How to impute missing values?

Genevieve Robin, Imke Mayer, Aude Sportisse

26 janvier 2021

Contents

The problem of missing data is ubiquitous in the practice of data analysis. Main approaches for handling missing data include imputation methods. In this Notebook, we first describe the main imputation methods available on R packages on synthetic data. Then, we compare them on both synthetic data for different missing-data mechanisms and percentage of missing values. Finally, we propose a function giving the comparison of the methods in one particular setting (missing-data mechanism, percentage of missing values) for a list of (complete) real datasets.

Description of imputation methods on synthetic data

In this section we provide, for some of the main packages (the list is of course not thorough) to impute missing values, links to vignettes and tutorials, as well as a description of their main functionalities and reusable code. The methods we focus on are gathered in the table below.

Package	Data Types	Underlying Method	Imputation	Computational Time	Comments
softImpute	quantitative	low-rank matrix completion with nuclear norm penalities	single	+	Very fast, strong theoretical guarantees, regularization parameter to tune

Package	Data Types	Underlying Method	Imputation	Computational Time	Comments
mice	mixed	multivariate imputation by chained equations	multiple	-	Very flexible to data types, no parameter to tune
missForest	mixed	random forests	single	-	Requires large sample sizes, no parameter to tune
missMDA	mixed	low-rank matrix completion with penality	single/multiple	+	Rank parameter to tune

```
library(Amelia)
library(mice)
library(missForest)
library(missMDA)
library(MASS)
library(softImpute)
library(dplyr)
library(tidyr)
library(ggplot2)
library(devtools)
```

Let us consider a gaussian data matrix of size n times p.

```
#### Simulation of the data matrix ####
set.seed(123)
n <- 1000
p <- 10
mu.X <- rep(1, 10)
Sigma.X <- diag(0.5, ncol = 10, nrow = 10) + matrix(0.5, nrow = 10, ncol =
10)
X <- mvrnorm(n, mu.X, Sigma.X)
head(X)
```

```
##            [,1]        [,2]       [,3]       [,4]      [,5]       [,6]
## [1,]  1.5055007  0.67922116  2.7536194  1.2679330 0.6046278  1.3820507
## [2,]  1.0378387  0.37754233  1.1173218  2.5266558 1.8242035  1.2260849
## [3,] -0.6442869 -0.06921970 -1.1689235 -1.5794364 0.9841500 -0.1273577
## [4,]  0.5859960  0.65298888  0.4614630  1.5170295 1.6277217  1.0015291
## [5,]  3.1452516  0.48649309  1.9582524 -0.1270187 1.5389924  1.8349234
## [6,] -0.5447334  0.01552382 -0.2932727 -0.1295848 0.0662147 -1.3493665
##            [,7]       [,8]       [,9]      [,10]
## [1,]  2.2882116  1.6928612  1.2250392  0.7575341
## [2,]  0.3781322  1.3943540  1.1698080  0.6551008
## [3,]  0.4227255  0.6584061 -0.2414336  0.2056861
## [4,]  0.6614958 -0.1730071  1.9422992  1.1995797
## [5,]  0.1036019 -0.4067036  1.4189708 -0.9115867
## [6,] -0.2420110  0.4353963 -1.2832304  0.6058015
```

We introduce some missing (here MCAR) values in the data matrix. One uses the function **produce_NA** detailed in “amputation.R” available in the related R source code of “How to generate missing values?”.

```
source_url('https://raw.githubusercontent.com/R-miss-tastic/website/master/static/how-to/generate/amput
```

```
## SHA-1 hash of file is a392b353c3ba88ecd276c2d94bd36009d5d40616
```

```
#### Introduction of missing values ####

XproduceNA <- produce_NA(X, mechanism = "MCAR", perc.missing = 0.3)
XNA <- as.matrix(as.data.frame(XproduceNA$data.incomp))
```

softImpute

The softImpute package can be used to impute quantitative data. It fits a low-rank matrix approximation to a matrix with missing values via nuclear-norm regularization. A vignette is available online, as well as the original article (Hastie et al. 2015).

The **softImpute** function computes, based on an incomplete dataset, a low-dimensional factorization which can be used to impute the missing values. The function is used as follows:

```
# perform softImpute
sft <- softImpute(x = XNA, rank.max = 2, lambda = 0, type = c("als", "svd"))
```

The main arguments are the following (more details can be found on the help page).

- `x`: the dataset with missing values (matrix).
- `rank.max`: the restricted rank of the solution, which should not be bigger than min(dim(x))-1.
- `lambda`: the nuclear-norm regularization parameter.
- `type`: indicates the algorithm which should be used, among "svd" and "als". "svd" returns an exact solution, while "als" returns an approximate solution (in exchange for a faster computation time).

