# Heart Failure Prediction using Bayesian Approach

Teemu Sormunen, Abdullah Günay, Nicola Brazzale

11/18/2020

## Contents

**1 Introduction** **2**
- 1.1 The problem . . . 2
- 1.2 The motivation . . . 2
- 1.3 Modeling idea . . . 2

**2 Dataset** **3**
- 2.1 Term explanation . . . 3
- 2.2 Dataset introduction . . . 3

**3 Packages** **4**

**4 Data preprocessing and visualization** **4**
- 4.1 Plot histograms . . . 4
- 4.2 Correlation matrix . . . 10

**5 Models** **12**
- 5.1 Model fitting . . . 12
- 5.2 Prior choices . . . 15
- 5.3 $\hat{R}$ convergence . . . 17
- 5.4 HMC specific convergence diagnostics (divergences, tree depth) with interpretation of the results 20
- 5.5 Effective sample size diagnostic (n_eff or ESS) and an interpretation of the results . . . 20
- 5.6 Posterior predictive checking and interpretation of the results . . . 20
- 5.7 Model comparison and interpretation of the results . . . 20
- 5.8 WIP: Predictive performance assessment (classification) . . . 20
- 5.9 Prior sensitivity analysis (alternative prior tested) . . . 23
- 5.10 Discussion of problems and further improvements . . . 23

**6 Conclusion** **23**
- 6.1 Self-reflection . . . 23

**Appendix** **24**
- A. Stan code of full model . . . 24
- B. Stan feature selected model . . . 25
- C. Stan hierarhical model . . . 26
- 6.2 D. Stan death time analysis model . . . 29

# 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**

In this project, we are observing the dataset from two distinct point of views. On the other hand, we want to do predictive analysis whether patient dies, or survives the research follow up period (4 - 285 days), and on the other hand we want to create an actual survival analysis, which takes time and censored variables in to consideration. For this reason we are inspecting the variables from two points. More about this in upcoming chapters.

## 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 4 models which are then compared based on $\hat{r}$, $n_{eff}$, using the loo package and the classification accuracy. The three first models ignore the time feature, and simply predict whether patient has died (1) during the experiment duration, or survived (0). We chose this approach to practise survival analysis with binary outcome.
We also created fourth model, which predicts the time with respect to death event. In this case, the death event 1 means the patient has died, and death event 0 means that the patient is censored from the study. Censoring practically means, that the patient has opted out of the study, and the researches couldn't reach him anymore. In this context, it doesn't necessarily mean that the patient has survived, but we just don't know the outcome.

The 1st model is the reduced model and consists in fewer varibles which are selected base on their correlation with the death event. The 2nd model consists in all variables except for the variable "time" as we believe that doesn'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 similar type of linear model, but this time we consider time as outcome variable with respect to death event. We use correlation matrix to select the strongest variable that correspond with time, and we also use BRMS internal function cens() for taking the censored variable DEATH_EVENT to consideration. More clear explanation is given later.

## 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]

- **Time** Time variable indicates the time since the research has started for that person (the time of ventricular systolic dysfunction). We have time variable included, because we have to inspect when the death events are happening. This variable is ignored in the first three models, because we wanted to also interpret this dataset from binary survival approach, so predict whether patient dies or not.

### 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 follow up period was 4 to 285 days, with average of 130 days. This has to be taken in to consideration when doing the survival analysis.
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

