clusterMI: Cluster Analysis with Missing Values by Multiple Imputation

`clusterMI` is an R package to perform clustering with missing values. Missing values are addressed by multiple imputation. The package offers various multiple imputation methods dedicated to clustered individuals (Audigier, Niang, and Resche-Rigon (2021)). In addition, it allows pooling results both in terms of partition and instability (Audigier and Niang (2022)). Among applications, these functionalities can be used to choose a number of clusters with missing values.

Contents

```
library(clusterMI)
```

1 Wine data set

The `wine` data set (Asuncion and Newman (2007)) is the result of a chemical analysis of 177 Italian wine samples from three different cultivars. Each wine is described by 13 continuous variables. This data set will be used to illustrate the `clusterMI` package. For achieving this goal, missing values will be added, and the cultivar variable will be omitted.

1.1 Full data set

```
require(stargazer)
set.seed(123456)
data(wine)
stargazer(wine, type = "text")
#>
#> ==========================================
#> Statistic  N   Mean   St. Dev.  Min    Max
#> ------------------------------------------
#> cult      178  1.900   0.780     1      3
#> alco      178 13.000   0.810   11.000 15.000
#> malic     178  2.300   1.100   0.740  5.800
#> ash       178  2.400   0.270   1.400  3.200
#> alca      178 19.000   3.300   11.000 30.000
#> mg        178 100.000  14.000    70    162
#> phe       178  2.300   0.630   0.980  3.900
#> fla       178  2.000   1.000   0.340  5.100
#> nfla      178  0.360   0.120   0.130  0.660
#> pro       178  1.600   0.570   0.410  3.600
#> col       178  5.100   2.300   1.300  13.000
#> hue       178  0.960   0.230   0.480  1.700
#> ratio     178  2.600   0.710   1.300  4.000
#> prol      178 747.000 315.000   278   1,680
#> ------------------------------------------
table(wine$cult)
#>
#>  1  2  3
#> 59 71 48
```

1.2 Adding missing values

Missing values are artificially added according to a missing completely at random mechanism so that each value of the data set is missing with a probability of 1/3 (independently to the values themselves).

```
ref <- wine$cult # "True" partition
nb.clust <- 3 # Number of clusters
wine.na <- wine
wine.na$cult <- NULL # Remove the reference partition
wine.na <- prodna(wine.na, pct = 1/3)
```

```
# proportion of missing values
colMeans(is.na(wine.na))
#>  alco malic   ash  alca    mg   phe   fla  nfla   pro   col   hue ratio  prol
```

```
#>  0.34  0.43  0.32  0.37  0.30  0.32  0.28  0.35  0.27  0.33  0.36  0.33  0.34

# proportion of incomplete individuals
mean(apply(is.na(wine.na), 1, any))
#> [1] 0.99
```

2 Multiple imputation

The `clusterMI` package offers various multiple imputation methods dedicated to clustered individuals. They can be divided into two categories: joint modelling (JM) imputation and fully conditional specification (FCS). The first assumes a joint distribution for all variables and imputation is performed using this model. The second proceeds variable per variable in a sequential manner by regression. FCS methods are more time consuming, but also more flexible, allowing a better fit of the imputation model. Multiple imputation is performed with the `imputedata` function. We start by presenting JM approaches in Section (2.1), while FCS approaches will be presented in Section (2.2).

2.1 Joint modelling imputation

The package proposes two JM methods: JM-GL and JM-DP. Both are based on a multivariate gaussian mixture model.

- `JM-GL` is implemented in the `mix` package (Joseph L. Schafer. (2022)). Initially, this method is dedicated to the imputation of mixed data, but it can be used by considering the partition variable as fully incomplete categorical variable. The method assumes constant variance in each cluster (for continuous data).
- `JM-DP` is a joint modelling method implemented in the R packages `DPImputeCont` (Kim (2020)), `NPBayesImputeCat` (Wang et al. (2022)), `MixedDataImpute` (Murray and Reiter (2016)) for continuous, categorical or mixed data respectively. Such a method has the advantage to automatically determine a number of clusters (but this number needs to by bounded by `nb.clust`). Furthermore, it allows various covariance matrices according to the clusters (for continuous data).

JM-GL is the default imputation method used in `imputedata`. To perform multiple imputation with this default method, we proceed as follows:

```
m <- 20 # Number of imputed data sets
res.imp.JM <- imputedata(data.na = wine.na,
                         nb.clust = nb.clust,
                         m = m)
```

and we specify `method = "JM-DP"` for imputation using the other JM method:

```
res.imp <- imputedata(data.na = wine.na,
                         method = "JM-DP",
                         nb.clust = nb.clust,
                         m = m)
```

Convergence

Both imputation methods consist in a data-augmentation algorithm alternating data imputation and drawing from a posterior distribution. The `m` imputed datasets are obtained by keeping one imputed dataset every `L` iterations. The first `Lstart` iterations consists in a burn-in period, which is required to reach convergence to the posterior distribution (expected from incomplete data). The `L` iterations between successive draws guarantee an independance between imputed values.

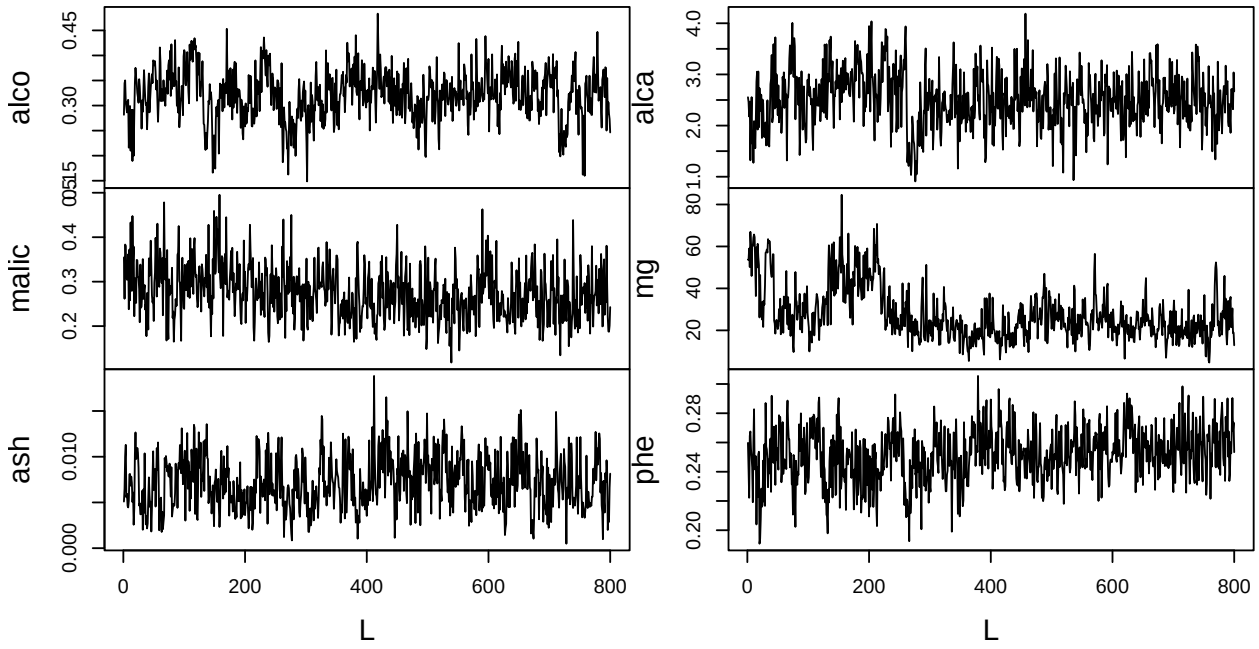
`Lstart` and `L` can be checked by graphical investigations. For achieving this goal, we track the between inertia for each variable over successive iterations of the data-augmentation algorithm (available in the `res.conv`

output from the `imputedata` function). In practice, we run imputation for a large number of imputed datasets `m` by keeping all intermediate imputed datatsets, i.e. at each iteration (`L = 1`) from the first (`Lstart = 1`):

```
res.imp.JM.conv <- imputedata(data.na = wine.na,
                          method = "JM-GL",
                          nb.clust = nb.clust,
                          m = 800,
                          Lstart = 1, # number of iterations for the burn-in period
                          L = 1 # number of iterations between each draw
                          )
```

and then plot the successive between inertia values for the six first variables (as an example):

