Label-free mass-spectrometry analysis for CLL

Zahra Abedi

2024-02-07

```
# Loading/installing packages
packages <- read.table("packages.txt", header = F)
suppressMessages({
  if (!requireNamespace("BiocManager", quietly = TRUE)) {
    install.packages("BiocManager")
  }
  for (package in packages$V1) {
    if (!requireNamespace(package, quietly = TRUE)) {
      BiocManager::install(package)
      require(package, character.only = TRUE)
    } else {
      require(package, character.only = TRUE)
    }
  }
})

set.seed(1234)
# Files preparation
pro_ab <- data.frame(read_tsv("Protein_abundance.tsv"))
```

```
## Rows: 3942 Columns: 50
## -- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr  (1): X1
## dbl (49): A_1_1, A_1_10, A_1_11, A_1_12, A_1_13, A_1_14, A_1_15, A_1_16, A_1...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
rownames(pro_ab)  <- pro_ab[,1]
pro_ab <- pro_ab[,-1]
anno <- read_excel("sampleAnnotation.xls")
numeric_part <- as.integer(sub("A_1_", "",
                               colnames(pro_ab)))
pro_ab <- pro_ab[, order(numeric_part)]
samples <- which(anno$`sample ID` %in% colnames(pro_ab))
anno <- anno[samples,]
anno$time <- as.numeric(as.Date(anno$`last known alive`) -
                          as.Date(anno$`date of diagnosis`))
identical(anno$`sample ID`, colnames(pro_ab))
```

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

```r
# The number of missing values
sum(is.na(pro_ab))
```

```
## [1] 32948
```

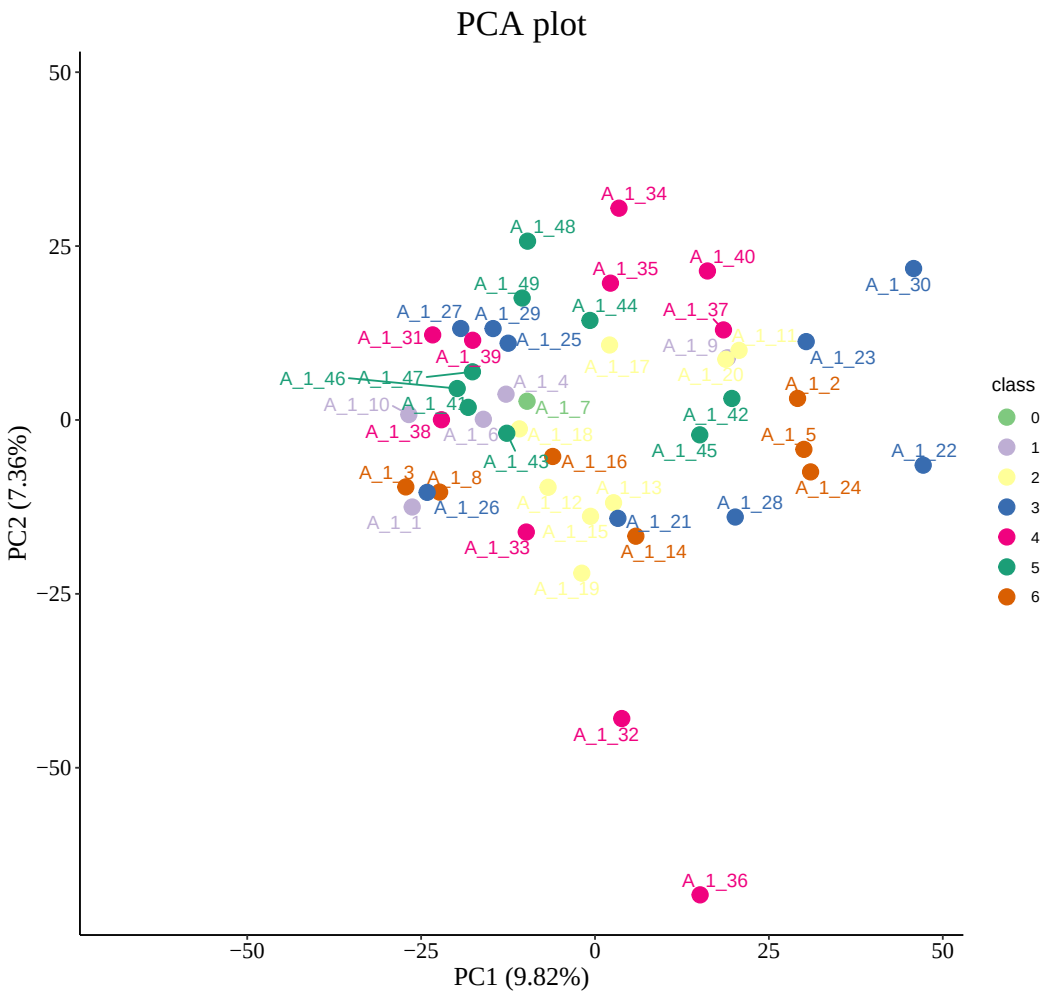
```r
randomforestinmpute <- function(samples) {
  num_cores <- detectCores()
  cl <- makeCluster(num_cores)
  registerDoParallel(cl)
  chunk_size <- 73
  num_chunks <- ceiling(nrow(samples) / chunk_size)
  impute_chunk <- function(start, end) {
    if (sum(complete.cases(samples[start:end, ])) >= 5) {
      im <- missForest(samples[start:end, ],
                       maxiter = 2,
                       ntree = 10)
      im[["ximp"]]

    } else {
      NA
    }
  }
  imputed_data <- foreach(i = 1:num_chunks,
                          .packages = "missForest") %dopar% {
    start <- (i - 1) * chunk_size + 1
    end <- min(i * chunk_size, nrow(samples))

    impute_chunk(start, end)
  }
  stopCluster(cl)
  imputed <- Filter(Negate(is.na), imputed_data)
  imputed <- do.call(rbind, imputed_data)
  return(imputed)
}
# Imputing the missing values using Random forest algorithm
pro_ab <- randomforestinmpute(samples = pro_ab)
# The number of missing values after imputation
sum(is.na(pro_ab))
```

```
## [1] 0
```

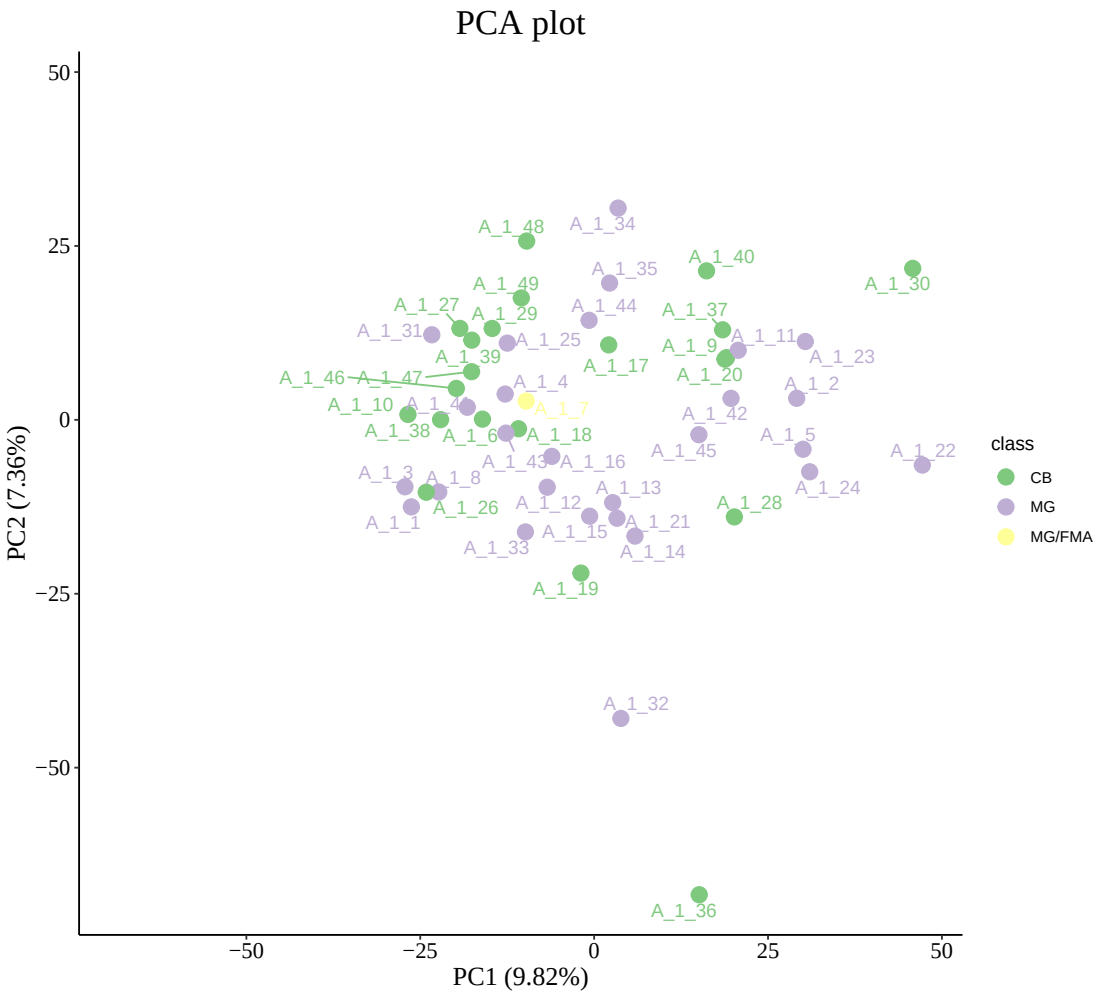
```r
VSNQuantilNorm <- function(counts) {
  vsnnorm <- normalizeVSN(counts)
  df_rank <- apply(vsnnorm,2,rank,ties.method="min")
  df_sorted <- data.frame(apply(vsnnorm, 2, sort))
  df_mean <- apply(df_sorted, 1, mean)
  index_to_mean <- function(my_index, my_mean){
    return(my_mean[my_index])
  }
  norm_final <- apply(df_rank, 2, index_to_mean, my_mean=df_mean)
  norm_final <- data.frame(norm_final)
  norm_final <- data.frame(sapply(norm_final, function(x) as.numeric(as.character(x))),
                           check.names=F)
  rownames(norm_final) <- rownames(counts)
  return(norm_final)
```

```
}
# VSN/Quantile normalization of protein abundance
pro_ab <- VSNQuantilNorm(counts = pro_ab)

mypalate <- function(){
  qual_colals <- brewer.pal.info[brewer.pal.info$category == 'qual',]
  col_vector <- unlist(mapply(brewer.pal,
                              qual_colals$maxcolors,
                              rownames(qual_colals)))
  col_vector <- col_vector[-c(3,7,8,12,14,17,24,40,41,44,45)]
  return(col_vector)
}

mytheme <- function() {
  theme(plot.title = element_text(hjust = 0.5),
        axis.text = element_text(family = "Times",
                                 size = 13 ,
                                 colour = "black"),
        axis.text.x = element_text(family = "Times",
                                   colour = "black",
                                   size = 13),
        axis.text.y = element_text(family = "Times",
                                   colour = "black"),
        plot.subtitle = element_text(family = "Times",
                                     size = 20,
                                     colour = "black",
                                     hjust = 0.5),
        axis.title.y = element_text(family = "Times",
                                    size = rel(1.4)),
        axis.title.x = element_text(family = "Times",
                                    size = rel(1.4)))
}


pcaplot <- function(counts,
                    class){
  class <- factor(class)
  counts <- t(counts)
  pca <- prcomp(counts, scale. = TRUE)
  varpca <- summary(pca)$importance[2,]
  pca_df <- data.frame(predict(pca, counts))
  label <- factor(rownames(pca_df))
  ggplot(pca_df, aes(x = PC1,
                         y = PC2,
                         color = class,
                         label = label)) +
    geom_point(size = 4) +
    labs(x = sprintf("PC1 (%2.2f%%)", varpca[1]*100),
         y = sprintf("PC2 (%2.2f%%)", varpca[2]*100),
         subtitle = "PCA plot") +
    geom_text_repel(max.overlaps = 50) +
    coord_obs_pred() +
    scale_color_manual(values = mypalate()) +
```