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

```
##
## -- 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

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

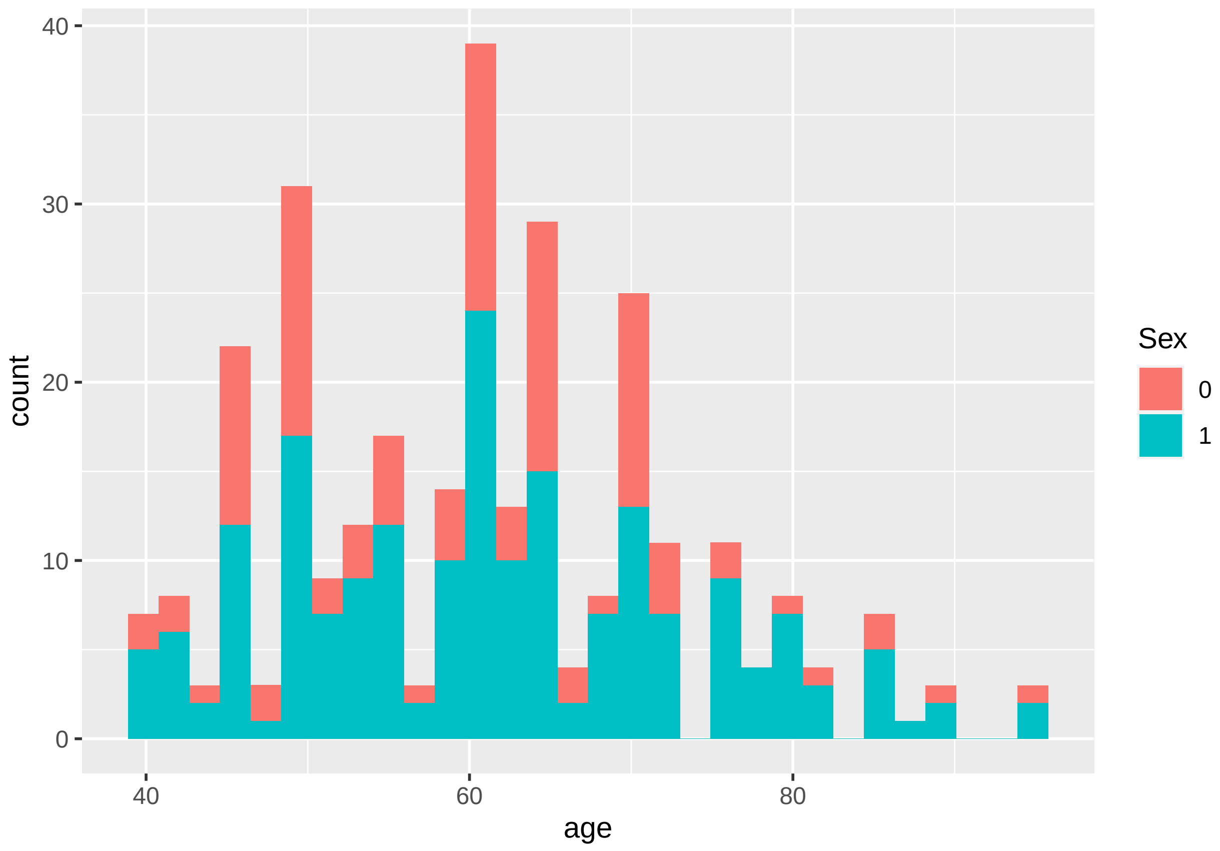
## 4 Data preprocessing and visualization

### 4.1 Plot histograms

We are first plotting the histograms to get an overview of the dataset.

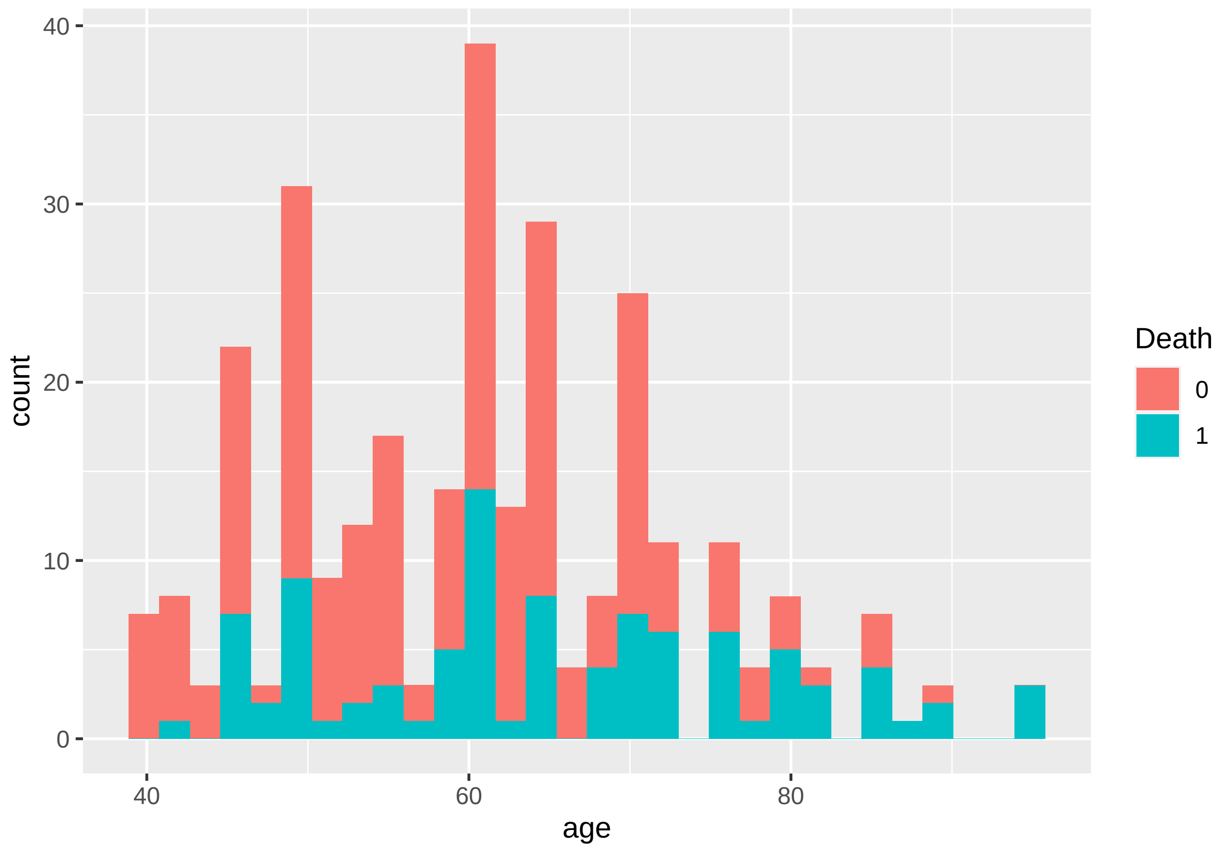
It seems like the sex between ages is distributed quite evenly, there's slightly more patients from the 50-60.

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



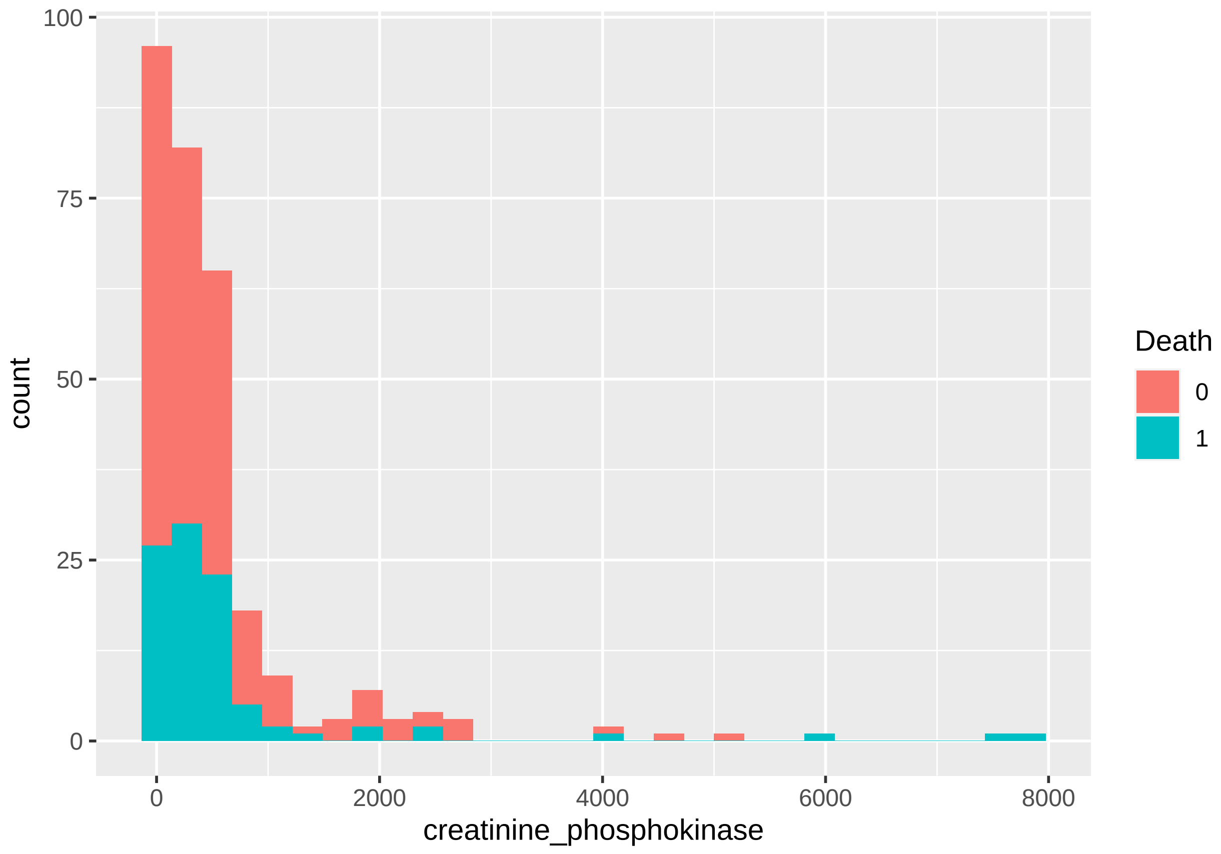
This histogram might suggest us that older people die during this follow up period with higher probability, and younger people either survive or opt-out of the study.

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



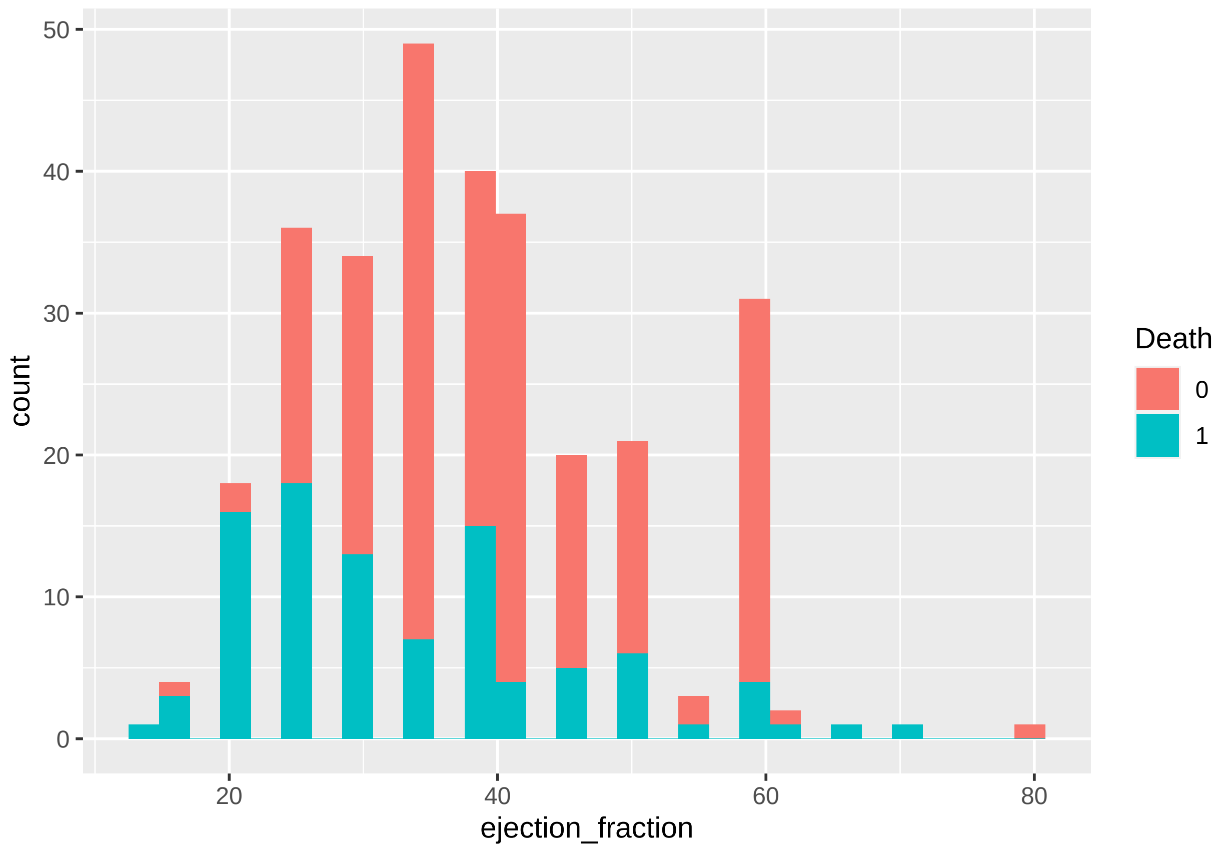
Creatinine phosphokinase doesn't bring too much information based on the histogram, although it shows us that people with high creatinine phosphokinase might tend to die more often. Because almost everyone who attended this study had already increased phosphokinase levels, we should take this interpretation with a slight grain of salt.

