Multiple Imputation of Missing Data in Simple and More Complex Settings

Nicole Erler

Department of Biostatistics, Erasmus MC
https://nerler.com

FGME 2019, Kiel
15 September, 2019

Outline

Part I: Multiple Imputation

How does multiple imputation work?

- The ideas behind MI
- Understanding sources of uncertainty
- Implementation of MI and MICE

Part II: Multiple Imputation Workflow

How to perform MI with the **mice** package in R, from getting to know the data to the final results.

Practicals: visualization & exploartion of incomplete data, imputation with **mice**, checking imputed data, analysis of imputed data

Outline (cont.)

Part III: When MICE might fail

Introduction to

- settings where standard use of **mice** is problematic
- alternative imputation approaches
- alternative R packages

Practicals: Imputation with non-linear functional forms, longitudinal outcomes and survival outcomes

Part IV: Multiple Imputation Strategies

Some tips & tricks

Part I
Multiple Imputation

Part II
Multiple Imputation Workflow

Outline of Part II

1. Know your data

1.1. Missing data patterns

To demonstrate the work flow when performing multiple imputation with the **mice** package, we use data from the National Health and Nutrition Examination Survey (NHANES).

There are several packages in R that provide functions to create and **plot the missing data pattern**.

Examples are:
mice, **JointAI**, **VIM**, **Amelia**, **visdat**, **naniar**, . . .

1. Know your data

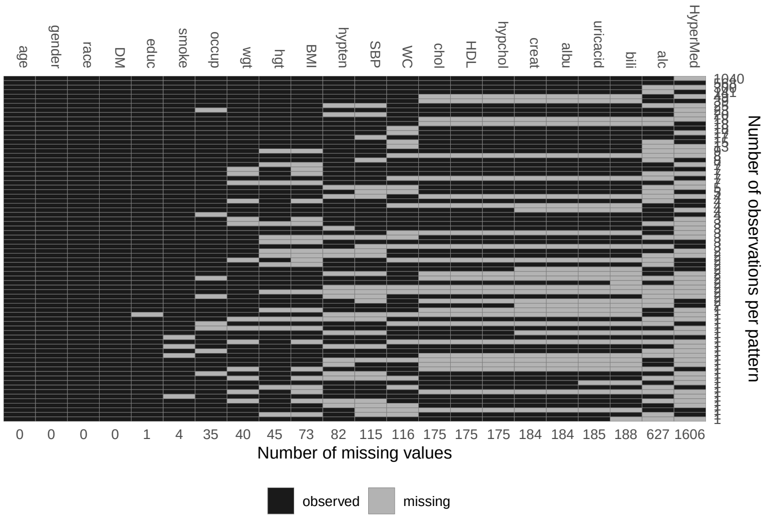
1.1. Missing data patterns

```
mdp <- mice::md.pattern(NHANES, plot = FALSE)
head(mdp[, -c(7:14)]) # omit some columns to fit it on the slide

##      age gender race DM educ smoke HDL hypchol creat albu uricacid bili alc HyperMed
## 568    1      1    1  1    1     1   1       1     1    1        1    1   1        1 0
## 1040   1      1    1  1    1     1   1       1     1    1        1    1   1        0 1
## 141    1      1    1  1    1     1   1       1     1    1        1    1   0        1 1
## 300    1      1    1  1    1     1   1       1     1    1        1    1   0        0 2
## 2      1      1    1  1    1     1   1       1     1    1        1    0   1        0 2
## 1      1      1    1  1    1     1   1       1     1    1        1    0   0        0 3

tail(mdp[, -c(7:14)])

##   age gender race DM educ smoke HDL hypchol creat albu uricacid bili alc HyperMed
## 1   1      1    1  1    1     0   1       1     1    1        1    1   1        1    1
## 1   1      1    1  1    1     0   1       1     1    1        1    1   1        0    2
## 1   1      1    1  1    1     0   0       0     0    0        0    0   0        0   10
## 1   1      1    1  1    1     0   1       1     1    1        1    1   1        0    4
## 1   1      1    1  1    0     1   0       0     0    0        0    0   1        0   12
##     0      0    0  0    1     4 175     175   184  184      185  188 627     1606 4010
```

1. Know your data

1.1. Missing data patterns

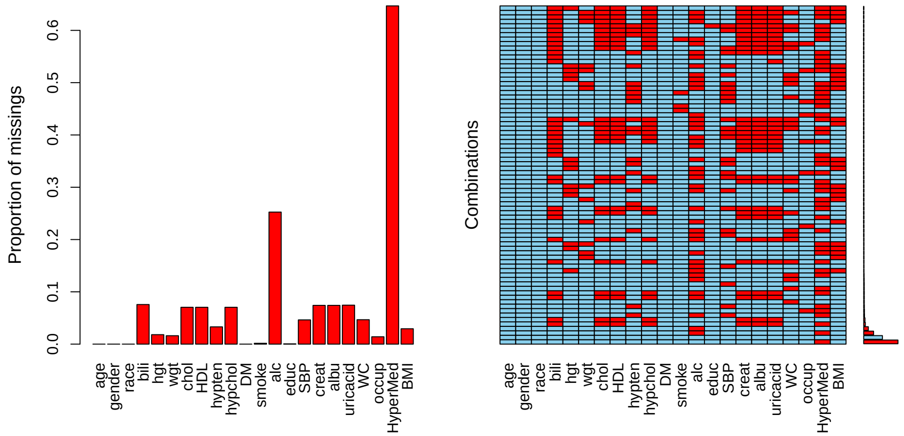
```
JointAI::md_pattern(NHANES)
```



1. Know your data

1.1. Missing data patterns

```r
VIM::aggr(NHANES, prop = TRUE, numbers = FALSE)
```



We are also interested in the number and proportion of (in)complete cases ...

```
cbind(
  "#" = table(ifelse(complete.cases(NHANES), 'incompl.', 'complete')),
  "%" = round(100 * table(complete.cases(NHANES))/nrow(NHANES), 2)
)

##               #     %
## complete 1915 77.12
## incompl.  568 22.88
```

1. Know your data

1.1. Missing data patterns

... and the proportion of missing values per variable:

```r
cbind("# NA" = sort(colSums(is.na(NHANES))),
      "% NA" = round(sort(colMeans(is.na(NHANES))) * 100, 2))
```