To compute the imputed dataset based on the softImpute results, one may use the following code:

```
# compute the factorization
X.sft <- sft$u %*% diag(sft$d) %*% t(sft$v)
# replace missing values by computed values
X.sft[which(!is.na(XNA))] <- XNA[which(!is.na(XNA))]
```

To calibrate the parameter lambda, one may perform cross-validation, the code is given here. Then, the imputation procedure can be performed using the value of lambda computed with cross-validation (the other parameters are set to their default value):

```
source('https://raw.githubusercontent.com/R-miss-tastic/website/master/static/how-to/impute/CrossValida
```

```
## SHA-1 hash of file is a392b353c3ba88ecd276c2d94bd36009d5d40616
```

```
lambda_sft <- cv_sft(XNA)
sft <- softImpute(x = XNA, lambda = lambda_sft)
X.sft <- sft$u %*% diag(sft$d) %*% t(sft$v)
X.sft[which(!is.na(XNA))] <- XNA[which(!is.na(XNA))]
head(X.sft)
```

```
##            [,1]        [,2]       [,3]        [,4]      [,5]       [,6]
## [1,]  1.5055007  0.67922116  2.7536194  1.43715743 0.6046278  1.3820507
## [2,]  1.0378387  1.51791753  1.4387799  2.52665578 1.8242035  1.2260849
## [3,] -0.6442869 -0.45482432 -1.1689235 -1.57943643 0.9841500 -0.1273577
## [4,]  1.0878332  0.65298888  0.4614630  1.51702953 1.6277217  1.0015291
## [5,]  3.1452516  0.48649309  1.9582524 -0.12701866 1.6844646  1.8349234
## [6,] -0.4426038  0.01552382 -0.2932727 -0.02670137 0.0662147 -1.3493665
##             [,7]       [,8]       [,9]      [,10]
## [1,]  1.18801573  1.3702736  1.2988266  0.7575341
## [2,]  1.25696435  1.5996256  1.1698080  0.6551008
## [3,] -0.04888931  0.6584061 -0.2414336  0.2056861
## [4,]  0.66149581  1.0839748  1.9422992  1.1995797
## [5,]  0.10360187 -0.4067036  0.4620751 -0.9115867
## [6,] -0.24201096  0.4353963 -1.2832304  0.6058015
```

mice

The `mice` package implements a multiple imputation methods for multivariate missing data. It can impute mixes of continuous, binary, unordered categorical and ordered categorical data, as well as two-level data. The original article describing the software, as well as the source package (Buuren and Groothuis-Oudshoorn 2011) and example code are available online here.

The **mice** function computes, based on an incomplete dataset, multiple imputations by chained equations and thus returns m imputed datasets.

```
mice_mice <- mice(data = XNA, m = 5, method = "pmm") #contains m=5 completed datasets.
#mice::complete(mice_mice, 1) #get back the first completed dataset of the five available in mice_res
```

The main arguments are the following (more details can be found on the help page).

- `data`: the dataset with missing values (matrix).
- `m`: number of multiple imputations.
- `method`: the imputation method to use.

By default, the predictive mean matching method is performed. Other imputation methods can be used, type `methods(mice)` for a list of the available imputation methods.

We aggregate the complete datasets using the mean of the imputations to get a simple imputation.

```
IMP <- 0
for (i in 1:5) { IMP <- IMP + mice::complete(mice_mice, i)}
X.mice  <-  IMP/5  #5 is the default number of multiple imputations
head(X.mice)
```

```
##           X1         X2         X3         X4        X5         X6         X7
## 1  1.5055007 0.67922116  2.7536194  1.5987788 0.6046278  1.3820507  1.1496207
## 2  1.0378387 1.32678259  0.7449380  2.5266558 1.8242035  1.2260849  2.0402851
## 3 -0.6442869 0.52689556 -1.1689235 -1.5794364 0.9841500 -0.1273577  0.2679790
## 4  1.3743102 0.65298888  0.4614630  1.5170295 1.6277217  1.0015291  0.6614958
## 5  3.1452516 0.48649309  1.9582524 -0.1270187 0.5513352  1.8349234  0.1036019
## 6 -0.3057233 0.01552382 -0.2932727 -0.7408587 0.0662147 -1.3493665 -0.2420110
##           X8         X9        X10
## 1  1.3947788  1.2504049  0.7575341
## 2  1.5649229  1.1698080  0.6551008
## 3  0.6584061 -0.2414336  0.2056861
## 4  1.1756348  1.9422992  1.1995797
## 5 -0.4067036  0.9726412 -0.9115867
## 6  0.4353963 -1.2832304  0.6058015
```

missForest

The `missForest` package can be used to impute mixed-type data (continuous or categorical data).

The **missForest** function imputes missing values iteratively by training random forests. A vignette is available online as well as the original paper (Stekhoven and Buehlmann 2012).

```
forest <- missForest(xmis = XNA, maxiter = 20, ntree = 100)
```

The main arguments are the following (more details can be found on the help page).

- `xmis`: the dataset with missing values (matrix).
- `maxiter`: maximum number of iterations to be performed given the stopping criterion is not met beforehand.
- `ntree`: number of trees for each forest.

```
X.forest<- forest$ximp
head(X.forest)
```

```
##               X1         X2         X3         X4        X5         X6
## [1,]  1.50550068 0.67922116  2.7536194  0.7204680 0.6046278  1.3820507
## [2,]  1.03783866 1.20481156  1.2616144  2.5266558 1.8242035  1.2260849
## [3,] -0.64428691 0.11530188 -1.1689235 -1.5794364 0.9841500 -0.1273577
## [4,]  1.03164058 0.65298888  0.4614630  1.5170295 1.6277217  1.0015291
## [5,]  3.14525161 0.48649309  1.9582524 -0.1270187 0.9306504  1.8349234
## [6,] -0.01190722 0.01552382 -0.2932727 -0.4963537 0.0662147 -1.3493665
##              X7         X8         X9        X10
## [1,]  1.0707752  0.8909557  1.1125409  0.7575341
## [2,]  1.1633055  1.5091224  1.1698080  0.6551008
## [3,] -0.3936652  0.6584061 -0.2414336  0.2056861
## [4,]  0.6614958  1.1829947  1.9422992  1.1995797
## [5,]  0.1036019 -0.4067036  0.8686855 -0.9115867
## [6,] -0.2420110  0.4353963 -1.2832304  0.6058015
```

missMDA

The `missMDA` package serves to impute mixed-type data (continuous or categorical data).