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



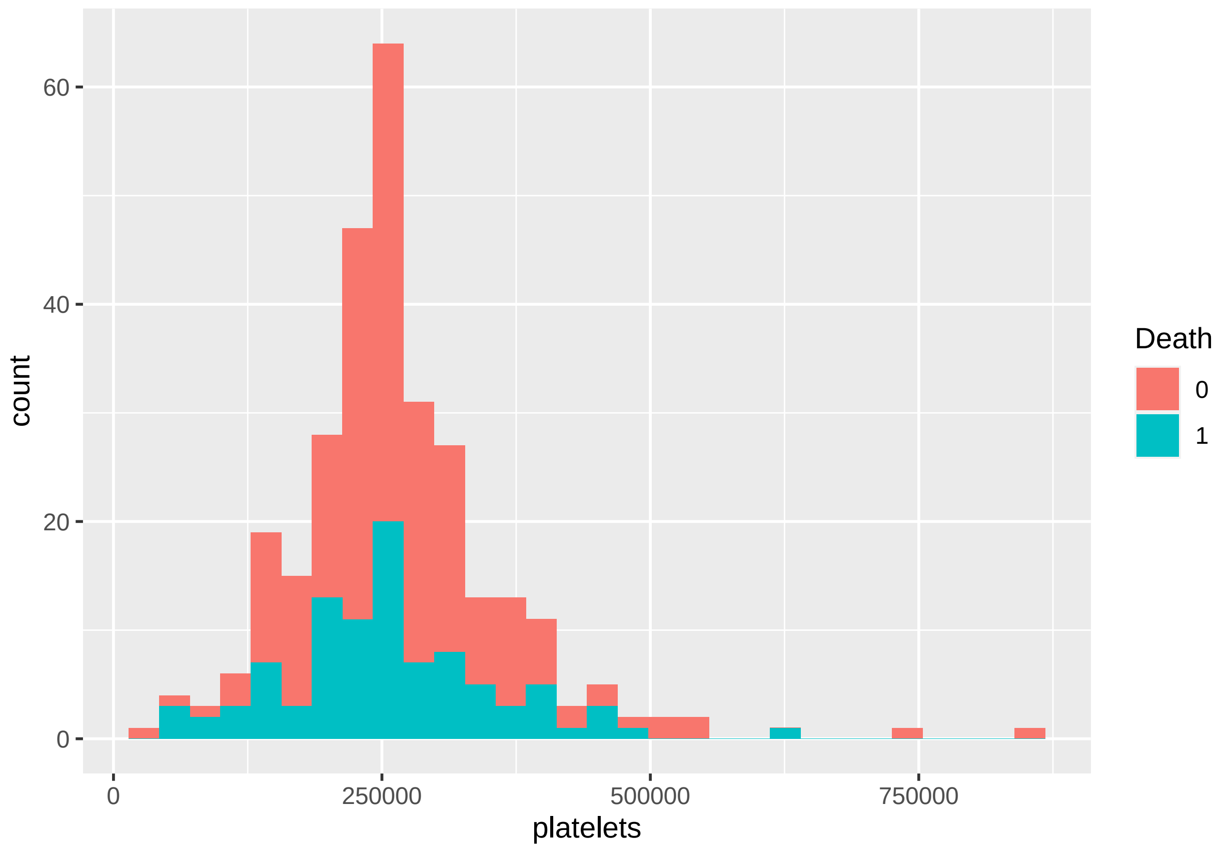
Ejection fraction seems to correlate strongly with death. This is logical, because ejection fraction measures the hearts ability to pump blood. Here we can again see that most of the people fall under normal levels of EF.

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



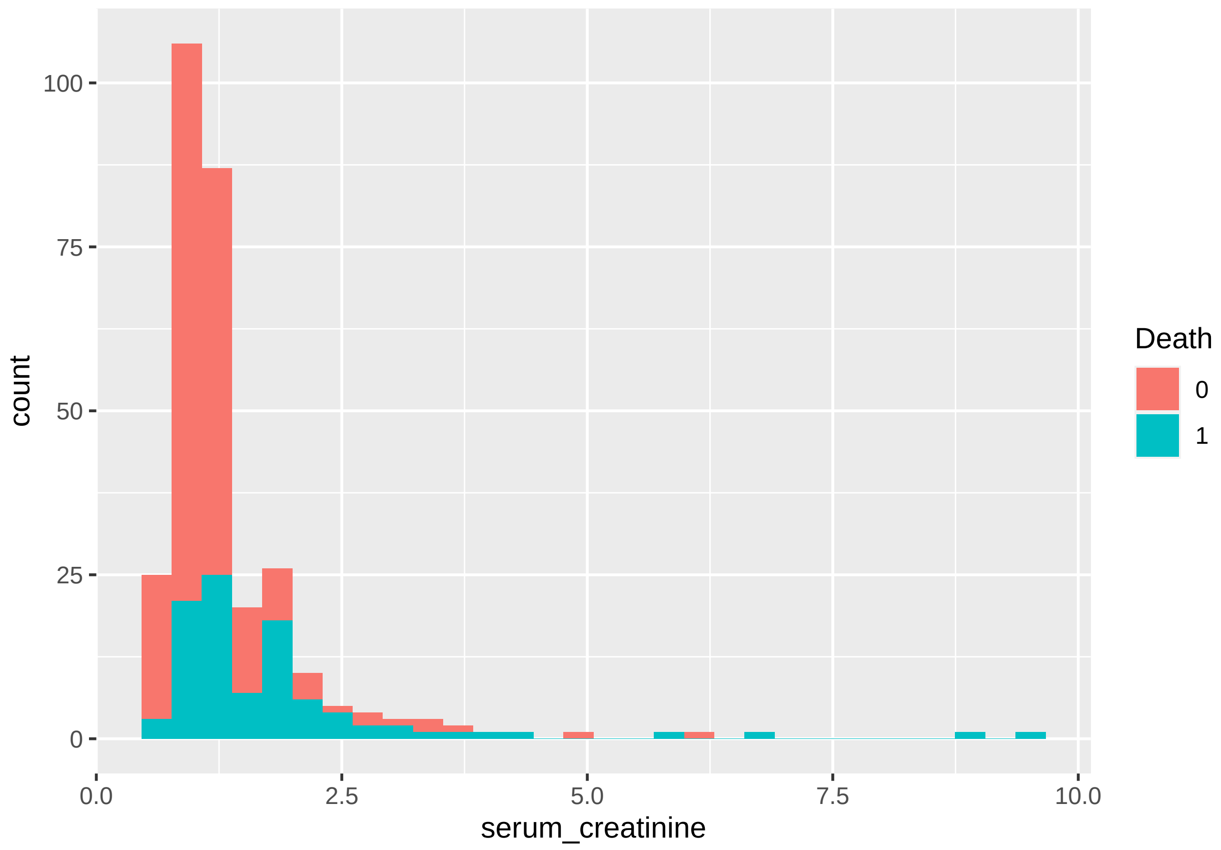
Platelets seems to follow quite even distribution with respect to death event, and there isn't too much information available only based on the histogram.

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



Serum creatinine seems to correlate quite highly with death. When the upper bound for serum creatinine 1.3 is passed, it seems that probability for death becomes high.

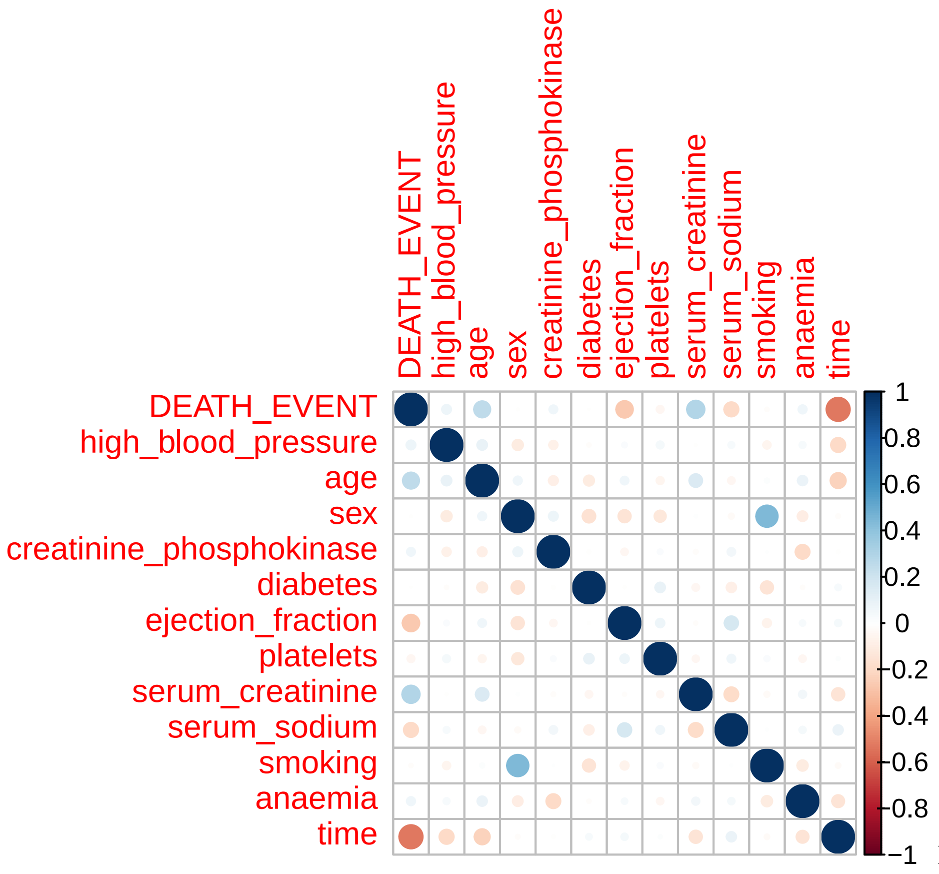
```
ggplot(heart, aes(x = serum_creatinine)) + geom_histogram(aes(fill = as.character(DEATH_EVENT)),
    bins = 30) + labs(fill = "Death")
```



## 4.2 Correlation matrix

By plotting the correlation matrix we can see the correlations.