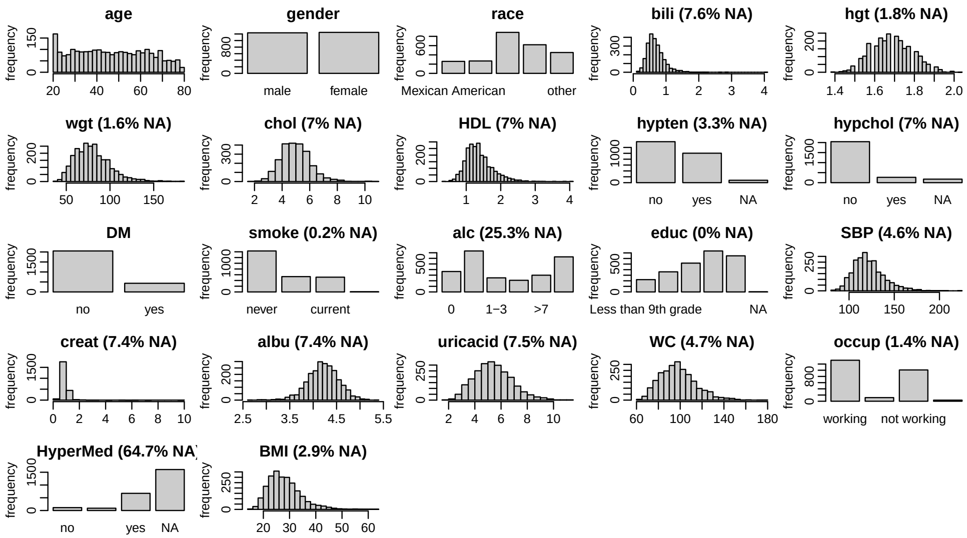
```
##          # NA % NA
## age         0 0.00
## gender      0 0.00
## race        0 0.00
## DM          0 0.00
## educ        1 0.04
## smoke       4 0.16
## occup      35 1.41
## wgt        40 1.61
## hgt        45 1.81
## BMI        73 2.94
## hypten     82 3.30
```

```
##            # NA  % NA
## SBP         115  4.63
## WC          116  4.67
## chol        175  7.05
## HDL         175  7.05
## hypchol     175  7.05
## creat       184  7.41
## albu        184  7.41
## uricacid    185  7.45
## bili        188  7.57
## alc         627 25.25
## HyperMed   1606 64.68
```

1. Know your data

1.2. Data distributions

```r
JointAI::plot_all(NHANES, nclass = 30)
```



1. Know your data

1.3. Correlations & patterns

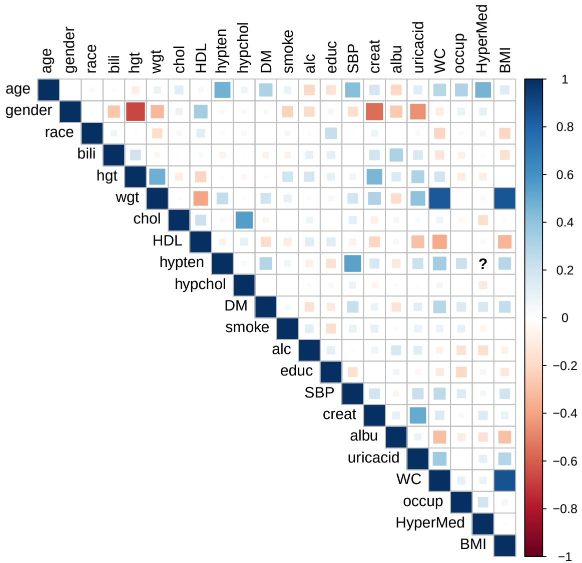
A quick (and dirty) way to check for strong correlations between variables is:

```
# re-code all variables as numeric and calculate spearman correlation
Corr <- cor(sapply(NHANES, as.numeric),
            use = "pairwise.complete.obs", method = "spearman")

## Warning in cor(sapply(NHANES, as.numeric), use =
"pairwise.complete.obs", :  the standard deviation is zero
```

```
corrplot::corrplot(Corr, method = "square", type = "upper",
                   tl.col = "black")
```

Note: We only use the correlation coefficient for categorical variables for visualization, not as a statistical result!

1.3. Correlations & patterns



1. Know your data

1.3. Correlations & patterns

Check out what the problem is with `hypertension` and `HyperMed`:

```
table(hypertension = NHANES$hypten,
      HyperMed = NHANES$HyperMed, exclude = NULL)
```

```
##              HyperMed
## hypertension   no previous  yes <NA>
##         no      0        0    0 1397
##         yes   114       90  673  127
##         <NA>    0        0    0   82
```

Knowing your data also means being able to answer these questions:

- Do missing values in multiple variables always **occur together**?
  (e.g. blood measurements)
- Are there **structural missing values**? (e.g. pregnancy status in men)
- Are there **patterns** in the missing values?
  (e.g. only patients with hypertension have observations of `HyperMed`)
- Are values **missing by design**?
- Is the **assumption of ignorable missingness** (MAR or MCAR) justifiable?

Auxiliary variables are variables that are not part of the analysis but **can help during imputation**.

Good auxiliary variables [17]

- are **related to the probability of missingness** in a variable, or
- are **related to the incomplete variable** itself,
- do **not have many missing values** themselves and
- are (mostly) **observed** when the incomplete variable of interest is missing.

The main arguments needed to impute data with `mice()` are:

- `data`: the dataset
- `m`: number of imputed datasets (default is 5)
- `maxit`: number of iterations (default is 5)
- `method`: vector of imputation methods
- `defaultMethod`: vector of default imputation methods for numerical, binary, unordered and ordered factors with > 2 levels
(default is `c("pmm", "logreg", "polyreg", "polr")`)
- `predictorMatrix`: matrix specifying roles of variables

2. Imputation with `mice()`
2.2. Imputation methods

mice has implemented many **imputation methods**, the most commonly used ones are:

- `pmm`: predictive mean matching (any)
- `norm`: Bayesian linear regression (numeric)
- `logreg`: binary logistic regression (binary)
- `polr`: proportional odds model (ordered factors)
- `polyreg`: polytomous logistic regression (unordered factors)

2. Imputation with mice()

2.2. Imputation methods

Change the default imputation method:
Example: To use norm instead of pmm for all continuous incomplete variables, use:

```
mice(NHANES, defaultMethod = c("norm", "logreg", "polyreg", "polr"))
```

Change the default imputation method:
Example: To use `norm` instead of `pmm` for all continuous incomplete variables, use:

```
mice(NHANES, defaultMethod = c("norm", "logreg", "polyreg", "polr"))
```

Change imputation method for a single variable:
To change the imputation method for single variables (but also for changes in other arguments) it is convenient to **do a setup run** of `mice()` without iterations (`maxit = 0`) and to extract and modify the parameters from there.

Change the default imputation method:
Example: To use norm instead of pmm for all continuous incomplete variables, use:

```
mice(NHANES, defaultMethod = c("norm", "logreg", "polyreg", "polr"))
```

Change imputation method for a single variable:
To change the imputation method for single variables (but also for changes in other arguments) it is convenient to **do a setup run** of mice() without iterations (maxit = 0) and to extract and modify the parameters from there.

Exclude variable from imputation:
When a variable that has missing values should not be imputed, the method needs to be set to "".

