RQdeltaCT - relative quantification of gene expression using delta Ct methods

Daniel Zalewski daniel.piotr.zalewski@gmail.com

19 February 2024

Introduction

`RQdeltaCT` is an R package developed to perform relative quantification of gene expression using delta Ct methods, proposed by Kenneth J. Livak and Thomas D. Schmittgen in Article1 and Article2.

These methods were designed to analyse gene expression data (Ct values) obtained from real-time PCR experiments. The main idea is to normalise gene expression values using endogenous control gene, present gene expression levels in linear form by using the $2^{-(value)}$ transformation, and calculate differences in gene expression levels between groups of samples (or technical replicates of a single sample).

There are three main delta Ct methods used for relative quantification. The choice of the best method depends on the study design. A short description of these methods is provided below; for more details, refer to articles in links provided above.

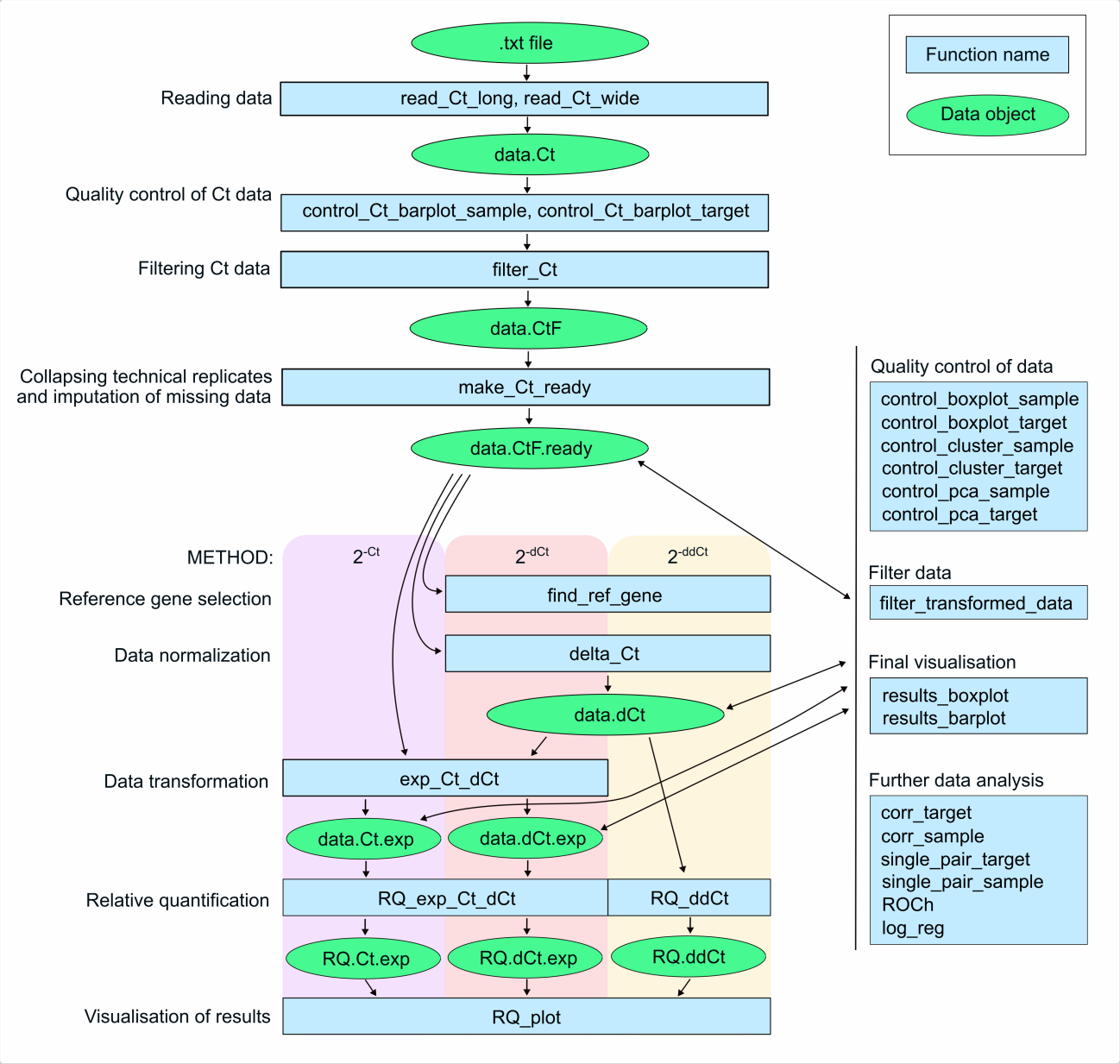
1. 2^{-Ct} method. This method can be used in studies where several genes are tested if they are influenced by the studied experimental condition and are suitable to be used as reference genes (also called endogenous controls or internal controls). The raw Ct values obtained from the experiments are transformed using the 2^{-Ct} formula, summarised by means in the compared study conditions, and a ratio of means (fold change) is calculated for the experimental condition. If the obtained fold change value is assessed as significant, the gene would be not a good choice as the reference gene. See example no. 3 in Article2.
2. 2^{-dCt} method. In this method, Ct values are normalised by the endogenous control gene (often GAPDH, beta-actin, or other) by subtracting the Ct value of the endogenous control in each sample from the Ct value of the gene of interest in the same samples, obtaining delta Ct (dCt) values. Subsequently, the dCt values are transformed using the 2^{-dCt} formula, summarised by means in the compared study groups, and a ratio of means (fold change) is calculated for a study group. This method is useful in scenarios where samples should be analysed as individual data points, e.g., in comparison between patients and healthy subjects. See example no 5. in Article2.
3. 2^{-ddCt} method. Similarly to the 2^{-dCt} method, Ct values are normalised by endogenous control gene, but the obtained delta Ct (dCt) values are not exponentially transformed, but summarised by means in the compared study groups, and the mean dCt in a control group is subtracted from the mean dCt in a study group, giving the delta delta Ct (ddCt) value. Subsequently, the ddCt values are transformed using the 2^{-ddCt} formula to obtain the fold change value (also called the RQ value). This method is useful where a compared groups contain rather technical than biological replicates, e.g. where samples of cell line before adding stimulant are compared to samples of the same cell line after stimulation. See examples no. 1 and 2 in Article2.

Presented `RQdeltaCT` package includes functions that encompass all of these methods. The selection of a suitable method for analysis is up to the user.

The functions developed within the `RQdeltaCT` package are designed to be maximally easy to use, even for the users who are beginners to R. The parameters of functions were prepared to sufficiently range all

essential tasks and options, and no additional, extensive coding steps are necessary in standard workflow. The package was developed with the intention to provide an opportunity to perform relative quantification analysis of gene expression using `RQdeltaCT` package by non-experts in R programming (only a very basic skills are required).

The summary of standard workflow, including function names and data flow



The entire standard workflow of analysis performed using the `RQdeltaCT` package requires the following external packages:

- `tidyverse` - main package for data processing (`dplyr`, `tidyr`) and visualisation (`ggplot2`),
- `coin` - used to perform the Mann-Whitney U test (`wilcox_test()` function),
- `ctrlGene` - used to calculate gene expression stability score using geNorm algorithm,
- `ggsignif` - used to add significance labels to plots,
- `Hmisc` - used to perform correlation analysis (`rcor()` function),
- `corrplot` - used to visualise results of correlation analysis (`corrplot()` function),
- `ggpmisc` - used to add linear regression results to the plot (`stat_poly_eq()` function),
- `pROC` - used to perform analysis (`roc()` function),
- `oddsratio` - used to compute odds ratio values (`or_glm()` function).

All plots created by functions of the `RQdeltaCT` package can be saved as a .tiff files in the working directory (if `save.to.tiff` parameter is set to TRUE). Image resolution, dimensions, and name can be specified by the user. Also, all generated tables can be saved as .txt files (if `save.to.txt` parameter is set to TRUE) with name specified by the user.

This vignette includes instructions and examples of usage of main parameters of functions from `RQdeltaCT` package. For all list of parameters available and their usage, refer to documentation of a particular function.

Data import

Data analysed with the `RQdeltaCT` package should be in tabular form and contain the following information: group names, sample names, gene names, and Ct values. Flag information can also be included for data filtering purposes. Any other information could exist in the data (user do not have to remove them), but will not be used for analysis.

Files with such tables typically can be exported from software coupled with PCR devices and used to analyse raw data files generated during real-time PCR experiments. Such files are also returned by external software, such as SDS, or even R packages, e.g. qpcR

For user convenience, the `RQdeltaCT` package provides two functions useful to import tables in .txt format: * `read_Ct_long()` - to import tables with long-format structure (each information in columns), * `read_Ct_wide()` - to import tables with wide-format structure (samples by columns, genes by rows).

NOTE: Imported tables must be free of empty lines.

1. Reading long-format data using the `read_Ct_long()` function.

Example of a long-format table with a structure suitable for the `read_Ct_long()` function:

Group	Sample	Gene	Ct	Flag
Disease	Disease1	Gene1	25.6	OK
Disease	Disease2	Gene2	32.9	Undetermined
Control	Control1	Gene1	Undetermined	OK
Control	Control2	Gene2	27.5	OK
…	…	…	…	…

For the purpose of presentation of the `read_Ct_long()` function, this function will be used to import long-format .txt table (data_Ct_long.txt) located in `RQdeltaCt` package directory:

```
# Set path to file:
path <- system.file("extdata",
                    "data_Ct_long.txt",
                    package = "RQdeltaCT")

# Import file using path; remember to specify proper separator, decimal character, and numbers of
         necessary columns:
library(RQdeltaCT)
library(tidyverse)
data.Ct <- read_Ct_long(path = path,
                        sep = "\t",
                        dec = ".",
                        skip = 0,
```

```
                    add.column.Flag = TRUE,
                    column.Sample = 1,
                    column.Gene = 2,
                    column.Ct = 5,
                    column.Group = 9,
                    column.Flag = 4)
```

Let's look at the data structure:

```
str(data.Ct)
#> 'data.frame':    1288 obs. of  5 variables:
#>  $ Sample: chr  "Disease1" "Disease10" "Disease12" "Disease13" ...
#>  $ Gene  : chr  "Gene1" "Gene1" "Gene1" "Gene1" ...
#>  $ Ct    : chr  "32.563" "34.648" "35.059" "37.135" ...
#>  $ Group : chr  "Disease" "Disease" "Disease" "Disease" ...
#>  $ Flag  : num  1.38 1.35 1.34 1.2 1.39 ...
```

The data were imported properly; however, the Flag variable is numeric, but character or factor is required. The Flag column contains a numeric AmpScore parameter, which is often used to evaluate the quality of the amplification curve - curves with AmpScore below 1 are typically considered as low quality and removed from data during analysis. The Flag variable can be changed into character by transforming the numeric AmpScore parameter into a binary variable that contains values "OK" and "Undetermined" according to the applied AmpScore criterion:

```
library(tidyverse)
data.Ct <- mutate(data.Ct,
                  Flag = ifelse(Flag < 1, "Undetermined", "OK"))
str(data.Ct)
#> 'data.frame':    1288 obs. of  5 variables:
#>  $ Sample: chr  "Disease1" "Disease10" "Disease12" "Disease13" ...
#>  $ Gene  : chr  "Gene1" "Gene1" "Gene1" "Gene1" ...
#>  $ Ct    : chr  "32.563" "34.648" "35.059" "37.135" ...
#>  $ Group : chr  "Disease" "Disease" "Disease" "Disease" ...
#>  $ Flag  : chr  "OK" "OK" "OK" "OK" ...
```

In this transformation, all AmpScore values in the Flag column that are below 1 were changed to "Undetermined", otherwise to "OK". The Flag variable now is character, as required, and the data are ready for further steps of analysis.

2. Reading wide-format data using the `read_Ct_wide()` function.

The `read_Ct_wide()` function was designed to import data with gene expression results in the form of a wide-format table (with sample names in the first row and gene names in the first column). Because such a structure does not include names of groups, an additional file is required, containing group names and assigned samples. This second file must contain two columns: column named "Sample" with the names of the samples and column named "Group" with the names of the groups assigned to the samples. The names of the samples in this file must be the same as the names of the columns in the file with Ct values (the order does not have to be kept).

Example structure of a wide-format table suitable for the `read_Ct_wide()` function:

Gene	Sample1	Sample2	Sample3	…
Gene1	25.4	24.9	25.6	…

Gene	Sample1	Sample2	Sample3	…
Gene2	21.6	22.5	20.8	…
Gene3	33.7	Undetermined	Undetermined	…
Gene4	15.8	16.2	17.5	…
…	…	…	…	…

Example structure of an additional file suitable for the `read_Ct_wide()` function:

Sample	Group
Sample1	Control
Sample2	Control
Sample3	Disease
Sample4	Disease
…	…

In the following example, the `read_Ct_wide()` function was used to import and merge a wide-format file with Ct values (data_Ct_wide.txt) and a file with names of groups (data_design.txt) located in `RQdeltaCt` package directory:

```
# Set paths to required files:
path.Ct.file <- system.file("extdata",
                            "data_Ct_wide.txt",
                            package = "RQdeltaCT")
path.design.file <- system.file("extdata",
                                "data_design.txt",
                                package = "RQdeltaCT")

# Import files:
library(tidyverse)
data.Ct <- read_Ct_wide(path.Ct.file = path.Ct.file,
                        path.design.file = path.design.file,
                        sep ="\t",
                        dec = ".")

# Look at the structure:
str(data.Ct)
#> tibble [1,216 × 4] (S3: tbl_df/tbl/data.frame)
#>  $ Gene  : chr [1:1216] "Gene1" "Gene1" "Gene1" "Gene1" ...
#>  $ Sample: chr [1:1216] "Disease1" "Disease10" "Disease12" "Disease13" ...
#>  $ Ct    : chr [1:1216] "32.563" "34.648" "35.059" "37.135" ...
#>  $ Group : chr [1:1216] "Disease" "Disease" "Disease" "Disease" ...
```

The table imported from the package data object can be directly subjected to further steps of analysis.

It also can be a situation, in which an imported wide-format table has samples by rows and genes by columns. There is no need to develop separate function to import file with such a table, we can do:

```r
# Import file, be aware to specify parameters that fit to imported data:
data.Ct.wide <- read.csv(file = "data/data.Ct.wide.vign.txt",
                         header = TRUE,
                         sep = ",")
str(data.Ct.wide)
#> 'data.frame':    64 obs. of  22 variables:
#>  $ X     : int  1 2 3 4 5 6 7 8 9 10 ...
#>  $ Group : chr  "Disease" "Disease" "Disease" "Disease" ...
#>  $ Sample: chr  "Disease1" "Disease10" "Disease12" "Disease13" ...
#>  $ Gene1 : num  32.6 34.6 35.1 37.1 34 ...
#>  $ Gene2 : chr  "36.554" "37.262" "36.977" "38.295" ...
#>  $ Gene3 : chr  "31.334" "31.161" "32.077" "33.982" ...
#>  $ Gene4 : num  23.4 22.4 24.3 24.9 24.1 ...
#>  $ Gene5 : chr  "35.608" "33.385" "36.374" "36.997" ...
#>  $ Gene7 : num  32.8 33.3 31.3 35.5 31.9 ...
#>  $ Gene8 : num  21.5 22.9 24.7 24 21.6 ...
#>  $ Gene9 : chr  "35.037" "36.36" "35.946" "36.885" ...
#>  $ Gene10: num  26.6 27.1 26.7 27.7 26.8 ...
#>  $ Gene11: num  36.7 34.1 35.4 37.1 35.2 ...
#>  $ Gene12: num  28.2 30.2 29.4 32.5 29.6 ...
#>  $ Gene13: num  28.2 30.5 29.9 33 29.8 ...
#>  $ Gene14: num  27.9 28.5 28.1 30.8 30.9 ...
#>  $ Gene15: num  30 31.2 32 32.1 29.4 ...
#>  $ Gene16: num  21.7 22.5 23.6 24.5 22.7 ...
#>  $ Gene17: num  26.3 27.1 27.6 28.5 27.6 ...
#>  $ Gene18: num  28.2 28.5 29.6 30.2 26.6 ...
#>  $ Gene19: chr  "28.263" "27.358" "28.867" "29.951" ...
#>  $ Gene20: num  32.7 32.9 31 35.9 32 ...

# The imported table is now transformed to a long-format structure. The "X" column is unnecessary
         and is removed. All variables also are converted to a character to unify the class of
         variables.
library(tidyverse)
data.Ct <- data.Ct.wide %>%
             select(-X) %>%
             mutate(across(everything(), as.character)) %>%
             pivot_longer(cols = -c(Group, Sample), names_to = "Gene", values_to = "Ct")
str(data.Ct)
#> tibble [1,216 × 4] (S3: tbl_df/tbl/data.frame)
#>  $ Group : chr [1:1216] "Disease" "Disease" "Disease" "Disease" ...
#>  $ Sample: chr [1:1216] "Disease1" "Disease1" "Disease1" "Disease1" ...
#>  $ Gene  : chr [1:1216] "Gene1" "Gene2" "Gene3" "Gene4" ...
#>  $ Ct    : chr [1:1216] "32.563" "36.554" "31.334" "23.415" ...
```

NOTE: At this stage, the Ct values do not have to be numeric.

