Introduction

Exploratory Data Analysis

Setting Up Models

Model Building

Test Performance for all models

Conclusion

Code ▾

Heart Failure Prediction

Mason Ma

Introduction

The purpose of this project is to generate a model that will predict whether people will tend to have heart failure given the 12 clinical features of themselves. The data we will be using originates from `kaggle` and we will utilize multiple machine learning techniques to yield the most accurate model for this binary classification problem.

Why is this model relevant?

Cardiovascular diseases (CVDs) are the number 1 cause of death globally, taking an estimated 17.9 million lives each year, which accounts for 31% of all deaths worlwide. Heart failure is a common event caused by CVDs and this dataset contains 12 features that can be used to predict mortality by heart failure.

Most cardiovascular diseases can be prevented by addressing behavioural risk factors such as tobacco use, unhealthy diet and obesity, physical inactivity and harmful use of alcohol using population-wide strategies.

People with cardiovascular disease or who are at high cardiovascular risk (due to the presence of one or more risk factors such as hypertension, diabetes, hyperlipidaemia or already established disease) need early detection and management wherein a machine learning model can be of great help.

Loading Data and Packages

Hide

```r
# loading packages
library(tidymodels)
library(patchwork)
library(tidyverse)
library(ISLR)
library(rpart.plot)
library(vip)
library(janitor)
library(randomForest)
library(xgboost)
library(ggplot2)
library(corrplot)
library(ggthemes)
library(klaR)
library(discrim)
library(poissonreg)
library(corrr)
library(extrafont)
library(wesanderson)
library(waffle)
library(ggridges)
library(ggpubr)
library(ranger)
library(kernlab)
# set seed
set.seed(2022)
```

Hide

```r
# loading data
original_data <- read.table('/Users/mason/Desktop/PSTAT131/masonma-131-final/data/unprocessed/heart_failure_original_data.csv',
                           sep=",",
                           header=TRUE)
```

We will analyze a dataset containing the medical records of 299 heart failure patients collected at the Faisalabad Institute of Cardiology and at the Allied Hospital in Faisalabad (Punjab, Pakistan), during April–December 2015. The full copy of the code book is available in my zipped files, here are some of the key variables that are helpful to be aware of for this report:

`Age` : Age of the patient // Numerical (Years)

`anaemia` : Decrease of red blood cells or hemoglobin // Boolean

`creatinine_phosphokinase` : Level of the CPK enzyme in the blood // Numerical (mcg/L)

`diabetes` : If the patient has diabetes // Boolean

`ejection_fraction` : Percentage of blood leaving the heart at each contraction // Numerical (Percentage)

`high_blood_pressure` : If a patient has hypertension // Boolean

`platelets` : Platelets in the blood // Numerical (kiloplatelets/mL)

`serum_creatinine` : Level of creatinine in the blood // Numerical (mg/dL)

`serum_sodium` : Level of sodium in the blood // Numerical (mEq/L)

`sex` : Woman or man // Boolean

`smoking` : If the patient smokes or not // Boolean

`time` : Follow-up period // Days

`DEATH_EVENT` : If the patient deceased during the follow-up period // Boolean

Note: mcg/L: micrograms per liter. mL: microliter. mEq/L: milliequivalents per litre

Exploratory Data Analysis

Before we implement any modeling techniques upon our data set, we need to have a look at what our data really looks like. When we load our data, not everything is going to be perfect and ready for application. For instance, there can be some variables that might need to converted to factors, or some missing values that might need to be cleaned. An exploratory data analysis is a thorough examination meant to uncover the underlying structure of a data set and is important for machine learning because it exposes trends, patterns, and relationships that are not readily apparent.

Hide

```
#checking for missing values in the dataset
table(is.na(original_data))
```

```
## 
## FALSE 
##  3887
```

Hide

```
original_data <- original_data %>%
  clean_names()

original_data %>%
  head()
```

```
##   age anaemia creatinine_phosphokinase diabetes ejection_fraction
## 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
##   high_blood_pressure platelets serum_creatinine serum_sodium sex smoking time
## 1                   1    265000              1.9          130   1       0    4
## 2                   0    263358              1.1          136   1       0    6
## 3                   0    162000              1.3          129   1       1    7
## 4                   0    210000              1.9          137   1       0    7
## 5                   0    327000              2.7          116   0       0    8
## 6                   1    204000              2.1          132   1       1    8
##   death_event
## 1           1
## 2           1
## 3           1
## 4           1
## 5           1
## 6           1
```

Hide

```
summary(original_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
```

How big is this data set that we have to work with now?

Hide

```
dim(original_data)
```

```
## [1] 299  13
```

There are 299 observations and 13 variables for each one of them.

We want the values of boolean variables like anaemia and sex to be of categorical type, so we will transform those into factor accordingly using the below code.

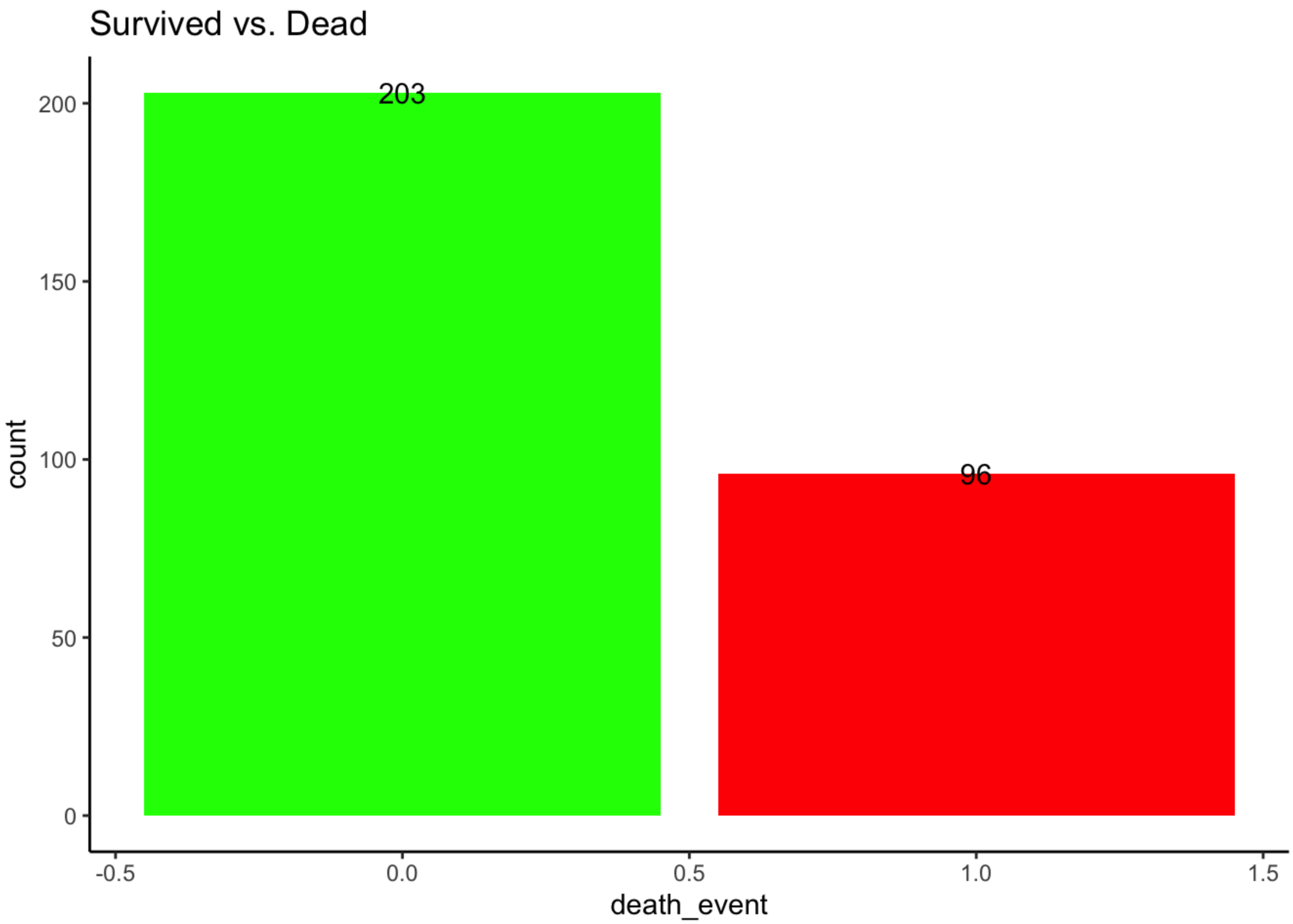
Hide

```
original_data$anaemia <- factor(original_data$anaemia)
original_data$diabetes <- factor(original_data$diabetes)
original_data$high_blood_pressure <- factor(original_data$high_blood_pressure)
original_data$sex <- factor(original_data$sex)
```

Survived vs. Dead

Then, we can have a look at how we can visualize the number of survived and dead patients in this data set.