2. Imputation with mice()

2.2. Imputation methods

```r
library("mice")
imp0 <- mice(NHANES, maxit = 0)
meth <- imp0$method
meth

##        age    gender      race      bili       hgt       wgt
##         ""        ""        ""     "pmm"     "pmm"     "pmm"
##       chol       HDL    hypten   hypchol        DM     smoke
##      "pmm"     "pmm"  "logreg"  "logreg"        ""    "polr"
##        alc      educ       SBP     creat      albu  uricacid
##     "polr" "polyreg"     "pmm"     "pmm"     "pmm"     "pmm"
##         WC     occup  HyperMed       BMI
##      "pmm" "polyreg"    "polr"     "pmm"

meth["albu"] <- "norm"
meth["HyperMed"] <- ""
# imp <- mice(NHANES, method = meth)
```

2. Imputation with mice()

2.3. Predictor matrix

The `predictorMatrix` is a matrix that specifies **which variables are used as predictors** in which imputation model.
Each row represents the model for the variable given in the rowname.

```
head(imp0$predictorMatrix)[, 1:11]

##          age gender race bili hgt wgt chol HDL hypten hypchol DM
## age        0      1    1    1   1   1    1   1      1       1  1
## gender     1      0    1    1   1   1    1   1      1       1  1
## race       1      1    0    1   1   1    1   1      1       1  1
## bili       1      1    1    0   1   1    1   1      1       1  1
## hgt        1      1    1    1   0   1    1   1      1       1  1
## wgt        1      1    1    1   1   0    1   1      1       1  1
```

Variables **not used as predictor** are (or have to be set to) **zero**.

By **default, all variables** (except the variable itself) **are used** as predictors.

2. Imputation with `mice()`

2.3. Predictor matrix

Important:
A variable that has **missing values needs to be imputed** in order to be used as a predictor for other imputation models!!!

Note:
By default, **ALL** variables with missing values are imputed and **ALL** variables are used as predictor variables.

➡ Make sure to adjust the `predictorMatrix` and `method` to avoid using ID variables or other columns of the data that should not be part of the imputation.

➡ Make sure all **variables are coded correctly**, so that the automatically chosen imputation models are appropriate.

2. Imputation with mice()

2.3. Predictor matrix

```r
library(mice)
# setup-run
imp0 <- mice(NHANES, maxit = 0,
             defaultMethod = c("norm", "logreg", "polyreg", "polr"))

# adjust imputation methods
meth <- imp0$method
meth["educ"] <- "polr"
meth["HyperMed"] <- ""

# adjust predictor matrix
pred <- imp0$predictorMatrix
pred[, "HyperMed"] <- 0

# run imputation with adjusted settings
imp <- mice(NHANES, method = meth, predictorMatrix = pred,
            printFlag = FALSE)
```

In some cases, variables are **functions of other variables**, e.g., $BMI = \frac{wgt}{hgt^2}$.

If we impute BMI directly, its values may be **inconsistent** with the (imputed) values of hgt and wgt.

```
DF1 <- complete(imp, 1) # select the first imputed dataset
round(cbind("wgt/hgt^2" = DF1$wgt/DF1$hgt^2,
            BMI = DF1$BMI)[is.na(NHANES$BMI), ], 2)[1:5, ]

##      wgt/hgt^2   BMI
## [1,]     23.87 25.91
## [2,]     28.75 27.95
## [3,]     23.73 21.67
## [4,]     25.25 24.95
## [5,]     27.43 26.58
```

The imputed values of BMI are impossible given the corresponding values of hgt and wgt.

Moreover, if some components of a variable are observed we want to use that **information to reduce uncertainty**.

```
table(wgt_missing = is.na(NHANES$wgt),
      hgt_missing = is.na(NHANES$hgt))

##              hgt_missing
## wgt_missing FALSE TRUE
##       FALSE  2410   33
##       TRUE     28   12
```

Here we have $33 + 28 = 61$ cases in which either `hgt` or `wgt` is observed.

We would like to impute `hgt` and `wgt` separately and calculate `BMI` from the (imputed) values of the two variables.

If `BMI` is not a relevant predictor in any of the other imputation models, we could just exclude BMI from the imputation and **re-calculate it afterwards**.

To use `BMI` as predictor in the imputation, it has to be **calculated in each iteration** of the algorithm. In **mice** this is possible with **passive imputation**.

2. Imputation with `mice()`

2.4. Passive imputation

If `BMI` is not a relevant predictor in any of the other imputation models, we could just exclude BMI from the imputation and **re-calculate it afterwards**.

To use `BMI` as predictor in the imputation, it has to be **calculated in each iteration** of the algorithm. In **mice** this is possible with **passive imputation**.

Instead of using a standard imputation `method`, we can specify a formula to calculate `BMI`:

```
meth["BMI"] <- "~I(wgt/hgt^2)"      # formula to impute BMI
pred[c("wgt", "hgt"), "BMI"] <- 0  # prevent feedback
```

To **prevent feedback** from `BMI` in the imputation of `hgt` and `wgt` the `predictorMatrix` needs to be modified.

2. Imputation with `mice()`

2.4. Passive imputation

Since `BMI` depends on `wgt`, and the two variables are highly correlated (ρ =0.87) it may be beneficial **not to use them simultaneously** as predictors in the other imputation models.
Which one to use may differ between imputation models.

Passive imputation can also be useful in settings where

- imputation models include **interaction terms** between incomplete variables (see [17, p. 133] for an example), or when
- a number of covariates is used to form a **sum score**. The sum score, instead of all single elements, can then be used as predictor in other imputation models.

`mice()` has an argument `post` that can be used to specify functions that modify imputed values.

Helpful functions are

- `squeeze()` to censor variables at given boundaries
- ~~`ifdo()` for conditional manipulation~~ (not yet implemented)

2. Imputation with mice()

2.5. Post processing

mice() has an argument post that can be used to specify functions that modify imputed values.

Helpful functions are

- squeeze() to censor variables at given boundaries
- ~~ifdo() for conditional manipulation~~ (not yet implemented)

Example:
When inspecting the imputed values from imp, we find that some imputed values in creat are negative.

```
# DF1 is the first imputed dataset we extracted earlier
summary(DF1$creat)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.2829  0.7000  0.8400  0.8882  0.9900  9.5100
```

2. Imputation with `mice()`

2.5. Post processing

With the following syntax all imputed values of `creat` that are outside the interval `c(0, 100)` will be **set to those limiting values**.

```r
post <- imp$post
post["creat"] <- "imp[[j]][,i] <- squeeze(imp[[j]][,i], c(0, 100))"
imp2 <- update(imp, post = post, maxit = 20, seed = 123)
```

Note:

When many observations are outside the limits it may be better to **change the imputation model** since the implied **assumption of the imputation model** apparently **does not fit the** (assumption about the) **complete data distribution**.

This **post-processing** of imputed values allows for many **more data manipulations** and is not restricted to `squeeze()` (and `ifdo()`).

Any strings of R commands provided will be evaluated after the corresponding variable is imputed, within each iteration.

