COVID-PRECISE - Multiple imputation & analysis template

Valentijn de Jong and Thomas Debray

May 2021

Contents

Introduction

This document can serve as a template for validating a model in a data set with missing data. We apply multiple imputation with mice, validate a model by estimating five performance measures, and then show how they can be combined with Rubin's rules.

We assume that the integrity of the data have been thoroughly checked. Variable names will need to be adapted before using the functions below. Further, variable definitions need to be compared with the variable definitions in the selected models. For instance, the Hu score expects D-dimer in Ferritin equivalent units (FEU), while at many hospitals it is measured in D-dimer units (DDU).

We use a mock data set as an example, to show how the methods can be applied.

Software

This code was developed using R version 4.1.1. Version 4 of R has been a major update, that alters the behavior of many existing packages. Please use version 4.0.0 or later. Open RStudio and verify if your installed version is up-to-date:

```
R.Version()$version.string
```

We will use the (inverse) logit function multiple times:

```
inv_logit <- function(x) {  1/(1+exp(-x)) }
logit <- function(p) log(p / (1 - p))
```

For the imputation, we use the mice package. We use pROC for estimating model discrimination, and metamisc for estimating the SE for the observed/expected ratio. For the calibration plots, we use tidyverse, gridExtra grid and gtable.

```
install.packages(c("mice", "pROC", "metamisc"))
install.packages(c("tidyverse", "gridExtra", "grid", "gtable"))
library(mice)
library(pROC)
library(metamisc)
library(tidyverse)
library(gridExtra)
library(grid)
library(gtable)
```

Please make sure you have mice version 3.12.0:

```
packageVersion("mice")
```

```
## [1] '3.12.0'
```

Data

Load the data:

```
load(file.choose())
```

In this template file, we use a fictional data set. If you'd like to experiment with the functions in this file, you can use this mock data set. For the remainder of this file, we assume that:

- You have thoroughly checked the validity of your data,
- categorical variables are stored as factors ("yes" vs "no) for present and absent respectively,
- continuous variables are stored as numeric.
- Within-hospital-mortality or 30-day-mortality is coded as "dead" and recovered or censored is recorded as "alive". Alternatively, supply the appropriate value for case_label to the performance measures given in the last section.

```r
# First we create some predictor variables
n <- 1000
sex <- !!rbinom(n, 1, .60)
age <- runif(n, 40, 90)
ldh <- rnorm(n, age + 85, 25)
lymphocyte_count <- rnorm(n, 2500, 1000)
c_reactive_protein <- rexp(n, 1/(age/35 + 83))
respiratory_rate <- rnorm(n, 14, 2)
temperature <- rnorm(n, 38, 1)
fever <- temperature > 38

diabetes <- !!rbinom(n, 1, .26)
chronic_cardiac_disease <- !!rbinom(n, 1, .30)
heart_rate <- rnorm(n, diabetes * 10 + chronic_cardiac_disease * 10, 15)

n_comobidities <- diabetes + chronic_cardiac_disease + rbinom(n, 5, .1)

glasgow_coma_scale <- 15 - rbinom(n, 12, .02)
diastolic <- diabetes * 10 + chronic_cardiac_disease * 10 + rnorm(n, 50, 25)
systolic <- diastolic + rnorm(n, 55, 20)

haemoglobin <- rnorm(n, 129, 30 * 3/4)
platelet_count <- rnorm(n, 216, 120 * 3/4)
oxygen_saturation <- runif(n, .88, 1)
urea <- rnorm(n, 7, 4)
ddimer <- rexp(n, .3)

# Then we predict the outcome
p <- inv_logit(-8 +
                 3 * (age/mean(age))^2 +
                 sex +
                 respiratory_rate/mean(respiratory_rate) +
                 2 * (oxygen_saturation < .92) +
                 c_reactive_protein/mean(c_reactive_protein))

# Convert categorical variables to factors
sex[sex] <- "male"
sex[sex == "FALSE"] <- "female"
sex <- factor(sex)

fever[fever] <- "yes"
fever[fever == "FALSE"] <- "no"
fever <- factor(fever)

diabetes[diabetes] <- "yes"
diabetes[diabetes == "FALSE"] <- "no"
diabetes <- factor(diabetes)

chronic_cardiac_disease[chronic_cardiac_disease] <- "yes"
chronic_cardiac_disease[chronic_cardiac_disease == "FALSE"] <- "no"
chronic_cardiac_disease <- factor(chronic_cardiac_disease)

# Draw the outcome, convert to factor
```

```
mortality <- !!rbinom(n, 1, p)
mortality[mortality] <- "dead"
mortality[mortality == FALSE] <- "alive"
mortality <- factor(mortality)

your_data_object <- data.frame(sex,
                age,
                ldh,
                lymphocyte_count,
                c_reactive_protein,
                respiratory_rate,
                temperature,
                fever = fever,
                diabetes = diabetes,
                chronic_cardiac_disease,
                heart_rate,
                n_comobidities,
                glasgow_coma_scale,
                diastolic,
                systolic,
                haemoglobin,
                platelet_count,
                oxygen_saturation,
                urea,
                ddimer,
                mortality = mortality)

# And finally, we can remove some random values to add missingness
for (i in seq_len(ncol(your_data_object)))
  your_data_object[as.logical(rbinom(n, 1, 0.2)), i] <- NA
```

```
data_miss <- your_data_object
```

Multiple imputation

First, have a look at the distribution of missingness across the variables in your data:

```
round(sort(apply(data_miss, 2, function(x) mean(is.na(x))), decreasing = T), 3)
```

```
##             temperature               mortality        lymphocyte_count
##                   0.236                   0.224                   0.222
##               diastolic                     sex                     age
##                   0.212                   0.211                   0.210
##                     ldh      glasgow_coma_scale                   fever
##                   0.210                   0.209                   0.207
##                systolic        respiratory_rate              heart_rate
##                   0.203                   0.202                   0.200
##      c_reactive_protein                diabetes       oxygen_saturation
##                   0.197                   0.196                   0.196
##                  ddimer             haemoglobin chronic_cardiac_disease
##                   0.196                   0.195                   0.194
##          platelet_count                   urea          n_comobidities
##                   0.190                  0.188                   0.186
```

Initialize mice

Before running mice for real, we check whether the data is compatible with mice. We use maxit=0, as we don't need any imputations just yet.

```
setup_imp <- mice(data_miss, maxit=0)
```

If you get the following warning message:

then inspect the logged events:

```
setup_imp$loggedEvents
```

```
## NULL
```

In this case there are no logged events. What may happen, for instance, is that the data set contains a collinear variable, which mice then automatically removes. In that case, go a step back, inspect the issue, remove the variable you think needs to be removed and then run the mice initialization again.

Now, inspect the methods that mice has chosen for your data. Do the methods make sense? Note that if a binary variable is coded as 1 vs 0, mice will chose "pmm" instead of "logreg". If coded as factor or Boolean, it should choose "logreg".

```
setup_imp$method
```

```
##                     sex                    age                    ldh
##                "logreg"                  "pmm"                  "pmm"
##        lymphocyte_count     c_reactive_protein       respiratory_rate
##                   "pmm"                  "pmm"                  "pmm"
##             temperature                  fever               diabetes
##                   "pmm"               "logreg"               "logreg"
## chronic_cardiac_disease             heart_rate         n_comobidities
##                "logreg"                  "pmm"                  "pmm"
##      glasgow_coma_scale              diastolic               systolic
##                   "pmm"                  "pmm"                  "pmm"
##             haemoglobin         platelet_count      oxygen_saturation
##                   "pmm"                  "pmm"                  "pmm"
##                    urea                 ddimer              mortality
##                   "pmm"                  "pmm"               "logreg"
```

If not, then it can be altered as follows:

```
setup_imp$method["sex"] <- "logreg"
```

Run mice

In order for the imputation to be reproducible, we set a seed for the imputation algorithm:

```
set.seed(2020)
```

Now we can perform the imputation. We set the number of imputed data sets to 50 (m = 50).

```
data_imp <- mice(data_miss, method = setup_imp$method, m = 50, maxit = 25,
                 printFlag = F)
```

Here you may get warnings that in an iteration one of the conditional imputation models did not converge. If this occurs in your data, especially if it occurs multiple times, consider the following steps:

- Check that the data do not contain collinear variables
- Try a different imputation method. E.g. pmm instead of logreg for binary variables or norm instead of pmm for binary variables. See Details for the mice function in the mice package for options.
- Increase maxit
- Alter the visitSequence
- Change the seed for the imputation, either through set.seed() or the seed argument in mice().

