Heart Failure Prediction
A Data Mining Case Study Project

AQUINO, Kerwin Dominique B.
LAGAZO, John Louise E.
MANLAPIG, Ralph Miguel A.
TADEO, Lorenz Christian E.

4CSC

Asst. Prof. Donata Acula, PhD
CS ELEC 4C: Data Mining

May 16, 2023

Introduction

The "Heart Failure Clinical Data" dataset available on Kaggle provides comprehensive clinical information about patients diagnosed with heart failure. This dataset was originally compiled by Andrew V. Dluhoslavskyi and encompasses a wide range of attributes related to the health status, medical history, and lifestyle choices of individuals affected by heart failure. The dataset aims to support the development of predictive models and the exploration of factors contributing to heart failure.

Heart failure is a serious medical condition that occurs when the heart is unable to pump blood efficiently, leading to a reduced supply of oxygen to the body's organs. It affects millions of people worldwide and is a major cause of morbidity and mortality. Predicting heart failure occurrence and understanding the associated risk factors are crucial for early detection and intervention, improving patient outcomes, and managing healthcare resources effectively.

The dataset comprises a total of 299 instances, each represented by 13 features along with the target variable indicating whether the patient experienced a heart failure event or not. The features include demographic information, various clinical and laboratory test results, and lifestyle-related factors such as smoking and anemia.

Background

Heart failure is a significant medical condition characterized by the heart's inability to pump blood effectively, resulting in reduced oxygen supply to the body's organs. To better understand the risk factors and predictors of heart failure, the "Heart Failure Clinical Data" dataset was created by Andrew V. Dluhoslavskyi and made available on Kaggle.

This dataset includes comprehensive clinical information from 299 patients diagnosed with heart failure. It encompasses demographic data, medical history, and various clinical and laboratory test results. By analyzing this dataset, researchers and data scientists can gain insights into the factors contributing to heart failure and develop predictive models for early detection and prevention.

With 13 features and a binary classification task, this dataset enables the development of accurate prediction models to identify individuals at risk. By leveraging machine learning techniques, healthcare professionals can improve patient care, implement targeted prevention strategies, and allocate resources efficiently. The table below describes the 13 features.

Attribute	Description

Age	Age
Anaemia	Decrease of red blood cells or hemoglobin (boolean)
Creatinine Phosphokinase	Level of the CPK enzyme in the blood (mcg/L)
diabetes	If the patient has diabetes (boolean)
ejection_fraction	Percentage of blood leaving the heart at each contraction (percentage)
high_blood_pressure	If the patient has hypertension (boolean)
platelets	Platelets in the blood (kiloplatelets/mL)
serum_creatinine	Level of serum creatinine in the blood (mg/dL)
serum_sodium	Level of serum sodium in the blood (mEq/L)
sex	Woman or man (binary)
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)

Exploring the "Heart Failure Clinical Data" dataset has the potential to advance cardiovascular research, enhance risk assessment models, and support personalized medicine approaches in heart failure management. By collaborating and leveraging data analysis, we can work towards reducing the impact of heart failure on individuals and society.

Objectives

Exploring this dataset can offer valuable insights into the factors associated with heart failure and aid in the development of machine learning models for predictive analysis. Researchers, healthcare professionals, and data scientists can leverage this dataset to investigate potential risk factors, identify significant predictors, and build accurate models for predicting heart failure events.

In this notebook, we will conduct exploratory data analysis, visualize key relationships, preprocess the data, and apply machine learning algorithms to predict heart failure events

based on the available clinical data. By doing so, we hope to contribute to the understanding of heart failure and facilitate the development of effective strategies for its prevention and management.

Methodologies and Tools

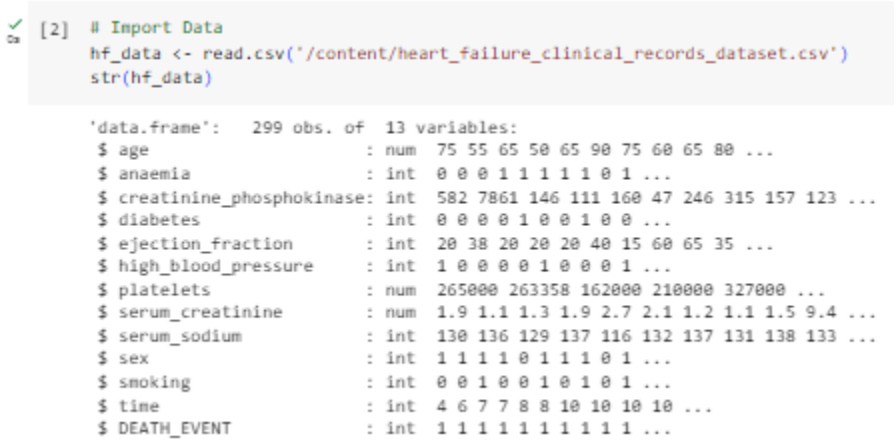
The dataset to be used in the study is the Heart Failure Prediction sourced from Kaggle, with the url of https://www.kaggle.com/datasets/andrewmvd/heart-failure-clinical-data. In order to perform the study, the proponents are to use data mining techniques namely data preprocessing, data discretization and binning, classification, and prediction. Google Colaboratory is to be used in this study, utilizing both Python and R languages. To accommodate the use of Python and R, per item, there shall be two Google Colaboratory files, one for Python and one for R.

Discussion

This section shows the data mining processes and their outputs using (1) Python, and (2) R. Included for each item is a summative narrative, discussing the results of a procedure. This section is divided into 4 major components pertaining to the different data mining techniques, (a) Data Preprocessing, (b) Data Discretization and Binning, (c) Classification, (d) Prediction.

Data Preprocessing

Figures 1.1 and 1.2 show the importing of the Heart Failure Clinical Records dataset to each of the Python and R dataframes, respectively.



```
[2]  # Import Data
     hf_data <- read.csv('/content/heart_failure_clinical_records_dataset.csv')
     str(hf_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 ...
```

Figure 1.1. Heart Failure Clinical Records Dataset in R.

```
[52] hf_data = pd.read_csv('/content/heart_failure_clinical_records_dataset _cleaned.csv')
     hf_data.head(5)
```

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time	DEATH_EVENT
0	75.0	0	582	0	20	1	265000.00	1.9	130	1	0	4	1
1	55.0	0	7861	0	38	0	263358.03	1.1	136	1	0	6	1
2	65.0	0	146	0	20	0	162000.00	1.3	129	1	1	7	1
3	50.0	1	111	0	20	0	210000.00	1.9	137	1	0	7	1
4	65.0	1	160	1	20	0	327000.00	2.7	116	0	0	8	1

```
hf_data.tail(5)
```

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time	DEATH_EVENT
294	62.0	0	61	1	38	1	155000.0	1.1	143	1	1	270	0
295	55.0	0	1820	0	38	0	270000.0	1.2	139	0	0	271	0
296	45.0	0	2060	1	60	0	742000.0	0.8	138	0	0	278	0
297	45.0	0	2413	0	38	0	140000.0	1.4	140	1	1	280	0
298	50.0	0	196	0	45	0	395000.0	1.6	136	1	1	285	0

Figure 1.2. Heart Failure Clinical Records Dataset in Python.

Figures 1.3 and 1.4 show if the dataset has any duplicates. Since the duplicate values are 0, there is no need to apply duplicate cleaning in the data frame.

```
[ ] # Check for duplicates
    get_dupes(hf_data)
```

```
No variable names specified - using all columns.

No duplicate combinations found of: age, anaemia, creatinine_phosphokinase, diabetes, ejection_fraction, high_blood_pressure, platelets, serum_creatinine, serum_sodium, ... and 4 other variables
```

A data.frame: 0 × 14

age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time	DEATH_EVENT	dupe_count
<dbl>	<int>	<int>	<int>	<int>	<int>	<dbl>	<dbl>	<int>	<int>	<int>	<int>	<int>	<int>

```
[ ] dim(hf_data)
```

```
299 · 13
```

Figure 1.3. Checking for Duplicates in the Dataset using R.

```
[54] # Double Check if data is clean
     hf_data.duplicated().sum()
```

```
0
```

Figure 1.4. Checking for Duplicates in the Dataset using Python.

Figures 1.5 and Figure 1.6 show that in order for the dataset to be fully cleaned, checking for null values is important. Since there are none, we can proceed to the next step.

```
[ ]  # Check for null values
     sum(is.na(hf_data))

     0

[ ]  # Since there are no null values, we can move forward.
```

Figure 1.5. Checking for Null Values in the Dataset using R.

```
[55] hf_data.isnull().sum()

age                         0
anaemia                     0
creatinine_phosphokinase    0
diabetes                    0
ejection_fraction           0
high_blood_pressure         0
platelets                   0
serum_creatinine            0
serum_sodium                0
sex                         0
smoking                     0
time                        0
DEATH_EVENT                 0
dtype: int64
```

Figure 1.6. Checking for Null Values in the Dataset using Python.

Data Discretization and Binning

Data discretization and Binning are techniques used to transform continuous numerical data into categorical data by dividing it into bins or intervals. These methods are often applied to simplify the analysis of data or to reduce the noise of the dataset. In choosing the columns to discretize, the group chose the top 3 columns with the highest range of values in the dataset. Figures 2.1 and 2.2 show the process of data discretization and binning using Python and R.

```
[ ]  # We are choosing to discretize these 3 columns as they have the highest range of values in the dataset
     age <- as.numeric(factor(hf_data2$age))
     ejection_fraction <- as.numeric(factor(hf_data2$ejection_fraction))
     platelets <- as.numeric(factor(hf_data2$platelets))
     time <- as.numeric(factor(hf_data2$time))

[ ]  age <- hf_data2[,"age"]
     ejection_fraction <- hf_data2[,"ejection_fraction"]
     platelets <- hf_data2[,"platelets"]
     time <- hf_data2[,"time"]

[ ]  age <- cut(age, "cluster", breaks = 3, include.lowest = TRUE, right = TRUE,labels = FALSE)
     ejection_fraction <- cut(ejection_fraction, "cluster", breaks = 3, include.lowest = TRUE, right = TRUE,labels = FALSE)
     platelets <- cut(platelets, "cluster", breaks = 4, include.lowest = TRUE, right = TRUE,labels = FALSE)
     time <- cut(time, "cluster", breaks = 3, include.lowest = TRUE, right = TRUE,labels = FALSE)
```

Figure 2.1 Discretizing Columns using R.

```
[ ] hf_data2 = hf_data
    le = LabelEncoder()
    hf_data2['age'] = le.fit_transform(hf_data2['age'].astype('str'))
    hf_data2['time'] = le.fit_transform(hf_data2['time'].astype('str'))
    hf_data2['ejection_fraction'] = le.fit_transform(hf_data2['ejection_fraction'].astype('str'))
    hf_data2['platelets'] = le.fit_transform(hf_data2['platelets'].astype('str'))

[ ] kmd= KBinsDiscretizer(n_bins = 3, encode = 'ordinal', strategy = 'kmeans')
    hf_data2[['age_kmd']] = kmd.fit_transform(np.array(hf_data2[['age']]))

[ ] kmd= KBinsDiscretizer(n_bins = 3, encode = 'ordinal', strategy = 'kmeans')
    hf_data2[['time_kmd']] = kmd.fit_transform(np.array(hf_data2[['time']]))

[ ] kmd= KBinsDiscretizer(n_bins = 3, encode = 'ordinal', strategy = 'kmeans')
    hf_data2[['ej_kmd']] = kmd.fit_transform(np.array(hf_data2[['ejection_fraction']]))

    kmd= KBinsDiscretizer(n_bins = 4, encode = 'ordinal', strategy = 'kmeans')
    hf_data2[['platelets_kmd']] = kmd.fit_transform(np.array(hf_data2[['platelets']]))
```

Figure 2.2 Discretizing Columns using Python.