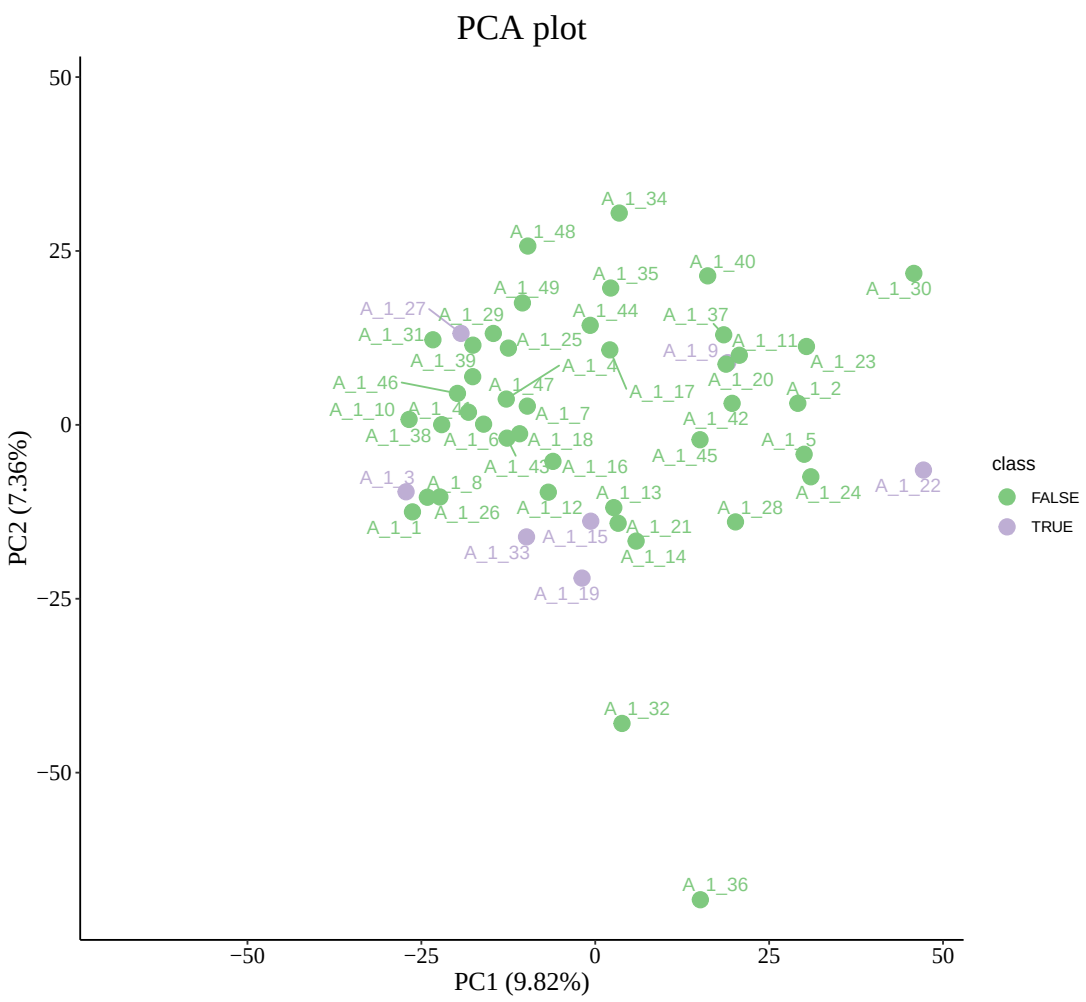
```
    theme_classic() +
    mytheme()
}

# PCA of different batches
pcaplot(counts = pro_ab,
        class = anno$batch)
```




```
# PCA of operator variable
pcaplot(counts = pro_ab,
        class = anno$operator)
```

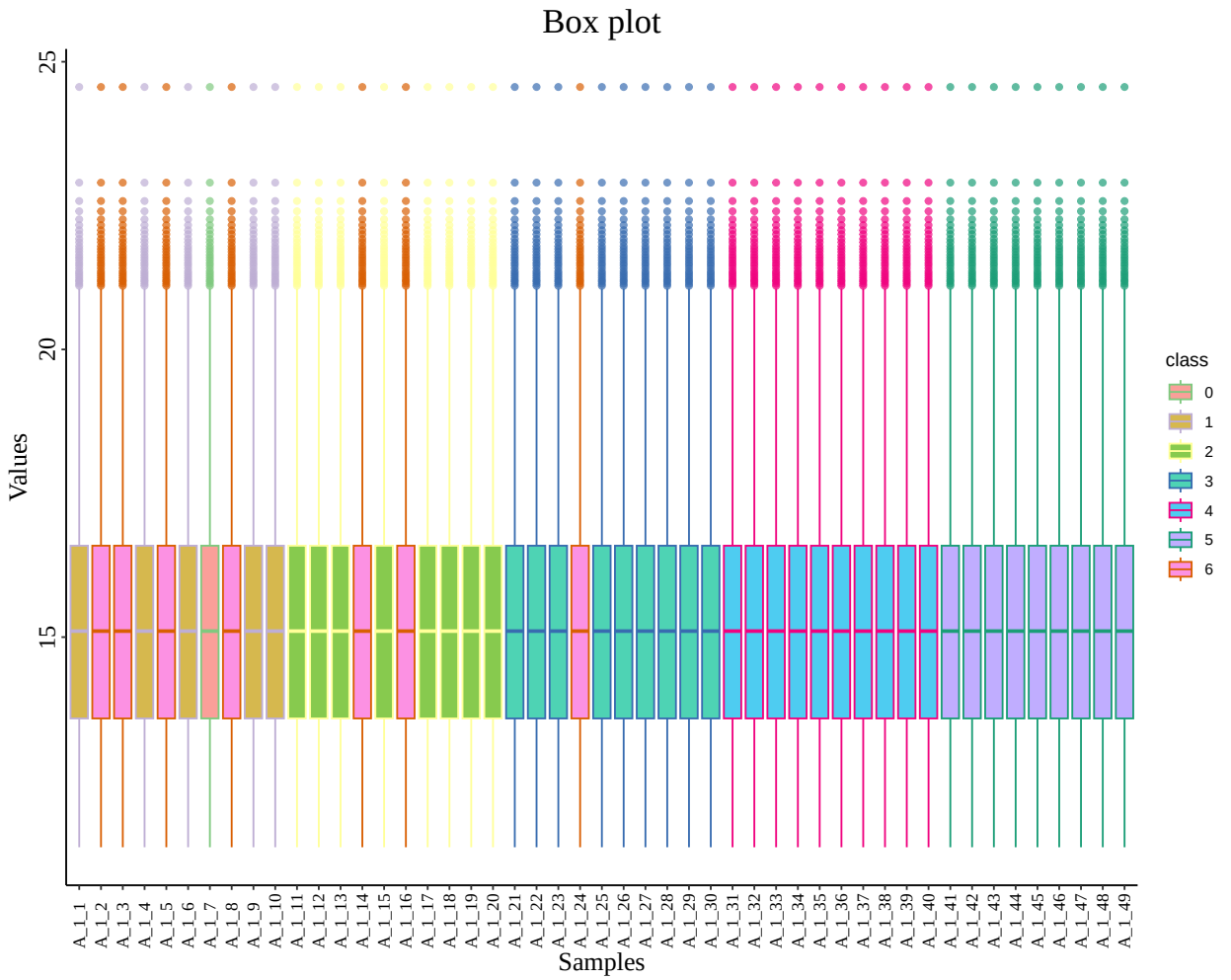



```
# PCA of two groups (died or alive)
pcaplot(counts = pro_ab,
        class = anno$died)
```