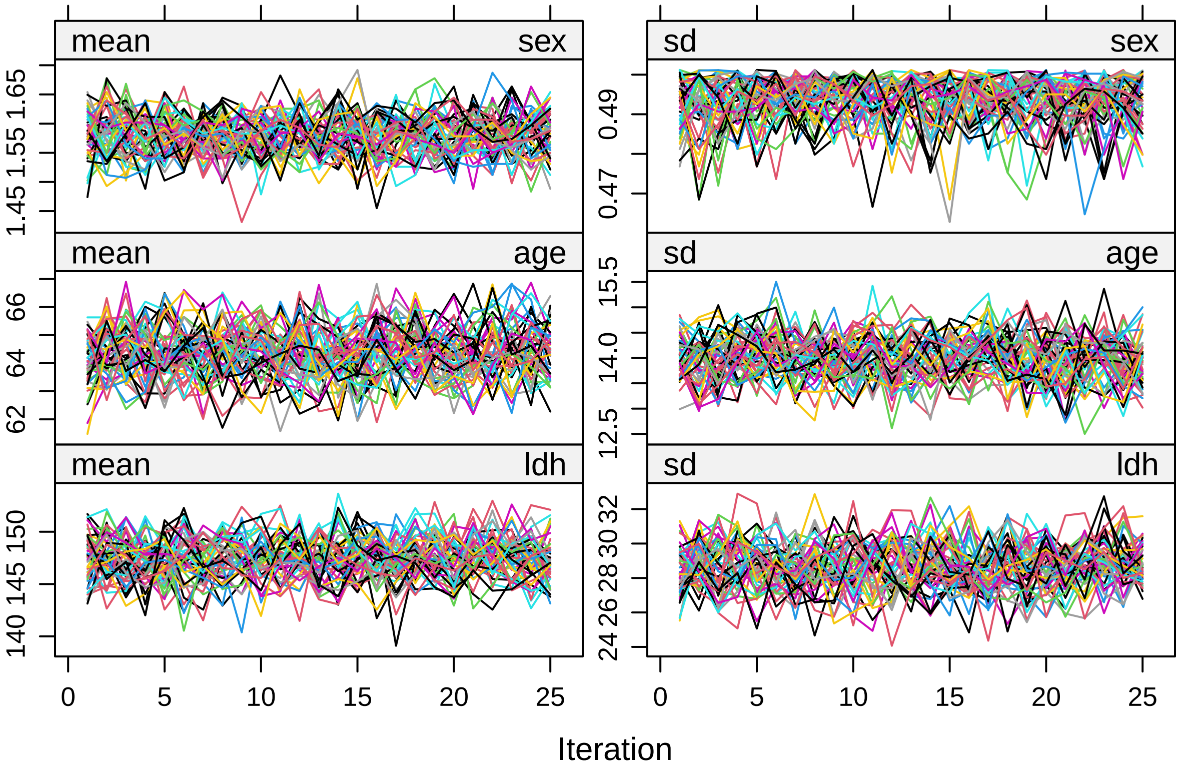
Inspect the validity of the imputation

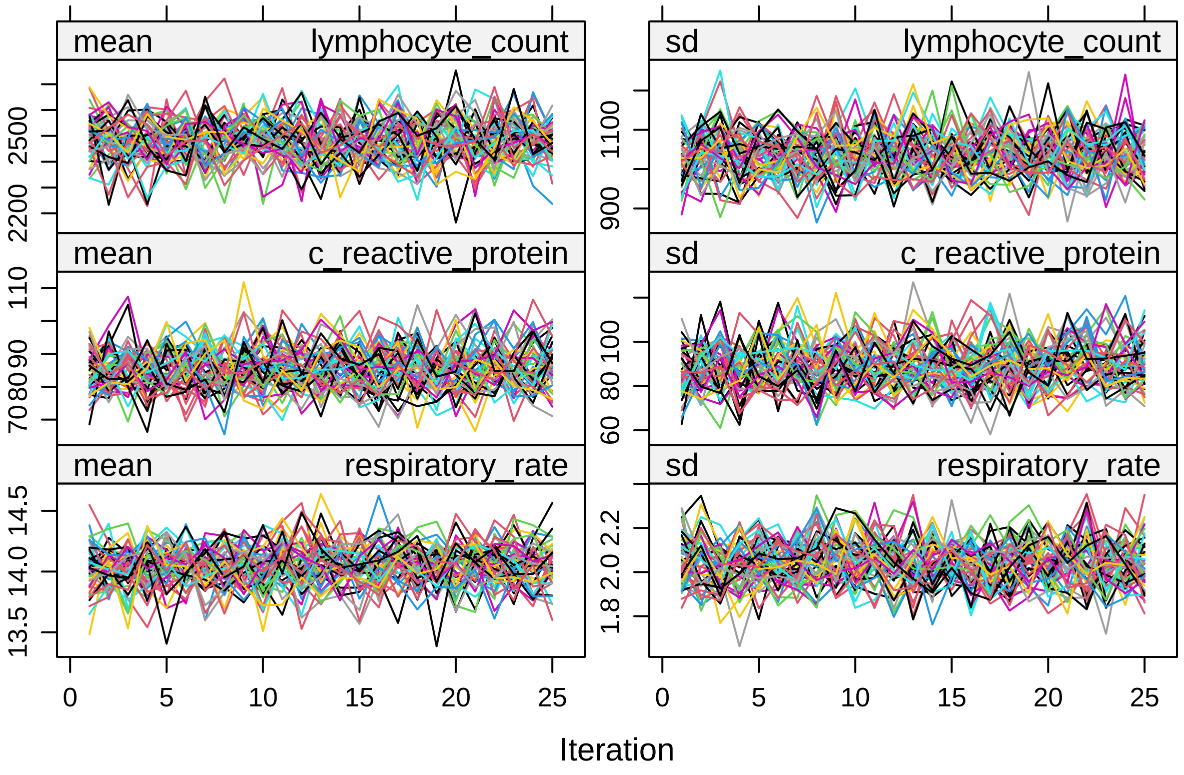
A simple plot can be generated as follows. It shows the means and standard deviations for the variables in the data set. The lines in each figure should appear as a random pattern. Specifically:

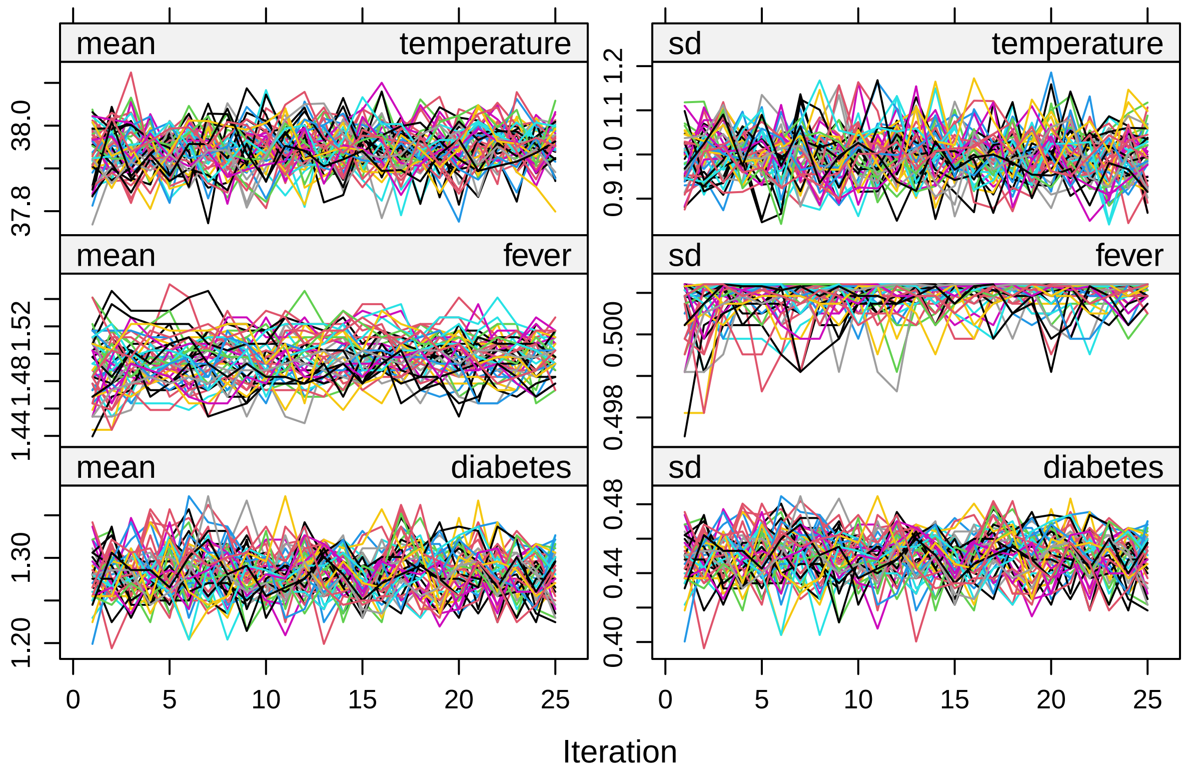
- The lines should cross multiple times.
- None of the lines should appear to be stuck at a specific value, i.e. it should not remain flat.
- In general, no pattern should be present (i.e. not increasing or decreasing)

The following plots all look ok:

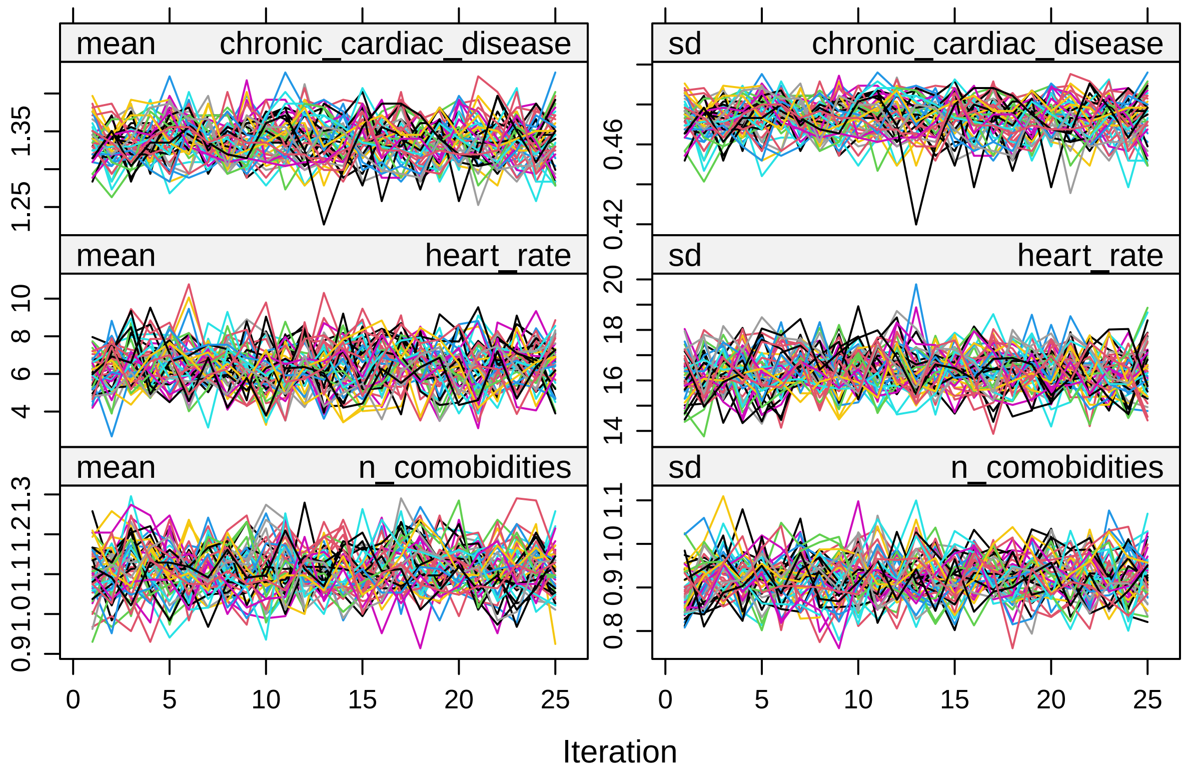
```
plot(data_imp)
```

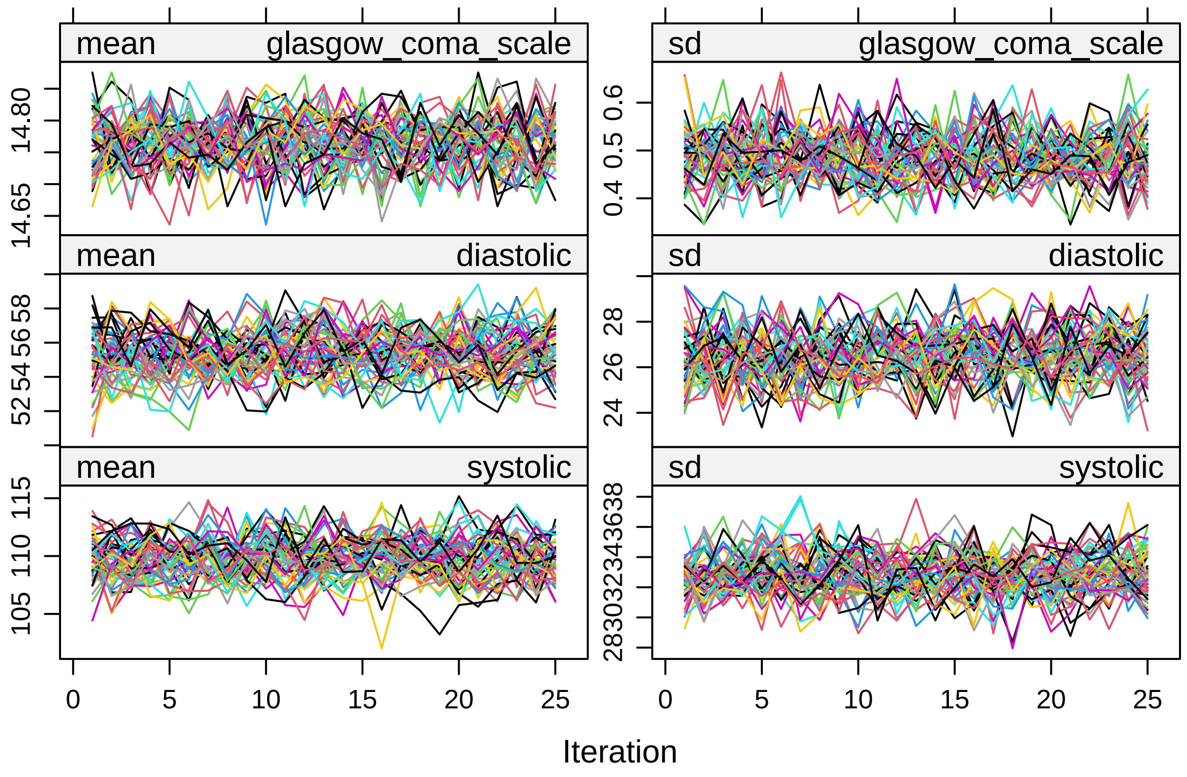

mean
sex
sd
sex
mean
age
sd
age
mean
ldh
sd
ldh
1.45 1.55 1.65
0.47 0.49
62 64 66
12.5 14.0 15.5
140 145 150
24 26 28 30 32
0 5 10 15 20 25
0 5 10 15 20 25
Iteration

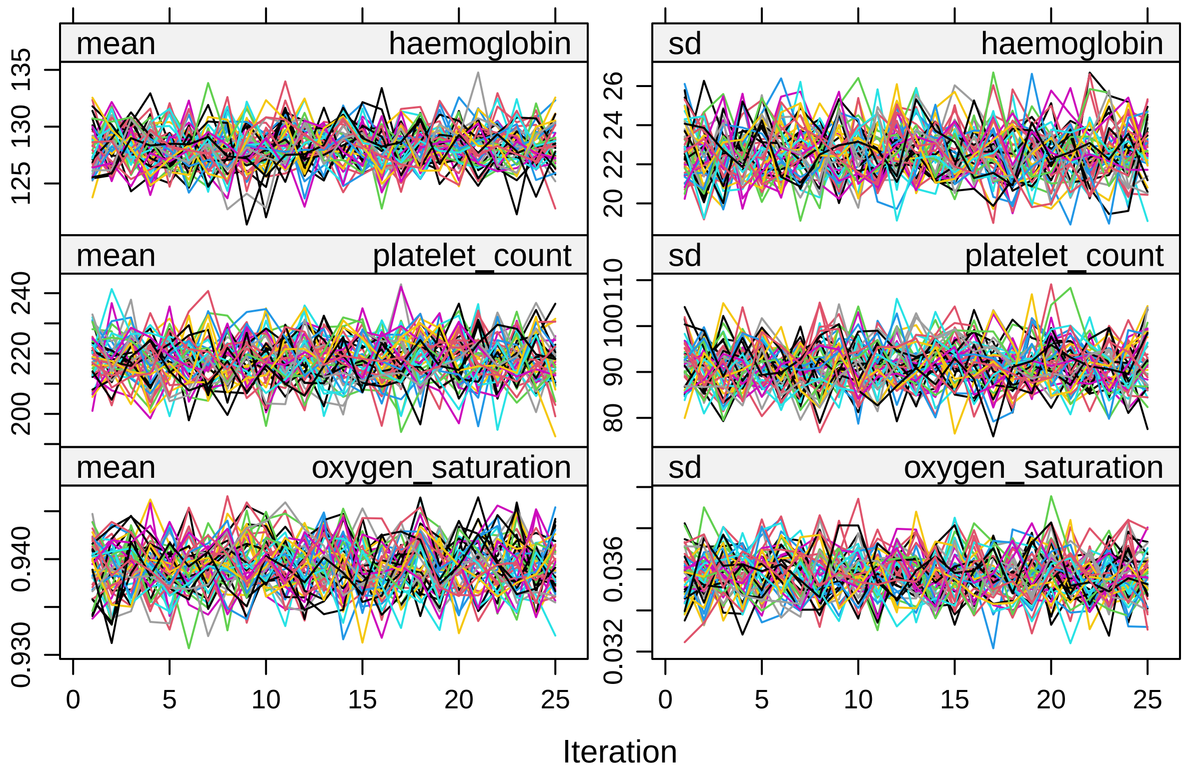

mean
lymphocyte_count
sd
lymphocyte_count
mean
c_reactive_protein
sd
c_reactive_protein
mean
respiratory_rate
sd
respiratory_rate
2200
2500
900
1100
70
80
90
110
60
80
100
13.5
14.0
14.5
1.8
2.0
2.2
0
5
10
15
20
25
Iteration

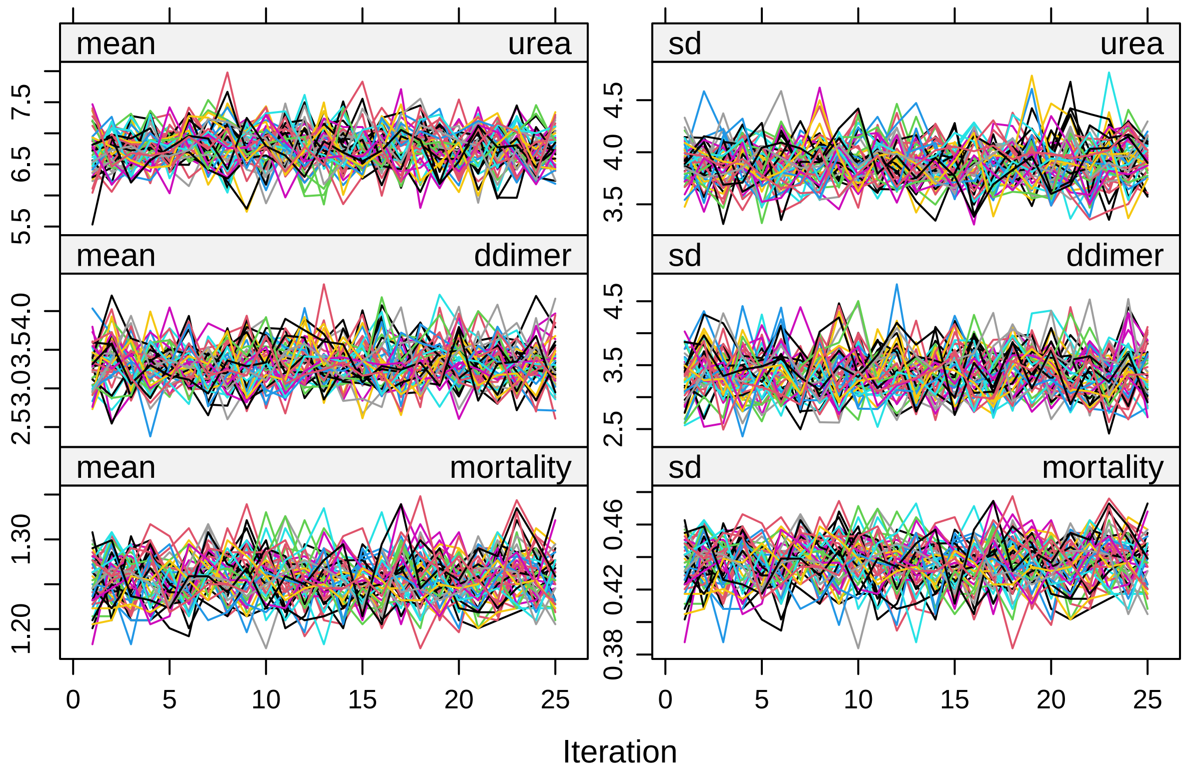


mean
temperature
sd
temperature
mean
fever
sd
fever
mean
diabetes
sd
diabetes
37.8
38.0
0.9
1.0
1.1
1.2
1.44
1.48
1.52
0.498
0.500
1.20
1.30
0.40
0.44
0.48
0
5
10
15
20
25
Iteration


mean
chronic_cardiac_disease
sd
chronic_cardiac_disease
mean
heart_rate
sd
heart_rate
mean
n_comobidities
sd
n_comobidities
1.25
1.35
0.42
0.46
4
6
8
10
14
16
18
20
0.9
1.0
1.1
1.2
1.3
0.8
0.9
1.0
1.1
0
5
10
15
20
25
Iteration


mean
glasgow_coma_scale
sd
glasgow_coma_scale
mean
diastolic
sd
diastolic
mean
systolic
sd
systolic
14.65
14.80
0.4 0.5 0.6
52 54 56 58
24 26 28
105 110 115
28303234363638
0 5 10 15 20 25
Iteration


mean
haemoglobin
sd
haemoglobin
mean
platelet_count
sd
platelet_count
mean
oxygen_saturation
sd
oxygen_saturation
Iteration



Model validation

Define prediction models

Here we define the 4C Mortality Score by Knight et al. It's easiest to define each prediction model with only a 'data' argument, so that they are compatible with the functions below.

