ProjectFirst

Teemu Sormunen, Abdullah Günay, Nicola Brazzale

11/18/2020

Contents

1 Introduction

In this project we analyse the data from Heart desease dataset [2]. We perform a Bayesian analysis of the data using this sequence of operations: • Overview of analysis problem and of the dataset • Data preprocsessing and visualisation • Prior choice discussion • Models used in our analysis • $\hat{R}$ convergence • HMC specific convergence diagnostics • Effective sample size diagnostic (n_eff or ESS) • Model comparison • Prior sensitivity analysis • Discussion and conclusion

1.1 The problem

Cardiovascular Heart Disease (CHD) is the top reason causing 31% of deaths globally. Pakistan is one of the countries where CHD is increasing significantly, and previous studies do not directly apply to Pakistani area due to different diet patterns. [2]

1.2 The motivation

With this project we aim to estimate death events and the major risk factors for heart failure with, possibly, high accuracy [2].

1.3 Modeling idea

We created 3 models which are then compared based on $\hat{r}$, n_{eff}, using the loo package and the classification accuracy. The 1st model is the reduced model and consists in fewer varibles which are selected base on their corrolection with the death event. The 2nd model consists in all varibles except for the varibale "time" as we believe that doens't represent an important factor in the death event scenario. The 3rd model used is a hierarchical model where we treated age class patients in a group with respect to the other selected variables. The 4th model is a non-linear model, as the hierarchical one we took in considertion only the selected varibales in the first model. Modeling is done with package brms, which is a interface for non-linear multivariate multilevel models in Stan.

2 Dataset

2.1 Term explanation

Some of the terms in the dataset might not be familiar, and they are opened briefly here.

- **Creatine phosphokinase (CPK)**
  CPK is an enzyme, which helps to regulate the concentration of adenosine triphosphate (ATP) in cells. ATP is responsible for carrying energy. If the CPK level is high, it often means that there has been an injury or stress on a muscle tissue. Although CPK is one the oldest markers of heart attack, high CPK might also indicate of acute muscle injury along with acute heart problems.
  Normal level of CPK ranges from 20 to 200 IU/L [5]
- **Ejection fraction (EF)**
  EF is a measurement in percentage which describes how much blood left ventricle pumps out of heart with each contraction. Low EF might indicate potential heart issues.
  Normal EF is 50 to 70 percent, while measurement under 40 percept might be an indicator of heart failure or cardiomyopathy. [1]
- **Platelets**
  Platelets are small cell fragments which can form clots. Too many platelets can lead to clotting of blood vessels, which in turn can lead to heart attack. Too Normal range of platelets is from 150 000 to 450 000. [4]
- **Serum creatinine**
  When creatine breaks down, it forms a waste product called creatinine. Kidneys normally remove

creatinine from body. Serum creatinine measures level of creatinine in the blood, indicating the kidney health. High levels of creatinine might indicate a kidney dysfunctioning.
Normal level of creatinine range from 0.9 to 1.3 mg/dL in men and 0.6 to 1.1 mg/dL in women who are 18 to 60 years old. [6]

- **Serum sodium**
  Serum sodium measures the amount of sodium in blood. Sodium enters blood through food and drink, and leaves by urine, stool and sweat. Too much sodium can cause blood pressure, while too little sodium can cause nausea, vomiting, exhaustion or dizziness.
  Normal levels of serum sodium are 135 to 145 mEq/L, according to Mayo Clinic. There are however different interpretations of "normal".[3]

2.2 Dataset introduction

The dataset of 299 patients was produced as a result of study [2] from Pakistani's city Faisalabad. All of the patients were over 40 years old, each having ventricular systolic dysfunction. This means that patient has poor left ventricular ejection fraction. The dataset has 105 women, and 194 men. EF, serum creatinine and platelets are categorical variables, and age, serum sodium and CPK are continuous variables.
Statistical analysis by [2] found age, creatinine, sodium, anemia and BP as significant variables.

3 Packages

Load data

```
file.name <- './data/heart_failure_clinical_records_dataset.csv'
heart <- read_csv(file.name)
```

```
## Parsed with column specification:
## cols(
##   age = col_double(),
##   anaemia = col_double(),
##   creatinine_phosphokinase = col_double(),
##   diabetes = col_double(),
##   ejection_fraction = col_double(),
##   high_blood_pressure = col_double(),
##   platelets = col_double(),
##   serum_creatinine = col_double(),
##   serum_sodium = col_double(),
##   sex = col_double(),
##   smoking = col_double(),
##   time = col_double(),
##   DEATH_EVENT = col_double()
## )
```

