Muliti-view sim

Stephen Coleman

2022-07-14

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
library(tagmReDraft)
library(mdiHelpR)
```

```
##
## Attaching package: 'mdiHelpR'
```

```
## The following object is masked from 'package:tagmReDraft':
##
##     findOrder
```

```
## The following object is masked from 'package:methods':
##
##     show
```

```
library(batchmix)
```

```
##
## Attaching package: 'batchmix'
```

```
## The following object is masked from 'package:mdiHelpR':
##
##     createSimilarityMat
```

```
## The following objects are masked from 'package:tagmReDraft':
##
##     calcAllocProb, gammaLogLikelihood, generateInitialLabels,
##     getLikelihood, invGammaLogLikelihood, invWishartLogLikelihood,
##     plotLikelihoods, processMCMCChain, processMCMCChains,
##     runMCMCChains, wishartLogLikelihood
```

```
library(magrittr)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.7     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
```

```
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x tidyr::extract()   masks magrittr::extract()
## x dplyr::filter()    masks stats::filter()
## x dplyr::lag()       masks stats::lag()
## x purrr::set_names() masks magrittr::set_names()
```

```
library(mcclust)
```

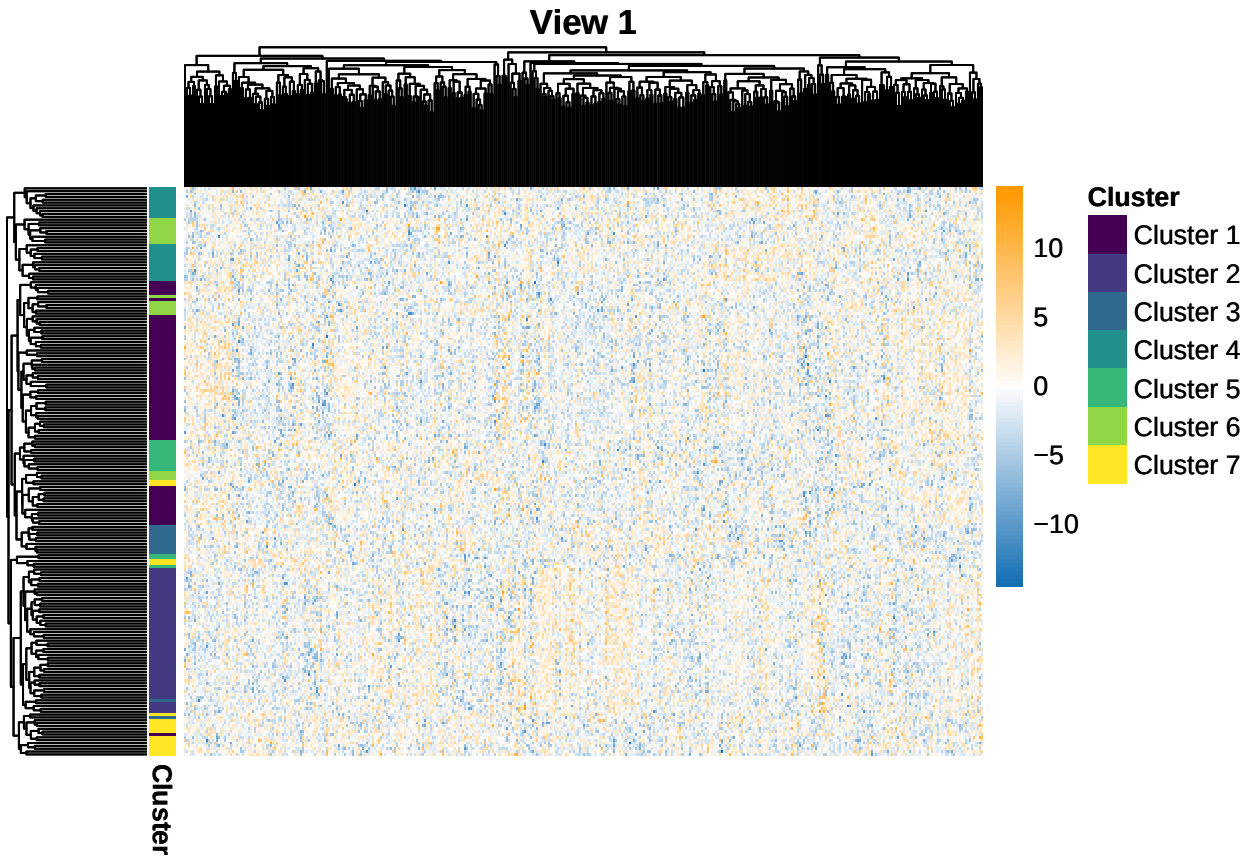
```
## Loading required package: lpSolve
```

```
RcppParallel::setThreadOptions()

setMyTheme()
set.seed(1)

makePhiDF <- function(mcmc) {

n_phis <- choose(mcmc$V, 2)
phi_names <- c()
for(v in seq(1, mcmc$V - 1)) {
  for(u in seq(v + 1, mcmc$V)) {
    .name <- paste0("Phi_", v, u)
    phi_names <- c(phi_names, .name)
  }
}

phi_df <- mcmc$phis |>
  data.frame() |>
  magrittr::set_colnames(phi_names) |>
  dplyr::mutate(Iteration = seq(mcmc$burn + mcmc$thin, mcmc$R, mcmc$thin)) |>
  tidyr::pivot_longer(-Iteration, values_to = "Sampled_value", names_to = "Parameter")

phi_df
}

generateGaussianView <- function(cluster_means, std_devs, N, P, labels,
                                 row_names = paste0("Person_", 1:n),
                                 col_names = paste0("Gene_", 1:p)) {
  gen_data <- matrix(0, N, P)

  for (ii in seq(1, P)) {
    reordered_cluster_means <- sample(cluster_means)
    reordered_std_devs <- sample(std_devs)

    # Draw n points from the K univariate Gaussians defined by the permuted means.
    for (jj in seq(1, N)) {

      .mu <- reordered_cluster_means[labels[jj]]
      .sd <- reordered_std_devs[labels[jj]]