Code

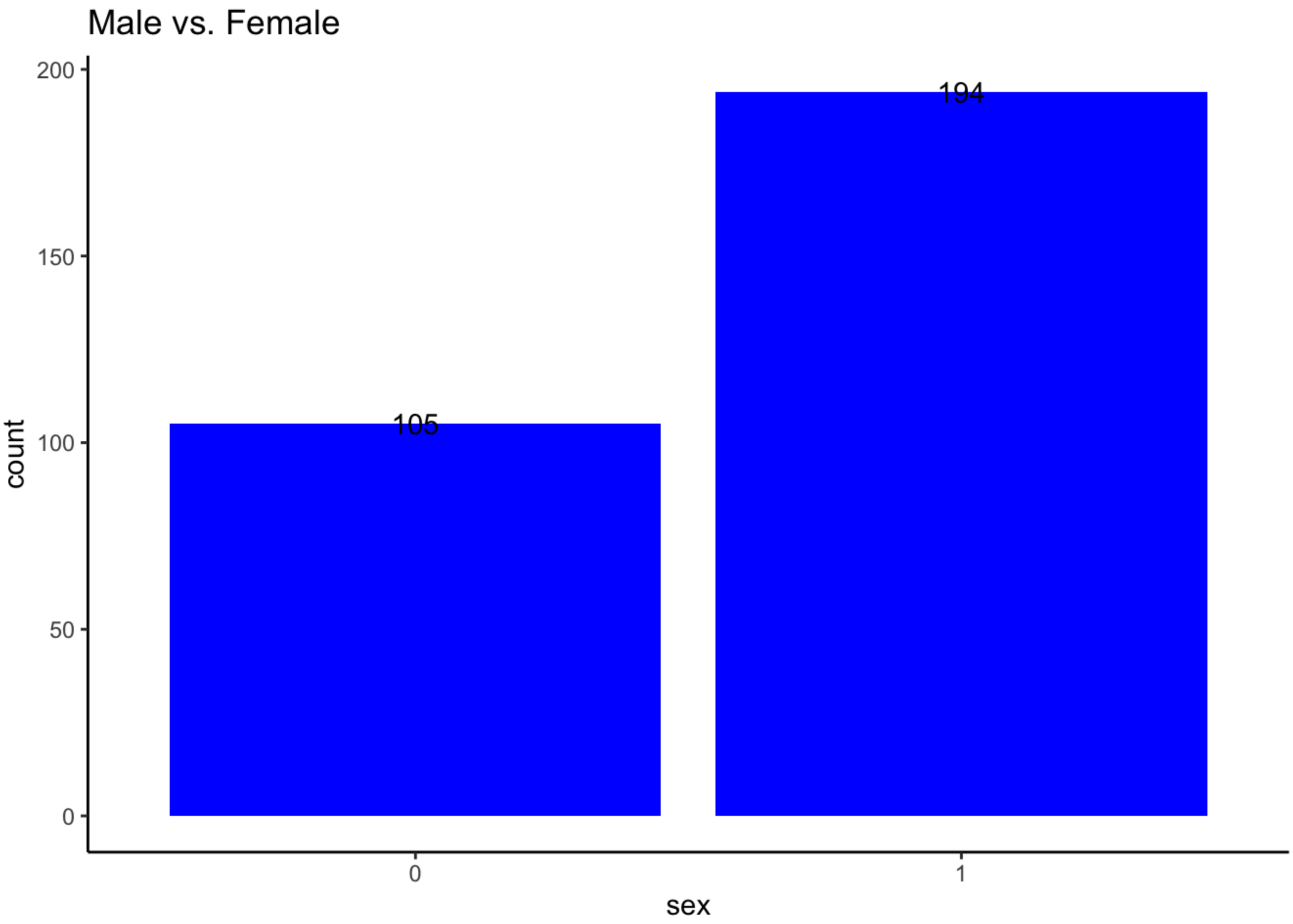


It is now quite obvious that our output is imbalanced, meaning that when we do the modeling, we need to stratify the output variable in order to achieve better results.

Males and Females

Let’s find the number of male and females in the dataset and see if it is evenly distributed.

Code



We can see that there exist 105 males and 194 females in this data set.

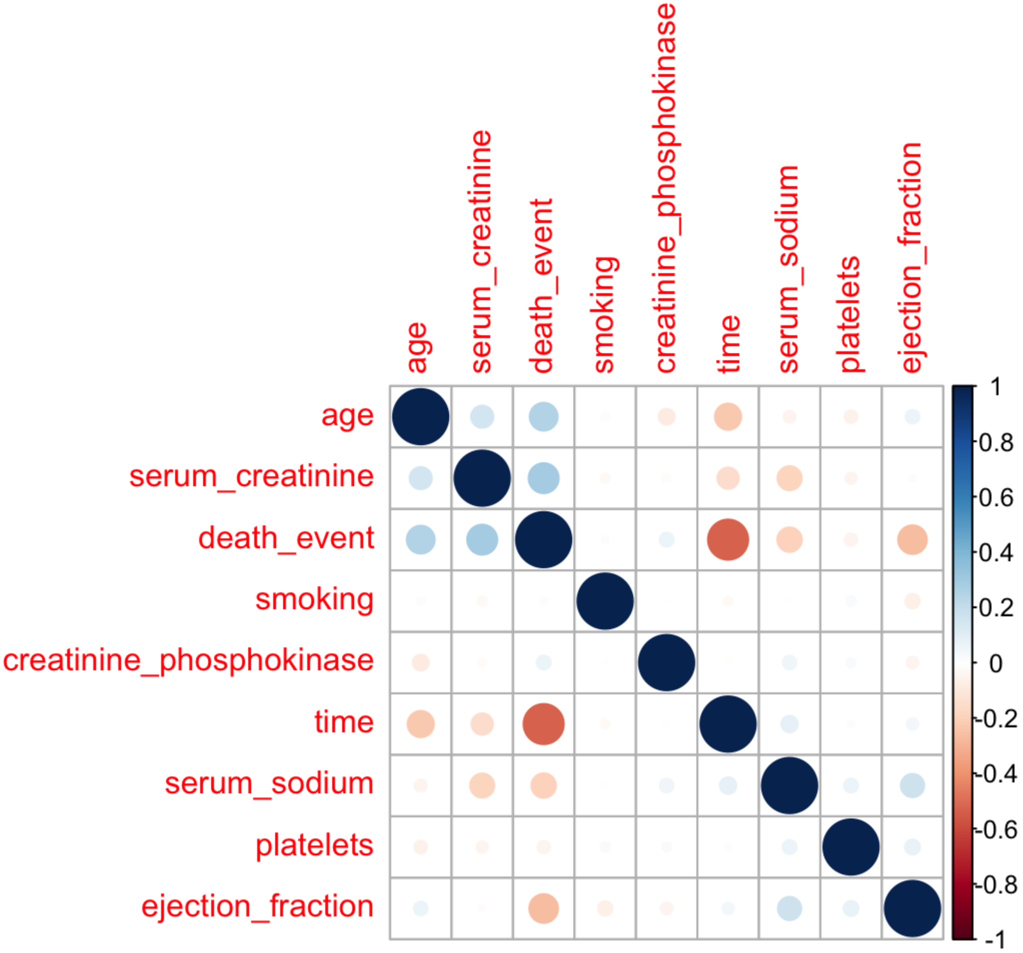
Correlation Plot

Correlation plots, also known as correlograms for more than two variables, help us to visualize the correlation between continuous variables. In this case, we can make a correlation heat map of the numeric variables to get an idea of their relationships.

Hide

```
original_data_numerical <- original_data %>%  # getting just the numeric data
  select_if(is.numeric)

ori_cor <- cor(original_data_numerical)  # calculating the correlation between each variable
ori_cor_plt <- corrplot(ori_cor,  # making the correlation plot
                               order = 'AOE') # Pink and Green color combo
```



At the first glance, we can find that there is such little correlation between a lot of our predictor variables, but after further analysis between each variable, it makes more sense.

==== Positively correlated relations:

Age vs. Level of creatinine in the blood

Age vs. Death_event

Level of creatinine in the blood vs. Death_event

Percentage of blood leaving the heart at each contraction vs. Level of sodium in the blood.

==== Negatively correlated relations:

Follow-up period vs. Age

Follow-up period vs. Level of creatinine in the blood

Follow-up period vs. Death_event

Level of creatinine in the blood vs. Level of sodium in the blood

Level of sodium in the blood vs. Death_event

Percentage of blood leaving the heart at each contraction vs. Death_event

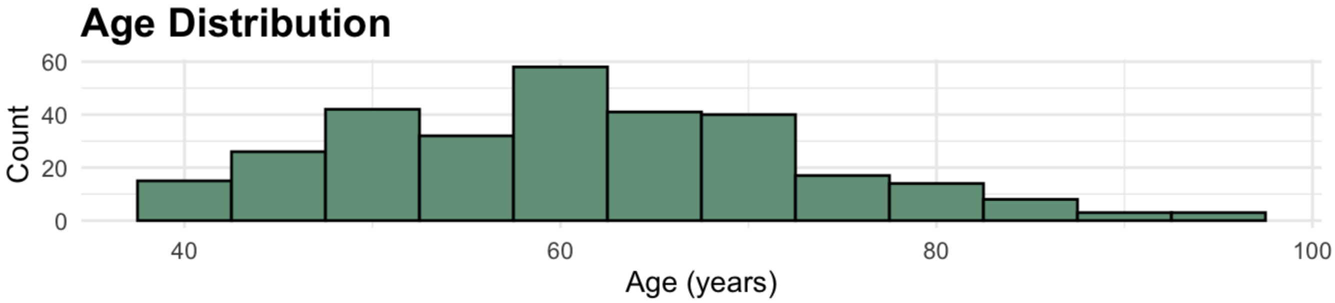
As can be seen, the outcome is most strongly correlated with time, age, ejection fraction, serum creatinine and serum sodium. Some of the key features are evaluated in more detail in the following.

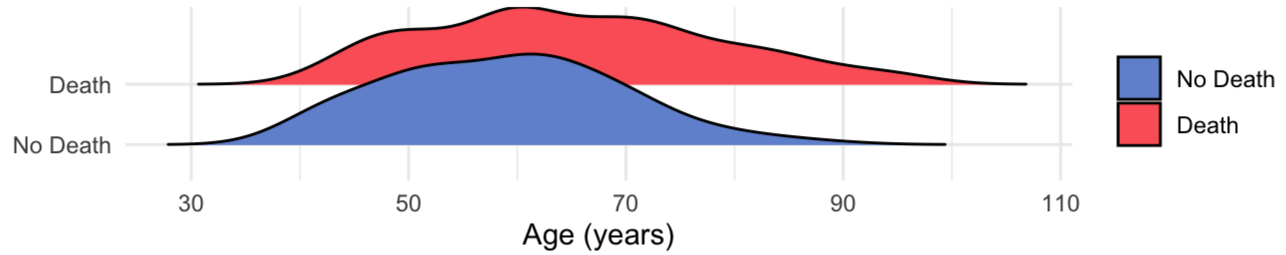
Ages

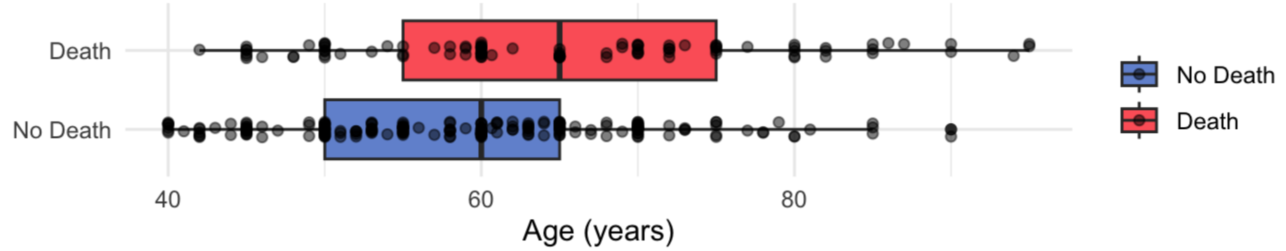
Higher age is expected to be associated with higher mortality. This assumption can be confirmed in the following histogram with overlayed density curves. The older patients died more often than the younger.