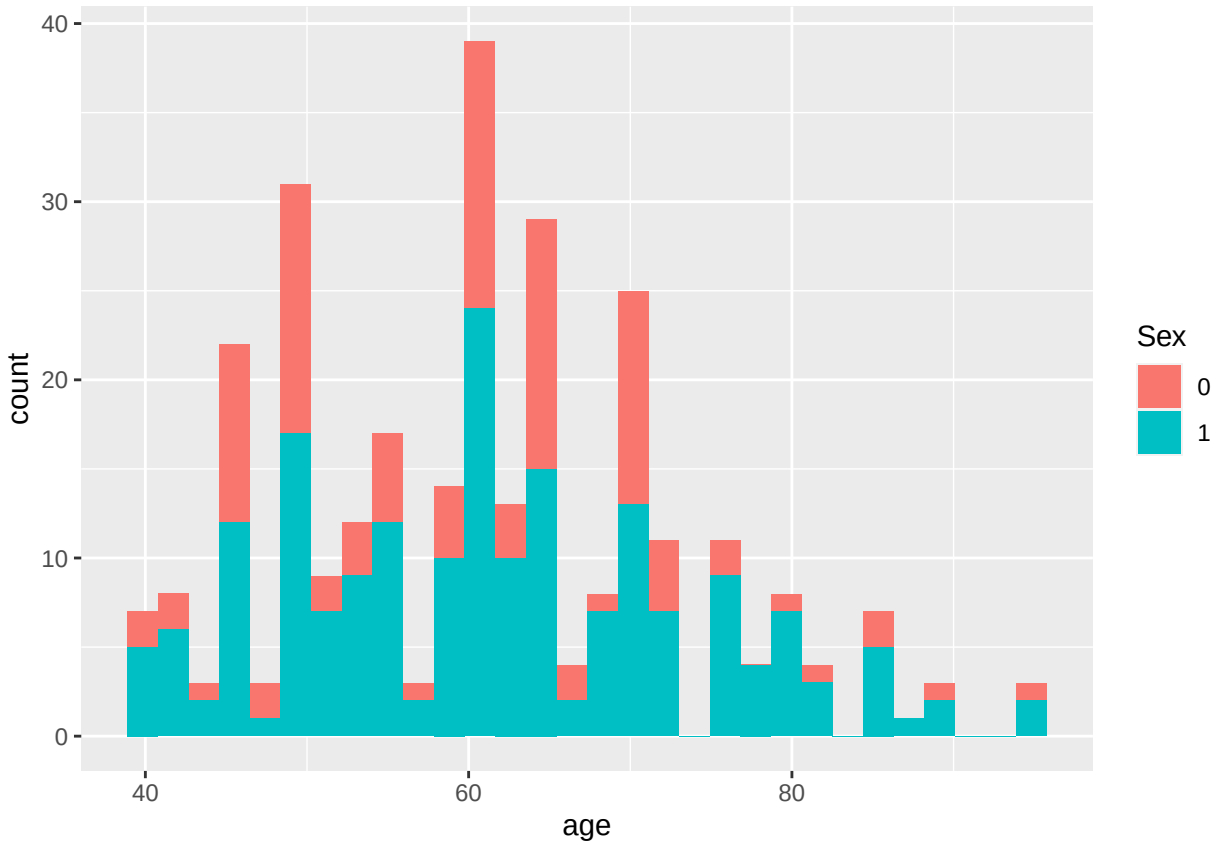
Prevent text overflow on PDF

```
library(knitr)
opts_chunk$set(tidy.opts=list(width.cutoff=60),tidy=TRUE)
```

4 Data preprocessing and visualization

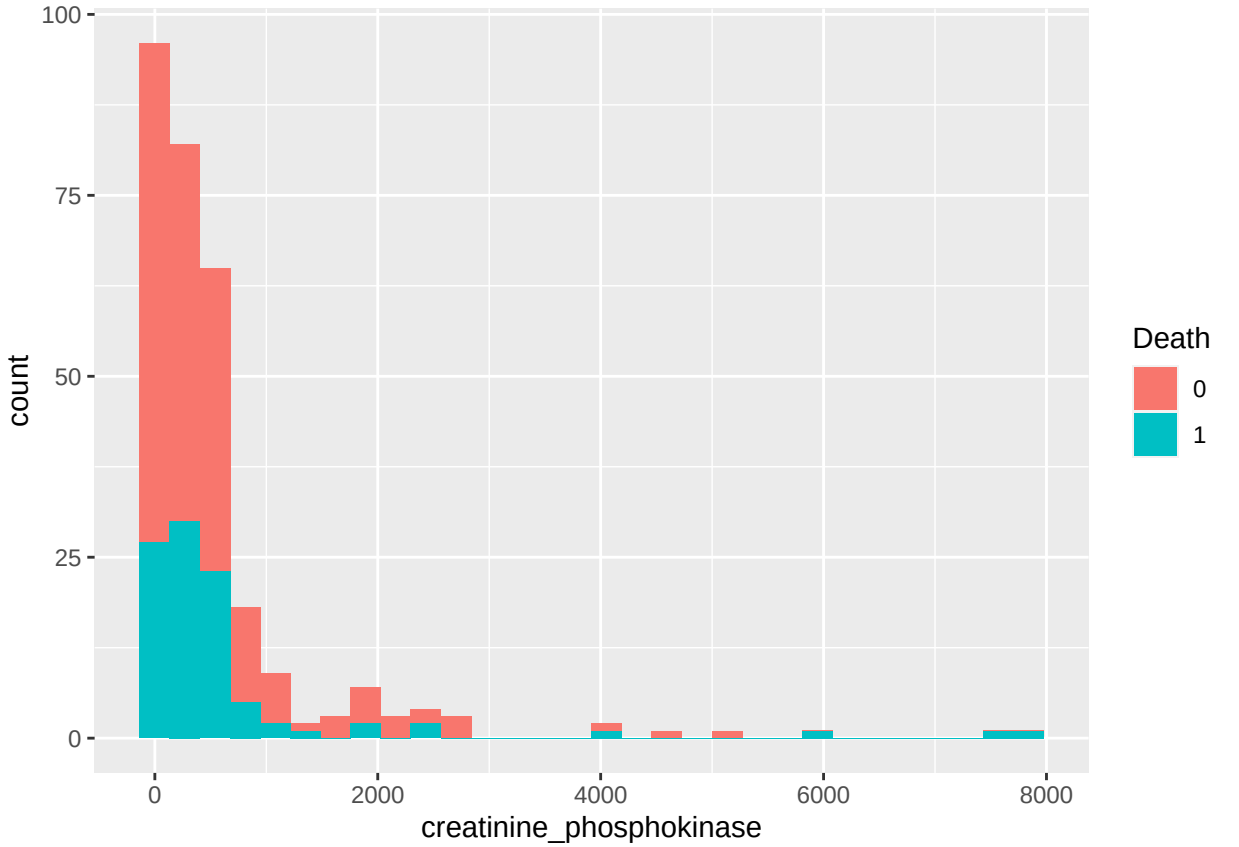
4.1 Plot histograms

```
ggplot(heart, aes(x = age)) + geom_histogram(aes(fill = as.character(sex)),
    bins = 30) + labs(fill = "Sex")
```