      # print(labels[jj])
      # print(.mu)
      # print(.sd)

      gen_data[jj, ii] <- rnorm(1,
```

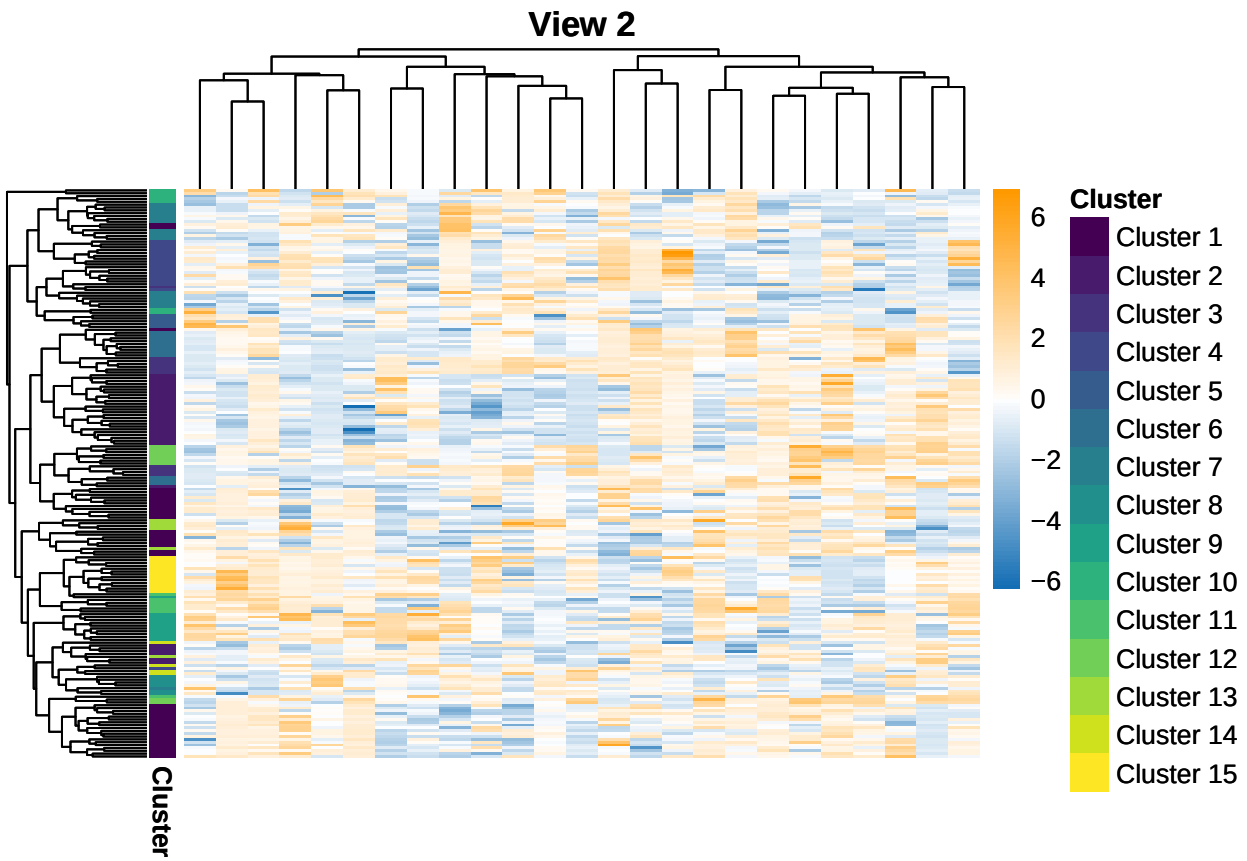
```
                                    mean = .mu,
                                    sd = .sd
      )
    }
  }
  gen_data
}

generateCategoricalView <- function(probs, N, P, labels,
                                    row_names = paste0("Person_", 1:n),
                                    col_names = paste0("Gene_", 1:p)) {
  gen_data <- matrix(0, N, P)

  for (ii in seq(1, P)) {
    reordered_probs <- sample(probs)

    # Draw n points from the K univariate Gaussians defined by the permuted means.
    for (jj in seq(1, N)) {

      .p <- reordered_probs[labels[jj]]

      gen_data[jj, ii] <- sample(c(0, 1), 1, prob = c(1 - .p, .p))
    }
  }
  gen_data
}

generateViewGivenStructure <- function(generating_structure, frac_present, P, K, class_weights, type, pa
  N <- length(generating_structure)
  labels_transferred <- sample(c(0, 1), N, replace = TRUE, prob = c(1 - frac_present, frac_present))
  labels_transferred_ind <- which(labels_transferred == 1)
  N_transferred <- sum(labels_transferred)
  N_new <- N - N_transferred
  identical_view <- N_new == N
  K_ind <- seq(1, K)
  labels <- rep(0, N)
  labels[labels_transferred_ind] <- generating_structure[labels_transferred_ind]
  if(! identical_view) {
    new_labels <- sample(K_ind, N - N_transferred, replace = TRUE, prob = class_weights)
    labels[-labels_transferred_ind] <- new_labels
  }
  gaussian <- type == "G"
  categorical <- type == "C"

  if(gaussian) {
    means <- params$means
    std_devs <- params$std_devs
    gen_data <- generateGaussianView(means, std_devs, N, P, labels)
  }
  if(categorical) {
    probs <- params$probs
    gen_data <- generateCategoricalView(probs, N, P, labels)
  }
```

```
  row.names(gen_data) <- names(generating_structure)

  list(
    gen_data = gen_data,
    labels = labels
  )
}

N <- 200
P <- 400
K <- 7
B <- 3

group_means <- rnorm(K, sd = 1)
group_sds <- rgamma(K, 4, 2)
group_dfs <- c(4, 10, 20, 50, 10, 20, 20)

batch_shift <- rnorm(B, sd = 0.5)
batch_scale <- 1 + rgamma(B, 10, 20)

group_weights <- c(0.3, 0.2, 0.1, rep(0.4 / 4, 4))
batch_weights <- c(0.4, 0.4, 0.2)

gen_data <- generateBatchData(N, P,
                              group_means = group_means,
                              group_std_devs = group_sds,
                              batch_shift = batch_shift,
                              batch_scale = batch_scale,
                              group_weights = group_weights,
                              batch_weights = batch_weights,
                              group_dfs = group_dfs)

names(gen_data$group_IDs) <- row.names(gen_data$observed_data)

frac_present_2 <- 0.5
P_2 <- 25
K_2 <- 15
class_weights_2 <- rgamma(K_2, 10)
class_weights_2 <- class_weights_2 / sum(class_weights_2)
params_2 <- list(means = rnorm(K_2, sd = 1),
                 std_devs = rgamma(K_2, 2, 2)
)

gen_view_2 <- generateViewGivenStructure(gen_data$group_IDs,
                                         frac_present = frac_present_2,
                                         P = P_2,
                                         K = K_2,
                                         type = "G",
                                         class_weights = class_weights_2,
                                         params = params_2)

frac_present_3 <- 0.4
P_3 <- 20
```

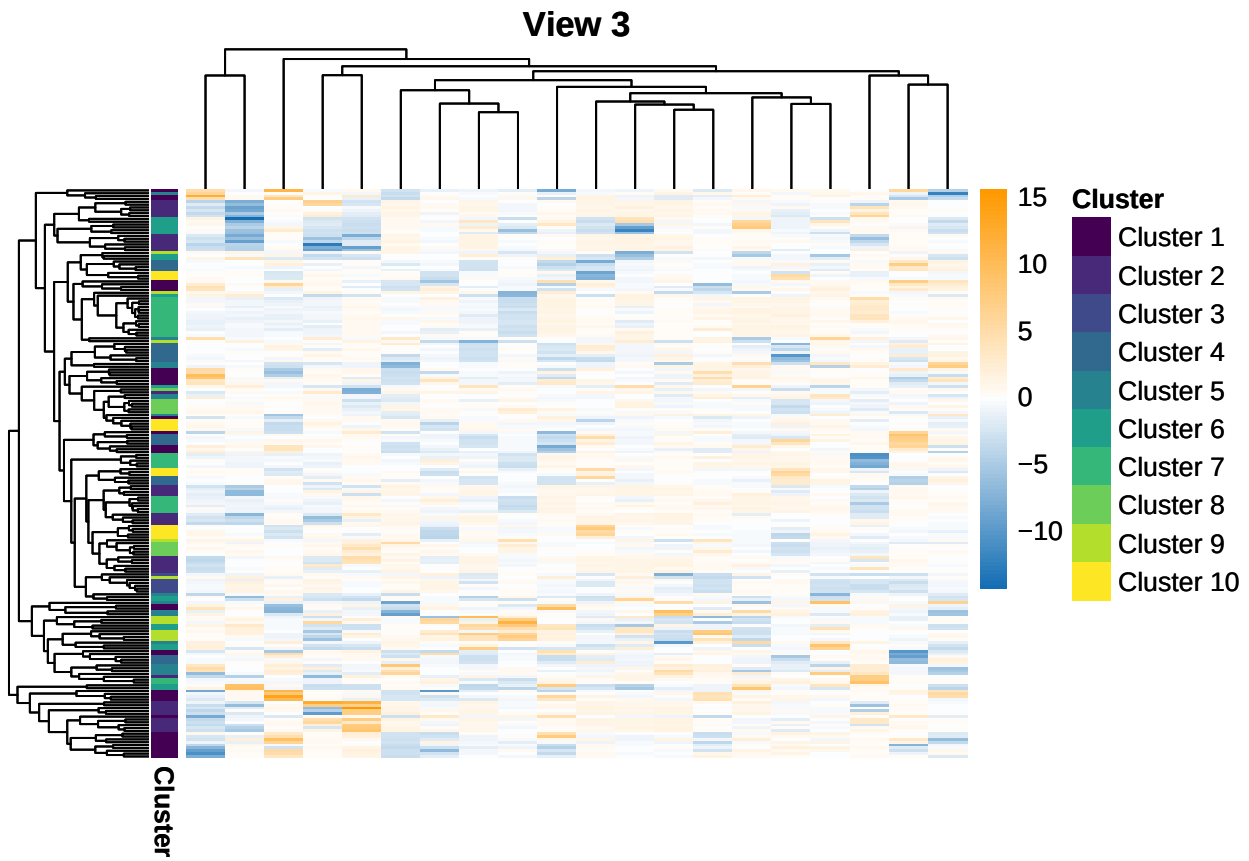
```r
K_3 <- 10
class_weights_3 <- rgamma(K_3, 10, 10)
class_weights_3 <- class_weights_3 / sum(class_weights_3)
# probs <- rbeta(K_3, 1, 1)
params_3 <- list(means = rnorm(K_3, sd = 1),
                 std_devs = rgamma(K_3, 2, 2)
)