The **imputePCA** function imputes missing values applying principal component methods. The missing values are predicted using the iterative PCA algorithm for a predefined number of dimensions. Some information are available in the original article (Josse and Husson 2016) and some videos are online here or here (in french).

```
pca <- imputePCA(X = XNA, ncp = 2, scale = TRUE, method = c("Regularized","EM"))
```

The main argument are the following (more details can be found on the help page).

- `X`: the dataset with missing values (matrix).
- `ncp`: number of components used to predict the missing entries.
- `scale`: if TRUE, it implies that the same weight is given for each variable.

The single imputation step requires tuning the number of dimensions used to impute the data. We use the function **estim_ncpPCA** which estimates the number of the dimensions using a cross-validation. Different cross-validation methods can be used to estimate the number of components, by default a generalized cross-validation is performed.

```
ncp.pca <- estim_ncpPCA(XNA,method.cv="gcv")$ncp
pca <- imputePCA(XNA, ncp = ncp.pca)
X.pca <- pca$comp
head(X.pca)
```

```
##              X1          X2         X3         X4        X5         X6
## [1,]  1.5055007  0.67922116  2.7536194  1.2726210 0.6046278  1.3820507
## [2,]  1.0378387  1.33062614  1.3160679  2.5266558 1.8242035  1.2260849
## [3,] -0.6442869 -0.05433620 -1.1689235 -1.5794364 0.9841500 -0.1273577
## [4,]  1.1114432  0.65298888  0.4614630  1.5170295 1.6277217  1.0015291
## [5,]  3.1452516  0.48649309  1.9582524 -0.1270187 0.7727156  1.8349234
## [6,] -0.2003831  0.01552382 -0.2932727 -0.1650226 0.0662147 -1.3493665
##               X7         X8         X9        X10
## [1,]  1.26132766  1.2308518  1.2264664  0.7575341
## [2,]  1.35708790  1.3301561  1.1698080  0.6551008
## [3,] -0.06973992  0.6584061 -0.2414336  0.2056861
## [4,]  0.66149581  1.0973240  1.9422992  1.1995797
## [5,]  0.10360187 -0.4067036  0.7301720 -0.9115867
## [6,] -0.24201096  0.4353963 -1.2832304  0.6058015
```

Numerical experiments to compare the different methods

Synthetic data

We compare the methods presented above for different percentage of missing values and for different missing-data mechanisms:

- Missing Completely At Random (MCAR) if the probability of being missing is the same for all observations
- Missing At Random (MAR) if the probability of being missing only depends on observed values.
- Missing Not At Random (MNAR) if the unavailability of the data depends on both observed and unobserved data such as its value itself.

We compare the methods in terms of mean squared error (MSE), i.e.:

$$MSE(X^{imp}) = \frac{1}{n_{NA}} \sum_i \sum_j 1_{X^{NA}_{ij}=NA}(X^{imp}_{ij} - X_{ij})^2$$

where $n_{NA} = \sum_i \sum_j 1_{X^{NA}_{ij}=NA}$ is the number of missing entries in X^{NA}.

Note that in order to evaluate this error, we need to know the true values of the missing entries.

```
MSE <- function(X, Xtrue, mask) {
  return(sqrt(sum((as.matrix(X) * mask - as.matrix(Xtrue) * mask) ^ 2) / sum(mask)))
}
```

The function **HowToImpute** compares the methods above with the naive imputation by the mean in terms of MSE on a complete dataset. More particularly, the function allows to introduce missing values on the complete dataset using different percentages of missing values and missing-data mechanisms and gives the MSE of the methods for the different missing-value settings. The final MSE for one specific missing-value setting is computed by aggregating the MSE's obtained for several simulations, where the stochasticity comes from the process of drawing several times the missing-data pattern.

The arguments are the following.

- `X`: the complete dataset where the missing values will be introduced (matrix).
- `perc.list`: list containing the different percentage of missing values.
- `mecha.list`: list containing the different missing-data mechanisms ("MCAR","MAR", "MNAR").
- `nbsim`: number of simulations performed.

It returns a table containing the mean of the MSEs for the simulations performed.