```
pred <- c("high_blood_pressure", "age", "sex", "creatinine_phosphokinase",
    "diabetes", "ejection_fraction", "platelets", "serum_creatinine",
    "serum_sodium", "smoking", "anaemia", "time")
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)
```



Looking at the correlations only by eye might not be enough, so let's list the correlations with respect to death_event in sorted order

```
sort(x[1, ])
```

```
##                  time        ejection_fraction             serum_sodium
##          -0.526963779             -0.268603312             -0.195203596
##             platelets                  smoking                      sex
##          -0.049138868             -0.012623153             -0.004316376
##              diabetes creatinine_phosphokinase                  anaemia
##          -0.001942883              0.062728160              0.066270098
##   high_blood_pressure                      age          serum_creatinine
##           0.079351058              0.253728543              0.294277561
##           DEATH_EVENT
##           1.000000000
```

We see that highly positively correlating variables are age and serum creatinine, which found to be true already earlier. Strongest negative correlations are time, ejection fraction and serum sodium. Time clearly indicates that people tend to die early after the ventricular systolic dysfunction has happened. Serum sodium also seems to correlate negatively, because the low sodium levels are usually occurring after heart failure. If we have to make conclusions, then we could conclude that the sodium levels are lower in the people who have had more severe ventricular systolic dysfunction.

For the fourth survival analysis model we should also look at the highest correlations according to time.

```
sort(x[nrow(x), ])
```

```
##           DEATH_EVENT                      age      high_blood_pressure
##          -0.526963779             -0.224068420             -0.196439479
##      serum_creatinine                  anaemia                  smoking
##              -0.149315418             -0.141413982             -0.022838942 
##                       sex creatinine_phosphokinase                platelets 
##              -0.015608220             -0.009345653              0.010513909 
##                  diabetes        ejection_fraction             serum_sodium 
##               0.033725509              0.041729235              0.087640000 
##                      time 
##               1.000000000
```

According to time we see that highest negative correlation when death_event is discarded are age, high blood pressure, serum creatinine and anemia. This means that old people tend to die early, with high blood pressure, with high serum creatinine, with anemia. This seems to somewhat run hand in hand with the previous conclusions, although this time high blood pressure and anemia was introduced. High blood pressure sounds like it could lead to heart attack easily, which makes sense. Also anemia seems to go hand in hand with heart attack.

## 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.

**Family argument** specifies the distribution family of the response variable.

**Prior argument** for each of the parameters. One can set different priors for each population level parameter, or group level parameter.

Create general function for splitting the train and test data.