NOTE: Data can also be imported to R using user's own code, but the final object must be a data frame and contain a table with column named "Sample" with sample names, column named "Gene" with gene names, column named "Ct" with raw Ct values, column named "Group" with group names, and optionally column named "Flag" containing flag information (this column should be a class of character or factor).

For package testing purposes, there is also a convenient possibility to use the data object included in the `RQdeltaCT` package, named `data.Ct`:

```
data(data.Ct)
str(data.Ct)
#> 'data.frame':    1288 obs. of  5 variables:
#>  $ Sample: chr  "Disease1" "Disease10" "Disease12" "Disease13" ...
#>  $ Gene  : chr  "Gene1" "Gene1" "Gene1" "Gene1" ...
#>  $ Ct    : chr  "32.563" "34.648" "35.059" "37.135" ...
#>  $ Group : chr  "Disease" "Disease" "Disease" "Disease" ...
#>  $ Flag  : chr  "OK" "OK" "OK" "OK" ...
```

Quality control of raw Ct data

The crucial step of each data analysis is an assessment of the quality and usefulness of the data used to investigate the studied problem. The `RQdeltaCT` package offers two functions for quality control of raw Ct data: `control_Ct_barplot_sample()` (for quality control of samples) and `control_Ct_barplot_gene()` (for quality control of genes). Both functions require specifying quality control criteria to be applied to Ct values, in order to label each Ct value as reliable or not. These functions return numbers of reliable and unreliable Ct values in each sample or each gene, as well as total number of Ct values obtained from each sample and each gene. These results are presented graphically on barplots. The obtained results are useful for inspecting the analysed data in order to identify samples and genes that should be considered to be removed from the data (based on applied reliability criteria).

Three selection criteria can be set for these functions:

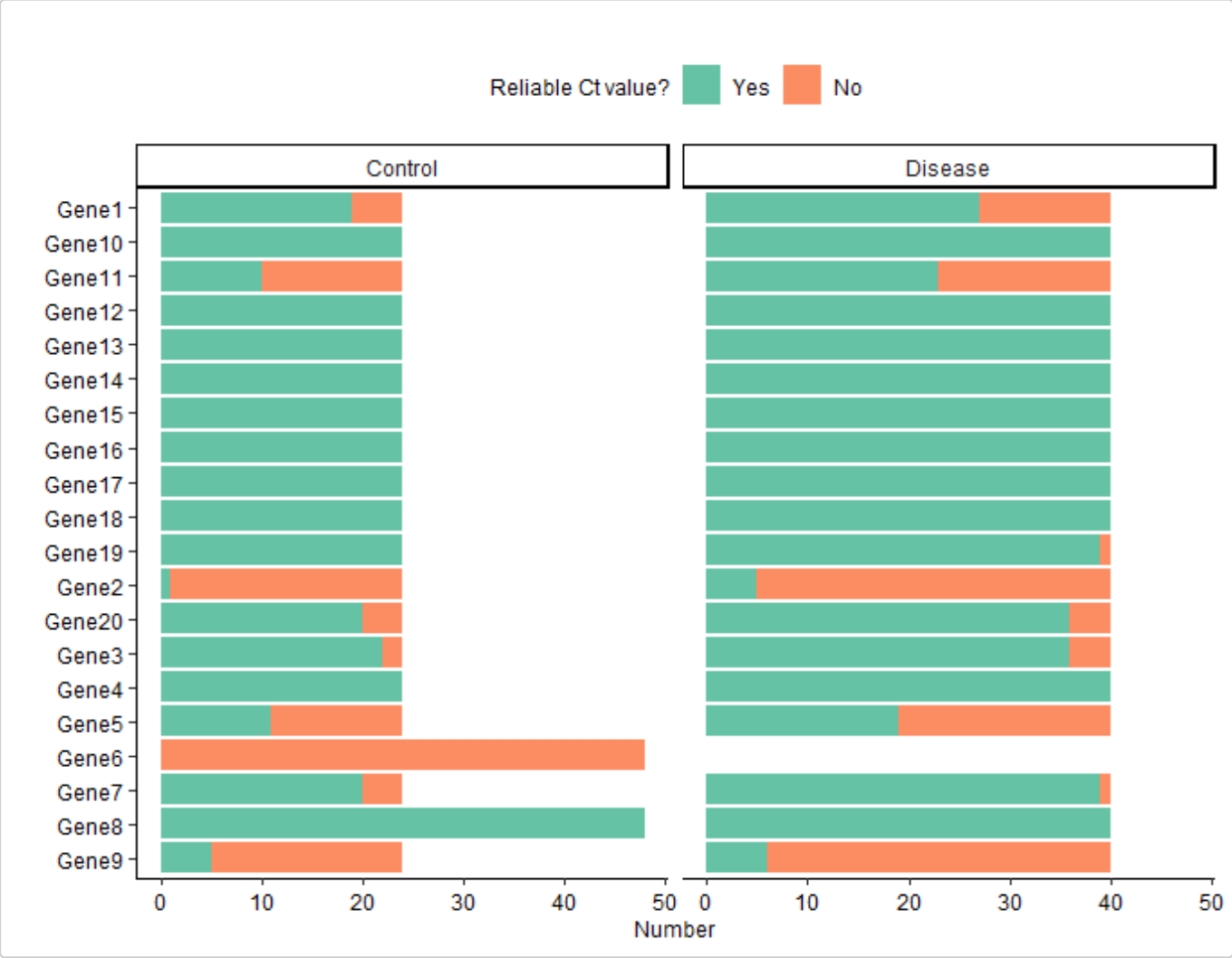
- a flag used for undetermined Ct values. Default to “Undetermined”.
- a maximum of Ct value allowed. Default to 35.
- a flag used in the Flag column for values which are unreliable. Default to “Undetermined”.

NOTE: This function does not perform data filtering, but only report numbers of Ct values labelled as reliable or not and presents them graphically.

An example of using these functions is provided below:

```
sample.Ct.control <- control_Ct_barplot_sample(data = data.Ct,
                                               flag.Ct = "Undetermined",
                                               maxCt = 35,
                                               flag = c("Undetermined"),
                                               axis.title.size = 9,
                                               axis.text.size = 7,
                                               plot.title.size = 9,
                                               legend.title.size = 9,
                                               legend.text.size = 9)
#> Returned table contains the numbers of Ct values labelled as reliable or not in each sample, as
        well as the fraction of unreliable Ct values in each sample.
```


```
gene.Ct.control <- control_Ct_barplot_gene(data = data.Ct,
                                           flag.Ct = "Undetermined",
                                           maxCt = 35,
                                           flag = c("Undetermined"),
                                           axis.title.size = 9,
                                           axis.text.size = 9,
                                           plot.title.size = 9,
                                           legend.title.size = 9,
                                           legend.text.size = 9)
#> Returned table contains the numbers of Ct values labelled as reliable or not in each gene, as
        well as the fraction of unreliable Ct values in each gene.
```



Created plots are displayed on the graphic device, and short information about the returned tables appears. Returned objects are lists that contain two elements: an object with plot and a table with numbers of Ct values labelled as reliable (Yes) and unreliable (No), as well as fraction of unreliable Ct values in each gene. To easily identify low quality samples or genes, tables are sorted by descending numbers of unreliable values. To access returned tables, the second element of returned objects should be called:

```
head(sample.Ct.control[[2]])
#> # A tibble: 6 × 4
#>   Sample    Not.reliable Reliable Not.reliable.fraction
#>   <fct>            <int>    <int>                 <dbl>
#> 1 Control08            9       13                 0.409
#> 2 Control16            9       13                 0.409
#> 3 Control22            9       13                 0.409
#> 4 Control13            8       14                 0.364
#> 5 Control05            7       15                 0.318
#> 6 Control20            7       15                 0.318
```

```
head(gene.Ct.control[[2]])
#> # A tibble: 6 × 5
#>   Gene  Group   Not.reliable Reliable Not.reliable.fraction
#>   <fct> <fct>          <int>    <int>                 <dbl>
#> 1 Gene6 Control           48        0                 1
#> 2 Gene2 Disease           35        5                 0.875
#> 3 Gene9 Disease           34        6                 0.85
#> 4 Gene2 Control           23        1                 0.958
```

```
#> 5 Gene5 Disease           21       19                0.525
#> 6 Gene9 Control           19        5                0.792
```

Visual inspection of returned plots and obtained tables gives a clear image of data quality. The results obtained in the examples show that the Disease group contains more samples than the Control group. Some of the samples have more Ct values (more technical replicates) than other samples. Furthermore, in all samples, the majority of Ct values are reliable.

Regarding genes, Gene6 and Gene8 were investigated in duplicates, other genes have single Ct values. Furthermore, Gene6 was analysed only in the Control group and has all values labeled as unreliable; therefore, it is obvious that this gene should be excluded from the analysis. Some other genes also have many unreliable Ct values (e.g. Gene2, Gene9) and maybe should be considered to be removed from the data.

In some situations, a unified fraction of unreliable data need to be established and used to make decision which samples or genes should be excluded from the analysis. It can be done using the following code, in which the second element of object returned by the `control_Ct_barplot_sample()` and `control_Ct_barplot_gene()` functions can be used directly, and a vector with samples or genes for which the fraction of unreliable Ct values is higher than a specified threshold is received:

```
# Finding samples with more than half of the unreliable Ct values.
low.quality.samples <- filter(sample.Ct.control[[2]], Not.reliable.fraction > 0.5)$Sample
low.quality.samples <- as.vector(low.quality.samples)
low.quality.samples
#> character(0)

# Finding genes with more than half of the unreliable Ct values in given group.
low.quality.genes <- filter(gene.Ct.control[[2]], Not.reliable.fraction > 0.5 & Group ==
         "Disease")$Gene
low.quality.genes <- as.vector(low.quality.genes)
low.quality.genes
#> [1] "Gene2" "Gene9" "Gene5"
```

In the above examples, there is no sample with more than half of the unreliable data. Furthermore, this criterion was met by 5 genes (Gene2, Gene5, Gene6, Gene9, and Gene11); therefore, these genes will be removed from the data in the next step of analysis.

Filtering of raw Ct data

When reliability criteria are finally established for Ct values, and some samples or genes are decided to be excluded from the analysis after quality control of the data, the data with raw Ct values can be filtered using the `filter_Ct()` function.

As a filtering criteria, a flag used for undetermined Ct values, a maximum of Ct threshold, and a flag used in Flag column can be applied. Furthermore, vectors with samples, genes, and groups to be removed can also be specified:

```
# Objects returned from the `low_quality_samples()` and `low_quality_genes()`functions can be used
         directly:
data.CtF <- filter_Ct(data = data.Ct,
                      flag.Ct = "Undetermined",
                      maxCt = 35,
                      flag = c("Undetermined"),
                      remove.Gene = low.quality.genes,
```

```
                        remove.Sample = low.quality.samples)

# Check dimensions of data before and after filtering:
dim(data.Ct)
#> [1] 1288    5
dim(data.CtF)
#> [1] 979   5
```

NOTE: If data contain more than two groups, it is good practice to remove groups that are out of comparison; however, a majority of other functions can deal with more groups unless only two groups are indicated to be given in function's parameters.

Collapsing technical replicates and imputation of missing data - `make_Ct_ready()` function

In the next step, filtered Ct data can be subjected to collapsing of technical replicates and (optional) data imputation by means within groups using the `make_Ct_ready()` function. The term 'technical replicates' means observations with the same group name, gene name, and sample name. In the scenario when data contain technical replicates but they should not be collapsed, these technical replicates must be distinguished by different sample names, e.g. Sample1_1, Sample1_2, Sample1_3, etc.

The parameter `imput.by.mean.within.groups` can be used to control data imputation. If it is set to TRUE, imputation will be done, otherwise missing values will be left in the data. For a better view of the amount of missing values in the data, the information about the number and percentage of missing values is displayed automatically:

```
# Without imputation:
data.CtF.ready <- make_Ct_ready(data = data.CtF,
                                imput.by.mean.within.groups = FALSE)
#> The data contain 69 missing values that constitute 6.73828 percent of the total data.
# A part of the data with missing values:
as.data.frame(data.CtF.ready)[19:25,]
#>      Group    Sample  Gene1 Gene10 Gene11 Gene12 Gene13 Gene14 Gene15 Gene16
#> 19 Control Control26 34.506 26.266     NA 31.305 30.384 28.956 31.174 22.844
#> 20 Control Control05     NA 28.339     NA 31.200 30.498 28.562 31.068 21.863
#> 21 Control Control08     NA 30.892     NA 32.381 33.224 31.643 33.485 25.132
#> 22 Control Control13     NA 28.976     NA 32.976 33.340 29.684 32.600 24.812
#> 23 Control Control16     NA 30.689     NA 33.656 32.489 30.335 32.432 24.516
#> 24 Control Control22     NA 26.988     NA 29.846 31.564 30.587 29.176 22.972
#> 25 Disease  Disease1 32.563 26.552     NA 28.179 28.227 27.905 30.048 21.691
#>    Gene17 Gene18 Gene19 Gene20  Gene3  Gene4  Gene7   Gene8
#> 19 28.401 29.012 26.911 33.386 34.256 23.801 31.987 22.5725
#> 20 26.798 29.582 28.194 33.680 33.657 24.123 31.019 23.3080
#> 21 30.133 31.399 29.908     NA 34.853 26.228     NA 26.4465
#> 22 28.904 30.251 28.085     NA 32.600 25.186 34.338 24.9500
#> 23 29.772 30.991 28.964     NA 33.432 25.435     NA 25.4190
#> 24 27.208 28.941 28.143     NA 34.886 25.056     NA 23.0315
#> 25 26.272 28.165 28.263 32.725 31.334 23.415 32.789 21.4550

# With imputation:
data.CtF.ready <- make_Ct_ready(data = data.CtF,
                                imput.by.mean.within.groups = TRUE)
```

```
#> The data contain 69 missing values that constitute 6.73828 percent of the total data.
#>  Missing values were imputed using means within compared groups.
# Missing values were imputed:
as.data.frame(data.CtF.ready)[19:25,]
#>      Group    Sample    Gene1 Gene10   Gene11 Gene12 Gene13 Gene14 Gene15
#> 19 Control Control26 34.50600 26.266 33.70620 31.305 30.384 28.956 31.174
#> 20 Control Control05 33.01921 28.339 33.70620 31.200 30.498 28.562 31.068
#> 21 Control Control08 33.01921 30.892 33.70620 32.381 33.224 31.643 33.485
#> 22 Control Control13 33.01921 28.976 33.70620 32.976 33.340 29.684 32.600
#> 23 Control Control16 33.01921 30.689 33.70620 33.656 32.489 30.335 32.432
#> 24 Control Control22 33.01921 26.988 33.70620 29.846 31.564 30.587 29.176
#> 25 Disease  Disease1 32.56300 26.552 32.70926 28.179 28.227 27.905 30.048
#>    Gene16 Gene17 Gene18 Gene19   Gene20  Gene3  Gene4    Gene7   Gene8
#> 19 22.844 28.401 29.012 26.911 33.38600 34.256 23.801 31.98700 22.5725
#> 20 21.863 26.798 29.582 28.194 33.68000 33.657 24.123 31.01900 23.3080
#> 21 25.132 30.133 31.399 29.908 32.88615 34.853 26.228 31.83495 26.4465
#> 22 24.812 28.904 30.251 28.085 32.88615 32.600 25.186 34.33800 24.9500
#> 23 24.516 29.772 30.991 28.964 32.88615 33.432 25.435 31.83495 25.4190
#> 24 22.972 27.208 28.941 28.143 32.88615 34.886 25.056 31.83495 23.0315
#> 25 21.691 26.272 28.165 28.263 32.72500 31.334 23.415 32.78900 21.4550
```

NOTE: The data imputation process can significantly influence data; therefore, no default value was set to the `imput.by.mean.within.groups` parameter to force the specification by the user. If there are missing data for a certain gene in the entire group, they will not be imputed and will remain NA.

