**Appendix A – Explanation Imputation Methods**

In this supplementary material we will explain which values are required to be calculated in the training data and which R packages are used per implemented method. We will also explain step by step what we do for each method. We will focus specifically on JMI and CMI as mean imputation is relatively straightforward. In addition, we will shortly cover the requirements and step-by-step instructions for each evaluation method used. All code is added in appendix C.

*Joint modelling imputation*As stated JMI allows tailored imputations, making use of covariances between all predictors. More specifically, imputations are randomly sampled from a (multivariate) normal distribution that is conditioned on the observed predictor values. For binary variables, a logistic regression model is used to transform the drawn continuous values into discrete imputations.

To implement JMI we first have to calculate the expectation (mean) of all variables included in the data and save this in a single vector. Additionally, a covariance matrix of the data has to be saved in a separate object. We also save the class of each variable included in the data. On a patient-by-patient basis we extract which variables are missing and which are not missing. From the variables that are not missing we save the observed values in a separate vector. Then, using the *rcmvnorm* function in R, we estimate the conditional multivariate normal distribution using the provided expectations (mu), covariance matrix (sigma), dependent variables (i.e. names of the missing variable), the given non-missing variables and all observed values (35).

For example, consider a situation where we have two variables of interest (e.g. blood pressure) and (e.g. Body Mass Index). These variables have been fully observed in a previous cohort study, where we calculated their respective means and , their respective variance and , and their correlation . Consider now the encounter with a single new patient for which the Body Mass Index has been measured (i.e. is known), but for which the blood pressure cannot be retrieved (and therefore is missing). Assuming that BMI and blood pressure follow a bivariate Normal distribution, likely values for (given that is known) can be described by a Normal distribution with mean and variance where:

and

Hence, imputations for can simply be generated by drawing random samples from the distribution N( ). If only a single imputation is desired, the most likely value for is simply given by .

Consider now that is a binary variable (e.g. smoking) instead of a continuous variable. In this case, samples from N( ), denoted as , may not be appealing to be used as imputation because they are unlikely to be discrete (e.g. 0 or 1) and may even take negative values. For this reason, imputations for are generated according to . Note that for each imputation, a new value of need to be sampled.

The code that was used can be found in appendix C (function: *joint.MI()*). Note that the amount of imputed values is specified beforehand (i.e. n.imp). Also note that the mean vector *mu* and covariance matrix *sigma* of the training data can simply be obtained by applying the R functions *colMeans()* and *cov()* to the corresponding data frame. In case the training data are affected by missing values, R packages such as *mvnmle* can be used to obtain maximum likelihood estimates for the original mean and covariance.

*Conditional modelling imputation*  
To implement CMI we, before calculating other separate values, estimate each conditional model based on the training data. This entails iterating over all columns in the training data, specifying a conditional model (e.g. logistic) based on the type of dependent variable (e.g. binary). We save the conditional models in a list to be used in our imputation.

Note that we use the function *estimice* instead of *glm* when modeling continuous variables. This approach is analogous to the imputation of missing continuous variables in the R package *mice* when adopting the *mice.impute.norm* function. The *estimice* function is a least squares implementation of ridge regression, and can therefore better handle situations where training samples are relatively small.

The fitted regression models (one for each variable of interest) can then be used to generate imputations in new patients. In similar fashion to JMI, our implementation of CMI requires the means, covariance and data classes of the training data. The method first checks, on a patient-by-patient basis, how many variables are missing. We start with this distinction as single and multiple missing variable require a different approach.

In short: when a single predictor has a missing value, the fitted regression model of that predictor can directly be used to generate an imputed value. When multiple predictors have jointly missing values, the conditional models need to be combined through Markov Chain Monte Carlo sampling (41). Missing values are then first initialized to an arbitrary value, and updated iteratively by applying the procedure for a single missing value successively on each missing value.

More specifically: when the patient has a single missing variable, we specify the variables on which each model should be based, thus excluding the missing variable. If the missing variable is binary we use the regression coefficients of the relevant imputation model (i.e. as estimated in *conditional.estimation* function) and its corresponding covariance matrix to draw a random sample of imputation coefficients. Hereto, we use a multivariate T-distribution as implemented in the R function *rmvt* *(42)*. The imputation coefficients are then used to calculate a probability, which is then used with a Bernoulli distribution to draw an imputation for the missing value. This process of drawing the betas, calculating a probability and drawing a value from the Bernoulli distribution is done the amount of times we specify (i.e. n.imp).

When the missing variable is continuous, we use the Bayesian multiple imputation approach described by van Buurenand implemented in the R function *mice.impute.norm* *(13)*. This approach generates imputation coefficients by sampling from a posterior distribution that is based on the regression coefficients of the relevant imputation model (i.e. as estimated in *conditional.estimation* function) and standard non-informative priors. This adaptation was necessary to ensure that estimation uncertainty for the residual error variance is also taken into account when generating imputations.

When two or more variables are missing for a single patient, the conditional imputation models need to be used in conjunction to generate reliable imputations. Because each imputation model requires complete data on all but one variable, we first initialize each missing variable with a random starting value. To this purpose, we use the means and covariance of the training sample. Then, on a variable-by-variable basis, the starting values are updated by imputing them using all other (original or initiated) values. This process of updating randomly initiated values iterates over each missing variable and is then repeated for a specified number of times to also replace the updated values numerous times. This cyclic generation of updated values is necessary to ensure that the imputed variables depend on each other and the observed data, but no longer on their initial values. Updated values from the last iteration are then extracted and used as the imputed values. The process of initializing starting values, updating these values and extracting them is repeated for a prior specified amount (i.e. n.imp).

The code in appendix C (functions: *conditional.estimation()* and *conditional.MI()*) was used to implement conditional modelling imputation. Note that the object *model\_estimation* is a list containing the conditional imputation models for each variable, and can be obtained using the function *conditional.estimation()*.