For this dataset, the group transformed the chosen columns with the highest range of values in the dataset into numeral data as it provides an easy way to divide continuous features into discrete intervals or bins based on different strategies.

Classification

Decision Tree Classification

Decision tree classification is a machine learning algorithm used for both regression and classification tasks. It creates a tree-like model of decisions and their possible consequences. In the context of classification, a decision tree algorithm takes a dataset as input and splits it based on different features to classify the data into different classes.

- In R (Figures 3.1, 3.2, 3.3, 3.4, 3.5)

DT without Discretized Values

```
[ ] dt_model2 <- rpart(hf_data2$DEATH_EVENT ~ hf_data2$anaemia + hf_data2$creatinine_phosphokinase + hf_data2$diabetes
                                            + hf_data2$high_blood_pressure + hf_data2$serum_creatinine + hf_data2$serum_sodium
                                            + hf_data2$sex + hf_data2$smoking + hf_data2$time + hf_data2$age + hf_data2$ejection_fraction
                                            + hf_data2$platelets, data = train_td, method = 'class')
    summary(dt_model2)
```

```
Call:
rpart(formula = hf_data2$DEATH_EVENT ~ hf_data2$anaemia + hf_data2$creatinine_phosphokinase +
    hf_data2$diabetes + hf_data2$high_blood_pressure + hf_data2$serum_creatinine +
    hf_data2$serum_sodium + hf_data2$sex + hf_data2$smoking +
    hf_data2$time + hf_data2$age + hf_data2$ejection_fraction +
    hf_data2$platelets, data = train_td, method = "class")
  n= 299

          CP nsplit rel error    xerror       xstd
1 0.52083333      0 1.0000000 1.0000000 0.08409628
2 0.02604167      1 0.4791667 0.5104167 0.06667472
3 0.02083333      3 0.4270833 0.5937500 0.07075193
4 0.01041667      4 0.4062500 0.5729167 0.06978618
5 0.01000000      6 0.3854167 0.5937500 0.07075193

Variable importance
                    hf_data2$time         hf_data2$serum_creatinine
                               64                                14
                     hf_data2$age        hf_data2$ejection_fraction
                                6                                 6
            hf_data2$serum_sodium hf_data2$creatinine_phosphokinase
                                4                                 3
               hf_data2$platelets                  hf_data2$smoking
                                2                                 1

Node number 1: 299 observations,    complexity param=0.5208333
  predicted class=0  expected loss=0.3210702  P(node) =1
    class counts:   203    96
   probabilities: 0.679 0.321
  left son=2 (223 obs) right son=3 (76 obs)
  Primary splits:
      hf_data2$time             < 73.5   to the right, improve=52.56870, (0 missing)
      hf_data2$serum_creatinine  < 1.815  to the left,  improve=19.04558, (0 missing)
      hf_data2$ejection_fraction < 27.5   to the right, improve=15.33700, (0 missing)
```

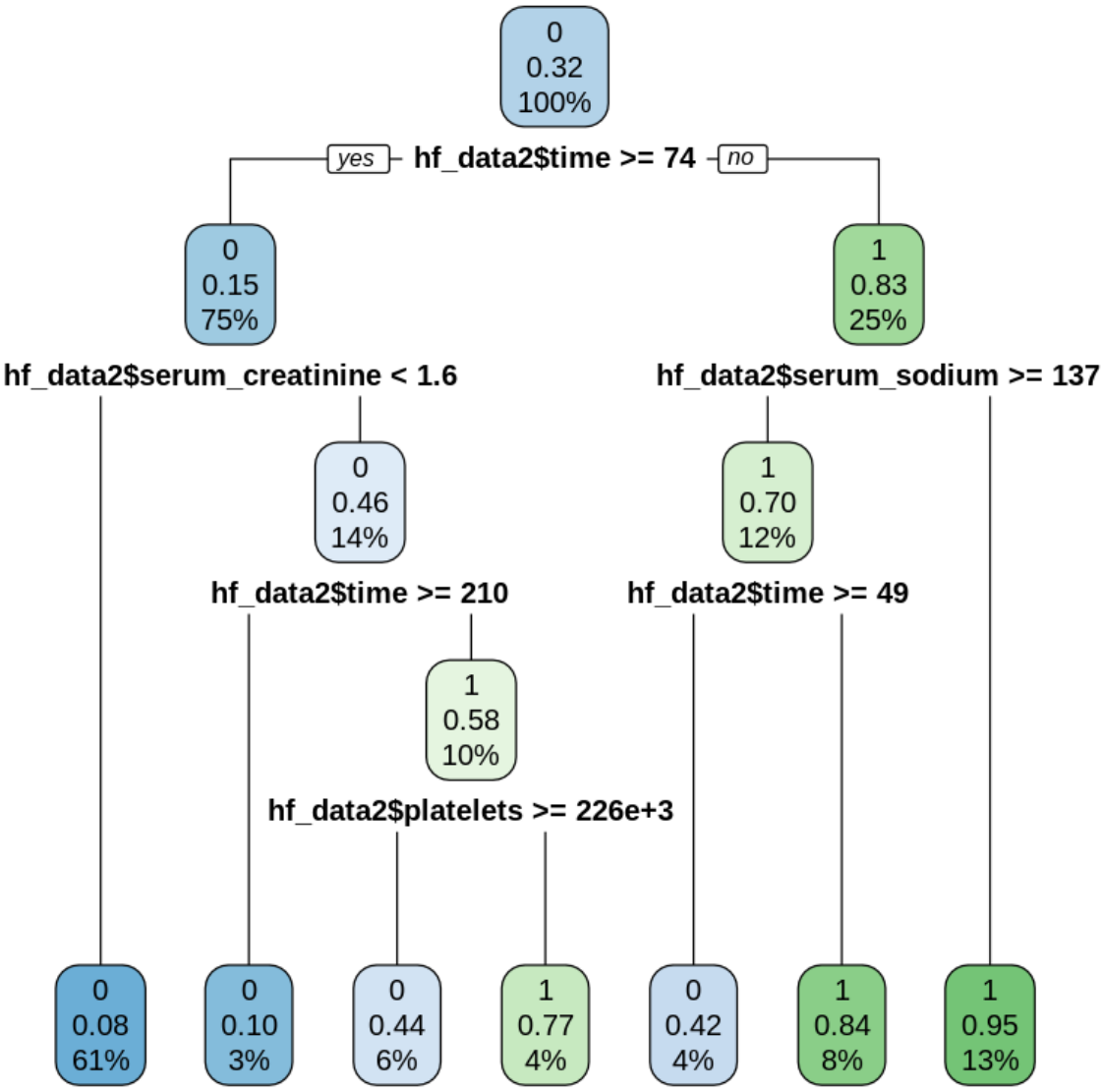


Figure 3.1. Decision Trees without Discretization using R

DT with Discretized Values

```
[ ] dt_model1 <- rpart(hf_data2$DEATH_EVENT ~ hf_data2$anaemia + hf_data2$creatinine_phosphokinase + hf_data2$diabetes
                                            + hf_data2$high_blood_pressure + hf_data2$serum_creatinine + hf_data2$serum_sodium
                                            + hf_data2$sex + hf_data2$smoking + hf_data2$disc_time + hf_data2$disc_age + hf_data2$disc_ej
                                            + hf_data2$disc_platelets, data = train_td, method = 'class')
    summary(dt_model1)
```

```
Call:
rpart(formula = hf_data2$DEATH_EVENT ~ hf_data2$anaemia + hf_data2$creatinine_phosphokinase +
    hf_data2$diabetes + hf_data2$high_blood_pressure + hf_data2$serum_creatinine +
    hf_data2$serum_sodium + hf_data2$sex + hf_data2$smoking +
    hf_data2$disc_time + hf_data2$disc_age + hf_data2$disc_ej +
    hf_data2$disc_platelets, data = train_td, method = "class")
  n= 299

          CP nsplit rel error    xerror       xstd
1 0.16666667      0 1.0000000 1.0000000 0.08409628
2 0.08854167      1 0.8333333 1.0416667 0.08498058
3 0.05208333      3 0.6562500 0.7604167 0.07737645
4 0.02604167      4 0.6041667 0.7604167 0.07737645
5 0.02083333      6 0.5520833 0.7916667 0.07842462
6 0.01000000      8 0.5104167 0.7708333 0.07773208

Variable importance
               hf_data2$disc_time         hf_data2$serum_creatinine
                               38                                28
                hf_data2$disc_age             hf_data2$serum_sodium
                               10                                 6
                 hf_data2$disc_ej hf_data2$creatinine_phosphokinase
                                6                                 5
     hf_data2$high_blood_pressure                 hf_data2$diabetes
                                5                                 1

Node number 1: 299 observations,    complexity param=0.1666667
  predicted class=0  expected loss=0.3210702  P(node) =1
    class counts:   203    96
   probabilities: 0.679 0.321
  left son=2 (173 obs) right son=3 (126 obs)
  Primary splits:
      hf_data2$disc_time        < 1.5   to the right, improve=25.595820, (0 missing)
      hf_data2$serum_creatinine < 1.815 to the left,  improve=19.045580, (0 missing)
      hf_data2$serum_sodium     < 135.5 to the right, improve= 7.939970, (0 missing)
      hf_data2$disc_age         < 2.5   to the left,  improve= 6.502066, (0 missing)
```