For example, if subjects with SBP > 140 should be classified as hypertensive:

```
post["hypten"] <- "imp[[j]][p$data[where[, j], 'SBP'] > 140, i] <- 'yes'"
```

This also allows for (some) **MNAR scenarios**, for example, by multiplying or adding a constant to the imputed values, or to re-impute values depending on their current value.

When the **post-processed or passively imputed values** of a variable depend on other variables, the **sequence in which the variables are imputed** may be important to obtain **consistent values**.

Example:
If `BMI` is passively imputed (calculated) before the new imputations for `hgt` and `wgt` are drawn, the resulting values of `BMI`, will match `hgt` and `wgt` from the **previous iteration**, but not the iteration given in the imputed dataset.

In `mice()` the argument `visitSequence` specifies in which order the columns of the `data` are imputed. By default `mice()` imputes in the order of the columns in `data`.

2. Imputation with mice()

2.6. Visit sequence

```
visitSeq <- imp2$visitSequence
visitSeq

##  [1] "age"      "gender"   "race"     "bili"     "hgt"      "wgt"
##  [7] "chol"     "HDL"      "hypten"   "hypchol"  "DM"       "smoke"
## [13] "alc"      "educ"     "SBP"      "creat"    "albu"     "uricacid"
## [19] "WC"       "occup"    "HyperMed" "BMI"
```

Currently, `hypten` is imputed before `SBP`, but the imputed values of `hypten` are post-processed depending on the current value of `SBP`. To get consistent values of these two variables, we need to change the `visitSequence`.

2. Imputation with mice()

2.6. Visit sequence

```
visitSeq <- c(visitSeq[-which(visitSeq == "hypten")],
              "hypten")
visitSeq
```

```
##  [1] "age"      "gender"   "race"     "bili"     "hgt"      "wgt"
##  [7] "chol"     "HDL"      "hypchol"  "DM"       "smoke"    "alc"
## [13] "educ"     "SBP"      "creat"    "albu"     "uricacid" "WC"
## [19] "occup"    "HyperMed" "BMI"      "hypten"
```

The `visitSequence` may specify that a column is visited multiple times during one iteration. All incomplete variables must be visited at least once.

```
visitSeq <- c(visitSeq[-which(visitSeq == "hypten")],
                "hypten")
visitSeq

##  [1] "age"      "gender"   "race"     "bili"     "hgt"      "wgt"
##  [7] "chol"     "HDL"      "hypchol"  "DM"       "smoke"    "alc"
## [13] "educ"     "SBP"      "creat"    "albu"     "uricacid" "WC"
## [19] "occup"    "HyperMed" "BMI"      "hypten"
```

The `visitSequence` may specify that a column is visited multiple times during one iteration. All incomplete variables must be visited at least once.

`visitSequence` can also be specified using one of the keywords `"roman"` (left to right), `"arabic"` (right to left), `"monotone"` (sorted in increasing amount of missingness), `"revmonotone"` (reverse of monotone).

2. Imputation with `mice()`
2.7. Good to know

`mice()` performs some **pre-processing** and **removes**

- incomplete variables that are not imputed but are specified as predictors,
- constant variables, and
- collinear variables.

In each iteration

- linearly dependent variables are removed and
- `polr` imputation models that do not converge are replaced by `polyreg`.

Why?
To avoid problems in the imputation models.

As a **consequence**

- imputation models may differ from what the user has specified or assumes is happening, or
- variables that should be imputed are not.

➡ Know your data
➡ Make sure `method` and `predictorMatrix` are specified appropriately
➡ Check the output and log of these automatic actions carefully

2. Imputation with `mice()`

A note

"Please realize that these choices are always needed. Imputation software needs to make default choices. These choices are intended to be useful across a wide range of applications. However, the **default choices are not necessarily the best for the data at hand. There is simply no magical setting that always works**, *so often some tailoring is needed." [17, p. 124]*

Practical

To practice the content of the previous section find the **html version** of the practical here:

https://nerler.github.io/EP16_Multiple_Imputation/practical/mimice/EP16_MImice.html

3. Convergence & diagnostics

3.1. Logged events

The log of the automatic changes is returned as part of the `mids` object:

```
demo <- NHANES[, 1:5]
demo$dupl <- demo[, 4]
demo$const <- 1
demo$age[demo$gender == 'male'] <- NA

demoimp <- mice(demo)
head(demoimp$loggedEvents)
```

```
## Warning:  Number of logged events:  8

##   it im dep      meth           out
## 1  0  0      constant         const
## 2  0  0     collinear          dupl
## 3  1  1 age       pmm genderfemale
## 4  1  2 age       pmm genderfemale
## 5  1  3 age       pmm genderfemale
## 6  2  1 age       pmm genderfemale
```

With columns

`it`	iteration number
`im`	imputation number
`dep`	dependent variable
`meth`	imputation method used
`out`	names of altered or removed predictors

Recall from slides **??** and **??**:
mice uses an **iterative algorithm** and imputations from the first few iterations may not be samples from the "correct" distributions.

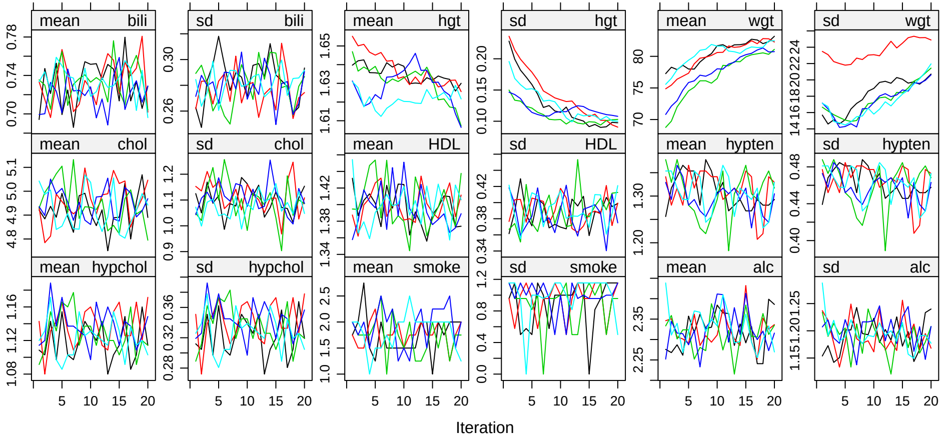
Traceplots can be used to visually assess **convergence**.

In **mice**, the function `plot()` produces traceplots of the mean and standard deviation (across subjects) per incomplete variable (see slide **??**).

3. Convergence & diagnostics

3.2. Convergence

```
plot(imp2, layout = c(6, 3))
```



Strong trends and traces that show **correlation** between variables indicate **problems of feedback**. This needs to be investigated and resolved in the specification of the `predictorMatrix`.

Weak trends may be artefacts that often disappear when the imputation is performed with more iterations.

When MCMC chains have converged, the **distributions of the imputed and observed values** can be compared to investigate differences between observed and imputed data.