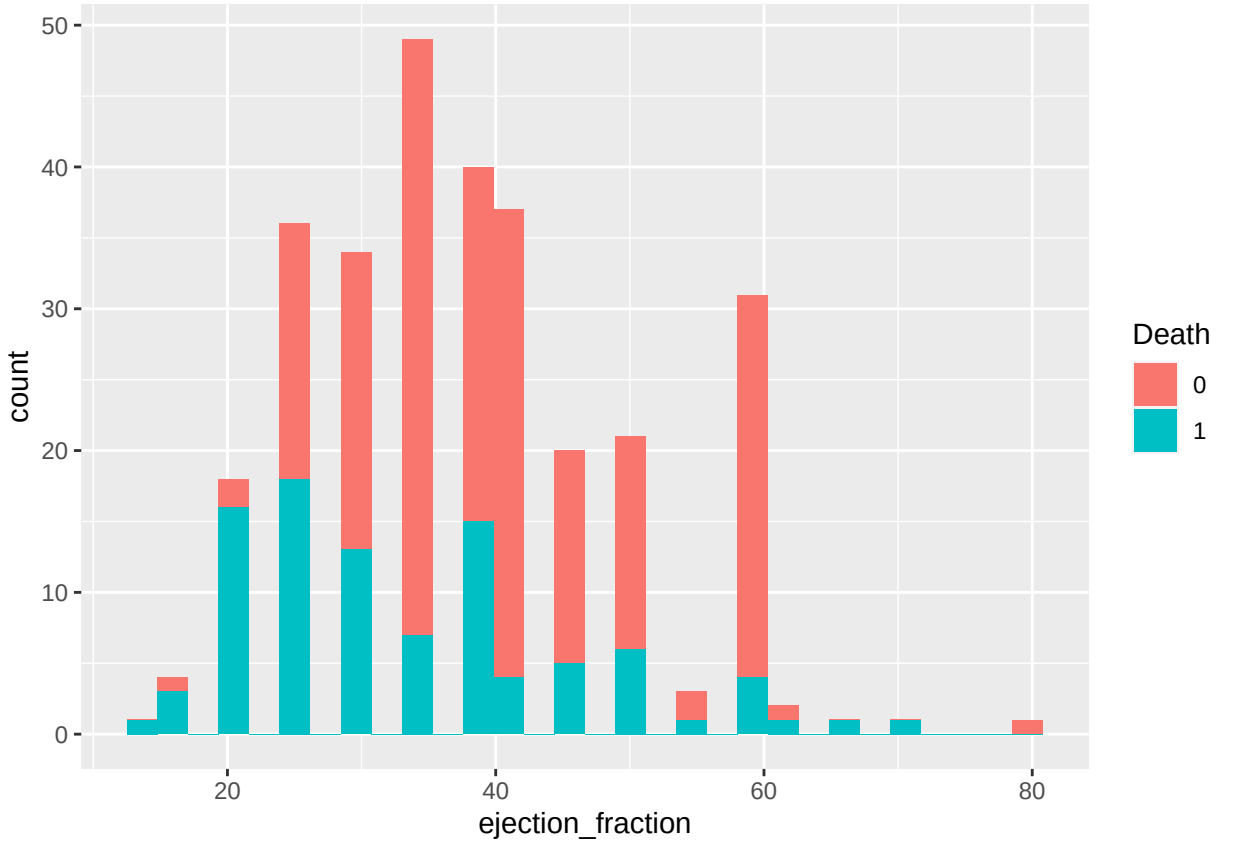



```
ggplot(heart, aes(x = age)) + geom_histogram(aes(fill = as.character(DEATH_EVENT)),
    bins = 30) + labs(fill = "Death")
```

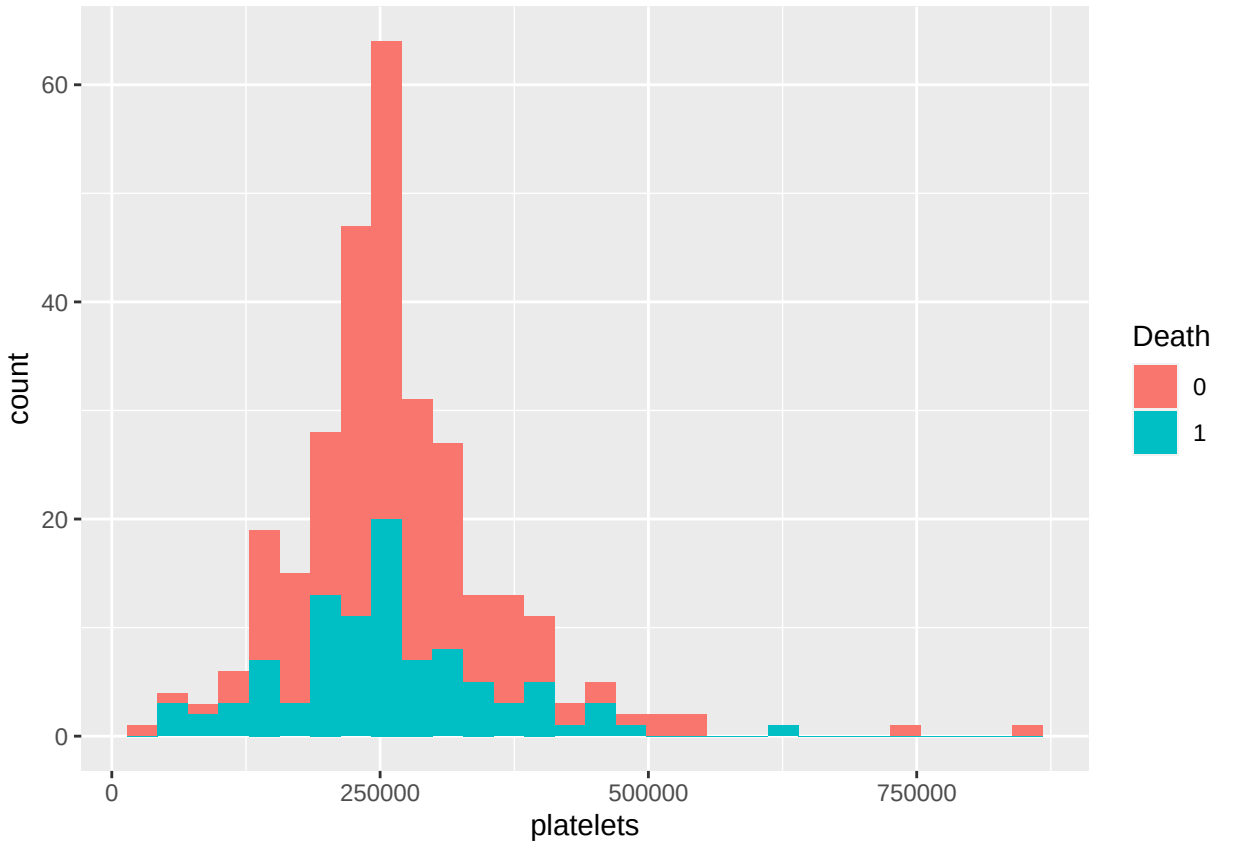

```
ggplot(heart, aes(x = creatinine_phosphokinase)) + geom_histogram(aes(fill = as.character(DEATH_EVENT))
    bins = 30) + labs(fill = "Death")
```