Note that the 4C mortality model does not say what to do with:

- Ages between 59 and 60, etc. We assume these are all integers or rounded downwards.
- respiratory rate between 29 and 30, etc. We assume these are rounded to the nearest integer.
- Same for c_reactive_protein.

```
hu_lp <- function(data) {
    -4.211 + 0.013 * data$c_reactive_protein + 0.059 * data$age +
    0.112 * data$ddimer - 1.984 * data$lymphocyte_count
}

humodel <- function(data)
  inv_logit(hu_lp(data))

knight4cmodel <- function(data) {
  age                  <- data$age
  sex                  <- data$sex
  n_comobidities     <- data$n_comobidities
  respiratory_rate   <- data$respiratory_rate
  oxygen_saturation  <- data$oxygen_saturation
  glasgow_coma_scale <- data$glasgow_coma_scale
  urea               <- data$urea
  c_reactive_protein <- data$c_reactive_protein

  age <- floor(age)
  respiratory_rate   <- round(respiratory_rate)
  oxygen_saturation  <- round(oxygen_saturation)
  urea               <- round(urea)
  c_reactive_protein <- round(c_reactive_protein)

  lp <- (
    -4.203 +

      0.687 * (age >= 50 & age <= 60) +
      1.337 * (age >= 60 & age <= 69) +
      1.842 * (age >= 70 & age <= 79) +
      2.252 * (age >= 80) +

      0.172 * (sex == "male") +

      0.300 * (n_comobidities == 1) +
      0.532 * (n_comobidities >= 2) +

      0.232 * (respiratory_rate >= 20 & respiratory_rate <= 29) + # (breaths/minute)
      0.649 * (respiratory_rate >= 30) +                           # (breaths/minute)

      0.577 * (oxygen_saturation < 92) +  # on room air (%)

      0.558 * (glasgow_coma_scale < 15) +

      0.439 * (urea >= 7 & urea <= 14) +  # (mmol/L)
      1.011 * (urea > 14) +               # (mmol/L)

      0.363 * (c_reactive_protein >= 50 & c_reactive_protein <= 99) + # (mg/L)
      0.740 * (c_reactive_protein >= 100)                              # (mg/L)
  )

  return(inv_logit(lp))
}

knight4cscore <- function(data) {
  age                <- data$age
  sex                <- data$sex
  n_comobidities     <- data$n_comobidities
  respiratory_rate   <- data$respiratory_rate
  oxygen_saturation  <- data$oxygen_saturation
  glasgow_coma_scale <- data$glasgow_coma_scale
  urea               <- data$urea
  c_reactive_protein <- data$c_reactive_protein
```

```r
age <- floor(age)
respiratory_rate   <- round(respiratory_rate)
oxygen_saturation  <- round(oxygen_saturation)
urea               <- round(urea)
c_reactive_protein <- round(c_reactive_protein)

score <- (
    2 * (age >= 50 & age <= 60) +
    4 * (age >= 60 & age <= 69) +
    6 * (age >= 70 & age <= 79) +
    7 * (age >= 80) +

    1 * (sex == "male") +

    1 * (n_comobidities == 1) +
    2 * (n_comobidities >= 2) +

    1 * (respiratory_rate >= 20 & respiratory_rate <= 29) + # (breaths/minute)
    2 * (respiratory_rate >= 30) +                          # (breaths/minute)

    2 * (oxygen_saturation < 92) +  # on room air (%)

    2 * (glasgow_coma_scale < 15) +

    1 * (urea >= 7 & urea <= 14) +  # (mmol/L)
    3 * (urea > 14) +               # (mmol/L)

    1 * (c_reactive_protein >= 50 & c_reactive_protein <= 99) + # (mg/L)
    2 * (c_reactive_protein >= 100)                             # (mg/L)
)

index <- score + 1 # The method below does not actually use the values
# within the " ". It uses the supplied value as an index instead. As zero is
# the first value in the list, its index equals score + 1.

# The 4C Score does not state what to do with values of 0. In the graphs of
# the manuscript the mortality rate is zero, but this would lead to fitting
# issues with the calibration slope and calibration in the large, so we use a
# value that is practically zero but not exactly.

c("0" = 0.00001,
  "1" = 0.003,
  "2" = 0.008,
  "3" = 0.023,
  "4" = 0.048,
  "5" = 0.075,
  "6" = 0.078,
  "7" = 0.117,
  "8" = 0.144,
  "9" = 0.192,
  "10" = 0.229,
  "11" = 0.269,
  "12" = 0.329,
    "13" = 0.401,
    "14" = 0.446,
    "15" = 0.516,
    "16" = 0.591,
    "17" = 0.661,
    "18" = 0.758,
    "19" = 0.774,
    "20" = 0.829,
    "21" = 0.875
  )[index]
}
```

Performance measures

Here we define the performance measures. All take the arguments p or lp (predicted risk of short term mortality, or the linear predictor, only one of the two is necessary), and y (observed short term mortality), so that we can use them in the functions below.

Note that we set the direction for auc to be "<", meaning that the the predictor values for the surviving patients are lower or equal than the values of the deceased patients. If we do not set this, a model that has an auc lower than 0.50 will receive a value greater than 0.50 (i.e. 0.40 becomes 1 - 0.40 = 0.60).

For net benefit, see Vickers AJ, Elkin EB. Decision curve analysis: a novel method for evaluating prediction models. Medical Decision Making. 2006 Nov;26(6):565-74.

```
calibration_in_the_large <- function(p, y, lp = logit(p), family = binomial, ...) {
  fit <- glm(y ~ 1, family = family, offset = lp, ...)
  variance <- diag(vcov(fit))
  c(estimate = coef(fit), se = sqrt(variance), var = variance)
}

calibration_slope <- function(p, y, lp = logit(p), family = binomial, ...) {
  fit <- glm(y ~ lp + 1, family = family, ...)
  variance <- diag(vcov(fit))[2]
  c(estimate = coef(fit)[2], se = sqrt(variance), var = variance)
}

auc <- function(p, y, lp = logit(p), method = "delong", direction = "<", ...) {
  if (missing(p))
    p <- inv_logit(lp)
  fit <- pROC::auc(response = y,
                   predictor = p,
                   direction = direction,
                   method = method, ...)
  variance <- var(fit)
  ci <- confint(fit)
  c(estimate = fit[[1]][[1]],
    se = sqrt(variance),
    var = variance,
    ci.lb = ci$ci.lb,
    ci.ub = ci$ci.ub)
}

# threshold is the decision threshold at which predictions are dichotimized
```

```r
# into positive or negative.
# equipoise is the value at which FP and FN are considered to be equally bad.
# i.e. equipoise = .20 means that four FP diagnoses are as bad as one FN.
# These two are the same when validating a prediction model. Only when
# considering the 'treat all' strategy they are not equal: threshold is then 0,
# and equipoise is the clinical equipoise value, (or all values from 0 to 1 in
# a decision curve analysis).

net_benefit <- function(p, y, lp = logit(p), threshold = .50, equipoise = threshold,
                        test_harm = 0, case_label = "dead", ...) {
  if (missing(p))
    p <- inv_logit(lp)
  tp <- sum((p >= threshold) & (y == case_label))
  fp <- sum((p >= threshold) & (y != case_label))
  n <- length(p)

  if (threshold == 1) { # by definition
    return(0)
  } else{return(tp/n - fp/n * equipoise / (1-equipoise) - test_harm)}
}

net_benefit_bootstrap <- function(p, y, lp = logit(p), threshold = .50, equipoise = threshold,
                                  test_harm = 0, case_label = "dead",
                                  I = 1000, ...) {
  if (missing(p))
    p <- inv_logit(lp)
  nb <- rep(NA, I)
  for (i in seq_len(I)) {
    s <- sample.int(length(p), length(p), replace = TRUE)
    nb[i] <- net_benefit(p[s], y[s], lp, threshold, equipoise, test_harm, case_label, ...)
  }
  se <- sd(nb)
  ci <- quantile(nb, probs = c(.025, .975))
  est <- net_benefit(p, y, lp, threshold, equipoise, test_harm, case_label, ...)

  c(estimate = est, se = se, var = se^2, ci.lb = ci[1], ci.ub = ci[2])
}

oe_ratio <- function(p, y, lp = logit(p), case_label = "dead", log = T, ...) {
  if (missing(p))
    p <- inv_logit(lp)
  O <- sum(y == case_label)
  E <- sum(p)
  N <- length(p)
  g <- if(log) "log(OE)" else NULL

  fit <- metamisc::oecalc(O = O, E = E, N = N, g = g, ...)

  c(estimate = fit$theta,
    se = fit$theta.se,
    var = fit$theta.se^2,
    ci.lb = fit$theta.cilb,
    ci.ub = fit$theta.ciub,
    O = O,
    E = E,
    N = N,
    log = log)
}
```

Validation for MI

Validating a model is a two step procedure:

1. Estimate the probability of mortality for each patient
2. Estimate the performance measure.

However, the functions in the mice package expect that a single function is applied on the imputed data sets. Further, the with() function does not work with the above function. So we make our own function that can do both tasks (note that some of it is copied from the mice package, specifically mice:::glm.mids). We make it generic, so that we can reuse it. The arguments are:

- *data* : Multiple imputed data set of class mids
- *model* : Prediction model (as function or character), is assumed to produce a vector of predicted values
- *measure* : Performance measure (as function or character)
- *alpha* : Error level for confidence intervals.
- ... : Arguments passed to measure.

```
validate.mids <- function(data, model, measure, alpha = .05, ...) {
  # From mice
  call <- match.call()
  if (!is.mids(data)) {
    stop("The data must have class mids")
  }

  # Make sure that these two are functions:
  model   <- match.fun(model)
  measure <- match.fun(measure)

  # Here we have two steps instead of one.
  # 1. Predict the outcome
  predicted_values <- lapply(seq_len(data$m),
                             function(i) model(data = complete(data, i)))

  # 2. Estimate performance
  analyses <- sapply(seq_len(data$m),
                     function(i) measure(
                       p = predicted_values[[i]],
                       y = complete(data, i)$mortality, ...))
  analyses <- as.data.frame(t(analyses))
  analyses$ci.lb <- analyses$estimate + qnorm(alpha/2)   * analyses$se
  analyses$ci.ub <- analyses$estimate + qnorm(1-alpha/2) * analyses$se

  # From mice
  object <- list(call = call,
                 call1 = data$call,
```

```r
                 nmis = data$nmis,
                 analyses = analyses)
  oldClass(object) <- c("mira")
  object
}

validate.lp.mids <- function(data, model, measure, alpha = .05, ...) {
  # From mice
  call <- match.call()
  if (!is.mids(data)) {
    stop("The data must have class mids")
  }

  # Make sure that these two are functions:
  model   <- match.fun(model)
  measure <- match.fun(measure)

  # Here we have two steps instead of one.
  # 1. Predict the outcome
  linear_predictor <- lapply(seq_len(data$m),
                             function(i) model(data = complete(data, i)))

  # 2. Estimate performance
  analyses <- sapply(seq_len(data$m),
                     function(i) measure(
                       lp = linear_predictor[[i]],
                       y = complete(data, i)$mortality, ...))
  analyses <- as.data.frame(t(analyses))
  analyses$ci.lb <- analyses$estimate + qnorm(alpha/2)   * analyses$se
  analyses$ci.ub <- analyses$estimate + qnorm(1-alpha/2) * analyses$se

  # From mice
  object <- list(call = call,
                 call1 = data$call,
                 nmis = data$nmis,
                 analyses = analyses)
  oldClass(object) <- c("mira")
  object
}

# For data sets for which we only have the outcomes and predicted values
# after MI has been performed, we can use this function which skips the
# first steps.
validate.mids.predicted <- function(data, p_name, y_name, measure, alpha = .05, ...) {
  measure <- match.fun(measure)
  call <- match.call()

  #  Estimate performance
  analyses <- sapply(seq_len(length(data)),
                     function(i) measure(
                       p = data[[i]][ , p_name],
                       y = data[[i]][ , y_name], ...))
  analyses <- as.data.frame(t(analyses))
  analyses$ci.lb <- analyses$estimate + qnorm(alpha/2)   * analyses$se
  analyses$ci.ub <- analyses$estimate + qnorm(1-alpha/2) * analyses$se

  # From mice
  object <- list(call = call,
                 call1 = data$call,
                 nmis = data$nmis,
                 analyses = analyses)
  oldClass(object) <- c("mira")
  object
}
```