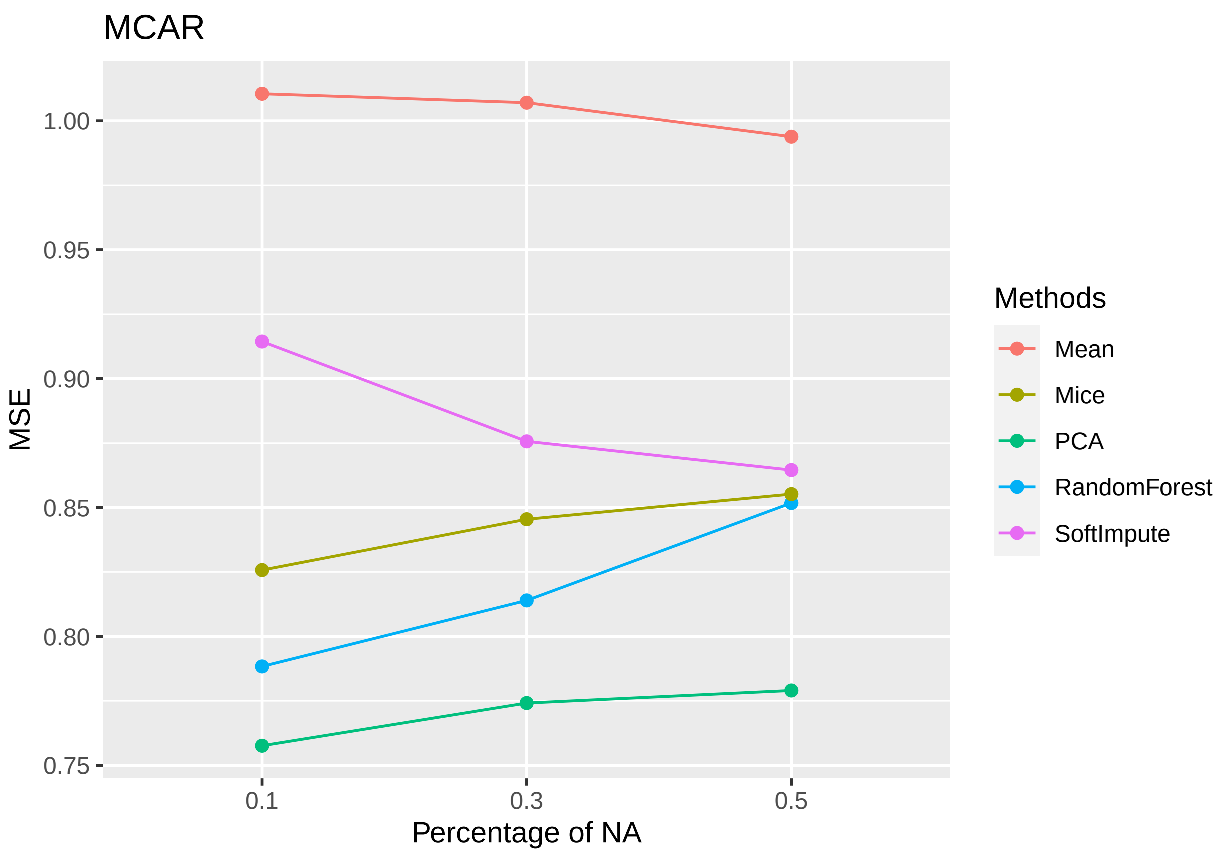
```
perc.list = c(0.1, 0.3, 0.5)
mecha.list = c("MCAR", "MAR", "MNAR")
res <- HowToImpute(X, perc.list = c(0.1, 0.3, 0.5), mecha.list = c("MCAR", "MAR", "MNAR"), nbsim = 2)
```

```
res
```