***Evaluation measures***

Each method provided an three-dimensional array of the data, where the third dimension consisted of the prior specified amount of imputations (i.e. 50 in our analysis) for each of the missing variables. When calculating the RMSE we square the difference between the mean of those imputations and the true value, which gives us a vector of squared deviations. The root of the mean of that vector is the RMSE reported in this study. We calculated the coverage rate by first calculating a 95% confidence interval for each imputed predictor in the hold-out patient according to

Where xi is the ith imputed value (out of a total of 50), and t is a value from a two-sided t-distribution with 50 degrees of freedom. We then specify a binary indicator showing if the confidence interval included the true value. Taking the mean of the binary indicator gives us the percentage of confidence intervals containing the true value.

The code in appendix C was used to calculate both evaluation measures for a single missing predictor (function: *test\_single\_missing()*). Note that the object *knn1* is the exemplar dataset where all predictors are fully observed. To accommodate deriving the necessary population characteristics from the training data we completed any missing values in the UCC data using K-nearest neighbor imputation (KNN) (28). In addition the character *test\_var* specifies the variable for which” missing values are introduced in the Jack-knife procedure.

**Appendix B – Descriptive statistics before imputation**

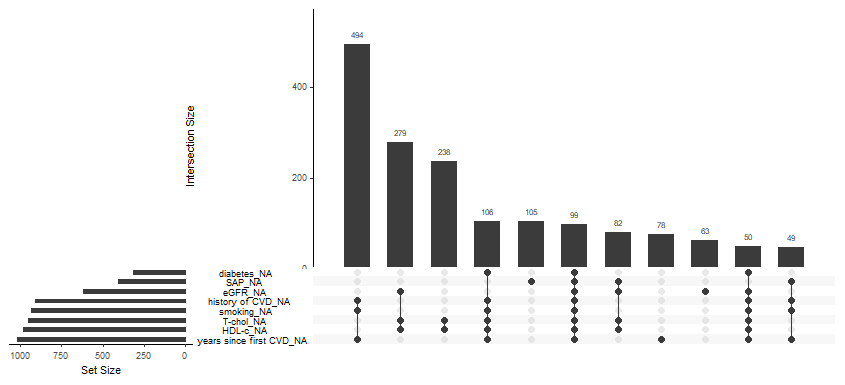
|  |  |  |
| --- | --- | --- |
|  | Mean (sd) or n/total (%)\* | % Missing |
| Age (years) | 61.7 (18.2) | 0.00 |
| Sex (1=female; 0=male) | 1987/3880 (51.2) | 0.00 |
| Smoking (1=yes; 0= no) | 363/2583 (14.05) | 24.07 |
| SBP (mmHg) | 141.49 (24.2) | 10.54 |
| TC (mmol/l) | 5.2 (1.4) | 24.54 |
| LDL-c (mmol/l) | 3.0 (1.2) | 26.01 |
| HDL-c (mmol/l) | 1.4 (0.4) | 25.39 |
| eGFR (mL/min/1.73m2) | 80.7 (25.6) | 15.98 |
| History of CVD (1=yes; 0= no) | 1063/1907 (55.7) | 23.45 |
| History of PAD (1=yes; 0= no) | 271/2699 (10.0) | 23.45 |
| History of CHD (1=yes; 0= no) | 472/2498 (18.9) | 23.45 |
| History of CHF (1=yes; 0= no) | 283/2687 (10.5) | 23.45 |
| History of CVA (1=yes; 0= no) | 449/2521 (17.8) | 23.45 |
| History of DM (1=yes; 0= no) | 607/2363 (25.6) | 23.45 |
| Polyvascular disease | 0.5 (0.8) | 23.45 |
| # of medications | 1.0 (1.9) | 27.24 |
| BP lowering medication (1=yes; 0= no) | 599/2224 (26.9) | 27.24 |
| Statin (1=yes; 0= no) | 395/2428 (16.3) | 27.24 |
| HbA1c (mmol/mol) | 40.7 (11.8) | 26.37 |
| Years since first CVD (years) | 3.8 (8.5) | 26.21 |
| Diabetes (1=yes; 0= no) | 755/2810 (26.9) | 8.12 |
| Diabetes duration (years) | 14.9 (12.0) | 86.11 |
| Pulse pressure (mmHg) | 61.7 (19.5) | 10.54 |

Legend – SBP: systolic blood pressure, TC: total cholesterol, LDL-c: low-density lipoprotein cholesterol, HDL-c: high-density lipoprotein cholesterol, eGFR: estimated glomerular filtration rate according to the CKD epi formula, CVD: cardiovascular disease, PAD: peripheral artery disease, CHD: coronary heart disease, CHF: chronic heart failure, CVA: cerebrovascular accident, DM: diabetes mellitus, BP: blood pressure, HbA1c: glycated hemoglobin. \* after KNN-imputation