Code

```
## Picking joint bandwidth of 4.04
```





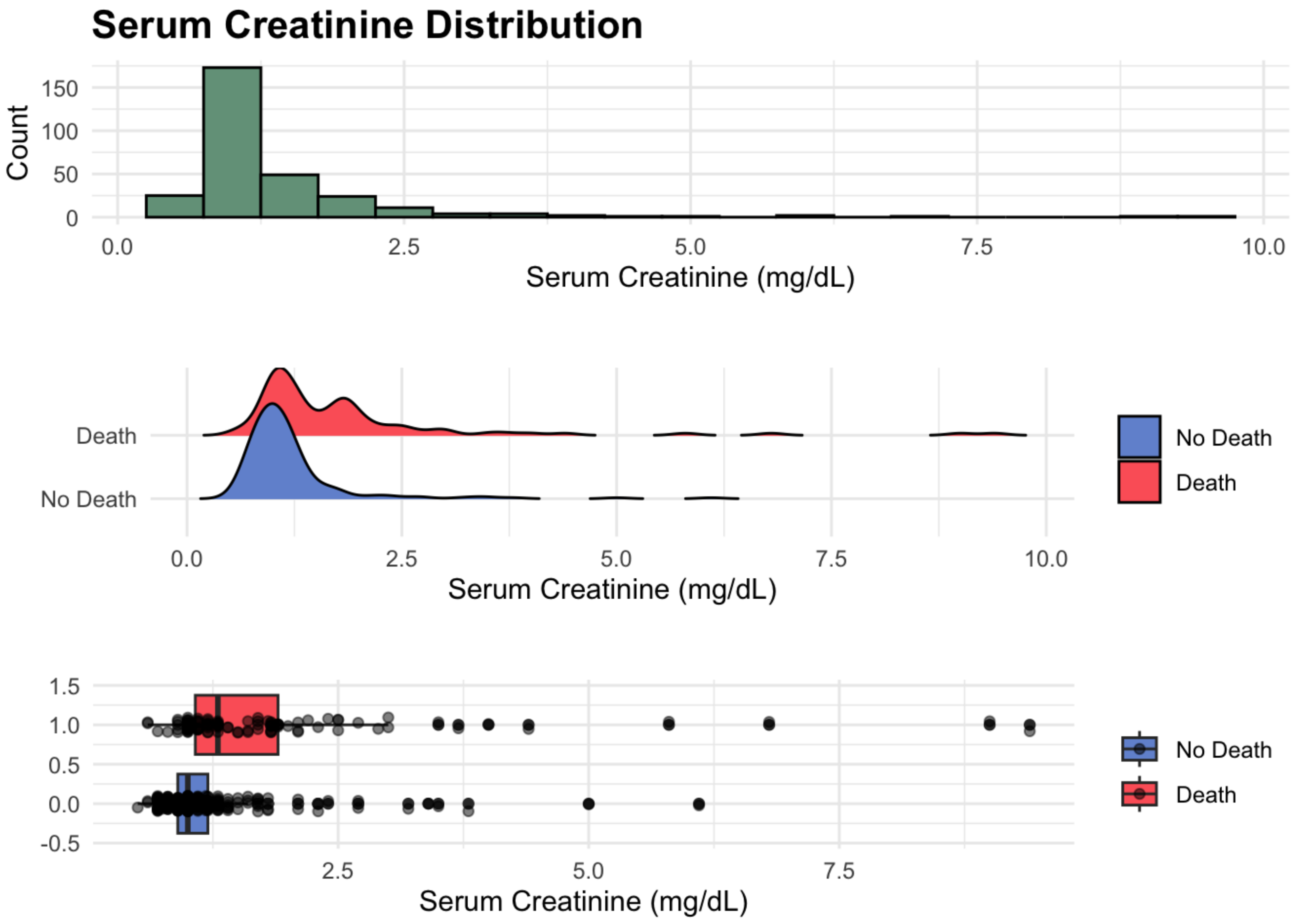


The distribution of age among those who died is centered slightly higher than the non death group. We can see that the aggregate age range is from 40 to just below 100.

Serum Creatinine

Code

```
## Picking joint bandwidth of 0.146
```

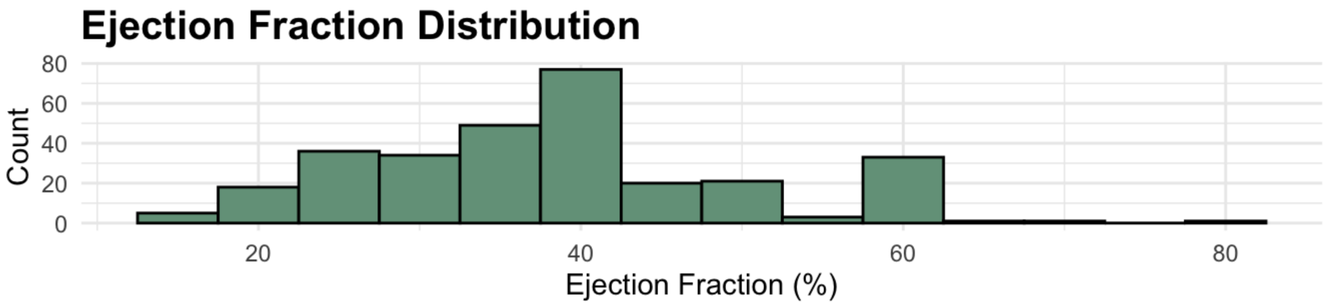


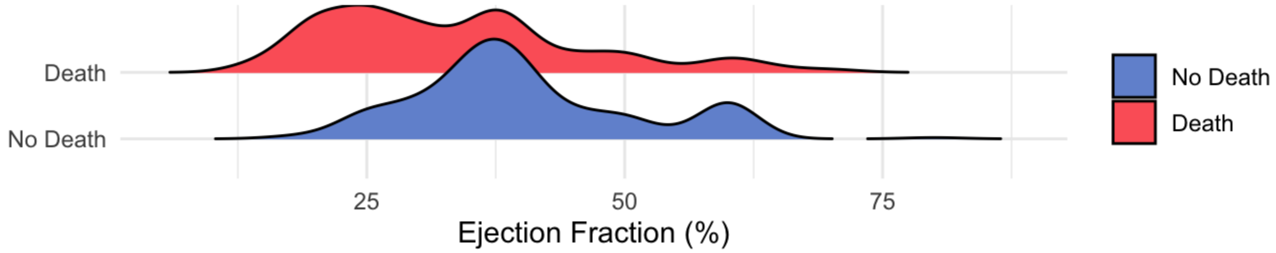
The Serum Creatinine distribution is right skewed and there are many outliers among the group who experienced death. Moreover, as we can see from the graphs above, the distribution in the death group is centered higher than that of the no death group.

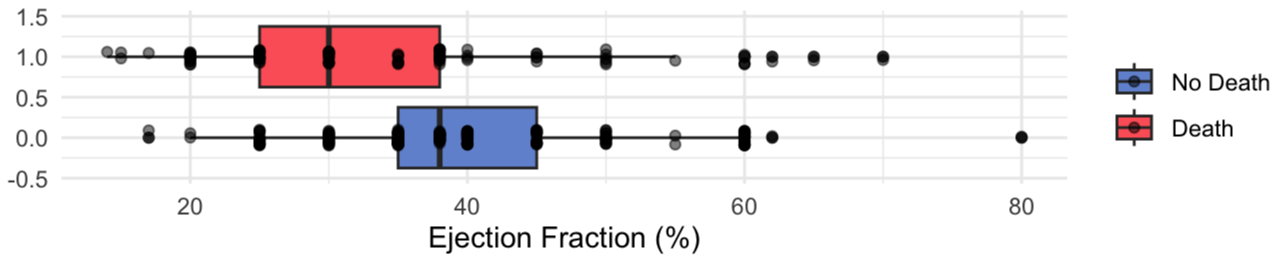
Ejection fraction

Code

```
## Picking joint bandwidth of 2.91
```







We know that ejection fraction strongly correlates with the outcome. It can be shown in the following graphs above. The distribution of Ejection Fraction is somewhat chaotic in this case, though the points in the boxplot reveal that the observations are clustered at certain values. This means the density plot is somewhat mis-leading. However, we can see that the distribution of Ejection Fraction among those who died is centered lower than those who did not die.

Setting Up Models

Before we do any model building, we already have some general idea about our data, and we have found that some variables such as age, serum creatinine, and ejection fraction are closely related to the response variable. Then, we have to perform a training / testing split on our data. I decided to go with 80/20 for this data because the testing data set will still have a significant amount of observations, but our model has more to train on and learn. The reason we do this is because we want to avoid over-fitting. Then we can build a recipe and make K-fold cross-validation sets.

Data Splitting

Hide

```
original_data$death_event<-factor(original_data$death_event)

data_split <- original_data %>%
  initial_split(prop = 0.8, strata = "death_event")

data_train <- training(data_split) # training split
data_test <- testing(data_split) # testing split
```

Hide

```
dim(data_train)
```

```
## [1] 238  13
```

Hide

```
dim(data_test)
```

```
## [1] 61 13
```

There are now 238 observations (13 variables) in the training dataset, and 61 observations (13 variables) in the testing dataset, both will be sufficient for model building. We did the stratified sampling because our output variable is highly imbalanced, so this step is necessary.

Recipe Building

Hide

```
data_recipe <- recipe(
  death_event~.,data_train) %>%
  step_dummy(all_nominal_predictors())%>%
  step_normalize()
```

We have to normalize the data since some of our variables are extremely small, while others are quite large compared to them.

K-Fold Cross Validation

Cross-validation is a statistical method used to estimate the skill of machine learning models. It is commonly used in applied machine learning to compare and select a model for a given predictive modeling problem because it is easy to understand, easy to implement, and results in skill estimates that generally have a lower bias than other methods. Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. In this case, we will use five folds since our data is not too large. Although this approach can be computationally expensive, we will make use of every bit of our data, which is a major advantage when our sample size is small.

Hide

```
data_folds <- vfold_cv(data_train, v = 5, strata = death_event)  # 5-fold CV
```

Model Building

We will be implementing six models: logistic regression, QDA, support vector machine, classification tree, random forest, and boosted tree like we did in the homework and labs.

We will do hyper-parameter tuning for all three tree-based models, and cross-validation for five models. For evaluation for all the models, we will make use of the confusion matrix, accuracy, and precision. Precision will be weighted more heavily due to the imbalance of response variables. We expect to see (non-parametric) tree-based models perform better than (parametric) logistic regression and quadratic discriminant analysis (QDA).

Logistic Regression

In statistics, the logistic model is a statistical model that models the probability of an event taking place by having the log-odds for the event be a linear combination of one or more independent variables. In regression analysis, logistic regression is estimating the parameters of a logistic model.