In general, a majority of functions in `RQdeltaCT` package can deal with missing data; however, some used methods are sensitive to missing data, including VIF calculating in `find_ref_gene()` (see Reference gene selection section) and PCA analysis (see PCA analysis section).

NOTE: The `make_Ct_ready()` function should be used even if the collapsing of technical replicates and data imputation is not required, because this function also prepares the data structure to fit to further functions.

Relative quantification: 2^{-Ct} method

This method is useful for testing genes for their suitability to be an endogenous control by investigating whether the studied experimental condition significantly influences their expression (see the Introduction section).

In this method, raw Ct values are transformed using thw 2^{-Ct} formula, summarised by means in the compared study conditions, and a ratio of means (fold change) is calculated for the experimental condition. All these processes are carried out by two functions:

- `exp_Ct_dCt()` - this function performs a Ct data transformation using the 2^{-Ct} formula. Before further processing, transformed Ct data should be subjected to quality control using functions and methods described in Quality control and filtering of transformed Ct data section.
- `RQ_exp_Ct_dCt()` - this function performs relative quantification by:
  - calculation of means (returned in columns with the "_mean" pattern) and standard deviations (returned in columns with the "_sd" pattern) of transformed Ct values of genes analysed in the compared groups.
  - normality testing (Shapiro_Wilk test) of transformed Ct values of analysed genes within compared groups, and returned p values are stored in columns with the "_norm_p" pattern.
  - calculation of the fold change values for each gene by dividing the mean of transformed Ct values in the study group by the mean of transformed Ct values in the reference group. Fold change values are returned in the "FCh" column.

- statistical testing of differences in transformed Ct values between the study group and the reference group. Student's t test and Mann-Whitney U test are implemented and resulting statistics (in column with the "_test_stat" pattern) and p values (in column with the "_test_p" pattern) are returned.

The `RQ_exp_Ct_dCt()` function gives a choice if normality and statistical tests should be done (`do.tests` parameter). In the situations where compared groups contain less than three samples (at least three samples are required for testing), `do.test` should be set to FALSE to avoid error.

```
data.Ct.exp <- exp_Ct_dCt(data = data.CtF.ready)
library(coin)
RQ.Ct.exp <- RQ_exp_Ct_dCt(data = data.Ct.exp,
                           do.tests = TRUE,
                           group.study = "Disease",
                           group.ref = "Control")

# Obtained table can be sorted by, e.g. p values from the Mann-Whitney U test:
head(as.data.frame(arrange(RQ.Ct.exp, MW_test_p)))
#>     Gene Control_mean Disease_mean   Control_sd   Disease_sd Control_norm_p
#> 1  Gene1 2.905106e-10 1.250993e-09 7.777413e-10 1.356761e-09   1.464914e-09
#> 2 Gene11 8.688525e-11 1.816876e-10 8.522952e-11 1.535215e-10   1.461879e-08
#> 3 Gene19 8.325615e-09 3.677473e-09 7.916086e-09 3.088826e-09   1.124554e-04
#> 4 Gene20 1.972469e-10 4.343676e-10 2.575527e-10 5.366581e-10   2.783597e-07
#> 5 Gene12 4.948244e-09 1.509389e-07 9.904616e-09 6.186651e-07   1.053129e-07
#> 6 Gene13 1.922705e-09 5.061451e-09 3.225020e-09 7.609794e-09   1.912822e-07
#>   Disease_norm_p        FCh     t_test_p t_test_stat    MW_test_p MW_test_stat
#> 1   2.103899e-06  4.3061856 0.0006357395   -3.598966 7.528138e-06    -4.478182
#> 2   3.309215e-08  2.0911219 0.0023421923   -3.174414 7.662081e-05    -3.954731
#> 3   1.086664e-05  0.4417059 0.0103674454    2.753545 2.178749e-03     3.064719
#> 4   1.342974e-07  2.2021520 0.0207506215   -2.375494 7.427546e-03    -2.677041
#> 5   5.430539e-13 30.5035252 0.1437039659   -1.492130 3.159770e-02    -2.149463
#> 6   3.937886e-08  2.6324631 0.0258248708   -2.288503 7.142346e-02    -1.802776
```

NOTE: Both functions also can be used to analyse transformed delta Ct data (see Relative quantification: 2^{-dCt} method).

Reference gene selection

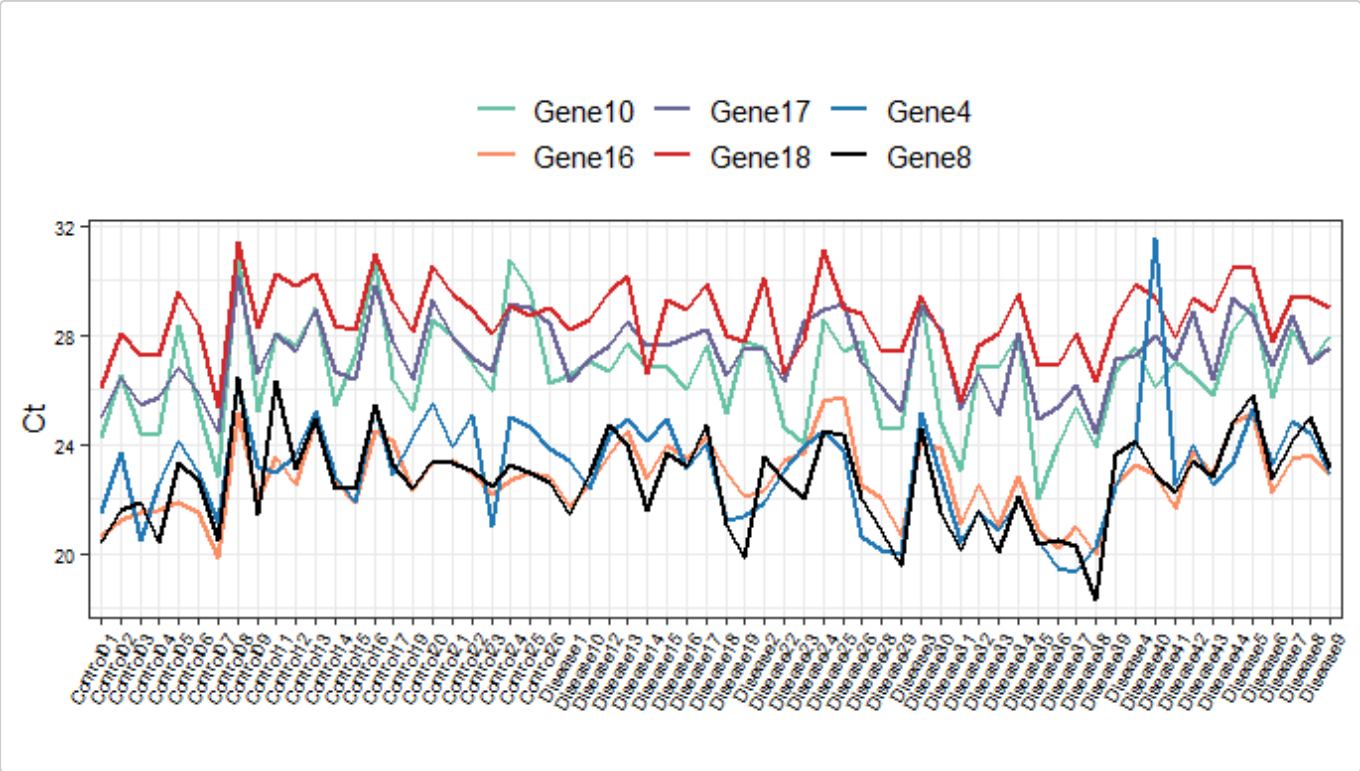
Ideally, the reference gene should have an identical expression level in all samples, but in many situations it is not possible to achieve this, especially when biological replicates are analysed. Therefore, differences between samples are allowed, but the variance should be as low as possible, and it is also recommended that Ct values would not be very low (below 15) or very high (above 30) Kozera and Rapacz 2013.

The `RQdeltaCT` package includes `find_ref_gene()` function that can be used to select the best reference gene for normalisation. This function calculates descriptive statistics such as minimum, maximum, standard deviation, and variance, as well as stability scores calculated using the NormFinder (Article) and geNorm (Article) algorithms. Ct values are also presented on a line plot.

NormFinder scores are computed using internal `RQdeltaCT::norm_finder()` function working on the code adapted from the original NormFinder code. To calculate NormFinder scores, at least two samples must be present in each group. For the geNorm score, the `geNorm()` function of the `ctrlGene` package is used. The `find_ref_gene()` function allows one to choose which of these algorithms should be done by setting the logical parameters: `norm.finder.score` and `genorm.score`.

The created plot is displayed on the graphic device. The returned object is a list that contains two elements: an object with plot and a table with results. In the example below, six genes are tested for suitability to be a reference gene:

```
library(ctrlGene)
# Remember that the number of colors in col parameter should be equal to the number of tested
        genes:
ref <- find_ref_gene(data = data.CtF.ready,
                     groups = c("Disease","Control"),
                     candidates = c("Gene4", "Gene8","Gene10","Gene16","Gene17", "Gene18"),
                     col = c("#66c2a5", "#fc8d62","#6A6599", "#D62728", "#1F77B4", "black"),
                     angle = 60,
                     axis.text.size = 7,
                     norm.finder.score = TRUE,
                     genorm.score = TRUE)
```




```
ref[[2]]
#>            Gene    min      max       sd      var NormFinder_score geNorm_score
#> 1        Gene10 22.043 30.8920 1.894611 3.589551             0.31    1.0511715
#> 2        Gene16 19.801 25.7370 1.359134 1.847245             0.34           NA
#> 3        Gene17 24.436 30.1330 1.370355 1.877874             0.19           NA
#> 4        Gene18 25.323 31.3990 1.329008 1.766262             0.14    0.8127723
#> 5         Gene4 19.295 31.5080 1.998908 3.995635             0.33    1.2335325
#> 6         Gene8 18.314 26.4465 1.724890 2.975244             0.22    0.9001852
#> 7 Gene16-Gene17     NA      NA       NA       NA               NA    0.6930721
```

Among tested genes, Gene8, Gene16, Gene17, and Gene18 seem to have the best characteristics to be a reference gene (they have low variance, high VIF values, and low NormFinder and geNorm scores).

Relative quantification: 2^{-dCt} method

This method is used in studies where samples should be analysed as individual data points, e.g. in analysis of biological replicates. In this method, Ct values are normalised by the endogenous control gene by subtracting the Ct value of the endogenous control from the Ct value of the gene of interest, in the same sample. Obtained delta Ct (dCt) values are subsequently transformed using the 2^{-dCt} formula, summarised by means in the compared study groups, and a ratio of means (fold change) is calculated for the study group (see Introduction section).

The whole process can be done using three functions:

- `delta_Ct` - this function calculates delta Ct (dCt) values by subtracting Ct values of reference gene from Ct values of gene of interest across all samples.
- `exp_Ct_dCt()` - this function performs a transformation of dCt data using the 2^{-dCt} formula. Before further processing, transformed dCt data should be subjected to quality control using functions and methods described in Quality control and filtering of transformed Ct data section.
- `RQ_exp_Ct_dCt()` - this function performs:
  - calculation of means (returned in columns with the "_mean" pattern) and standard deviations (returned in columns with the "_sd" pattern) of transformed dCt values of genes analysed in the compared groups.
  - normality testing (Shapiro_Wilk test) of transformed dCt values of analysed genes in compared groups and returned p values are stored in columns with the "_norm_p" pattern.
  - calculation of fold change values for each gene by dividing the mean of transformed dCt values in the study group by the mean of transformed dCt values in the reference group. Fold change values are returned in the "FCh" column.
  - statistical testing of differences in transformed Ct values between the study group and the reference group. Student's t test and Mann-Whitney U test are implemented and the resulting statistics (in column with the "_test_stat" pattern) and p values (in column with the "_test_p" pattern) are returned. If compared groups contain less than three samples, normality and statistical tests are not possible to perform (the `do.test` parameter should be set to FALSE to avoid error).

```
data.dCt <- delta_Ct(data = data.CtF.ready,
                     ref = "Gene8")
data.dCt.exp <- exp_Ct_dCt(data = data.dCt)
library(coin)
RQ.dCt.exp <- RQ_exp_Ct_dCt(data = data.dCt.exp,
                            do.tests = TRUE,
                            group.study = "Disease",
                            group.ref = "Control")

# Obtained table can be sorted by, e.g. p values from the Mann-Whitney U test:
head(as.data.frame(arrange(RQ.dCt.exp, MW_test_p)))
#>     Gene Control_mean Disease_mean  Control_sd  Disease_sd Control_norm_p
#> 1  Gene1  0.001779634  0.007572040 0.002442710 0.007962192   3.435812e-06
#> 2 Gene19  0.058796934  0.024802219 0.060456391 0.027097216   3.400562e-07
#> 3 Gene13  0.013651779  0.013343277 0.028406905 0.008790133   5.964156e-09
#> 4 Gene16  1.509777018  0.967490766 1.244618540 0.609112278   1.086815e-06
#> 5 Gene12  0.016030664  0.340252219 0.018392782 1.013461602   1.391742e-05
#> 6  Gene7  0.003424749  0.001911581 0.004925526 0.001866461   1.322123e-07
#>   Disease_norm_p        FCh     t_test_p t_test_stat    MW_test_p MW_test_stat
#> 1   1.175080e-05  4.2548290 8.483229e-05 -4.27774491 5.136855e-05    -4.049311
#> 2   4.020200e-08  0.4218284 1.450713e-02  2.60232741 5.449915e-05     4.035444
#> 3   2.002333e-03  0.9774021 9.591380e-01  0.05173793 1.009811e-02    -2.572452
#> 4   4.020298e-03  0.6408170 5.517804e-02  1.99590948 1.255492e-02     2.496151
#> 5   6.309057e-12 21.2250864 4.998438e-02 -2.02276492 1.410617e-02    -2.454548
#> 6   3.737597e-07  0.5581668 1.602119e-01  1.44408971 6.717362e-02     1.830511
```

Relative quantification: 2^{-ddCt} method

Similarly to the 2^{-dCt} method, in the 2^{-ddCt} method Ct values are normalised by the endogenous control gene, obtaining dCt values. Subsequently, dCt values are summarised by means in the compared groups, and for each gene, the obtained mean in the control group is subtracted from the mean in the study group, giving the delta delta Ct (ddCt) value. Finally, the ddCt values are transformed using the 2^{-ddCt} formula to obtain the fold change values. This method is recommended for analysis of technical replicates (see the Introduction section).

The whole process can be done using three functions:

- `delta_Ct` - this function calculates delta Ct (dCt) values by subtracting the Ct values of the reference gene from the Ct values of gene of interest across all samples. Before further processing, obtained dCt data should be subjected to the quality control using functions and methods described in Quality control and filtering of transformed Ct data section.
- `RQ_ddCt()` - this function performs:
  - calculation of means (returned in columns with the "_mean" pattern) and standard deviations (returned in columns with the "_sd" pattern) of delta Ct values of the analyzed genes in the compared groups.
  - normality testing (Shapiro_Wilk test) of delta Ct values of the analyzed genes in the compared groups and returned p values are stored in columns with the "_norm_p" pattern.
  - calculation of differences in the mean delta Ct values of genes between compared groups, returned in "ddCt" column,
  - calculation of fold change values (returned in "FCh" column) for each gene by transforming the ddCt values using the 2^{-ddCt} formula.
  - statistical testing of differences between the compared groups. Student's t test and Mann-Whitney U test are implemented and the resulted statistics (in column with the "_test_stat" pattern) and p values (in column with the "_test_p" pattern) are returned. If compared groups contain less than three samples, normality and statistical tests are not possible to perform and `do.test` parameter should be set to FALSE to avoid error.

```
data.dCt <- delta_Ct(data = data.CtF.ready,
                     ref = "Gene8")
library(coin)
RQ.ddCt <- RQ_ddCt(data = data.dCt,
                   group.study = "Disease",
                   group.ref = "Control",
                   do.tests = TRUE)

# Obtained table can be sorted by, e.g. p values from the Mann-Whitney U test:
head(as.data.frame(arrange(RQ.ddCt, MW_test_p)))
#>     Gene Control_mean Disease_mean Control_sd Disease_sd Control_norm_p
#> 1  Gene1    10.117002     7.959426  1.6881153  1.8871311    0.426642224
#> 2 Gene19     4.522833     5.938756  1.1662848  1.3467583    0.007240964
#> 3 Gene13     7.216583     6.565200  1.4577193  1.0627016    0.038235161
#> 4 Gene16    -0.329125     0.338025  0.8145069  0.9570775    0.213259767
#> 5 Gene12     6.675667     5.058075  1.4070518  2.9513180    0.172997230
#> 6  Gene7     8.932742     9.536064  1.4167152  1.2052427    0.614472418
#>   Disease_norm_p       ddCt       FCh     t_test_p t_test_stat    MW_test_p
#> 1     0.17446884 -2.1575763 4.4616467 1.690067e-05    4.733411 5.136855e-05
#> 2     0.56682258  1.4159231 0.3747699 4.581993e-05   -4.432997 5.449915e-05
#> 3     0.17567779 -0.6513833 1.5706735 6.426170e-02    1.906189 1.009811e-02
#> 4     0.57399623  0.6671500 0.6297495 4.445141e-03   -2.967530 1.255492e-02
```

```
#> 5     0.04414803 -1.6175917 3.0686235 4.508310e-03    2.952083 1.410617e-02
#> 6     0.61294050  0.6033224 0.6582363 8.872007e-02   -1.742051 6.717362e-02
#>   MW_test_stat
#> 1     4.049311
#> 2    -4.035444
#> 3     2.572452
#> 4    -2.496151
#> 5     2.454548
#> 6    -1.830511
```

Quality control and filtering of transformed Ct data

Data intended for relative quantification should be subjected to quality assessment. The main purpose is to identify outlier samples that could introduce bias into the results. The `RQdeltaCT` package offers several functions which facilitate finding outlier samples in data by implementing distribution analysis (`control_boxplot_sample()` function), hierarchical clustering (`control_cluster_sample()` function) and principal component analysis PCA (`control_pca_sample()` function). However, corresponding functions for genes are also provided to evaluate similarities and differences between expression of analysed genes (`control_boxplot_gene()`, `control_cluster_gene()` and `control_pca_gene()` functions).