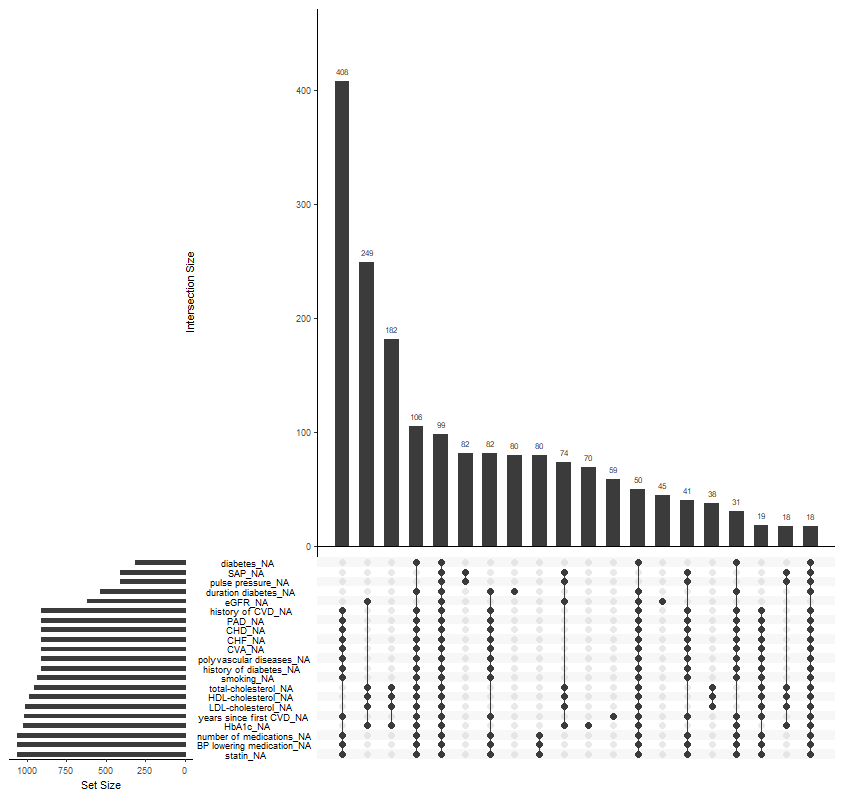
**Appendix C – Selection of variables**

Given that the interest of the study is to provide a method with which a prediction model is able to be used whilst missing predictor values are present, we looked at combinations of missing predictor values that are observed in real data (see below). This figure describes the most common missing intersections of predictor variables. No distinction concerning variable importance is made. All single missing predictor scenarios are included, regardless of their occurrence in real data, as such the apparent single scenarios in the figure below can be ignored.

Each of these intersections is used in the study as a possible scenario for which the imputation methods should realistically work well. For a combination of missing predictor values to be included in the study it should at least be apparent in >1% of patients. This resulted in the inclusion of eight distinct multiple missing predictor scenarios.



The next part of variable selection was identifying the auxiliary variables that are inextricably linked to any of the predictor variables. Using these variables in an attempt to impute their respective predictor value via JMI or CMI would overestimate their performance as they are highly reliant on the relationship between available variables and the missing predictor value to be imputed. As such it is important that these auxiliary variables are not available for information extraction when their respective predictor values are missing. The variables were identified using the clinical experience of the authors as well as by using visualizations of the various combinations of missing value scenarios in the complete data (see next figure). For example, it was noticed that pulse pressure, or SAP, were never exclusively missing.

The combinations identified are: (1) SAP and pulse pressure, (2) diabetes and diabetes duration, (3) history of CVD and history of PAD, CHD, CHF, CVA and polyvascular disease and (4) total cholesterol, HDL-cholesterol and LDL-cholesterol.

**Appendix D – R code**

The completed UCC data is available from *knn1*. This data frame was used to evaluate the various missing data scenarios as follows:

|  |
| --- |
| load("knn1.RData")  source("functions.r")  ########################################  # Simulation study for single missings #  ########################################  knn1.DM <- knn1  knn1.DM[,"m0\_duur\_diabetes\_quest"] <- NULL  results\_DM <- test\_single\_missing(data = knn1.DM, test\_var = "M0\_diabetes", seed = 13331)  rm(knn1.DM)  results\_DM  save.image(WD)  knn1.SBP <- knn1  knn1.SBP[,"m0\_pulse\_pressure"] <- NULL  results\_SBP <- test\_single\_missing(data = knn1.SBP, test\_var = "m0\_SAP", seed = 13332)  rm(knn1.SBP)  results\_SBP  save.image(WD)  results\_eGFR <- test\_single\_missing(data = knn1, test\_var = "m0\_Lab\_eGFR\_CKD\_EPI", seed = 13333)  results\_eGFR  save.image(WD)  knn1.histCVD <- knn1  knn1.histCVD[,c("m0\_History\_PAD","m0\_History\_CHD","m0\_History\_CHF",  "m0\_History\_CVA","m0\_polyvascular\_disease")] <- list(NULL)  results\_histCVD <- test\_single\_missing(data = knn1.histCVD, test\_var = "m0\_CardVascHistory", seed = 13334)  rm(knn1.histCVD)  results\_histCVD  save.image(WD)  results\_yrssinceCVD <- test\_single\_missing(data = knn1, test\_var = "M0\_year\_since\_first\_CVD", seed = 13335)  results\_smoking <- test\_single\_missing(data = knn1, test\_var = "m0\_Intox\_Smoking\_current", seed = 13336)  results\_yrssinceCVD  results\_smoking  save.image(WD)  knn1.TC <- knn1  knn1.TC[,c("m0\_Lab\_HDLchol","m0\_Lab\_LDLchol")] <- list(NULL)  results\_TC <- test\_single\_missing(data = knn1.TC, test\_var = "m0\_Lab\_Chol", seed = 13337)  rm(knn1.TC)  results\_TC  save.image(WD)  knn1.HDL <- knn1  knn1.HDL[,c("m0\_Lab\_Chol","m0\_Lab\_LDLchol")] <- list(NULL)  results\_HDL <- test\_single\_missing(data = knn1.HDL, test\_var = "m0\_Lab\_HDLchol", seed = 13338)  rm(knn1.HDL)  results\_HDL  save.image(WD)  ############################  # Results simulation study #  ############################  results\_DM  results\_SBP  results\_eGFR  results\_histCVD  results\_yrssinceCVD  results\_smoking  results\_TC  results\_HDL  ##########################################  # Simulation study for multiple missings #  ##########################################  knn1.pat1 <- knn1  knn1.pat1[,c("m0\_History\_PAD","m0\_History\_CHD","m0\_History\_CHF",  "m0\_History\_CVA","m0\_polyvascular\_disease")] <- list(NULL)  results\_pat1 <- test\_multiple\_missing(data = knn1.pat1, test\_var = c("m0\_CardVascHistory",  "M0\_year\_since\_first\_CVD",  "m0\_Intox\_Smoking\_current"),  seed = 2408001)  rm(knn1.pat1)  results\_pat1  save.image(WD)  knn1.pat2 <- knn1  knn1.pat2[,c("m0\_Lab\_LDLchol")] <- NULL  results\_pat2 <- test\_multiple\_missing(data = knn1.pat2, test\_var = c("m0\_Lab\_eGFR\_CKD\_EPI",  "m0\_Lab\_Chol",  "m0\_Lab\_HDLchol"),  seed = 2408002)  rm(knn1.pat2)  results\_pat2  save.image(WD)  knn1.pat3 <- knn1  knn1.pat3[,c("m0\_Lab\_LDLchol")] <- NULL  results\_pat3 <- test\_multiple\_missing(data = knn1.pat3, test\_var = c("m0\_Lab\_Chol",  "m0\_Lab\_HDLchol"),  seed = 2408003)  rm(knn1.pat3)  results\_pat3  save.image(WD)  knn1.pat4 <- knn1  knn1.pat4[,c("m0\_Lab\_LDLchol","m0\_duur\_diabetes\_quest","m0\_History\_PAD","m0\_History\_CHD","m0\_History\_CHF",  "m0\_History\_CVA","m0\_polyvascular\_disease")] <- list(NULL)  results\_pat4 <- test\_multiple\_missing(data = knn1.pat4, test\_var = c("M0\_diabetes",  "m0\_CardVascHistory",  "M0\_year\_since\_first\_CVD",  "m0\_Intox\_Smoking\_current",  "m0\_Lab\_Chol",  "m0\_Lab\_HDLchol"),  seed = 2408004)  rm(knn1.pat4)  results\_pat4  save.image(WD)  knn1.pat5 <- knn1  knn1.pat5[,c("m0\_Lab\_LDLchol","m0\_duur\_diabetes\_quest","m0\_History\_PAD","m0\_History\_CHD","m0\_History\_CHF",  "m0\_History\_CVA","m0\_polyvascular\_disease","m0\_pulse\_pressure")] <- list(NULL)  results\_pat5 <- test\_multiple\_missing(data = knn1.pat5, test\_var = c("M0\_diabetes",  "m0\_SAP",  "m0\_Lab\_eGFR\_CKD\_EPI",  "m0\_CardVascHistory",  "M0\_year\_since\_first\_CVD",  "m0\_Intox\_Smoking\_current",  "m0\_Lab\_Chol",  "m0\_Lab\_HDLchol"),  seed = 2408005)  rm(knn1.pat5)  results\_pat5  save.image(WD)  knn1.pat6 <- knn1  knn1.pat6[,c("m0\_Lab\_LDLchol","m0\_pulse\_pressure")] <- list(NULL)  results\_pat6 <- test\_multiple\_missing(data = knn1.pat6, test\_var = c("m0\_SAP",  "m0\_Lab\_eGFR\_CKD\_EPI",  "m0\_Lab\_Chol",  "m0\_Lab\_HDLchol"),  seed = 2408006)  rm(knn1.pat6)  results\_pat6  save.image(WD)  knn1.pat7 <- knn1  knn1.pat7[,c("m0\_Lab\_LDLchol","m0\_duur\_diabetes\_quest","m0\_History\_PAD","m0\_History\_CHD","m0\_History\_CHF",  "m0\_History\_CVA","m0\_polyvascular\_disease","m0\_pulse\_pressure")] <- list(NULL)  results\_pat7 <- test\_multiple\_missing(data = knn1.pat7, test\_var = c("M0\_diabetes",  "m0\_SAP",  "m0\_CardVascHistory",  "M0\_year\_since\_first\_CVD",  "m0\_Intox\_Smoking\_current",  "m0\_Lab\_Chol",  "m0\_Lab\_HDLchol"),  seed = 2408007)  rm(knn1.pat7)  results\_pat7  save.image(WD)  knn1.pat8 <- knn1  knn1.pat8[,c("m0\_History\_PAD","m0\_History\_CHD","m0\_History\_CHF",  "m0\_History\_CVA","m0\_polyvascular\_disease","m0\_pulse\_pressure")] <- list(NULL)  results\_pat8 <- test\_multiple\_missing(data = knn1.pat8, test\_var = c("m0\_SAP",  "m0\_CardVascHistory",  "M0\_year\_since\_first\_CVD",  "m0\_Intox\_Smoking\_current"),  seed = 2408008)  rm(knn1.pat8)  results\_pat8  save.image(WD)  ############################  # Results simulation study #  ############################  results\_pat1  results\_pat2  results\_pat3  results\_pat4  results\_pat5  results\_pat6  results\_pat7  results\_pat8  ################  # Impact study #  ################  library(matrixcalc)  knnCVD <- subset(knn1, m0\_CardVascHistory == T)  knnCVD$m0\_pulse\_pressure <- NULL  results.m0SAP <- data.frame(y.orig = rep(NA, nrow(knnCVD)),  y.meanimp = rep(NA, nrow(knnCVD)),  y.jointimp = rep(NA, nrow(knnCVD)),  y.condimp = rep(NA, nrow(knnCVD)))  pb <- txtProgressBar(min = 0, max = nrow(results.m0SAP), style = 3)  for (i in 1:nrow(knnCVD)) {  setTxtProgressBar(pb, i)  dat.train <- knnCVD[-i,]  test\_case <- knnCVD[i,]  results.m0SAP$y.orig[i] <- test\_case$m0\_SAP  test\_case$m0\_SAP <- NA  # Estimate the necessary imputation models/parameters  mu <- sapply(dat.train, mean)  sigma <- cov(dat.train)  n.imp <- 50  data\_classes <- sapply(knn1, class)  if (!is.positive.definite(sigma)) {  sigma <- sigma + diag(ncol(sigma))\*0.01  }  cond\_model <- conditional.estimation(training\_data = dat.train, skip = (!colnames(dat.train) %in% "m0\_SAP"))  # Perform the imputation  imp.joint <- joint.MI(data = test\_case,  mu = mu,  sigma = sigma,  data\_classes = data\_classes,  n.imp = n.imp)  imp.cond <- conditional.MI(data = test\_case,  model\_estimation = cond\_model,  mu = mu,  sigma = sigma,  data\_classes = data\_classes,  n.imp = n.imp)  # Save results  results.m0SAP$y.meanimp[i] <- mu["m0\_SAP"]  results.m0SAP$y.jointimp[i] <- mean(imp.joint[1, "m0\_SAP",])  results.m0SAP$y.condimp[i] <- mean(imp.cond[1, "m0\_SAP",])  }  close(pb)  ########################  # Impact study Results #  ########################  results.m0SAP  # Mean imputation  with(results.m0SAP, sum(y.orig>=140 & y.meanimp>=140))  with(results.m0SAP, sum(y.orig>=140 & y.meanimp<140))  with(results.m0SAP, sum(y.orig<140 & y.meanimp>=140))  with(results.m0SAP, sum(y.orig<140 & y.meanimp<140))  # Joint Modeling Imputation  with(results.m0SAP, sum(y.orig>=140 & y.jointimp>=140))  with(results.m0SAP, sum(y.orig>=140 & y.jointimp<140))  with(results.m0SAP, sum(y.orig<140 & y.jointimp>=140))  with(results.m0SAP, sum(y.orig<140 & y.jointimp<140))  # Condtional Modeling Imputation  with(results.m0SAP, sum(y.orig>=140 & y.condimp>=140))  with(results.m0SAP, sum(y.orig>=140 & y.condimp<140))  with(results.m0SAP, sum(y.orig<140 & y.condimp>=140))  with(results.m0SAP, sum(y.orig<140 & y.condimp<140))  ##############  # Prediction #  ##############  data\_eGFR <- data\_imputation(data = knn1, test\_var = "m0\_Lab\_eGFR\_CKD\_EPI", seed = 13336)  prediction\_eGFR <- as.data.frame(prediction\_single\_missing(data\_eGFR, seed=13338))  knn1.pat6 <- knn1  knn1.pat6[,c("m0\_Lab\_LDLchol","m0\_pulse\_pressure")] <- list(NULL) #linked var  data\_pat6 <- data\_imputation(data = knn1.pat6,  test\_var = c("m0\_SAP",  "m0\_Lab\_eGFR\_CKD\_EPI",  "m0\_Lab\_Chol",  "m0\_Lab\_HDLchol"),  seed = 2408006)  prediction\_pat6 <- as.data.frame(prediction\_single\_missing(data\_pat6,  test\_var = c("m0\_SAP",  "m0\_Lab\_eGFR\_CKD\_EPI",  "m0\_Lab\_Chol",  "m0\_Lab\_HDLchol"),  seed=2408010)) |