Hide

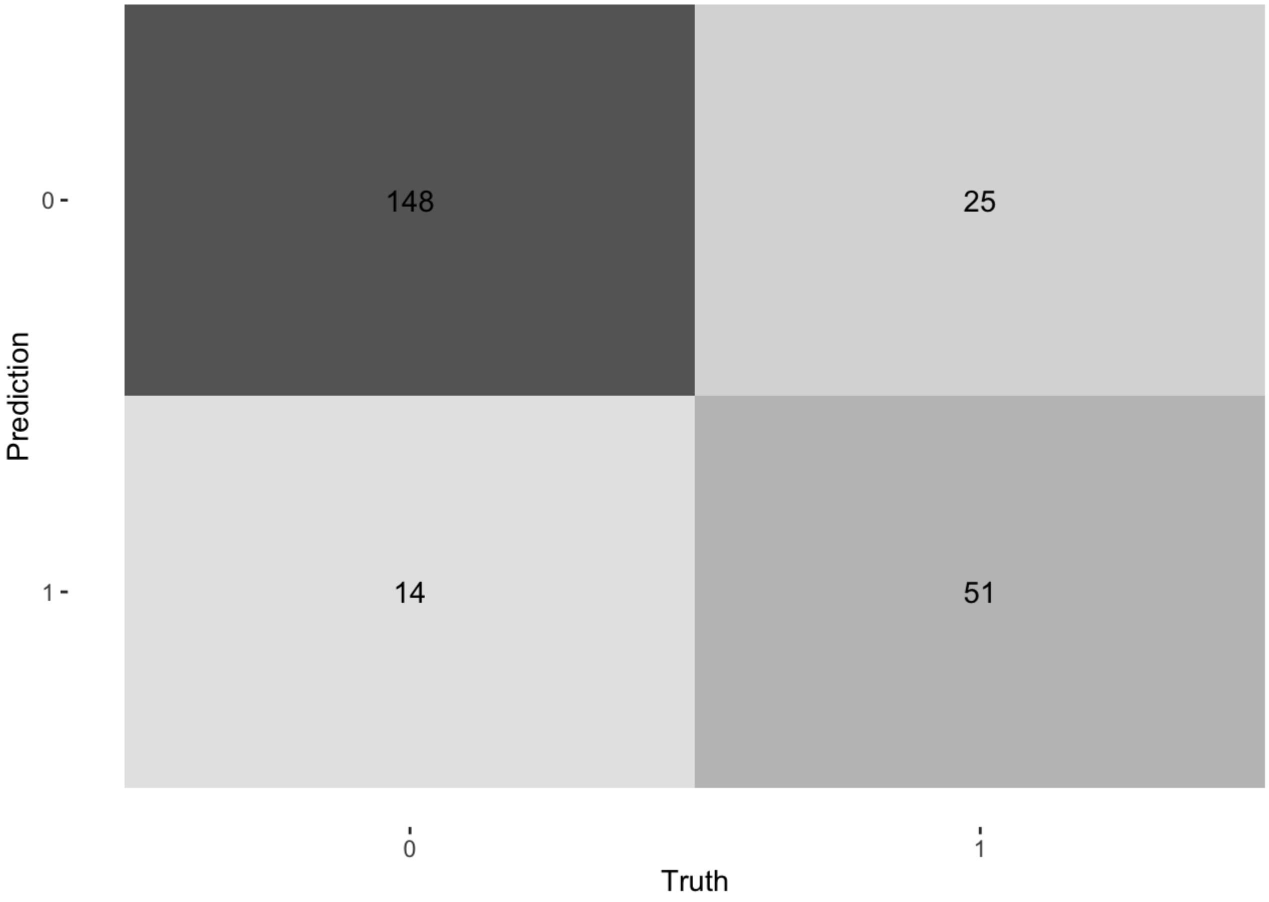
```
log_reg <-logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

log_wkflow<-workflow() %>%
  add_model(log_reg)%>%
  add_recipe(data_recipe)

log_fit_res<-tune_grid(
  object=log_wkflow,
  resamples=data_folds # for cross-validation
  )

log_fit<-fit(log_wkflow,data_train)

augment(log_fit,new_data=data_train) %>%
  conf_mat(truth=death_event,estimate=.pred_class)%>%
  autoplot(type="heatmap")
```



Hide

```
log_acc<-augment(log_fit,new_data=data_train)%>%
  accuracy(truth=death_event,estimate=.pred_class)
print(log_acc)
```

```
## # A tibble: 1 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.836
```

Hide

```
log_precision<-augment(log_fit,data_train)%>%
  precision(death_event,.pred_class)
print(log_precision)
```

```
## # A tibble: 1 × 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision binary         0.855
```

In this case, the accuracy for this Logistic Regression Model is 0.836 and the precision is 0.855. As for a relatively simple modeling method, it does perform pretty well on whether a patient will have a heart attack under the conditions given in the data set. Accuracy is the proportion of correct predictions over total predictions and precision is the ratio of correctly predicted positive observations to the total predicted positive observations. Since the output variable (death_event) is greatly imbalanced as we have discussed above, precision is actually a better metric than accuracy because a false negative does not give us too much information given the fact of the majority of the response is false (death_event=0).

Quadratic Discriminant Analysis

Quadratic Discriminant Analysis (QDA) is a generative model. QDA assumes that each class follow a Gaussian distribution. The class-specific prior is simply the proportion of data points that belong to the class. The class-specific mean vector is the average of the input variables that belong to the class.

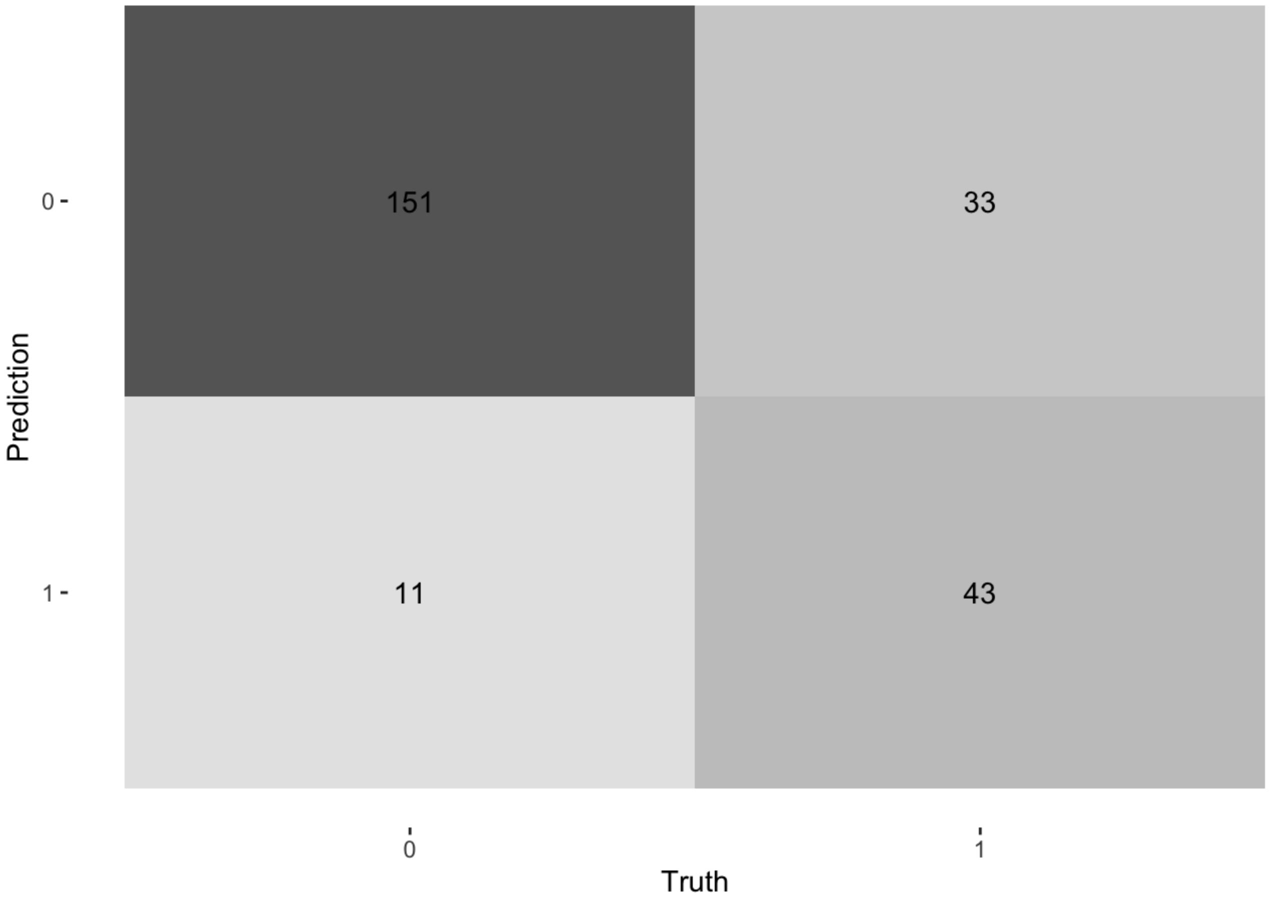
Hide

```
qda_mod<-discrim_quad()%>%
  set_mode("classification")%>%
  set_engine("MASS")

qda_wkflow<-workflow()%>%
  add_model(qda_mod)%>%
  add_recipe(data_recipe)

qda_fit<-fit(qda_wkflow,data_train)

augment(qda_fit,new_data=data_train) %>%
  conf_mat(truth=death_event,estimate=.pred_class)%>%
  autoplot(type="heatmap")
```



Hide

```
qda_acc<-augment(qda_fit,new_data=data_train)%>%
  accuracy(truth=death_event,estimate=.pred_class)
print(qda_acc)
```

```
## # A tibble: 1 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.815
```

Hide

```
qda_precision<-augment(qda_fit,data_train)%>%
  precision(truth=death_event,.pred_class)
print(qda_precision)
```

```
## # A tibble: 1 × 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision binary         0.821
```

In this case, the accuracy for this Logistic Regression Model is 0.815 and the precision is 0.821. Indeed, this QDA model performs similarly to logistic regression and it did a pretty decent job. However, since the precision/accuracy of the two models are less than 0.9, we will try to improve this or even higher using other advanced models.

Support Vector Machine

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection. The advantages of support vector machines are: 1)Effective in high dimensional spaces. 2)Still effective in cases where number of dimensions is greater than the number of samples. 3)Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient. 4)Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

Hide

```
svm_rbf_spec <- svm_rbf() %>%
  set_mode("classification") %>%
  set_engine("kernlab", scaled = FALSE)

svm_rbf_fit <- svm_rbf_spec %>%
  fit(death_event ~ ., data=data_train)
```

Hide

```
mtx<-augment(svm_rbf_fit, new_data = data_train) %>%
  conf_mat(truth = death_event, estimate = .pred_class)%>%
  autoplot(type="heatmap")

svm_acc<-augment(svm_rbf_fit,new_data=data_train)%>%
  accuracy(truth=death_event,estimate=.pred_class)
print(svm_acc)
```

```
## # A tibble: 1 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.697
```

Hide

```
svm_precision<-augment(svm_rbf_fit,data_train)%>%
  precision(death_event,.pred_class)
print(svm_precision)
```

```
## # A tibble: 1 × 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision binary         0.694
```

The accuracy is 0.702 and the precision is 0.696 in this case. Using SVM does not seem to perform better than the previous methods.