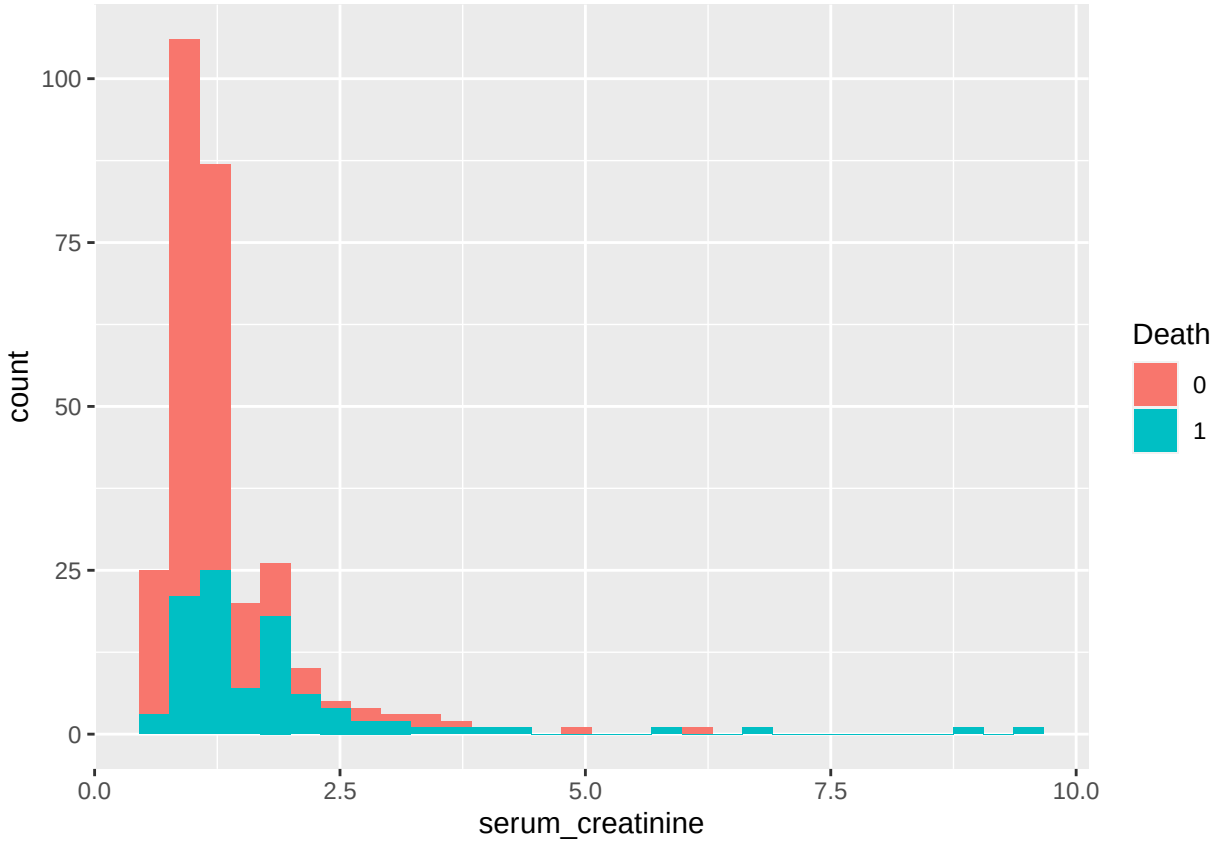

```
ggplot(heart, aes(x = ejection_fraction)) + geom_histogram(aes(fill = as.character(DEATH_EVENT)),
    bins = 30) + labs(fill = "Death")
```




```
ggplot(heart, aes(x = platelets)) + geom_histogram(aes(fill = as.character(DEATH_EVENT)),
    bins = 30) + labs(fill = "Death")
```

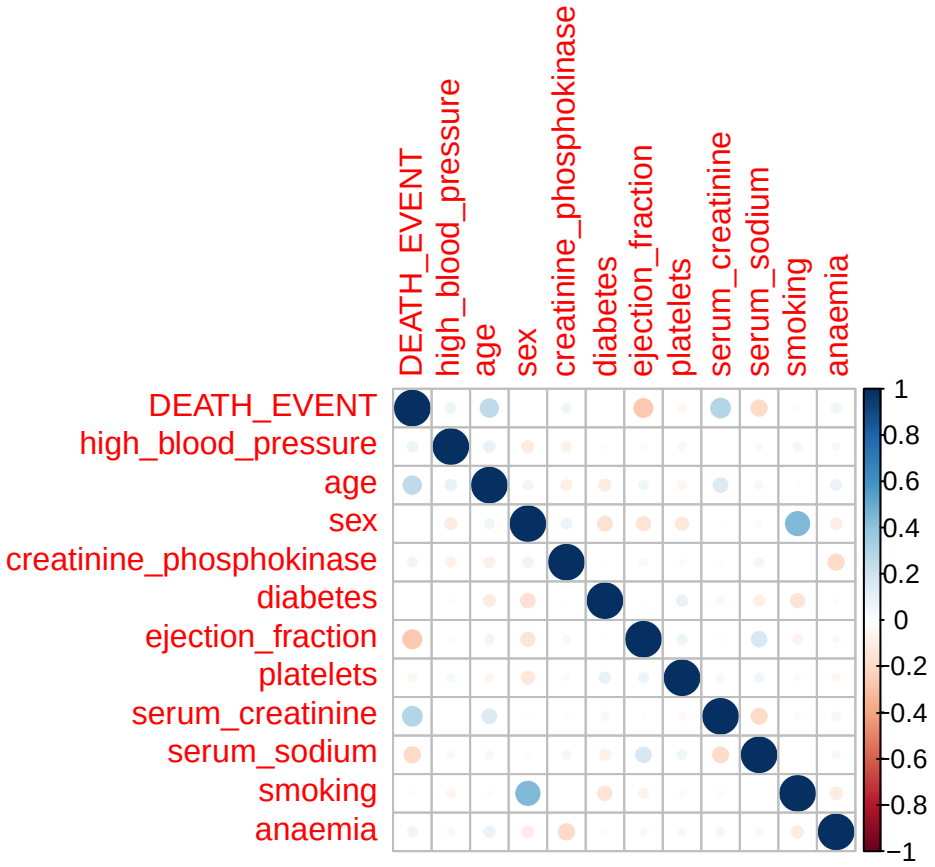



```
ggplot(heart, aes(x = serum_creatinine)) + geom_histogram(aes(fill = as.character(DEATH_EVENT)),
    bins = 30) + labs(fill = "Death")
```