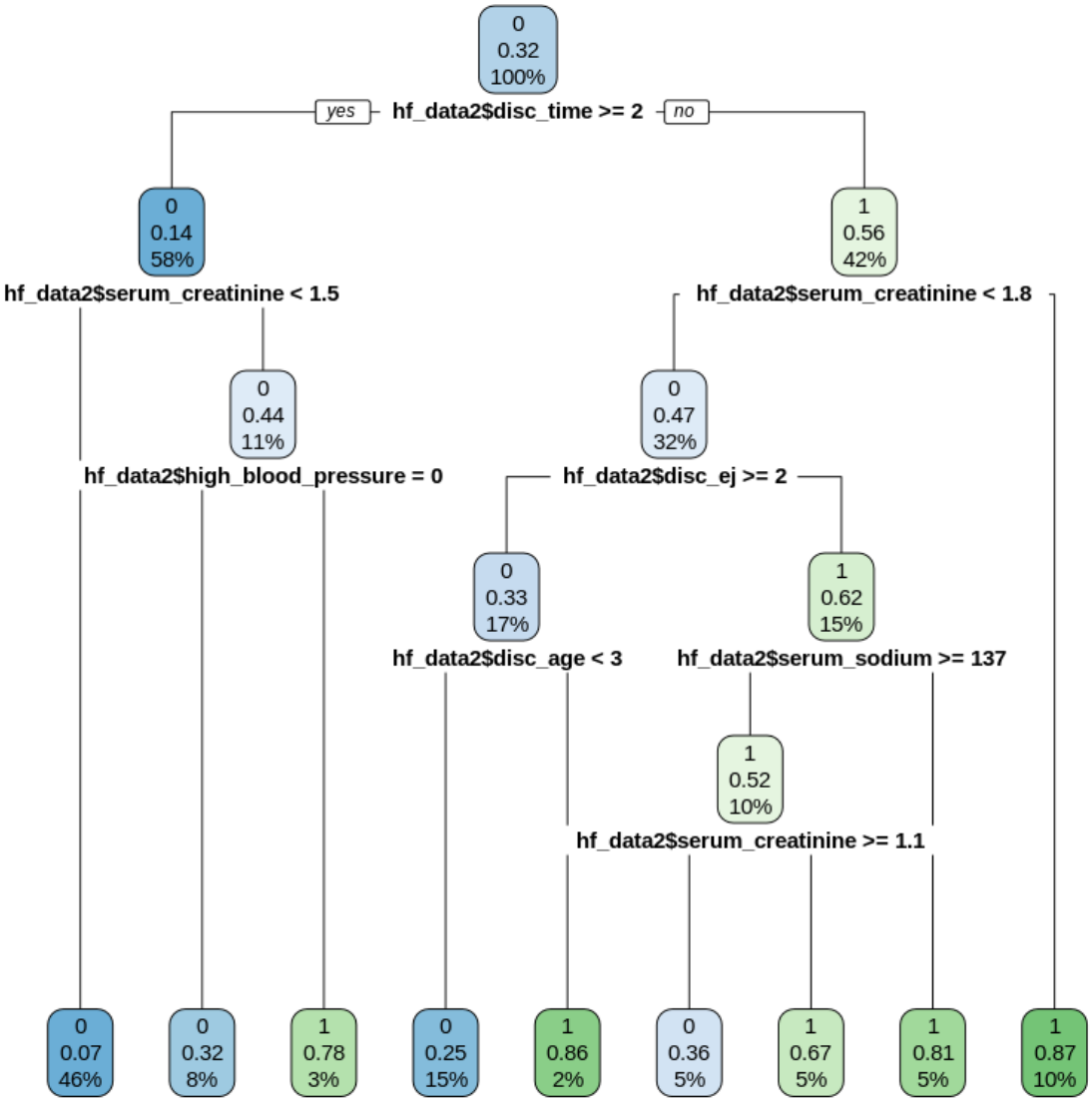


Figure 3.2. Decision Trees with Discretization using R

DT with all variables but with Discretized Platelets

```
[64] dt_model3 <- rpart(hf_data2$DEATH_EVENT ~ hf_data2$anaemia + hf_data2$creatinine_phosphokinase + hf_data2$diabetes
                                            + hf_data2$high_blood_pressure + hf_data2$serum_creatinine + hf_data2$serum_sodium
                                            + hf_data2$sex + hf_data2$smoking + hf_data2$time + hf_data2$age + hf_data2$ejection_fraction
                                            + hf_data2$disc_platelets, data = train_td, method = 'class')
     summary(dt_model3)
```

```
Call:
rpart(formula = hf_data2$DEATH_EVENT ~ hf_data2$anaemia + hf_data2$creatinine_phosphokinase +
    hf_data2$diabetes + hf_data2$high_blood_pressure + hf_data2$serum_creatinine +
    hf_data2$serum_sodium + hf_data2$sex + hf_data2$smoking +
    hf_data2$time + hf_data2$age + hf_data2$ejection_fraction +
    hf_data2$disc_platelets, data = train_td, method = "class")
  n= 299

          CP nsplit rel error    xerror       xstd
1 0.52083333      0 1.0000000 1.0000000 0.08409628
2 0.02604167      1 0.4791667 0.5000000 0.06612271
3 0.01041667      5 0.3645833 0.5520833 0.06878578
4 0.01000000      7 0.3437500 0.5520833 0.06878578

Variable importance
                    hf_data2$time         hf_data2$serum_creatinine
                               61                                13
                   hf_data2$age hf_data2$creatinine_phosphokinase
                                7                                 5
       hf_data2$ejection_fraction             hf_data2$serum_sodium
                                5                                 4
                hf_data2$diabetes                  hf_data2$smoking
                                4                                 1

Node number 1: 299 observations,    complexity param=0.5208333
  predicted class=0  expected loss=0.3210702  P(node) =1
    class counts:   203    96
   probabilities: 0.679 0.321
  left son=2 (223 obs) right son=3 (76 obs)
  Primary splits:
      hf_data2$time              < 73.5   to the right, improve=52.56870, (0 missing)
      hf_data2$serum_creatinine  < 1.815  to the left,  improve=19.04558, (0 missing)
      hf_data2$ejection_fraction < 27.5   to the right, improve=15.33700, (0 missing)
      hf_data2$age               < 71     to the left,  improve= 9.52658, (0 missing)
      hf_data2$serum_sodium      < 135.5  to the right, improve= 7.93997, (0 missing)
```