The content of the file *functions.r* is as follows:

|  |
| --- |
| ###########  # Authors #  ###########  # Steven Nijman  # Jeroen Hoogland  # Thomas Debray  ########################  # Package Requirements #  ########################  require(condMVNorm) # version 2015.2-1  require(mice) # version 3.6.0  ########################  # Function to retrieve columns with binary variables  binary <- function(data){  binnames <- sapply(data, function(x) length(unique(x[complete.cases(x)])) == 2)  binnames <- names(binnames)[binnames]  binnames  }  # Extra function to facilitate compatibility with glm objects  coef.estimice <- function (object){  return (object$c)  }  # Extra function to facilitate compatibility with glm objects  vcov.estimice <- function (object) {  return(object$v)  }  # Estimate conditional model for each variable in the training data  # We can skip certain models (e.g. in case a variable does not contain missings)  # Note that this function requires training data without missing values. If missing  # values are present, the conditional models need to be estimated using a Gibbs  # sampler (similar to the imputation process)  conditional.estimation <- function(training\_data,  skip = rep(FALSE, ncol(training\_data)) # By default, estimate all the conditional models  ) {  if (length(skip)!=ncol(training\_data)) {  stop ("The size of argument 'skip' should match the number of columns in 'training\_data'")  }  model.estimation <- list()  binnames <- binary(training\_data)  for(i in 1:ncol(training\_data)) {  if (!skip[i]) {  formula <- paste(names(training\_data[-i]), collapse = " + ")  formula <- paste(c(names(training\_data[i]), formula), collapse = " ~ ")  if(names(training\_data)[i] %in% binnames) {  model.estimation[[i]] <- glm(formula, training\_data, family="binomial")  } else {  # We use ridge regression as adopted in mice to facilitate implementation of  # imputation processes. An additional advantage is that the ridge penalty  # accommodates for some estimation problems in sparse datasets  X <- cbind(1, as.matrix(training\_data[,-i]))  fit <- tryCatch({  estimice(x = X, y = training\_data[,i])  }, error = function(e) {  estimice(x = X, y = training\_data[,i], ls.meth = "ridge")  }, finally = {  })  fit$df.residual <- fit$df # Facilitate compatibility with glm objects  class(fit) <- "estimice" # Facilitate compatibility with glm objects  model.estimation[[i]] <- fit  }  } else {  model.estimation[[i]] <- NULL  }  }  return(model.estimation)  }  joint.MI <- function(data, mu, sigma, data\_classes, n.imp) {  dat.imputed <- array(rep(as.matrix(data), n.imp), dim=c(nrow(data),ncol(data),n.imp))  colnames(dat.imputed) <- colnames(data)  missing.col <- c()  for (i in 1:nrow(data)) {  if(any(is.na(data[i,]))){  x <- data[i,] #match names of x and mu.  dep <- names(x[which(is.na(x))])  given <- names(x[which(!is.na(x))])  missing.col <- which(is.na(x))  x.obs <- as.numeric(x[which(names(x) %in% given)])    condMVN <- rcmvnorm(n=n.imp, mean=mu, sigma=sigma, dep=dep, given=given, X=x.obs)  }    for(l in missing.col){  index <- which(missing.col == l)  if(data\_classes[l]=="logical"){  condMVN[,index] <- ifelse(condMVN[,index] > 1, 1, condMVN[,index])  condMVN[,index] <- ifelse(condMVN[,index] < 0, 0, condMVN[,index])  condMVN[,index] <- as.logical(rbinom(n = n.imp, size = 1, prob = condMVN[,index]))  }  dat.imputed[i, l, ] <- condMVN[,index]  }  }  return(dat.imputed)  }  conditional.MI <- function(data, # Data frame with the patient data  model\_estimation, # List of conditional imputation models  mu = rep(0, ncol(data)), # Mean vector to initialize Gibbs sampler  sigma = diag(rep(1000, ncol(data))), # Covariance matrix to initialize Gibbs sampler  data\_classes,  n.imp, # Number of required imputed datasets  maxit = 15) # set maximum iterations for convergence imputations  {    dat.imputed <- array(rep(as.matrix(data), n.imp), dim=c(nrow(data),ncol(data),n.imp))  colnames(dat.imputed) <- colnames(data)    # get variable for which models were fitted with family "binomial"  binnames <- colnames(dat.imputed)[unlist(lapply(model\_estimation, function(x) x$family$family == "binomial"))]    # each row/patient is taken separately  for(i in 1:nrow(data)) {  # identify missing variables  missing.col <- which(is.na(data[i,]))    # No Gibbs sampler needed if only 1 patient has missing values  if(length(missing.col) == 1) {  dat.imputed[i,missing.col,1:n.imp] <- conditional.MI.single(test\_case = data[i,],  model\_estimation,  mu,  sigma,  data\_classes,  n.imp)  } else if(length(missing.col) > 1) {    # initialize vector for patient -> these will for be the concurrent imputations  gibbsdata <- lapply(data\_classes, vector, length = 1)    # iterate over till all multiple imputations are done for this patient  for(l in 1:n.imp) {  # Generate initial draws for imputation & iterate from 1 to 23 variables  for(j in 1:length(data)) {  gibbsdata[[j]][1] <- data[i,j]  # itial value = draw from multivariate normal distribution  gibbsdata[[j]][2] <- ifelse(j %in% missing.col, mvtnorm::rmvnorm(1, mean=mu, sigma=sigma)[j], gibbsdata[[j]][1])  }  # convert vector to dataframe  temp\_data <- as.data.frame(gibbsdata)    # first 2 iterations already done in intialization, iterate till max iterations for convergence (15)    # Iterate over the imputation cycles  for (iter in 3:maxit) {  temp\_data[iter,] <- temp\_data[iter-1,]    # Iterate ver the different variables with missing values  # Each time, use the most recently imputed value from the remaining variables    for(k in 1:length(missing.col)){  # Extract last available data  test\_case <- temp\_data[iter,]  test\_case[missing.col[k]] <- NA  test\_case[missing.col[k]] <- conditional.MI.single(test\_case = test\_case,  model\_estimation,  mu,  sigma,  data\_classes,  n.imp = 1)  # fill temporary data with last iteration  temp\_data[iter,] <- test\_case  }  }    # Save most recent imputation as imputed dataset 'l' for patient 'i',  dat.imputed[i,missing.col,l] <- unlist(temp\_data[maxit, missing.col])  }  } # end if-structure of the Gibbs sampler  } # end iteration over the patients  return(dat.imputed)  }  # Multiple Imputation of a single missing value for a single patient.  