Classification Tree Model

Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. The Classification and regression tree(CART) methodology are one of the oldest and most fundamental algorithms. It is used to predict outcomes based on certain predictor variables. They are excellent for data mining tasks because they require very little data pre-processing.

Hide

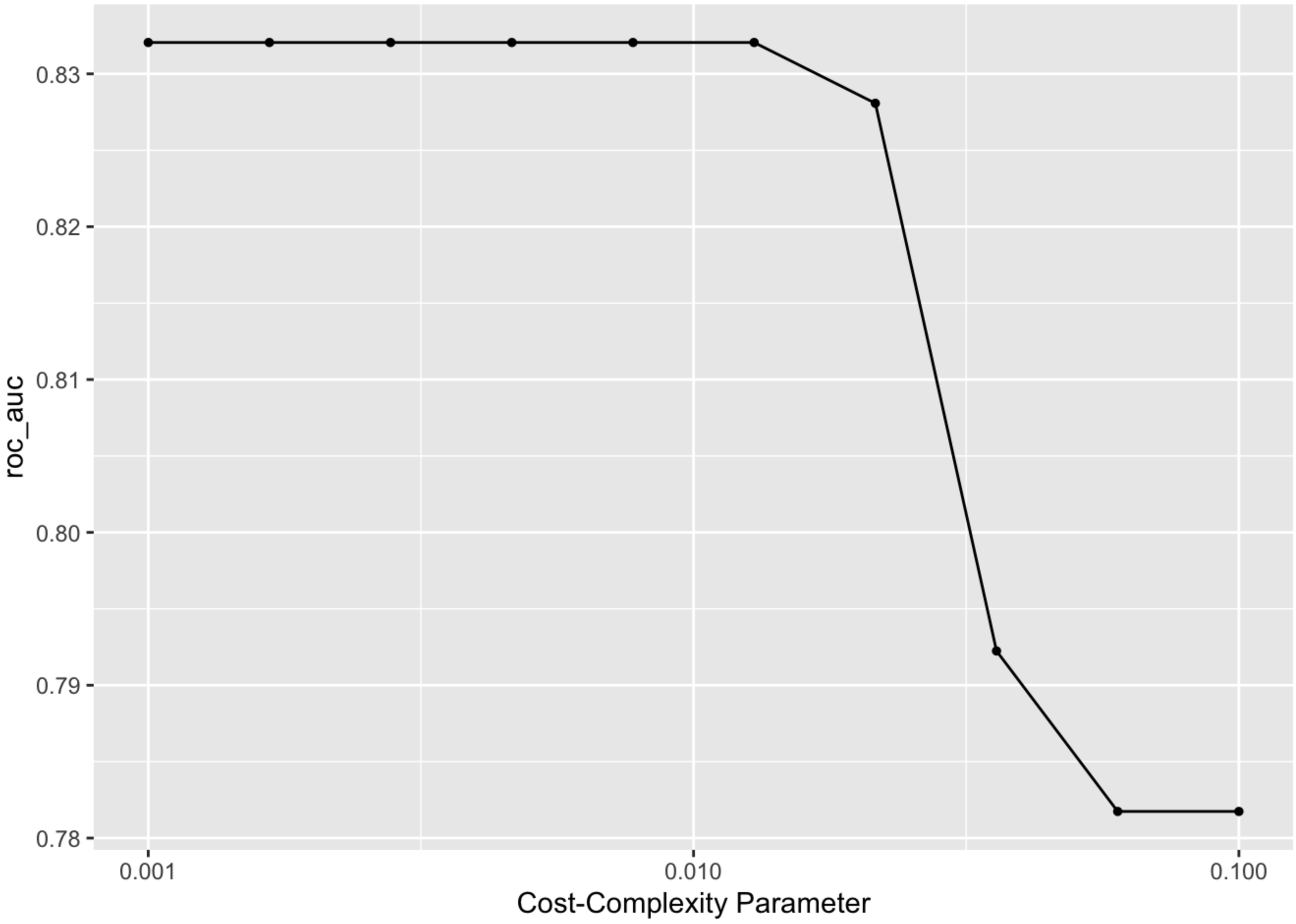
```
data_tree_spec <-decision_tree() %>%
  set_engine("rpart")

data_class_tree_spec <- data_tree_spec %>%
  set_mode("classification")

data_class_tree_wkflow <- workflow() %>%
  add_model(data_class_tree_spec %>%
  set_args(cost_complexity=tune())) %>%
  add_formula(death_event~.)

data_param_grid <-grid_regular(cost_complexity(range=c(-3,-1)),levels=10)

data_class_tree_res <-tune_grid(
  data_class_tree_wkflow,
  resamples=data_folds,
  grid=data_param_grid,
  metrics=metric_set(roc_auc)) # for cross-validation
autoplot(data_class_tree_res)
```



Hide

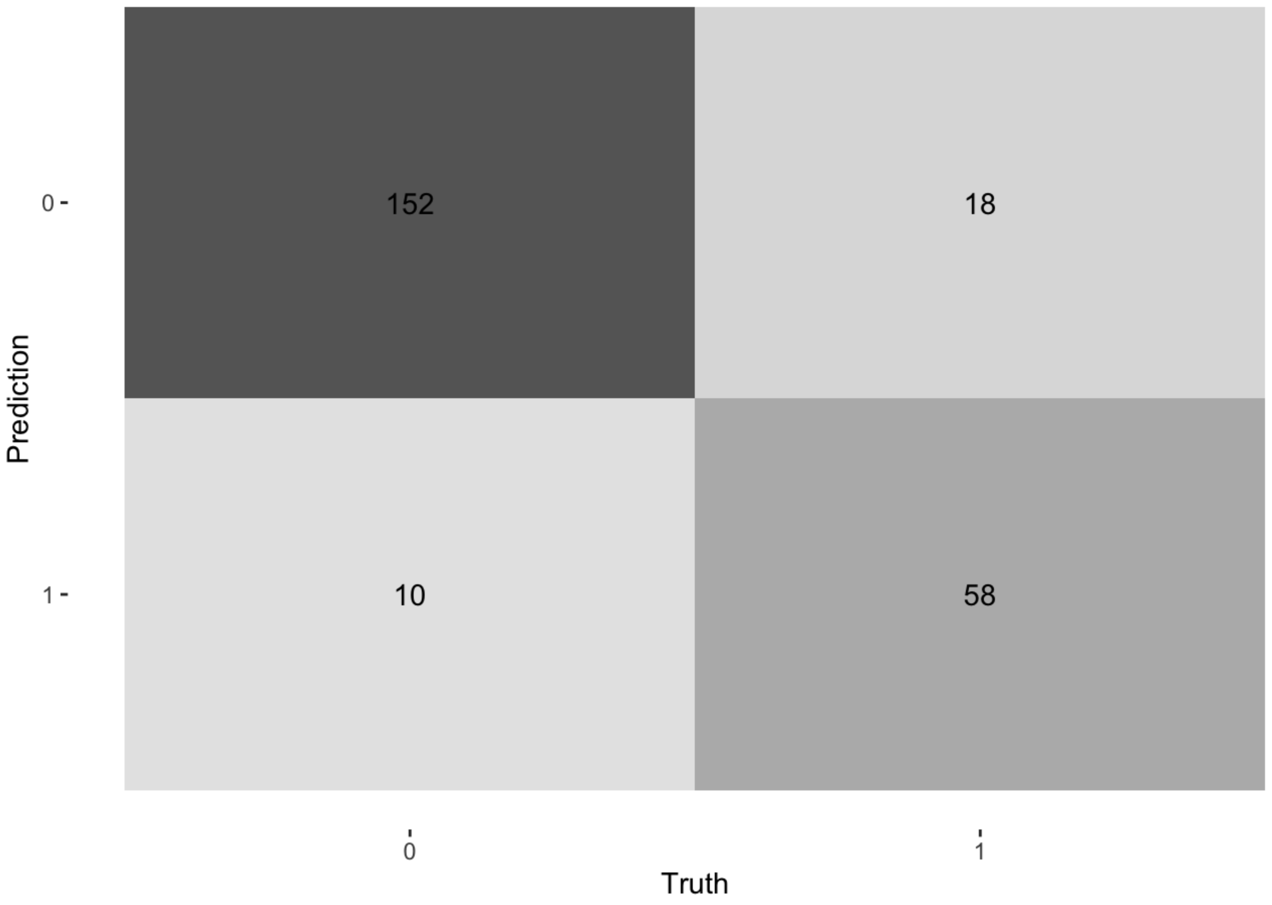
```
best_class_tree_complexity <-select_best(data_class_tree_res)
collect_metrics(data_class_tree_res) %>% arrange(-mean)
```

```
## # A tibble: 10 × 7
##    cost_complexity .metric .estimator  mean     n std_err .config
##              <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1         0.001   roc_auc binary     0.832     5  0.0343 Preprocessor1_Model01
##  2         0.00167 roc_auc binary     0.832     5  0.0343 Preprocessor1_Model02
##  3         0.00278 roc_auc binary     0.832     5  0.0343 Preprocessor1_Model03
##  4         0.00464 roc_auc binary     0.832     5  0.0343 Preprocessor1_Model04
##  5         0.00774 roc_auc binary     0.832     5  0.0343 Preprocessor1_Model05
##  6         0.0129  roc_auc binary     0.832     5  0.0343 Preprocessor1_Model06
##  7         0.0215  roc_auc binary     0.828     5  0.0360 Preprocessor1_Model07
##  8         0.0359  roc_auc binary     0.792     5  0.0451 Preprocessor1_Model08
##  9         0.0599  roc_auc binary     0.782     5  0.0406 Preprocessor1_Model09
## 10         0.1     roc_auc binary     0.782     5  0.0406 Preprocessor1_Model10
```

From the autoplot of cost complexity parameter, it seems to be the one with the smallest complexity cost yields the highest ROC_AUC, which is 0.832.