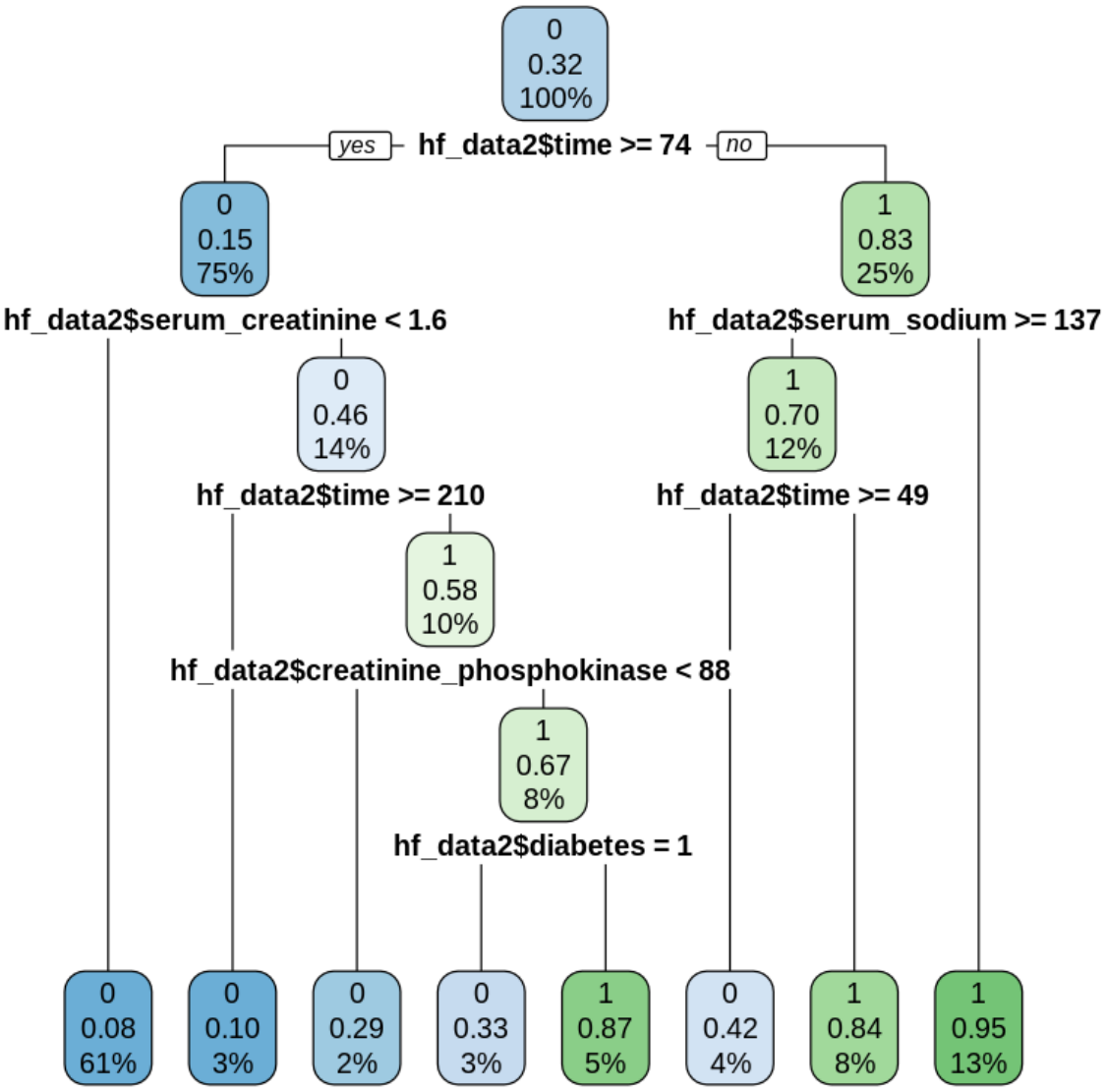


Figure 3.3. Decision Tree with all variables but with Discretized Platelets

DT with all variables but with Discretized Ejection Fraction

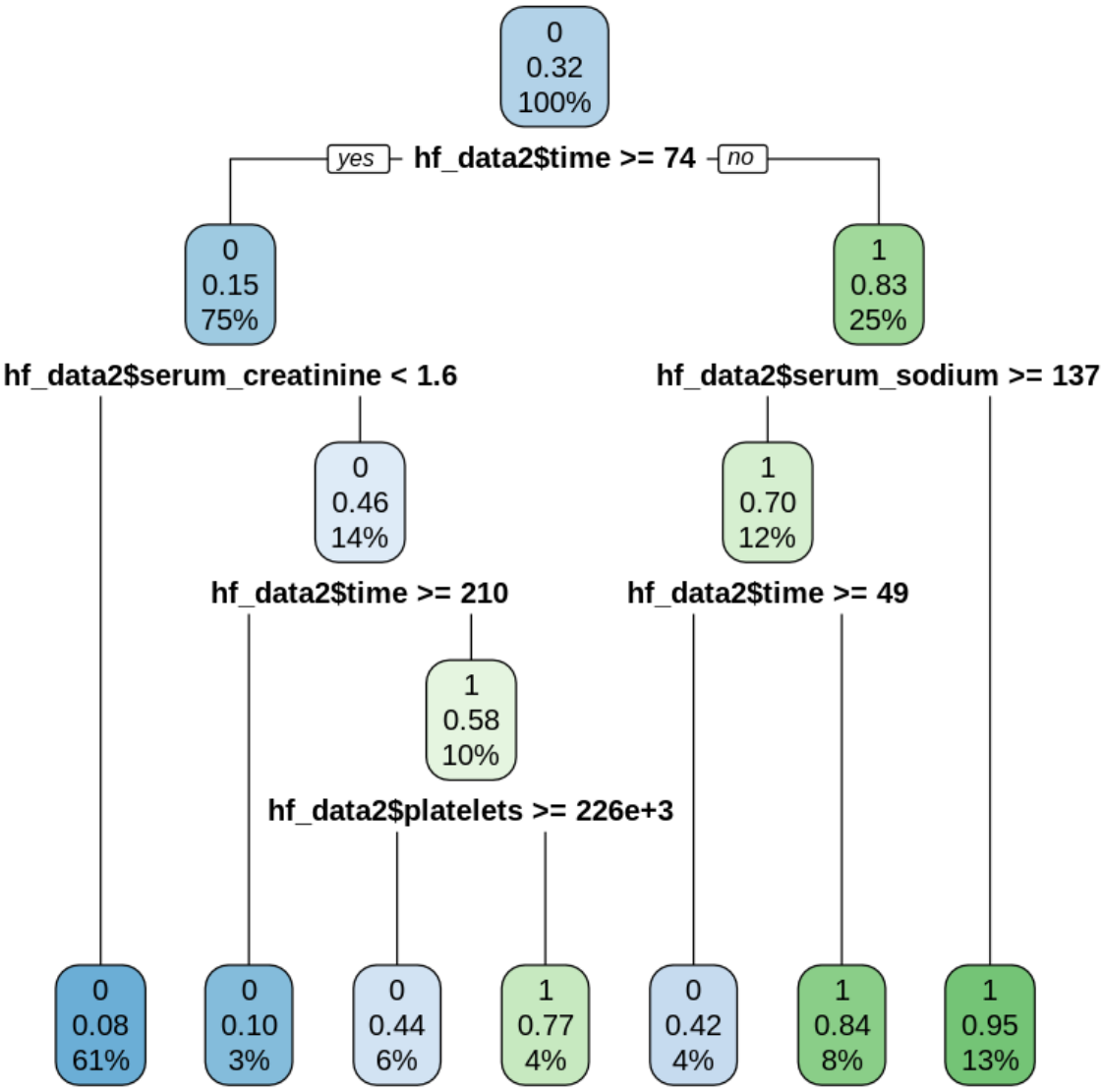
```
dt_model4 <- rpart(hf_data2$DEATH_EVENT ~ hf_data2$anaemia + hf_data2$creatinine_phosphokinase + hf_data2$diabetes
                                        + hf_data2$high_blood_pressure + hf_data2$serum_creatinine + hf_data2$serum_sodium
                                        + hf_data2$sex + hf_data2$smoking + hf_data2$time + hf_data2$age + hf_data2$disc_ej
                                        + hf_data2$platelets, data = train_td, method = 'class')
summary(dt_model4)
```

```
Call:
rpart(formula = hf_data2$DEATH_EVENT ~ hf_data2$anaemia + hf_data2$creatinine_phosphokinase +
    hf_data2$diabetes + hf_data2$high_blood_pressure + hf_data2$serum_creatinine +
    hf_data2$serum_sodium + hf_data2$sex + hf_data2$smoking +
    hf_data2$time + hf_data2$age + hf_data2$disc_ej + hf_data2$platelets,
    data = train_td, method = "class")
  n= 299

          CP nsplit rel error    xerror       xstd
1 0.52083333      0 1.0000000 1.0000000 0.08409628
2 0.02604167      1 0.4791667 0.5104167 0.06667472
3 0.02083333      3 0.4270833 0.5625000 0.06929040
4 0.01041667      4 0.4062500 0.5625000 0.06929040
5 0.01000000      6 0.3854167 0.5625000 0.06929040

Variable importance
                    hf_data2$time         hf_data2$serum_creatinine
                               68                                15
                     hf_data2$age             hf_data2$serum_sodium
                                6                                 4
hf_data2$creatinine_phosphokinase                hf_data2$platelets
                                3                                 2
                 hf_data2$smoking
                                1

Node number 1: 299 observations,    complexity param=0.5208333
  predicted class=0  expected loss=0.3210702  P(node) =1
    class counts:   203    96
   probabilities: 0.679 0.321
  left son=2 (223 obs) right son=3 (76 obs)
  Primary splits:
      hf_data2$time             < 73.5   to the right, improve=52.568700, (0 missing)
      hf_data2$serum_creatinine < 1.815  to the left,  improve=19.045580, (0 missing)
      hf_data2$age              < 71     to the left,  improve= 9.526580, (0 missing)
      hf_data2$serum_sodium     < 135.5  to the right, improve= 7.939970, (0 missing)
      hf_data2$disc_ej          < 1.5    to the right, improve= 4.129701, (0 missing)
```


0
0.32
100%
yes
hf_data2$time >= 74
no
0
0.15
75%
1
0.83
25%
hf_data2$serum_creatinine < 1.6
hf_data2$serum_sodium >= 137
0
0.46
14%
1
0.70
12%
hf_data2$time >= 210
hf_data2$time >= 49
1
0.58
10%
hf_data2$platelets >= 226e+3
0
0.08
61%
0
0.10
3%
0
0.44
6%
1
0.77
4%
0
0.42
4%
1
0.84
8%
1
0.95
13%

```
[70] #prediction
     dt_predict4 <- predict(dt_model4, test_td, type= 'class' )
     dt_cm4 <- table(hf_data2$DEATH_EVENT, dt_predict4)
     dt_cm4

     Warning message:
     "'newdata' had 79 rows but variables found have 299 rows"
        dt_predict4
           0   1
       0 194   9
       1  28  68

[71] dt_accuracy_m4 = sum(diag(dt_cm4)/sum(dt_cm4)) * 100
     dt_accuracy_m4

     87.6254180602007
```

Figure 3.4. Decision Trees with all variables but with Discretized Ejection Fraction

DT with all variables but with Discretized Time

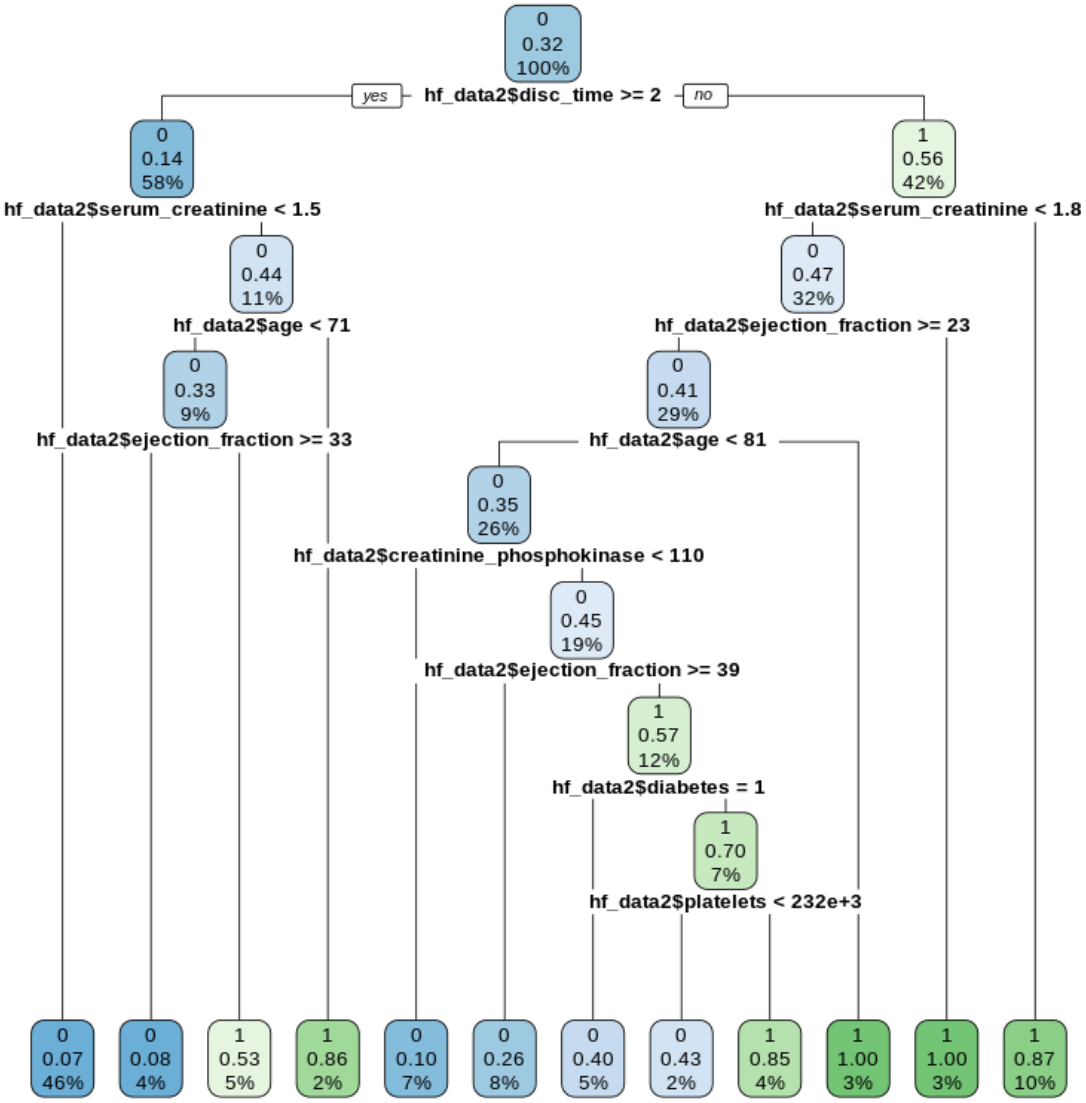
```
[72] dt_model5 <- rpart(hf_data2$DEATH_EVENT ~ hf_data2$anaemia + hf_data2$creatinine_phosphokinase + hf_data2$diabetes
                                            + hf_data2$high_blood_pressure + hf_data2$serum_creatinine + hf_data2$serum_sodium
                                            + hf_data2$sex + hf_data2$smoking + hf_data2$disc_time + hf_data2$age + hf_data2$ejection_fraction
                                            + hf_data2$platelets, data = train_td, method = 'class')
     summary(dt_model5)
```

```
Call:
rpart(formula = hf_data2$DEATH_EVENT ~ hf_data2$anaemia + hf_data2$creatinine_phosphokinase +
    hf_data2$diabetes + hf_data2$high_blood_pressure + hf_data2$serum_creatinine +
    hf_data2$serum_sodium + hf_data2$sex + hf_data2$smoking +
    hf_data2$disc_time + hf_data2$age + hf_data2$ejection_fraction +
    hf_data2$platelets, data = train_td, method = "class")
  n= 299

          CP nsplit rel error    xerror       xstd
1 0.16666667      0 1.0000000 1.0000000 0.08409628
2 0.07812500      1 0.8333333 1.0000000 0.08409628
3 0.02604167      4 0.5937500 0.8229167 0.07941784
4 0.01041667      9 0.4583333 0.8333333 0.07973706
5 0.01000000     11 0.4375000 0.8333333 0.07973706

Variable importance
               hf_data2$disc_time         hf_data2$serum_creatinine
                               30                                22
                    hf_data2$age        hf_data2$ejection_fraction
                               15                                15
hf_data2$creatinine_phosphokinase             hf_data2$serum_sodium
                                8                                 4
               hf_data2$platelets                 hf_data2$diabetes
                                3                                 2
    hf_data2$high_blood_pressure                  hf_data2$anaemia
                                1                                 1

Node number 1: 299 observations,    complexity param=0.1666667
  predicted class=0  expected loss=0.3210702  P(node) =1
    class counts:   203    96
   probabilities: 0.679 0.321
  left son=2 (173 obs) right son=3 (126 obs)
  Primary splits:
      hf_data2$disc_time         < 1.5    to the right, improve=25.59582, (0 missing)
      hf_data2$serum_creatinine  < 1.815  to the left,  improve=19.04558, (0 missing)
      hf_data2$ejection_fraction < 27.5   to the right, improve=15.33700, (0 missing)
```


0
0.32
100%
yes
hf_data2$disc_time >= 2
no
0
0.14
58%
1
0.56
42%
hf_data2$serum_creatinine < 1.5
hf_data2$serum_creatinine < 1.8
0
0.44
11%
0
0.47
32%
hf_data2$age < 71
hf_data2$ejection_fraction >= 23
0
0.33
9%
0
0.41
29%
hf_data2$ejection_fraction >= 33
hf_data2$age < 81
0
0.35
26%
hf_data2$creatinine_phosphokinase < 110
0
0.45
19%
hf_data2$ejection_fraction >= 39
1
0.57
12%
hf_data2$diabetes = 1
1
0.70
7%
hf_data2$platelets < 232e+3
0
0.07
46%
0
0.08
4%
1
0.53
5%
1
0.86
2%
0
0.10
7%
0
0.26
8%
0
0.40
5%
0
0.43
2%
1
0.85
4%
1
1.00
3%
1
1.00
3%
1
0.87
10%

```
[74] #prediction
     dt_predict5 <- predict(dt_model5, test_td, type= 'class' )
     dt_cm5 <- table(hf_data2$DEATH_EVENT, dt_predict5)
     dt_cm5

     Warning message:
     "'newdata' had 79 rows but variables found have 299 rows"
        dt_predict5
           0   1
       0 189  14
       1  28  68

[75] dt_accuracy_m5 = sum(diag(dt_cm5)/sum(dt_cm5)) * 100
     dt_accuracy_m5

     85.9531772575251
```

Figure 3.5. Decision Trees with all variables but with Discretized Time

- In Python (Figures 3.6, 3.7, 3.8, 3.9, 3.10)
    - In the case of the Decision Trees classifier without the utilization of discretized values, we observed that the resulting tree was larger than anticipated. Consequently, we proceeded to assess the optimization approach to enhance its comprehensibility. Our optimization efforts encompassed a comprehensive review and adjustment of various parameters to ensure that the decision tree was optimized for ease of interpretation and understanding.