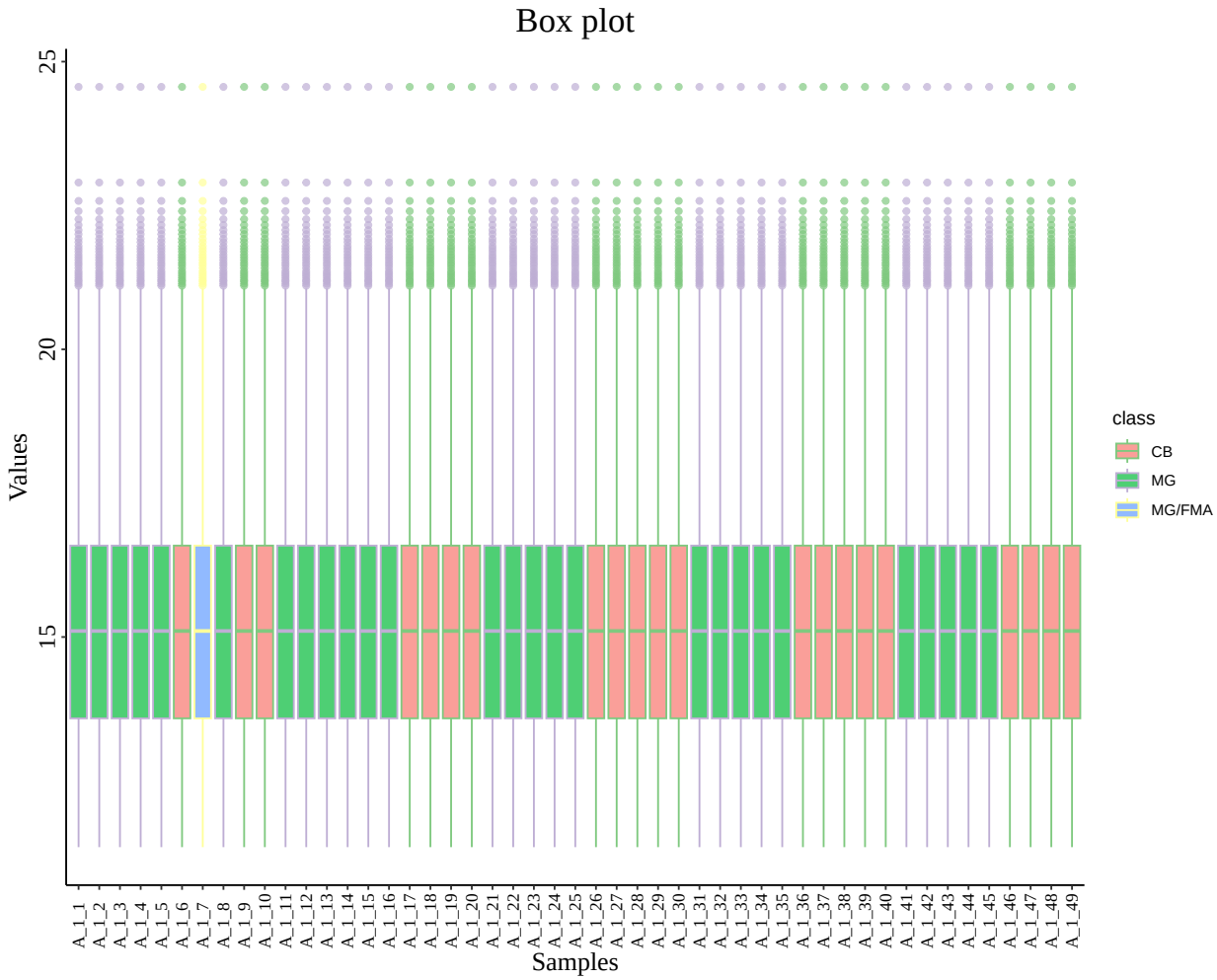



```
boxplot <- function(counts,
                    class) {
  class <- factor(class)
  counts <- data.frame(counts)
  counts <- cbind(rownames(counts),counts)
  colnames(counts)[1] <- "symbol"
  melted <- melt(counts, id.vars= "symbol")
  melted$class <- rep(class,
                      each = nrow(melted) / length(class))
  ggplot(data = melted, aes(x = variable,
                            y = value,
                            color = class,
                            fill = class)) +
      geom_boxplot(width=0.8,alpha=0.7) +
      labs(x= 'Samples', y= 'Values', subtitle = "Box plot") +
      scale_y_continuous(labels = scales::comma) +
      scale_color_manual(values = mypalate()) +
      theme_classic() +
      theme(axis.text = element_text(family = "Times",size = 13 , colour = "black", angle = 90),
            axis.text.x = element_text(colour = "black", size = 10,vjust = 0.5 , hjust = 0),
            axis.text.y = element_text(family = "Times",colour = "black", size = 14, angle = 90, hjust =
            plot.subtitle = element_text(family = "Times",size = 20, colour = "black", hjust = 0.5),
```

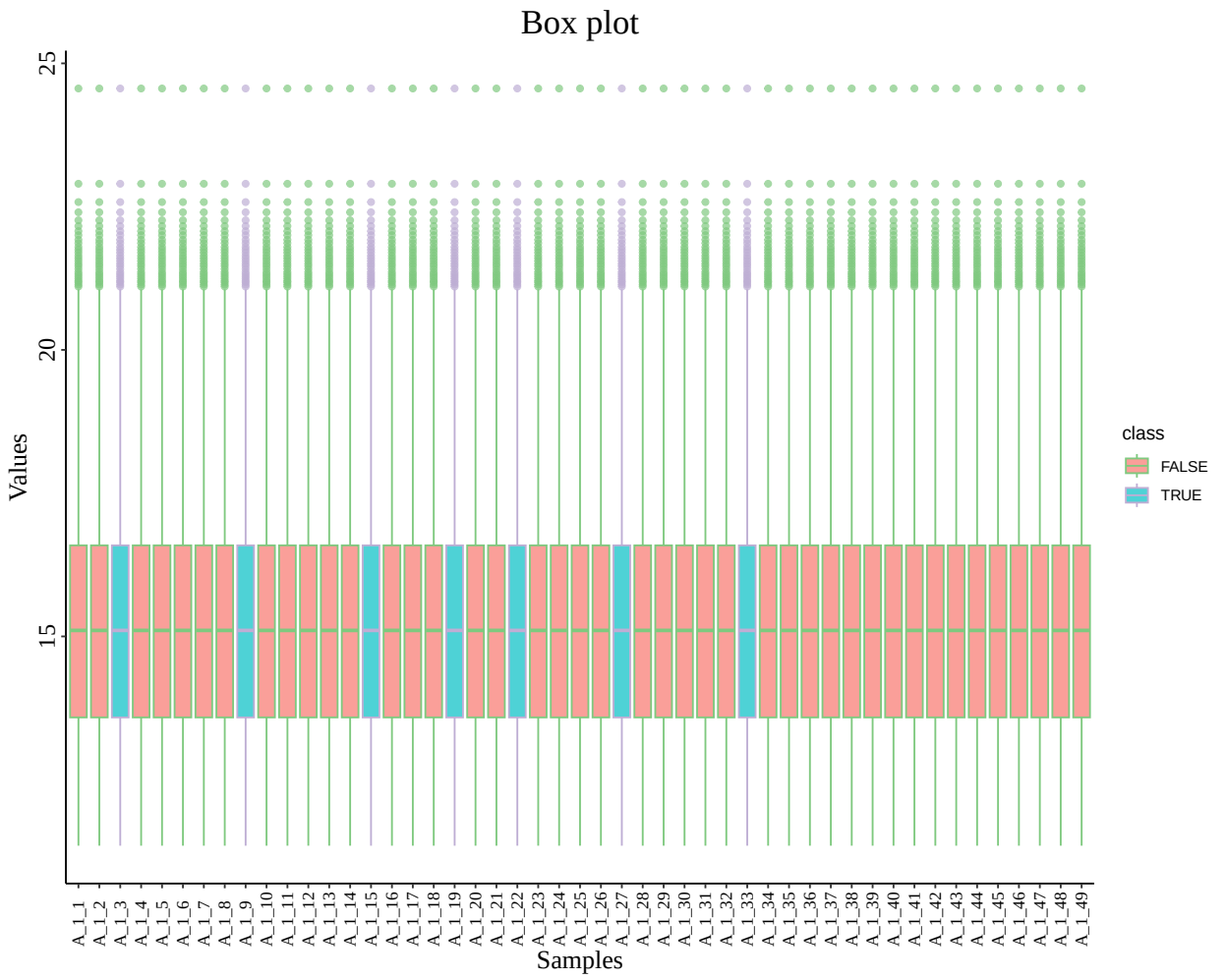
```
            axis.title.y = element_text(family = "Times", size = rel(1.4)),
            axis.title.x = element_text(family = "Times", size = rel(1.4)))
}

# Box plot of different batches
boxplot(counts = pro_ab,
        class = anno$batch)
```




```
# Box plot of operator variable
boxplot(counts = pro_ab,
        class = anno$operator)
```

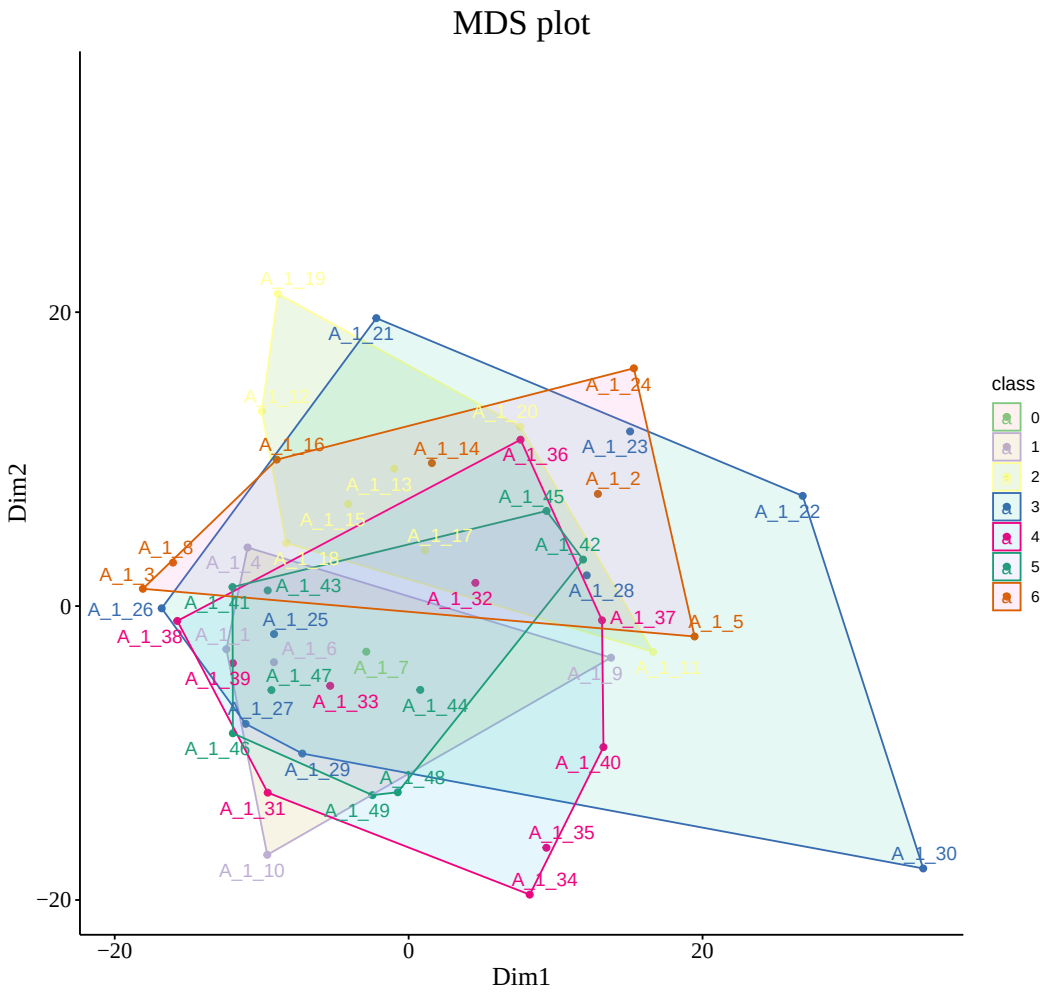



```
# Box plot of two groups (died or alive)
boxplot(counts = pro_ab,
        class = anno$died)
```