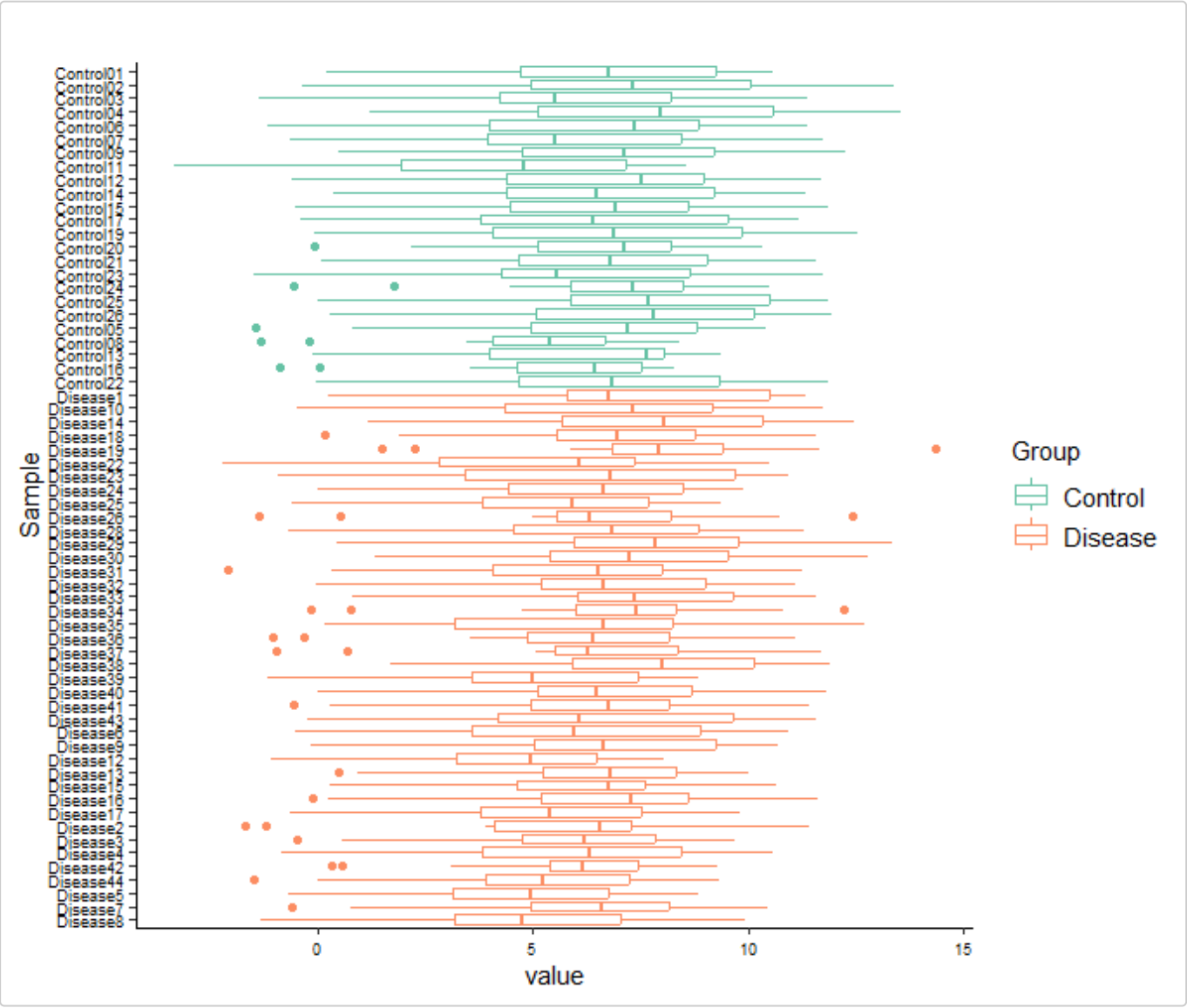
The abovementioned data quality control functions are designed to be directly applied to data objects returned from the `make_Ct_ready()`, `exp_Ct_dCt()` and `delta_Ct()` functions (see The summary of standard workflow, including function names and data flow).

Analysis of data distribution

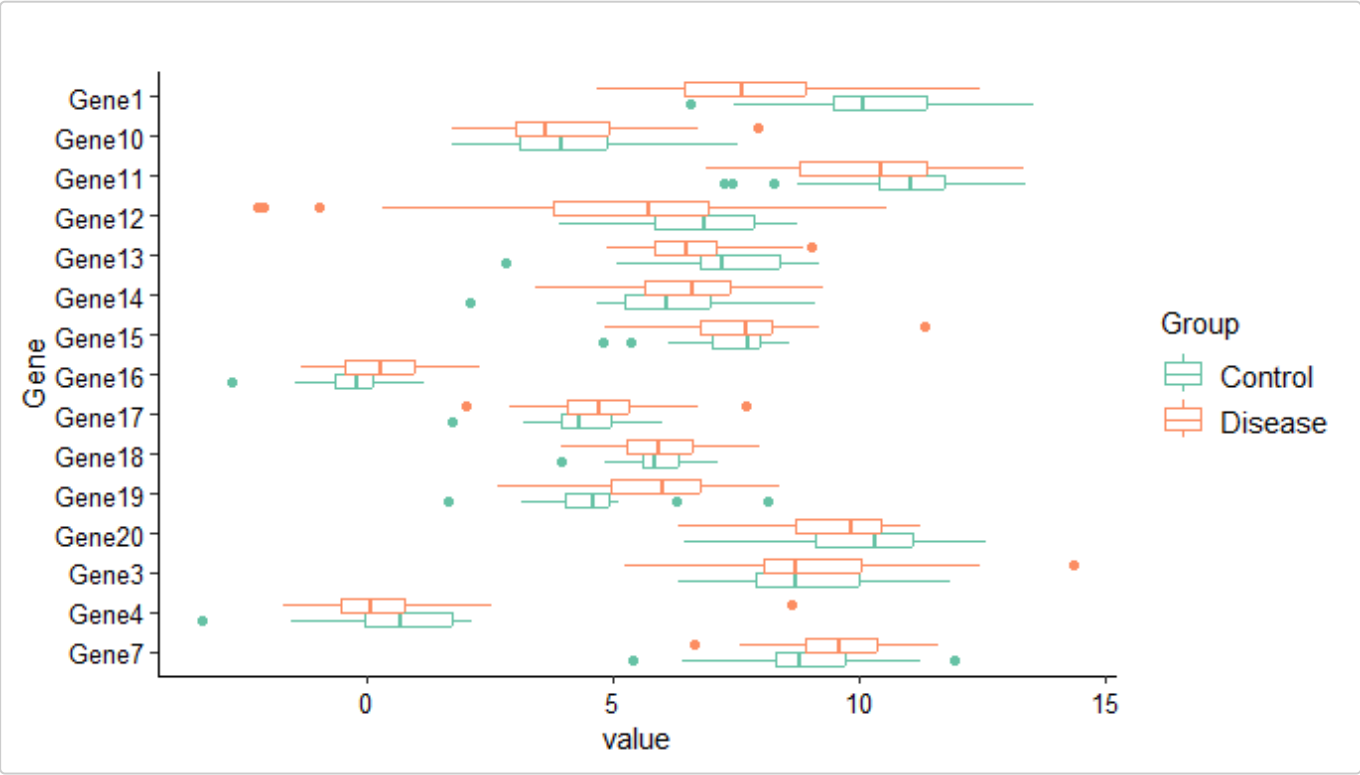
One of the ways to gain insight into data distribution is drawing boxplot that show several statistics, such as median (labelled inside box), first and third quartiles (ranged by box), extreme point in interquartile range (ranged by whiskers), and more distant data as separated points. In the `RQdeltaCT` package, the `control_boxplot_sample()` and `control_boxplot_gene()` functions are included to draw boxplots that present the distribution of samples and genes, respectively.

These functions return objects with a plot. The created plots are also displayed on the graphic device.

```
control_boxplot_sample <- control_boxplot_sample(data = data.dCt,
                                                 axis.text.size = 7)
```




```
control_boxplot_gene <- control_boxplot_gene(data = data.dCt,
                                             by.group = TRUE,
                                             axis.text.size = 10)
```

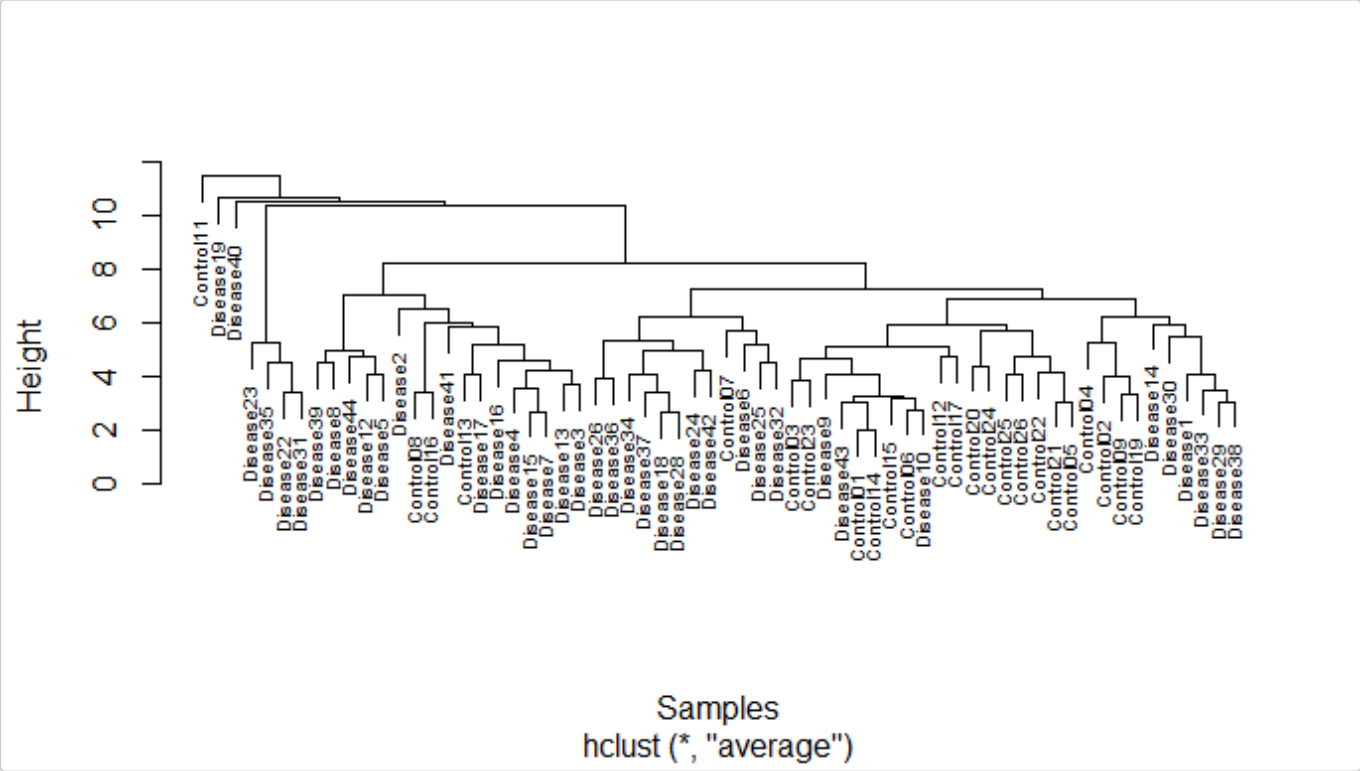


NOTE: If missing values are present in the data, they will be automatically removed with warning.

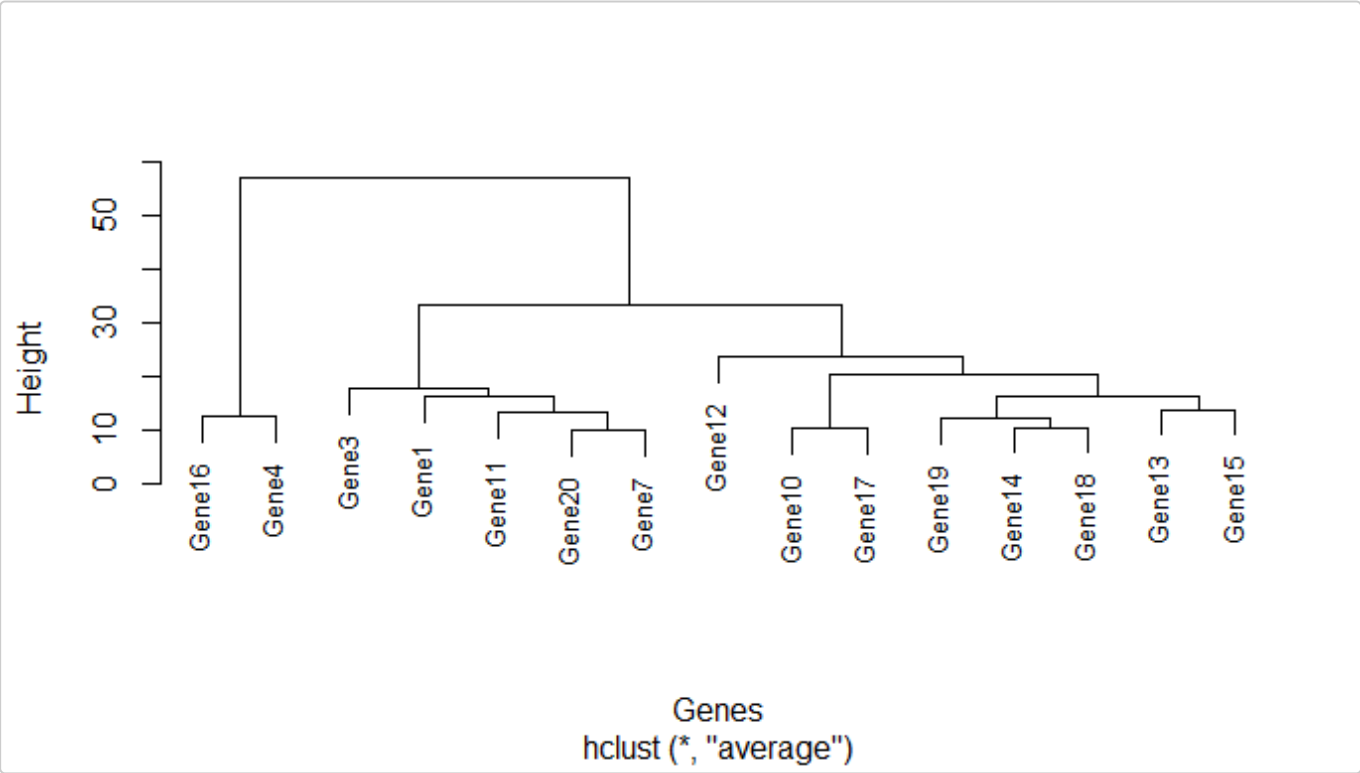
Hierarchical clustering

Hierarchical clustering is a convenient method to investigate similarities between variables. Hierarchical clustering of samples and genes can be done using the `control_cluster_sample()` and `control_cluster_gene()` functions, respectively. These functions allow drawing dendrograms using various methods of distance calculation (e.g. euclidean, canberra) and agglomeration (e.g. complete, average, single). For more details, refer to the functions documentation.

```
control_cluster_sample(data = data.dCt,
                       method.dist = "euclidean",
                       method.clust = "average",
                       label.size = 0.6)
```




```
control_cluster_gene(data = data.dCt,
                     method.dist = "euclidean",
                     method.clust = "average",
                     label.size = 0.8)
```