DT without Discretization

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(hf_data2.iloc[:, [0,1,2,3,4,5,6,7,8,9,10,11]], hf_data2.iloc[:, 12], test_size = 0.3, random_state = 50)
```

```
[ ]  model_dt = DecisionTreeClassifier()
     model_dt.fit(X_train, y_train)
     y_pred = model_dt.predict(X_test)
     # y_pred
```

```
[ ]  cm_dt = confusion_matrix(y_test, y_pred)
     cm_dt
```

```
array([[43, 17],
       [ 9, 21]])
```

```
[ ]  accuracy_dt = (accuracy_score(y_test, y_pred)) * 100
     accuracy_dt
```

```
71.11111111111111
```

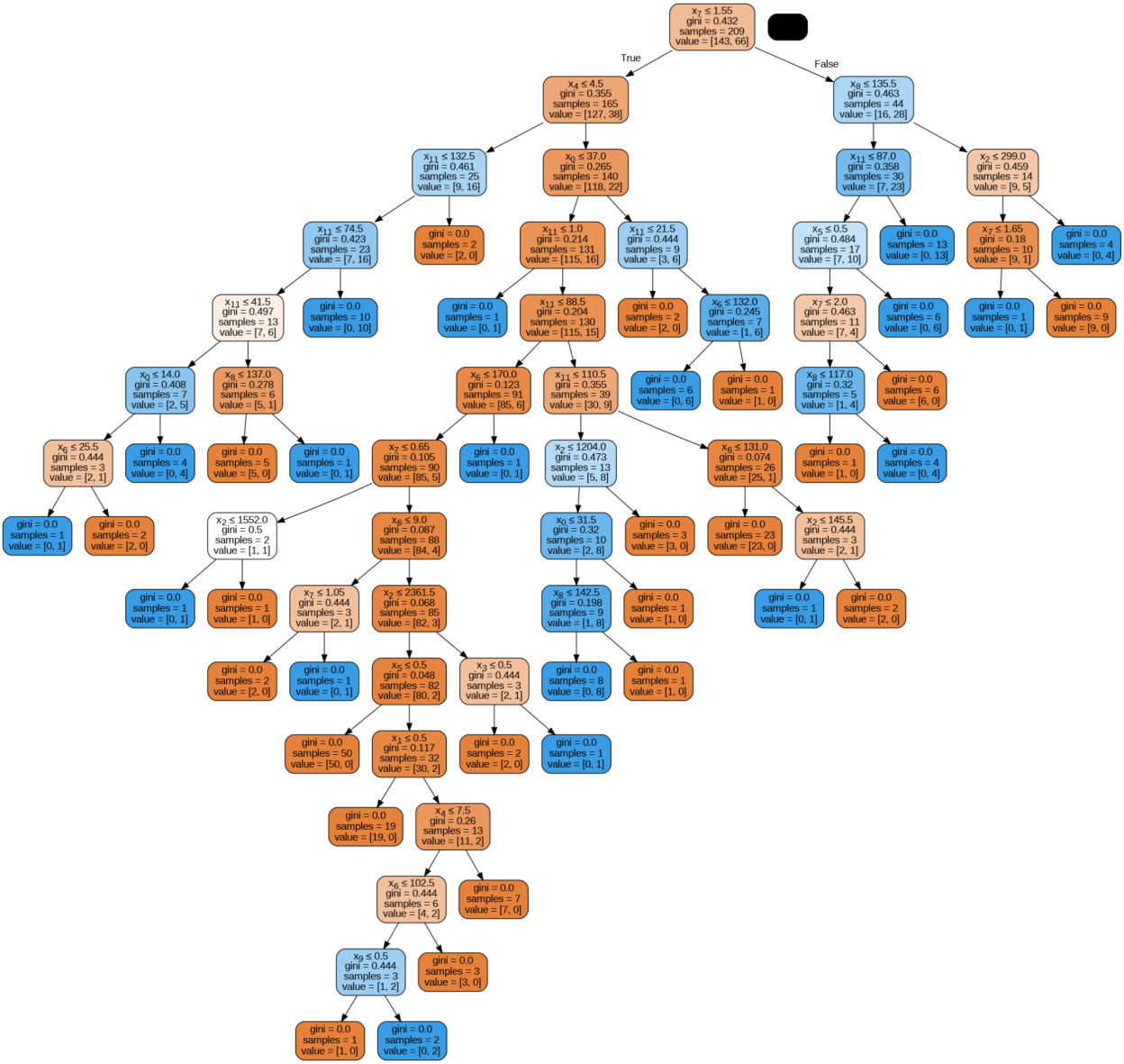


Figure 3.6. Decision tree without Discretization using Python

```
[ ]  #Optimization
     model_dt1 = DecisionTreeClassifier(criterion = 'entropy', max_depth=4)
     model_dt1.fit(pd.get_dummies(X_train), y_train)
     y_pred1 = model_dt.predict(pd.get_dummies(X_test))
     (accuracy_score(y_test, y_pred1))*100

     71.11111111111111
```

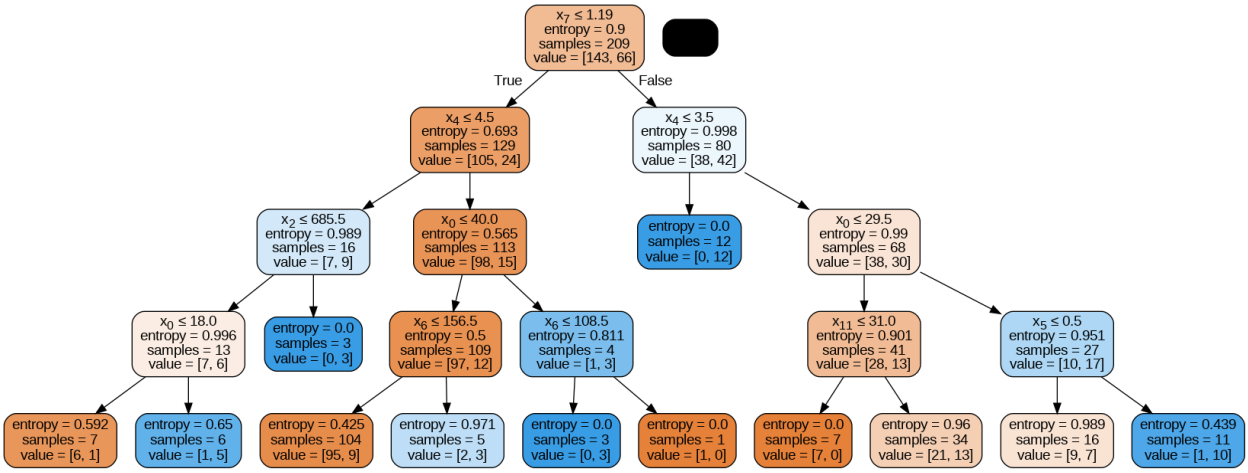


Figure 3.7. Optimized Decision tree without Discretization using Python

▾ DT with Discretized Values

```
[ ] hf_data2.head()
```

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time	DEATH_EVENT	age_kmd	time_kmd	ej_kmd	platelets_kmd
0	34	0	582	0	3	1	86	1.9	130	1	0	104	1	2.0	2.0	0.0	1.0
1	15	0	7861	0	7	0	84	1.1	136	1	0	115	1	0.0	2.0	1.0	1.0
2	26	0	146	0	3	0	18	1.3	129	1	1	124	1	1.0	2.0	0.0	0.0
3	10	1	111	0	3	0	41	1.9	137	1	0	124	1	0.0	2.0	0.0	0.0
4	26	1	160	1	3	0	126	2.7	116	0	0	134	1	1.0	2.0	0.0	2.0

```
[ ] X_train, X_test, y_train, y_test = train_test_split(hf_data2.iloc[:, [1,2,3,5,7,8,9,10,13,14,15,16]], hf_data2.iloc[:, 12], test_size = 0.3, random_state = 50)
```

```
[ ] #Optimization
    model_dt2 = DecisionTreeClassifier(criterion = 'entropy', max_depth=4)
    model_dt2.fit(pd.get_dummies(X_train), y_train)
    y_pred2 = model_dt2.predict(pd.get_dummies(X_test))
    (accuracy_score(y_test, y_pred2))*100

    72.22222222222221
```

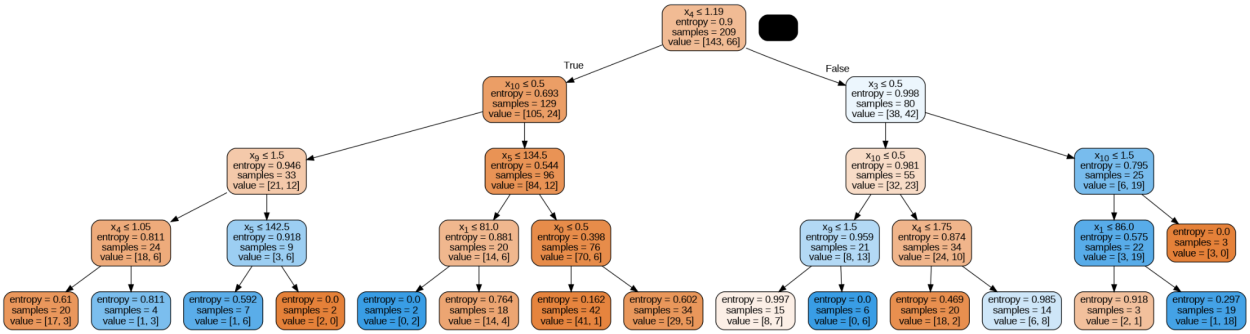


Figure 3.8. Optimized Decision tree with Discretization using Python

▾ DT with all variables but with Discretized Platelets

```
[ ] X_train, X_test, y_train, y_test = train_test_split(hf_data2.iloc[:, [0,1,2,3,4,5,7,8,9,10,11,16]], hf_data2.iloc[:, 12], test_size = 0.3, random_state = 50)
```