4.2 Correlation matrix

```
pred <- c("high_blood_pressure", "age", "sex", "creatinine_phosphokinase",
    "diabetes", "ejection_fraction", "platelets", "serum_creatinine",
    "serum_sodium", "smoking", "anaemia")
target <- c("DEATH_EVENT")
# formula <- paste('DEATH_EVENT ~', paste(pred, collapse =
# '+'))
p <- length(pred)
n <- nrow(heart)
x = cor(heart[, c(target, pred)])
corrplot(x)
```



4.3 General functions

For testing purposes

```
predict.pointwise.accuracy <- function(fitted.model, test.data) {
    preds <- round(predict(fitted.model, newdata = test.data)[,
        1])
    preds.correct <- preds == test.data$DEATH_EVENT

    pointwise.accuracy <- length(preds.correct[preds.correct ==
        TRUE])/nrow(test.data)

    return(pointwise.accuracy)
}
```

5 Models

We chose to use BRMS for modeling. It stands for Bayesian Regression Models for Stan. It's an interface to fit Bayesian generalized (non-)linear multivariate models using Stan. As we need to do analysis for binary response variables, we need some sort of a generalized linear model. We also chose BRMS due to its ease of use when fitting such complicated multilevel generalized linear models.

In BRMS modeling, the parameters are said to either be population level or group level. Population-level parameters means the same thing as regular parameters in our course, and group-level parameters mean hyperparameters in hierarchical case.

Brms example, investigate results based on age. **Family argument** specifies the distribution family of the

output.

Prior argument for each of the parameters, in this case only age. One can set different priors for each population level parameter, or group level parameter.

```
# Split test and train data
test.size <- 0.3
train.indice <- sample(nrow(heart), nrow(heart) * (1 - test.size))
train.data <- heart[train.indice, ]
test.data <- heart[-train.indice, ]
```

5.1 Prior choices

The types of priors we considered in this project are uniformative priors and regularizing priors. Surely this dataset is not the only one about heart failure but since no prior knowledge are provided in the original papers we opted for using uninformative priors.

By default BRMS uses improper flat prior over the reals for population level parameters, and

5.2 Model fitting

5.2.1 Full model

Stan code for full model can be found in Appendix A.
Full model includes all the parameters that are specified in the dataset.

```
fit.full <- brm(formula = DEATH_EVENT ~ age + ejection_fraction +
    serum_creatinine + serum_sodium + high_blood_pressure + creatinine_phosphokinase +
    diabetes + smoking + anaemia, data = train.data, family = bernoulli(),
    refresh = 0)
```

```
## Compiling Stan program...
```

```
## Start sampling
```

5.2.2 WIP: Feature selected model

Stan code for feature selected model can be found in Appendix B.

In feature selected model, we hand pick the features that seems to be the most promising with regards to fitting the model. As described above, we saw that ejection_fraction, serum_creatinine, serum_sodium and age were correlating highly to death.

```
fit.feature_selected <- brm(formula = DEATH_EVENT ~ ejection_fraction +
    serum_creatinine + serum_sodium + age, data = train.data,
    family = bernoulli(), refresh = 0)
```

```
## Compiling Stan program...
```

```
## recompiling to avoid crashing R session
```

```
## Start sampling
```

5.2.3 Hierarchical model

Stan code for hierarchical model can be found in Appendix C.

In hierarchical model, we choose age as hyperparameter.
First we will discretize age data in to 3 equal depth bins.
Doing

```
heart$age[heart$age <= 55] = 0
heart$age[heart$age > 55 & heart$age < 70] = 1
heart$age[heart$age >= 70] = 2
hist(heart$age)
```

Then we can fit the model

```
fit.hierarchical <- brm(formula = DEATH_EVENT ~ ejection_fraction +
    serum_creatinine + serum_sodium + (ejection_fraction + serum_creatinine +
    serum_sodium | age), data = train.data, family = bernoulli(),
    refresh = 0, control = list(adapt_delta = 0.99), save_all_pars = T,
    save_pars = save_pars(all = T))
```

```
## Warning: Argument 'save_all_pars' is deprecated. Please use argument 'all' in
## function 'save_pars()' instead.

## Compiling Stan program...

## Start sampling

## Warning: There were 1 divergent transitions after warmup. See
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.

## Warning: Examine the pairs() plot to diagnose sampling problems

## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#bulk-ess
```