Note:
Plots usually show the **marginal** distributions of observed and imputed values, which do not have do be identical under MAR.

Recall:
The **conditional** distributions (given all the other variables in the imputation model) of the imputed values are assumed to be the same as the conditional distributions of the observed data.

mice provides several functions for visual diagnosis of imputed values:

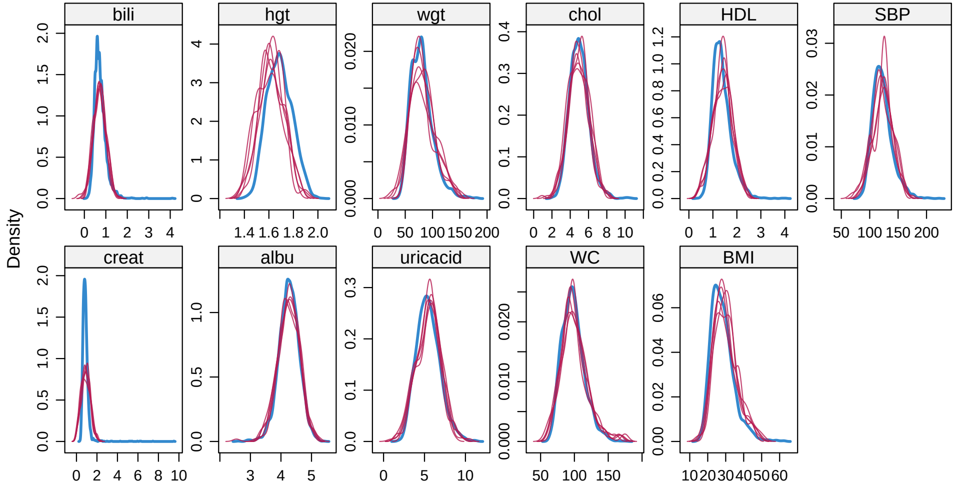
- `densityplot()` (for large datasets and variables with many NAs)
- `stripplot()` (for smaller datasets and/or variables with few NAs)
- `bwplot()`
- `xyplot()`

These functions create lattice graphics, which can be modified analogously to their parent functions from the **lattice** package.

3. Convergence & diagnostics

3.3. Diagnostics

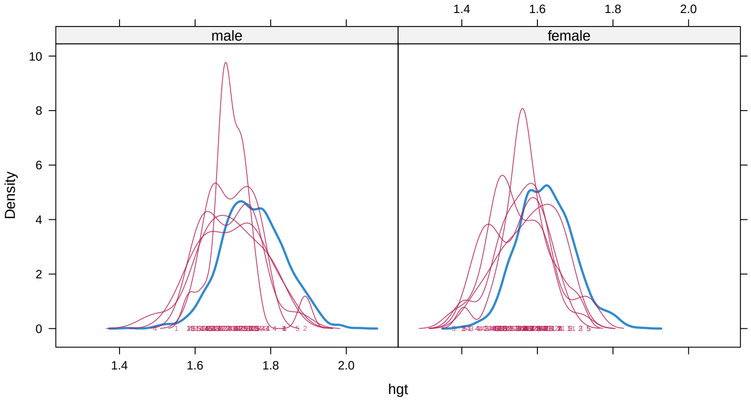
```
densityplot(imp2)
```



The `densityplot()` shows that the distribution of imputed values of `creat` is wider than the distribution of the observed values and that imputed values of `hgt` are smaller than the observed values.

In some cases differences in distributions can be explained by strata in the data, however, here, `gender` does not explain the difference in observed and imputed values.

```
densityplot(imp2, ~hgt|gender, plot.points = TRUE)
```



3. Convergence & diagnostics

3.3. Diagnostics

As an alternative, we might consider `race` to explain the differences

```
densityplot(imp2, ~hgt|race)

## Error in density.default(x = c(NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, :
need at least 2 points to select a bandwidth automatically
```

As an alternative, we might consider `race` to explain the differences

```
densityplot(imp2, ~hgt|race)

## Error in density.default(x = c(NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, :
need at least 2 points to select a bandwidth automatically
```

However, there are not enough missing values of `hgt` per categories of `race` to estimate densities.

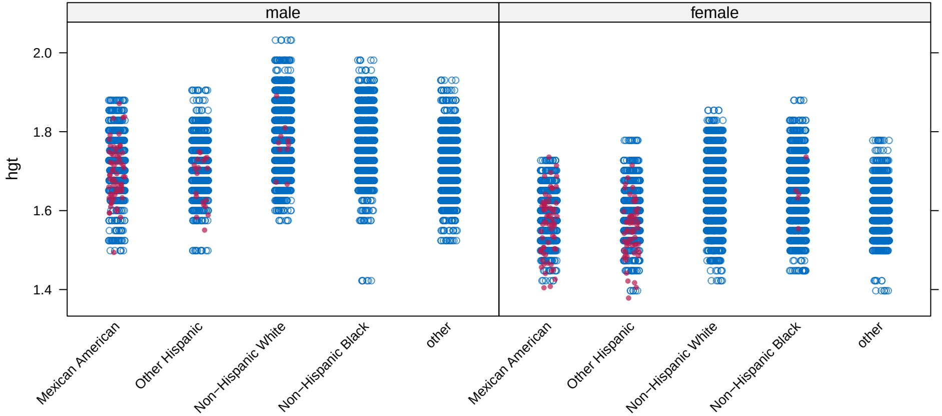
```
with(NHANES, table(race = race, "hgt missing" = is.na(hgt)))

##                     hgt missing
## race                 FALSE TRUE
##   Mexican American     233   26
##   Other Hispanic       252   16
##   Non-Hispanic White   884    2
##   Non-Hispanic Black   618    1
##   other                451    0
```

3. Convergence & diagnostics

3.3. Diagnostics

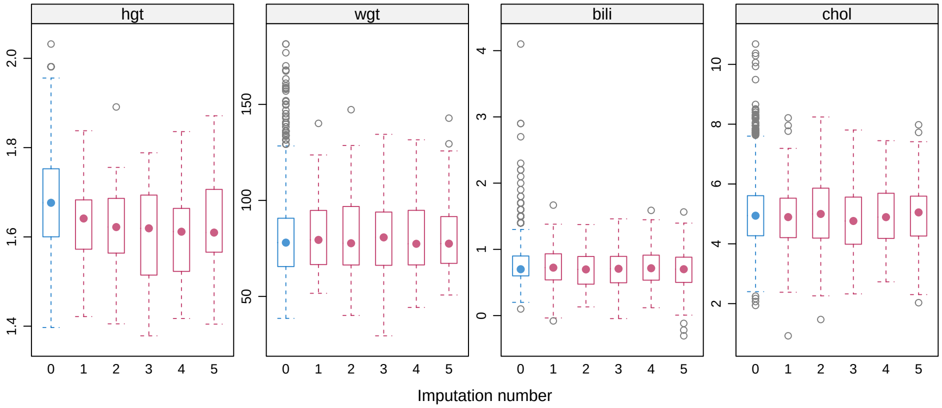
In that case, a `stripplot()` may be better suited. Here we can also split the data for `gender` and `race`.

```
stripplot(imp2, hgt ~ race|gender, pch = c(1, 20),
          scales = list(x = list(rot = 45)))
```