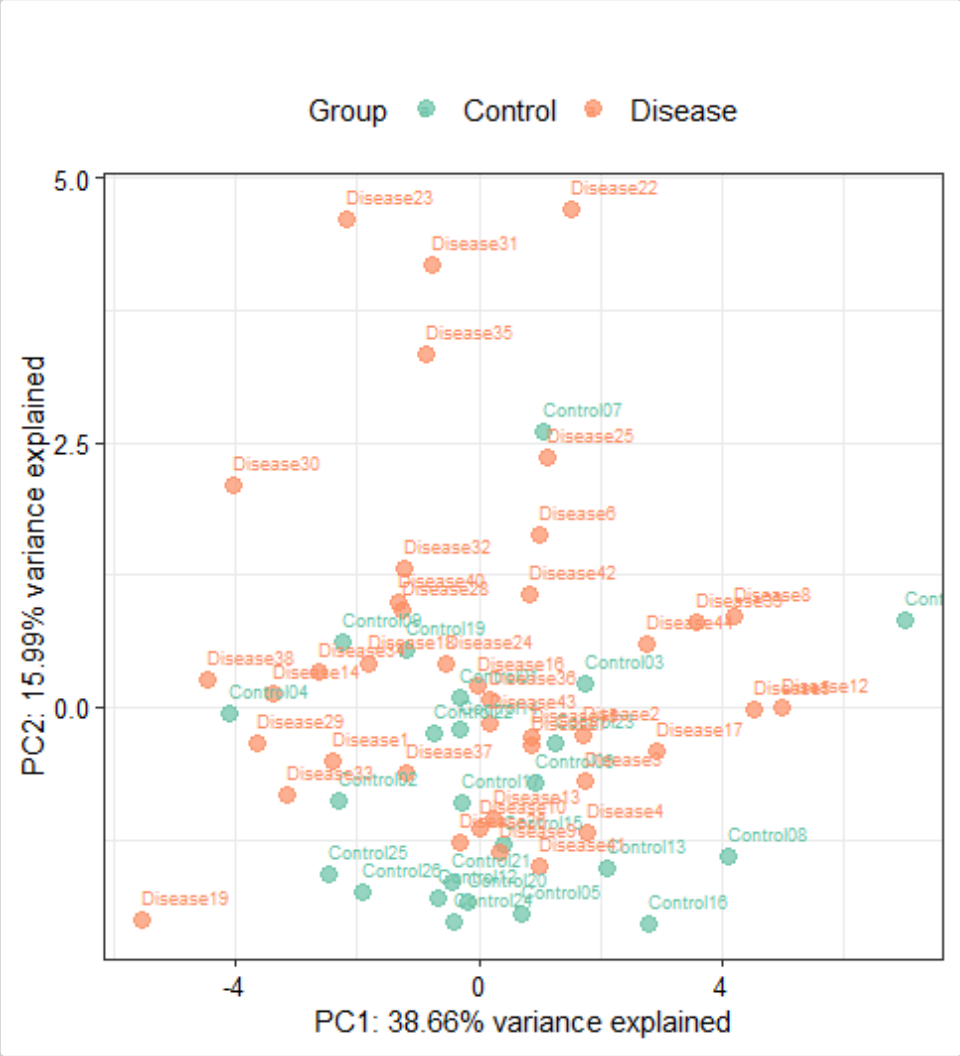
The created plots are displayed on the graphic device.

NOTE: Minimum three samples or genes in data is required for clustering analysis.

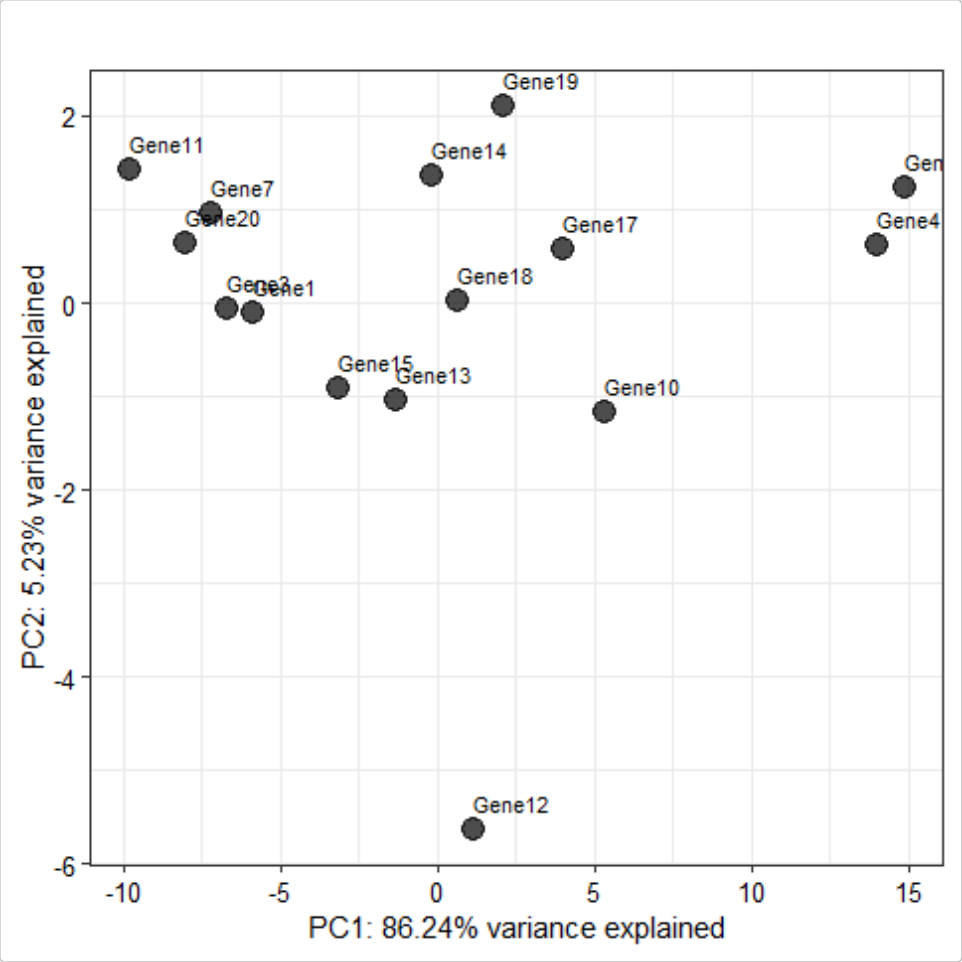
PCA analysis

Principal component analysis (PCA) is a data exploratory method, that is commonly used to investigate similarities between variables based on the first principal components that contain the most information about variance. In the `RQdeltaCT` package, the `control_pca_sample()` and `control_pca_gene()` functions are developed to perform PCA analysis for samples and genes, respectively. These functions return objects with a plot. Created plots are also displayed on the graphic device.

```
control.pca.sample <- control_pca_sample(data = data.dCt,
                                         point.size = 3,
                                         label.size = 2.5,
                                         legend.position = "top")
```




```
control.pca.gene <- control_pca_gene(data = data.dCt)
```



NOTE: PCA algorithm can not deal with missing data (NAs); therefore, variables with NA values are removed before analysis. If at least one NA value occurs in all variables in at least one of the compared group, the analysis can not be done. Imputation of missing data will avoid this issue. Also, a minimum of three samples or genes in the data are required for analysis.

Data filtering after quality control

If any sample or gene was decided to be removed from the data, the `filter_transformed_data()` function can be used for filtering. Similarly to quality control functions, this function can be directly applied to data objects returned from the `make_Ct_ready()`, `exp_Ct_dCt()` and `delta_Ct()` functions (see The summary of standard workflow, including function names and data flow).

```
data.dCtF <- filter_transformed_data(data = data.dCt,
                                     remove.Sample = c("Control11"))
```

Final visualisations

For visualisation of final results, three functions can be used:

- `RQ_plot()` that allows to illustrate fold change values of genes,
- `results_barplot` that show mean and standard deviation values of genes across the compared groups,
- `results_boxplot()` that illustrate the data distribution of genes across the compared groups.

All these functions can be run on all data or on finally selected genes (see `sel.Gene` parameter), and also allow to add customised statistical significance labels to plots using `ggsignif` package. These functions have a large number of parameters, and the user should familiarise with all of them to properly adjust created plots to the user needs.

NOTE: If statistical significance labels should be added to the plot, a vector with labels (e.g., “ns”, “*“,”p = 0.03”) should be provided in the `signif.labels` parameter. There are two important points that must be taking into account when preparing this vector:

- The order of labels should correspond to the order of genes presented on the plot, not the order of genes in the data, which can be different.
- In this vector, due to restrictions of the `ggsignif` package, all values must be different (the same values are not allowed). Thus, if the same labels are needed, they should be distinguishable by adding symmetrically a different number of white spaces, e.g., “ns.”, ” ns. “,” ns. “,” ns. “, etc. (see the examples below).

The `RQ_plot()` function

This function creates a barplot that illustrates fold change values obtained from the analysis, together with an indication of statistical significance. Data returned from the `RQ_exp_Ct_dCt()` and `RQ_ddCt()` functions can be directly applied to this function (see The summary of standard workflow, including function names and data flow).

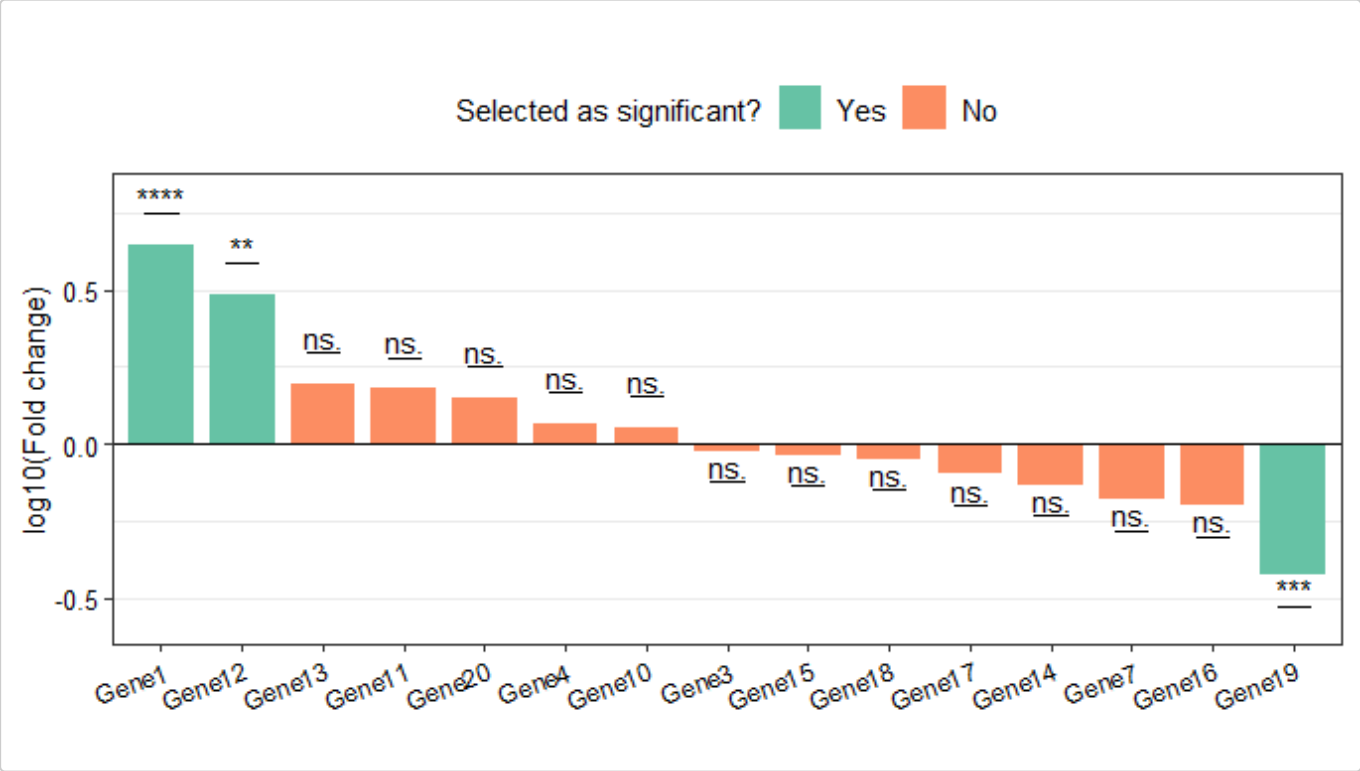
On the barplot, bars of significant genes are distinguished by colors and/or significance labels. The significance of genes can be established by two criteria: p values and (optionally) fold change values. Thresholds for both criteria can be specified. The `RQ_plot()` function offers various options of which p values are used on the plot:

- p values from the Student’s t test (if `mode` = “t”).
- p values from the Mann-Whitney U test (if `mode` = “mw”).
- p values depend on the normality of data (if `mode` = “depends”). If the data in both compared groups were considered derived from normal distribution (p value of Shapiro_Wilk test > 0.05) - p values of Student’s t test will be used, otherwise p values of Mann-Whitney U test will be used.
- external p values provided by the user (if `mode` = “user”). If the user intends to use the p values obtained from the other statistical test, the `mode` parameter should be set to “user”. In this scenario, before running the `RQ_plot()` function, the user should prepare a data.frame object named “user” containing two columns: the first column with gene names and the second column with p values (see example below).