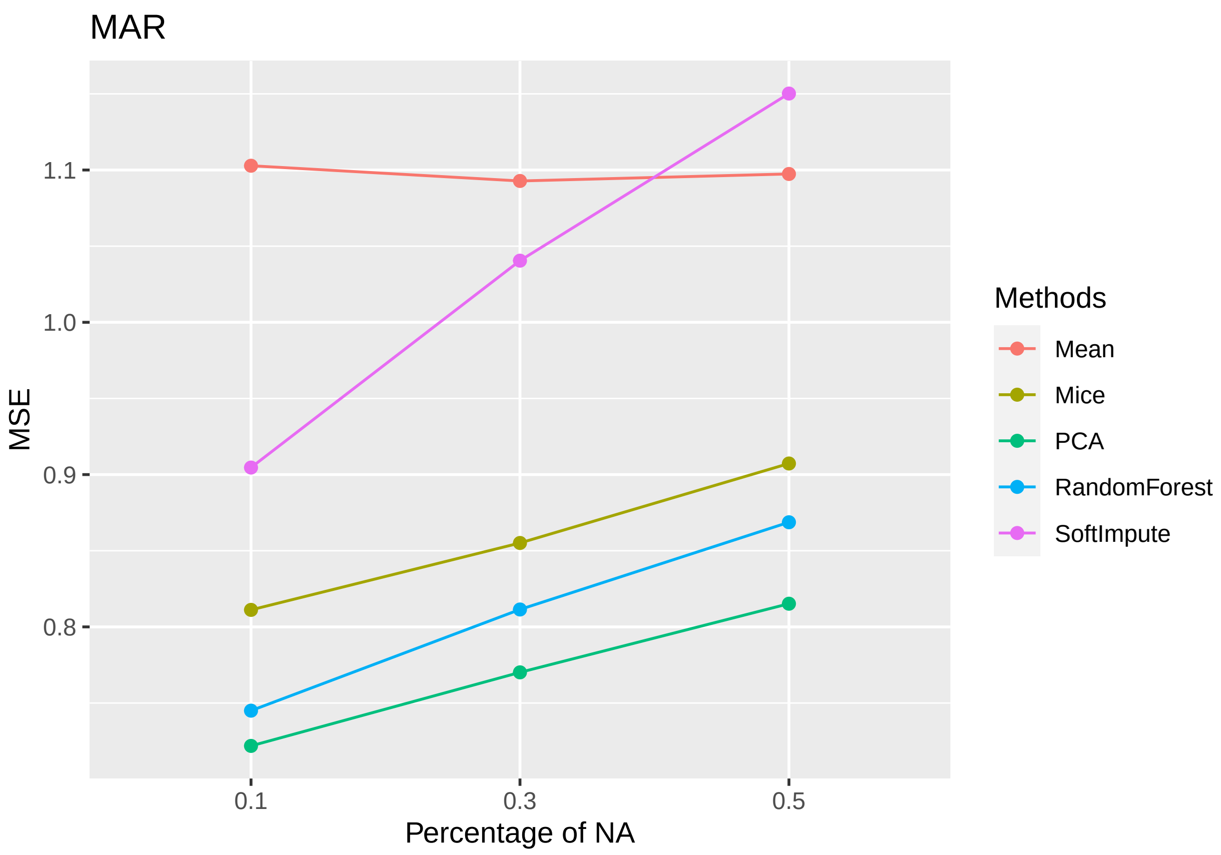
```
##            0.1 MCAR  0.3 MCAR  0.5 MCAR   0.1 MAR   0.3 MAR   0.5 MAR  0.1 MNAR
## X.pca    0.7576072 0.7741622 0.7790291 0.7218513 0.7701498 0.8152175 0.7533311
## X.forest 0.7883638 0.8139701 0.8517523 0.7449872 0.8114357 0.8687116 0.7725694
## X.mice   0.8257387 0.8454492 0.8552083 0.8111651 0.8551045 0.9072788 0.8269845
## X.soft   0.9143930 0.8756646 0.8645364 0.9046218 1.0404788 1.1501977 0.9738993
## X.mean   1.0105069 1.0070442 0.9938599 1.1028040 1.0927979 1.0973960 1.1503537
##            0.3 MNAR  0.5 MNAR
## X.pca    0.7899003 0.8161719
## X.forest 0.8231763 0.8733805
## X.mice   0.8750060 0.9133565
## X.soft   1.0349526 1.1835038
## X.mean   1.1161488 1.1265492
```

```
plotdf <- do.call(c, res)
plotdf <- as.data.frame(plotdf)
names(plotdf) <- 'mse'
meth <- rep(c("PCA", "RandomForest",  "Mice", "SoftImpute", "Mean"), length(perc.list) * length(mecha.l
plotdf <- cbind(plotdf, meth)
perc <- rep(rep(as.character(perc.list), each = 5),length(mecha.list))
plotdf <- cbind(plotdf, perc)
mecha <- rep(mecha.list, each = 5 * length(perc.list))
plotdf <- cbind(plotdf, mecha)
```

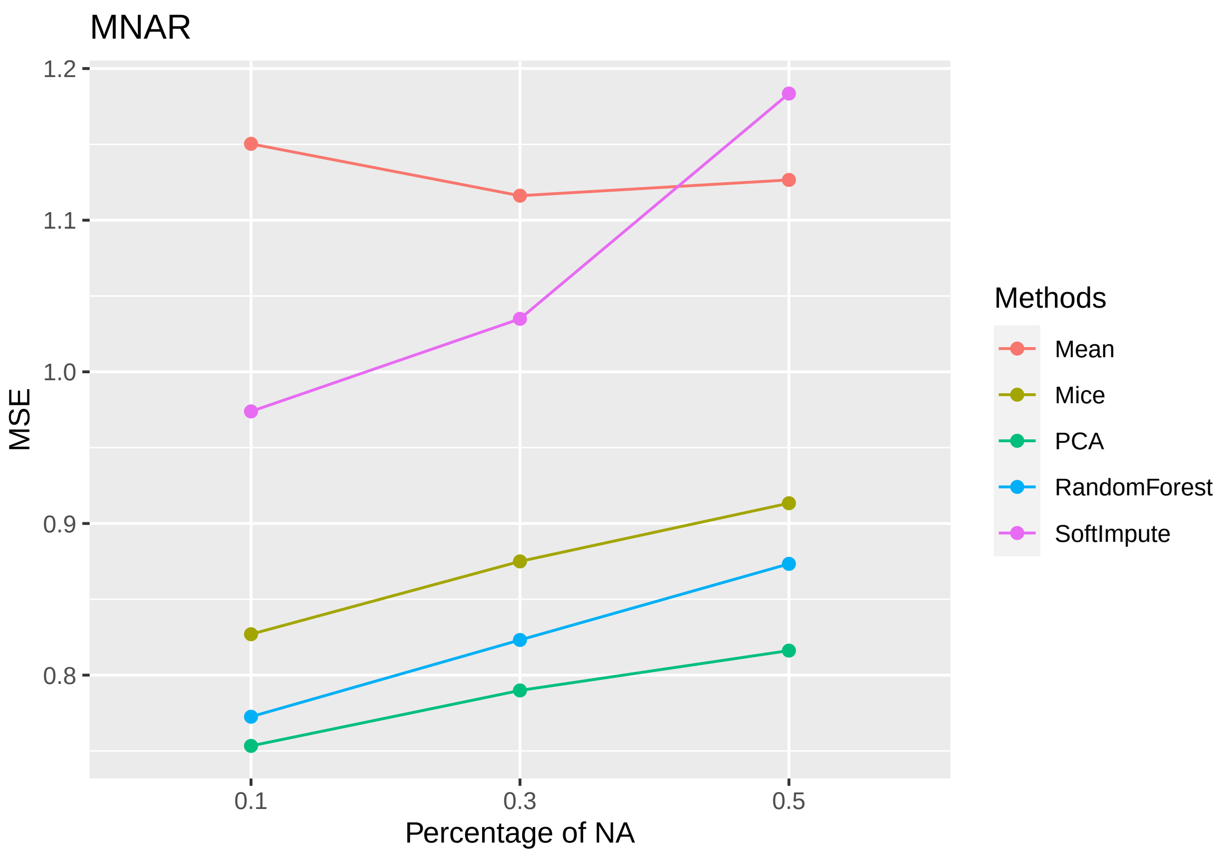
```
ggplot(plotdf[plotdf$mecha == "MCAR", ]) + geom_point(aes(x = perc, y = mse, color = meth), size = 1.8)
```