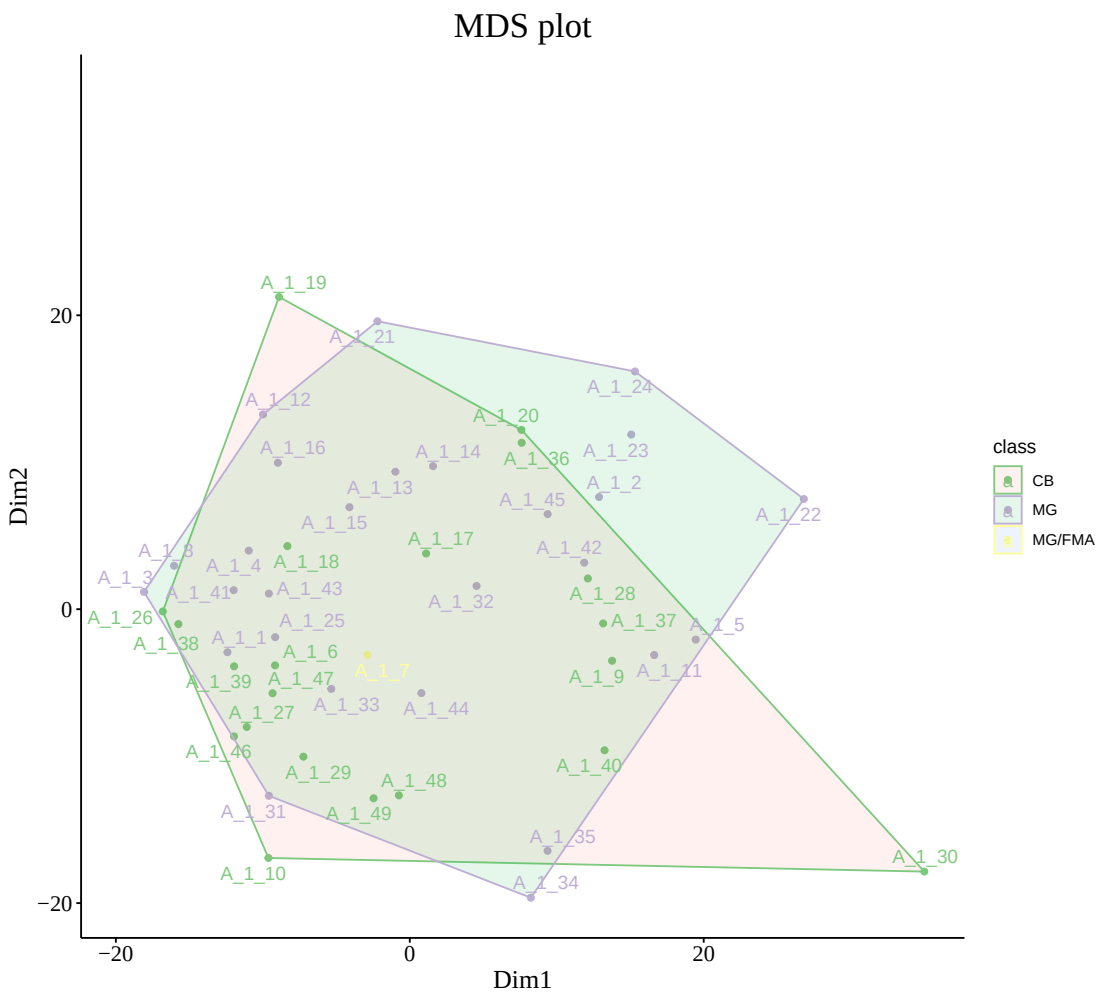



```
mdsplot <- function(counts,
                    class) {

  class <- factor(class)
  counts <- t(counts)
  mds <- counts %>%
    dist() %>%
    cmdscale() %>%
    data.frame()
  mds <- cbind(rownames(mds), mds)
  colnames(mds) <- c("label","Dim1", "Dim2")
  mds$class <- class
  ggscatter(mds, x = "Dim1", y = "Dim2",
            label = mds$label,
            ggtheme = theme_classic(),
            color = "class",
            size = 1.5,
            ellipse = TRUE,
            ellipse.type = "convex",
            repel = TRUE) +
    scale_y_continuous(labels = scales::comma) +
    scale_x_continuous(labels = scales::comma) +
```

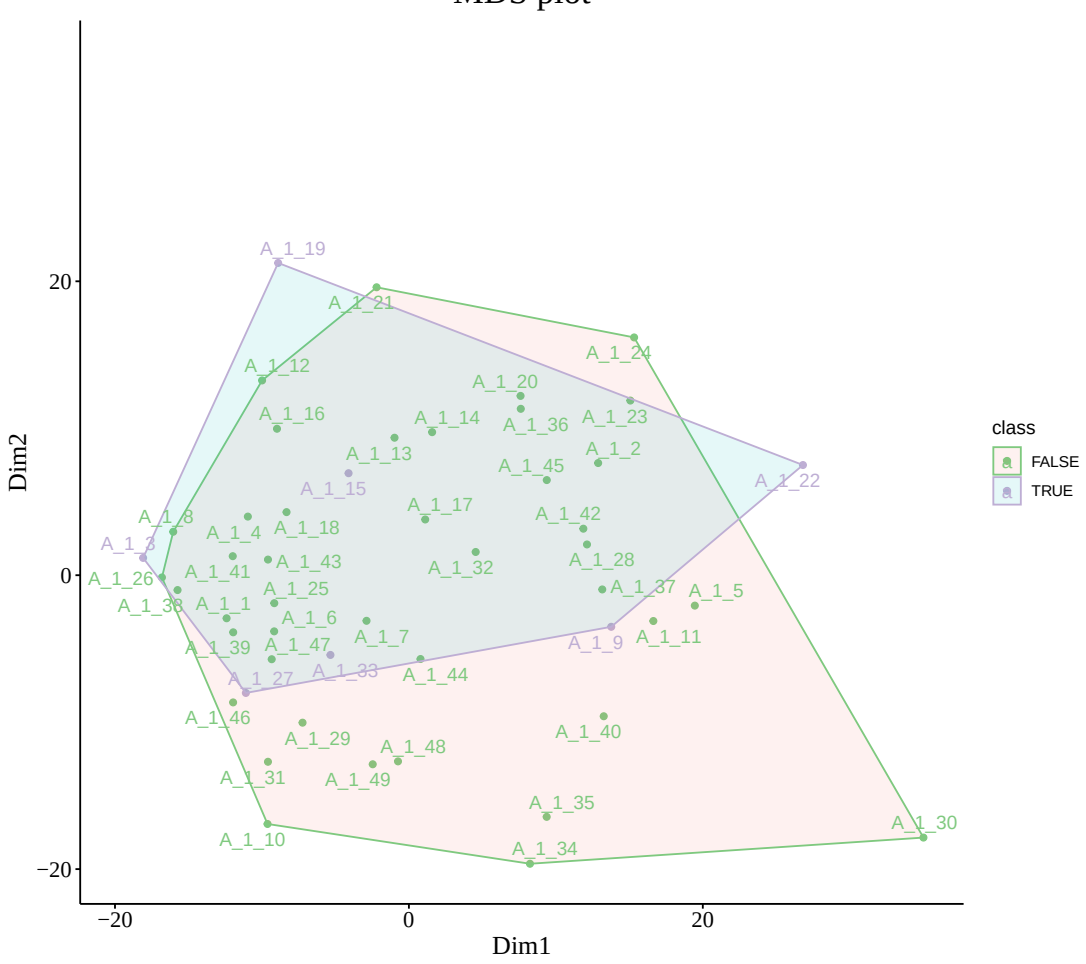
```
    scale_color_manual(values = mypalate()) +
    coord_obs_pred() +
    labs(subtitle = "MDS plot") +
    mytheme()
}

# MDS plot of different batches
mdsplot(counts = pro_ab,
        class = anno$batch)
```