The created plots are displayed on the graphic device. The returned objects are a lists that contain two elements: an object with plot and a table with results.

```
# Variant with p values depending on the normality of the data:
library(ggsignif)
# Specifying vector with significance labels:
signif.labels <- c("****","**","ns."," ns. ","  ns.  ","   ns.   ","    ns.    ","     ns.     ","
         ns.      ","       ns.       ","        ns.        ","         ns.          ","
         ns.          ","           ns.           ","***")
# Genes with p < 0.05 and 2-fold changed expression between compared groups are considered
         significant:
RQ.plot <- RQ_plot(data = RQ.ddCt,
                   use.p = TRUE,
                   mode = "depends",
                   p.threshold = 0.05,
                   use.log10FCh = TRUE,
                   log10FCh.threshold = 0.30103,
                   signif.show = TRUE,
```

```
                signif.labels = signif.labels,
                angle = 20)
```

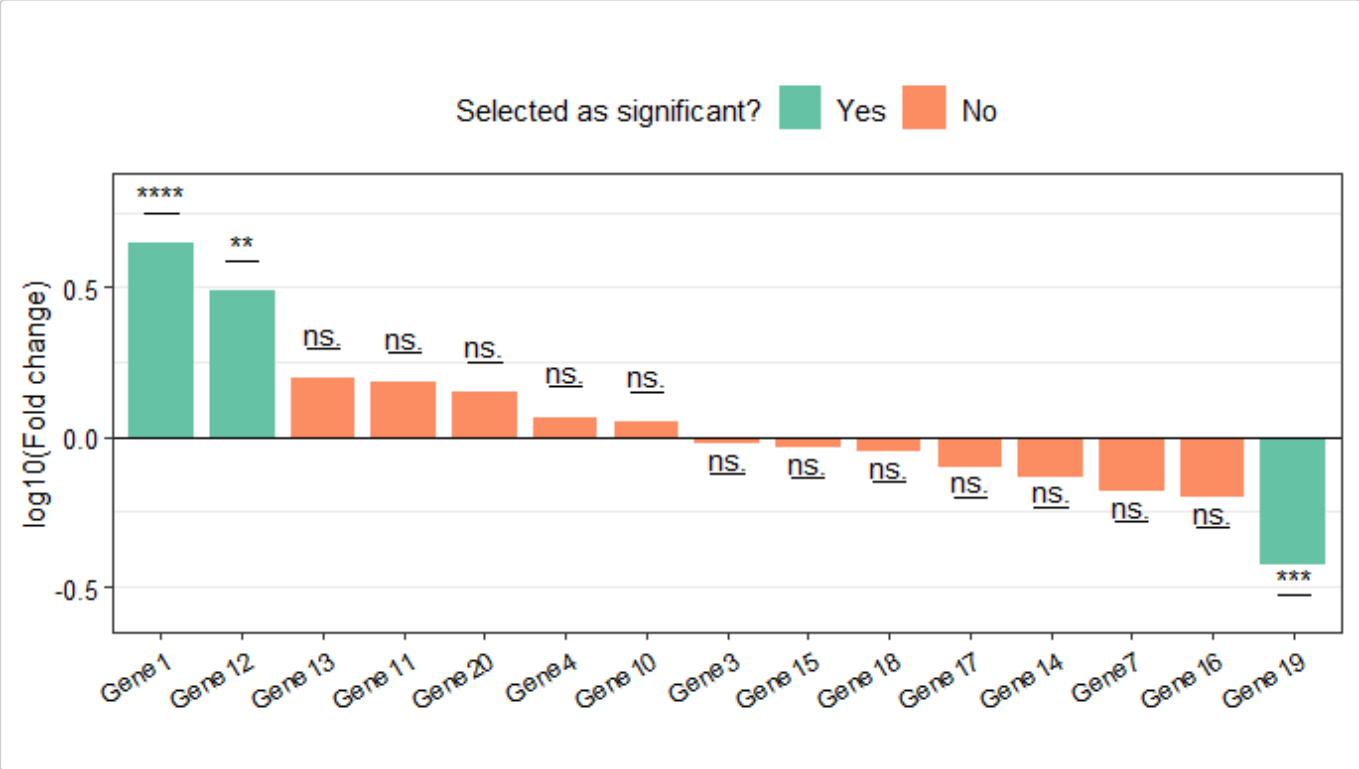



```
# Access the table with results:
head(as.data.frame(RQ.plot[[2]]))
#>     Gene Control_mean Disease_mean Control_sd Disease_sd Control_norm_p
#> 1  Gene1    10.117002     7.959426   1.688115   1.887131     0.42664222
#> 2 Gene12     6.675667     5.058075   1.407052   2.951318     0.17299723
#> 3 Gene13     7.216583     6.565200   1.457719   1.062702     0.03823516
#> 4 Gene11    10.803992    10.199761   1.578909   1.608706     0.05806953
#> 5 Gene20     9.983942     9.481083   1.571200   1.143782     0.68169170
#> 6  Gene4     0.584125     0.360150   1.358816   1.670593     0.03149060
#>   Disease_norm_p       ddCt      FCh     t_test_p t_test_stat    MW_test_p
#> 1   1.744688e-01 -2.1575763 4.461647 1.690067e-05   4.7334112 5.136855e-05
#> 2   4.414803e-02 -1.6175917 3.068624 4.508310e-03   2.9520830 1.410617e-02
#> 3   1.756778e-01 -0.6513833 1.570674 6.426170e-02   1.9061886 1.009811e-02
#> 4   3.043468e-01 -0.6042308 1.520168 1.474652e-01   1.4716714 2.020225e-01
#> 5   2.041532e-02 -0.5028583 1.417018 1.801127e-01   1.3657411 1.271530e-01
#> 6   2.060155e-07 -0.2239750 1.167947 5.610445e-01   0.5847601 8.551052e-02
#>   MW_test_stat test.for.comparison       p.used Selected as significant?
#> 1     4.049311    t.student's.test 1.690067e-05                     Yes
#> 2     2.454548   Mann-Whitney.test 1.410617e-02                     Yes
#> 3     2.572452   Mann-Whitney.test 1.009811e-02                      No
#> 4     1.275810    t.student's.test 1.474652e-01                      No
#> 5     1.525426   Mann-Whitney.test 1.271530e-01                      No
#> 6     1.719571   Mann-Whitney.test 8.551052e-02                      No


# Variant with user p values - the used p values are calculated using the stats::wilcox.test()
        function:
user <- data.dCt %>%
          pivot_longer(cols = -c(Group, Sample),
                       names_to = "Gene",
```

```
                      values_to = "dCt") %>%
          group_by(Gene) %>%
          summarise(MW_test_p = wilcox.test(dCt ~ Group)$p.value,
                    .groups = "keep")
# The stats::wilcox.test() functions is limited to cases without ties; therefore, a warning "cannot
        compute exact p-value with ties" will appear when ties occur.

RQ.plot <- RQ_plot(data = RQ.ddCt,
                   use.p = TRUE,
                   mode = "user",
                   p.threshold = 0.05,
                   use.log10FCh = TRUE,
                   log10FCh.threshold = 0.30103,
                   signif.show = TRUE,
                   signif.labels = signif.labels,
                   angle = 30)
```




```
# Access the table with results:
head(as.data.frame(RQ.plot[[2]]))
#>     Gene Control_mean Disease_mean Control_sd Disease_sd Control_norm_p
#> 1  Gene1    10.117002     7.959426   1.688115   1.887131     0.42664222
#> 2 Gene12     6.675667     5.058075   1.407052   2.951318     0.17299723
#> 3 Gene13     7.216583     6.565200   1.457719   1.062702     0.03823516
#> 4 Gene11    10.803992    10.199761   1.578909   1.608706     0.05806953
#> 5 Gene20     9.983942     9.481083   1.571200   1.143782     0.68169170
#> 6  Gene4     0.584125     0.360150   1.358816   1.670593     0.03149060
#>   Disease_norm_p       ddCt      FCh     t_test_p t_test_stat    MW_test_p
#> 1   1.744688e-01 -2.1575763 4.461647 1.690067e-05   4.7334112 5.136855e-05
#> 2   4.414803e-02 -1.6175917 3.068624 4.508310e-03   2.9520830 1.410617e-02
#> 3   1.756778e-01 -0.6513833 1.570674 6.426170e-02   1.9061886 1.009811e-02
#> 4   3.043468e-01 -0.6042308 1.520168 1.474652e-01   1.4716714 2.020225e-01
#> 5   2.041532e-02 -0.5028583 1.417018 1.801127e-01   1.3657411 1.271530e-01
#> 6   2.060155e-07 -0.2239750 1.167947 5.610445e-01   0.5847601 8.551052e-02
```

```
#>   MW_test_stat       p.used Selected as significant?
#> 1     4.049311 2.570256e-05                      Yes
#> 2     2.454548 1.360267e-02                      Yes
#> 3     2.572452 1.030219e-02                       No
#> 4     1.275810 2.061331e-01                       No
#> 5     1.525426 1.295905e-01                       No
#> 6     1.719571 8.683564e-02                       No
```

Three genes (Gene1, Gene12, and Gene19) are shown to meet the used significance criteria.

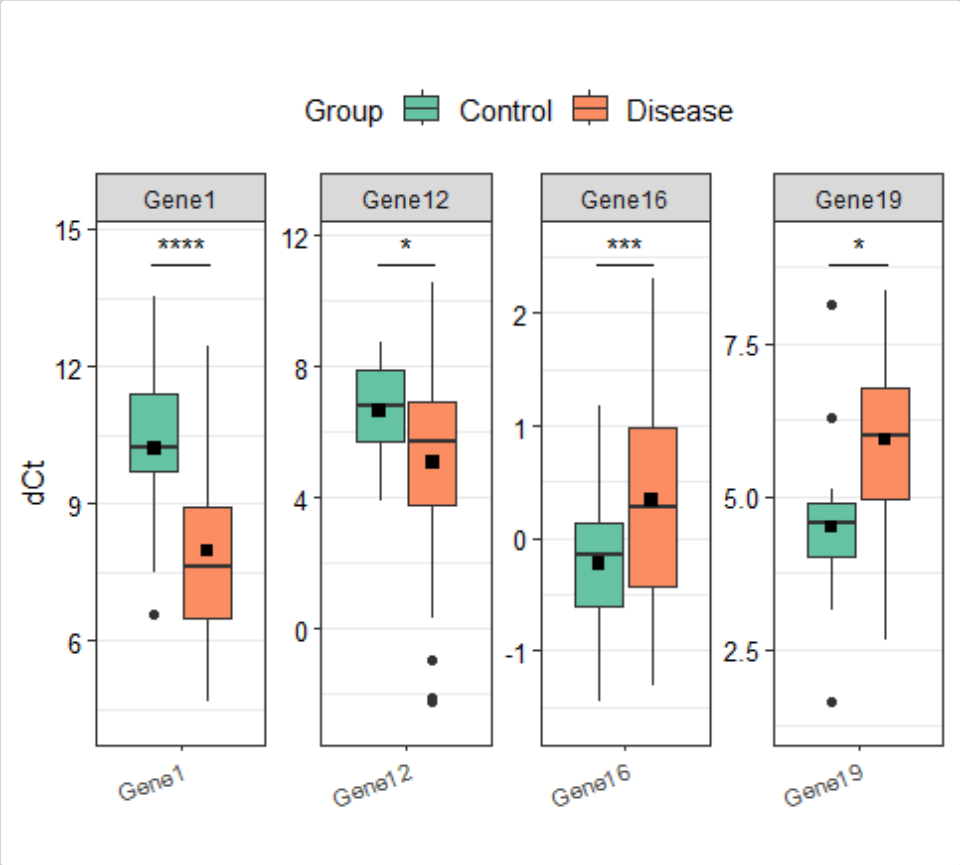
NOTE: If p values were not calculated due to the low number of samples, or they are not intended to be used to create a plot, the `use.p` parameter should be set to FALSE.

The `results_boxplot()` function

This function creates a boxplot that illustrates the distribution of the data for the genes. It is similar to `control_boxplot_gene()` function; however, some new options are added, including gene selection, faceting, addition of mean points to boxes, and statistical significance labels.

Data objects returned from the `make_Ct_ready()`, `exp_Ct_dCt()` and `delta_Ct() functions` can be directly applied to this function (see The summary of standard workflow, including function names and data flow).

```
final_boxplot <- results_boxplot(data = data.dCtF,
                                 sel.Gene = c("Gene1","Gene12", "Gene16","Gene19"),
                                 by.group = TRUE,
                                 signif.show = TRUE,
                                 signif.labels = c("****","*","***"," * "),
                                 signif.dist = 1.05,
                                 faceting = TRUE,
                                 facet.row = 1,
                                 facet.col = 4,
                                 y.exp.up = 0.1,
                                 angle = 20,
                                 y.axis.title = "dCt")
```



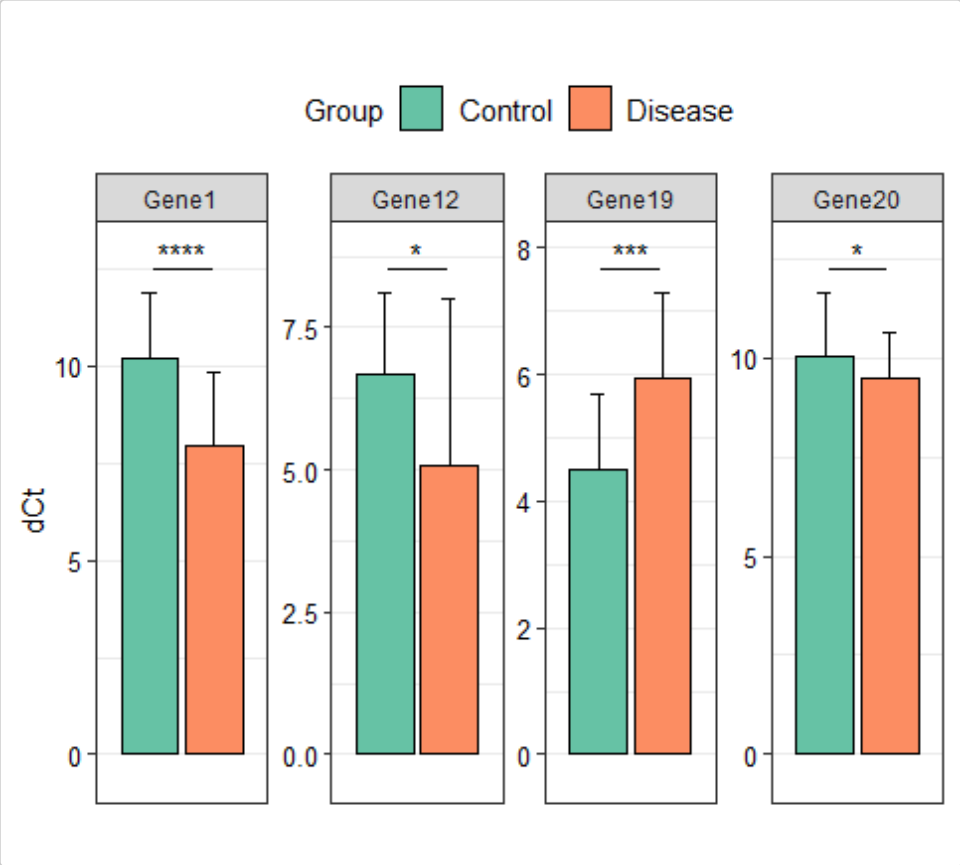
NOTE: If missing values are present in the data, they will be automatically removed, and a warning will appear.

The `results_barplot()` function

This function creates a barplot that illustrates mean and standard deviation values of the data for all or selected genes.

Data objects returned from the `make_Ct_ready()`, `exp_Ct_dCt()` and `delta_Ct() functions` can be directly applied to this function (see The summary of standard workflow, including function names and data flow).

```
final_barplot <- results_barplot(data = data.dCtF,
                                 sel.Gene = c("Gene1","Gene12","Gene19","Gene20"),
                                 signif.show = TRUE,
                                 signif.labels = c("****","*","***"," * "),
                                 angle = 30,
                                 signif.dist = 1.05,
                                 faceting = TRUE,
                                 facet.row = 1,
                                 facet.col = 4,
                                 y.exp.up = 0.1,
                                 y.axis.title = "dCt")
```



NOTE: At least two samples in each group are required to calculate the standard deviation and properly generate the plot.

Further analyses

Gene expression levels and differences between groups can be further analysed using the following methods and corresponding functions of the `RQdeltaCT` package:

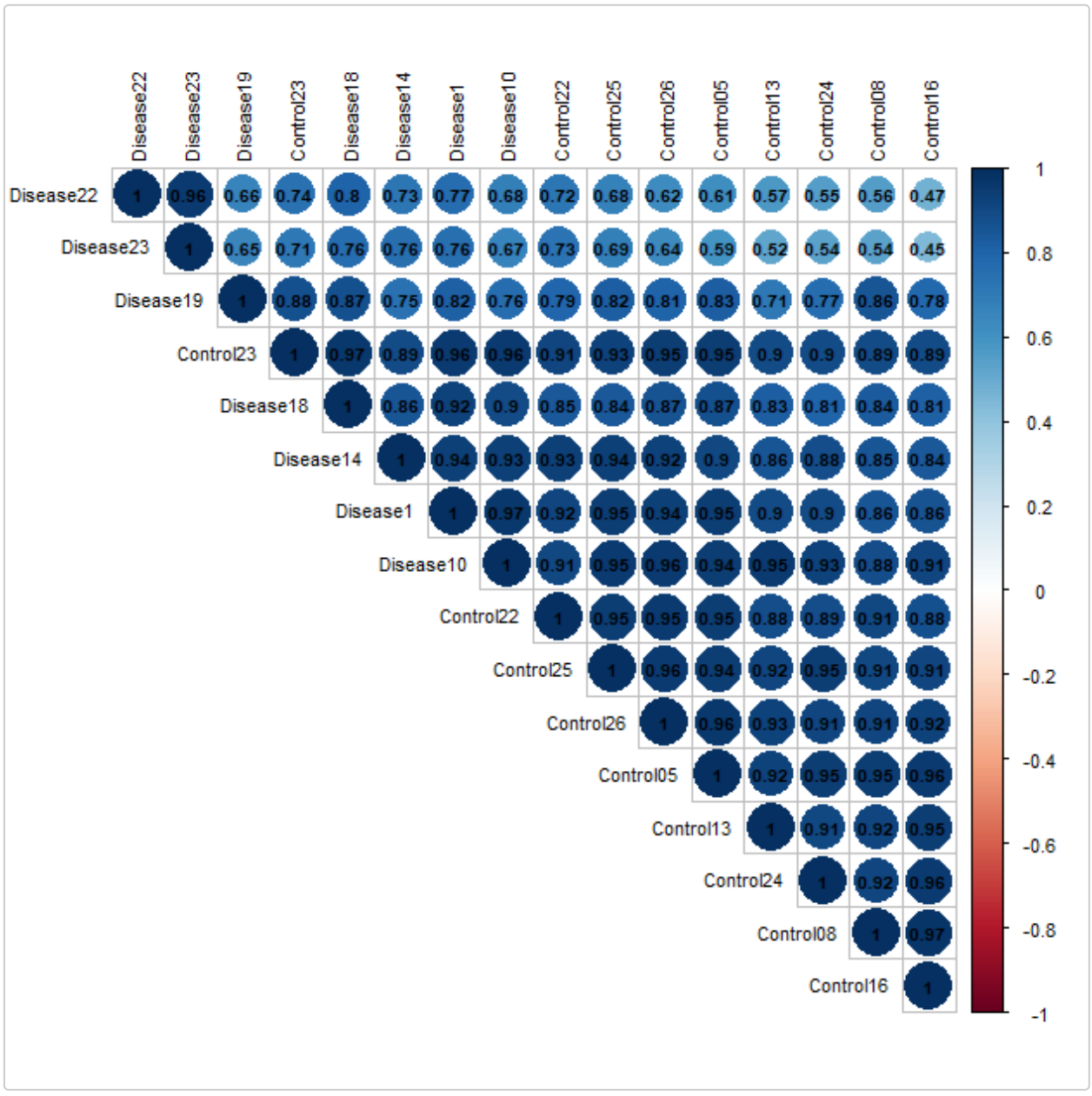
- correlation analysis - used to generate and visualise the correlation matrix of samples (`corr_sample()` function) and genes (`corr_gene()` function).
- simple linear regression - used to analysis and visualisation of relationships between pair of samples (`single_pair_sample()` function) and genes (`single_pair_gene()` function).
- Receiver Operating Characteristic (ROC) analysis - used to evaluate performance of sample classification by gene expression data (`ROCh()` function).
- simple logistic regression analysis - used to calculate odds ratio values for genes (`log_reg()` function).