gen_view_3 <- generateViewGivenStructure(gen_data$group_IDs,
                                         frac_present = frac_present_3,
                                         P = P_3,
                                         K = K_3,
                                         type = "G",
                                         class_weights = class_weights_3,
                                         params = params_3)


annotatedHeatmap(gen_data$observed_data, gen_data$group_IDs, show_rownames = FALSE, main = "View 1")
```




```r
# annotatedHeatmap(gen_data$corrected_data, gen_data$group_IDs, show_rownames = FALSE)

annotatedHeatmap(gen_view_2$gen_data, gen_view_2$labels, show_rownames = FALSE, main = "View 2")
```




```
annotatedHeatmap(gen_view_3$gen_data, gen_view_3$labels, show_rownames = FALSE, main = "View 3") #, col
```




```
arandi(gen_data$group_IDs, gen_view_3$labels)
```

```
## [1] 0.2302798
```

```
arandi(gen_data$group_IDs, gen_view_2$labels)
```

```
## [1] 0.3355122
```

```
arandi(gen_view_2$labels, gen_view_3$labels)
```

```
## [1] 0.03873106
```

```
R <- 10000
thin <- 50
types <- c("G", "G", "G")
K_max <- c(50, 50, 50)
fixed <- initial_labels <- matrix(0, nrow = N, 3)
n_chains <- 4

fixed[, 1] <- gen_data$fixed
initial_labels[ , 1] <- gen_data$group_IDs
X <- list(gen_data$observed_data,
          gen_view_2$gen_data,
          gen_view_3$gen_data
          )

mcmc_semi <- tagmReDraft::runMCMCChains(X, n_chains,
                      R = R,
```

```
                     thin = thin,
                     types = types,
                     K = K_max,
                     fixed = fixed,
                     initial_labels = initial_labels
)
```

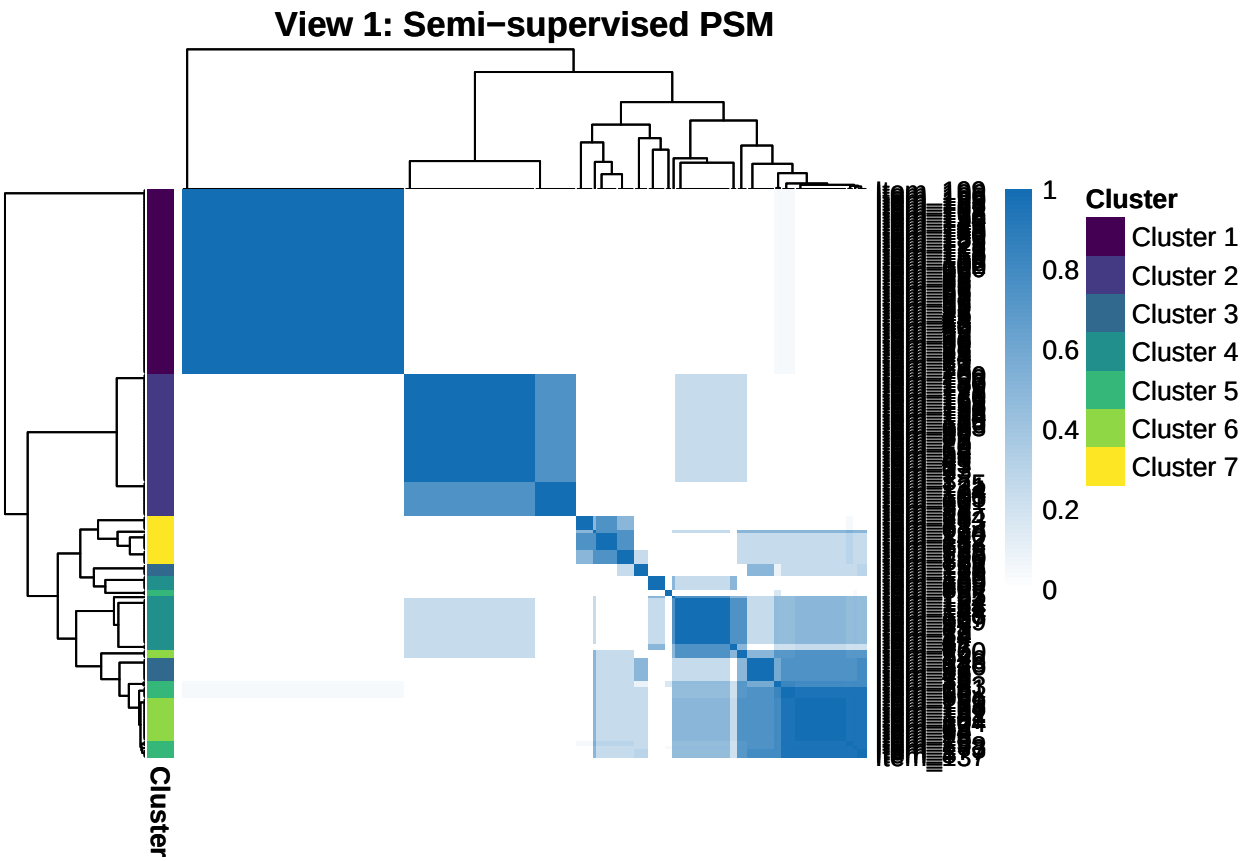
```
##
## Phi map:
##         0        0        1
##         0        0        2
##         1        2        0
##
## Phi map:
##         0        0        1
##         0        0        2
##         1        2        0
##
## Phi map:
##         0        0        1
##         0        0        2
##         1        2        0
##
## Phi map:
##         0        0        1
##         0        0        2
##         1        2        0
```