Next, we pool the performance estimates across important data sets. As the pool function from the mice package does not work with some performance measures, we use the older function pool.scalar instead. As the auc is not normally distributed (as it is bounded from 0 to 1), we cannot use a method that assumes normality, like pool.scalar. We use some modified code from the psfmi package instead.

```
pool.validate.mids <- function(object, alpha = .05, ...) {
  fit <- pool.scalar(Q = object$analyses$estimate,
                     U = object$analyses$var,
                     n = nrow(object$analyses))

  list("invididual" = data.frame(estimate = fit$qhat,
                                 var = fit$u,
                                 se = sqrt(fit$u),
                                 ci.lb = object$analyses$ci.lb,
                                 ci.ub = object$analyses$ci.ub),
       "pooled" = data.frame(estimate = fit$qbar,
                             variance = fit$t,
                             within = fit$ubar,
                             between = fit$b,
                             se = sqrt(fit$t),
                             ci.lb = fit$qbar + qnorm(alpha/2) * sqrt(fit$t),
                             ci.ub = fit$qbar + qnorm(1 - alpha/2) * sqrt(fit$t),
                             df = fit$df,
                             r = fit$r,
                             fmi = fit$fmi,
                             m = fit$m))
}

pool.auc.mids <- function(object, alpha = .05, ...) {
  m <- length(object$analyses$se)

  logit <- function(x) log(x / (1-x))
  inv_logit <- function(x) {1/(1+exp(-x))}

  # Estimates from individual studies
  ind <- list(auc = data.frame(est = object$analyses$estimate,
                               se = object$analyses$se,
                               ci.lb = object$analyses$ci.lb,
                               ci.ub = object$analyses$ci.ub))
  ind$logit.auc = data.frame(est = logit(ind$auc$est),
                             se = ind$auc$se / (ind$auc$est * (1 - ind$auc$est)),
                              ci.lb = logit(ind$auc$ci.lb),
                              ci.ub = logit(ind$auc$ci.ub))

  # Pooled across imputed data sets
  logit.auc <- list()
  logit.auc$estimate <- mean(ind$logit.auc$est)
  logit.auc$within <- mean(ind$logit.auc$se^2)
  logit.auc$between <- (1 + (1/m)) * var(ind$logit.auc$est)
  logit.auc$var <- logit.auc$within + logit.auc$between
  logit.auc$se <- sqrt(logit.auc$var)
  logit.auc$ci.lb <- logit.auc$est + qnorm(alpha/2)     * logit.auc$se
  logit.auc$ci.ub <- logit.auc$est + qnorm(1 - alpha/2) * logit.auc$se
  logit.auc$m <- m

  auc <- list()
  auc$estimate <- inv_logit(logit.auc$est)
  auc$ci.lb <- inv_logit(logit.auc$ci.lb)
  auc$ci.ub <- inv_logit(logit.auc$ci.ub)
  auc$m <- m

  return(list(individual = ind,
              pooled = list(logit.auc = as.data.frame(logit.auc),
                            auc = as.data.frame(auc))))
}

pool.oe.mids <- function(object, alpha = .05, ...) {
  fit <- pool.validate.mids(object, alpha = alpha, ...)

  if (!object$analyses$log[[1]]) return(list(oe = fit))

  oe <- list(individual = exp(fit$invididual))
  oe$individual$var <- NULL
  oe$individual$se <- NULL

  oe$pooled <- data.frame(estimate = exp(fit$pooled$estimate),
                          ci.lb = exp(fit$pooled$ci.lb),
                          ci.ub = exp(fit$pooled$ci.ub))

  list(log_oe = fit,
       oe = oe)
}
```

For calibration plots, we use cal_plot to make a plot and model_cal_plot to apply it to our data and prediction models, and align_plots to align the scales of the plots. The arguments are as follows:

- *data* : A data.frame that contains all the predictors for the model to be validated
- *model_name* : character name of the model to be validated. Will be added to the title of the plot.
- *model* : The function that produces the predicted outcomes for this prediction model. This argument can be left out if model_name is also the name of the function (as in this example).
- *data_name* : character name of your data. Will be added to the title of the plot.
- *imp* : integers that describe which imputed data sets are to be imputed, if a mids object is supplied. Defaults to all data sets.
- *case_label* : character name of the factor level that describes a case of mortality.

```r
model_cal_plot <- function(data,
                           model_name,
                           model = match.fun(model_name),
                           data_name = "fictional data",
                           imp = seq_len(data$m),
                           case_label = "dead") {
  # If data is a multiple imputed object, we extract all imputed data sets and
  # stack them.
  if (is.mids(data)) data <- complete(data, imp)

  # Predict outcomes
  model <- match.fun(model)
  p <- model(data)
  pp <<- p

  # Convert outcome to binary
  y <- data$mortality
  y_binary <- rep(0, length(y))
  y_binary[y == case_label] <- 1

  title <- paste0(model_name, " in ", data_name)

  cal_plot(data.frame(p = p, y = y_binary), title)
}

#https://stackoverflow.com/questions/26159495/align-multiple-ggplot-graphs-with-and-without-legends
align_plots <- function(...) {
  plots.grobs <- lapply(list(...), ggplotGrob)
  max.widths <- do.call(unit.pmax, lapply(plots.grobs, "[[", "widths"))
  plots.grobs.eq.widths <- lapply(plots.grobs, function(x) {
    x$widths <- max.widths
    x
  })

  LegendWidth <- function(x) x$grobs[[8]]$grobs[[1]]$widths[[4]]
  legends.widths <- lapply(plots.grobs, LegendWidth)

  max.legends.width <- if(!is.null(unlist(legends.widths)))
    do.call(max, legends.widths) else 0

  out <- lapply(plots.grobs.eq.widths, function(x) {
    if (is.gtable(x$grobs[[8]])) {
      x$grobs[[8]] <- gtable_add_cols(x$grobs[[8]],
                                      unit(abs(diff(c(LegendWidth(x),
                                                      max.legends.width))),
                                           "mm"))
    }
    x
  })

  out
}
```

```r
cal_plot <- function(data, title){

  library(tidyverse)
  library(gridExtra)
  library(grid)
  library(gtable)

  # we reduce the expand param, as the plot should not exceed c(0,1) by too much
  e <- 0.01

  # The calibration plot
    # Bin prediction into 10ths
  g1 <- mutate(data, bin = ntile(p, 10)) %>%
    group_by(bin) %>%

    # Estimate Observed and expected per bin
    mutate(n = n(),
           bin_pred = mean(p),
           bin_prob = mean(y)) %>%
    ungroup() %>%

    # Plot figure and bins
    ggplot(aes(x = bin_pred, y = bin_prob, ymin = bin_prob, ymax = bin_prob)) +
    geom_pointrange(size = 0.5, color = "black") +

    # plot axes
    scale_y_continuous(expand = c(e, e), breaks = seq(0, 1, by = 0.1)) +
    scale_x_continuous(expand = c(e, e), breaks = seq(0, 1, by = 0.1)) +
    coord_cartesian(xlim = c(0, 1), ylim = c(0, 1)) +

    # 45 degree line indicating perfect calibration
    geom_abline()  +

    # straight line fit through estimates
    geom_smooth(method = "lm", se = FALSE, linetype = "dashed",
                color = "black", formula = y ~ x) +

    # loess fit through estimates
    geom_smooth(method = "loess", se = FALSE, formula = y ~ x, span = 1) +

    xlab("") +
    ylab("Observed Probability") +
    theme_minimal()+
    ggtitle(title)

  # The distribution plot
  g2 <- ggplot(data, aes(x = p)) +
    geom_histogram(fill = "black", bins = 100) +
    scale_x_continuous(expand = c(e, e),
                       breaks = seq(0, 1, by = 0.1)) +
    coord_cartesian(xlim = c(0, 1)) +
    xlab("Predicted Probability") +
    ylab("Count") +
    theme_minimal() +
    scale_y_continuous() +
    theme(panel.grid.minor = element_blank())

  # Combine them
  g12 <- align_plots(g1, g2)
  do.call(function(...) grid.arrange(..., heights = c(1, .35)), g12)
}
```

Estimate performance

We apply the functions we defined above to validate the model in the imputed data sets.

```
est_auc <- validate.mids(data = data_imp,
                         model = "knight4cscore",
                         measure = auc,
                         quiet = F)
```

```
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
## Setting levels: control = alive, case = dead
```

```
est_slope <- validate.mids(data = data_imp,
                     model = "knight4cscore",
                     measure = calibration_slope)

est_int <- validate.mids(data = data_imp,
                     model = "knight4cscore",
                     measure = calibration_in_the_large)


est_int <- validate.lp.mids(data = data_imp,
                     model = "hu_lp",
                     measure = calibration_in_the_large)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
est_log_oe <- validate.mids(data = data_imp,
                            model = "knight4cscore",
                            measure = oe_ratio,
                            log = TRUE)

est_log_oe <- validate.mids(data = data_imp,
                            model = "knight4cmodel",
                            measure = oe_ratio,
                            log = TRUE)
```

```
est_nb <- validate.mids(data = data_imp,
                        model = "knight4cscore",
                        measure = net_benefit_bootstrap)
```