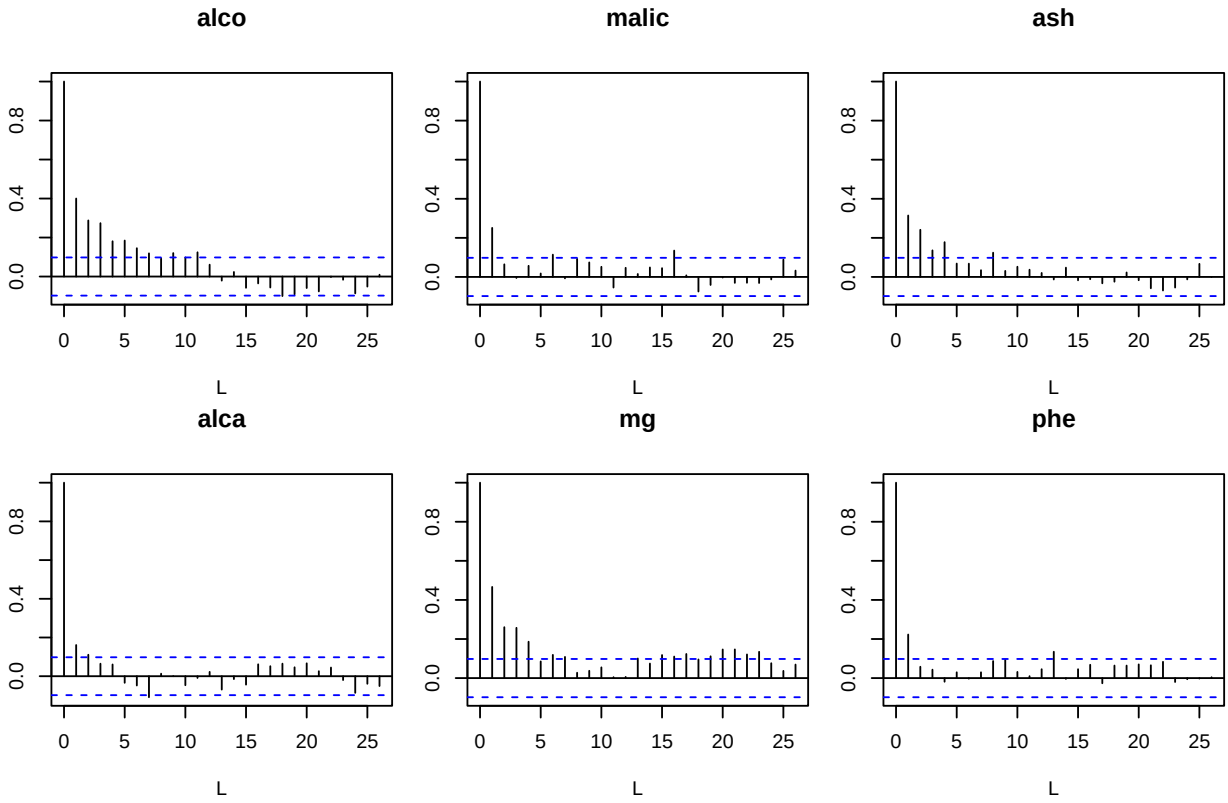
```
res.conv <- res.imp.JM.conv$res.conv
res.conv.ts <- ts(t(res.conv)) # conversion as time-series object
plot(res.conv.ts[, 1:6]) # diagnostic from the 6 first variables
```



Here, convergence seems to be reached at 400 iterations, meaning `Lstart = 400` is a suitable choice.

Next, the number of iterations between successive draws can be checked by visualising the autocorrelograms. Here, an autocorrelogram represents the correlation between the vector of the successive between inertia and its shifted version for several lags. We seek to find a lag `L` sufficiently large to avoid correlation between the vectors of between inertia. Such graphics can be obtained as follows:

```
Lstart <- 400
# extraction of summaries after Lstart iterations for the 6 first variables
res.conv.ts <- res.conv.ts[Lstart:nrow(res.conv.ts), 1:6]
apply(res.conv.ts, 2, acf)
```



Following such graphics `L = 20` (the default value) seems sufficient. Thus, the imputation step can be rerun as follows:

```
Lstart <- 400
L <- 20
res.imp.JM <- imputedata(data.na = wine.na,
                         nb.clust = nb.clust,
                         Lstart = Lstart,
                         L = L,
                         m = m)
```

Note that the imputation also requires a pre-specified number of clusters (`nb.clust`). Here it is tuned to 3 (corresponding to the number of varieties). We explain in Section 4.3 how it can be tuned. Furthermore, the number of imputed data sets is tuned to `m = 20` which is generally enough (this choice will be discussed in Section 4.2).

2.2 Fully conditional specification

Fully conditional specification methods consist in a variable per variable imputation. The two fully conditional imputation methods proposed are `FCS-homo` and `FCS-hetero` (Audigier, Niang, and Resche-Rigon (2021)). They essentially differ by the assumption about the covariance in each cluster (constant or not respectively).

To perform multiple imputation, we proceed as follows:

```
maxit <- 20 # Number of iterations for FCS imputation, should be larger in practice
res.imp.FCS <- imputedata(data.na = wine.na,
                          method = "FCS-homo",
                          nb.clust = nb.clust,
                          maxit = maxit,
                          m = m)
```

With FCS methods, the `imputedata` function alternates cluster analysis and imputation given the partition of individuals. When the cluster analysis is performed, the `imputedata` function calls the `mice` function from the mice R package (van Buuren and Groothuis-Oudshoorn (2011)). The `mice` package proposes various methods for imputation. By default, `imputedata` uses the default method used in `mice` (predictive mean matching for continuous data), but others can be specified by tuning the `method.mice` argument. For instance, for imputation under the normal model, use

```
imputedata(data.na = wine.na,
           method = "FCS-homo",
           nb.clust = nb.clust,
           maxit = maxit,
           m = m,
           method.mice = "norm")
```