```
mcmc_un <- tagmReDraft::runMCMCChains(X, n_chains,
                     R = R,
                     thin = thin,
                     types = types,
                     K = rep(50, 3),
                     fixed = matrix(0, N, 3),
                     initial_labels = initial_labels
)
```

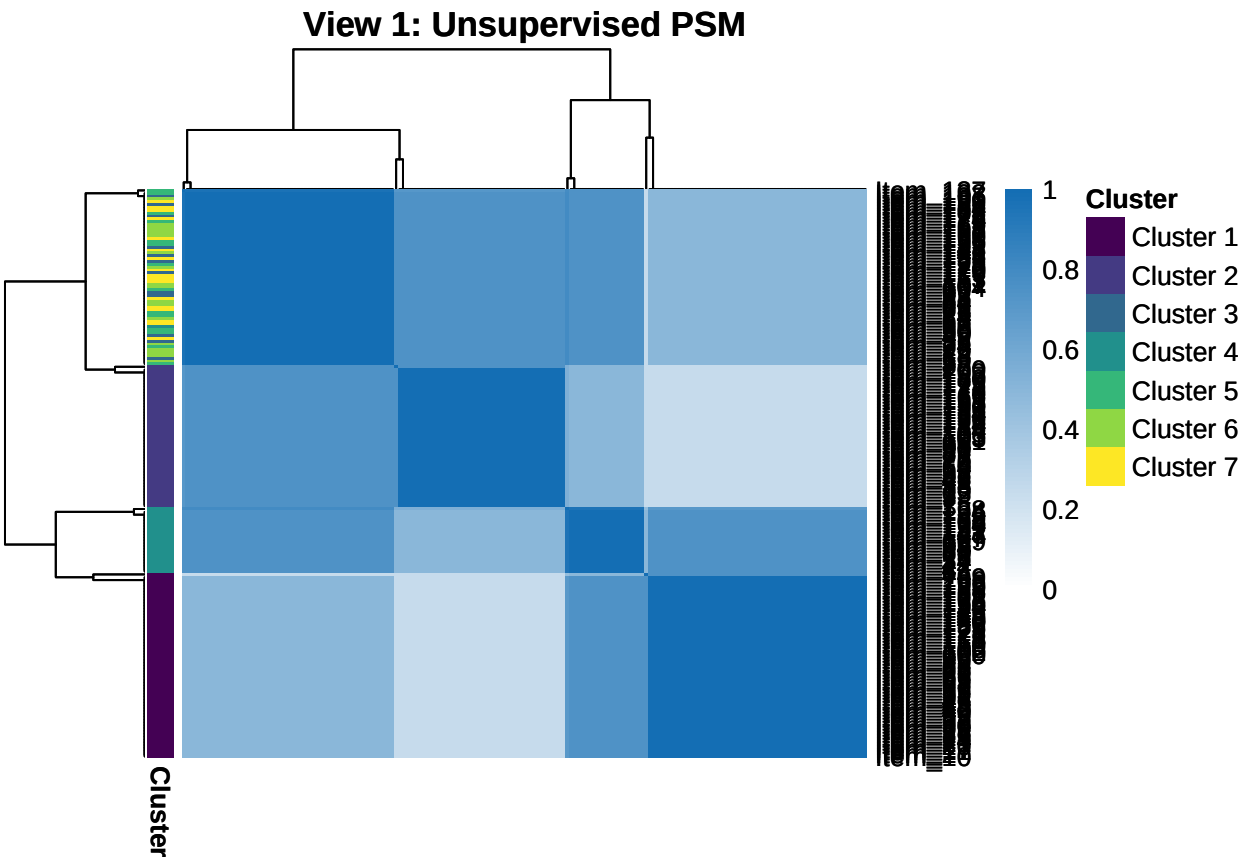
```
##
## Phi map:
##         0        0        1
##         0        0        2
##         1        2        0
##
## Phi map:
##         0        0        1
##         0        0        2
##         1        2        0
##
## Phi map:
##         0        0        1
##         0        0        2
##         1        2        0
##
## Phi map:
##         0        0        1
```

```
##         0        0        2
##         1        2        0
```

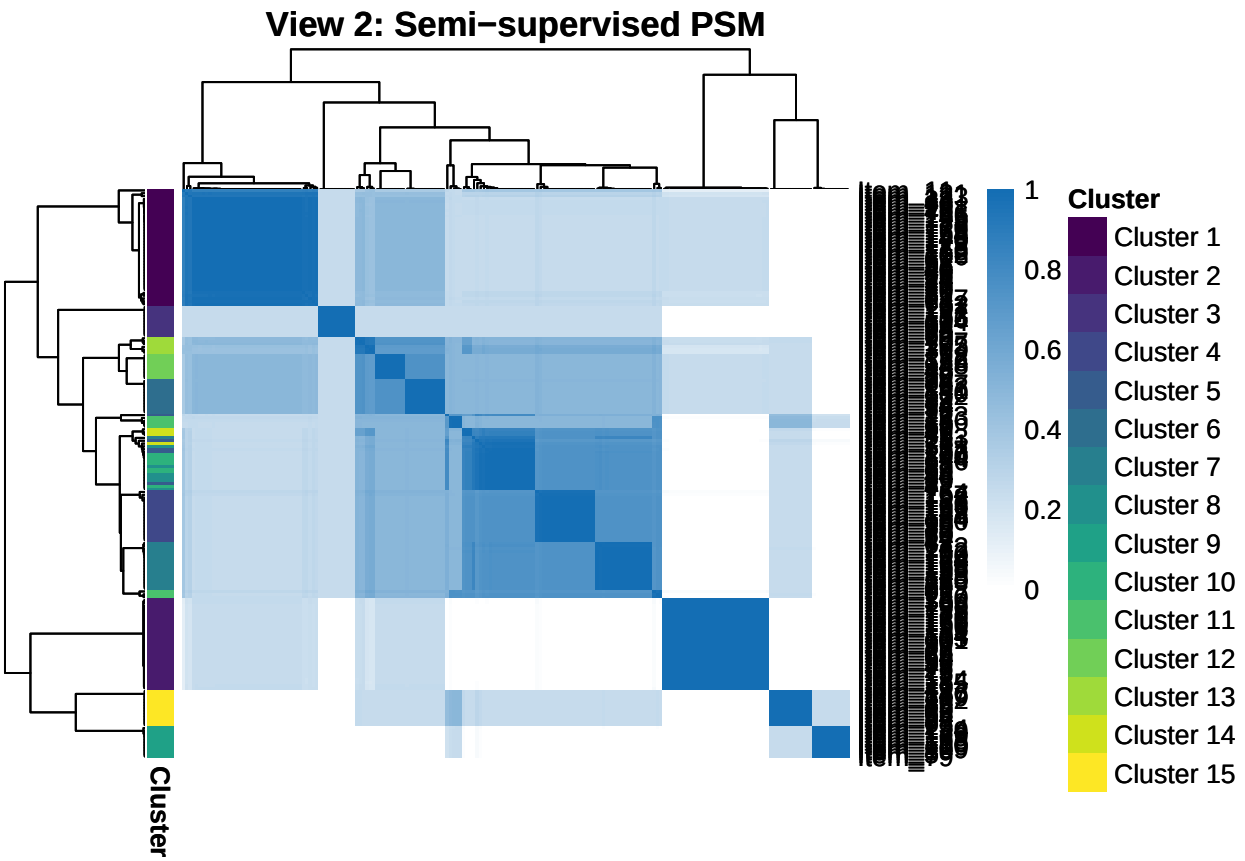
```
burn <- R * 0.2
mcmc_semi <- tagmReDraft::predictFromMultipleChains(mcmc_semi, burn)
mcmc_un <- tagmReDraft::predictFromMultipleChains(mcmc_un, burn)