Hide

```
class_tree_final<-finalize_workflow(data_class_tree_wkflow,best_class_tree_complexity)
class_tree_final_fit<-fit(class_tree_final,data=data_train)
augment(class_tree_final_fit,new_data=data_train) %>%
  conf_mat(truth=death_event,estimate=.pred_class)%>%
  autoplot(type="heatmap")
```



Hide

```
class_tree_acc<-augment(class_tree_final_fit,new_data=data_train)%>%
  accuracy(truth=death_event,estimate=.pred_class)
print(class_tree_acc)
```

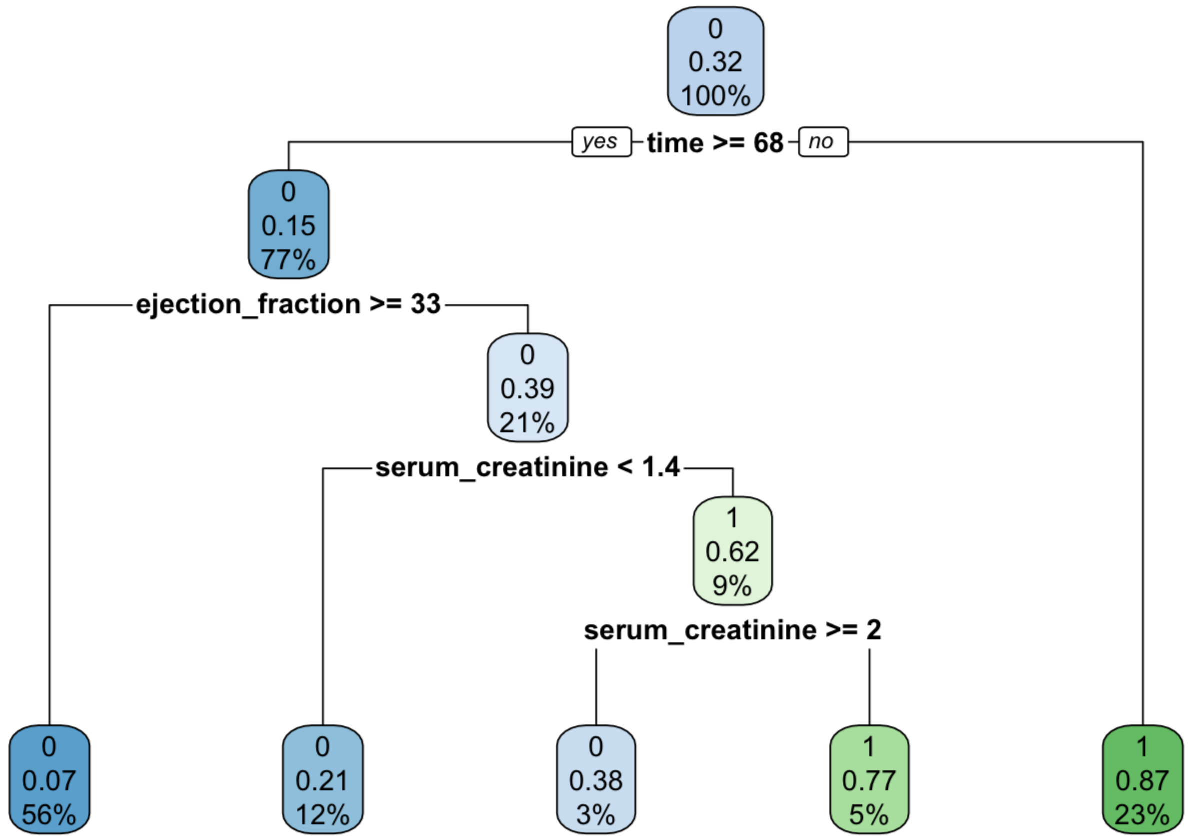
```
## # A tibble: 1 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.882
```

Hide

```
class_tree_precision<-augment(class_tree_final_fit,data_train)%>%
  precision(death_event,.pred_class)
print(class_tree_precision)
```

```
## # A tibble: 1 × 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision binary         0.894
```

Code



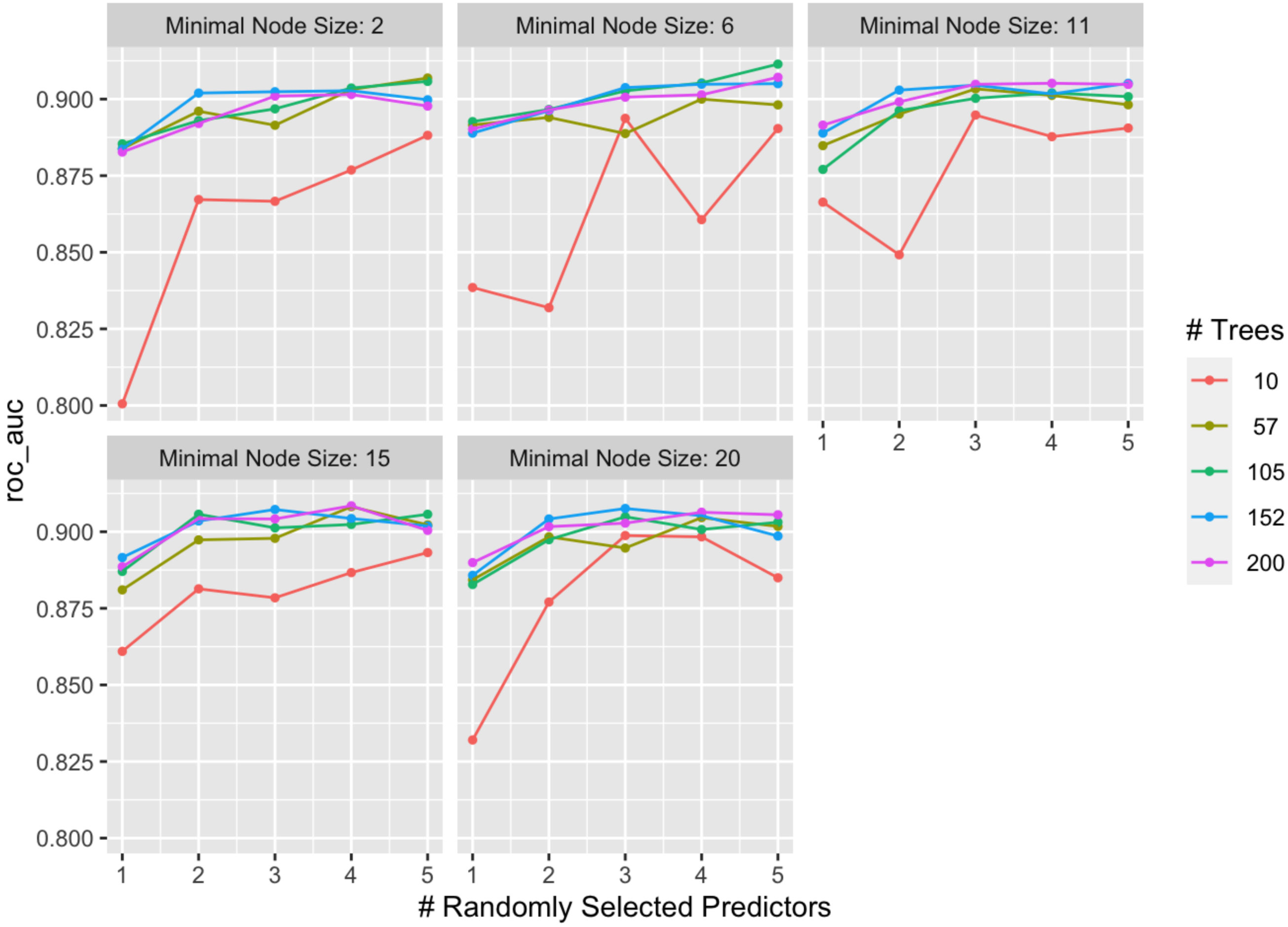
In this case where we implement the Classification tree model, we received excellent results: accuracy is 0.882 and precision is 0.894. These results are much better than the previous ones. One critical reason that contributed to this improvement is that our data set is relatively simple and convenient to predict under this method.

Random Forest Model

The random forest is a classification algorithm consisting of many decisions trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

Hide

```
data_f_spec<-rand_forest() %>%
  set_engine("ranger",importance="impurity")%>%
  set_mode("classification")
data_f_wkflow<-workflow()%>%
  add_model(data_f_spec %>% set_args(mtry=tune(),trees=tune(),min_n=tune()))%>%
  add_formula(death_event~.)
param_grid_f<-grid_regular(mtry(range= c(1,5)),
                           trees(range = c(10,200)),min_n(range = c(2,20)),levels = 5)
data_res_f <-tune_grid(
  data_f_wkflow,
  resample=data_folds,
  grid=param_grid_f,
  metrics=metric_set(roc_auc)
)
autoplot(data_res_f)
```



From the output above, we will see when the tree size is small, the roc_auc is noticeably low, but when the tree size is above 100, the difference of roc_auc is not substantial across all the graphs. Among all the tree hyper-parameters, the number of trees seems to determine the result the most significantly.