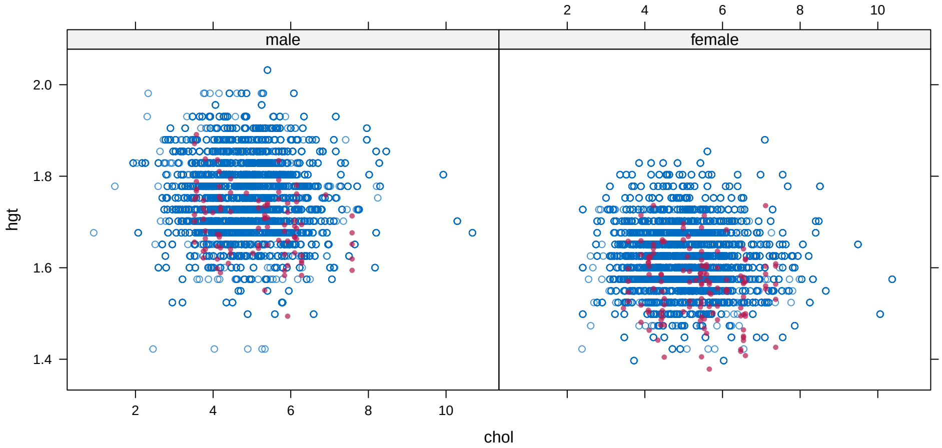
Alternatively, observed and imputed data can be represented by box-and-whisker plots:

```
bwplot(imp2, hgt + wgt + bili + chol ~.imp)
```



The function `xyplot()` allows multivariate investigation of the imputed versus observed values.

```
xyplot(imp2, hgt ~ chol|gender, pch = c(1,20))
```



3. Convergence & diagnostics

3.3. Diagnostics

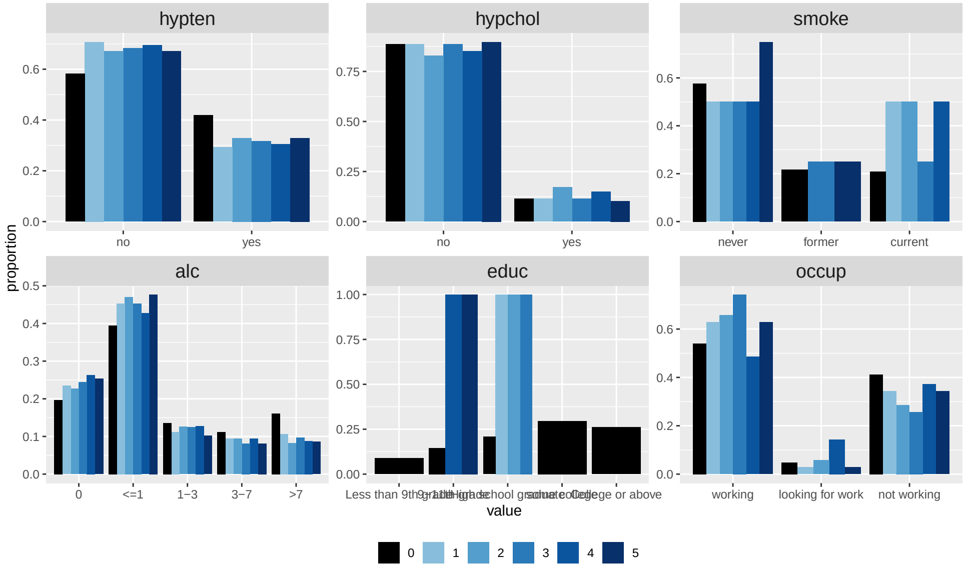
All of the above graphs displayed only continuous imputed variables.
For categorical variables we can compare the proportion of values in each category.

mice does not provide a function to do this, but we can write one ourselves, as for instance the function `propplot()`, for which the syntax can be found here:
https://gist.github.com/NErler/0d00375da460dd33839b98faeee2fdab

3. Convergence & diagnostics

3.3. Diagnostics

```
propplot(imp2, strip.text = element_text(size = 14))
```



`smoke` and `educ` have very few missing values (4 and 1, respectively), so we do not need to worry about differences between observed and imputed data for those variables.

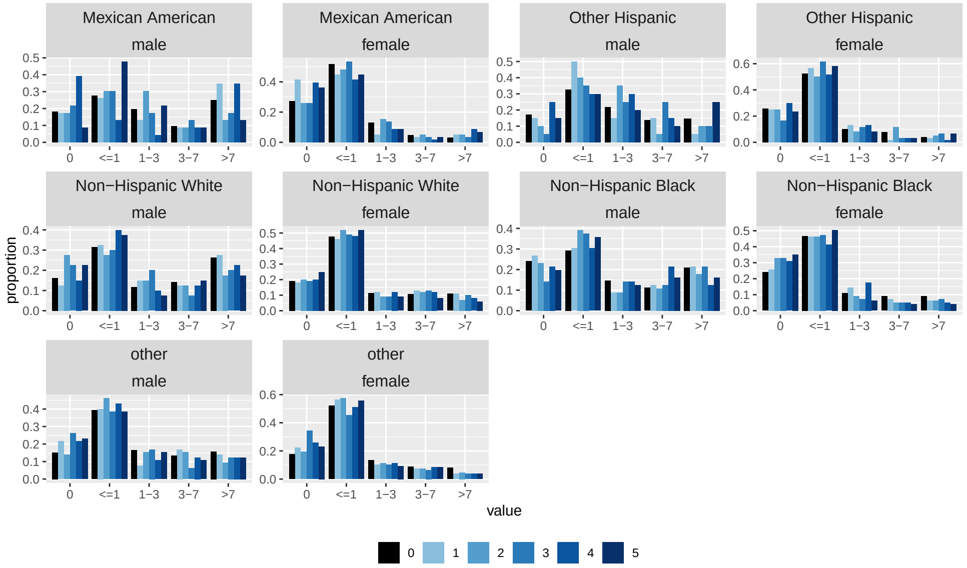
- `alc`: missing values are imputed in the lower consumption categories more often than we would expect from the observed data
- `hypten` is less frequent and
- `hypchol` a bit more frequent, in the imputed data compared to the observed.

If we expect that `gender` and `race` might explain the differences for `alc`, we can include those factors into the plot.