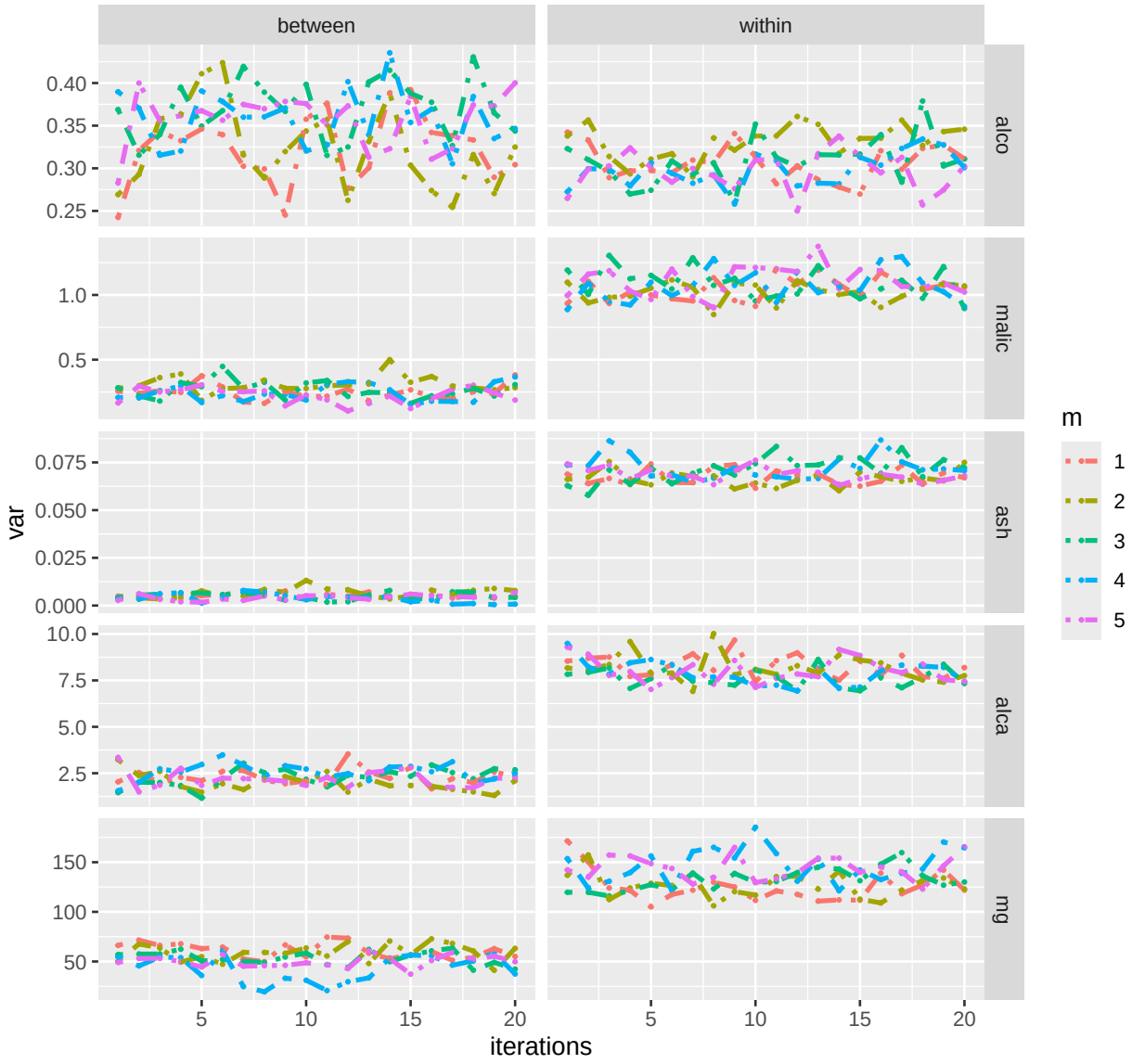
`FCS-hetero` allows imputation of continuous variables according to linear mixed models (various methods are available in the `micemd` R package Audigier and Resche-Rigon (2023)). Furthermore, contrary to FCS-homo, FCS-hetero updates the partition without assuming constant variance in each cluster.

2.2.1 Convergence

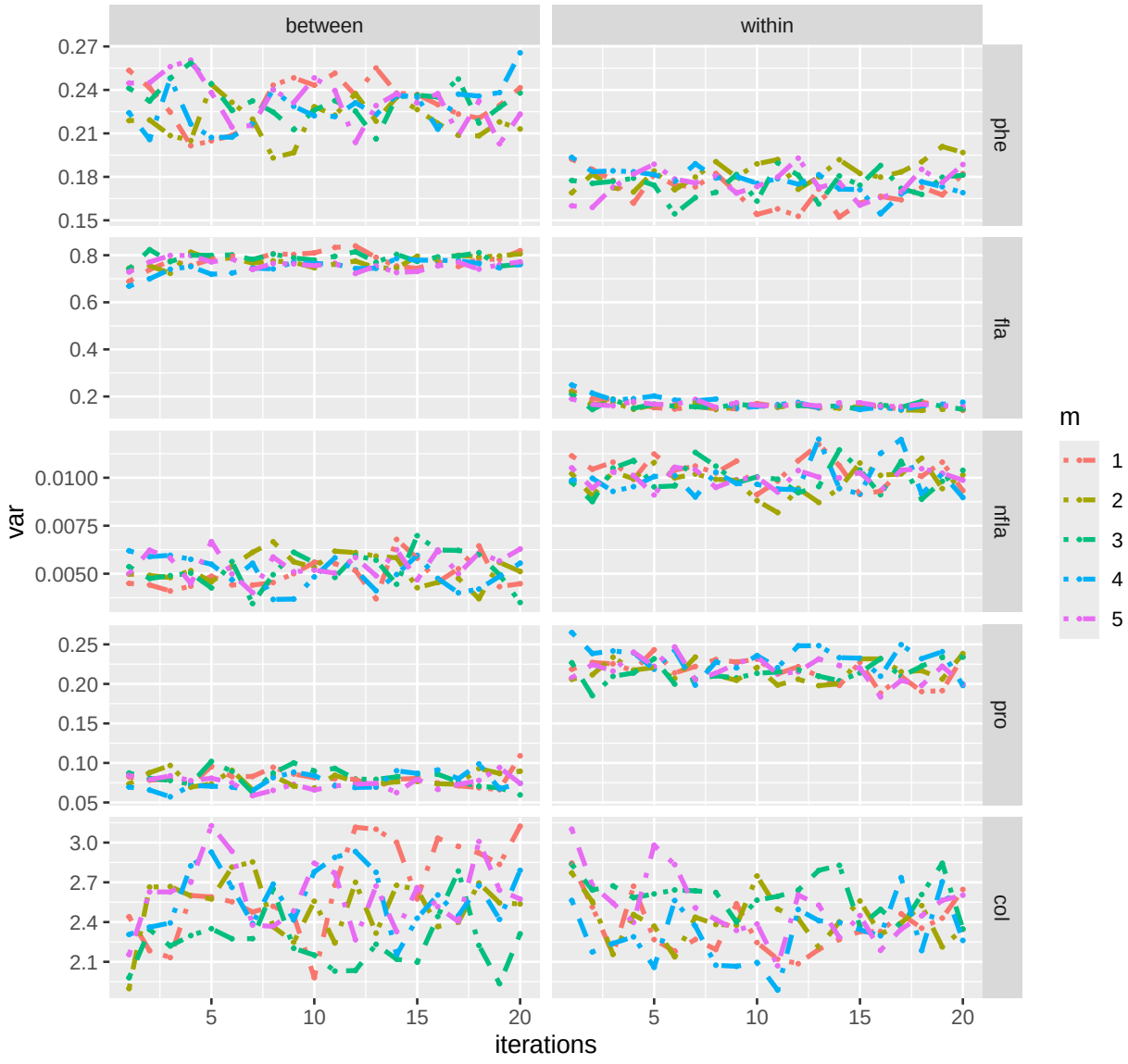
FCS imputation consists in imputing each variable sequentially several times. Many iterations can be required (`maxit` argument). For checking convergence, the within and between inertia of each imputed variable can be plotted at each iteration, as proposed by the `choosemaxit` function