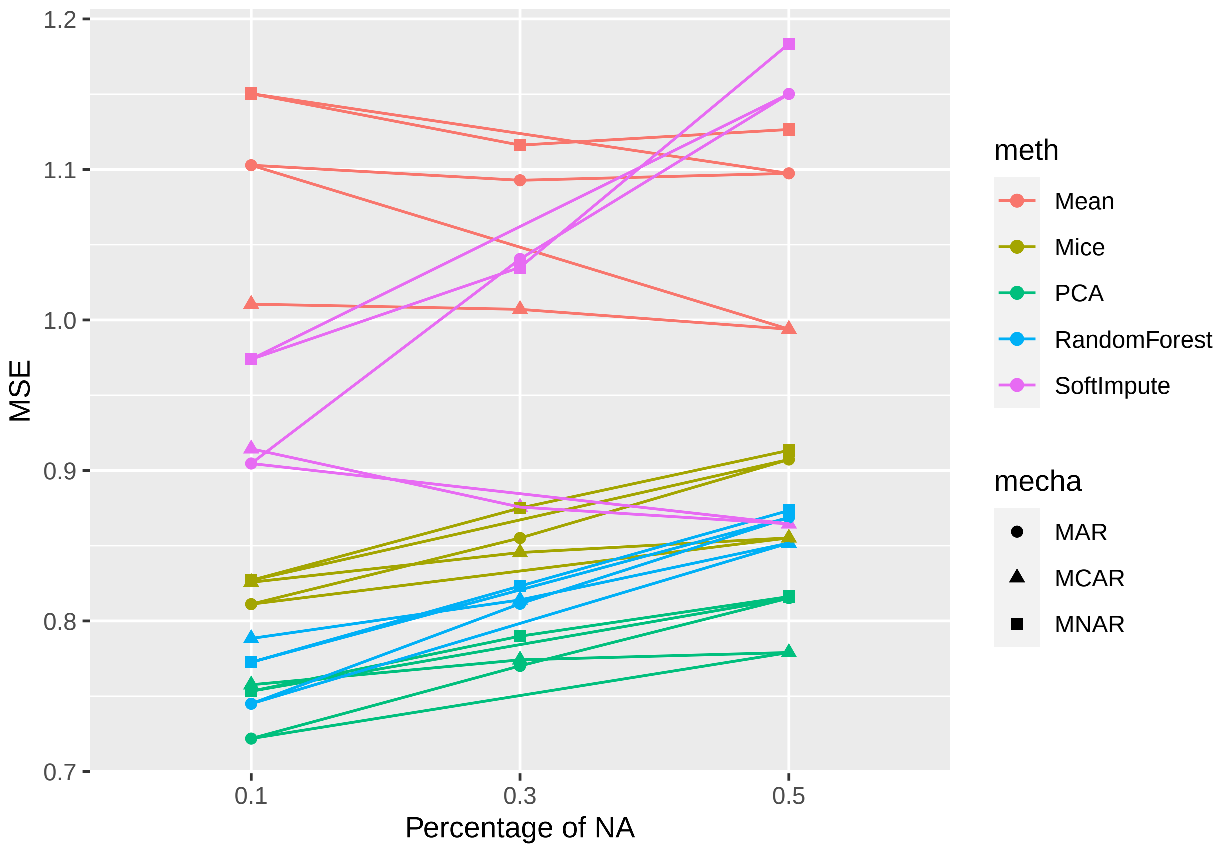
```
ggplot(plotdf[plotdf$mecha == "MAR", ]) + geom_point(aes(x= perc, y = mse, color = meth), size = 1.8) +
```



```
ggplot(plotdf[plotdf$mecha == "MNAR", ])+geom_point(aes(x = perc, y = mse,color = meth), size = 1.8) + 
```



```
ggplot(plotdf) + geom_point(aes(x = perc, y = mse, color = meth, shape = mecha), size = 1.8) + ylab("MSE
```



Real datasets

We will now compare the methods on real complete dataset taken from the UCI repository in which we will introduce missing values. In the present workflow, we propose a selection of several datasets (here, the datasets contain only quantitative variables) :