3. Convergence & diagnostics

3.3. Diagnostics

```
propplot(imp2, formula = alc ~ race + gender)
```



3. Convergence & diagnostics

3.3. Diagnostics

Since hypertension is more common in older individuals, we may want to investigate if age can explain the differences in imputed values of hypten.

```
round(sapply(split(NHANES[, "age"], addNA(NHANES$hypten)), summary), 1)

##             no  yes <NA>
## Min.      20.0 20.0 20.0
## 1st Qu.   28.0 47.0 30.0
## Median    38.0 59.0 38.5
## Mean      40.7 56.9 41.5
## 3rd Qu.   51.0 68.0 50.8
## Max.      79.0 79.0 78.0
```

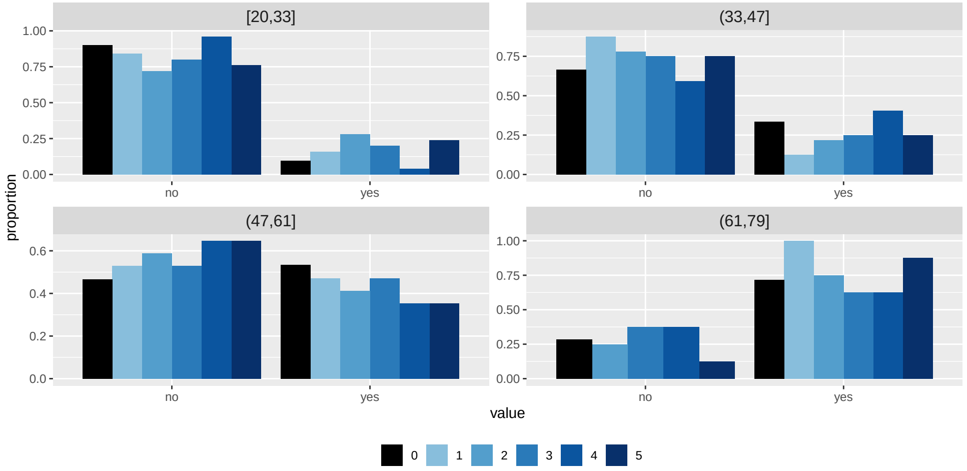
The table shows that the distribution of age in participants with missing hypten is very similar to the distribution of age in participants without hypten.

3. Convergence & diagnostics

3.3. Diagnostics

Plotting the proportions of observed and imputed `hypten` separately per quartile of `age`:

```r
propplot(imp2, formula = hypten ~ cut(age, quantile(age), include.lowest = T))
```



Practical

To practice the content of the previous section in an **interactive tutorial** go to

`https://emcbiostatistics.shinyapps.io/EP16_MIcheck`

or find the **html version** of the practical here:

`https://nerler.github.io/EP16_Multiple_Imputation/practical/micheck/EP16_MIcheck.html`

Once we have confirmed that our imputation was successful, we can move on to the **analysis of the imputed data**.

For example, we might be interested in the following logistic regression model:

```
glm(DM ~ age + gender + hypchol + BMI + smoke + alc,
    family = "binomial")
```

4. Analyse & pool the imputed data

4.1. Analysing imputed data

Once we have confirmed that our imputation was successful, we can move on to the **analysis of the imputed data**.

For example, we might be interested in the following logistic regression model:

```
glm(DM ~ age + gender + hypchol + BMI + smoke + alc,
    family = "binomial")
```

To fit the model on each of the imputed datasets, we do not need to extract the data from the `mids` object, but can use `with()`.

```
mod1 <- with(imp2, glm(DM ~ age + gender + hypchol + BMI + smoke + alc,
                       family = "binomial"))
```

`mod1` is an object of class `mira`.

4. Analyse & pool the imputed data

4.2. Pooling results

Pooled results can be obtained using `pool()` and its summary.

```
options(width = 90)
res1 <- summary(pool(mod1), conf.int = TRUE)
round(res1, 3)

##                estimate std.error statistic       df p.value  2.5 % 97.5 %
## (Intercept)      -7.484     0.404   -18.520 2146.569   0.000 -8.277 -6.692
## age               0.056     0.004    12.698 1363.291   0.000  0.047  0.065
## genderfemale     -0.424     0.127    -3.349 2333.764   0.001 -0.673 -0.176
## hypcholyes        0.009     0.201     0.043   87.509   0.966 -0.392  0.409
## BMI               0.105     0.009    11.509 2440.963   0.000  0.087  0.123
## smoke.L           0.063     0.116     0.545 2401.105   0.586 -0.165  0.291
## smoke.Q          -0.075     0.115    -0.650 2409.227   0.515 -0.300  0.151
## alc.L            -0.562     0.165    -3.402  197.275   0.001 -0.887 -0.236
## alc.Q             0.182     0.189     0.963   50.461   0.340 -0.197  0.560
## alc.C             0.017     0.188     0.089   49.989   0.930 -0.361  0.395
## alc^4            -0.045     0.206    -0.217   45.455   0.829 -0.460  0.371
```

4. Analyse & pool the imputed data

4.2. Pooling results

Pooling with `mice::pool()` is available for most types of models.

It extracts the model coefficients and variance-covariance matrices using `tidy()` from the package **broom**. Hence, pooling using the `pool()` function from **mice** only works for models of classes for which a method `tidy()` exists.

An alternative is offered by the package **mitools** and the function `MIcombine()`.

4. Analyse & pool the imputed data

4.3. Functions for pooled results

mice currently has two functions available for evaluating model fit / model comparison

For **linear** regression models the pooled R^2 can be calculated using `pool.r.squared()`.

```
mod2 <- with(imp2, lm(SBP ~ DM + age + hypten))
pool.r.squared(mod2, adjusted = TRUE)

##                est     lo 95     hi 95 fmi
## adj R^2 0.3252735 0.2943749 0.3562265 NaN
```

The argument `adjusted` specifies whether the adjusted R^2 or the standard R^2 is returned.

The function `pool.compare()` allows comparison of **nested models** (i.e., models where one is a special case of the other, with some parameters fixed to zero) using a **Wald test**.

Example: To test if `smoke` has a relevant contribution to the model for `DM` from above we re-fit the model without `smoke` and compare the two models:

```
mod3 <- with(imp2, glm(DM ~ age + gender + hypchol + BMI + alc,
                       family = "binomial"))
# Wald test
pool.compare(mod1, mod3)$pvalue

##            [,1]
## [1,] 0.6978098
```

`anova()` allows comparison of multiple nested models

4. Analyse & pool the imputed data

4.3. Functions for pooled results

The package **miceadds** extends **mice**, for example with the following functionality:

Combine χ^2 or F statistics from multiply imputed data:

```
miceadds::micombine.chisquare(dk, df, ...)
miceadds::micombine.F(values, df1, ...)
```

These functions take vectors of statistics computed on each imputed dataset and pool them.

4.3. Functions for pooled results

The package **miceadds** extends **mice**, for example with the following functionality:

Combine χ^2 or F statistics from multiply imputed data:

```
miceadds::micombine.chisquare(dk, df, ...)
miceadds::micombine.F(values, df1, ...)
```

These functions take vectors of statistics computed on each imputed dataset and pool them.

Calculate correlation or covariance of imputed data:

```
miceadds::micombine.cor(mi.res, ...)
miceadds::micombine.cov(mi.res, ...)
```

These functions take `mids` objects as input.