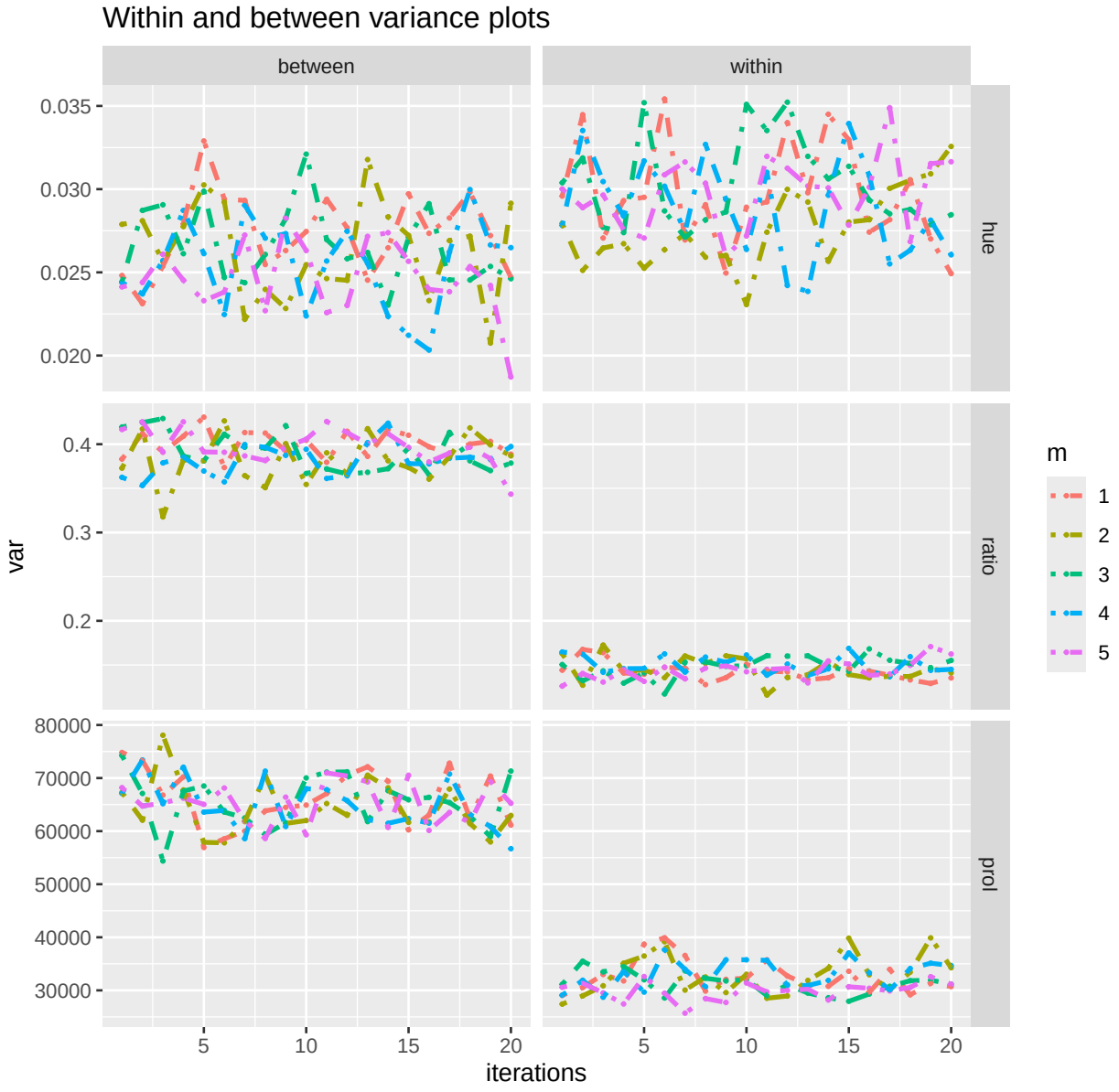
```
choosemaxit(res.imp.FCS)
```

Within and between variance plots



Within and between variance plots





Note that by default, only the five first imputed data sets are plotted (corresponding to the number of curves plotted for each variable). The `plotm` argument can be tuned to modify which curves should be drawn.

In this case, the number of iterations could be potentially increased. For achieving this goal, the imputation should be rerun by increasing the `maxit` argument as follows:

```
res.imp <- imputedata(data.na = wine.na,
                      method = "FCS-homo",
                      nb.clust = nb.clust,
                      maxit = 100,
                      m = m)
choosemaxit(res.imp)
```

For computational reasons, convergence diagnostic can be achieved by decreasing the number of imputed datasets `m`. When the number of iterations `maxit` will be chosen, then multiple imputation with a larger

value for `m` could be considered.

2.2.2 Specifying imputation models

Fully conditional imputation methods are quickly limited when the number of variables is large since imputation models become overfit. To address this issue, we can use penalised regression as proposed in `mice` by specifying `method.mice = lasso.norm` for instance. Another way consists in specifying conditional imputation models by tuning the `predictmat` argument. This argument is a binary matrix where each row indicates which explanatory variables (in column) should be used for imputation.

2.2.2.1 The `varselbest` procedure To tune this matrix in an automatic way, the `varselbest` function proposes to perform variable selection following Bar-Hen and Audigier (2022). Briefly, `varselbest` performs variable selection on random subsets of variables and then, combines them to recover which explanatory variables are related to the response. More precisely, the outlines of the algorithm are as follows: let consider a random subset of `sizeblock` among p variables. Then, any selection variable scheme can be applied (lasso, stepwise and knockoff are proposed by tuning the `method.select` argument). By resampling (`B` times) a sample of size `sizeblock` among the p variables, we may count how many times a variable is considered as significantly related to the response and how many times it is not. We need to define a threshold (`r`) to conclude if a given variable is significantly related to the response (by default, $\mathtt{r} = 0.3$). The main advantage of this function is that it handles both missing values and high-dimensional data.

By default, the `varselbest` function performs variable selection by knockoff (Barber and Candès (2015)) based on $\mathtt{B} = 200$ bootstrap subsets. The threshold `r` is tuned at 0.3 allowing to omit only variables very poorly predictive. The choices of `B` and `r` are discussed in next sections.

Since the method is time consuming, the function allows parallel computing by tuning the `nnodes` argument. In the next example, the imputation model for the variable `alco` is obtained using the algorithm previously described.

```
nnodes <- 2
# Number of CPU cores used for parallel computation.
# Use parallel::detectCores() to choose an appropriate number

# variable selection to impute the "alco" variable
B <- 50 # number of bootstrap subsets, should be increased in practice
res.varsel <- varselbest(res.imputedata = res.imp.FCS, B = B, listvar = "alco",
                         nnodes = nnodes, graph = FALSE)

res.varsel$predictormatrix["alco", ]
```

```
#>  alco malic   ash  alca    mg   phe   fla  nfla   pro   col   hue ratio  prol
#>     0     0     1     1     1     0     1     0     1     1     1     1     1
```

The function suggests considering the variables ash, alca, mg, fla, pro, col, hue, ratio, prol to impute the `alco` variable. Then, imputation can be rerun by specifying the `predictmat` argument returned by the `varselbest` function

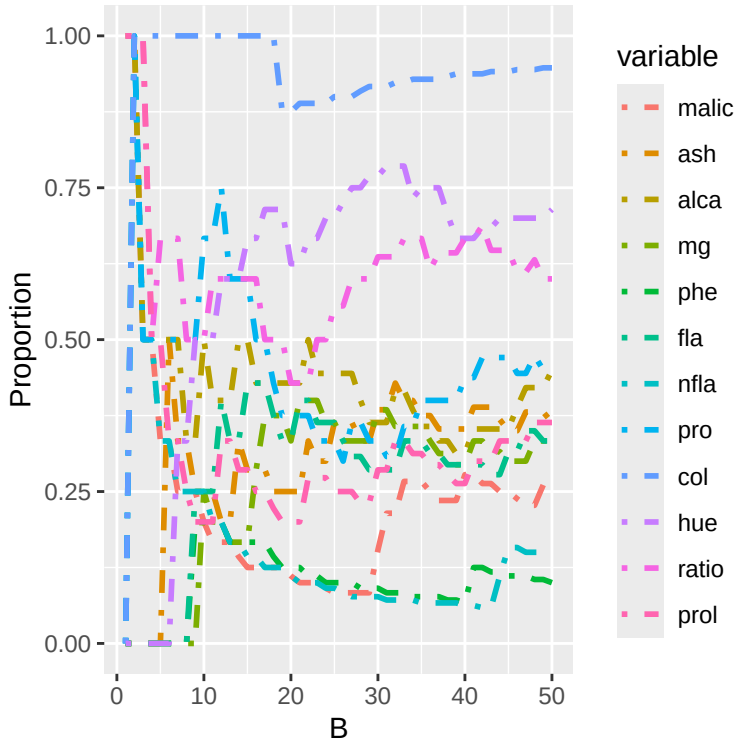
```
# multiple imputation with the new model
res.imp.select <- imputedata(data.na = wine.na,
                             method = "FCS-homo",
                             nb.clust = nb.clust,
                             maxit = maxit,
                             m = m,
                             predictmat = res.varsel$predictormatrix)
```