5.3 $\hat{R}$ convergence

5.4 HMC specific convergence diagnostics (divergences, tree depth) with interpretation of the results

5.5 Effective sample size diagnostic (n_eff or ESS) and an interpretation of the results

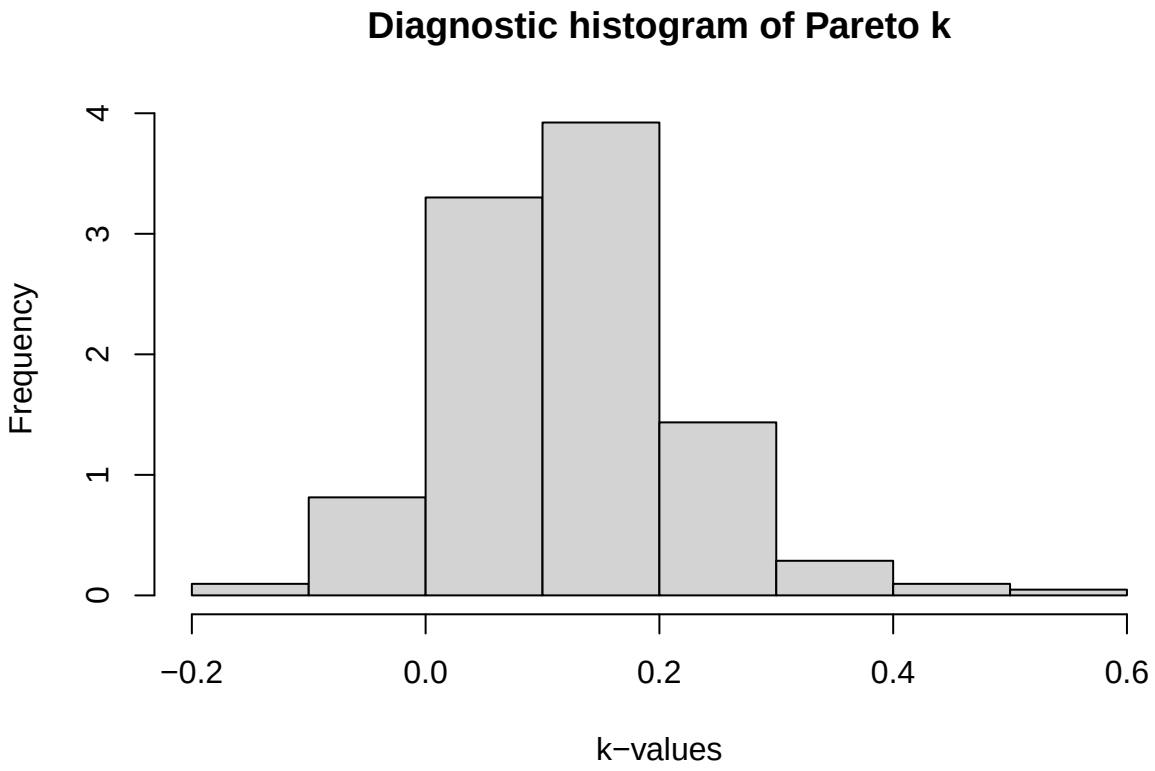
5.6 Posterior predictive checking and interpretation of the results

5.7 Model comparison and interpretation of the results

5.8 WIP: Predictive performance assessment (classification)

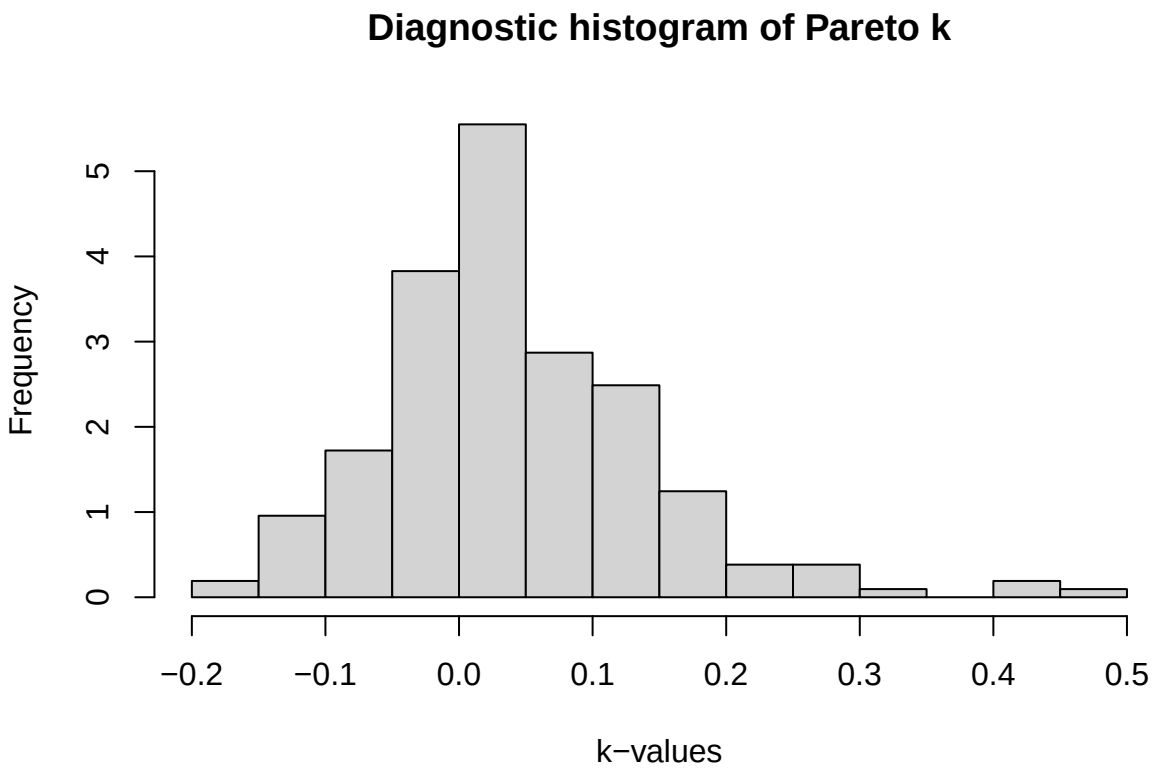
5.8.1 Full model

```
loo.full <- loo(fit.full)

hist(loo.full$diagnostics$pareto_k, main = "Diagnostic histogram of Pareto k",
    xlab = "k-values", ylab = "Frequency", freq = FALSE)
```



5.8.2 Feature selected model

```
loo.feature_selected <- loo(fit.feature_selected)

hist(loo.feature_selected$diagnostics$pareto_k, main = "Diagnostic histogram of Pareto k",
    xlab = "k-values", ylab = "Frequency", freq = FALSE)
```