# print(str(mcmc_semi))

psm_semi_1 <- mcmc_semi$allocations[[1]] |> createSimilarityMat()
row.names(psm_semi_1) <- row.names(gen_data$observed_data)

psm_semi_2 <- mcmc_semi$allocations[[2]] |> createSimilarityMat()
row.names(psm_semi_2) <- row.names(gen_data$observed_data)

psm_semi_3 <- mcmc_semi$allocations[[3]] |> createSimilarityMat()
row.names(psm_semi_3) <- row.names(gen_data$observed_data)

psm_un_1 <- mcmc_un$allocations[[1]] |> createSimilarityMat()
row.names(psm_un_1) <- row.names(gen_data$observed_data)

psm_un_2 <- mcmc_un$allocations[[2]] |> createSimilarityMat()
row.names(psm_un_2) <- row.names(gen_data$observed_data)

psm_un_3 <- mcmc_un$allocations[[3]] |> createSimilarityMat()
row.names(psm_un_3) <- row.names(gen_data$observed_data)

annotatedHeatmap(psm_semi_1, gen_data$group_IDs, col_pal = simColPal(), my_breaks = defineBreaks(simCol
```



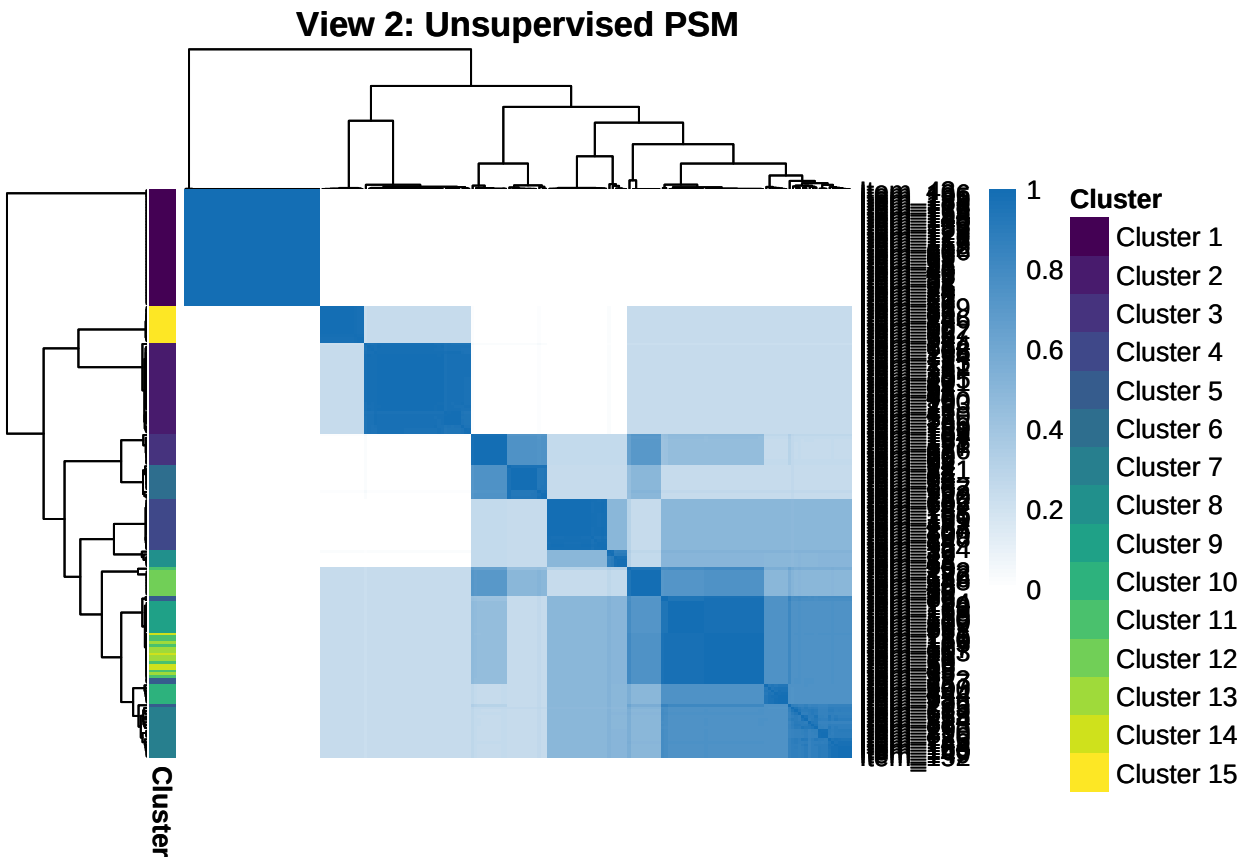

```
annotatedHeatmap(psm_un_1, gen_data$group_IDs, col_pal = simColPal(), my_breaks = defineBreaks(simColPal
```



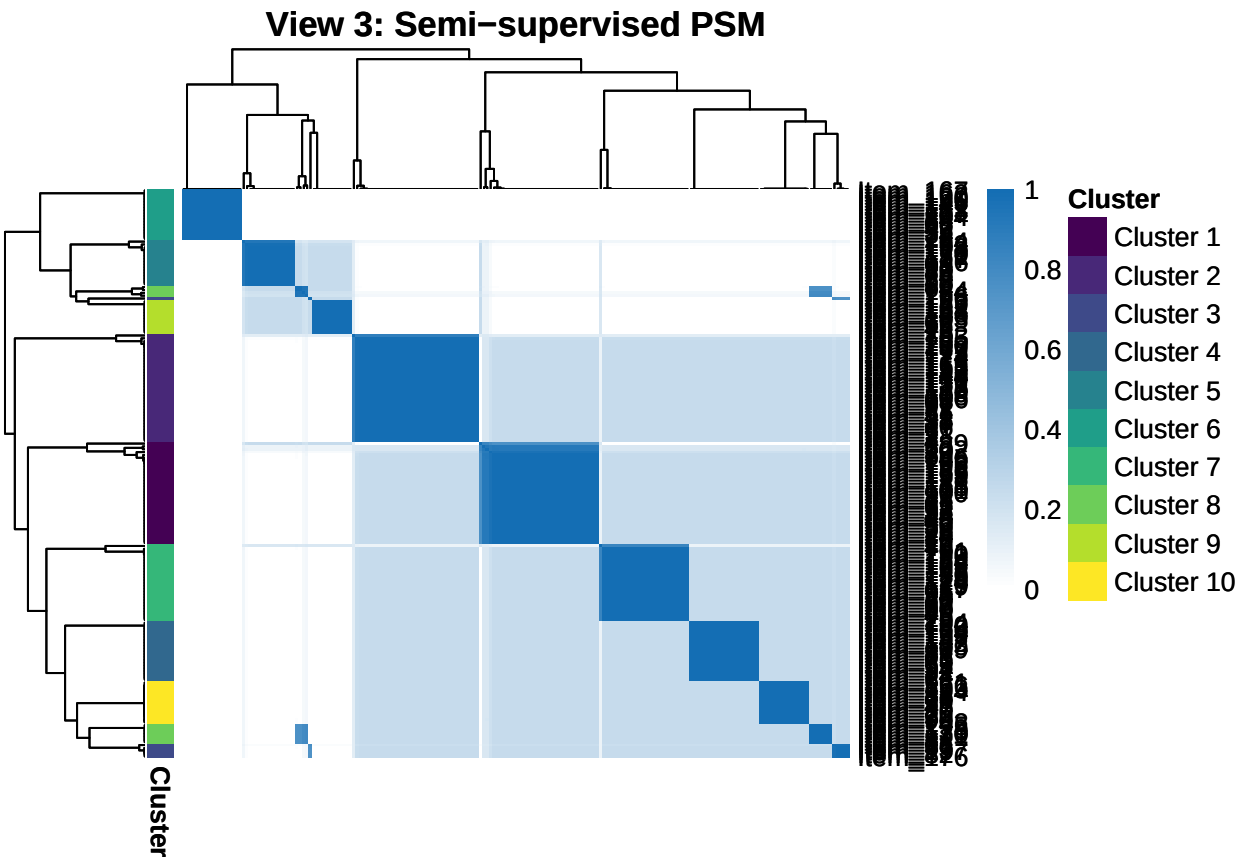