```
split.train.test <- function(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, ]
    return(list(train = train.data, test = test.data))
}
new.data <- split.train.test(heart)
train.data <- new.data$train
test.data <- new.data$test
```

### 5.1 Model fitting

The Generalised Linear Model used for every parametrisation is Bernoulli-Logit Generalised Linear Model for the first three models, which is logistic regression. It's expressed in mathematical terms as following:
$BernoulliLogitGLM(y|x, \alpha, \beta) = \prod_{1 \le i \le n} Bernoulli(y_i|logit^{-1}(\alpha_i + x_i \times \beta))$

#### 5.1.1 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, prior = c(set_prior("normal(0,50)",
    class = "b", coef = "age"), set_prior("normal(0,100)", class = "b",
    coef = "serum_creatinine"), set_prior("normal(0,100)", class = "b",
    coef = "ejection_fraction"), set_prior("normal(0,1000)",
    class = "b", coef = "serum_sodium"), set_prior("normal(.5, .5)",
    class = "b", coef = "anaemia"), set_prior("normal(.5, .5)",
    class = "b", coef = "diabetes"), set_prior("normal(.5, .5)",
    class = "b", coef = "smoking"), set_prior("normal(.5, .5)",
    class = "b", coef = "high_blood_pressure")), data = train.data,
    family = bernoulli(), refresh = 0)
```

```
## Compiling Stan program...

## Start sampling
```

### 5.1.2 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 in correlation analysis, we saw that ejection_fraction, serum_creatinine, serum_sodium and age were correlating to death.

Based on this we can choose these variables to be the important ones, and build a model with them.

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

```
## Compiling Stan program...

## Start sampling
```

### 5.1.3 Hierarchical model

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

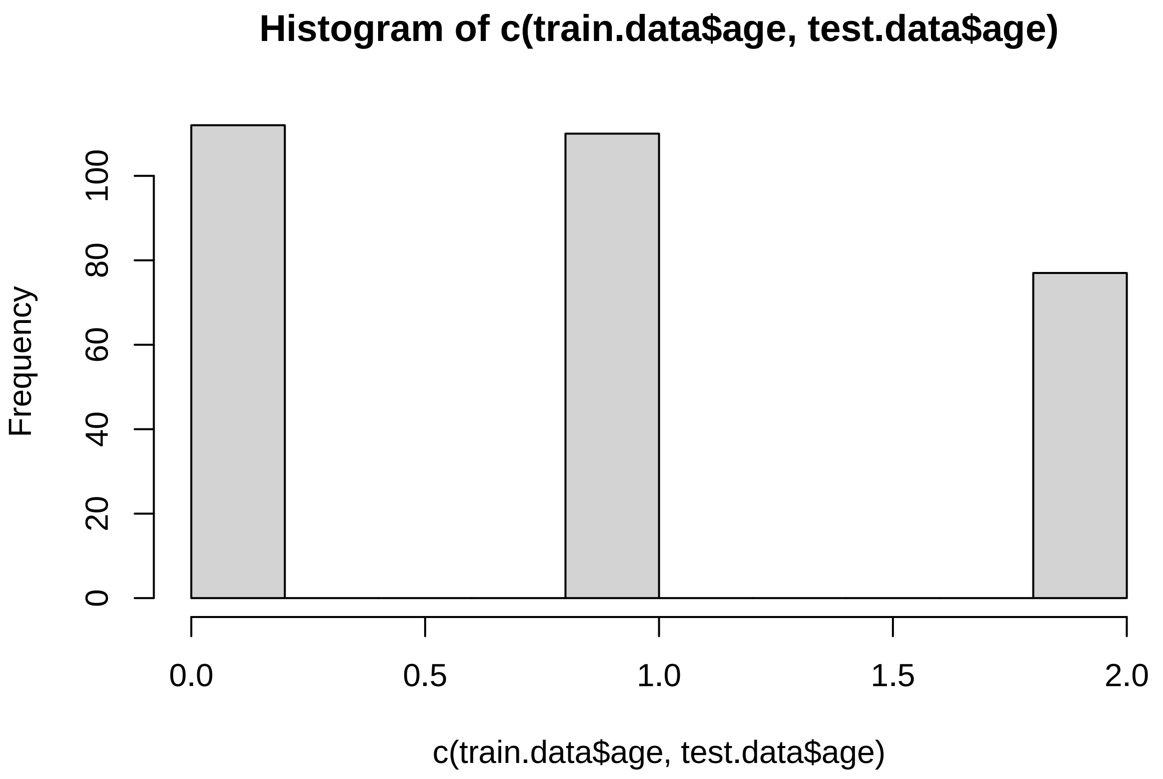
In hierarchical model, we choose age as hyperparameter, because by intuition we thought that different aged people tend to have different medical conditions by default. This intuition is supported by calculating the absolute row sums of the correlation matrix rows, and seeing absolute correlations of variables.

```
sort(rowSums(abs(x)))
```

```
##                platelets creatinine_phosphokinase                 diabetes
##                 1.651267                 1.651341                 1.710253
##      high_blood_pressure                  anaemia                  smoking
##                 1.767362                 1.908799                 1.940394
##        ejection_fraction         serum_creatinine             serum_sodium
##                 1.950445                 1.999252                 2.016403
##                      age                      sex                     time
##                 2.243391                 2.276192                 2.459603
##              DEATH_EVENT
##                 2.815147
```

First we will discretize age data in to 3 equal depth bins.

```r
test.data$age[test.data$age <= 55] = 0
test.data$age[test.data$age > 55 & test.data$age < 70] = 1
test.data$age[test.data$age >= 70] = 2

train.data$age[train.data$age <= 55] = 0
train.data$age[train.data$age > 55 & train.data$age < 70] = 1
train.data$age[train.data$age >= 70] = 2
hist(c(train.data$age, test.data$age))
```



Then we can fit the model

```r
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)
```

```
## Compiling Stan program...

## Start sampling

## Warning: There were 18 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

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

### 5.1.4 Model for death time analysis

As a fourth model we have model with response variable according to Weibull distribution, and features are selected based on correlation matrix.
In death time analysis model we are using different functions based on whether sample is censored or not. If sample is censored, then we are using log probability density function
$Weibull(y|\alpha,\sigma) = \frac{\alpha}{\sigma}\left(\frac{y}{\sigma}\right)^{-\alpha-1} exp\left(-\left(\frac{y}{\sigma}\right)^{\alpha}\right)$. If sample is not censored, then we are using the cumulative function of previously mentioned probability density function.
First we need to retrieve the original data with original age values

```
new.data <- split.train.test(heart)
train.data <- new.data$train
test.data <- new.data$test
```

Then we can fit the model

```
fit.weibull <- brm(formula = time | cens(DEATH_EVENT) ~ high_blood_pressure +
    age + serum_creatinine + serum_sodium, data = train.data,
    family = weibull(), refresh = 0)
```

```
## Compiling Stan program...

## Start sampling
```

## 5.2 Prior choices

We chose priors based on articles that we read about medical measurements. We used really weakly informative priors, but all the priors were based on what

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

```
## Compiling Stan program...