Hide

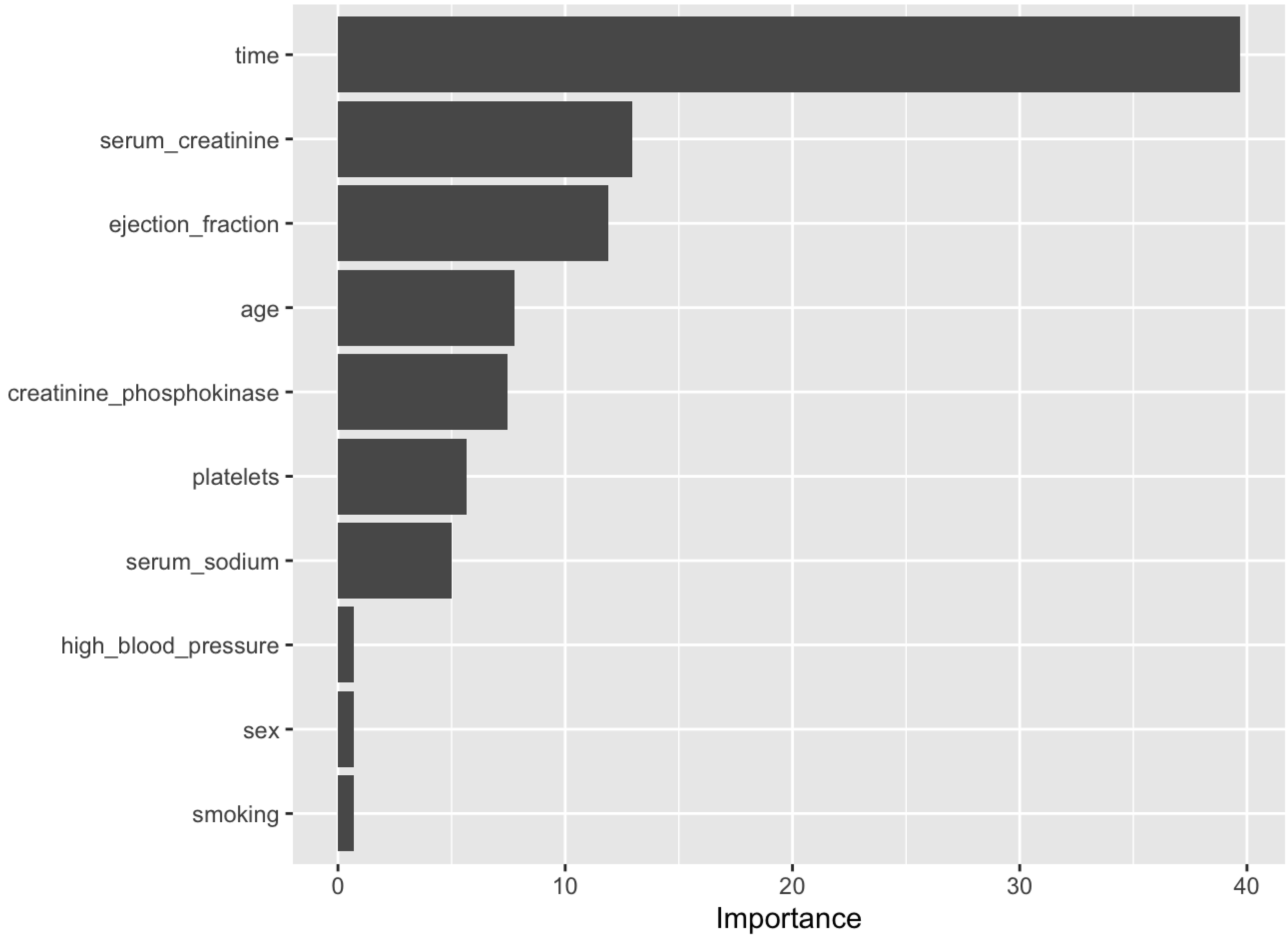
```
collect_metrics(data_res_f) %>% arrange(-mean)
```

```
## # A tibble: 125 × 9
##     mtry trees min_n .metric .estimator  mean     n std_err .config
##    <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1     5   105     6 roc_auc binary     0.911     5  0.0231 Preprocessor1_Model…
##  2     4   200    15 roc_auc binary     0.908     5  0.0243 Preprocessor1_Model…
##  3     4    57    15 roc_auc binary     0.908     5  0.0202 Preprocessor1_Model…
##  4     3   152    20 roc_auc binary     0.908     5  0.0242 Preprocessor1_Model…
##  5     3   152    15 roc_auc binary     0.907     5  0.0255 Preprocessor1_Model…
##  6     5   200     6 roc_auc binary     0.907     5  0.0234 Preprocessor1_Model…
##  7     5    57     2 roc_auc binary     0.907     5  0.0232 Preprocessor1_Model…
##  8     4   200    20 roc_auc binary     0.906     5  0.0265 Preprocessor1_Model…
##  9     5   105     2 roc_auc binary     0.906     5  0.0233 Preprocessor1_Model…
## 10     2   105    15 roc_auc binary     0.906     5  0.0243 Preprocessor1_Model…
## # … with 115 more rows
```

From the autoplot of cost complexity parameter, it seems that the highest ROC_AUC is 91.14.

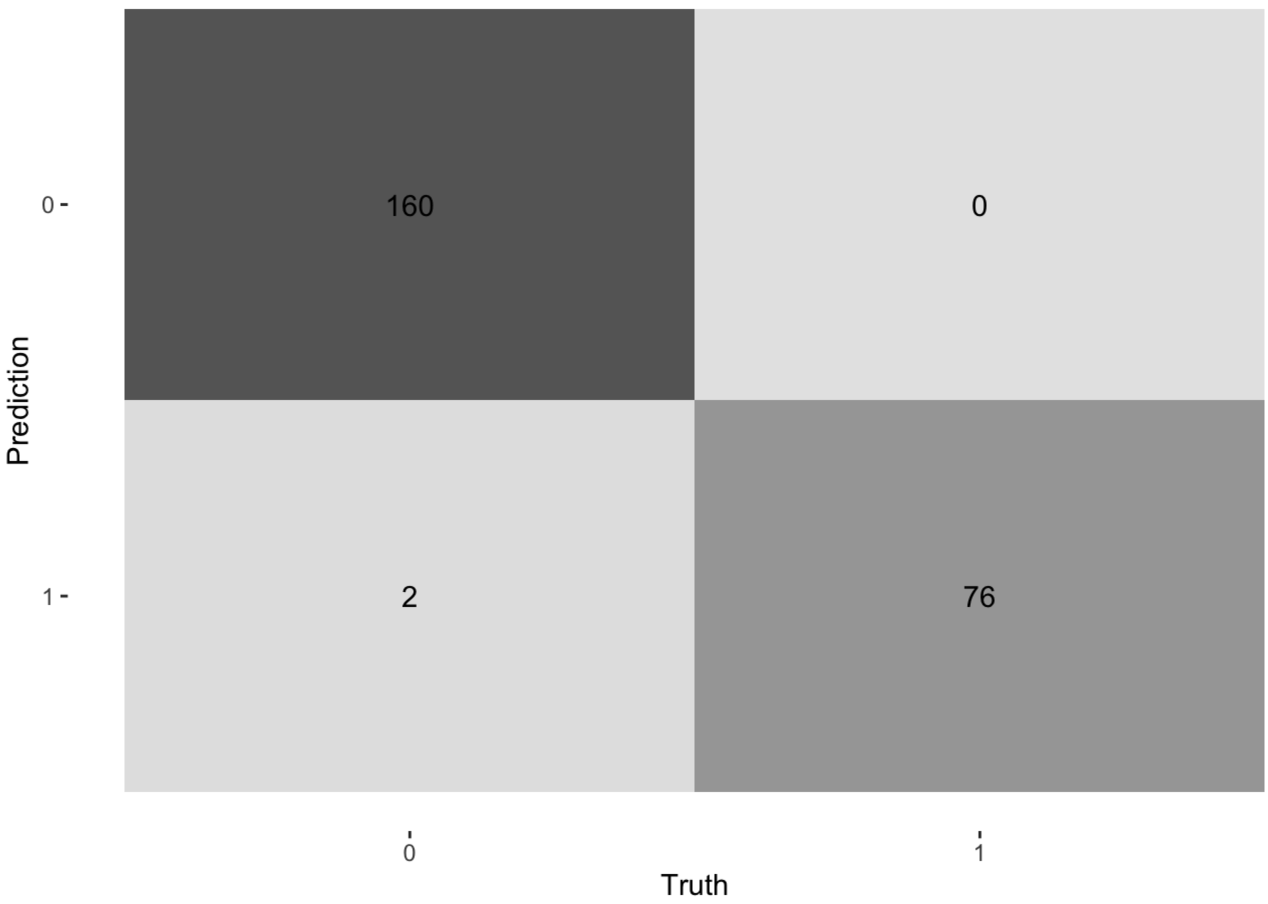
Hide

```
best_complexity_f<-select_best(data_res_f)
f_final<-finalize_workflow(data_f_wkflow,best_complexity_f)
f_final_fit<-fit(f_final,data=data_train)
f_final_fit %>%
  extract_fit_engine()%>%
  vip()
```



Hide

```
best_complexity_f<-select_best(data_res_f)
f_final<-finalize_workflow(data_f_wkflow,best_complexity_f)
f_final_fit<-fit(f_final,data=data_train)
augment(f_final_fit,new_data=data_train) %>%
  conf_mat(truth=death_event,estimate=.pred_class)%>%
  autoplot(type="heatmap")
```



Hide

```
f_acc<-augment(f_final_fit,new_data=data_train)%>%
  accuracy(truth=death_event,estimate=.pred_class)
print(f_acc)
```

```
## # A tibble: 1 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.992
```

Hide

```
f_precision<-augment(f_final_fit,data_train)%>%
  precision(death_event,.pred_class)
print(f_precision)
```

```
## # A tibble: 1 × 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision binary             1
```

In this case, the random forest model performs extraordinary and even outperforms the classification tree model. The roc_auc is 0.914, accuracy is 0.9915, and our precision is 1. Compared to all the models we have used previously, this one seems to be the best on every aspect.

Boosted Tree Model

Boosting is a method of combining many weak learners (typically decision trees) into a strong classifier.

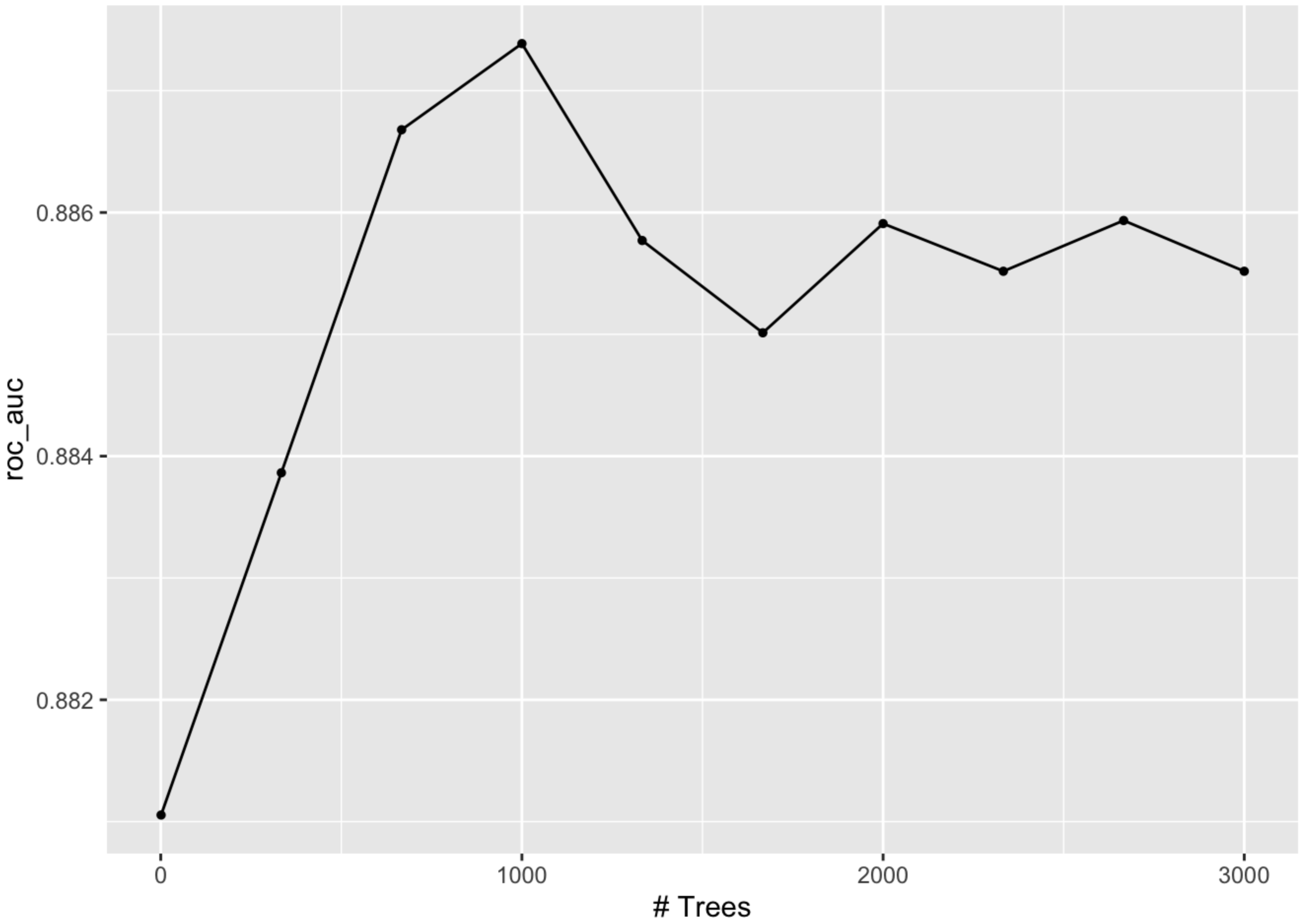
Hide

```
data_boost_spec<-boost_tree()%>%
  set_engine("xgboost")%>%
  set_mode("classification")

data_boost_Wkflow<-workflow()%>%
  add_model(data_boost_spec %>% set_args(trees=tune()))%>%
  add_formula(death_event~.)

param_grid_boost<-grid_regular(trees(range = c(1,3000)),levels = 10)

tune_res_boost <-tune_grid(
  data_boost_Wkflow,
  resample=data_folds,
  grid=param_grid_boost,
  metrics=metric_set(roc_auc)
)
autoplot(tune_res_boost)
```