```
annotatedHeatmap(psm_semi_2, gen_view_2$labels, col_pal = simColPal(), my_breaks = defineBreaks(simColPa
```



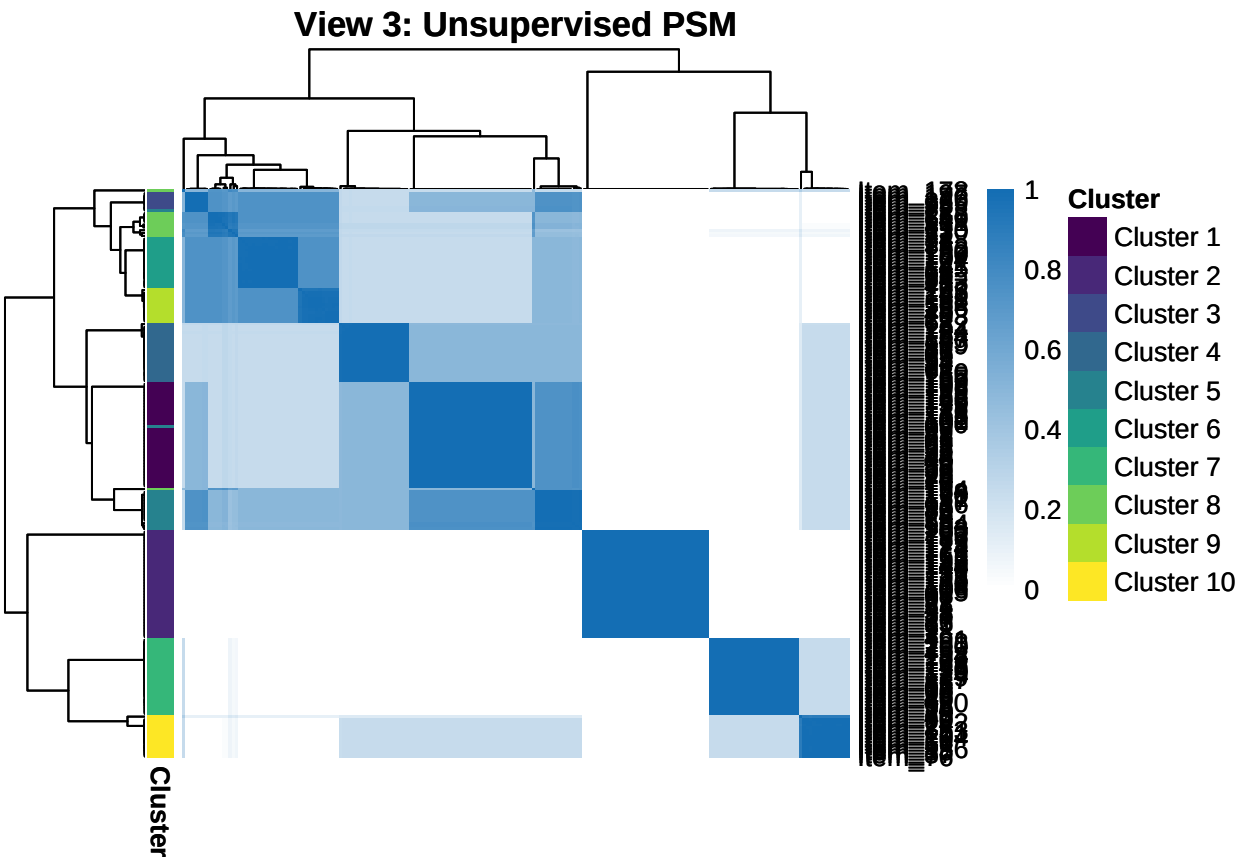

```
annotatedHeatmap(psm_un_2, gen_view_2$labels, col_pal = simColPal(), my_breaks = defineBreaks(simColPal
```



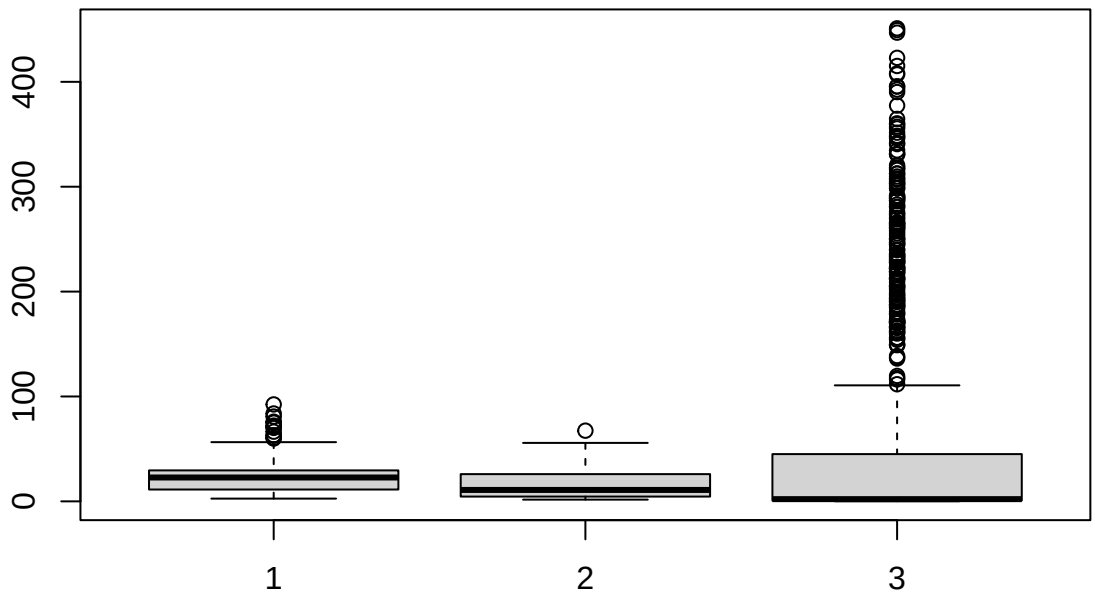