Note that for specifying all conditional imputation models you should use

```
varselbest(res.imputedata = res.imp.FCS, B = B, nnodes = nnodes) # (time consuming)
```

2.2.2.2 Convergence The number of iterations `B` should be large so that the proportion of times a variable is selected becomes stable. The `chooseB` function plots the proportion according to the number of iterations.

```
res.B <- chooseB(res.varsel)
```

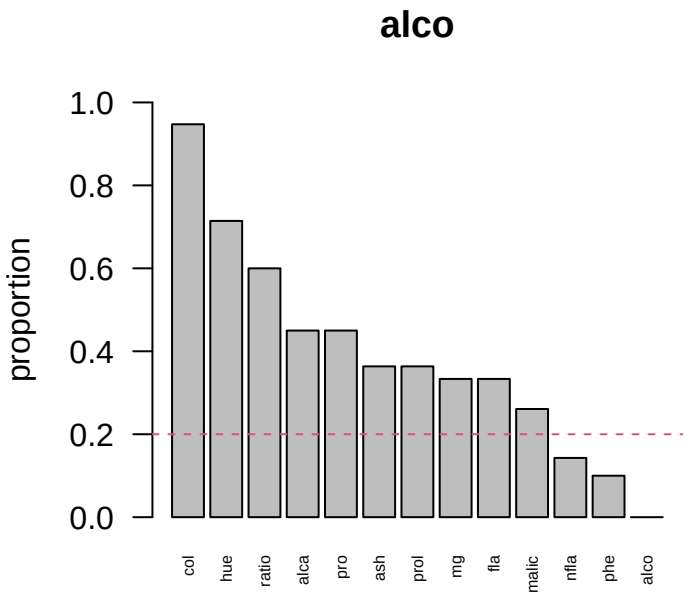


`B=50` iterations seems not enough.

2.2.2.3 Tuning the threshold r To tune the threshold `r`, the vector of proportions can be graphically investigated.

```
# check the variable importance
round(res.varsel$proportion["alco",], 2)
#>  alco malic   ash  alca    mg   phe   fla  nfla   pro   col   hue ratio  prol
#>  0.00  0.26  0.36  0.45  0.33  0.10  0.33  0.14  0.45  0.95  0.71  0.60  0.36
barplot(sort(res.varsel$proportion["alco",], decreasing=TRUE),
        ylab = "proportion",
        main = "alco",
        ylim = c(0, 1),
        las = 2,
        cex.names = .5)

r <- 0.2 # a new threshold value (r = 0.3 by default)
abline(h = r, col = 2, lty = 2)
```



Then, the predictor matrix can be easily updated at hand as follows

```
predictormatrix <- res.varsel$predictormatrix
predictormatrix[res.varsel$proportion>r] <- 1
predictormatrix[res.varsel$proportion<=r] <- 0
predictormatrix["alco", ]
#>  alco malic   ash  alca    mg   phe   fla  nfla   pro   col   hue ratio  prol
#>     0     1     1     1     1     0     1     0     1     1     1     1     1
```

By decreasing the threshold, more variables are used in the imputation model.

A more automatic way to tune `r` consists in using the function `chooser`. `chooser` computes an optimal threshold using K-fold cross-validation. The call to `chooser` is highly time consuming, but the optimal value of `r` can be follows during the process through graphical outputs. By this way, the user can stop the process early. This can be achieved as follows:

```
chooser(res.varsel = res.varsel)
```

3 Analysis and pooling

After multiple imputation, cluster analysis and partition pooling can be done through the `clusterMI` function (Audigier and Niang (2022)). Next, kmeans clustering is applied on the imputed data sets as an example.

3.1 K means clustering and other implemented methods

kmeans is the clustering method used by default.

```
# kmeans clustering
res.pool.kmeans <- clusterMI(res.imp.JM, nnodes = nnodes)
```

The `clusterMI` function returns a consensus partition (`part` object) as well as a instability measure

(`instability` object). The `instability` object gathers the instability of each contributory partition (`U`), their average (`Ubar`), the between instability (`B`) and the total instability (`T`).

```
part <- res.pool.kmeans$part
table(part) #compute cluster sizes
#> part
#>  1  2  3
#> 53 63 62
table(part, ref) #compare the partition with the reference partition
#>     ref
#> part  1  2  3
#>    1  0  5 48
#>    2  2 61  0
#>    3 57  5  0
res.pool.kmeans$instability # look at instabilitiy measures
#> $U
#>  [1] 0.0181 0.0095 0.0225 0.0182 0.0255 0.0191 0.0332 0.0271 0.0182 0.0194
#> [11] 0.0241 0.0170 0.0383 0.0180 0.0179 0.0230 0.0349 0.0310 0.0191 0.0280
#>
#> $Ubar
#> [1] 0.023
#>
#> $B
#> [1] 0.072
#>
#> $Tot
#> [1] 0.095
```

Among other clustering methods, `clusterMI` allows cluster analysis by k-medoids (`method.clustering = "pam"`), clustering large applications (`method.clustering = "clara"`), hierarchical clustering (`method.clustering = "hclust"`), fuzzy c-means (`method.clustering = "cmeans"`), or model-based method (`method.clustering = "mixture"`)

```
res.pool.all <- lapply(c("kmeans", "pam", "clara","hclust", "mixture", "cmeans"),
                   FUN = clusterMI,
                   nnodes = nnodes,
                   output = res.imp.JM)
```

3.2 Custom clustering methods

The user can also use custom clustering methods. For instance, to use reduced k-means, as implemented in the R package `clustrd`, analysis and pooling can be achieved as follows:

```
library(clustrd)
res.ana.rkm <- lapply(res.imp.JM$res.imp,
                      FUN = cluspca,
                      nclus = nb.clust,
                      ndim = 2,
                      method= "RKM")

# extract the set of partitions (as list)
res.ana.rkm <- lapply(res.ana.rkm, "[[", "cluster")

# pooling by NMF
res.pool.rkm <- fastnmf(res.ana.rkm, nb.clust = nb.clust)
part.rkm <- res.pool.rkm$best$clust# extract the best solution based on several initialisations
```

Note that in this case, the instability is not computed.

4 Diagnostics

4.1 Imputation model

To check if the imputation model correctly fit the data, a classical way is to perform overimputation (Blackwell, Honaker, and King (2015)), as proposed by the `overimpute` function. Overimputation consists in imputing observed values several times (100 or more) and to compare the observed values with their imputed values.