## recompiling to avoid crashing R session

## Start sampling

##
## SAMPLING FOR MODEL 'fc5c5b8a5161ca9226ca5d50c5c16cee' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 3e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.3 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 2.38679 seconds (Warm-up)
## Chain 1:                0.717727 seconds (Sampling)
## Chain 1:                3.10452 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'fc5c5b8a5161ca9226ca5d50c5c16cee' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.9e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.19 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 2.21949 seconds (Warm-up)
## Chain 2:                0.776383 seconds (Sampling)
## Chain 2:                2.99587 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'fc5c5b8a5161ca9226ca5d50c5c16cee' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 1.7e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.17 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 2.0984 seconds (Warm-up)
## Chain 3:                0.78406 seconds (Sampling)
## Chain 3:                2.88246 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'fc5c5b8a5161ca9226ca5d50c5c16cee' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1.8e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.18 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 2.26074 seconds (Warm-up)
## Chain 4:                0.69057 seconds (Sampling)
## Chain 4:                2.95131 seconds (Total)
## Chain 4:
```

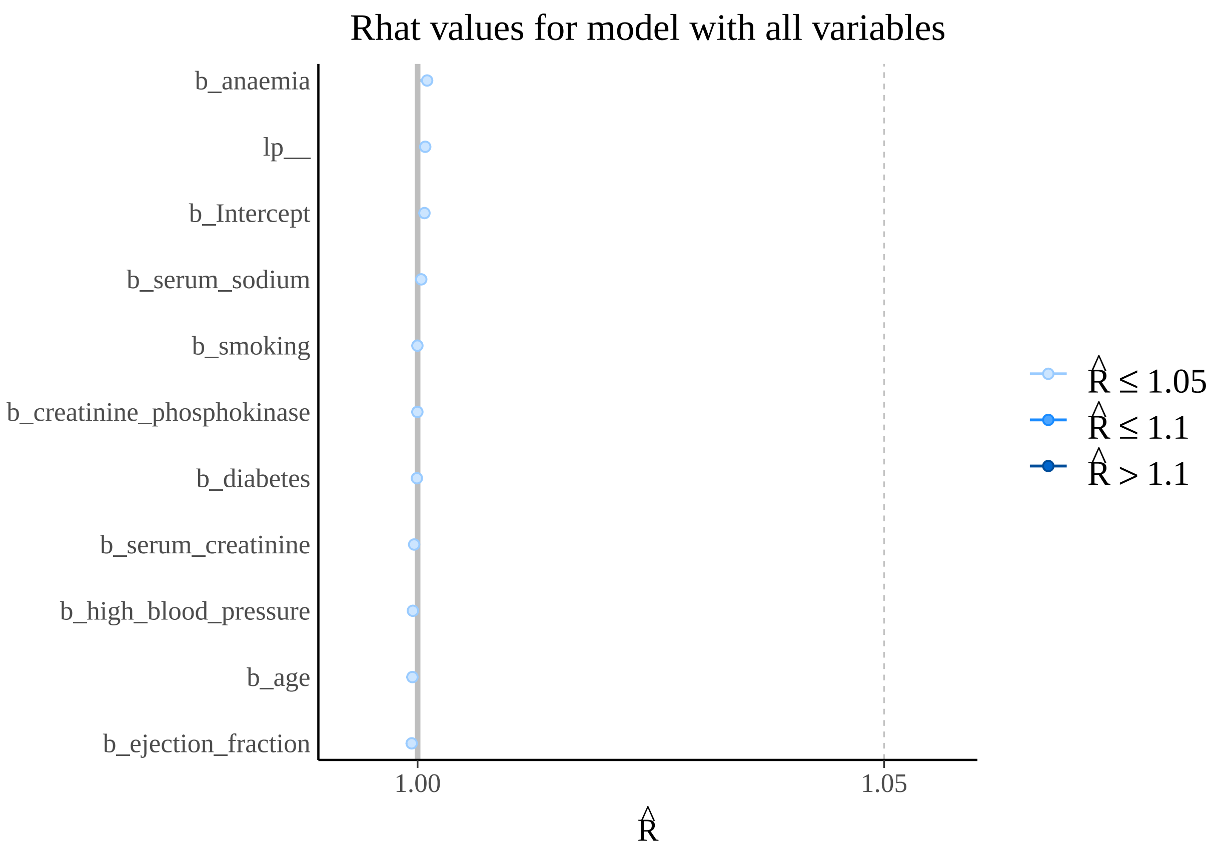
The types of priors we considered in this project are uninformative priors and regularizing priors. Surely this dataset is not the only one about heart failure but since no prior knowledge are provided in the original papers we opted for using uninformative priors. The domain is also highly specialized, as the target group is specific group of people over 40 years who have gone through ventricular systolic dysfunction. We can't simply infer any prior knowledge on this group.

By default BRMS uses improper flat prior over the reals for population level parameters. Group level parameter is assumed to come from multivariate normal distribution with zero mean and unknown covariance matrix.

## 5.3 $\hat{R}$ convergence

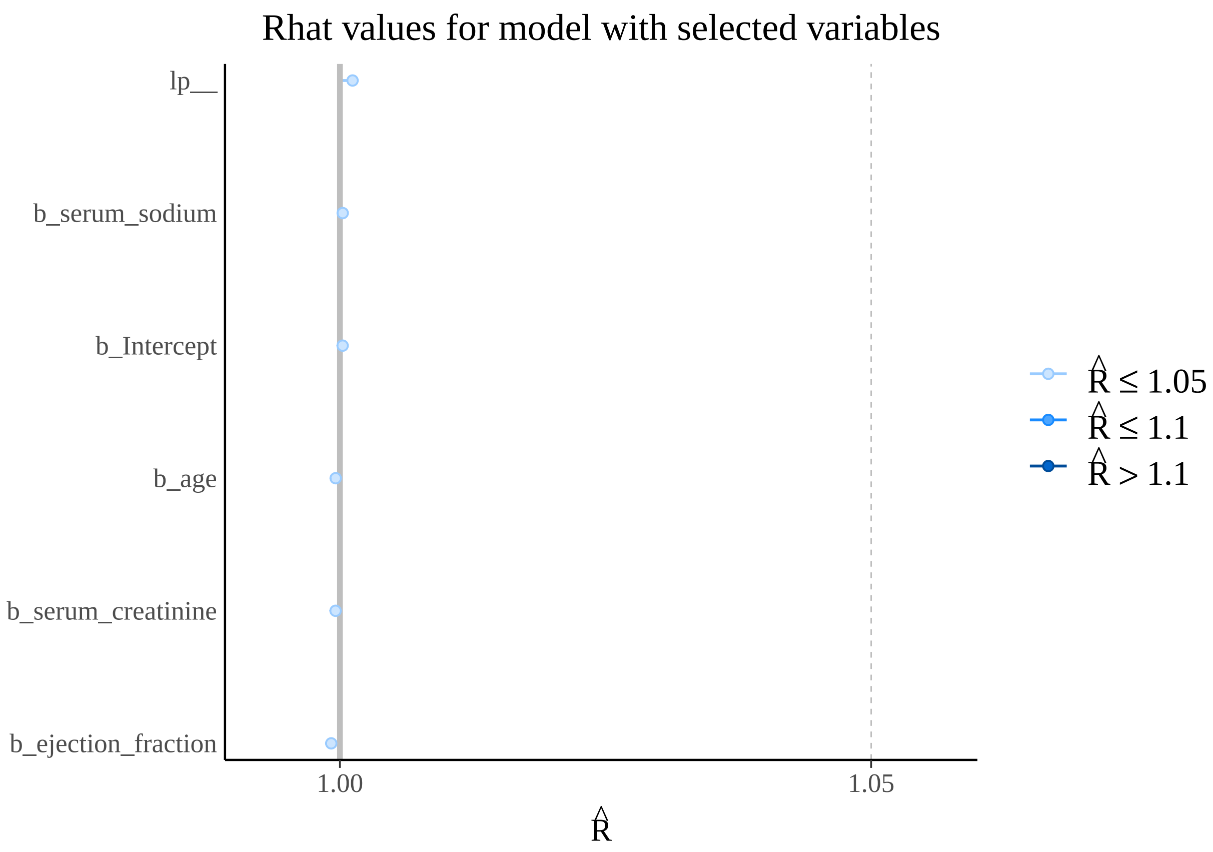
### 5.3.1 Summary for the linear model with all variables