And then we pool the results across the imputed data sets, according to Rubin's rules. The code we defined above produces a dataframe, where "estimate" is the pooled estimate (i.e. the one of interest), and "variance" is its variance. "within" is the mean of the estimated variance within each imputed data set, whereas "between" is the variance across imputed data sets. These two are pooled using Rubin's rules, to obtain "estimate", as described previously. fmi is the fraction of missing information due to nonresponse. For the other statistics, see the mice package, under pool.scalar

```
pool.auc.mids(est_auc)
```

```
## $individual
## $individual$auc
##          est         se     ci.lb     ci.ub
## 1  0.7601693 0.01661320 0.7276081 0.7927306
## 2  0.7641941 0.01631576 0.7322158 0.7961724
## 3  0.7700945 0.01668181 0.7373988 0.8027903
## 4  0.7784641 0.01608894 0.7469304 0.8099979
## 5  0.7658365 0.01679832 0.7329124 0.7987606
## 6  0.7502715 0.01684636 0.7172533 0.7832898
## 7  0.7753534 0.01631630 0.7433741 0.8073328
## 8  0.7908082 0.01596811 0.7595113 0.8221051
## 9  0.7539978 0.01692048 0.7208343 0.7871613
## 10 0.7635066 0.01671423 0.7307473 0.7962659
## 11 0.7629574 0.01680986 0.7300107 0.7959041
## 12 0.7747600 0.01603223 0.7433374 0.8061826
## 13 0.7793894 0.01639417 0.7472574 0.8115214
## 14 0.7712899 0.01629052 0.7393611 0.8032188
## 15 0.7863164 0.01613649 0.7546895 0.8179434
## 16 0.7649292 0.01691085 0.7317846 0.7980739
## 17 0.7751813 0.01598687 0.7438476 0.8065149
## 18 0.7797193 0.01605892 0.7482444 0.8111942
## 19 0.7637987 0.01698764 0.7305035 0.7970939
## 20 0.7866539 0.01564644 0.7559874 0.8173203
## 21 0.7647830 0.01616802 0.7330943 0.7964717
## 22 0.7895478 0.01571603 0.7587450 0.8203507
## 23 0.7752852 0.01640309 0.7431358 0.8074347
## 24 0.7603783 0.01676080 0.7275277 0.7932288
## 25 0.7751968 0.01612480 0.7435928 0.8068009
## 26 0.7799532 0.01625915 0.7480858 0.8118205
## 27 0.7707720 0.01625895 0.7389050 0.8026389
## 28 0.7722816 0.01642867 0.7400820 0.8044812
## 29 0.7808757 0.01663133 0.7482789 0.8134725
## 30 0.7756835 0.01618536 0.7439607 0.8074062
## 31 0.7622453 0.01691705 0.7290885 0.7954021
## 32 0.7600983 0.01657181 0.7276181 0.7925785
## 33 0.7609330 0.01680081 0.7280040 0.7938620
## 34 0.7919513 0.01554385 0.7614859 0.8224167
## 35 0.7832735 0.01616595 0.7515888 0.8149581
## 36 0.7649433 0.01660293 0.7324022 0.7974845
## 37 0.7877790 0.01545255 0.7574926 0.8180655
## 38 0.7540204 0.01710343 0.7204983 0.7875425
## 39 0.7901612 0.01568279 0.7594235 0.8208989
## 40 0.7521437 0.01680218 0.7192120 0.7850754
## 41 0.7645733 0.01688153 0.7314861 0.7976604
## 42 0.7650628 0.01673755 0.7322578 0.7978678
## 43 0.7805951 0.01592275 0.7493871 0.8118031
## 44 0.7662217 0.01692381 0.7330516 0.7993917
## 45 0.7674804 0.01667882 0.7347905 0.8001703
## 46 0.7880774 0.01578379 0.7571417 0.8190130
## 47 0.7725387 0.01691651 0.7393829 0.8056944
## 48 0.7594480 0.01721715 0.7257030 0.7931930
## 49 0.7788312 0.01620978 0.7470606 0.8106017
## 50 0.7737644 0.01634445 0.7417299 0.8057990
##
## $individual$logit.auc
##         est         se     ci.lb    ci.ub
## 1  1.153608 0.09112514 0.9825205 1.341464
## 2  1.175813 0.09054176 1.0058937 1.362541
## 3  1.208845 0.09422143 1.0324922 1.403826
## 4  1.256738 0.09329210 1.0823074 1.449996
## 5  1.184949 0.09367219 1.0094496 1.378566
## 6  1.100061 0.08991236 0.9308775 1.284942
## 7  1.238791 0.09367469 1.0635798 1.432772
## 8  1.329804 0.09652454 1.1500020 1.530676
## 9  1.120049 0.09122282 0.9486036 1.307899
## 10 1.172002 0.09256663 0.9984175 1.363117
## 11 1.168962 0.09294745 0.9946766 1.360889
## 12 1.235387 0.09187160 1.0633877 1.425394
## 13 1.262111 0.09534733 1.0840381 1.459926
## 14 1.215609 0.09234894 1.0426506 1.406535
## 15 1.302863 0.09603741 1.1237816 1.502475
## 16 1.179897 0.09404712 1.0036957 1.374299
## 17 1.237802 0.09173342 1.0660635 1.427522
## 18 1.264032 0.09349786 1.0892710 1.457789
## 19 1.173620 0.09416119 0.9971788 1.368229
## 20 1.304872 0.09322817 1.1308048 1.498297
## 21 1.179084 0.08987728 1.0103788 1.364387
## 22 1.322202 0.09458254 1.1458111 1.518725
## 23 1.238399 0.09415263 1.0623310 1.433427
## 24 1.154754 0.09198962 0.9821152 1.344499
## 25 1.237892 0.09252945 1.0647267 1.429355
## 26 1.265393 0.09473583 1.0884292 1.461883
## 27 1.212675 0.09202349 1.0402851 1.402870
## 28 1.221239 0.09341757 1.0463947 1.414541
## 29 1.270777 0.09719738 1.0894539 1.472733
## 30 1.240686 0.09302006 1.0666574 1.433243
## 31 1.165029 0.09334708 0.9900028 1.357801
## 32 1.153219 0.09087971 0.9825714 1.340538
## 33 1.157801 0.09235582 0.9845192 1.348364
## 34 1.336728 0.09433981 1.1608433 1.532807
## 35 1.284845 0.09523040 1.1071038 1.482555
## 36 1.179975 0.09233851 1.0068445 1.370646
## 37 1.311590 0.09242882 1.1389816 1.503296
## 38 1.120171 0.09221485 0.9469345 1.310175
## 39 1.325897 0.09458491 1.1495214 1.522449
## 40 1.110078 0.09012906 0.9405565 1.295492
## 41 1.177918 0.09378574 1.0021754 1.371736
## 42 1.180639 0.09312000 1.0061079 1.373021
## 43 1.269138 0.09297064 1.0953460 1.461769
## 44 1.187098 0.09447993 1.0101607 1.382497
## 45 1.194139 0.09346274 1.0190656 1.387359
## 46 1.313375 0.09450723 1.1370722 1.509675
## 47 1.222702 0.09626825 1.0427638 1.422273
## 48 1.149655 0.09424413 0.9729293 1.344280
## 49 1.258868 0.09410436 1.0829964 1.453925
## 50 1.229691 0.09336856 1.0549792 1.422940
##
##
## $pooled
## $pooled$logit.auc
##   estimate      within    between        var       se     ci.lb    ci.ub  m
## 1 1.219149 0.008702568 0.00395909 0.01266166 0.112524 0.9986064 1.439693 50
##
## $pooled$auc
##    estimate     ci.lb    ci.ub  m
## 1 0.7719138 0.7307845 0.808407 50
```

```
pool.validate.mids(est_slope)
```

```