- Wine Quality - Red (1599x11)
- Wine Quality - White (4898x11)
- Slump (103x9)
- Movement (360x90)
- Decathlon (41x10)

But you can test the methods on any complete dataset you want.

```
wine_white <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequali
wine_white <- wine_white[, -ncol(wine_white)]

wine_red <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality
wine_red <- wine_red[, -ncol(wine_red)]

slump <-read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/concrete/slump/slump_test.da
slump <- slump[, -ncol(slump)]

movement <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/libras/movement_libras.d
movement  <- movement[, -ncol(movement)]

library(FactoMineR)
```

```r
data(decathlon)
decathlon <- decathlon[, 1:10]
```

You can choose to scale data prior to running the experiments, which implies that the variable have the same weight in the analysis. Scaling data may be performed on complete datasets but is more difficult for incomplete datasets[1].

```r
sc <- TRUE
if(sc){
  wine_white <- scale(wine_white)
  wine_red <- scale(wine_red)
  slump <- scale(slump)
  movement <- scale(movement)
  decathlon <- scale(decathlon)
}
```

We can then apply the **HowToImpute_real** function. It compares in terms of MSE several imputation methods for different complete datasets where missing values are introduced with a given percentage of missing values and a given missing-data mechanism.

The arguments are the following.

- `datasets_list`: dictionary of complete datasets.
- `perc`: percentage of missing values.
- `mecha`: missing-data mechanism ("MCAR","MAR" or "MNAR").
- `nbsim`: number of simulations performed.
- `names_dataset`: list containing the names of the datasets (for plotting results).

It returns a table containing the mean of the MSEs for the simulations performed.

```r
datasets_list <- list(
                  wine_white = wine_white,
                  wine_red = wine_red,
                  slump = slump,
                  movement = movement,
                  decathlon = decathlon
                 )
names_dataset <- c("winequality-white","winequality-red","slump","movement","decathlon")
perc <- 0.2
mecha <- "MCAR"
nbsim <- 2
howimp_real <-  HowToImpute_real(
                  datasets_list = list(
                    wine_white = wine_white,
                    wine_red = wine_red,
                    slump = slump,
                    movement = movement,
                    decathlon = decathlon
                  ) ,
                  perc = 0.2,
                  mech = "MCAR",
                  nbsim = 2,
```

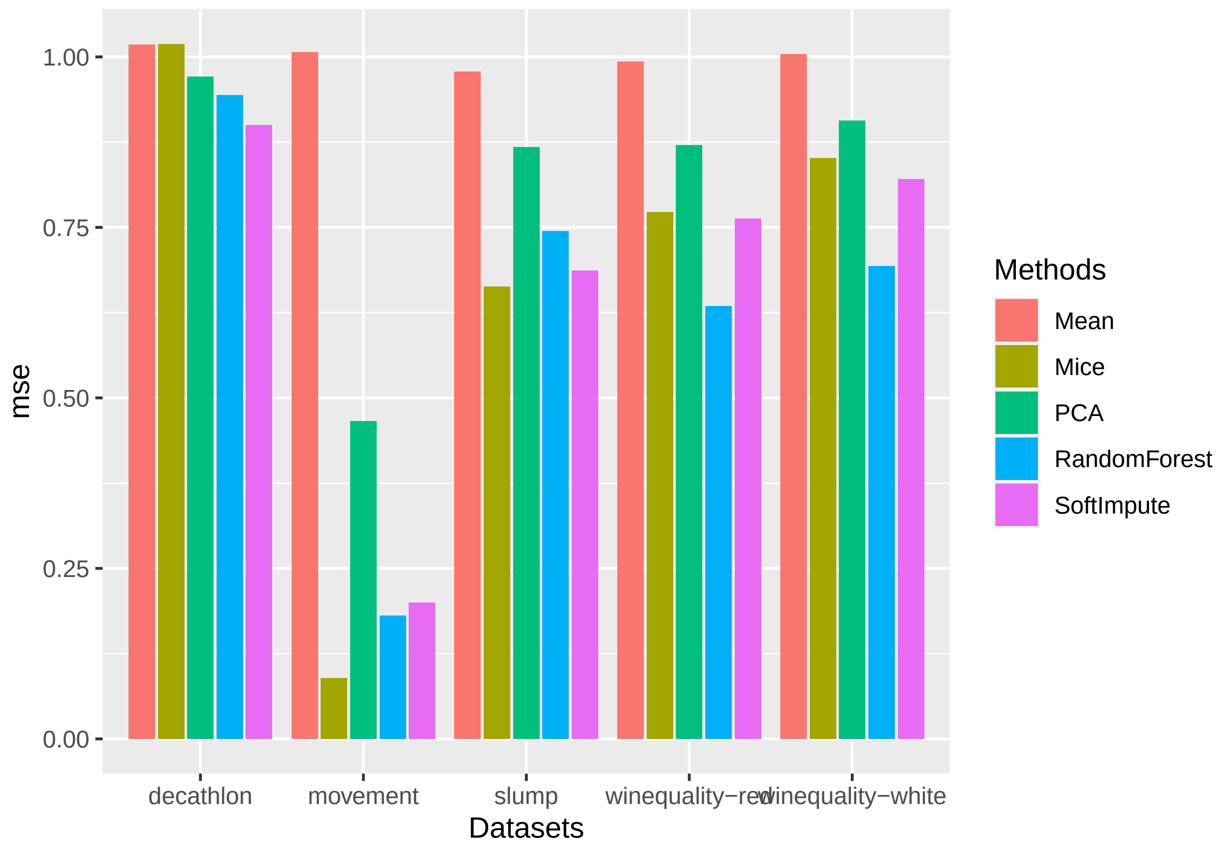
[1]For MCAR values, the estimations of the standard deviation can be unbiased. However, for MNAR values, the estimators will suffer from biases.

```
                  names_dataset = c(
                    "winequality-white",
                    "winequality-red",
                    "slump",
                    "movement",
                    "decathlon"
                  )
                )
plotdf_fin <- howimp_real$plot
res <- howimp_real$res
```

```
res
```

```
##             winequality-white winequality-red     slump  movement decathlon
## X.pca               0.9062828       0.8709428 0.8675272 0.4660087 0.9712461
## X.forest            0.6933932       0.6344437 0.7443832 0.1806609 0.9436397
## X.mice              0.8516444       0.7723729 0.6631778 0.0887146 1.0188197
## X.soft              0.8204904       0.7629633 0.6868858 0.1997366 0.8997186
## X.mean              1.0039933       0.9928671 0.9783253 1.0068527 1.0180233
```

```
ggplot(data=plotdf_fin, aes(x =Datasets, y = mse, fill=Methods)) + geom_bar(stat="identity", position=p
```