```r
rhats <- rhat(fit.full)
color_scheme_set("brightblue")  # see help('color_scheme_set')
mcmc_rhat(rhats) + yaxis_text(hjust = 1) + ggtitle("Rhat values for model with all variables") +
    theme(plot.title = element_text(hjust = 0.5))
```



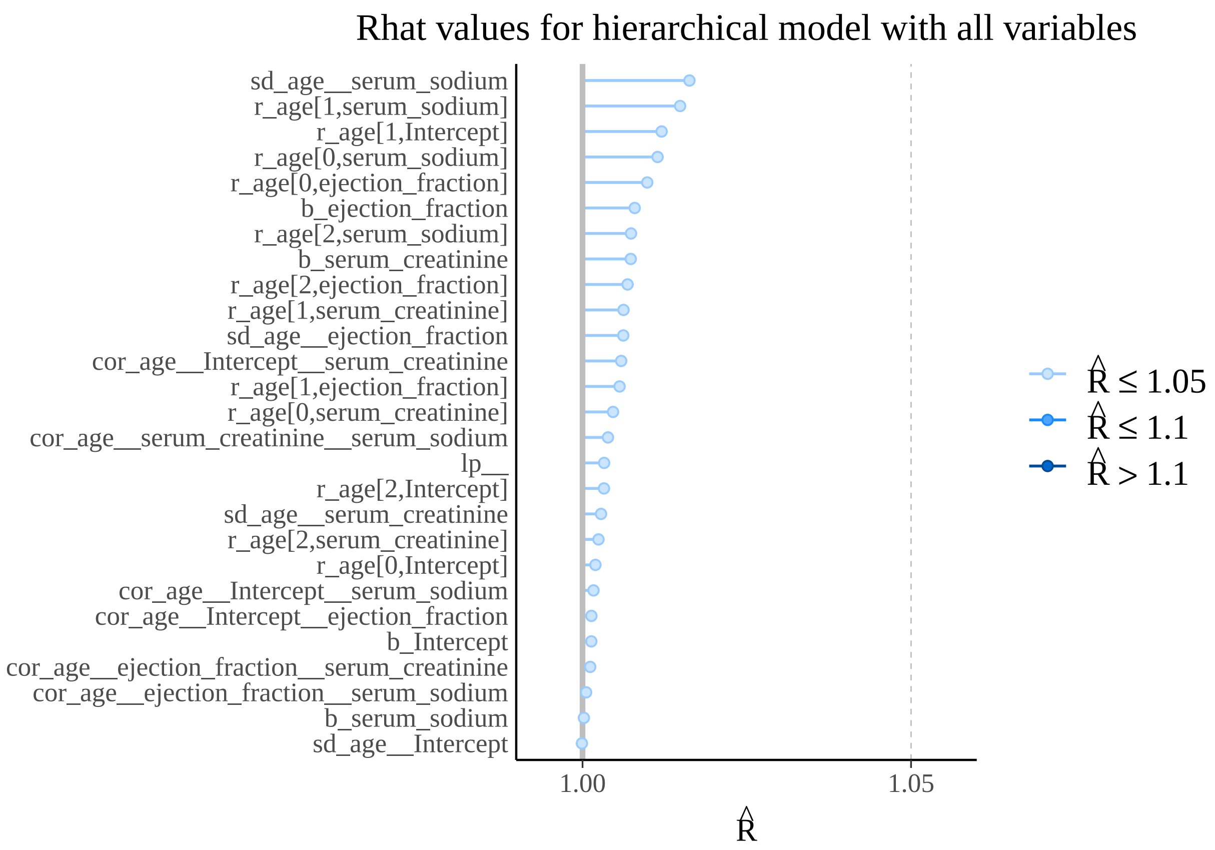
### 5.3.2 Summary for the linear model with selected variables

```
rhats <- rhat(fit.feature_selected)
color_scheme_set("brightblue")  # see help('color_scheme_set')
mcmc_rhat(rhats) + yaxis_text(hjust = 1) + ggtitle("Rhat values for model with selected variables") +
    theme(plot.title = element_text(hjust = 0.5))
```



### 5.3.3 Summary for the hierarchical model with all variables

```
rhats <- rhat(fit.hierarchical)
color_scheme_set("brightblue")  # see help('color_scheme_set')
mcmc_rhat(rhats) + yaxis_text(hjust = 1) + ggtitle("Rhat values for hierarchical model with all variabl
    theme(plot.title = element_text(hjust = 0.5))
```



From the model summaries, it is observable that $\hat{R}$ values for all models are around 1.0 which is below the threshold value of 1.05 as mentioned in Stan documentation. With this information, we can interpret that the chains have mixed well and the samples are reliable.

### 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

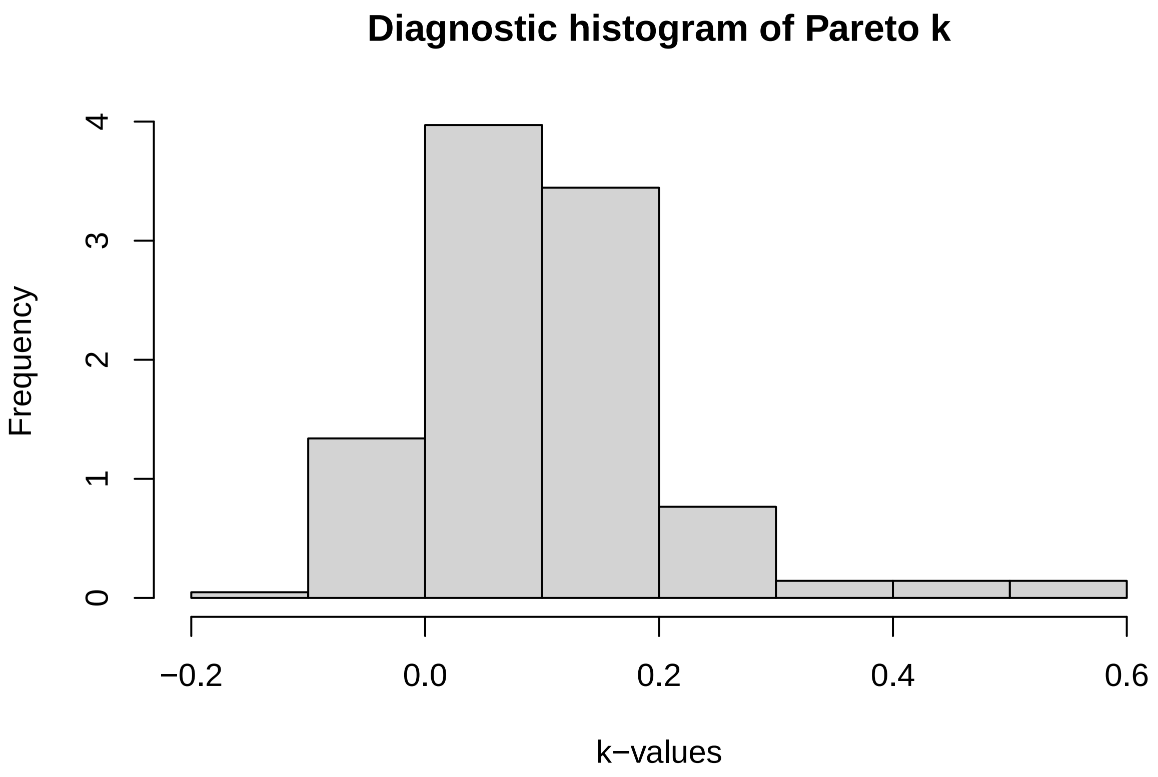
### 5.8 WIP: Predictive performance assessment (classification)

Function for predicting pointwise accuracy