Practical

To practice the content of the previous section in an **interactive tutorial** go to

https://emcbiostatistics.shinyapps.io/EP16_AnalysisMI

or find the **html version** of the practical here:

https://nerler.github.io/EP16_Multiple_Imputation/practical/analysismi/EP16_AnalysisMI.html

5. Additional functions in `mice()`

5.1. Extract & export imputed data

The function `complete()` allows **extraction of the imputed data** from a `mids` object:

```
mice::complete(data, action = 1, include = FALSE, ...)
```

- `data`: the `mids` object
- `action`:
  - `1, ..., m` (single imputed dataset)
  - `"long"`: long format (imputed data stacked vertically)
  - `"broad"`: wide format (imputed data combined horizontally; ordered by imputation)
  - `"repeated"`: (like `"broad"`, but ordered by variable)
- `include`: include the original data?
  (if `action` is `"long"`, `"broad"` or `"repeated"`)

The function `mids2spss()` allows the **export of imputed data** (`mids` objects) to SPSS.

```
mids2spss(imp2,
          filedat = "datafile.txt", # the file containing the data
          filesps = "importsyntax.sps", # syntax to get .sav from .txt
          silent = TRUE, ...
)
```

Data from `mids` objects can also be exported to MPLUS using `mids2mplus()`.

5. Additional functions in mice()

5.2. Combining mids objects

To **increase the number of imputed datasets** without re-doing the initial *m* imputations, a second set of imputations can be done and the two `mids` objects combined using `ibind()`.

```
# same syntax as before, but different seed
imp2b <- update(imp2, post = post, maxit = 20, seed = 456)
imp2combi <- ibind(imp2, imp2b)
```

```
# check the new number of impute datasets:
imp2combi$m

## [1] 10
```

5. Additional functions in mice()

5.3. Adding variables to mids objects

The function cbind.mids() allows us to **add columns** to a mids object. The extra columns can either be a data.frame, matrix, vector or factor or another mids object.

For example data columns that should be part of the imputed data for completeness, but are not needed in the imputation.

```
extravar <- rnorm(nrow(NHANES))
impextra <- mice::cbind.mids(x = imp2, extravar = extravar)
```

Note: cbind() just adds columns to the data, you need to make sure they are **sorted correctly** so that the rows of the new data are from the same subjects as the corresponding rows in the imputed data.

Part III
When MICE might fail

Part IV
Multiple Imputation Strategies

References

References

[1] Jonathan W Bartlett, Shaun R Seaman, Ian R White, James R Carpenter, and Alzheimer's Disease Neuroimaging Initiative.
Multiple imputation of covariates by fully conditional specification: accommodating the substantive model.
Statistical methods in medical research, 24(4):462–487, 2015.

[2] James Carpenter and Michael Kenward.
Multiple imputation and its application.
John Wiley & Sons, 2012.

[3] Nicole S Erler, Dimitris Rizopoulos, Vincent WV Jaddoe, Oscar H Franco, and Emmanuel MEH Lesaffre.
Bayesian imputation of time-varying covariates in linear mixed models.
Statistical Methods in Medical Research, 28(2):555 – 568, 2019.

References (cont.)

[4] Nicole S Erler, Dimitris Rizopoulos, Joost van Rosmalen, Vincent WV Jaddoe, Oscar H Franco, and Emmanuel MEH Lesaffre.
Dealing with missing covariates in epidemiologic studies: a comparison between multiple imputation and a full Bayesian approach.
Statistics in Medicine, 35(17):2955–2974, 2016.

[5] John W Graham, Allison E Olchowski, and Tamika D Gilreath.
How many imputations are really needed? some practical clarifications of multiple imputation theory.
Prevention science, 8(3):206–213, 2007.

[6] Shahab Jolani.
Hierarchical imputation of systematically and sporadically missing data: An approximate bayesian approach using chained equations.
Biometrical Journal, 60(2):333–351, 2018.

References (cont.)

[7] Shahab Jolani, Thomas Debray, Hendrik Koffijberg, Stef Buuren, and Karel GM Moons.
Imputation of systematically missing predictors in an individual participant data meta-analysis: a generalized approach using mice.
Statistics in medicine, 34(11):1841–1863, 2015.

[8] Roderick JA Little.
Missing-data adjustments in large surveys.
Journal of Business & Economic Statistics, 6(3):287–296, 1988.

[9] Donald B Rubin.
Statistical matching using file concatenation with adjusted weights and multiple imputations.
Journal of Business & Economic Statistics, 4(1):87–94, 1986.

[10] Donald B. Rubin.
Multiple Imputation for Nonresponse in Surveys.
Wiley Series in Probability and Statistics. Wiley, 1987.

References (cont.)

[11] Donald B Rubin.
Multiple imputation after 18+ years.
Journal of the American statistical Association, 91(434):473–489, 1996.

[12] Donald B Rubin.
The design of a general and flexible system for handling nonresponse in sample surveys.
The American Statistician, 58(4):298–302, 2004.

[13] Joseph L Schafer.
Analysis of incomplete multivariate data.
CRC press, 1997.

[14] Joseph L Schafer and Recai M Yucel.
Computational strategies for multivariate linear mixed-effects models with missing values.
Journal of computational and Graphical Statistics, 11(2):437–457, 2002.

References (cont.)

[15] Nathaniel Schenker and Jeremy MG Taylor.
Partially parametric techniques for multiple imputation.
Computational statistics & data analysis, 22(4):425–446, 1996.

[16] Juned Siddique and Thomas R Belin.
Multiple imputation using an iterative hot-deck with distance-based donor selection.
Statistics in medicine, 27(1):83–102, 2008.

[17] Stef van Buuren.
Flexible Imputation of Missing Data.
Chapman & Hall/CRC Interdisciplinary Statistics. Taylor & Francis, 2012.

[18] Stef Van Buuren, Hendriek C Boshuizen, Dick L Knook, et al.
Multiple imputation of missing blood pressure covariates in survival analysis.
Statistics in Medicine, 18(6):681–694, 1999.

References (cont.)

[19] Gerko Vink and Stef van Buuren.
Multiple imputation of squared terms.
Sociological Methods & Research, 42(4):598–607, 2013.

[20] Ian R White and Patrick Royston.
Imputing missing covariate values for the cox model.
Statistics in medicine, 28(15):1982–1998, 2009.

[21] Ian R White, Patrick Royston, and Angela M Wood.
Multiple imputation using chained equations: issues and guidance for practice.
Statistics in medicine, 30(4):377–399, 2011.

[22] Recai M Yucel.
Multiple imputation inference for multivariate multilevel continuous data with ignorable non-response.
Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 366(1874):2389–2403, 2008.

Erasmus MC
University Medical Center Rotterdam

n.erler@erasmusmc.nl

N_Erler

NErler

www.nerler.com

Dep. Biostatistics: `www.erasmusmc.nl/biostatistiek`