Overimputation is a time-consuming process. To limit the time required for achieving overimputation the user can:

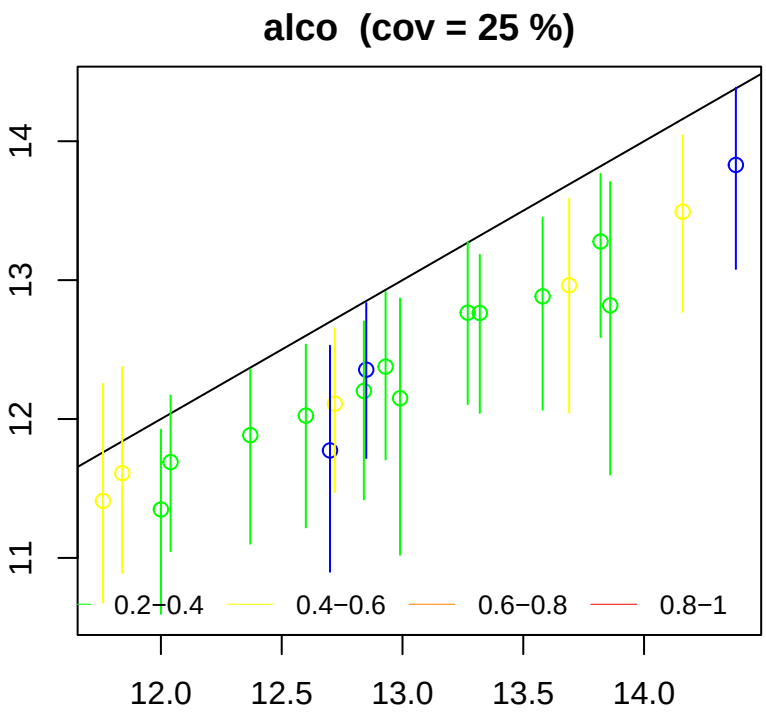
- use parallel computing by specifying the `nnodes` argument
- perform imputation for a subset of individuals by tuning the `plotinds` argument
- perform imputation for a subset of variables by tuning the `plotvars` argument

In the next example, we use parallel computing and perform overimputation on the first variable (`alco`) for 20 individuals (at random) only.

```
# Multiple imputation is rerun with more imputed data sets (m = 100)
res.imp.over <- imputedata(data.na = wine.na,
                      nb.clust = nb.clust,
                      m = 100,
                      Lstart = Lstart,
                      L = L,
                      verbose = FALSE)

# selection of 20 complete individuals on variable "alco"
plotinds <- sample(which(!is.na(wine.na[, "alco"])),
                   size = 20)

res.over <- overimpute(res.imp.over,
                       nnodes = nnodes,
                       plotvars = "alco",
                       plotinds = plotinds)
```



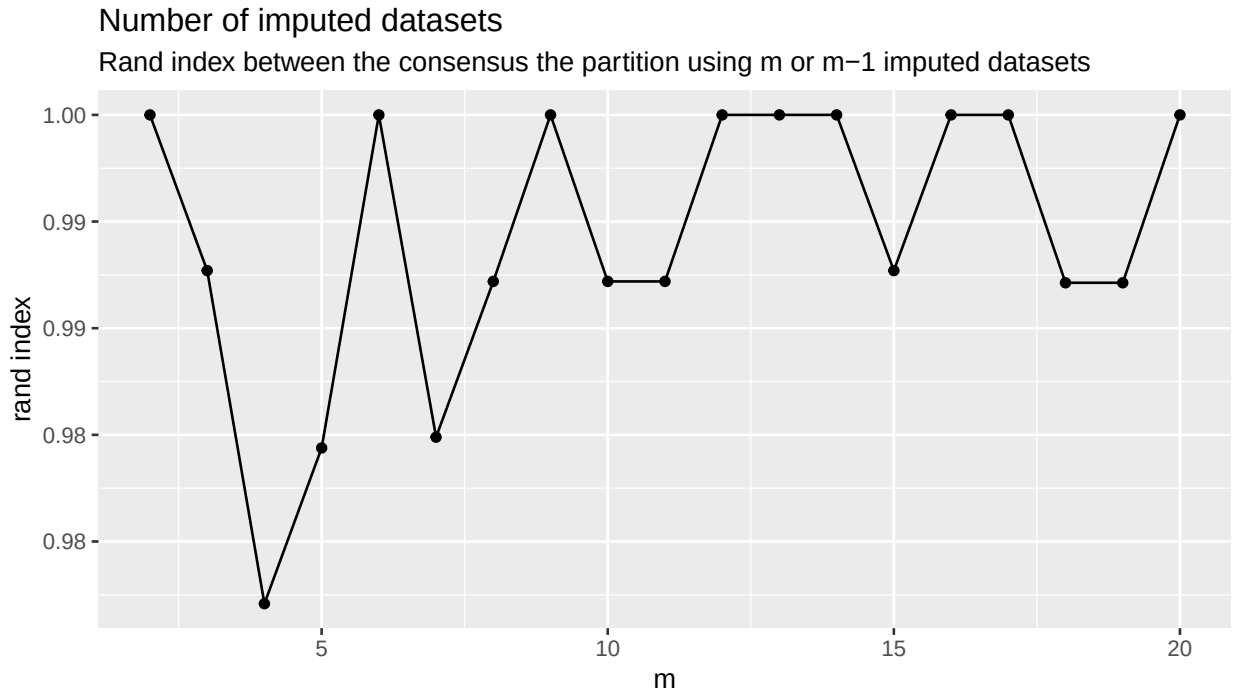
The graphic represents the observed values (x-axis) versus the 90% prediction interval (y-axis) for these values. Various colours are used according to the proportion of observed value used to build the interval (which depends on the missing data pattern on each individual). If the conditional model fits the data well, then 90% of intervals cutting the first bisector (indicating the observed values are gathered in the interval) are expected.

Here, the imputation model for the `alco` variable does not fit the observed data very well since the coverage is 25%. The fit could be improved by investigating FCS imputation methods.

4.2 Number of imputed data sets (`m`)

The number of imputed data sets (`m`) should be sufficiently large to improve the partition accuracy. The `choosem` function can be used to check if this number is suitable. This function computes the consensus partition by considering only the first imputed data sets. By this way, a sequence of `m` consensus partitions is obtained. Then, the rand index between successive partitions is computed and reported in a graph. The rand index measures proximity between partitions. If the rand index between the last consensus partitions of the sequence reaches its maximum values (1), then the number of imputed data sets does not modify the consensus partition and this number can be considered as sufficiently large.

```
res.m <- choosem(res.pool.kmeans)
```

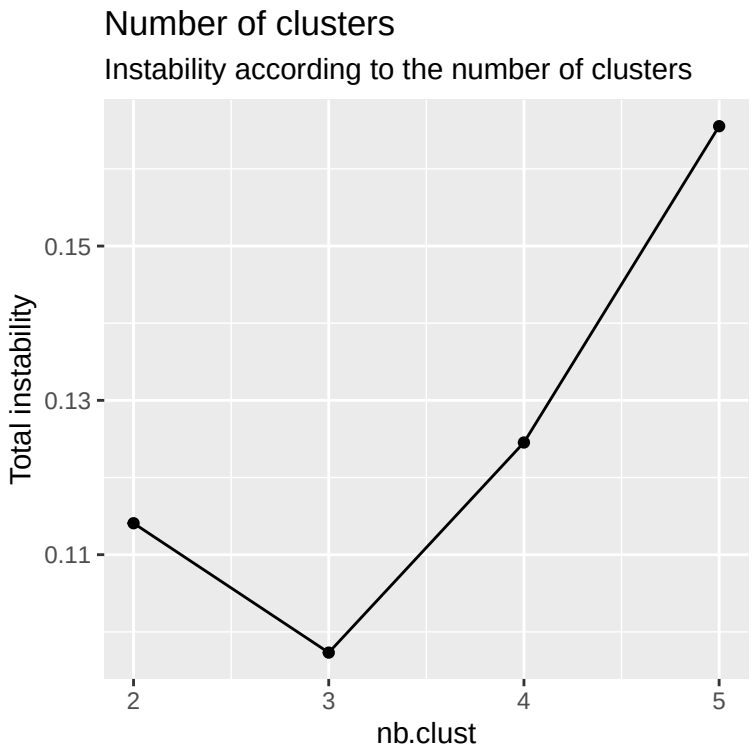


Here, the rand index is quickly close to 1, meaning the consensus partition is almost identical using `m`imputed data sets or `m-1`. The rand index is even equal to 1 at `m = 20`, meaning the consensus partition remains unchanged between `m = 19` and `m = 20`. Consequently, `m = 20` seems to be a suitable choice.