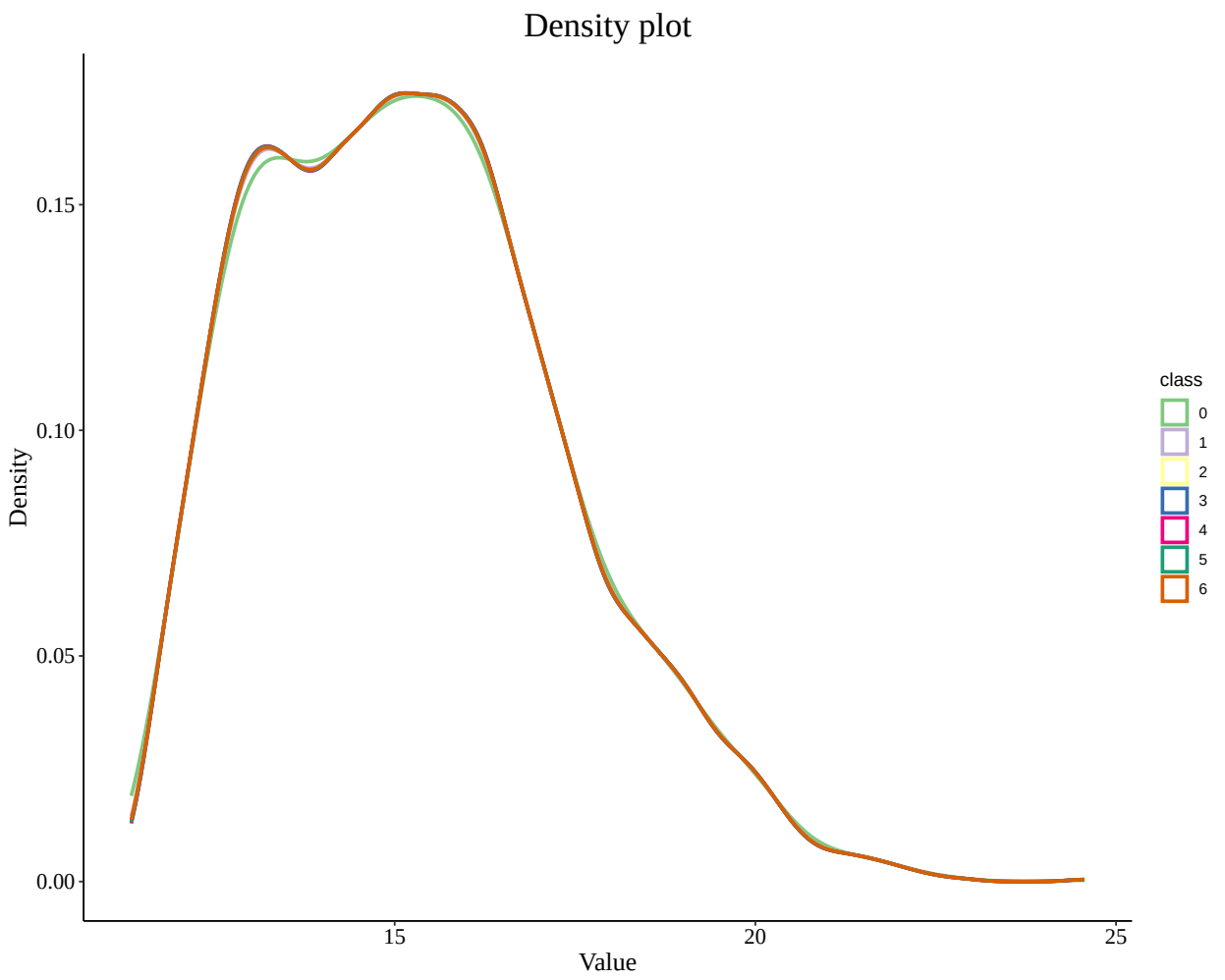

```
# MDS plot of operator variable
mdsplot(counts = pro_ab,
        class = anno$operator)
```



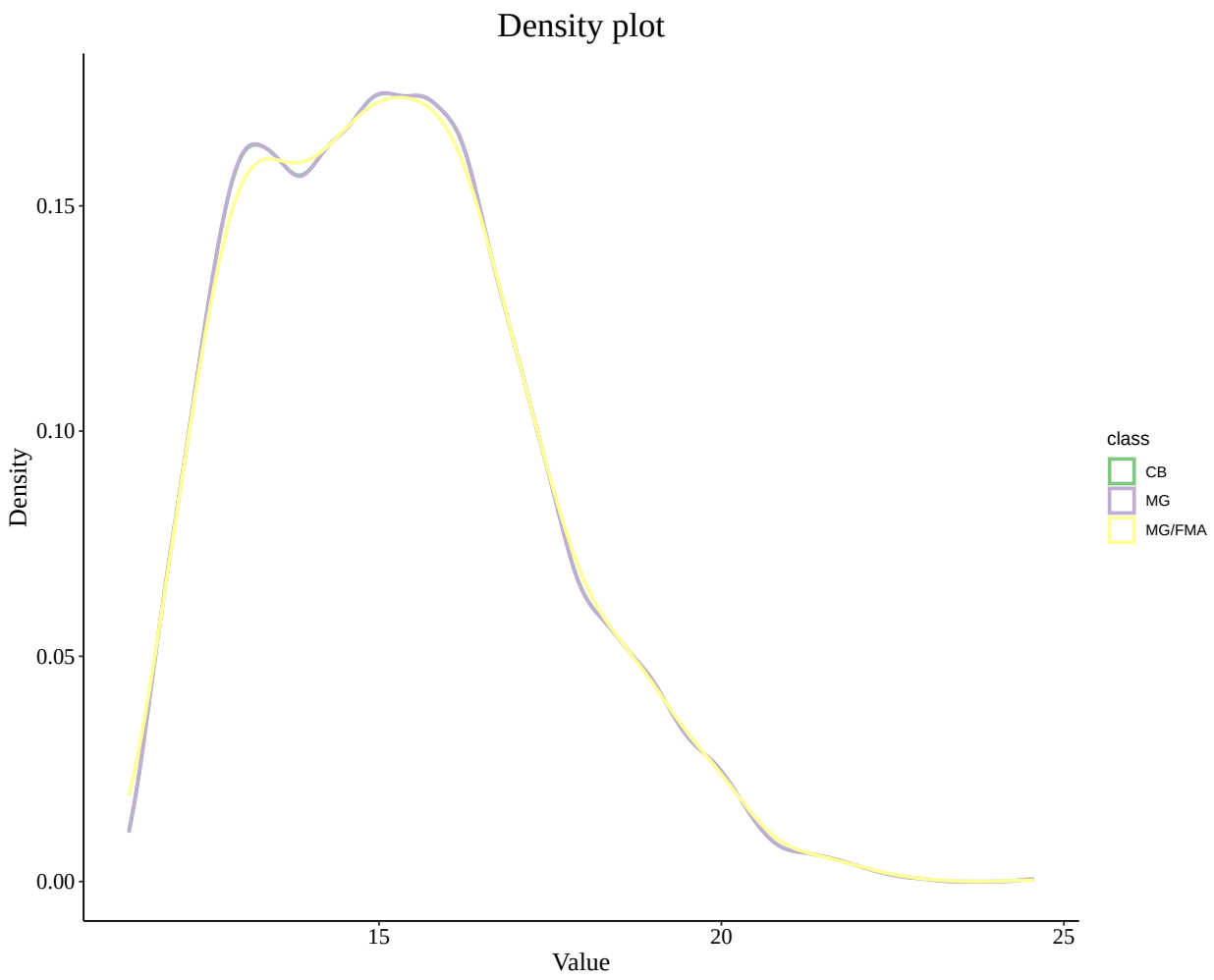

```
# MDS plot of two groups (died or alive)
mdsplot(counts = pro_ab,
        class = anno$died)
```



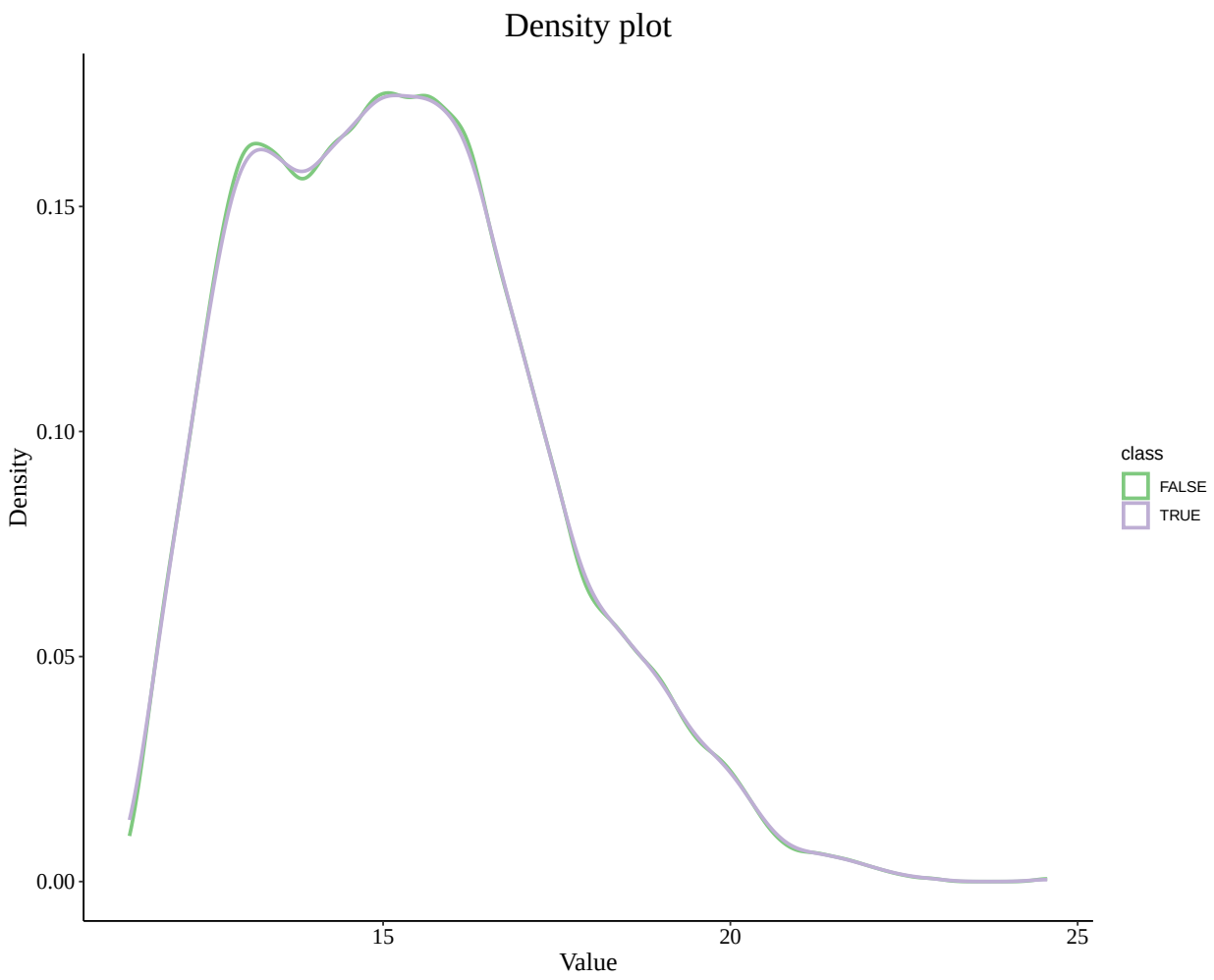

```
densityplot <- function(counts,
                        class) {
  class <- factor(class)
  counts <- data.frame(counts)
  counts <- cbind(rownames(counts),counts)
  colnames(counts)[1] <- "symbol"
  melted <- melt(counts, id.vars= "symbol")
  melted$class <- rep(class,
                      each = nrow(melted) / length(class))
  ggplot(data=melted,aes(x=value,
                         color=class)) +
    geom_density(linewidth = 1) +
    scale_color_manual(values = mypalate()) +
    labs(y= 'Density', x= 'Value',
         subtitle = "Density plot") +
    theme_classic() +
    mytheme()
}

# Density plot of different batches
densityplot(counts = pro_ab,
            class = anno$batch)
```




```
# Density plot of operator variable
densityplot(counts = pro_ab,
            class = anno$operator)
```