```
annotatedHeatmap(psm_semi_3, gen_view_3$labels, col_pal = simColPal(), my_breaks = defineBreaks(simColPa
```



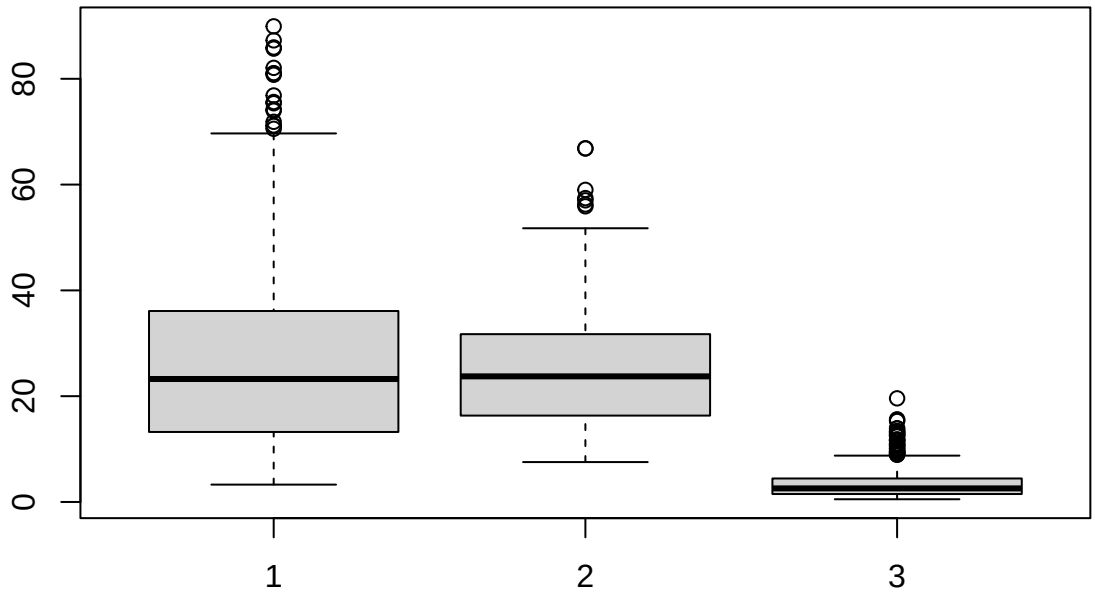

```
annotatedHeatmap(psm_un_3, gen_view_3$labels, col_pal = simColPal(), my_breaks = defineBreaks(simColPal
```