Data objects returned from the `make_Ct_ready()`, `exp_Ct_dCt()`, and `delta_Ct() functions` can be directly applied to all of these functions (see The summary of standard workflow, including function names and data flow). Moreover, all functions can be run on the entire data or only on selected samples/genes.

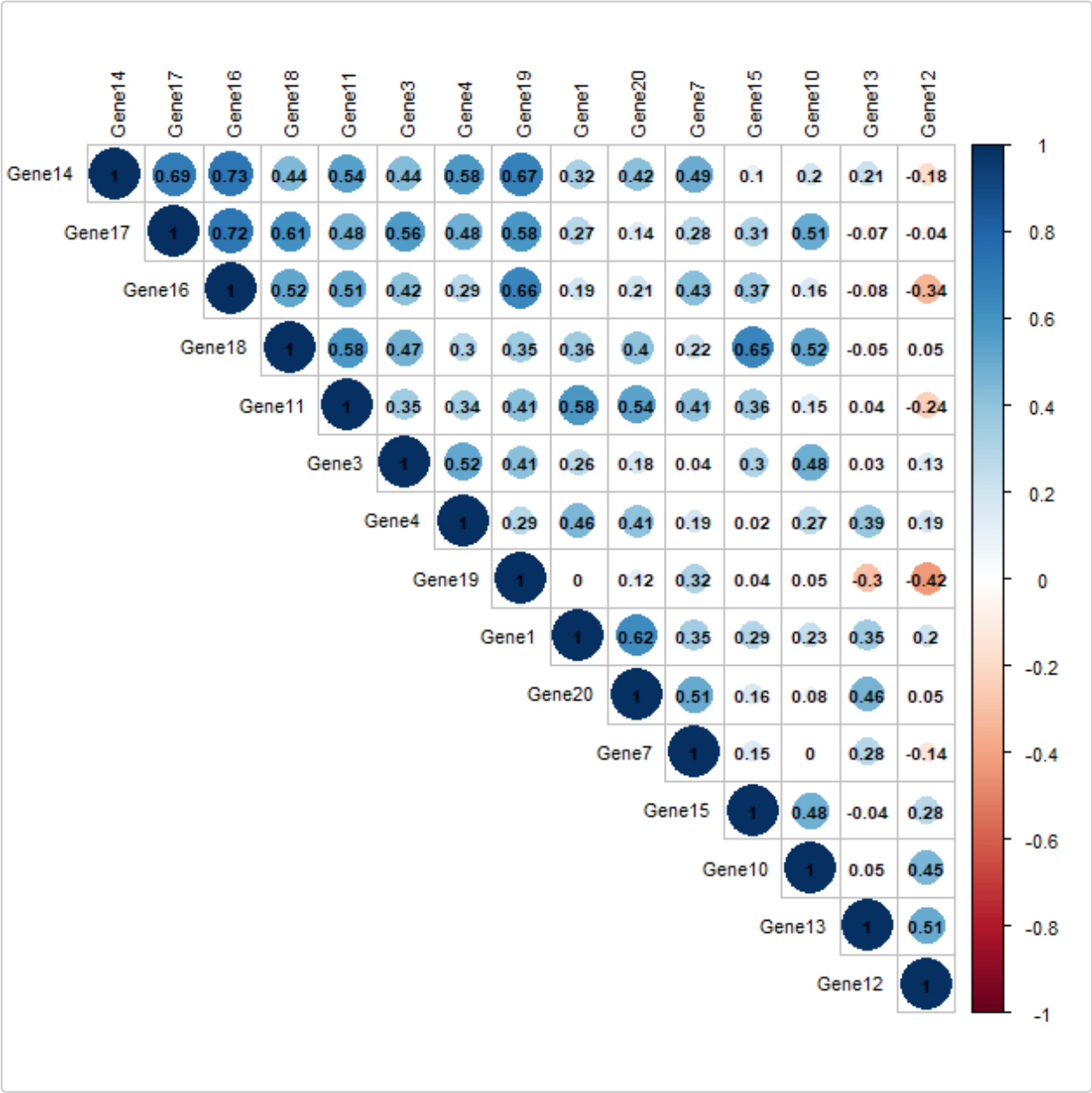
Correlation analysis

Correlation analysis is a very useful method to explore linear relationships between variables. The `RQdeltaCT` package offers `corr_sample()` and `corr_gene()` functions to generate and plot correlation matrices of samples and genes, respectively. The correlation coefficients can be calculated using either the Pearson or Spearman algorithm. To facilitate plot interpretation, these functions also have possibilities to order samples or genes according to several methods, e.g. hierarchical clustering or PCA first component (see `order` parameter).

```
library(Hmisc)
library(corrplot)
# To make the plot more readable, only part of the data was used:
corr.samples <- corr_sample(data = data.dCtF[15:30, ],
                            method = "pearson",
                            order = "hclust",
                            size = 0.7,
                            p.adjust.method = "BH")
```




```
library(Hmisc)
library(corrplot)
corr.genes <- corr_gene(data = data.dCt,
                            method = "spearman",
                            order = "FPC",
                            size = 0.7,
                            p.adjust.method = "BH")
```



The created plots are displayed on the graphic device. The returned objects are tables with computed correlation coefficients, p values, and p values adjusted by Benjamini-Hochberg correction (by default). Tables are sorted by the absolute values of correlation coefficients in descending order.

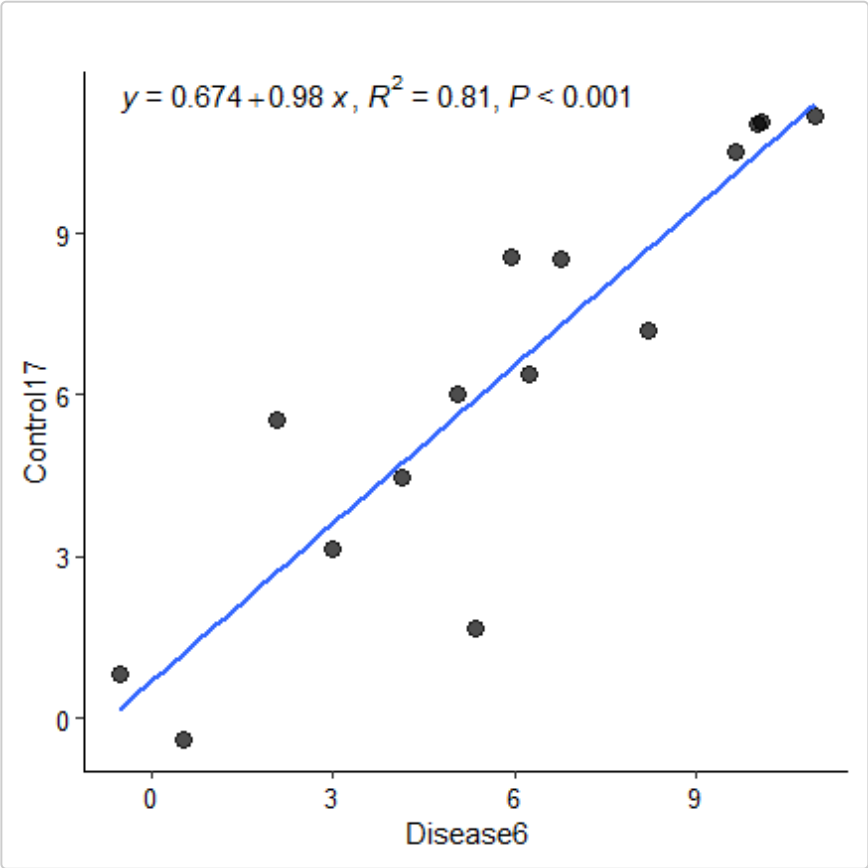
NOTE: A minimum of 5 samples/target are required for correlation analysis.

Simple linear regression analysis

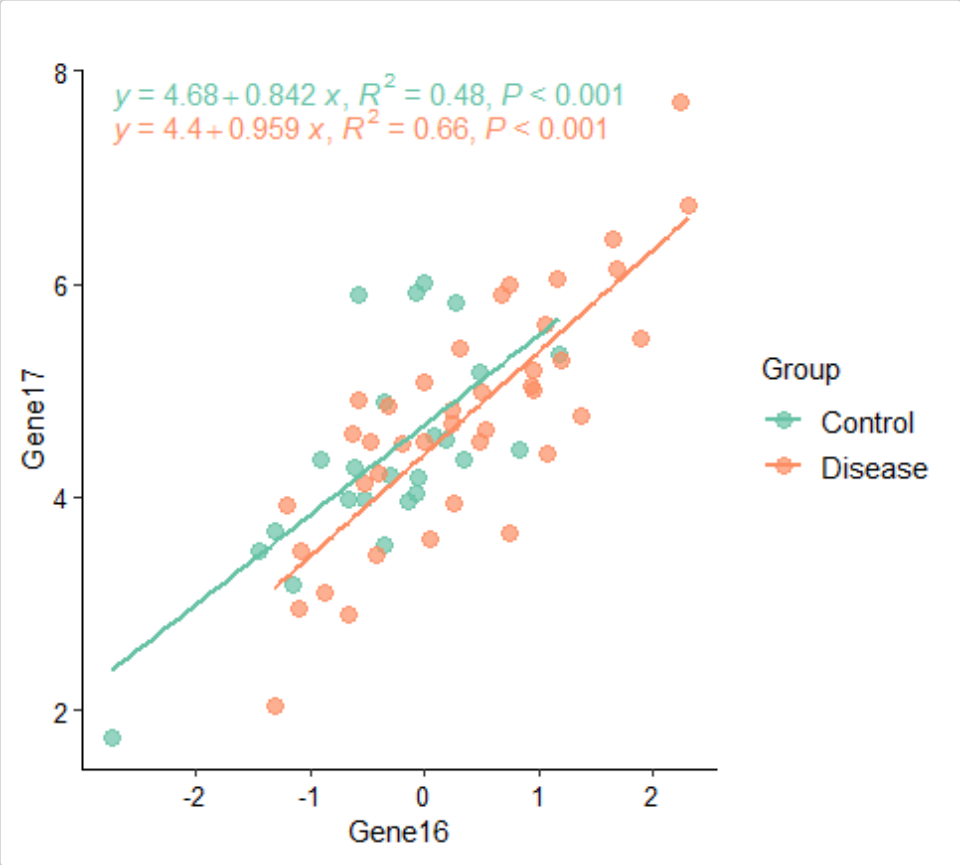
Linear relationships between pairs of samples or genes can be further analysed using simple linear regression models using the `single_pair_sample()` function (for analysis of samples) and the `single_pair_gene()` function for analysis of genes. These functions draw a scatter plot with a simple linear regression line. Regression results such as regression equation, coefficient of determination, F value, or p value can be optionally added to the plot.

```
library(ggpmisc)
Disease6_Control17 <- single_pair_sample(data = data.dCt,
                                         x = "Disease6",
                                         y = "Control17",
                                         point.size = 3,
                                         labels = TRUE,
```

```
                                label = c("eq", "R2", "p"),
                                label.position.x = 0.05)
```




```
library(ggpmisc)
Gene16_Gene17 <- single_pair_gene(data.dCt,
                                  x = "Gene16",
                                  y = "Gene17",
                                  by.group = TRUE,
                                  point.size = 3,
                                  labels = TRUE,
                                  label = c("eq", "R2", "p"),
                                  label.position.x = c(0.05),
                                  label.position.y = c(1,0.95))
```

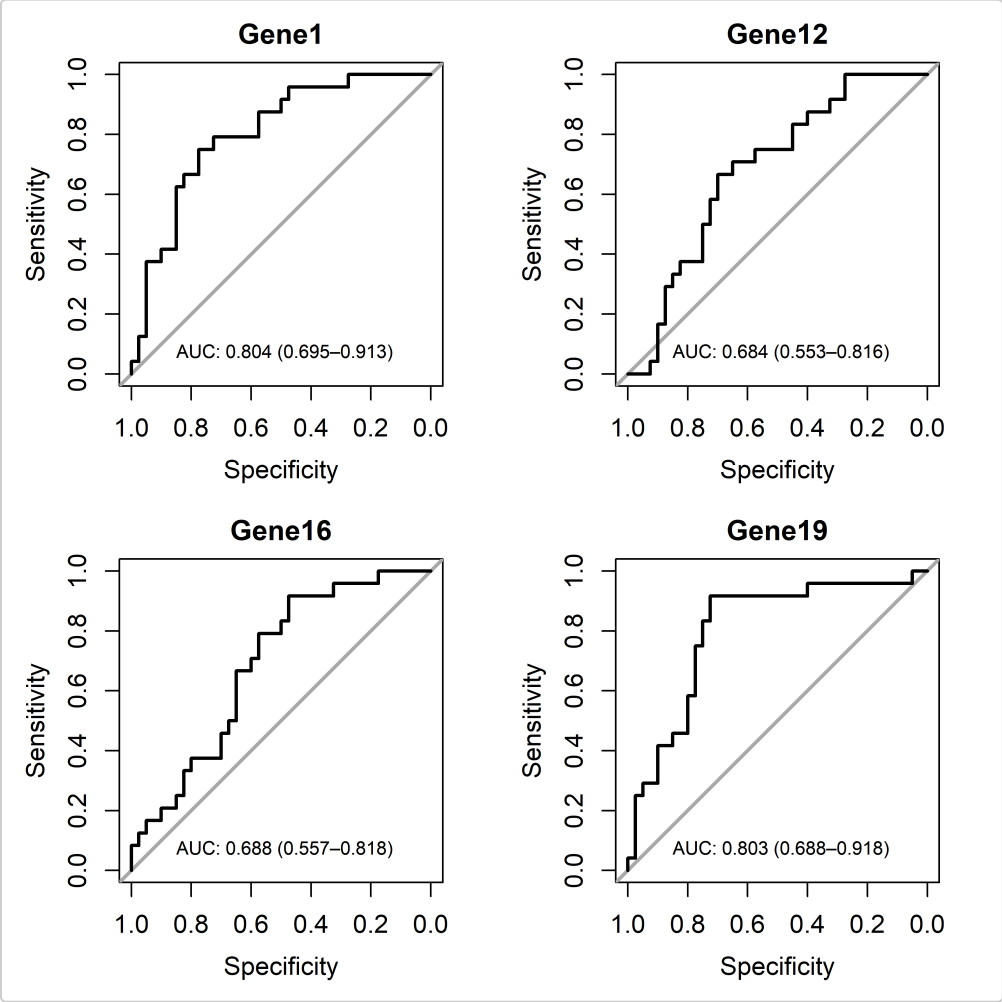


Receiver Operating Characteristic (ROC) analysis

The Receiver Operating Characteristic (ROC) analysis is useful for assessing the performance of sample classification to the particular group, based on gene expression data. In this analysis, ROC curves together with parameters such as the area under curve (AUC), specificity, sensitivity, accuracy, positive and negative predictive value are received. The `ROCh()` function was designed to perform all of these tasks. This function returns a table with calculated parameters and a plot with multiple panels, each with ROC curve for one gene.