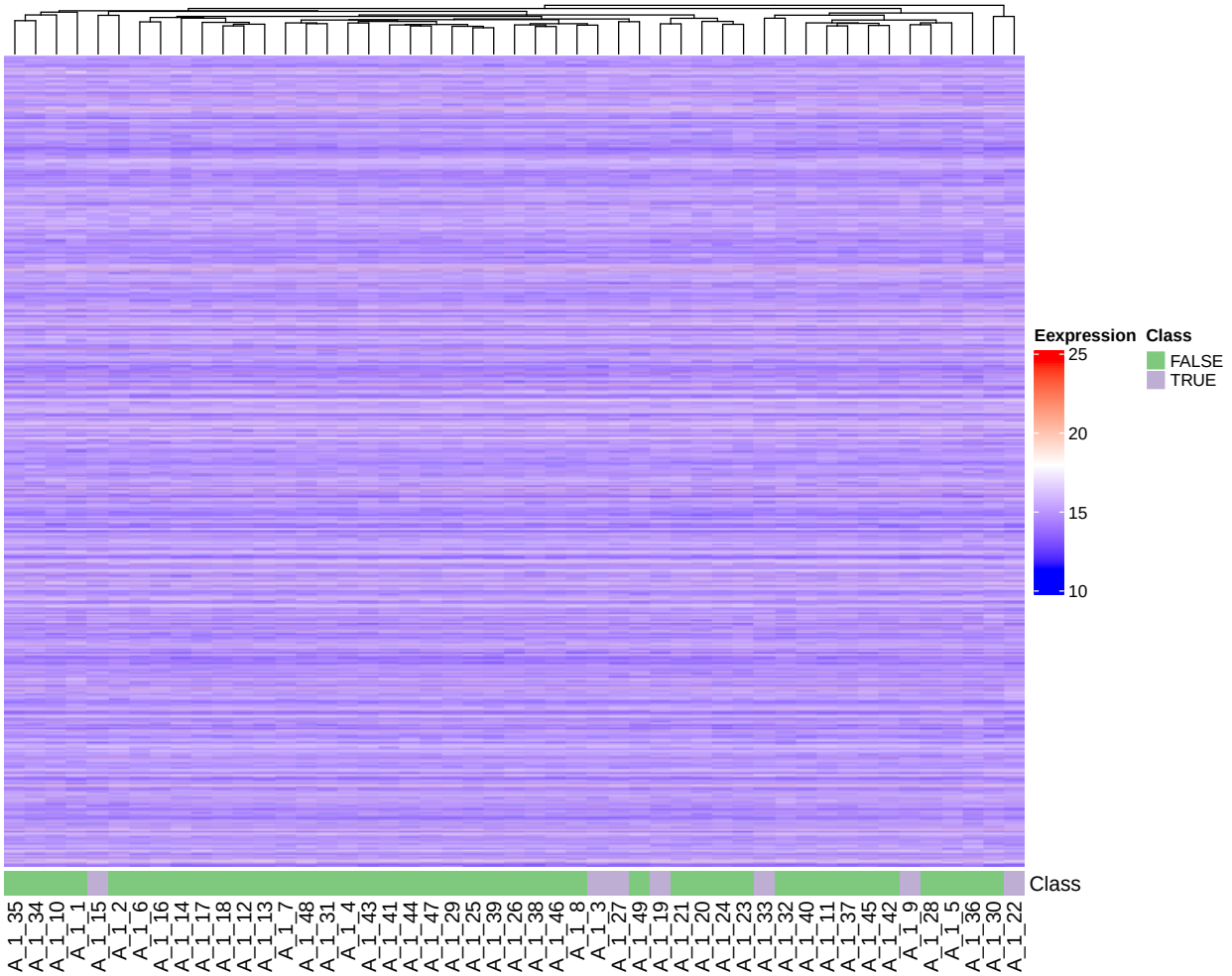



```
# Density plot of two groups (died or alive)
densityplot(counts = pro_ab,
            class = anno$died)
```

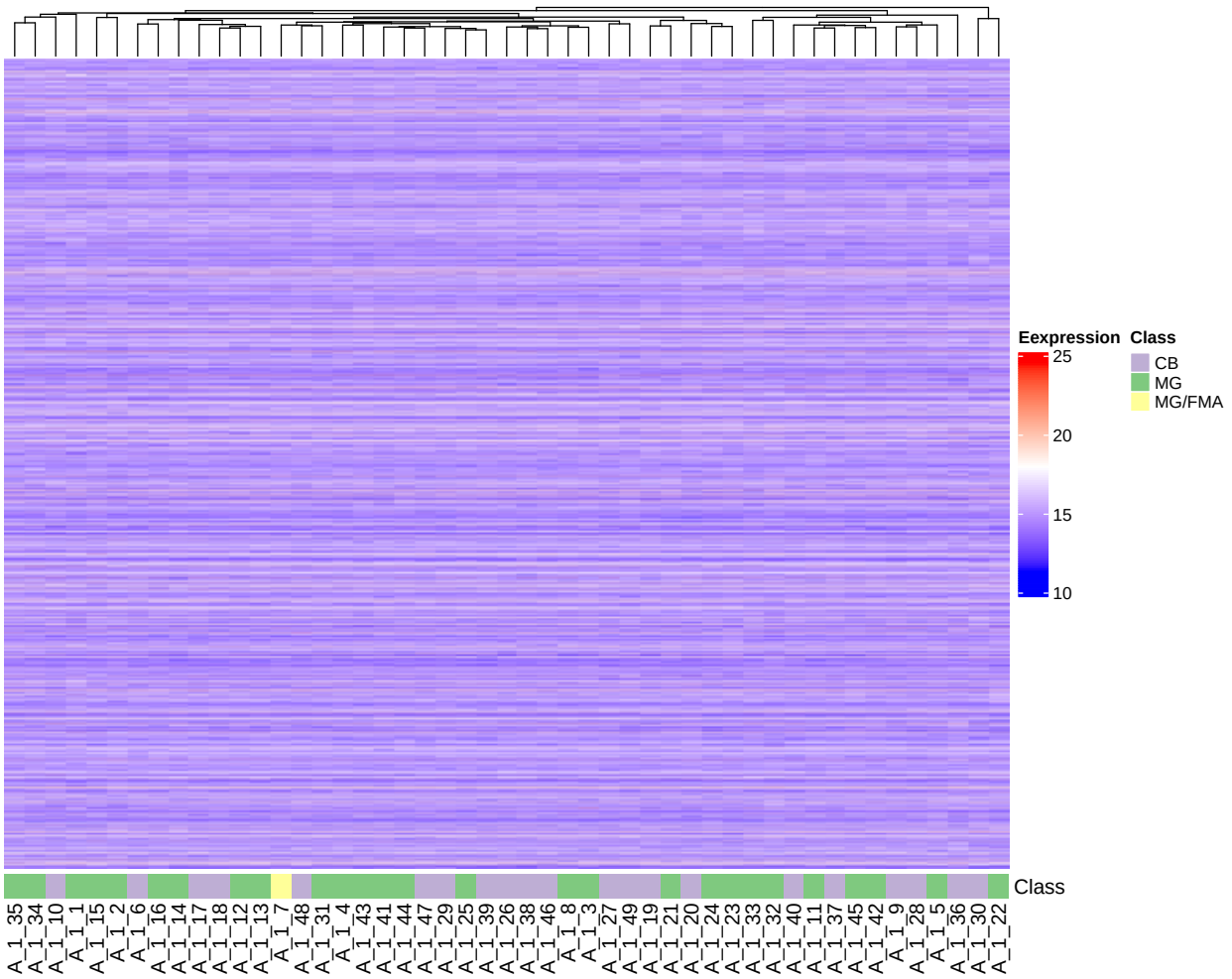



```
heatmapplot <- function(counts,
                        class){
  hm_df <- as.matrix(counts)

  Class <- factor(class)
  colors <- mypalate()[1:length(unique(Class))]
  names(colors) <- unique(Class)
  colors <- list(Class = colors)
  ha <- HeatmapAnnotation(
    df = data.frame(Class = Class),
    col = colors,
    annotation_height = unit(4, "mm")
  )

  Heatmap(hm_df,
          row_names_gp = gpar(fontszie = 4),
          bottom_annotation = ha,
          col = c("blue",
                  "white",
                  "red"),
          use_raster = TRUE,
          show_row_names = FALSE,
```