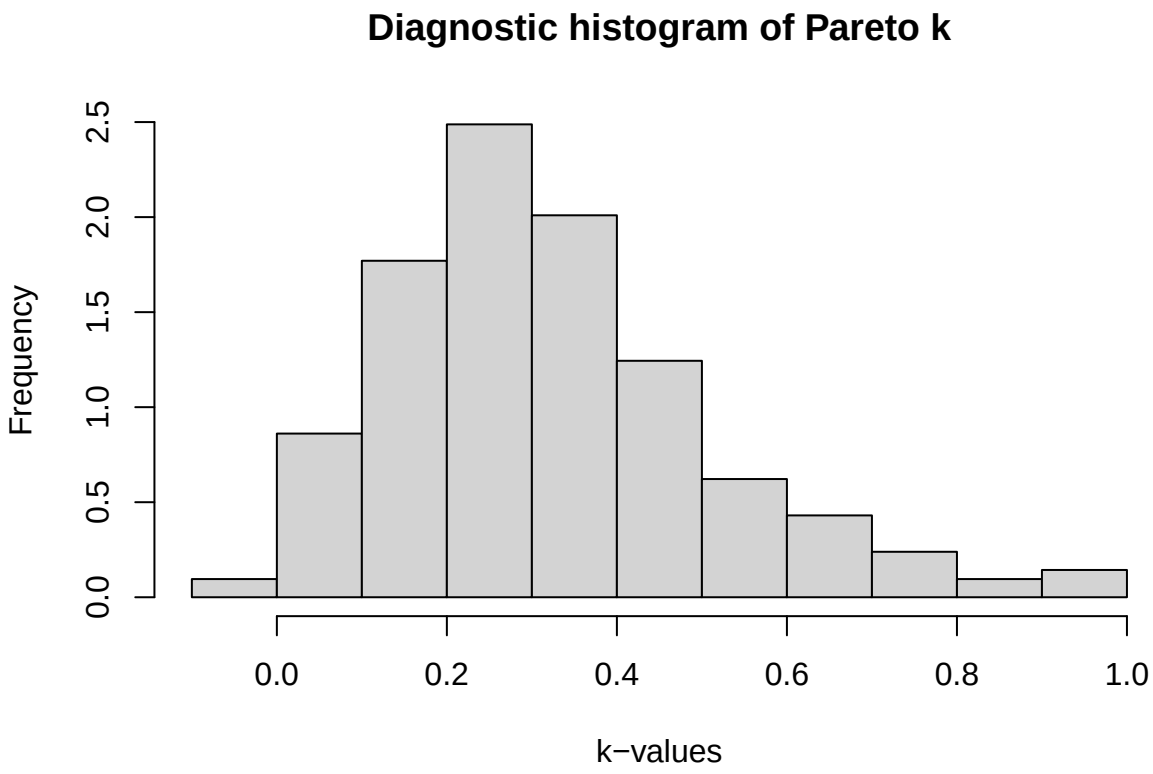


5.8.3 Hierarchical model

```
loo.hier <- loo(fit.hierarchical)
```

```
## Warning: Found 10 observations with a pareto_k > 0.7 in model
## 'fit.hierarchical'. It is recommended to set 'moment_match = TRUE' in order to
## perform moment matching for problematic observations.
```

```
hist(loo.hier$diagnostics$pareto_k, main = "Diagnostic histogram of Pareto k",
    xlab = "k-values", ylab = "Frequency", freq = FALSE)
```



5.9 Prior sensitivity analysis (alternative prior tested)

5.10 Discussion of problems and further improvements

NON-LINEAR SCRAP CODE:

```
# NON LINEAR
fitNonLinear <- brm(formula = DEATH_EVENT ~ s(ejection_fraction) +
    s(serum_creatinine) + s(serum_sodium), data = train.data,
    family = "gaussian", refresh = 0)
```

```
## Compiling Stan program...
```

```
## Start sampling
```

```
## Warning: There were 47 divergent transitions after warmup. See
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

6 Conclusion

7 Appendix

A. Stan code of full model

```
stancode(fit.full)
```

```
## // generated with brms 2.14.4
## functions {
## }
## data {
##   int<lower=1> N;  // total number of observations
##   int Y[N];  // response variable
##   int<lower=1> K;  // number of population-level effects
##   matrix[N, K] X;  // population-level design matrix
##   int prior_only;  // should the likelihood be ignored?
## }
## transformed data {
##   int Kc = K - 1;
##   matrix[N, Kc] Xc;  // centered version of X without an intercept
##   vector[Kc] means_X;  // column means of X before centering
##   for (i in 2:K) {
##     means_X[i - 1] = mean(X[, i]);
##     Xc[, i - 1] = X[, i] - means_X[i - 1];
##   }
## }
## parameters {
##   vector[Kc] b;  // population-level effects
##   real Intercept;  // temporary intercept for centered predictors
## }
## transformed parameters {
## }
## model {
##   // likelihood including all constants
##   if (!prior_only) {
##     target += bernoulli_logit_glm_lpmf(Y | Xc, Intercept, b);
##   }
##   // priors including all constants
##   target += student_t_lpdf(Intercept | 3, 0, 2.5);
## }
## generated quantities {
##   // actual population-level intercept
##   real b_Intercept = Intercept - dot_product(means_X, b);
## }
```

B. Stan feature selected model

```
stancode(fit.feature_selected)
```

```
## // generated with brms 2.14.4
## functions {
## }
## data {
##   int<lower=1> N;  // total number of observations
##   int Y[N];  // response variable
##   int<lower=1> K;  // number of population-level effects
##   matrix[N, K] X;  // population-level design matrix
##   int prior_only;  // should the likelihood be ignored?
## }
## transformed data {
##   int Kc = K - 1;
##   matrix[N, Kc] Xc;  // centered version of X without an intercept
##   vector[Kc] means_X;  // column means of X before centering
##   for (i in 2:K) {
##     means_X[i - 1] = mean(X[, i]);
##     Xc[, i - 1] = X[, i] - means_X[i - 1];
##   }
## }
## parameters {
##   vector[Kc] b;  // population-level effects
##   real Intercept;  // temporary intercept for centered predictors
## }
## transformed parameters {
## }
## model {
##   // likelihood including all constants
##   if (!prior_only) {
##     target += bernoulli_logit_glm_lpmf(Y | Xc, Intercept, b);
##   }
##   // priors including all constants
##   target += student_t_lpdf(Intercept | 3, 0, 2.5);
## }
## generated quantities {
##   // actual population-level intercept
##   real b_Intercept = Intercept - dot_product(means_X, b);
## }
```

C. Stan hierarhical model