```
[ ] #Optimization
    model_dt3 = DecisionTreeClassifier(criterion = 'entropy', max_depth=4)
    model_dt3.fit(pd.get_dummies(X_train), y_train)
    y_pred3 = model_dt3.predict(pd.get_dummies(X_test))
    (accuracy_score(y_test, y_pred3))*100
```

```
72.22222222222221
```

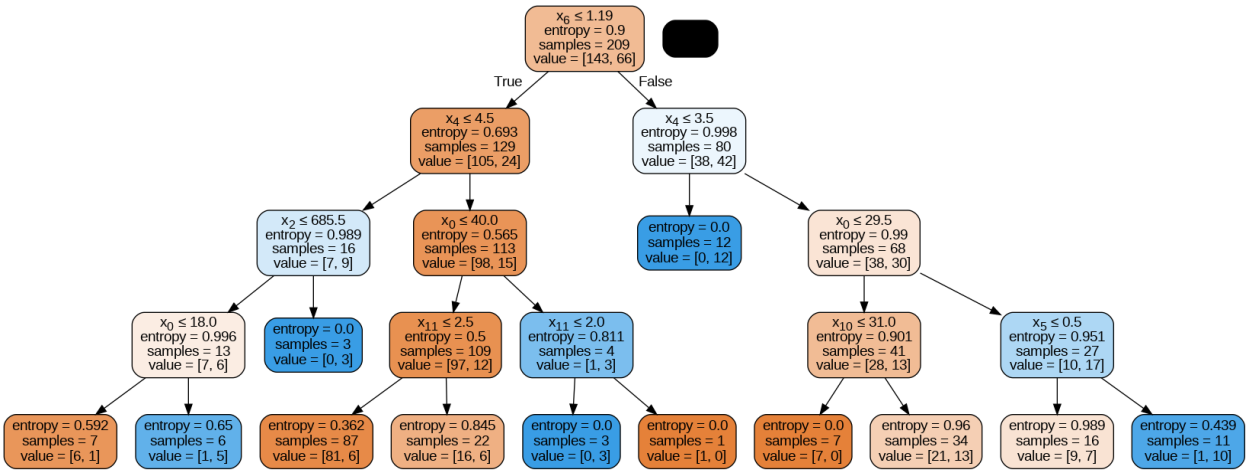


Figure 3.9. Optimized Decision tree with all variables but with Discretized Platelets using Python

▾ DT with all variables but with Discretized Time

```
[ ] X_train, X_test, y_train, y_test = train_test_split(hf_data2.iloc[:, [0,1,2,3,4,5,6,7,8,9,10,14]], hf_data2.iloc[:, 12], test_size = 0.3, random_state = 50)
```

```
[ ] #Optimization
    model_dt4 = DecisionTreeClassifier(criterion = 'entropy', max_depth=4)
    model_dt4.fit(pd.get_dummies(X_train), y_train)
    y_pred4 = model_dt4.predict(pd.get_dummies(X_test))
    (accuracy_score(y_test, y_pred4))*100
```

```
71.11111111111111
```

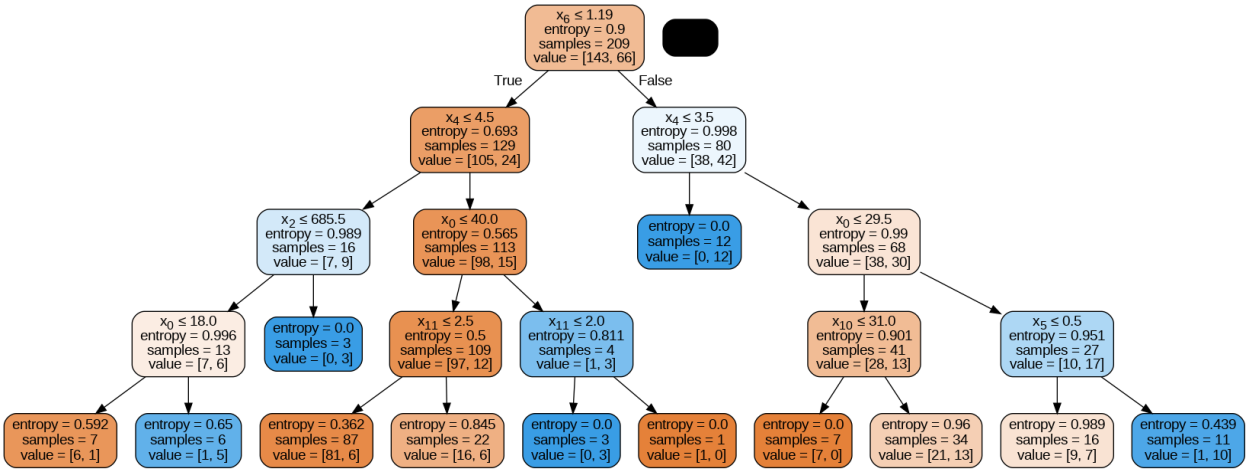


Figure 3.10. Optimized Decision tree with all variables but with Discretized Time using Python

OneR Classification

OneR (One Rule) is a simple and interpretable classification algorithm often used as a baseline for comparison with more complex machine learning algorithms. It aims to find a single rule based on one feature that provides the best accuracy in classifying the data.

Figure 4.1. shows the use of OneR classification without discretized values. While Figure 4.2 shows the use of said classification technique with discretized values. These two classifications will be the baseline of the case study's comparison for OneR.

OneR without Discretized Values

```
hf_1R_1 <- OneR(hf_data2$DEATH_EVENT ~ hf_data2$anaemia + hf_data2$creatinine_phosphokinase + hf_data2$diabetes
                                     + hf_data2$high_blood_pressure + hf_data2$serum_creatinine + hf_data2$serum_sodium
                                     + hf_data2$sex + hf_data2$smoking + hf_data2$time + hf_data2$age + hf_data2$ejection_fraction
                                     + hf_data2$platelets)
summary(hf_1R_1)
```

```
Call:
OneR.formula(formula = hf_data2$DEATH_EVENT ~ hf_data2$anaemia +
    hf_data2$creatinine_phosphokinase + hf_data2$diabetes + hf_data2$high_blood_pressure +
    hf_data2$serum_creatinine + hf_data2$serum_sodium + hf_data2$sex +
    hf_data2$smoking + hf_data2$time + hf_data2$age + hf_data2$ejection_fraction +
    hf_data2$platelets)

Rules:
If hf_data2$time = (3.72,60.2] then hf_data2$DEATH_EVENT = 1
If hf_data2$time = (60.2,116]  then hf_data2$DEATH_EVENT = 0
If hf_data2$time = (116,173]   then hf_data2$DEATH_EVENT = 0
If hf_data2$time = (173,229]   then hf_data2$DEATH_EVENT = 0
If hf_data2$time = (229,285]   then hf_data2$DEATH_EVENT = 0

Accuracy:
248 of 299 instances classified correctly (82.94%)

Contingency table:
                    hf_data2$time
hf_data2$DEATH_EVENT (3.72,60.2] (60.2,116] (116,173] (173,229] (229,285] Sum
                   0            9       * 66      * 27      * 61      * 40 203
                   1          * 54         22        11         7         2  96
                   Sum          63         88        38        68        42 299
---
Maximum in each column: '*'

Pearson's Chi-squared test:
X-squared = 114.52, df = 4, p-value < 2.2e-16
```

Figure 4.1. OneR Classification Technique without Discretized Values

▾ OneR with Discretized Values

```
[42] hf_1R_2 <- OneR(hf_data2$DEATH_EVENT ~ hf_data2$anaemia + hf_data2$creatinine_phosphokinase + hf_data2$diabetes
                                          + hf_data2$high_blood_pressure + hf_data2$serum_creatinine + hf_data2$serum_sodium
                                          + hf_data2$sex + hf_data2$smoking + hf_data2$disc_time + hf_data2$disc_age + hf_data2$disc_ej
                                          + hf_data2$disc_platelets)
     summary(hf_1R_2)
```

```
Call:
OneR.formula(formula = hf_data2$DEATH_EVENT ~ hf_data2$anaemia +
    hf_data2$creatinine_phosphokinase + hf_data2$diabetes + hf_data2$high_blood_pressure +
    hf_data2$serum_creatinine + hf_data2$serum_sodium + hf_data2$sex +
    hf_data2$smoking + hf_data2$disc_time + hf_data2$disc_age +
    hf_data2$disc_ej + hf_data2$disc_platelets)

Rules:
If hf_data2$disc_time = 1 then hf_data2$DEATH_EVENT = 1
If hf_data2$disc_time = 2 then hf_data2$DEATH_EVENT = 0
If hf_data2$disc_time = 3 then hf_data2$DEATH_EVENT = 0

Accuracy:
219 of 299 instances classified correctly (73.24%)

Contingency table:
                    hf_data2$disc_time
hf_data2$DEATH_EVENT    1    2    3 Sum
                   0     55 * 67 * 81 203
                   1   * 71   18    7  96
                   Sum  126   85   88 299
---
Maximum in each column: '*'

Pearson's Chi-squared test:
X-squared = 62.178, df = 2, p-value = 3.15e-14
```

Figure 4.2. OneR Classification Technique with Discretized Values

Figure 4.3. shows the use of the OneR Classification Technique with all variables but with Discretized Platelets. In addition, Figure 4.4. shows the use of the said classification technique with all variables but with Discretized Ejection Fraction.

OneR with all variables but with Discretized Platelets

```
[43] hf_1R_3 <- OneR(hf_data2$DEATH_EVENT ~ hf_data2$anaemia + hf_data2$creatinine_phosphokinase + hf_data2$diabetes
                                          + hf_data2$high_blood_pressure + hf_data2$serum_creatinine + hf_data2$serum_sodium
                                          + hf_data2$sex + hf_data2$smoking + hf_data2$time + hf_data2$age + hf_data2$ejection_fraction
                                          + hf_data2$disc_platelets)
     summary(hf_1R_3)
```

```
Call:
OneR.formula(formula = hf_data2$DEATH_EVENT ~ hf_data2$anaemia +
    hf_data2$creatinine_phosphokinase + hf_data2$diabetes + hf_data2$high_blood_pressure +
    hf_data2$serum_creatinine + hf_data2$serum_sodium + hf_data2$sex +
    hf_data2$smoking + hf_data2$time + hf_data2$age + hf_data2$ejection_fraction +
    hf_data2$disc_platelets)

Rules:
If hf_data2$time = (3.72,60.2] then hf_data2$DEATH_EVENT = 1
If hf_data2$time = (60.2,116]  then hf_data2$DEATH_EVENT = 0
If hf_data2$time = (116,173]   then hf_data2$DEATH_EVENT = 0
If hf_data2$time = (173,229]   then hf_data2$DEATH_EVENT = 0
If hf_data2$time = (229,285]   then hf_data2$DEATH_EVENT = 0

Accuracy:
248 of 299 instances classified correctly (82.94%)

Contingency table:
                     hf_data2$time
hf_data2$DEATH_EVENT (3.72,60.2] (60.2,116] (116,173] (173,229] (229,285] Sum
                   0            9       * 66       * 27       * 61       * 40 203
                   1          * 54         22         11          7          2  96
                   Sum          63         88         38         68         42 299
---
Maximum in each column: '*'

Pearson's Chi-squared test:
X-squared = 114.52, df = 4, p-value < 2.2e-16
```

Figure 4.3. OneR Classification with all variables but with Discretized Platelets

OneR with all variables but with Discretized Ejection Fraction