conditional.MI.single <- function(test\_case, model\_estimation, mu, sigma, data\_classes, n.imp) {    missing.col <- which(is.na(test\_case))  if (length(missing.col) > 1) {  stop("This function is only allowed for imputation of a single missing value.")  }    out <- rep(NA, n.imp)    # observed variables (explanatory variables)  predictors <- paste(names(test\_case[,-missing.col]), collapse = " + ")    # model matrix of missing values explained by explanatory variables  pred.data <- model.matrix(formula(paste("~", predictors)), data = test\_case)    if(data\_classes[missing.col]=="logical") {    # Directly draw all 'n.imp' draws for the regression coefficients  beta\_star <- rmvt(n = n.imp,  delta = coef(model\_estimation[[missing.col]]),  sigma = vcov(model\_estimation[[missing.col]]),  df = model\_estimation[[missing.col]]$df.residual)    prob <- rep(NA, n.imp)  for(j in 1:n.imp) {  prob[j] <- 1/(1+exp(-as.numeric(pred.data[1,]) %\*% beta\_star[j,]))  }  out <- rbinom(n = n.imp, size = 1, prob = prob)  } else {  # In case we are dealing with the imputation of a continuous variable, we will use the same functionalities  # as mice.impute.norm, which corresponds to Bayesian Linear Regression. The code below is adapted from the  # function mice::.norm.draw    p <- model\_estimation[[missing.col]] # This should be an object of class 'estimice'    sigma\_star <- sqrt(sum((p$r)^2)/rchisq(n.imp, p$df))  for(j in 1:n.imp) {  beta\_star <- p$c + (t(chol(micesym(p$v))) %\*% rnorm(length(p$c))) \* sigma\_star[j]  out[j] <- pred.data[1,] %\*% beta\_star + rnorm(1) \* sigma\_star[j]  }  }  return(out)  }  # Function extracted from the mice package  micesym <- function(x) {  (x + t(x))/2  }  joint.imputation.cv <- function(training\_data, test\_case, n.imp = 50, ...) {  mu <- sapply(training\_data, mean)  sigma <- cov(training\_data)  data\_classes <- sapply(training\_data, class)  joint\_model <- joint.MI(data=test\_case,mu, sigma, data\_classes, n.imp)  class(joint\_model) <- "joint.imputation.cv"  return(joint\_model)  }  conditional.imputation.cv <- function(training\_data, test\_case, n.imp = 50, ...) {  mu <- sapply(training\_data, mean)  sigma <- cov(training\_data)  data\_classes <- sapply(training\_data, class)    model\_estimation <- conditional.estimation(training\_data)  conditional\_model <- conditional.MI(test\_case, model\_estimation, mu, sigma, data\_classes, n.imp)  class(conditional\_model) <- "conditional.imputation.cv"  return(conditional\_model)  }  cv.function <- function(data, patterns, ...) {  # Create n training sets (n=3880) and n model estimations and imputations with each training set for each patient  results.joint = array(NA, dim=c(nrow(data),ncol(data), length(patterns)))  results.conditional = array(NA, dim=c(nrow(data),ncol(data),length(patterns)))  results.variance = array(NA, dim=c(2,ncol(data),length(patterns)))  results <- list(results.joint, results.conditional, results.variance)      data\_classes <- sapply(data, class)    pb <- txtProgressBar(min = 0, max = nrow(data), style = 3)    for(i in 1:nrow(data)) {  setTxtProgressBar(pb, i)    training\_data <- data[-i,]    # Estimate condtional imputation models  model\_conditional <- conditional.estimation(training\_data)    # Estimate parameters joint imputation  mu <- sapply(training\_data, mean)  sigma <- cov(training\_data)    for(j in seq\_along(patterns)) {  test\_case <- data[i,]  y <- as.numeric(unlist(patterns[[j]]))  test\_case[,y] <- NA  #ifelse(data\_classes[y]=="logical",as.logical(NA),as.numeric(NA))    imp1 <- joint.MI(data = test\_case, mu = mu, sigma = sigma, data\_classes = data\_classes, n.imp = 50)  imp2 <- conditional.MI(data = test\_case, model\_conditional, mu, sigma, data\_classes, n.imp=50)    for(k in 1:ncol(data)) {  if(k %in% y) {  results[[1]][i,k,j] <- mean(imp1[1, k,])  results[[2]][i,k,j] <- mean(imp2[1, k,])  results[[3]][1,k,j] <- var(imp1[1,k,])  results[[3]][2,k,j] <- var(imp2[1,k,])  }  }  }  }  close(pb)  return(results)  }  # Perform jack-knife validation for the imputation of a single missing value  # Author: Thomas Debray  test\_single\_missing<- function(data = knn1,  test\_var = "m0\_Intox\_Smoking\_current",  seed = NA,  n.imp = 50,  ...)  {  biassq.meanimp <- biassq.joint <- biassq.cond <- cov.joint <- cov.cond <- rep(NA, nrow(data))    if (!test\_var %in% colnames(data)) {  stop(paste("Test variable ", test\_var, "not found!"))  }    if (!is.na(seed)) {  set.seed(seed)  }    pb <- txtProgressBar(min = 0, max = nrow(data), style = 3)  for (i in 1:nrow(data)) {  setTxtProgressBar(pb, i)    training\_data <- data[-i,]  test\_case <- data[i,]    y\_ref <- as.numeric(test\_case[,test\_var])  test\_case[,test\_var]<- NA    data\_classes <- sapply(training\_data, class)    # Estimate the necessary imputation models/parameters  mu <- sapply(training\_data, mean)  sigma <- cov(training\_data)  cond\_model <- conditional.estimation(training\_data, skip = (!colnames(data) %in% test\_var))    # Perform the imputation  imp.joint <- joint.MI(data = test\_case,  mu = mu,  sigma = sigma,  data\_classes = data\_classes,  n.imp = n.imp)  imp.cond <- conditional.MI(data = test\_case,  model\_estimation = cond\_model,  mu = mu,  sigma = sigma,  data\_classes = data\_classes,  n.imp = n.imp)    # Extract the average imputation  pred.joint <- mean(imp.joint[1, test\_var,])  pred.cond <- mean(imp.cond[1, test\_var,])    # Assess Mean Square Error  biassq.meanimp[i] <- (mean(training\_data[,test\_var]) - y\_ref)\*\*2  biassq.joint[i] <- (pred.joint - y\_ref)\*\*2  biassq.cond[i] <- (pred.cond - y\_ref)\*\*2    # Assess