4.3 Number of clusters (`nb.clust`)

In practice, the number of clusters is generally unknown. A way to tune this number consists in inspecting the instability according to the number of clusters (Fang and Wang (2012)). The more stable partition could be retained. The `choosenbclust` function browses a grid of values for the number of clusters and for each one imputes the data and computes the instability.

```
res.nbclust <- choosenbclust(res.pool.kmeans)
```

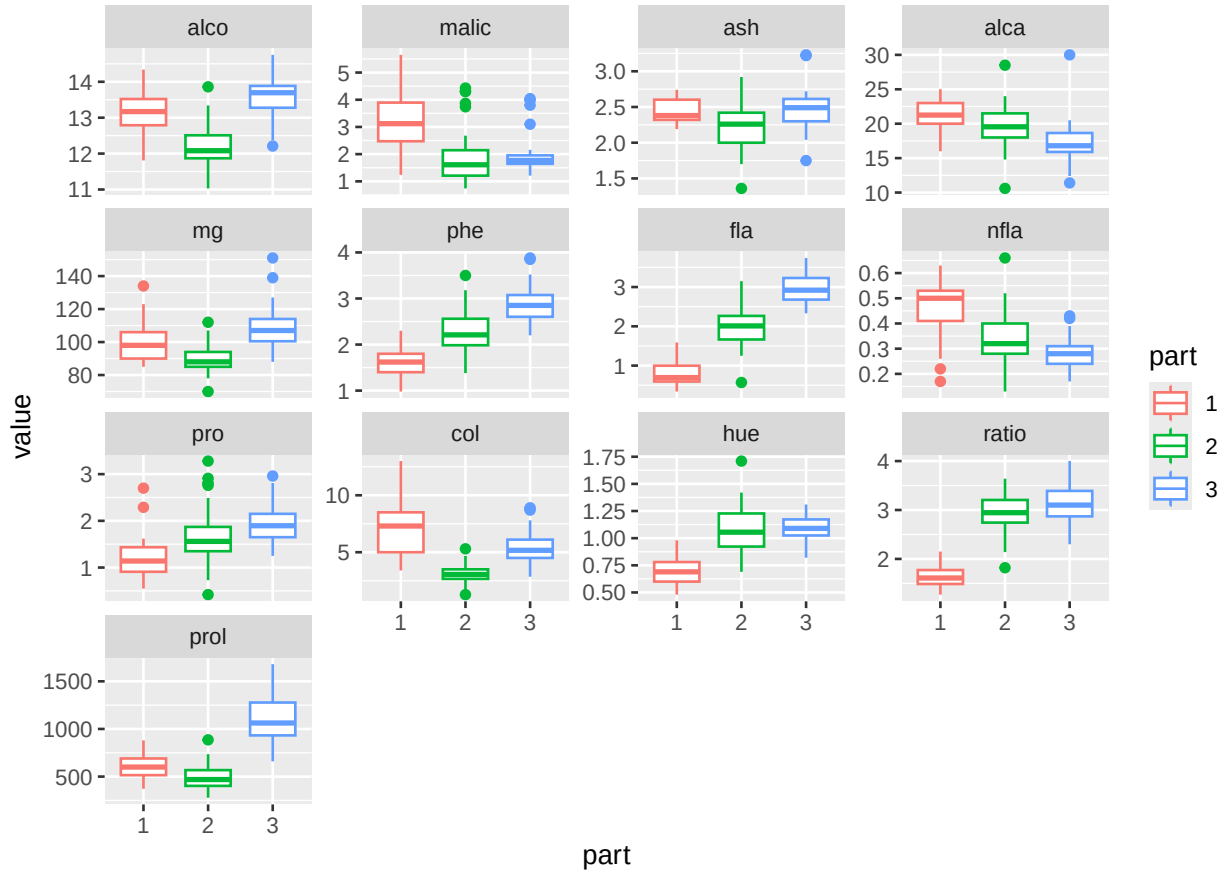


On the wine data set, the number of clusters suggested is clearly 3.

5 Cluster description

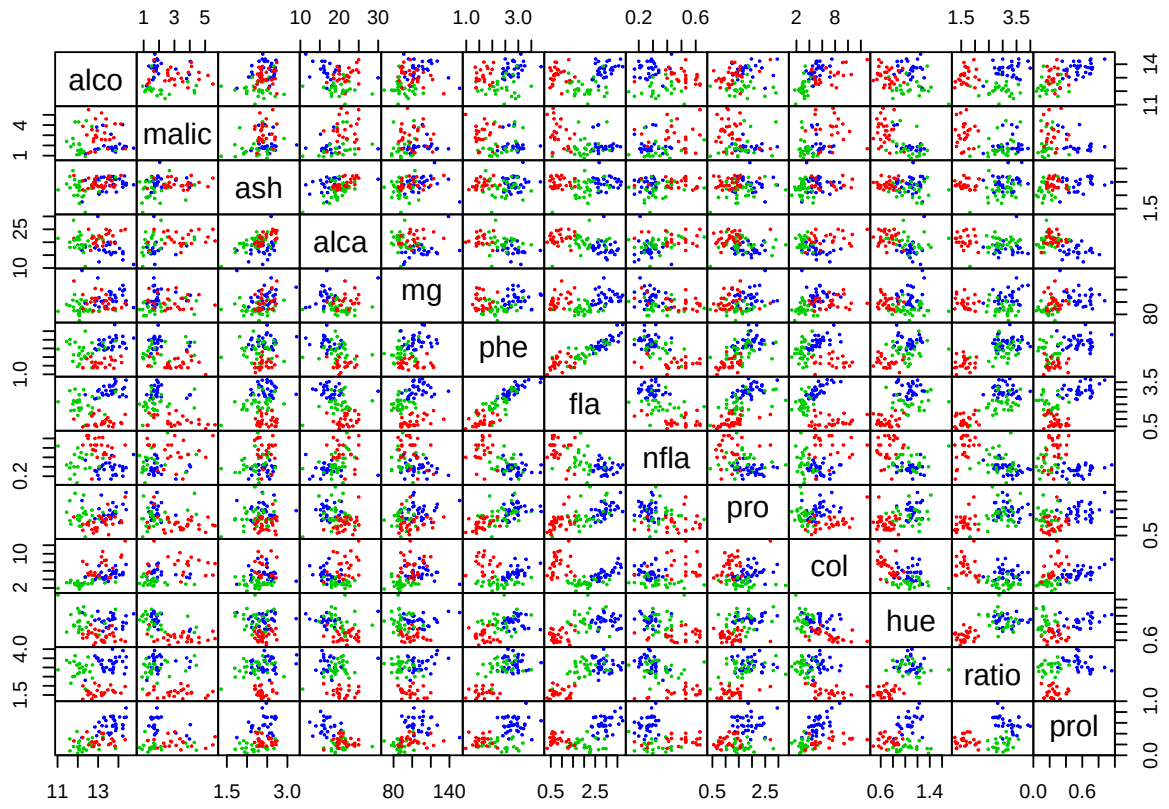
After building a partition with missing values, a description of each cluster can be performed variable per variable as follows:

```
require(reshape2)
require(ggplot2)
dat.m = melt(data.frame(wine.na, part = as.factor(part)), id.var=c("part"))
ggplot(dat.m, aes(part, value, col = part)) +
  facet_wrap(variable~., scales = "free_y") +
  geom_boxplot(width = 0.7)
```