```
stancode(fit.hierarchical)
```

```
## // generated with brms 2.14.4
## functions {
##   /* turn a vector into a matrix of defined dimension
##    * stores elements in row major order
##    * Args:
##    *   X: a vector
##    *   N: first dimension of the desired matrix
##    *   K: second dimension of the desired matrix
##    * Returns:
##    *   a matrix of dimension N x K
##    */
##   matrix as_matrix(vector X, int N, int K) {
##     matrix[N, K] Y;
##     for (i in 1:N) {
##       Y[i] = to_row_vector(X[((i - 1) * K + 1):(i * K)]);
##     }
##     return Y;
##   }
##  /* compute correlated group-level effects
##   * Args:
##   *   z: matrix of unscaled group-level effects
##   *   SD: vector of standard deviation parameters
##   *   L: cholesky factor correlation matrix
##   * Returns:
##   *   matrix of scaled group-level effects
##   */
##   matrix scale_r_cor(matrix z, vector SD, matrix L) {
##     // r is stored in another dimension order than z
##     return transpose(diag_pre_multiply(SD, L) * z);
##   }
## }
## data {
##   int<lower=1> N;  // total number of observations
##   int Y[N];  // response variable
##   int<lower=1> K;  // number of population-level effects
##   matrix[N, K] X;  // population-level design matrix
##   // data for group-level effects of ID 1
##   int<lower=1> N_1;  // number of grouping levels
##   int<lower=1> M_1;  // number of coefficients per level
##   int<lower=1> J_1[N];  // grouping indicator per observation
##   // group-level predictor values
##   vector[N] Z_1_1;
##   vector[N] Z_1_2;
##   vector[N] Z_1_3;
##   vector[N] Z_1_4;
##   int<lower=1> NC_1;  // number of group-level correlations
##   int prior_only;  // should the likelihood be ignored?
## }
## transformed data {
##   int Kc = K - 1;
##   matrix[N, Kc] Xc;  // centered version of X without an intercept
```

```
##   vector[Kc] means_X;  // column means of X before centering
##   for (i in 2:K) {
##     means_X[i - 1] = mean(X[, i]);
##     Xc[, i - 1] = X[, i] - means_X[i - 1];
##   }
## }
## parameters {
##   vector[Kc] b;  // population-level effects
##   real Intercept;  // temporary intercept for centered predictors
##   vector<lower=0>[M_1] sd_1;  // group-level standard deviations
##   matrix[M_1, N_1] z_1;  // standardized group-level effects
##   cholesky_factor_corr[M_1] L_1;  // cholesky factor of correlation matrix
## }
## transformed parameters {
##   matrix[N_1, M_1] r_1;  // actual group-level effects
##   // using vectors speeds up indexing in loops
##   vector[N_1] r_1_1;
##   vector[N_1] r_1_2;
##   vector[N_1] r_1_3;
##   vector[N_1] r_1_4;
##   // compute actual group-level effects
##   r_1 = scale_r_cor(z_1, sd_1, L_1);
##   r_1_1 = r_1[, 1];
##   r_1_2 = r_1[, 2];
##   r_1_3 = r_1[, 3];
##   r_1_4 = r_1[, 4];
## }
## model {
##   // likelihood including all constants
##   if (!prior_only) {
##     // initialize linear predictor term
##     vector[N] mu = Intercept + rep_vector(0.0, N);
##     for (n in 1:N) {
##       // add more terms to the linear predictor
##       mu[n] += r_1_1[J_1[n]] * Z_1_1[n] + r_1_2[J_1[n]] * Z_1_2[n] + r_1_3[J_1[n]] * Z_1_3[n] + r_1_
##     }
##     target += bernoulli_logit_glm_lpmf(Y | Xc, mu, b);
##   }
##   // priors including all constants
##   target += student_t_lpdf(Intercept | 3, 0, 2.5);
##   target += student_t_lpdf(sd_1 | 3, 0, 2.5)
##     - 4 * student_t_lccdf(0 | 3, 0, 2.5);
##   target += std_normal_lpdf(to_vector(z_1));
##   target += lkj_corr_cholesky_lpdf(L_1 | 1);
## }
## generated quantities {
##   // actual population-level intercept
##   real b_Intercept = Intercept - dot_product(means_X, b);
##   // compute group-level correlations
##   corr_matrix[M_1] Cor_1 = multiply_lower_tri_self_transpose(L_1);
##   vector<lower=-1,upper=1>[NC_1] cor_1;
##   // extract upper diagonal of correlation matrix
##   for (k in 1:M_1) {
##     for (j in 1:(k - 1)) {
```

```
##       cor_1[choose(k - 1, 2) + j] = Cor_1[j, k];
##     }
##   }
## }
```

References

[1] Ejection fraction heart failure measurement, 2017.

[2] Ahmad T, Munir A, Bhatti SH, Aftab M, Raza MA. Survival analysis of heart failure patients: A case study. 2017. doi: https://doi.org/10.1371/journal.pone.0181001.

[3] Christine Case-Lo. Blood sodium test, 2018. URL https://www.healthline.com/health/sodium-blood.

[4] Gregg D, Goldschmidt-Clermont P. J. Platelets and cardiovascular disease. *Journal of the American Heart Association*, 108, 2003. doi: https://doi.org/10.1161/01.CIR.0000086897.15588.4B.

[5] Roshan Patel Ravinder S. Aujla. Creatine phosphokinase. *StatPearls*, 2020. URL https://www.ncbi.nlm.nih.gov/books/NBK546624/.

[6] Roth Erica. Creatinine blood test, 2019. URL https://www.healthline.com/health/creatinine-blood.