```
hf_1R_4 <- OneR(hf_data2$DEATH_EVENT ~ hf_data2$anaemia + hf_data2$creatinine_phosphokinase + hf_data2$diabetes
                                     + hf_data2$high_blood_pressure + hf_data2$serum_creatinine + hf_data2$serum_sodium
                                     + hf_data2$sex + hf_data2$smoking + hf_data2$time + hf_data2$age + hf_data2$disc_ej
                                     + hf_data2$platelets)
summary(hf_1R_4)
```

```
Call:
OneR.formula(formula = hf_data2$DEATH_EVENT ~ hf_data2$anaemia +
    hf_data2$creatinine_phosphokinase + hf_data2$diabetes + hf_data2$high_blood_pressure +
    hf_data2$serum_creatinine + hf_data2$serum_sodium + hf_data2$sex +
    hf_data2$smoking + hf_data2$time + hf_data2$age + hf_data2$disc_ej +
    hf_data2$platelets)

Rules:
If hf_data2$time = (3.72,60.2] then hf_data2$DEATH_EVENT = 1
If hf_data2$time = (60.2,116]  then hf_data2$DEATH_EVENT = 0
If hf_data2$time = (116,173]   then hf_data2$DEATH_EVENT = 0
If hf_data2$time = (173,229]   then hf_data2$DEATH_EVENT = 0
If hf_data2$time = (229,285]   then hf_data2$DEATH_EVENT = 0

Accuracy:
248 of 299 instances classified correctly (82.94%)

Contingency table:
                     hf_data2$time
hf_data2$DEATH_EVENT (3.72,60.2] (60.2,116] (116,173] (173,229] (229,285] Sum
                   0            9       * 66       * 27       * 61       * 40 203
                   1          * 54         22         11          7          2  96
                   Sum          63         88         38         68         42 299
---
Maximum in each column: '*'

Pearson's Chi-squared test:
X-squared = 114.52, df = 4, p-value < 2.2e-16
```

Figure 4.4. OneR Classification with all variables but with Discretized Ejection Fraction

Lastly, Figure 4.5 shows the use of the said technique but with Discretized Time.

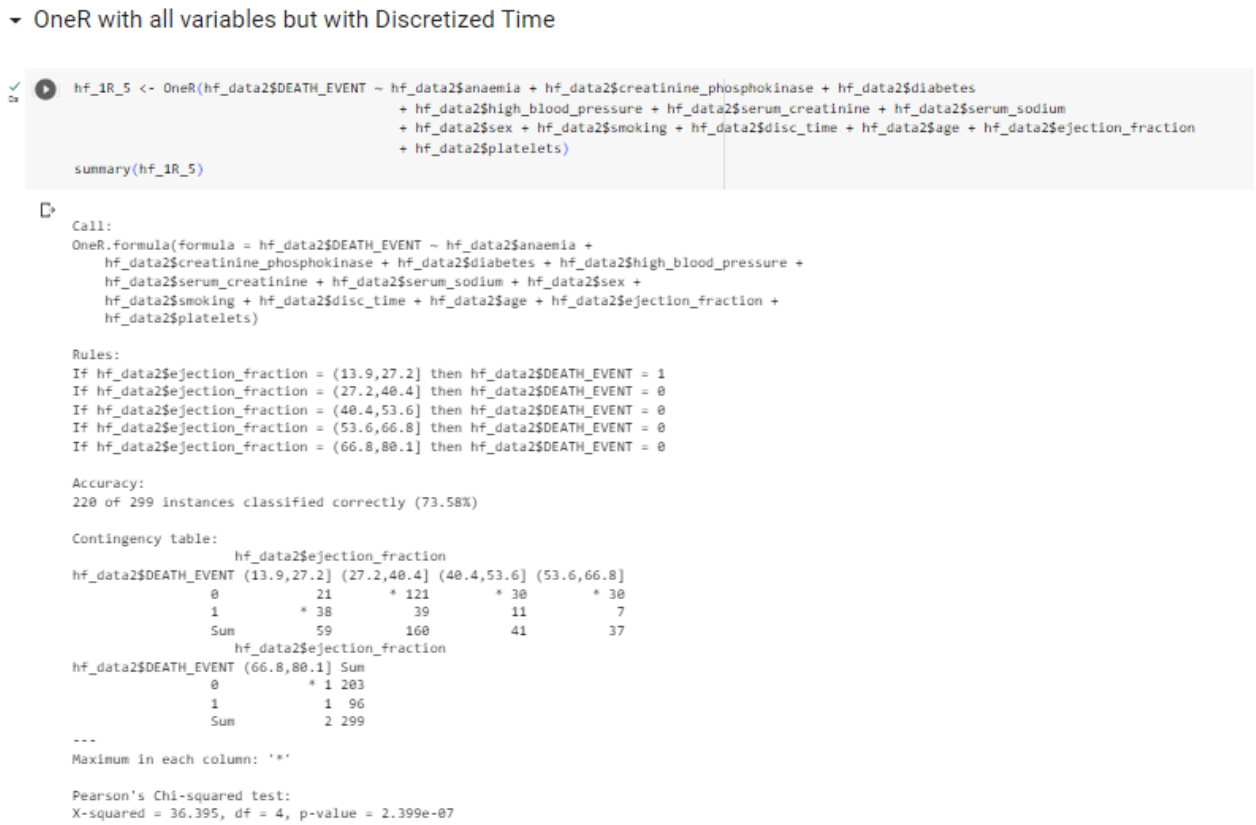


Figure 4.5. OneR Classification with all variables but with Discretized Time

Naive Bayes Prediction

Naive Bayes is a popular algorithm used for classification tasks, particularly in natural language processing. It is based on Bayes' theorem and assumes that the features are independent of each other. For R, our hypothesis draws parallels to classification techniques, proposing that the utilization of discretized values would enhance accuracy. However, if accuracy does not improve, we attribute this outcome to the discretization of time. In our study, we exclusively employed discretized platelets to achieve more accurate results, akin to the approach employed in OneR Classification.

In R, it is imperative to demonstrate the consistency of values. Our hypothesis postulates that the limited size of the dataset restricts the availability of additional testing data, thereby hindering the attainment of a more representative accuracy. This discrepancy in accuracy arises from the implementation of the Naive Bayes algorithm in Python, where NB1, NB3, and NB4

yield an accuracy of 71.111%, while NB2 exhibits an accuracy of 70%. We contend that the variance in accuracy can be attributed to the insufficiency of testing data due to the relatively small number of entries within our model.

- Naive Bayes Classification
    a. Using R (Figures 5.1, 5.2, 5.3, 5.4)

Naive Bayes without Discretization

```
[ ]  str(hf_data2)
     str(test_td)
```

```
'data.frame':   299 obs. of  17 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 ...
 $ disc_age                : int  2 1 2 1 2 3 2 2 2 3 ...
 $ disc_ej                 : int  1 2 1 1 1 2 1 3 3 1 ...
 $ disc_platelets          : int  2 2 1 1 2 1 1 3 2 2 ...
 $ disc_time               : int  1 1 1 1 1 1 1 1 1 1 ...
'data.frame':   79 obs. of  17 variables:
 $ age                     : num  45 80 94 85 50 50 69 82 70 51 ...
 $ anaemia                 : int  1 0 0 0 1 1 0 1 0 0 ...
 $ creatinine_phosphokinase: int  981 148 582 23 249 159 582 855 582 1380 ...
 $ diabetes                : int  0 1 1 0 1 1 1 1 0 0 ...
 $ ejection_fraction       : int  30 38 38 45 35 30 35 50 20 25 ...
 $ high_blood_pressure     : int  0 0 1 0 1 0 0 1 1 1 ...
 $ platelets               : num  136000 149000 263358 360000 319000 ...
 $ serum_creatinine        : num  1.1 1.9 1.83 3 1 1.2 3.5 1 1.83 0.9 ...
 $ serum_sodium            : int  137 144 134 132 128 138 134 145 134 130 ...
 $ sex                     : int  1 1 1 1 0 0 1 0 1 1 ...
 $ smoking                 : int  0 1 0 0 0 0 0 0 1 0 ...
 $ time                    : int  11 23 27 28 28 29 30 30 31 38 ...
 $ DEATH_EVENT             : int  1 1 1 1 1 0 1 1 1 1 ...
 $ disc_age                : int  1 3 3 3 1 1 2 3 2 1 ...
 $ disc_ej                 : int  1 2 2 2 1 1 1 2 1 1 ...
```

Figure 5.1. Naive Bayes Prediction without Discretization Using R

Naive Bayes with Discretization

```
[ ] nb2 <- naiveBayes(hf_data2$DEATH_EVENT ~ hf_data2$anaemia + hf_data2$creatinine_phosphokinase + hf_data2$diabetes
                                           + hf_data2$high_blood_pressure + hf_data2$serum_creatinine + hf_data2$serum_sodium
                                           + hf_data2$sex + hf_data2$smoking + hf_data2$disc_time + hf_data2$disc_age + hf_data2$disc_ej
                                           + hf_data2$disc_platelets,data = train_td, method = 'class')
    summary(nb2)
```

```
          Length Class  Mode
apriori    2     table  numeric
tables    12     -none- list
levels     2     -none- character
isnumeric 12     -none- logical
call       5     -none- call
```

```
[ ] nb2_pred <- predict(nb2, test_td, type ='class')
    nb2_pred
```

```
Warning message in predict.naiveBayes(nb2, test_td, type = "class"):
“Type mismatch between training and new data for variable 'hf_data2$anaemia'. Did you use factors with numeric labels for training, and numeric values for new data?”
Warning message in predict.naiveBayes(nb2, test_td, type = "class"):
“Type mismatch between training and new data for variable 'hf_data2$creatinine_phosphokinase'. Did you use factors with numeric labels for training, and numeric values for new data?”
Warning message in predict.naiveBayes(nb2, test_td, type = "class"):
“Type mismatch between training and new data for variable 'hf_data2$diabetes'. Did you use factors with numeric labels for training, and numeric values for new data?”
Warning message in predict.naiveBayes(nb2, test_td, type = "class"):
“Type mismatch between training and new data for variable 'hf_data2$high_blood_pressure'. Did you use factors with numeric labels for training, and numeric values for new data?”
Warning message in predict.naiveBayes(nb2, test_td, type = "class"):
“Type mismatch between training and new data for variable 'hf_data2$serum_creatinine'. Did you use factors with numeric labels for training, and numeric values for new data?”
Warning message in predict.naiveBayes(nb2, test_td, type = "class"):
“Type mismatch between training and new data for variable 'hf_data2$serum_sodium'. Did you use factors with numeric labels for training, and numeric values for new data?”
Warning message in predict.naiveBayes(nb2, test_td, type = "class"):
“Type mismatch between training and new data for variable 'hf_data2$sex'. Did you use factors with numeric labels for training, and numeric values for new data?”
Warning message in predict.naiveBayes(nb2, test_td, type = "class"):
“Type mismatch between training and new data for variable 'hf_data2$smoking'. Did you use factors with numeric labels for training, and numeric values for new data?”
Warning message in predict.naiveBayes(nb2, test_td, type = "class"):
“Type mismatch between training and new data for variable 'hf_data2$disc_time'. Did you use factors with numeric labels for training, and numeric values for new data?”
```

Figure 5.2. Naive Bayes Prediction with Discretization Using R

Naive Bayes with all variables but with Discretized Platelets