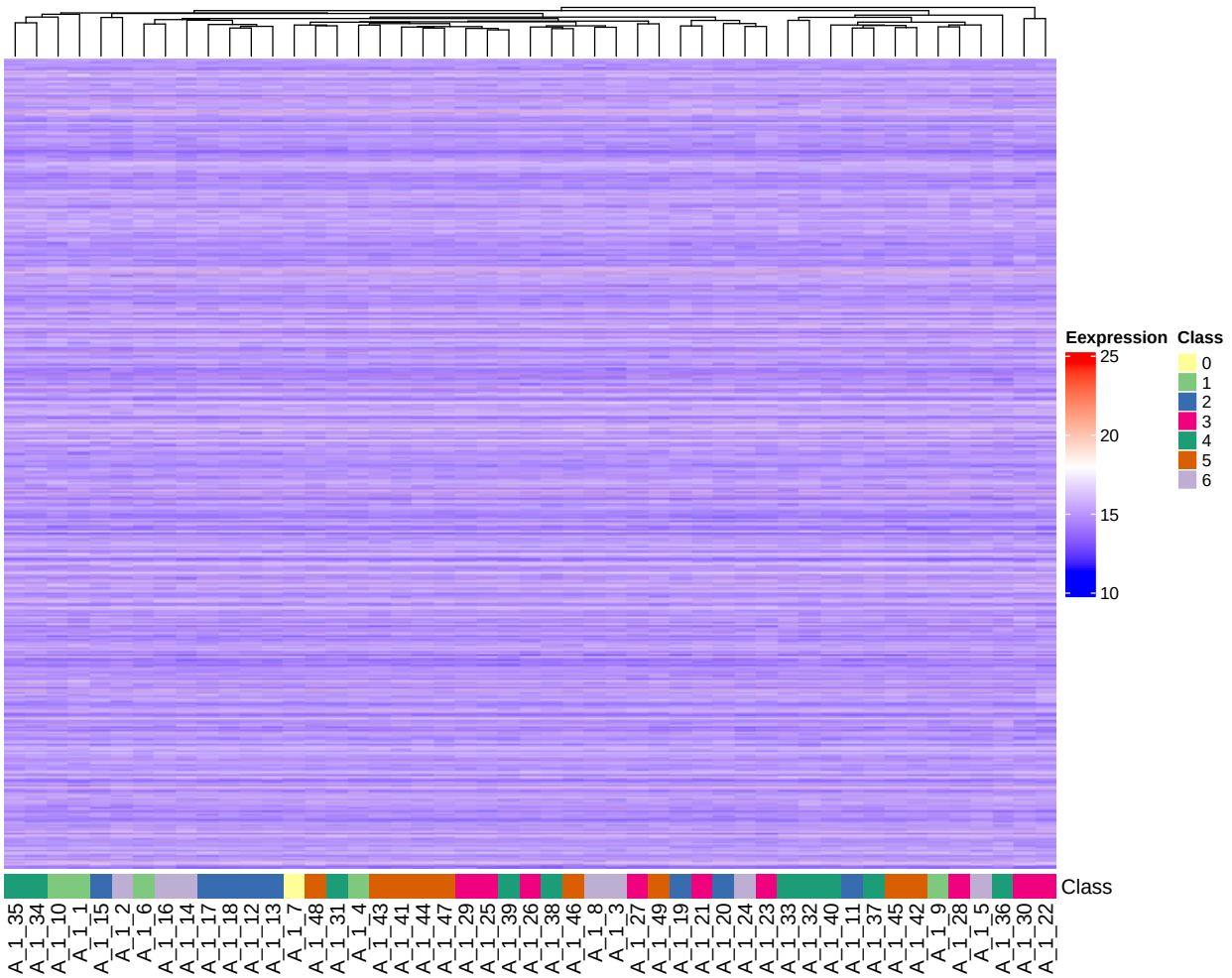
```
          cluster_rows = FALSE,
          cluster_columns = TRUE,
          show_column_names = TRUE,
          heatmap_legend_param = list(legend_direction = "vertical",
                                      legend_height = unit(50, "mm"),
                                      grid_width = unit(6, "mm"),
                                      grid_height = unit(50, "cm"),
                                      title = "Eexpression"))
}

# Heatmap of two groups (died or alive)
heatmapplot(counts = pro_ab,
            class = anno$died)
```




```
# Heatmap of operator variable
heatmapplot(counts = pro_ab,
            class = anno$operator)
```



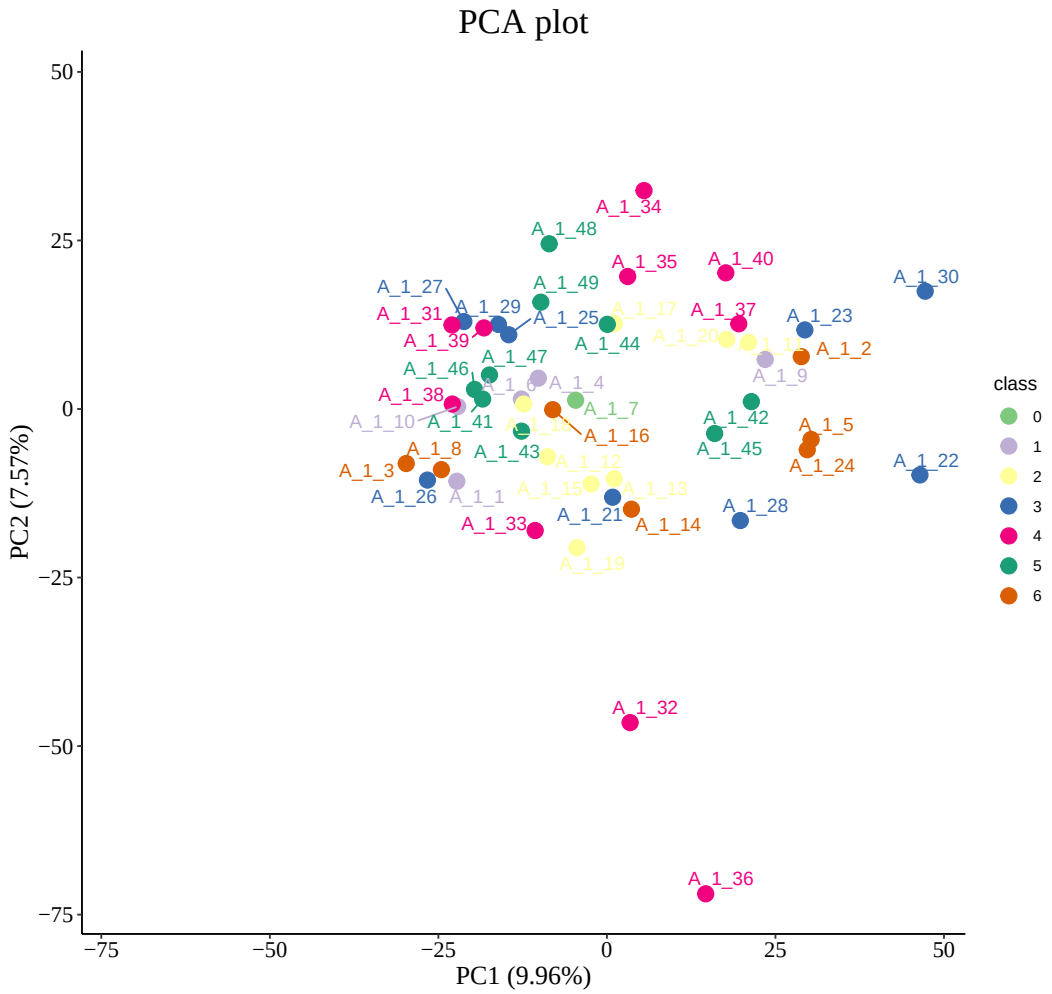

```
# Heatmap of batches
heatmapplot(counts = pro_ab,
            class = anno$batch)
```




```
batchcorrection <- function(counts, batch) {
  batch <- factor(batch)
  counts <- ComBat(counts,
                   batch = batch,
                   par.prior = TRUE)
  return(counts)
}

# Batch effect correction
corbatch <- batchcorrection(counts = pro_ab,
                            batch = anno$batch)
```

```
## Found 1 genes with uniform expression within a single batch (all zeros); these will not be adjusted
## Using the 'mean only' version of ComBat
## Found7batches
## Note: one batch has only one sample, setting mean.only=TRUE
## Adjusting for0covariate(s) or covariate level(s)
## Standardizing Data across genes
## Fitting L/S model and finding priors
## Finding parametric adjustments
## Adjusting the Data
```

```
# PCA after batch effect correction
pcaplot(counts = corbatch,
        class = anno$batch)
```




```
topproteins <- function(counts, ntop) {
  mad <- data.frame(mad = apply(counts, 1, mad))
  mad <- cbind(rownames(mad), mad)
  mad <- mad[order(mad$mad, decreasing = T),]
  mad <- rownames(mad)[1:ntop]
  counts <- counts[which(rownames(counts) %in% mad),]
  return(counts)
}
# Top variable proteins (1000) using MAD
top_pro <- topproteins(counts = corbatch,
                       ntop = 1000)
# Adding class label to the protein abundance matrix
# 0 = died and 1 = alive
top_pro <- data.frame(t(top_pro))
top_pro$event <- factor(anno$died, labels = c(0,1))

# Checking for class imbalance
as.data.frame(table(top_pro$event))
```

```
##   Var1 Freq
## 1    0   42
## 2    1    7
```

```
classimbalancecor <- function(counts, class) {
  formula <- as.formula(paste0(class, "~."))
  balanced <- SMOTE(formula,
                    counts,
                    perc.over = 300,
                    k = 3,
                    perc.under = 200)
  return(balanced)
}

# Correcting the class imbalance using
# creating new synthetic samples
balanced.data <- classimbalancecor(counts = top_pro,
                                   class = "event")

# Checking for class imbalance after correction
as.data.frame(table(balanced.data$event))
```