```r
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)
}
```

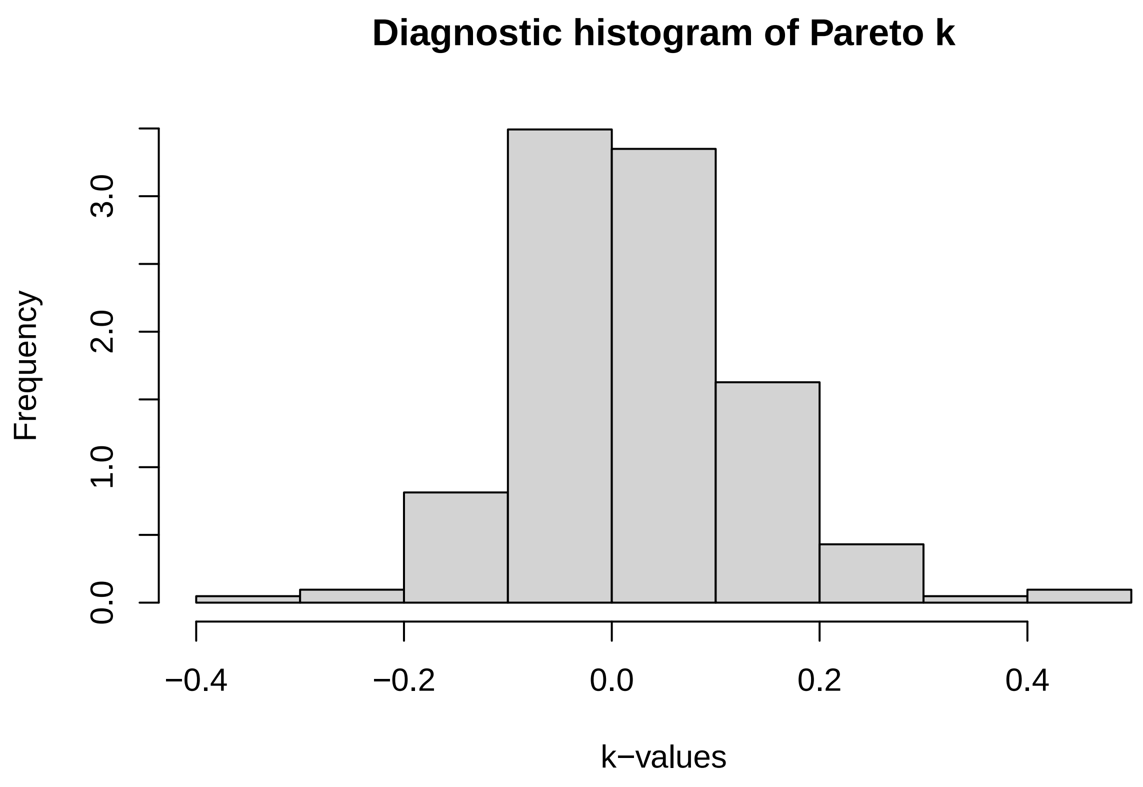
### 5.8.1 Full model

```
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)
```

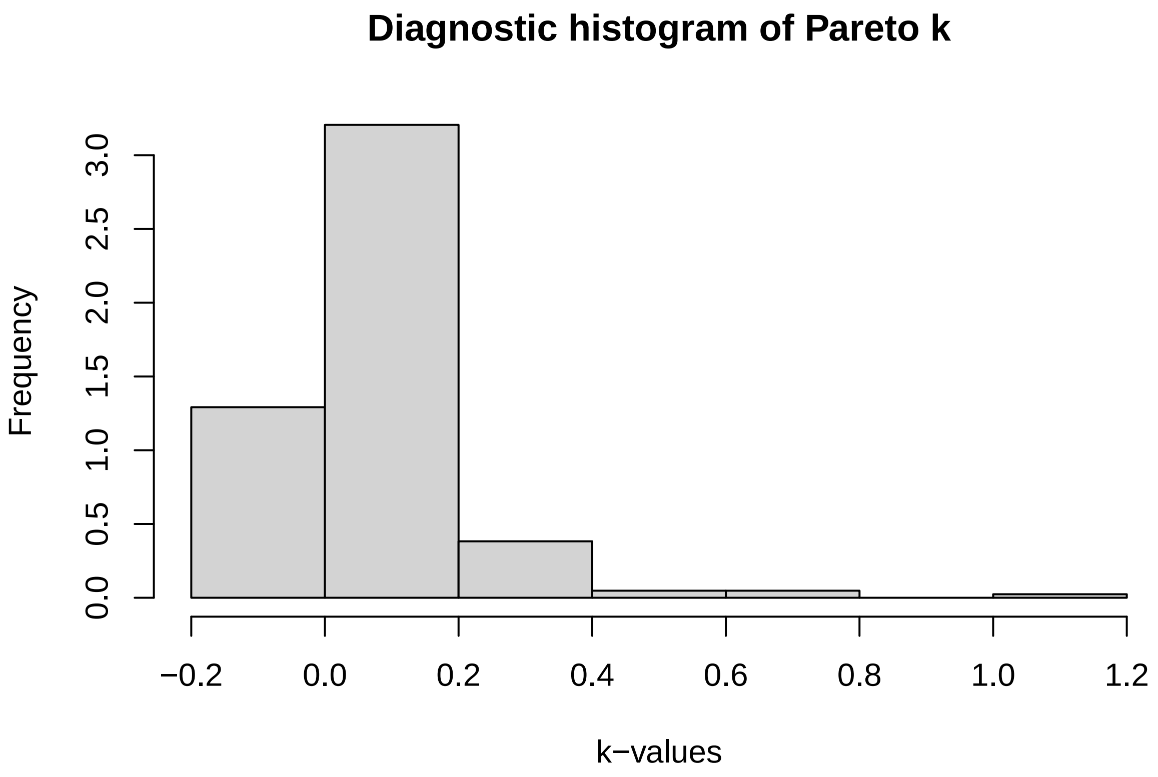


### 5.8.3 Hierarchical model

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

```
## Warning: Found 3 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)
```



#### 5.8.4

## 5.9 Prior sensitivity analysis (alternative prior tested)

## 5.10 Discussion of problems and further improvements

# 6 Conclusion

## 6.1 Self-reflection

As a group we have learned. . .

# Appendix

## A. Stan code of full model

```r
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 += normal_lpdf(b[1] | 0,50);
##   target += normal_lpdf(b[2] | 0,100);
##   target += normal_lpdf(b[3] | 0,100);
##   target += normal_lpdf(b[4] | 0,1000);
##   target += normal_lpdf(b[5] | .5, .5);
##   target += normal_lpdf(b[7] | .5, .5);
##   target += normal_lpdf(b[8] | .5, .5);
##   target += normal_lpdf(b[9] | .5, .5);
##   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

```r
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];
##     }
##   }
## }
```

## 6.2 D. Stan death time analysis model

```r
stancode(fit.weibull)
```

```
## // generated with brms 2.14.4
## functions {
## }
## data {
##   int<lower=1> N;  // total number of observations
##   vector[N] Y;  // response variable
##   int<lower=-1,upper=2> cens[N];  // indicates censoring
##   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
##   real<lower=0> shape;  // shape parameter
## }
## transformed parameters {
## }
## model {
##   // likelihood including all constants
##   if (!prior_only) {
##     // initialize linear predictor term
##     vector[N] mu = Intercept + Xc * b;
##     for (n in 1:N) {
##       // apply the inverse link function
##       mu[n] = exp(mu[n]) / tgamma(1 + 1 / shape);
##     }
##     for (n in 1:N) {
##     // special treatment of censored data
##       if (cens[n] == 0) {
##         target += weibull_lpdf(Y[n] | shape, mu[n]);
##       } else if (cens[n] == 1) {
##         target += weibull_lccdf(Y[n] | shape, mu[n]);
##       } else if (cens[n] == -1) {
##         target += weibull_lcdf(Y[n] | shape, mu[n]);
##       }
##     }
##   }
##   // priors including all constants
##   target += student_t_lpdf(Intercept | 3, 4.8, 2.5);
##   target += gamma_lpdf(shape | 0.01, 0.01);
## }
## generated quantities {
##   // actual population-level intercept
##   real b_Intercept = Intercept - dot_product(means_X, b);
## }
```

## 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.