NOTE: The created plot is not displayed on the graphic device, but should be saved as .tiff image (`save.to.txt` = TRUE) and can be opened directly from the file in the working directory.

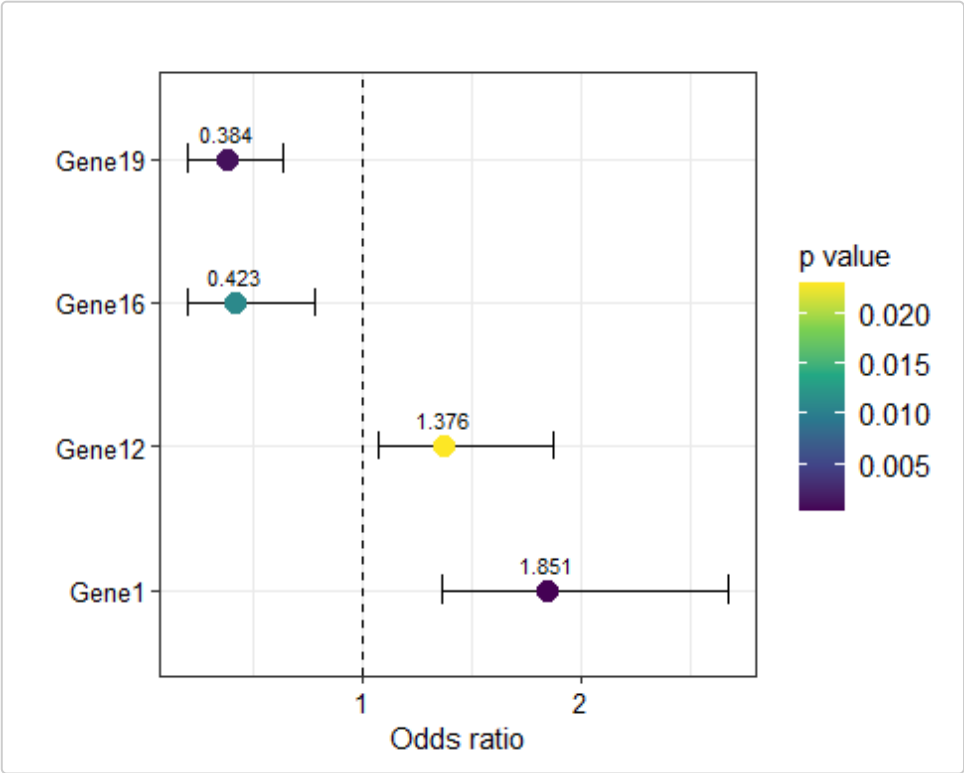
```
library(pROC)
# Remember to specify the numbers of rows (panels.row parameter) and columns (panels.col parameter)
         to be sufficient to arrange panels:
roc_parameters <- ROCh(data = data.dCt,
                       sel.Gene = c("Gene1","Gene12","Gene16","Gene19"),
                       groups = c("Disease","Control"),
                       panels.row = 2,
                       panels.col = 2)
roc_parameters
#>     Gene Threshold Specificity Sensitivity Accuracy       ppv       npv
#> 1  Gene1   9.40925       0.775   0.7500000 0.765625 0.6666667 0.8378378
#> 2 Gene12   6.40600       0.700   0.6666667 0.687500 0.5714286 0.7777778
#> 3 Gene16   0.48200       0.475   0.9166667 0.640625 0.5116279 0.9047619
#> 4 Gene19   5.11675       0.725   0.9166667 0.796875 0.6666667 0.9354839
#>     youden       AUC
#> 1 1.525000 0.8041667
#> 2 1.366667 0.6843750
#> 3 1.391667 0.6875000
#> 4 1.641667 0.8031250
```



Simple logistic regression

Logistic regression is a useful method to investigate the impact of the analysed variable on the odds of the occurrence of the studied experimental condition. In the `RQdeltaCT` package, `log_reg()` function allows to calculate for each gene a chances (odds ratio, OR) of being included in the study group when gene expression level increases by one unit. This function returns a plot and table with the calculated parameters (OR, confidence interval, intercept, coefficient, and p values).

```
library(oddsratio)
log.reg.results <- log_reg(data = data.dCt,
                           sel.Gene = c("Gene1","Gene12","Gene16","Gene19"),
                           group.study = "Disease",
                           group.ref = "Control")
```

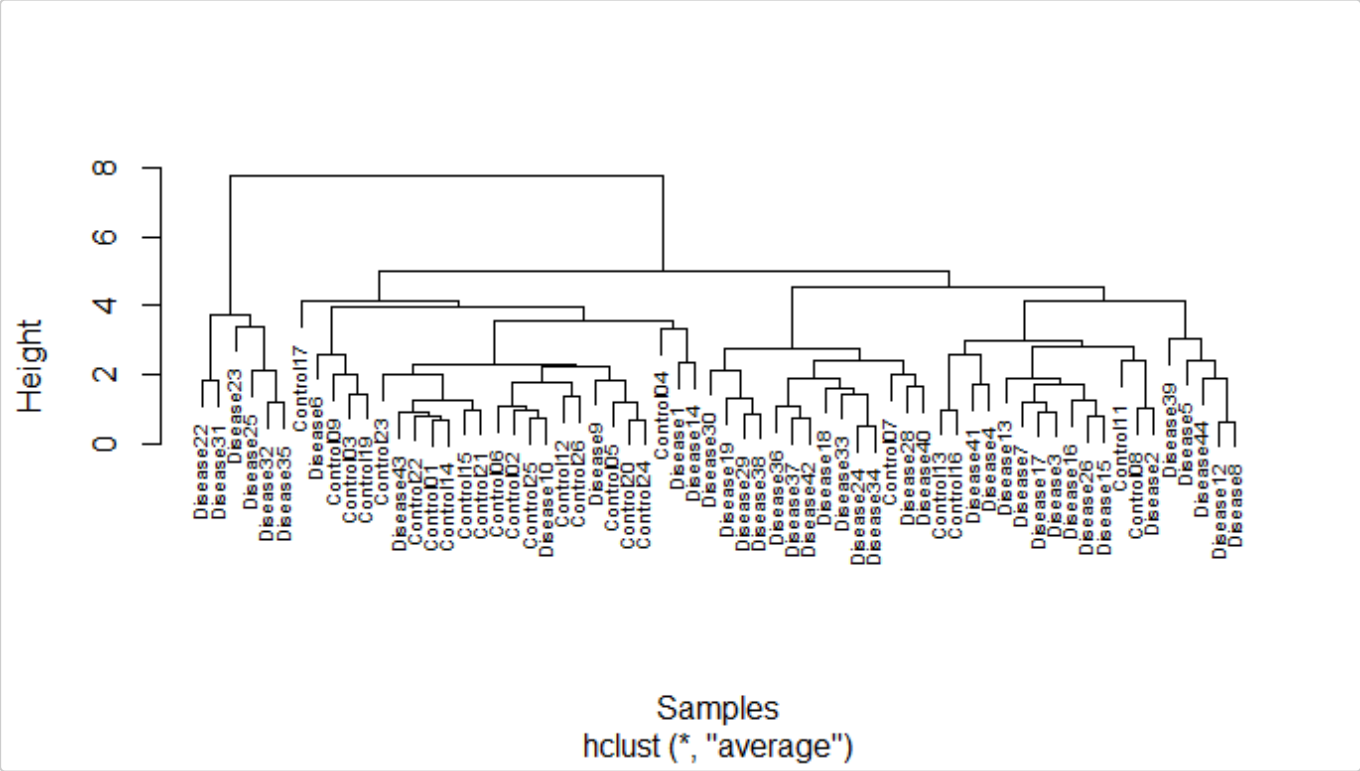



```
log.reg.results[[2]]
#>     Gene oddsratio CI_low CI_high  Intercept coeficient  p_intercept
#> 1  Gene1     1.851  1.363   2.675 -6.0820841  0.6159817 0.0001466894
#> 2 Gene12     1.376  1.076   1.874 -2.4160726  0.3189009 0.0085244809
#> 3 Gene16     0.423  0.204   0.781 -0.5101473 -0.8603227 0.0645722078
#> 4 Gene19     0.384  0.199   0.636  4.4442069 -0.9567034 0.0027620282
#>         p_coef
#> 1 0.0002879457
#> 2 0.0229318870
#> 3 0.0109936805
#> 4 0.0010298471
```

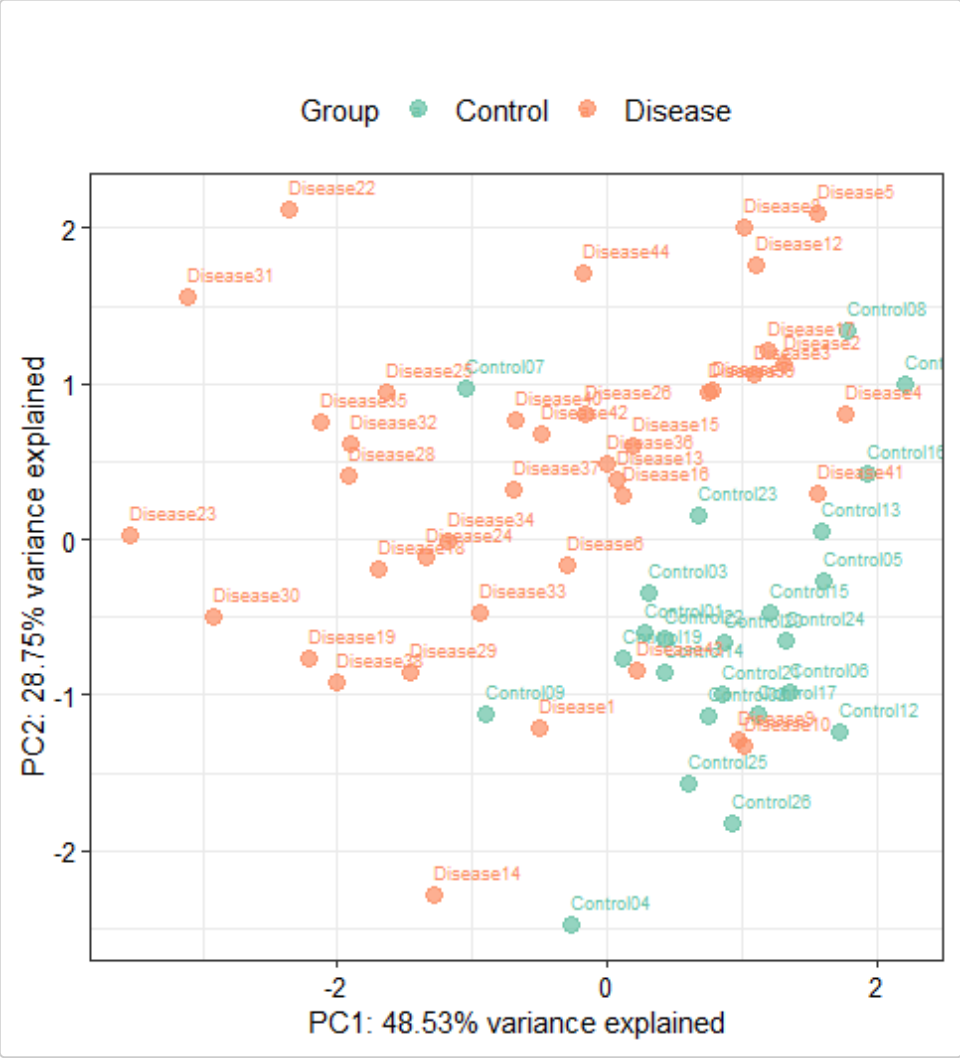
Hierarchical clustering and PCA

For final visualisation of results, quality control methods used earlier, including hierarchical clustering and PCA analysis, can be used to show the efficiency of the separation of the compared groups by the expression of finally selected genes. For this purpose, hierarchical clustering and PCA analysis can be run on samples using previously used `control_cluster_sample()` and `control_pca_sample()` functions, respectively, but for selected genes:

```
control_cluster_sample(data = data.dCt,
                       sel.Gene = c("Gene1","Gene12","Gene16","Gene19"),
                       method.dist = "euclidean",
                       method.clust = "average",
                       label.size = 0.6)
```




```
control.pca.sample <- control_pca_sample(data = data.dCt,
                                         sel.Gene = c("Gene1","Gene12","Gene16","Gene19"),
                                         point.size = 3,
                                         label.size = 2.5,
                                         legend.position = "top")
```



The created plots are displayed on the graphic device.

NOTE: Minimum of three samples or genes in the data is required for these analyses. In PCA analysis, samples with missing values will be removed before computation.

Session info

```
sessionInfo()
#> R version 4.3.2 (2023-10-31 ucrt)
#> Platform: x86_64-w64-mingw32/x64 (64-bit)
#> Running under: Windows 10 x64 (build 19045)
#>
#> Matrix products: default
#>
#>
#> locale:
#> [1] LC_COLLATE=Polish_Poland.utf8  LC_CTYPE=Polish_Poland.utf8
#> [3] LC_MONETARY=Polish_Poland.utf8 LC_NUMERIC=C
#> [5] LC_TIME=Polish_Poland.utf8
#>
#> time zone: Europe/Warsaw
#> tzcode source: internal
#>
#> attached base packages:
#> [1] stats     graphics  grDevices utils     datasets  methods   base
#>
#> other attached packages:
```

```
#>  [1] oddsratio_2.0.1 pROC_1.18.5     ggpmisc_0.5.5   ggpp_0.5.5
#>  [5] corrplot_0.92   Hmisc_5.1-1     ggsignif_0.6.4  ctrlGene_1.0.1
#>  [9] coin_1.4-3      survival_3.5-7  lubridate_1.9.3 forcats_1.0.0
#> [13] stringr_1.5.0   dplyr_1.1.3     purrr_1.0.2     readr_2.1.4
#> [17] tidyr_1.3.0     tibble_3.2.1    ggplot2_3.4.3   tidyverse_2.0.0
#> [21] RQdeltaCT_1.0.0
#>
#> loaded via a namespace (and not attached):
#>  [1] tidyselect_1.2.0   viridisLite_0.4.2  farver_2.1.1       libcoin_1.0-10
#>  [5] fastmap_1.1.1      TH.data_1.1-2      digest_0.6.33      rpart_4.1.21
#>  [9] timechange_0.2.0   lifecycle_1.0.3    cluster_2.1.4      magrittr_2.0.3
#> [13] compiler_4.3.2     rlang_1.1.1        sass_0.4.7         tools_4.3.2
#> [17] utf8_1.2.3         yaml_2.3.7         data.table_1.14.8  knitr_1.44
#> [21] labeling_0.4.3     htmlwidgets_1.6.4  plyr_1.8.9         multcomp_1.4-25
#> [25] foreign_0.8-85     withr_2.5.1        nnet_7.3-19        grid_4.3.2
#> [29] stats4_4.3.2       fansi_1.0.5        colorspace_2.1-0   scales_1.2.1
#> [33] MASS_7.3-60        cli_3.6.1          mvtnorm_1.2-4      rmarkdown_2.25
#> [37] generics_0.1.3     rstudioapi_0.15.0  tzdb_0.4.0         polynom_1.4-1
#> [41] cachem_1.0.8       modeltools_0.2-23  splines_4.3.2      parallel_4.3.2
#> [45] matrixStats_1.0.0  base64enc_0.1-3    vctrs_0.6.3        Matrix_1.6-4
#> [49] sandwich_3.1-0     jsonlite_1.8.7     SparseM_1.81       confintr_1.0.2
#> [53] hms_1.1.3          Formula_1.2-5      htmlTable_2.4.2    jquerylib_0.1.4
#> [57] glue_1.6.2         codetools_0.2-19   stringi_1.7.12     gtable_0.3.4
#> [61] munsell_0.5.0      pillar_1.9.0       htmltools_0.5.7    quantreg_5.97
#> [65] R6_2.5.1           evaluate_0.22      lattice_0.21-9     backports_1.4.1
#> [69] bslib_0.5.1        MatrixModels_0.5-3 Rcpp_1.0.11        gridExtra_2.3
#> [73] nlme_3.1-163       checkmate_2.3.1    mgcv_1.9-0         xfun_0.40
#> [77] zoo_1.8-12         pkgconfig_2.0.3
```