coverage  tval <- abs(qt(0.05/2, 50))  ub <- pred.joint + tval \* sd(imp.joint[1, test\_var,])  lb <- pred.joint - tval \* sd(imp.joint[1, test\_var,])  cov.joint[i] <- ifelse(lb < y\_ref && y\_ref < ub, 1, 0)    ub <- pred.cond + tval \* sd(imp.cond[1, test\_var,])  lb <- pred.cond - tval \* sd(imp.cond[1, test\_var,])  cov.cond[i] <- ifelse(lb < y\_ref && y\_ref < ub, 1, 0)  }  close(pb)    # Calculate mean square error for joint modeling imputation and for conditional modeling imputation  return(list(mse\_meanimp = mean(biassq.meanimp),  mse\_jointMI = mean(biassq.joint),  mse\_condMI = mean(biassq.cond),  cov\_jointMI = mean(cov.joint),  cov\_condMI = mean(cov.cond)))  }  # Perform jack-knife validation for the imputation of multiple missing values  # Author: Thomas Debray  test\_multiple\_missing<- function(data = knn1,  test\_var = c("m0\_Intox\_Smoking\_current", "m0\_SAP"),  seed = NA,  ...)  {  biassq.meanimp <- biassq.joint <- biassq.cond <- cov.joint <- cov.cond <- array(NA, dim=c(nrow(data), length(test\_var)))  colnames(biassq.meanimp) <- colnames(biassq.joint) <- colnames(biassq.cond) <- colnames(cov.joint) <- colnames(cov.cond) <- test\_var    pb <- txtProgressBar(min = 0, max = nrow(data), style = 3)    if (!is.na(seed)) {  set.seed(seed)  }    for (i in 1:nrow(data)) {  setTxtProgressBar(pb, i)    training\_data <- data[-i,]  test\_case <- data[i,]    y\_ref <- as.numeric(test\_case[,test\_var])  test\_case[,test\_var]<- NA    imp.joint <- joint.imputation.cv(training\_data, test\_case)  imp.cond <- conditional.imputation.cv(training\_data, test\_case)    pred.joint <- apply(imp.joint[1, test\_var,], 1, mean)  pred.cond <- apply(imp.cond[1, test\_var,], 1, mean)    biassq.meanimp[i,] <- (colMeans(training\_data[,test\_var]) - y\_ref)\*\*2  biassq.joint[i,] <- (pred.joint - y\_ref)\*\*2  biassq.cond[i,] <- (pred.cond - y\_ref)\*\*2    # Assess coverage  tval <- abs(qt(0.05/2, 50))  ub <- pred.joint + tval \* apply(imp.joint[1, test\_var,], 1, sd)  lb <- pred.joint - tval \* apply(imp.joint[1, test\_var,], 1, sd)  ubcond <- pred.cond + tval \* apply(imp.cond[1, test\_var,], 1, sd)  lbcond <- pred.cond - tval \* apply(imp.cond[1, test\_var,], 1, sd)  for (j in 1:length(test\_var)) {  cov.joint[i,j] <- ifelse(lb[j] < y\_ref[j] && y\_ref[j] < ub[j], 1, 0)  cov.cond[i,j] <- ifelse(lbcond[j] < y\_ref[j] && y\_ref[j] < ubcond[j], 1, 0)  }  }  close(pb)    # Calculate mean square error for joint modeling imputation and for conditional modeling imputation  return(list(mse\_meanimp = colMeans(biassq.meanimp),  mse\_jointMI = colMeans(biassq.joint),  mse\_condMI = colMeans(biassq.cond),  cov\_jointMI = colMeans(cov.joint),  cov\_condMI = colMeans(cov.cond)))  }  # Extract imputed data to run prediction model on  data\_imputation <- function(data = knn1,  test\_var = "m0\_Intox\_Smoking\_current",  seed = NA,  n.imp = 50,  ...)  {  imp.joint <- imp.cond <- array(NA, dim=c(nrow(data), ncol(data), n.imp))  imp.mean <- array(NA, dim=c(nrow(data), ncol(data)))  colnames(imp.joint) <- colnames(imp.cond) <- colnames(imp.mean) <- colnames(data)    if (!is.na(seed)) {  set.seed(seed)  }    pb <- txtProgressBar(min = 0, max = nrow(data), style = 3)  for (i in 1:nrow(data)) {  setTxtProgressBar(pb, i)    training\_data <- data[-i,]  test\_case <- data[i,]    y\_ref <- as.numeric(test\_case[,test\_var])  test\_case[,test\_var]<- NA    data\_classes <- sapply(training\_data, class)    # Estimate the necessary imputation models/parameters  mu <- sapply(training\_data, mean)  sigma <- cov(training\_data)  cond\_model <- conditional.estimation(training\_data, skip = (!colnames(data) %in% test\_var))    # Perform the imputation  imp.joint[i,,] <- joint.MI(data = test\_case,  mu = mu,  sigma = sigma,  data\_classes = data\_classes,  n.imp = n.imp)  imp.cond[i,,] <- conditional.MI(data = test\_case,  model\_estimation = cond\_model,  mu = mu,  sigma = sigma,  data\_classes = data\_classes,  n.imp = n.imp)  imp.mean[i,] <- as.numeric(test\_case)  if(length(test\_var) > 1) {  imp.mean[i,test\_var] <- as.numeric(colMeans(data[,test\_var]))  } else {  imp.mean[i,test\_var] <- mean(data[,test\_var])  }    }  return(list(imp.joint,imp.cond,imp.mean))  }  # Calculate SMART prediction model  SMART <- function(data = data, ...) {  data <- as.data.frame(data)  prediction\_data <- rep(NA, nrow(data))  A <- -0.085\*data$m0\_Age+0.00105\*(data$m0\_Age^2)+0.156\*data$m0\_Sex+0.262\*data$m0\_Intox\_Smoking\_current+  0.00429\*data$m0\_SAP+0.223\*data$M0\_diabetes+0.406\*data$m0\_History\_CVA+0.283\*data$m0\_History\_PAD+  0.0229\*data$m0\_polyvascular\_disease-0.426\*data$m0\_Lab\_HDLchol+0.0959\*data$m0\_Lab\_Chol-  0.0532\*data$m0\_Lab\_eGFR\_CKD\_EPI+0.000306\*(data$m0\_Lab\_eGFR\_CKD\_EPI^2)  for(i in 1:nrow(data)) {  prediction\_data[i] <- (1-0.81066^exp(A[i]+2.099))\*100  ten\_year\_risk <- mean(prediction\_data)  }  return(ten\_year\_risk)  }  # Combine SMART prediction model and imputed data to make predictions on imputed data  prediction\_single\_missing <- function(data = data,  real\_data = knn1,  test\_var = "m0\_Intox\_Smoking\_current",  seed = NA,  n.imp = 50,  ...)  {  cur\_data\_joint <- cur\_data\_cond <- prediction\_joint <- prediction\_cond <- prediction\_mean <- prediction\_real <- rep(NA, n.imp)    if (!is.na(seed)) {  set.seed(seed)  }    pb <- txtProgressBar(min = 0, max = nrow(data[1][[1]]), style = 3)  for(i in 1:nrow(data[1][[1]])) {  setTxtProgressBar(pb, i)  real\_data <- data[i,]  joint\_data <- t(data[1][[1]][i,,])  cond\_data <- t(data[2][[1]][i,,])  mean\_data <- t(data[3][[1]][i,])    prediction\_real[i] <- SMART(real\_data)  prediction\_joint[i] <- SMART(joint\_data)  prediction\_cond[i] <- SMART(cond\_data)  prediction\_mean[i] <- SMART(mean\_data)  }  close(pb)    return(list(SMART\_real = prediction\_real,  SMART\_joint = prediction\_joint,  SMART\_cond = prediction\_cond,  SMART\_mean = prediction\_mean))  } |