```
mcmc_semi$phis |> boxplot()
```

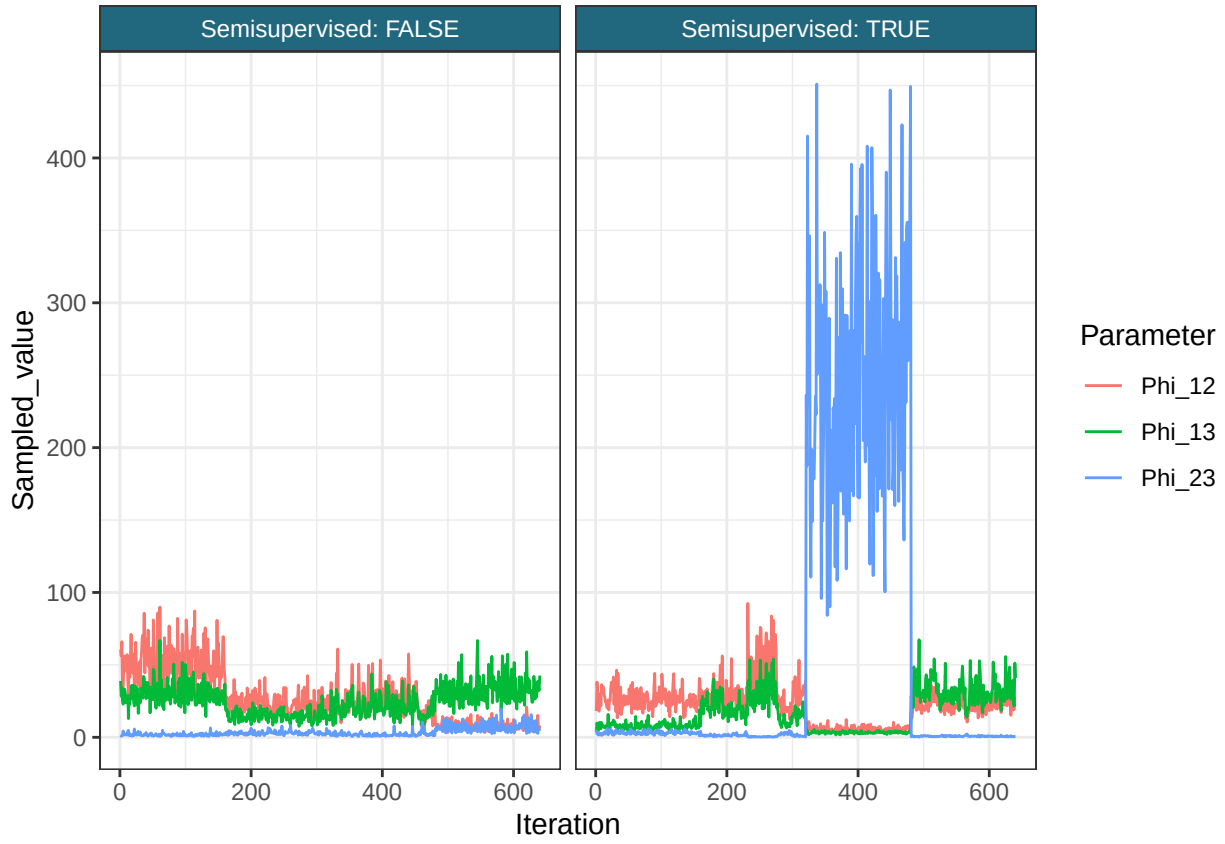



```
mcmc_un$phis |> boxplot()
```



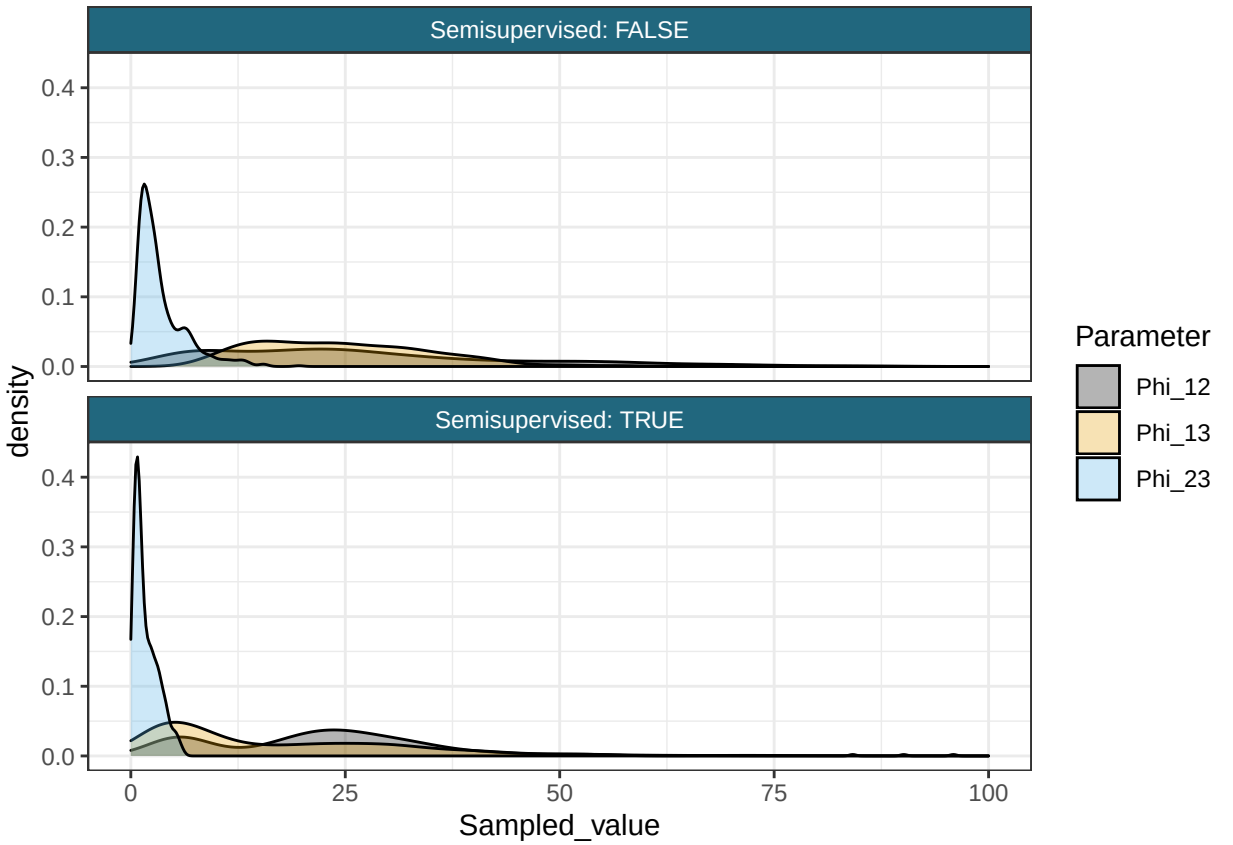

```
n_phis <- choose(3, 2)
phi_names <- c()
for(v in seq(1, 3 - 1)) {
  for(u in seq(v + 1, 3)) {
    .name <- paste0("Phi_", v, u)
    phi_names <- c(phi_names, .name)
  }
}

phi_semi_df <- mcmc_semi$phis |>
  data.frame() |>
  magrittr::set_colnames(phi_names) |>
  dplyr::mutate(Iteration = seq(1, nrow(mcmc_semi$phis))) |>
  tidyr::pivot_longer(-Iteration, values_to = "Sampled_value", names_to = "Parameter")
phi_un_df <- mcmc_un$phis |>
  data.frame() |>
  magrittr::set_colnames(phi_names) |>
  dplyr::mutate(Iteration = seq(1, nrow(mcmc_un$phis))) |>
  tidyr::pivot_longer(-Iteration, values_to = "Sampled_value", names_to = "Parameter")

# phi_semi_df <- makePhiDF(mcmc_semi)
# phi_un_df <- makePhiDF(mcmc_un)
#
phi_semi_df$Semisupervised <- TRUE
phi_un_df$Semisupervised <- FALSE
```

```
phi_df <- rbind(phi_semi_df, phi_un_df)

phi_df |>
  ggplot2::ggplot(ggplot2::aes(x = Iteration, y = Sampled_value, colour = Parameter)) +
  ggplot2::geom_line() +
  facet_wrap(~Semisupervised, labeller = label_both)
```

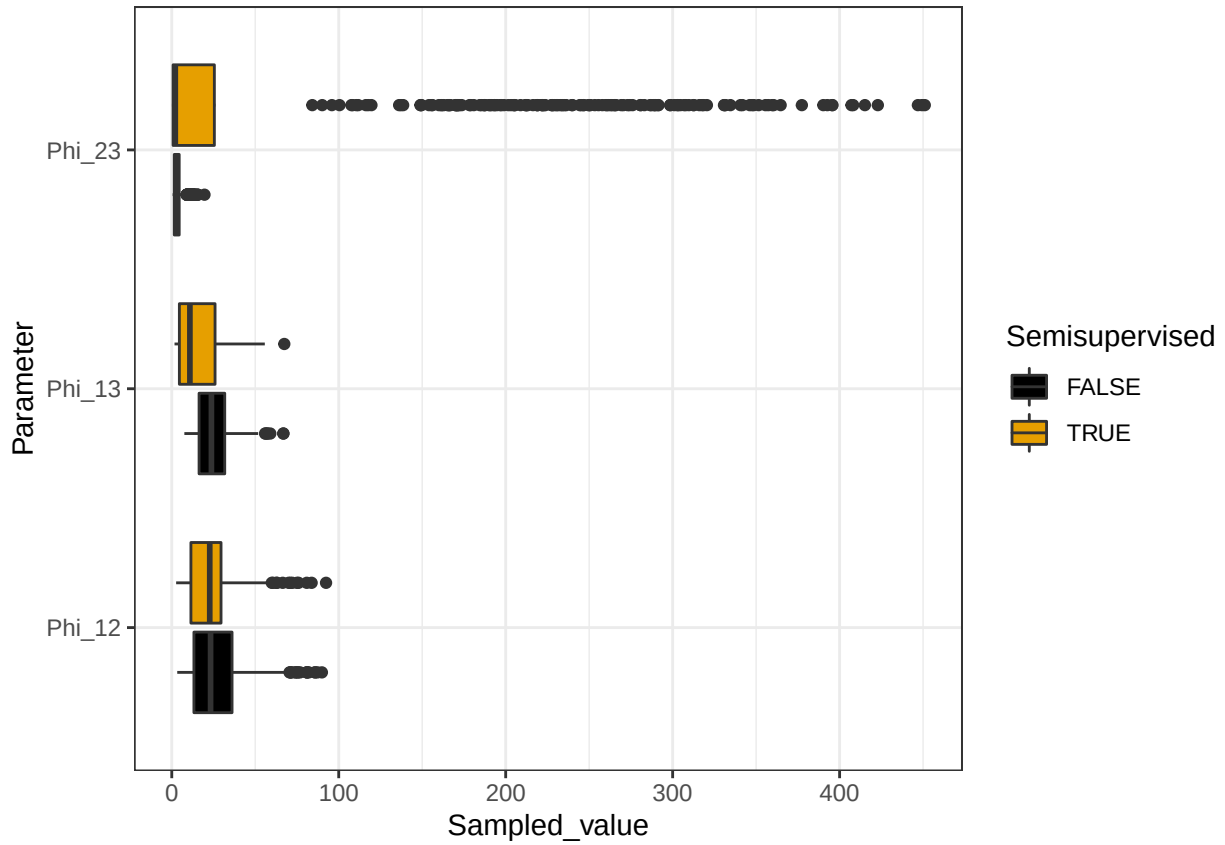



```
phi_df |>
  # dplyr::filter(Sampled_value < 100) |>
  ggplot2::ggplot(ggplot2::aes(x = Sampled_value, fill = Parameter)) +
  ggplot2::geom_density(alpha = 0.3) +
  ggthemes::scale_fill_colorblind() +
  xlim(c(0, 100)) +
  facet_wrap(~Semisupervised, ncol = 1, labeller = label_both)
```

```
## Warning: Removed 157 rows containing non-finite values (stat_density).
```




```
phi_df |>
  # dplyr::filter(Sampled_value < 40) |>
  ggplot2::ggplot(ggplot2::aes(x = Sampled_value, y = Parameter, fill = Semisupervised)) +
  ggplot2::geom_boxplot() +
  ggthemes::scale_fill_colorblind()
```




```
  # facet_wrap(~Semisupervised, ncol = 1)

pred_1_un <- maxpear(psm_un_1)$cl

pred_2_semi <- maxpear(psm_semi_2)$cl
pred_2_un <- maxpear(psm_un_2)$cl

pred_3_semi <- maxpear(psm_semi_3)$cl
pred_3_un <- maxpear(psm_un_3)$cl

arandi(gen_data$group_IDs, mcmc_semi$pred[[1]])
```

```
## [1] 0.8573806
```

```
arandi(gen_data$group_IDs, pred_1_un)
```

```
## [1] 0.3334867
```

```
arandi(gen_view_2$labels, pred_2_semi)
```

```
## [1] 0.4363212
```

```
arandi(gen_view_2$labels, pred_2_un)
```

```
## [1] 0.4319828
```

```
arandi(gen_view_3$labels, pred_3_semi)
```

```
## [1] 1
```

```
arandi(gen_view_3$labels, pred_3_un)
```

```
## [1] 0.5883826
```