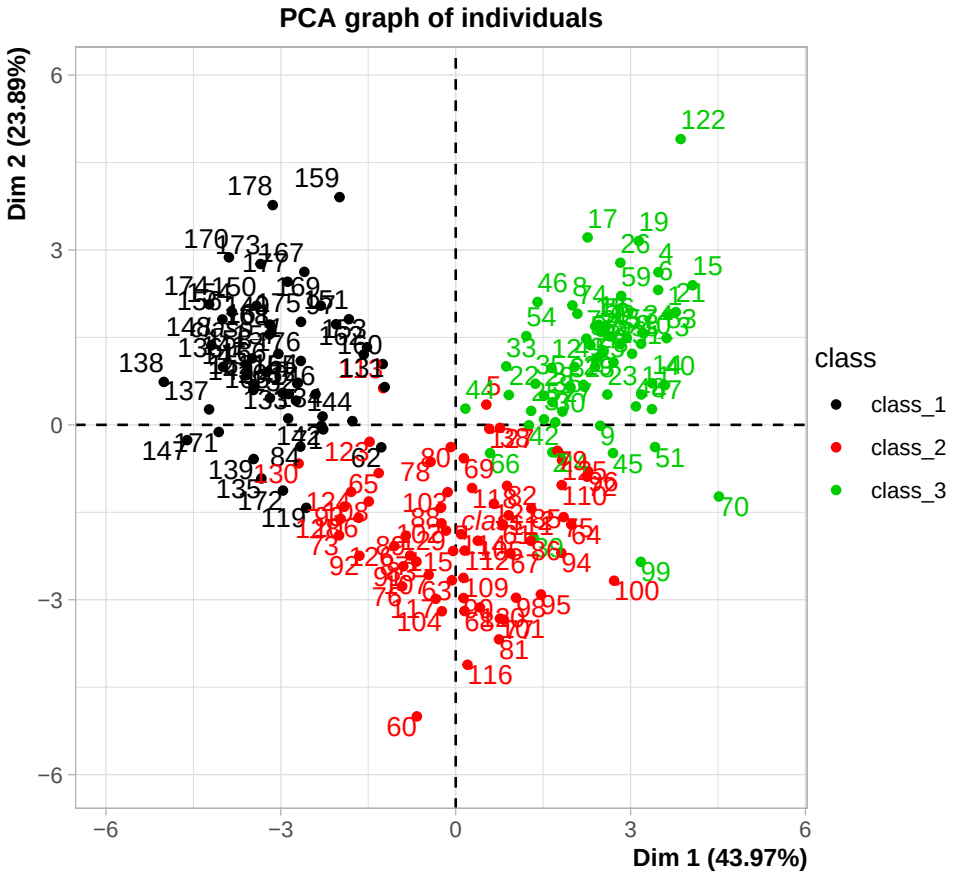
or by investigating the relationships for each pair of variables

```
library(VIM)
pairsVIM(wine.na,
         pch = 21,
         col = c("red", "green3", "blue")[part],
         cex = .2,
         gap = 0)
```



or from a multivariate point of view by using the principal component analysis, as proposed in the R packages `FactoMineR` (Lê, Josse, and Husson (2008)) and `missMDA` (Josse and Husson (2016))

```
library(FactoMineR)
library(missMDA)
# merge the partition variable with the incomplete data set
data.pca <- cbind.data.frame(class = factor(part, levels = seq(nb.clust)),
                             wine.na)
# perform PCA with missing values by specifying where is the partition variable
res.imputepca <- imputePCA(data.pca, quali.sup = 1)
res.pca <- PCA(res.imputepca$completeObs, quali.sup = 1, graph = FALSE)
plot(res.pca, habillage = 1)
```



Finally, the consensus partition can be analysed by computing external clustering comparison indices as proposed in the `clusterCrit` R package (Desgraupes (2023)) as follows:

```
library(clusterCrit)
res.crit <- extCriteria(part, ref, crit = "all")
round(unlist(res.crit), 2)
#> czekanowski_dice   folkes_mallows           hubert          jaccard
#>             0.87             0.87             0.80             0.77
#>       kulczynski          mcnemar              phi        precision
#>             0.87            -2.73             0.00             0.86
#>             rand           recall  rogers_tanimoto       russel_rao
#>             0.91             0.88             0.84             0.29
#>    sokal_sneath1    sokal_sneath2
#>             0.62             0.95
```

References

Asuncion, A., and D. J. Newman. 2007. "UCI Machine Learning Repository." University of California, Irvine, School of Information; Computer Sciences. http://archive.ics.uci.edu/ml.

Audigier, V., and N. Niang. 2022. "Clustering with missing data: which equivalent for Rubin's rules?" *Advances in Data Analysis and Classification*, September. https://doi.org/10.1007/s11634-022-00519-1.

Audigier, V., N. Niang, and M. Resche-Rigon. 2021. "Clustering with Missing Data: Which Imputation Model for Which Cluster Analysis Method?" https://arxiv.org/abs/2106.04424.

Audigier, V., and M. Resche-Rigon. 2023. *Micemd: Multiple Imputation by Chained Equations with Multilevel Data*. https://CRAN.R-project.org/package=micemd.

Barber, R. F., and E. J. Candès. 2015. "Controlling the false discovery rate via knockoffs." *The Annals of Statistics* 43 (5): 2055–85. https://doi.org/10.1214/15-AOS1337.

Bar-Hen, A., and V. Audigier. 2022. "An Ensemble Learning Method for Variable Selection: Application to High-Dimensional Data and Missing Values." *Journal of Statistical Computation and Simulation* 0 (0): 1–23. https://doi.org/10.1080/00949655.2022.2070621.

Blackwell, M., J. Honaker, and G. King. 2015. "A Unified Approach to Measurement Error and Missing Data: Overview and Applications." *Sociological Methods and Research*, 1–39.

Desgraupes, B. 2023. *clusterCrit: Clustering Indices*. https://CRAN.R-project.org/package=clusterCrit.

Fang, Y., and J. Wang. 2012. "Selection of the Number of Clusters via the Bootstrap Method." *Comput. Stat. Data Anal.* 56 (3): 468–77. https://doi.org/10.1016/j.csda.2011.09.003.

Joseph L. Schafer., Original by. 2022. *Mix: Estimation/Multiple Imputation for Mixed Categorical and Continuous Data*. https://CRAN.R-project.org/package=mix.

Josse, J., and F. Husson. 2016. "missMDA: A Package for Handling Missing Values in Multivariate Data Analysis." *Journal of Statistical Software* 70 (1): 1–31. https://doi.org/10.18637/jss.v070.i01.

Kim, H. J. 2020. *DPImputeCont*. https://github.com/hang-j-kim/DPImputeCont.

Lê, S., J. Josse, and F. Husson. 2008. "FactoMineR: A Package for Multivariate Analysis." *Journal of Statistical Software* 25 (1): 1–18. https://doi.org/10.18637/jss.v025.i01.

Murray, Jared S., and Jerome P. Reiter. 2016. "Multiple Imputation of Missing Categorical and Continuous Values via Bayesian Mixture Models with Local Dependence." *Journal of the American Statistical Association* 111 (516): 1466–79. https://doi.org/10.1080/01621459.2016.1174132.

van Buuren, S., and K. Groothuis-Oudshoorn. 2011. "*Mice*: Multivariate Imputation by Chained Equations in R." *Journal of Statistical Software* 45 (3): 1–67. https://www.jstatsoft.org/v45/i03/.

Wang, Q., D. Manrique-Vallier, J. P. Reiter, and J. Hu. 2022. *NPBayesImputeCat: Non-Parametric Bayesian Multiple Imputation for Categorical Data*. https://CRAN.R-project.org/package=NPBayesImputeCat.