Session info

```
sessionInfo()
```

```
## R version 4.0.3 (2020-10-10)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Mojave 10.14.3
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] fr_FR.UTF-8/fr_FR.UTF-8/fr_FR.UTF-8/C/fr_FR.UTF-8/fr_FR.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] FactoMineR_2.4      mltools_0.3.5       gdata_2.18.0
##  [4] devtools_2.3.2      usethis_2.0.0       ggplot2_3.3.3
##  [7] tidyr_1.1.2         dplyr_1.0.3         softImpute_1.4
## [10] Matrix_1.2-18       MASS_7.3-53         missMDA_1.18
## [13] missForest_1.4      itertools_0.1-3     iterators_1.0.13
## [16] foreach_1.5.1       randomForest_4.6-14 mice_3.12.0
## [19] Amelia_1.7.6        Rcpp_1.0.6
##
## loaded via a namespace (and not attached):
##  [1] httr_1.4.2           pkgload_1.1.0        gtools_3.8.2
##  [4] assertthat_0.2.1     yaml_2.2.1           remotes_2.2.0
##  [7] ggrepel_0.9.1        sessioninfo_1.1.1    pillar_1.4.7
## [10] backports_1.2.1      lattice_0.20-41      glue_1.4.2
## [13] digest_0.6.27        colorspace_2.0-0     htmltools_0.5.1.1
## [16] pkgconfig_2.0.3      broom_0.7.3          purrr_0.3.4
## [19] mvtnorm_1.1-1        scales_1.1.1         processx_3.4.5
## [22] tibble_3.0.5         farver_2.0.3         generics_0.1.0
## [25] ellipsis_0.3.1       DT_0.17              withr_2.4.1
## [28] cli_2.2.0            magrittr_2.0.1       crayon_1.3.4
## [31] memoise_1.1.0        evaluate_0.14        ps_1.5.0
## [34] fs_1.5.0             fansi_0.4.2          doParallel_1.0.16
## [37] foreign_0.8-80       pkgbuild_1.2.0       data.table_1.13.6
## [40] tools_4.0.3          prettyunits_1.1.1    lifecycle_0.2.0
## [43] stringr_1.4.0        munsell_0.5.0        cluster_2.1.0
## [46] callr_3.5.1          flashClust_1.01-2    compiler_4.0.3
## [49] rlang_0.4.10         grid_4.0.3           htmlwidgets_1.5.3
## [52] leaps_3.1            labeling_0.4.2       rmarkdown_2.6
## [55] testthat_3.0.1       gtable_0.3.0         codetools_0.2-16
## [58] curl_4.3             R6_2.5.0             knitr_1.30
## [61] rprojroot_2.0.2      desc_1.2.0           stringi_1.5.3
## [64] parallel_4.0.3       vctrs_0.3.6          scatterplot3d_0.3-41
## [67] tidyselect_1.1.0     xfun_0.20
```

References

Buuren, Stef van, and Karin Groothuis-Oudshoorn. 2011. “mice: Multivariate Imputation by Chained Equations in R.” *Journal of Statistical Software* 45 (3): 1–67. http://www.jstatsoft.org/v45/i03/.

Hastie, Trevor, Rahul Mazumder, Jason D Lee, and Reza Zadeh. 2015. “Matrix Completion and Low-Rank Svd via Fast Alternating Least Squares.” *The Journal of Machine Learning Research* 16 (1). JMLR. org: 3367–3402.

Josse, Julie, and François Husson. 2016. “missMDA: A Package for Handling Missing Values in Multivariate Data Analysis.” *Journal of Statistical Software* 70 (1): 1–31. doi:10.18637/jss.v070.i01.

Stekhoven, Daniel J., and Peter Buehlmann. 2012. “MissForest - Non-Parametric Missing Value Imputation for Mixed-Type Data.” *Bioinformatics* 28 (1). Oxford Univ Press: 112–18.