From the output above, we can see the best result when the number of trees is around 1000. It seems that when we increase the tree size at the beginning, the roc_auc improves significantly. However, when the number of trees exceeds around 1000, the roc_auc will drop slightly. Also, we can see the range between the best one and the worst one is only about 0.04, implying that the model performs well under all situations.

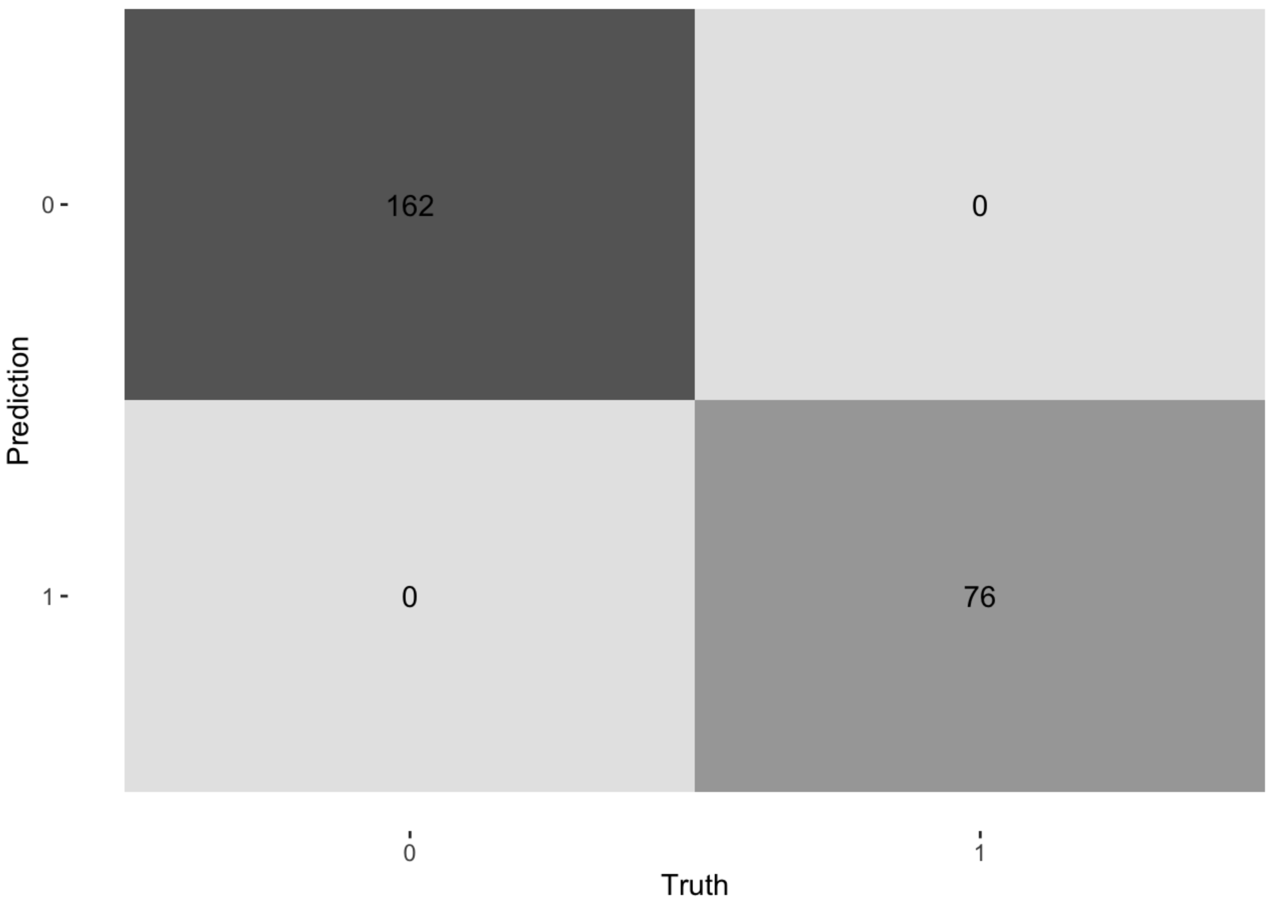
Hide

```
collect_metrics(tune_res_boost)%>%arrange(-mean)
```

```
## # A tibble: 10 × 7
##    trees .metric .estimator  mean     n std_err .config
##    <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1  1000 roc_auc binary     0.887     5  0.0228 Preprocessor1_Model04
##  2   667 roc_auc binary     0.887     5  0.0238 Preprocessor1_Model03
##  3  2666 roc_auc binary     0.886     5  0.0242 Preprocessor1_Model09
##  4  2000 roc_auc binary     0.886     5  0.0234 Preprocessor1_Model07
##  5  1333 roc_auc binary     0.886     5  0.0231 Preprocessor1_Model05
##  6  2333 roc_auc binary     0.886     5  0.0240 Preprocessor1_Model08
##  7  3000 roc_auc binary     0.886     5  0.0240 Preprocessor1_Model10
##  8  1667 roc_auc binary     0.885     5  0.0236 Preprocessor1_Model06
##  9   334 roc_auc binary     0.884     5  0.0240 Preprocessor1_Model02
## 10     1 roc_auc binary     0.881     5  0.0294 Preprocessor1_Model01
```

Hide

```
best_tree_boost<-select_best(tune_res_boost)
boost_final<-finalize_workflow(data_boost_Wkflow,best_tree_boost)
boost_final_fit<-fit(boost_final,data=data_train)
augment(boost_final_fit,new_data=data_train) %>%
  conf_mat(truth=death_event,estimate=.pred_class)%>%
  autoplot(type="heatmap")
```



Hide

```
boost_acc<-augment(boost_final_fit,new_data=data_train)%>%
  accuracy(truth=death_event,estimate=.pred_class)
print(boost_acc)
```

```
## # A tibble: 1 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary             1
```

Hide

```
boost_precision<-augment(boost_final_fit,data_train)%>%
  precision(death_event,.pred_class)
print(boost_precision)
```

```
## # A tibble: 1 × 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision binary             1
```

In this case, the accuracy and precision are both 1. This is really amazing since it will not make a single mistake using this machine learning method. But, we must admit that our data set is comparatively simple and convenient to predict. We will see how well it will be on the testing data set.

Test Performance for all models

Although the value of accuracy seems to be high for all models on training data, it is important to understand that accuracy cannot evaluate the overall performance of models . Precision will be weighted more heavily due to the imbalance of response variables.

On training data set, Random Forest Model and Boosted Tree Model are the two models with comparatively high accuracy and precision (0.93 and 1). Then, we apply these two methods separately on the testing data set.

For Random Forest Model:

Hide

```
augment(f_final_fit,new_data=data_test) %>%
  conf_mat(truth=death_event,estimate=.pred_class)
```

```
##           Truth
## Prediction  0  1
##          0 37  5
##          1  4 15
```

Hide

```
f_acc<-augment(f_final_fit,new_data=data_test)%>%
  accuracy(truth=death_event,estimate=.pred_class)
print(f_acc)
```

```
## # A tibble: 1 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.852
```

Hide

```
f_precision<-augment(f_final_fit,data_test)%>%
  precision(death_event,.pred_class)
print(f_precision)
```

```
## # A tibble: 1 × 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision binary         0.881
```

Accuracy: 0.8852

Precision: 0.9047

For Boosted Tree Model:

Hide

```
augment(boost_final_fit,new_data=data_test) %>%
  conf_mat(truth=death_event,estimate=.pred_class)
```

```
##           Truth
## Prediction  0  1
##          0 36  5
##          1  5 15
```

Hide

```
boost_acc<-augment(boost_final_fit,new_data=data_test)%>%
  accuracy(truth=death_event,estimate=.pred_class)
print(boost_acc)
```

```
## # A tibble: 1 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.836
```

Hide

```
boost_precision<-augment(boost_final_fit,data_test)%>%
  precision(death_event,.pred_class)
print(boost_precision)
```

```
## # A tibble: 1 × 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision binary         0.878
```

Accuracy: 0.8360

Precision: 0.8780

We can see that the precision are 0.9047 and 0.8780 respectively. There exists a difference between the train accuracy and test accuracy that cannot be ignored. By definition, when training accuracy is higher than testing, then it will be an overfitting model. In essence, the model has learned particulars that help it perform better in training data that are not applicable to the larger data population and therefore result in worse performance. Moreover, the size of the whole data set is probably another reason for it.

Conclusion

In this project we have trained and evaluated several machine learning models to predict mortality caused by heart failure based on patients’ features.

The data set was analyzed and evaluated in order to find important data relationships and to select meaningful predictors for the model training. The data was prepared for the training, whereas numeric predictors were scaled. Different algorithms were used then to train the models and the best algorithm was identified.

Random Forest and boosted Tree showed the highest performance of all models in the 5-fold cross-validation and in the final performance check on the test set. The models ’ precision are 0.9047 and 0.8780 respectively The performance might be improved by using a larger training data set, 299 observations indeed present a modest data collection. More information would have been a profound base for training/cross-validation providing better and deeper learning. Presumably, several of the tried algorithms would show higher performance having sufficient data. The other tried algorithms showed lower performance and some of them showed a strong overfitting. The attempts to improve the performance and to reduce overfitting were fruitless, though.

The selection of algorithms used to train the models was chosen to include simple as well as more sophisticated methods of different types. For future work, a more accurate selection based on extensive literature research can be performed. Algorithms better tailored for small data sets might help to improve the prediction.