## $invididual
##    estimate        var        se     ci.lb    ci.ub
## 1  1.296448 0.01297454 0.1139059 1.0731969 1.519700
## 2  1.321831 0.01389753 0.1178878 1.0907749 1.552887
## 3  1.387383 0.01492891 0.1221839 1.1479073 1.626859
## 4  1.456017 0.01529380 0.1236681 1.2136321 1.698402
## 5  1.290374 0.01299002 0.1139738 1.0669900 1.513759
## 6  1.206268 0.01245506 0.1116023 0.9875318 1.425005
## 7  1.408178 0.01432100 0.1196704 1.1736284 1.642728
## 8  1.518524 0.01572160 0.1253858 1.2727727 1.764276
## 9  1.245304 0.01259249 0.1122163 1.0253639 1.465244
## 10 1.325713 0.01310632 0.1144829 1.1013308 1.550095
## 11 1.321859 0.01363913 0.1167867 1.0929617 1.550757
## 12 1.431315 0.01498502 0.1224133 1.1913889 1.671240
## 13 1.420362 0.01473199 0.1213754 1.1824708 1.658254
## 14 1.387721 0.01410394 0.1187600 1.1549554 1.620486
## 15 1.458121 0.01494698 0.1222578 1.2184997 1.697742
## 16 1.326298 0.01390167 0.1179053 1.0952075 1.557388
## 17 1.425236 0.01434290 0.1197618 1.1905066 1.659964
## 18 1.460236 0.01489056 0.1220269 1.2210673 1.699404
## 19 1.307845 0.01377440 0.1173644 1.0778153 1.537875
## 20 1.491270 0.01546990 0.1243781 1.2474936 1.735047
## 21 1.345421 0.01350702 0.1162197 1.1176347 1.573207
## 22 1.519678 0.01577261 0.1255890 1.2735277 1.765828
## 23 1.445086 0.01531049 0.1237356 1.2025684 1.687603
## 24 1.307211 0.01364505 0.1168120 1.0782639 1.536159
## 25 1.430877 0.01465367 0.1210524 1.1936184 1.668135
## 26 1.458976 0.01515006 0.1230856 1.2177330 1.700220
## 27 1.369898 0.01413885 0.1189069 1.1368452 1.602952
## 28 1.378432 0.01379491 0.1174517 1.1482310 1.608633
## 29 1.421485 0.01453575 0.1205643 1.1851836 1.657787
## 30 1.432262 0.01505105 0.1226827 1.1918086 1.672716
## 31 1.303892 0.01355512 0.1164264 1.0757007 1.532084
## 32 1.287918 0.01284415 0.1133320 1.0657916 1.510045
## 33 1.303475 0.01330638 0.1153533 1.0773868 1.529563
## 34 1.517611 0.01542386 0.1241928 1.2741971 1.761024
## 35 1.449867 0.01496985 0.1223513 1.2100625 1.689671
## 36 1.293538 0.01335549 0.1155659 1.0670330 1.520043
## 37 1.511654 0.01564324 0.1250730 1.2665155 1.756792
## 38 1.246116 0.01334999 0.1155421 1.0196575 1.472574
## 39 1.517015 0.01551671 0.1245661 1.2728696 1.761160
## 40 1.276358 0.01319186 0.1148558 1.0512444 1.501471
## 41 1.337517 0.01376708 0.1173332 1.1075481 1.567486
## 42 1.345525 0.01390164 0.1179052 1.1144350 1.576615
## 43 1.456168 0.01475911 0.1214871 1.2180574 1.694278
## 44 1.366760 0.01447280 0.1203030 1.1309705 1.602549
## 45 1.359493 0.01394211 0.1180767 1.1280665 1.590919
## 46 1.482386 0.01455099 0.1206275 1.2459605 1.718811
## 47 1.319646 0.01368885 0.1169994 1.0903310 1.548960
## 48 1.274289 0.01301588 0.1140872 1.0506818 1.497895
## 49 1.424981 0.01461051 0.1208740 1.1880723 1.661889
## 50 1.391979 0.01465158 0.1210437 1.1547378 1.629220
##
## $pooled
##   estimate   variance     within     between        se    ci.lb    ci.ub
## 1 1.381236 0.02119312 0.01422289 0.006833556 0.1455786 1.095908 1.666565
##         df         r       fmi  m
## 1 29.55647 0.4900711 0.3701185 50
```

```
pool.validate.mids(est_int)
```

```
## $invididual
##          estimate          var           se         ci.lb         ci.ub
## 1  -9.097271e+14 4.503600e+12 2.122169e+06 -9.097271e+14 -9.097271e+14
## 2  -9.502595e+14 4.503600e+12 2.122169e+06 -9.502595e+14 -9.502595e+14
## 3  -2.242793e+15 4.503600e+12 2.122169e+06 -2.242793e+15 -2.242793e+15
## 4  -2.161728e+15 4.503600e+12 2.122169e+06 -2.161728e+15 -2.161728e+15
## 5  -1.112389e+15 4.503600e+12 2.122169e+06 -1.112389e+15 -1.112389e+15
## 6  -9.637703e+14 4.503600e+12 2.122169e+06 -9.637703e+14 -9.637703e+14
## 7  -9.907919e+14 4.503600e+12 2.122169e+06 -9.907919e+14 -9.907919e+14
## 8  -2.161728e+15 4.503600e+12 2.122169e+06 -2.161728e+15 -2.161728e+15
## 9  -2.170735e+15 4.503600e+12 2.122169e+06 -2.170735e+15 -2.170735e+15
## 10 -9.232379e+14 4.503600e+12 2.122169e+06 -9.232379e+14 -9.232379e+14
## 11 -9.907919e+14 4.503600e+12 2.122169e+06 -9.907919e+14 -9.907919e+14
## 12 -1.125900e+15 4.503600e+12 2.122169e+06 -1.125900e+15 -1.125900e+15
## 13 -1.152922e+15 4.503600e+12 2.122169e+06 -1.152922e+15 -1.152922e+15
## 14 -9.232379e+14 4.503600e+12 2.122169e+06 -9.232379e+14 -9.232379e+14
## 15 -1.098878e+15 4.503600e+12 2.122169e+06 -1.098878e+15 -1.098878e+15
## 16 -1.031324e+15 4.503600e+12 2.122169e+06 -1.031324e+15 -1.031324e+15
## 17 -8.962163e+14 4.503600e+12 2.122169e+06 -8.962163e+14 -8.962163e+14
## 18 -9.907920e+14 4.503600e+12 2.122169e+06 -9.907921e+14 -9.907920e+14
## 19 -1.098878e+15 4.503600e+12 2.122169e+06 -1.098878e+15 -1.098878e+15
## 20 -2.202260e+15 4.503600e+12 2.122169e+06 -2.202260e+15 -2.202260e+15
## 21 -7.746191e+14 4.503600e+12 2.122169e+06 -7.746191e+14 -7.746191e+14
## 22 -9.907919e+14 4.503600e+12 2.122169e+06 -9.907919e+14 -9.907919e+14
## 23 -1.017814e+15 4.503600e+12 2.122169e+06 -1.017814e+15 -1.017814e+15
## 24 -9.772811e+14 4.503600e+12 2.122169e+06 -9.772811e+14 -9.772811e+14
## 25 -1.004303e+15 4.503600e+12 2.122169e+06 -1.004303e+15 -1.004303e+15
## 26 -1.044835e+15 4.503600e+12 2.122169e+06 -1.044835e+15 -1.044835e+15
## 27 -1.017814e+15 4.503600e+12 2.122169e+06 -1.017814e+15 -1.017814e+15
## 28 -8.962164e+14 4.503600e+12 2.122169e+06 -8.962165e+14 -8.962164e+14
## 29 -9.637703e+14 4.503600e+12 2.122169e+06 -9.637703e+14 -9.637703e+14
## 30 -9.502595e+14 4.503600e+12 2.122169e+06 -9.502595e+14 -9.502595e+14
## 31 -1.085368e+15 4.503600e+12 2.122169e+06 -1.085368e+15 -1.085368e+15
## 32 -9.772811e+14 4.503600e+12 2.122169e+06 -9.772811e+14 -9.772811e+14
## 33 -1.004303e+15 4.503600e+12 2.122169e+06 -1.004303e+15 -1.004303e+15
## 34 -1.004303e+15 4.503600e+12 2.122169e+06 -1.004303e+15 -1.004303e+15
## 35  3.609198e+03 4.201505e+00 2.049757e+00  3.605181e+03  3.613216e+03
## 36 -9.772811e+14 4.503600e+12 2.122169e+06 -9.772811e+14 -9.772811e+14
## 37 -9.502596e+14 4.503600e+12 2.122169e+06 -9.502596e+14 -9.502596e+14
## 38 -1.166432e+15 4.503600e+12 2.122169e+06 -1.166432e+15 -1.166432e+15
## 39 -9.772811e+14 4.503600e+12 2.122169e+06 -9.772811e+14 -9.772811e+14
## 40 -9.637703e+14 4.503600e+12 2.122169e+06 -9.637703e+14 -9.637703e+14
## 41 -9.502595e+14 4.503600e+12 2.122169e+06 -9.502595e+14 -9.502595e+14
## 42 -1.031324e+15 4.503600e+12 2.122169e+06 -1.031324e+15 -1.031324e+15
## 43 -1.017814e+15 4.503600e+12 2.122169e+06 -1.017814e+15 -1.017814e+15
## 44  3.746843e+03 2.333721e+00 1.527652e+00  3.743849e+03  3.749837e+03
## 45 -9.502596e+14 4.503600e+12 2.122169e+06 -9.502596e+14 -9.502596e+14
## 46 -8.151515e+14 4.503600e+12 2.122169e+06 -8.151515e+14 -8.151515e+14
## 47 -1.112526e+15 4.503600e+12 2.122169e+06 -1.112526e+15 -1.112526e+15
## 48 -1.053842e+15 4.503600e+12 2.122169e+06 -1.053842e+15 -1.053842e+15
## 49 -1.004303e+15 4.503600e+12 2.122169e+06 -1.004303e+15 -1.004303e+15
## 50 -2.233785e+15 4.503600e+12 2.122169e+06 -2.233785e+15 -2.233785e+15
##
## $pooled
##        estimate     variance       within      between           se
## 1 -1.100232e+15 2.154871e+29 4.323456e+12 2.112618e+29 4.642059e+14
##           ci.lb         ci.ub df           r fmi  m
## 1 -2.010059e+15 -1.904054e+14  0 4.98414e+16   1 50
```

```
pool.oe.mids(est_log_oe)
```