```
[ ] nb3 <- naiveBayes(hf_data2$DEATH_EVENT ~ hf_data2$anaemia + hf_data2$creatinine_phosphokinase + hf_data2$diabetes
                                           + hf_data2$high_blood_pressure + hf_data2$serum_creatinine + hf_data2$serum_sodium
                                           + hf_data2$sex + hf_data2$smoking + hf_data2$time + hf_data2$age + hf_data2$ejection_fraction
                                           + hf_data2$disc_platelets,data = train_td, method = 'class')
    summary(nb3)
```

```
          Length Class  Mode
apriori    2     table  numeric
tables    12     -none- list
levels     2     -none- character
isnumeric 12     -none- logical
call       5     -none- call
```

```
[ ] nb3_pred <- predict(nb3, test_td, type ='class')
    nb3_pred
```

```
Warning message in predict.naiveBayes(nb3, test_td, type = "class"):
“Type mismatch between training and new data for variable 'hf_data2$anaemia'. Did you use factors with numeric labels for training, and numeric values for new data?”
Warning message in predict.naiveBayes(nb3, test_td, type = "class"):
“Type mismatch between training and new data for variable 'hf_data2$creatinine_phosphokinase'. Did you use factors with numeric labels for training, and numeric values for new data?”
Warning message in predict.naiveBayes(nb3, test_td, type = "class"):
“Type mismatch between training and new data for variable 'hf_data2$diabetes'. Did you use factors with numeric labels for training, and numeric values for new data?”
Warning message in predict.naiveBayes(nb3, test_td, type = "class"):
“Type mismatch between training and new data for variable 'hf_data2$high_blood_pressure'. Did you use factors with numeric labels for training, and numeric values for new data?”
Warning message in predict.naiveBayes(nb3, test_td, type = "class"):
“Type mismatch between training and new data for variable 'hf_data2$serum_creatinine'. Did you use factors with numeric labels for training, and numeric values for new data?”
Warning message in predict.naiveBayes(nb3, test_td, type = "class"):
“Type mismatch between training and new data for variable 'hf_data2$serum_sodium'. Did you use factors with numeric labels for training, and numeric values for new data?”
Warning message in predict.naiveBayes(nb3, test_td, type = "class"):
“Type mismatch between training and new data for variable 'hf_data2$sex'. Did you use factors with numeric labels for training, and numeric values for new data?”
Warning message in predict.naiveBayes(nb3, test_td, type = "class"):
“Type mismatch between training and new data for variable 'hf_data2$smoking'. Did you use factors with numeric labels for training, and numeric values for new data?”
Warning message in predict.naiveBayes(nb3, test_td, type = "class"):
“Type mismatch between training and new data for variable 'hf_data2$time'. Did you use factors with numeric labels for training, and numeric values for new data?”
```

Figure 5.3. Naive Bayes Prediction with all variables but with Discretized Platelets Using R

Naive Bayes with all variables but with Discretized Time

```
[ ] nb4 <- naiveBayes(hf_data2$DEATH_EVENT ~ hf_data2$anaemia + hf_data2$creatinine_phosphokinase + hf_data2$diabetes
                                           + hf_data2$high_blood_pressure + hf_data2$serum_creatinine + hf_data2$serum_sodium
                                           + hf_data2$sex + hf_data2$smoking + hf_data2$disc_time + hf_data2$age + hf_data2$ejection_fraction
                                           + hf_data2$platelets,data = train_td, method = 'class')
    summary(nb4)

            Length Class  Mode
apriori     2      table  numeric
tables     12      -none- list
levels      2      -none- character
isnumeric  12      -none- logical
call        5      -none- call

[ ] nb4_pred <- predict(nb4, test_td, type ='class')
    nb4_pred

    Warning message in predict.naiveBayes(nb4, test_td, type = "class"):
    "Type mismatch between training and new data for variable 'hf_data2$anaemia'. Did you use factors with numeric labels for training, and numeric values for new data?"
    Warning message in predict.naiveBayes(nb4, test_td, type = "class"):
    "Type mismatch between training and new data for variable 'hf_data2$creatinine_phosphokinase'. Did you use factors with numeric labels for training, and numeric values for new data?"
    Warning message in predict.naiveBayes(nb4, test_td, type = "class"):
    "Type mismatch between training and new data for variable 'hf_data2$diabetes'. Did you use factors with numeric labels for training, and numeric values for new data?"
    Warning message in predict.naiveBayes(nb4, test_td, type = "class"):
    "Type mismatch between training and new data for variable 'hf_data2$high_blood_pressure'. Did you use factors with numeric labels for training, and numeric values for new data?"
    Warning message in predict.naiveBayes(nb4, test_td, type = "class"):
    "Type mismatch between training and new data for variable 'hf_data2$serum_creatinine'. Did you use factors with numeric labels for training, and numeric values for new data?"
    Warning message in predict.naiveBayes(nb4, test_td, type = "class"):
    "Type mismatch between training and new data for variable 'hf_data2$serum_sodium'. Did you use factors with numeric labels for training, and numeric values for new data?"
    Warning message in predict.naiveBayes(nb4, test_td, type = "class"):
    "Type mismatch between training and new data for variable 'hf_data2$sex'. Did you use factors with numeric labels for training, and numeric values for new data?"
    Warning message in predict.naiveBayes(nb4, test_td, type = "class"):
    "Type mismatch between training and new data for variable 'hf_data2$smoking'. Did you use factors with numeric labels for training, and numeric values for new data?"
    Warning message in predict.naiveBayes(nb4, test_td, type = "class"):
    "Type mismatch between training and new data for variable 'hf_data2$disc_time'. Did you use factors with numeric labels for training, and numeric values for new data?"
```

Figure 5.4. Naive Bayes Prediction with all variables but with Discretized Time Using R

b. Using Python (Figures 5.5, 5.6, 5.7, 5.8)

Naive Bayes without Discretization

```
[ ] X_train, X_test, y_train, y_test = train_test_split(hf_data2.iloc[:, [0,1,2,3,4,5,6,7,8,9,10,11]], hf_data2.iloc[:, 12], test_size = 0.3, random_state = 50)

[ ] model_nb1 = GaussianNB()
    model_nb1.fit(X_train, y_train)
    y_pred_nb1 = model_nb1.predict(X_test)

[ ] cm_nb1 = confusion_matrix(y_test, y_pred_nb1)
    cm_nb1

    array([[57,  3],
           [23,  7]])

[ ] accuracy_nb1 = (accuracy_score(y_test, y_pred_nb1)) * 100
    accuracy_nb1

    71.11111111111111
```

Figure 5.5. Naive Bayes Prediction without Discretization Using Python

Naive Bayes with Discretized Values

```
[ ] X_train, X_test, y_train, y_test = train_test_split(hf_data2.iloc[:, [1,2,3,5,7,8,9,10,13,14,15,16]], hf_data2.iloc[:, 12], test_size = 0.3, random_state = 50)

[ ] model_nb2 = GaussianNB()
    model_nb2.fit(X_train, y_train)
    y_pred_nb2 = model_nb2.predict(X_test)

[ ] cm_nb2 = confusion_matrix(y_test, y_pred_nb2)
    cm_nb2

    array([[57,  3],
           [24,  6]])

[ ] accuracy_nb2 = (accuracy_score(y_test, y_pred_nb2)) * 100
    accuracy_nb2

    70.0
```

Figure 5.6. Naive Bayes Prediction with Discretization Using Python

```
▾ Naive Bayes with all variables but with Discretized Platelets

[ ] X_train, X_test, y_train, y_test = train_test_split(hf_data2.iloc[:, [0,1,2,3,4,5,7,8,9,10,11,16]], hf_data2.iloc[:, 12], test_size = 0.3, random_state = 50)

[ ] model_nb3 = GaussianNB()
    model_nb3.fit(X_train, y_train)
    y_pred_nb3 = model_nb3.predict(X_test)

[ ] cm_nb3 = confusion_matrix(y_test, y_pred_nb3)
    cm_nb3

    array([[58,  2],
           [24,  6]])

[ ] accuracy_nb3 = (accuracy_score(y_test, y_pred_nb3)) * 100
    accuracy_nb3

    71.11111111111111
```

Figure 5.7. Naive Bayes Prediction with all variables but with Discretized Platelets Using Python

```
▾ Naive Bayes with all variabes but with Discretized Time

[ ] X_train, X_test, y_train, y_test = train_test_split(hf_data2.iloc[:, [0,1,2,3,4,5,6,7,8,9,10,14]], hf_data2.iloc[:, 12], test_size = 0.3, random_state = 50)

[ ] model_nb4 = GaussianNB()
    model_nb4.fit(X_train, y_train)
    y_pred_nb4 = model_nb4.predict(X_test)

[ ] cm_nb4 = confusion_matrix(y_test, y_pred_nb4)
    cm_nb4

    array([[58,  2],
           [24,  6]])

[ ] accuracy_nb4 = (accuracy_score(y_test, y_pred_nb4)) * 100
    accuracy_nb4

    71.11111111111111
```

Figure 5.8. Naive Bayes Prediction with all variables but with Discretized Time Using Python

Conclusion

The study used data mining techniques such as data preprocessing, data discretization and binning, classification, and prediction, in order to discover data in order to gather accuracy on each rule inside the model for predicting death events. The conclusion for this study are as follows:

- Classification with Discretized Values: The application of discretized values in the classification process yielded improved accuracy. By discretizing the data, the model was able to capture important patterns and relationships more effectively.
- Impact of Dataset Size: A limitation of the study was the relatively small size of the dataset, resulting in a restricted amount of testing data. This constraint could potentially

affect the generalizability of the accuracy results and hinder the ability to demonstrate a more robust performance.

- Decision Tree Optimization: Initially, when the Decision Trees classifier was applied without discretized values, the resulting tree was found to be larger than expected. To enhance understandability, an optimization process was undertaken. Various parameters were fine-tuned and adjusted to simplify the decision tree structure and improve interpretability.

Recommendations

To improve the results on the heart failure dataset, the following recommendations can be considered:

- Since the dataset size is relatively small, consider applying data augmentation techniques to artificially increase the number of samples. This can involve techniques such as oversampling, undersampling, or generating synthetic data points.
- Conducting a thorough analysis of feature importance and selecting the most relevant features for model training. Utilize techniques such as correlation analysis, feature importance from tree-based models, or recursive feature elimination to identify the most informative features.
- Experimenting with different classification models to identify the most suitable one for the heart failure prediction task. Additionally, perform hyperparameter tuning to find the optimal configuration for the chosen model. Grid Search or Bayesian optimization techniques can be used for this purpose.
- Employ cross-validation techniques, such as k-fold cross-validation, to assess the model's performance more reliably. Consider using appropriate evaluation metrics, such as accuracy, precision, recall, F1 score, depending on the specific requirements of the heart failure prediction problem.

Reference

Heart Prediction Failure Dataset from Kaggle

- https://kaggle.com/datasets/andrewmvd/heart-failure-clinical-data

Contribution Table:

NAME	CONTRIBUTION
AQUINO, Kerwin Dominique B.	● Data Discretization and Binning in Python ● Decision Trees Classification in Python ● Documentation ● Presentation
LAGAZO, John Louise E.	● OneR Classification in R ● Naive Bayes in R ● Documentation ● Presentation
MANLAPIG, Ralph Miguel A.	● Data Discretization and Binning in R ● Decision Trees Classification in R ● OneR Classification in R ● Documentation ● Presentation
TADEO, Lorenz Christian E.	● Decision Trees in Python ● Naive Bayes in Python ● Documentation ● Presentation