and test accuracies, indicating potential overfitting. SVM, on the other hand, had a smaller discrepancy between training and test accuracies, suggesting a decent generalization to unseen data. Decision Trees exhibited similar performance on both training and testing sets, highlighting its stability. However, KNN's performance was relatively lower, even at its optimal number of neighbors (K=9). Upon comparing the models' accuracies, it's evident that not all algorithms perform equally well on this dataset. This could be attributed to various factors including the nature of the data, the hyperparameters used, or the inherent strengths and weaknesses of each algorithm. Improvments can be made in the future that might involve further hyperparameter tuning, feature engineering, or exploring other machine learning algorithms to achieve better results.

Step 7: Conclusions

Using a dataset from the NHS on heart failure, our aim was to dive deep and understand the factors linked to this condition. Here's process through the data: Data Setup: We began by loading and tidying up our dataset. This step ensured we had a strong foundation for the tasks ahead. EDA: Several visuals were crafted: An age graph indicated most patients were aged 70-90. Another chart highlighted prevalent health conditions accompanying heart failure, such as hypertension and diabetes. Interestingly, hospital stay durations were linked to the number of health conditions a patient had. Despite exploring connections between various health issues, none emerged as significantly interlinked. Survival Analysis: The Kaplan-Meier approach gave us insights into how follow-up time might influence survival rates. Streamlining Our Data: We experimented with PCA and t-SNE in hopes of simplifying our dataset. But given the nature of our data (a lot of yes/no type answers), these techniques didn't yield significant improvements. Predictive Modeling: application of four prediction techniques. Each brought unique insights, with their own sets of advantages and limitations. Random Forest emerged as a particularly effective algorithm for our dataset, and the reasons can be linked to the nature of heart failure and the specific type of data we're dealing with. As mentioned, our dataset primarily consists of boolean health history attributes, which means we're often splitting data based on 'yes' or 'no' criteria. Decision trees and, by extension, Random Forests excel at this. They naturally work with such binary splits, allowing the model to easily analyze and derive patterns from boolean attributes. Heart failure isn't typically caused by a single factor. It's the culmination of various health issues and risk factors interacting in multifaceted ways. Random Forest can capture these intricate interactions between features, making it particularly apt for this kind of analysis. The Non-linearity of the data, as well as the ensemble learning method of the Random Forest approch can also justify our findings.

While heart failure remains a condition influenced by myriad factors, the insights we've gathered through

predictive modeling can offer perspectives for healthcare practitioners.