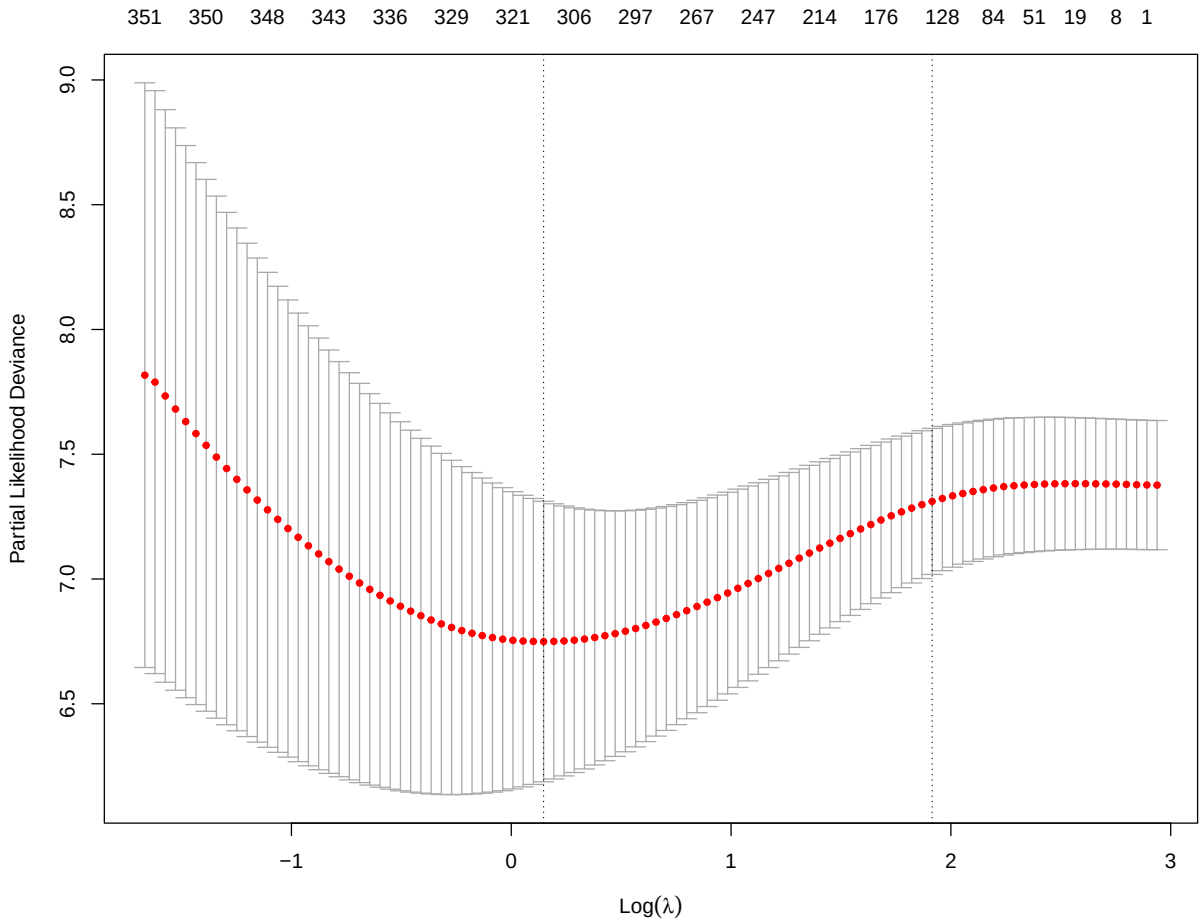
```
##   Var1 Freq
## 1    0   42
## 2    1   28
```

```
featureselection <- function(balanced,
                             ori_data,
                             class) {
  fac <- grep(class, colnames(balanced))
  split <- sample.split(balanced[, fac], SplitRatio = 0.8)
  training_set <- subset(balanced, split == TRUE)
  test_set <- subset(balanced, split == FALSE)
  formula <- as.formula(paste0(class, "~."))
  rfimp <- randomForest(formula,
                        data = training_set,
                        ntree = 300,
                        importance=TRUE)
  y_pred <- predict(rfimp, newdata = test_set[-fac])
  print(MBMethPred::ConfusionMatrix(y_true = test_set[, fac],
                                    y_pred = y_pred))
  imp <- varImp(rfimp)
  imp <- imp[imp != 0,]
  imp <- na.omit(imp)
  counts <- ori_data[, c(which(colnames(ori_data) %in% rownames(imp)))]
  return(counts)
}

# Feature selection using Random Forest model
imp_pro <- featureselection(balanced = balanced.data,
                            ori_data = top_pro,
                            class = "event")
```

```
##       y_pred
## y_true 0 1
##      0 8 0
```

```
##      1 0 6
##
##   Accuracy Precision Sensitivity F1_Score Specificity AUC
## 1        1         1           1        1           1   1
```

```r
# Combining the survival data with top selected proteins

imp_pro <- cbind(anno[,c(7,9)], imp_pro)
colnames(imp_pro)[1] <- "event"

elasticnet <- function(data, alpha) {
  X <- model.matrix(Surv(time = data$time,
                         event = data$event) ~ .,
                         data = data)
  Y <- Surv(time = data$time, event = data$event)
  model <- cv.glmnet(X, Y,
                     family = "cox",
                     alpha = alpha)
  plot(model)
  coefficients <- coef(model, s = "lambda.min")
  coefficients <- as.matrix(coefficients)
  coefficients <- cbind(rownames(coefficients), coefficients)
  coefficients <- as.data.frame(coefficients)
  coefficients <- coefficients[-1,]
  colnames(coefficients) <- c("Protein", "Coefficient")
  coefficients <- coefficients[which(coefficients$Coefficient != 0),]
  coefficients$Protein <- gsub("^sp\\.([^\\.]+)\\..+", "\\1",
                               coefficients$Protein)
  return(coefficients)

}

# Modeling the top proteins using Elastic net
# For finding biomarkers
coef <- elasticnet(data = imp_pro, alpha = 0.009)
```



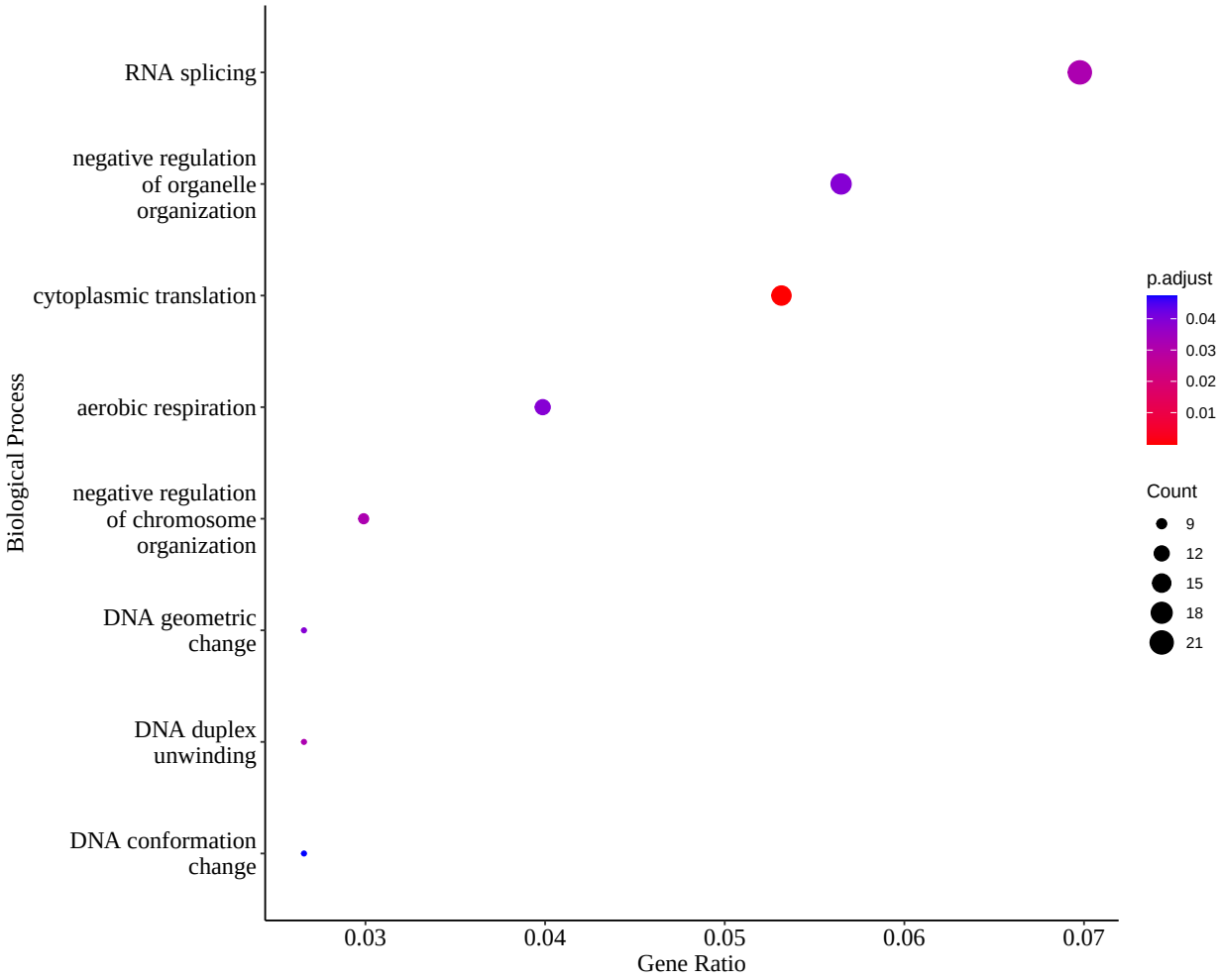

```
head(coef)
```

```
##                         Protein          Coefficient
## sp.A5YKK6.CNOT1_HUMAN  A5YKK6   0.0180857392860116
## sp.A6NIH7.U119B_HUMAN  A6NIH7  -0.0206561426758042
## sp.B0I1T2.MYO1G_HUMAN  B0I1T2 -0.00592674312287593
## sp.C9J7I0.UMAD1_HUMAN  C9J7I0  0.00984232802051917
## sp.O00257.CBX4_HUMAN   O00257  0.00718132104407253
## sp.O00267.SPT5H_HUMAN  O00267   0.0312008449439501
```

```
write_csv(coef, "coefficients.csv")

goresult <- function(coefficients,
                     p_adj,
                     gotop){

  coefficients$entrez <- mapIds(org.Hs.eg.db,
                                keys = coefficients$Protein,
                                column = "ENTREZID",
                                keytype = "UNIPROT",
                                multiVals = "first")
  coefficients <- na.omit(coefficients)
  go_enrich <- enrichGO(gene = coefficients$entrez,
```

```
                        OrgDb = org.Hs.eg.db,
                        ont = "BP")

  go_enrich <- go_enrich@result
  go_enrich <- go_enrich[which(go_enrich$p.adjust < p_adj), ]
  write_tsv(go_enrich, "enriched_proteins.tsv")
  if(nrow(go_enrich) > 0) {
    go_enrich$total <- as.numeric(gsub("[0-9]+/", "", go_enrich$GeneRatio))
    go_enrich$GeneRatio <- go_enrich$Count / go_enrich$total
    go_enrich <- go_enrich[1:gotop, ]
    go_enrich <- na.omit(go_enrich)
    N <- 2
    go_enrich$Description <- sapply(strsplit(go_enrich$Description, " "), function(words) {
      n <- length(words)
      if (n > N) {
        chunks <- split(words, ceiling(seq_along(words) / N))
        chunks <- lapply(chunks, paste, collapse = " ")
        paste(chunks, collapse = "\n")
      } else {
        paste(words, collapse = " ")
      }
    })
    ggplot(data = go_enrich, aes(x = reorder(Description, GeneRatio, sum),
                                 y = GeneRatio, color = p.adjust, size = Count)) +
      geom_point() +
      coord_flip() +
      scale_color_gradient(low = "red", high = "blue") +
      xlab("Biological Process") +
      ylab("Gene Ratio") +
      theme_classic() +
      theme(axis.line = element_line(linetype = "solid"),
            axis.title = element_text(family = "Times",size = 14),
            axis.text = element_text(family = "Times",
                                     size = 14, colour = "black"))
  }
}

# Finding biological processes related to biomarkers
goresult(coefficients = coef,
         p_adj = 0.05,
         gotop = 10)
```

```
## 'select()' returned 1:many mapping between keys and columns
```


RNA splicing
negative regulation of organelle organization
cytoplasmic translation
aerobic respiration
negative regulation of chromosome organization
DNA geometric change
DNA duplex unwinding
DNA conformation change
Biological Process
0.03
0.04
0.05
0.06
0.07
Gene Ratio
p.adjust
0.04
0.03
0.02
0.01
Count
9
12
15
18
21