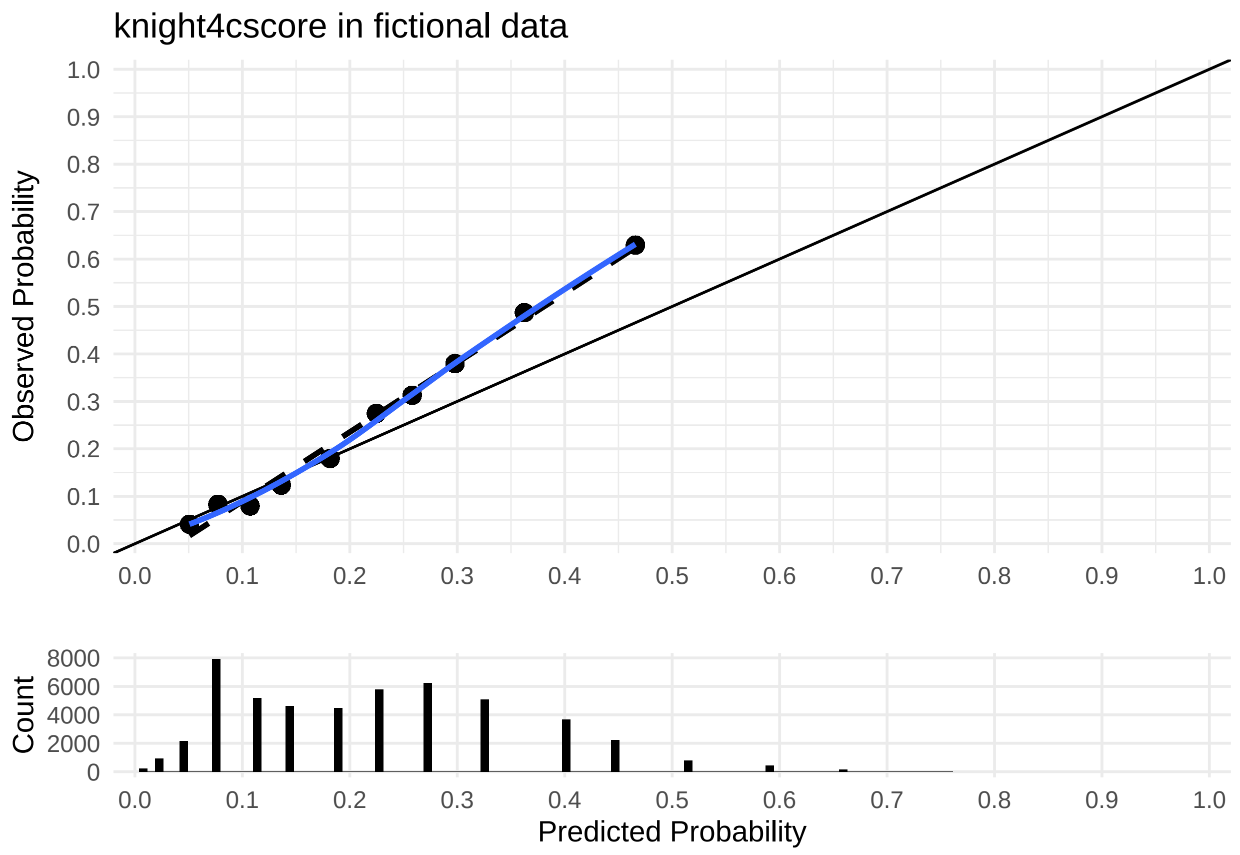
```
## $log_oe
## $log_oe$invididual
##      estimate         var         se        ci.lb     ci.ub
## 1  0.11063227 0.002759398 0.05252998  0.007675409 0.2135891
## 2  0.07455578 0.002802281 0.05293658 -0.029198003 0.1783096
## 3  0.04943308 0.002984064 0.05462658 -0.057633055 0.1564992
## 4  0.08630980 0.002846154 0.05334936 -0.018253016 0.1908726
## 5  0.05392494 0.002984064 0.05462658 -0.053141200 0.1609911
## 6  0.08679604 0.002816794 0.05307348 -0.017226058 0.1908181
## 7  0.06699018 0.002846154 0.05334936 -0.037572638 0.1715530
## 8  0.06676214 0.002846154 0.05334936 -0.037800675 0.1713250
## 9  0.05858273 0.002861004 0.05348835 -0.046252511 0.1634180
## 10 0.09596460 0.002773585 0.05266484 -0.007256582 0.1991858
## 11 0.07130272 0.002846154 0.05334936 -0.033260097 0.1758655
## 12 0.03548938 0.003000000 0.05477226 -0.071862267 0.1428410
## 13 0.04141708 0.003032258 0.05506594 -0.066510186 0.1493443
## 14 0.09407575 0.002773585 0.05266484 -0.009145435 0.1972969
## 15 0.04067971 0.002968254 0.05448168 -0.066102431 0.1474618
## 16 0.05636016 0.002891051 0.05376849 -0.049024145 0.1617445
## 17 0.10984891 0.002745318 0.05239579  0.007155052 0.2125428
## 18 0.07722492 0.002846154 0.05334936 -0.027337894 0.1817877
## 19 0.04067723 0.002968254 0.05448168 -0.066104911 0.1474594
## 20 0.05044002 0.002952569 0.05433755 -0.056059619 0.1569397
## 21 0.13042649 0.002623188 0.05121707  0.030042872 0.2308101
## 22 0.07928524 0.002846154 0.05334936 -0.025277575 0.1838481
## 23 0.05975016 0.002875969 0.05362806 -0.045358905 0.1648592
## 24 0.08579575 0.002831418 0.05321107 -0.018496022 0.1900875
## 25 0.07473325 0.002861004 0.05348835 -0.030101990 0.1795685
## 26 0.05564611 0.002906250 0.05390965 -0.050014858 0.1613071
## 27 0.05786619 0.002875969 0.05362806 -0.047242882 0.1629753
## 28 0.08473396 0.002745318 0.05239579 -0.017959895 0.1874278
## 29 0.07288578 0.002816794 0.05307348 -0.031136326 0.1769079
## 30 0.08727031 0.002802281 0.05293658 -0.016483475 0.1910241
## 31 0.05329583 0.002952569 0.05433755 -0.053203804 0.1597955
## 32 0.08995150 0.002831418 0.05321107 -0.014340279 0.1942433
## 33 0.08544208 0.002861004 0.05348835 -0.019393160 0.1902773
## 34 0.06580124 0.002861004 0.05348835 -0.039034005 0.1706365
## 35 0.07813238 0.002831418 0.05321107 -0.026159398 0.1824242
## 36 0.06763657 0.002831418 0.05321107 -0.036655202 0.1719283
## 37 0.06136739 0.002802281 0.05293658 -0.042386403 0.1651212
## 38 0.01124316 0.003048583 0.05521397 -0.096974245 0.1194606
## 39 0.06177152 0.002831418 0.05321107 -0.042520253 0.1660633
## 40 0.08937257 0.002816794 0.05307348 -0.014649535 0.1933947
## 41 0.07104704 0.002802281 0.05293658 -0.032706748 0.1748008
## 42 0.06336920 0.002891051 0.05376849 -0.042015109 0.1687535
## 43 0.08234975 0.002875969 0.05362806 -0.022759316 0.1874588
## 44 0.04554538 0.002952569 0.05433755 -0.060954254 0.1520450
## 45 0.06850880 0.002802281 0.05293658 -0.035244987 0.1722626
## 46 0.12147494 0.002663004 0.05160430  0.020332377 0.2226175
## 47 0.04609964 0.002984064 0.05462658 -0.060966499 0.1531658
## 48 0.06380806 0.002906250 0.05390965 -0.041852908 0.1694690
## 49 0.07497822 0.002861004 0.05348835 -0.029857027 0.1798135
## 50 0.03464157 0.002968254 0.05448168 -0.072140567 0.1414237
##
## $log_oe$pooled
##     estimate    variance      within      between         se       ci.lb
## 1 0.06983395 0.003393691 0.002862045 0.0005212219 0.05825539 -0.04434452
##       ci.ub      df         r       fmi  m
## 1 0.1840124 38.9591 0.1857575 0.1968556 50
##
##
## $oe
## $oe$individual
##    estimate     ci.lb    ci.ub
## 1  1.116984 1.0077049 1.238114
## 2  1.077405 0.9712241 1.195195
## 3  1.050675 0.9439963 1.169410
## 4  1.090144 0.9819126 1.210305
## 5  1.055405 0.9482461 1.174674
## 6  1.090674 0.9829215 1.210239
## 7  1.069285 0.9631245 1.187147
## 8  1.069041 0.9629049 1.186876
## 9  1.060333 0.9548008 1.177529
## 10 1.100720 0.9927697 1.220409
## 11 1.073906 0.9672869 1.192278
## 12 1.036127 0.9306591 1.153546
## 13 1.042287 0.9356534 1.161073
## 14 1.098643 0.9908963 1.218106
## 15 1.041518 0.9360350 1.158889
## 16 1.057979 0.9521581 1.175560
## 17 1.116109 1.0071807 1.236819
## 18 1.080285 0.9730324 1.199360
## 19 1.041516 0.9360327 1.158886
## 20 1.051734 0.9454828 1.169925
## 21 1.139314 1.0304987 1.259620
## 22 1.082513 0.9750392 1.201833
## 23 1.061571 0.9556544 1.179227
## 24 1.089584 0.9816740 1.209355
## 25 1.077597 0.9703466 1.196701
## 26 1.057223 0.9512153 1.175046
## 27 1.059573 0.9538557 1.177008
## 28 1.088427 0.9822004 1.206143
## 29 1.075608 0.9693434 1.193521
## 30 1.091192 0.9836516 1.210489
## 31 1.054742 0.9481867 1.173271
## 32 1.094121 0.9857621 1.214392
## 33 1.089198 0.9807937 1.209585
## 34 1.068014 0.9617180 1.186060
## 35 1.081266 0.9741798 1.200123
## 36 1.069976 0.9640085 1.187593
## 37 1.063289 0.9584993 1.179536
## 38 1.011307 0.9075794 1.126889
## 39 1.063719 0.9583711 1.180648
## 40 1.093488 0.9854572 1.213362
## 41 1.073632 0.9678223 1.191009
## 42 1.065420 0.9588553 1.183828
## 43 1.085836 0.9774977 1.206181
## 44 1.046599 0.9408663 1.164213
## 45 1.070910 0.9653689 1.187990
## 46 1.129161 1.0205405 1.249343
## 47 1.047179 0.9408548 1.165518
## 48 1.065888 0.9590108 1.184676
## 49 1.077861 0.9705843 1.196994
## 50 1.035249 0.9304001 1.151913
##
## $oe$pooled
##   estimate     ci.lb    ci.ub
## 1  1.07233 0.9566243 1.202031
```

```
pool.validate.mids(est_nb)
```

```
## $invididual
##    estimate          var          se         ci.lb      ci.ub
## 1     0.014 2.955293e-05 0.005436261  0.0033451251 0.02465487
## 2     0.004 2.671895e-05 0.005169038 -0.0061311276 0.01413113
## 3     0.009 2.538666e-05 0.005038517 -0.0008753127 0.01887531
## 4     0.008 2.171475e-05 0.004659909 -0.0011332536 0.01713325
## 5     0.005 3.091485e-05 0.005560112 -0.0058976202 0.01589762
## 6     0.011 3.080400e-05 0.005550135  0.0001219345 0.02187807
## 7     0.017 2.660018e-05 0.005157537  0.0068914139 0.02710859
## 8     0.014 2.504502e-05 0.005004500  0.0041913603 0.02380864
## 9     0.011 3.377374e-05 0.005811518 -0.0003903653 0.02239037
## 10    0.015 3.160383e-05 0.005621729  0.0039816140 0.02601839
## 11    0.013 2.774566e-05 0.005267415  0.0026760556 0.02332394
## 12    0.011 2.611461e-05 0.005110246  0.0009841026 0.02101590
## 13    0.012 2.903327e-05 0.005388253  0.0014392178 0.02256078
## 14    0.013 3.013796e-05 0.005489805  0.0022401794 0.02375982
## 15    0.011 2.950102e-05 0.005431484  0.0003544867 0.02164551
## 16    0.007 3.088949e-05 0.005557831 -0.0038931491 0.01789315
## 17    0.011 2.555187e-05 0.005054886  0.0010926064 0.02090739
## 18    0.011 3.004180e-05 0.005481040  0.0002573593 0.02174264
## 19    0.011 2.285893e-05 0.004781102  0.0016292126 0.02037079
## 20    0.011 2.919912e-05 0.005403621  0.0004090978 0.02159090
## 21    0.011 2.621479e-05 0.005120038  0.0009649096 0.02103509
## 22    0.010 2.668312e-05 0.005165571 -0.0001243328 0.02012433
## 23    0.011 2.245279e-05 0.004738437  0.0017128334 0.02028717
## 24    0.008 2.390310e-05 0.004889080 -0.0015824205 0.01758242
## 25    0.014 2.476883e-05 0.004976830  0.0042455932 0.02375441
## 26    0.015 2.654961e-05 0.005152631  0.0049010289 0.02509897
## 27    0.010 3.200621e-05 0.005657403 -0.0010883064 0.02108831
## 28    0.009 3.006393e-05 0.005483059 -0.0017465980 0.01974660
## 29    0.012 2.477810e-05 0.004977761  0.0022437682 0.02175623
## 30    0.008 2.350728e-05 0.004848431 -0.0015027500 0.01750275
## 31    0.012 2.678736e-05 0.005175651  0.0018559104 0.02214409
## 32    0.015 3.316481e-05 0.005758890  0.0037127839 0.02628722
## 33    0.009 2.963003e-05 0.005443347 -0.0016687647 0.01966876
## 34    0.016 2.846064e-05 0.005334851  0.0055438839 0.02645612
## 35    0.012 2.153942e-05 0.004641058  0.0029036937 0.02109631
## 36    0.010 2.899087e-05 0.005384317 -0.0005530675 0.02055307
## 37    0.014 3.322574e-05 0.005764178  0.0027024193 0.02529758
## 38    0.007 2.623385e-05 0.005121899 -0.0030387377 0.01703874
## 39    0.014 3.485413e-05 0.005903739  0.0024288849 0.02557112
## 40    0.008 2.610354e-05 0.005109162 -0.0020137741 0.01801377
## 41    0.016 2.739780e-05 0.005234291  0.0057409790 0.02625902
## 42    0.003 2.687804e-05 0.005184404 -0.0071612446 0.01316124
## 43    0.012 2.470514e-05 0.004970426  0.0022581438 0.02174186
## 44    0.015 2.411189e-05 0.004910386  0.0053758210 0.02462418
## 45    0.018 3.235341e-05 0.005688006  0.0068517136 0.02914829
## 46    0.013 3.407652e-05 0.005837510  0.0015586911 0.02444131
## 47    0.008 2.899690e-05 0.005384877 -0.0025541644 0.01855416
## 48    0.010 2.992171e-05 0.005470074 -0.0007211486 0.02072115
## 49    0.007 2.601574e-05 0.005100563 -0.0029969192 0.01699692
## 50    0.009 2.782943e-05 0.005275360 -0.0013395160 0.01933952
##
## $pooled
##   estimate    variance       within      between           se        ci.lb
## 1   0.0111 3.87844e-05 2.790787e-05 1.066327e-05 0.006227712 -0.001106091
##        ci.ub       df          r       fmi  m
## 1 0.02330609 32.15301 0.3897299 0.3213747 50
```

We use model_cal_plot to produce a calibration plot of all the imputed data sets stacked in one one data set. The advantage of this approach is that it retains the uncertainty that we modeled with mice. An issue that remains with this approach is that we cannot compute standard errors.

```
model_cal_plot(data_imp, "knight4cscore", data_name = "fictional data")
```



```
# model_cal_plot(data_imp, "knight4cmodel", data_name = "fictional data")
```

Alternatively, to reduce the number of plots, we could plot the validation results in each of the validation data sets in one plot, for each model. If all contributing partners send the predicted probabilities of the outcomes and the observed outcomes, we can do this. As the outcomes are then not connected to any other data they are fully anonymous.

```
data_stacked <- data.frame(complete(data_imp, seq_len(data_imp$m)))
p_stacked_knight4cscore <- knight4cscore(data_stacked)

oe <- data.frame(o = data_stacked$mortality,
                 knight4cscore = p_stacked_knight4cscore)
```