DAPAR and *ProStaR* user manual

Samuel Wieczorek*, Florence Combes*, Cosmin Lazar, Quentin Giai Gianetto, Laurent Gatto, Alexia Dorffer, Anne-Marie Hesse, Yohann Coute, Myriam Ferro, Christophe Bruley and Thomas Burger*

April 19, 2016

Abstract

DAPAR (Differential Analysis of Protein Abundance with R) and *ProStaR* (Proteomics and Statistics with R) are two Bioconductor packages that contain the necessary functions to analyze proteomics data (*DAPAR*), as well as the corresponding graphical user interfaces (*ProStaR*). This document guides the practitioner through the use of *DAPAR* (R command lines) and *ProStaR* (click-button interface, so that no programming skill is required).

*firstname.lastname@cea.fr

Contents

1 Introduction

DAPAR and *ProStaR* are a series of software dedicated to the processing of proteomics data. More precisely, they are devoted to the analysis of quantitative datasets produced in bottom-up discovery proteomics with a LC-MS/MS pipe-line (Liquid Chromatography and Tandey Mass spectrometry).

DAPAR (Differential Analysis of Protein Abundance with R) is an R package that contains all the necessary functions to:

- Import/export a quantitative dataset. Here, a quantitative dataset denotes a table where each protein is represented by a line and and each replicate is represented by a column; each cell of the table contain the abundance of a given protein in a given sample; the replicates are clustered into different conditions (or groups), and the purpose of the analysis is to isolate the few proteins the abundance of which significantly differ between the conditions (or groups).
- Compute and display meaningful statistics regarding the quantitative dataset.
- Perform the various processing steps of a complete quantitative analysis: (i) filtering and data cleaning; (ii) cross-replicate normalization; (iii) missing value imputation; (iv) aggregation of peptide intensities into protein intensities; (v) statistical tests and false discovery rate computation.

This package can be used on its own; or as a complement to the numerous Bioconductor packages (https://www.bioconductor.org/) it is compliant with; or through the *ProStaR* interface. *ProStaR* (Proteomics and Statistics with R) is a web-interface based on Shiny (http://shiny.rstudio.com/) that provides Graphical User Interfaces (GUI) to all the *DAPAR* functionalities, so as to guide any practitioner that is not comfortable with R programming through the complete quantitative analysis process.

In DAPAR, a serie of functions may be called from two types of variables: a dataframe that contains quantitative data and an object of class MSnSet. The first case has been developped to make DAPAR easier to use (in command line) for users who do not want to work with MSnSet files and to be compliant in the future with the Proline Suite (http://proline.profiproteomics.fr/).

2 Installation

There are 3 ways to use *DAPAR*:

- The first one is to use *DAPAR* alone, through command lines or scripts. To do so, the user simply has to install *DAPAR* on his/her own work station, as instructed in Section 2.2;
- The second one is to use *DAPAR* along with its graphical interface *ProStaR*, and to have them running on the user's station (referred to as stand-alone install). In such case, it is necessary to install *DAPAR* first, as instructed in Section 2.2, and *ProStaR* then, as instructed in Section 2.1.1;
- In the case where several *ProStaR* users who are not confortable with R (programming or installing), it is best to have a single version of *DAPAR* and *ProStaR* running on a Unix/Linux server. The users will use *ProStaR* through a web browser, exactly if it were locally installed, yet, a single install has to be administrated. In that case, *DAPAR* has to be classically installed

(Section 2.2), while on the other hand, the install of *ProStaR* is slightly different on a server (Section 2.1.2).

For a stand-alone use, both *DAPAR* and *ProStaR* can run on any operating system (Unix/Linux, Mac OS X and Windows) as long as R is installed. In any case (stand-alone or server), a recent version of R ($\geq$ 3.2) is needed.

2.1 ProStaR (with DAPAR)

ProStaR can be run in two differents ways: standalone or server. The pre-requested packages described above have to be installed on the server if the user run a shiny-server to distribute *ProStaR* or on a local machine if *ProStaR* is run locally.

2.1.1 Stand-alone version

To run the stand-alone version, it is necessary to install the package in a directory where the user have read/write permissions. If the user have administrator privileges, then in a R console, enter:

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("Prostar")
```

This step will automatically install the following packages: *shiny*, *reshape2*, *rhandsontable*, *quantmod*, *data.table*, *DT*, *shinyjs*.

Once the package is installed, to launch *ProStaR*, then enter:

```
> library(Prostar)
> Prostar()
```

A new window of the default web browser opens.

2.1.2 Server version

This version uses a Shiny Server (https://github.com/rstudio/shiny-server). It is a server program that makes Shiny applications available over the web. Please follow installation instructions if you do not have a server yet.

The first step is to install *Prostar* as described in Section 2.1.1 in order to have the dependencies installed.

In the sequel, we suppose the directory of shiny-server is `/srv/shiny-server` (this is the default configuration of a Shiny Server). In the home directory, unzip the package tarball and move to the R directory.

```
# unzip Prostar_xxxx.tar.gz
# cd Prostar/R
```

Create a directory named Prostar in the Shiny Server directory and then copy the 3 files: ui.R, global.R and server.R.

```
# sudo mkdir /srv/shiny-server/Prostar
# sudo cp R/ui.R /srv/shiny-server/Prostar/.
# sudo cp R/global.R /srv/shiny-server/Prostar/.
# sudo cp R/server.R /srv/shiny-server/Prostar/.
```

Then, complete the installation by copying the 'www' directory of *ProStaR*:

```
# sudo cp -R inst/extdata/www /srv/shiny-server/Prostar/.
```

Check if the configuration file of shiny-server is correct.
For more details, please visit http://rstudio.github.io/shiny-server/latest/.

Now, the application should be available via a web browser at http://*servername*:*port*/Prostar.

2.2 DAPAR (alone)

To install the package *DAPAR* from the source file with administrator rights, start R and enter:

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("DAPAR")
```

This step will automatically install the following packages:

- From CRAN: *RColorBrewer*, *Cairo*, *png*, *lattice*, *reshape2*, *tmvtnorm*, *norm*, *ggplot2*, *imputeLCMD*, *gplots*, *XLConnect*, *knitr*, *cp4p*,*doParallel*, *scales*, *stats*, *grDevices graphics*, *utils*
- From Bioconductor: *MSnbase*, *preprocessCore*, *impute*, *limma*, *pcaMethods*

3 Navigating through the ProStaR interface

3.1 Overview of the interface

As illustrated on Fig. 1, ProStaR proposes a classical Graphical User Interface (GUI) to visualize and interact with the data. On the top, a navbar menu helps navigating throught the various DAPAR functionalities and running them. It is divided into five submenus:

- **ProStar**: The welcome page, depicted on Fig. 1.
- **Dataset manager**: It contains the tools to import and export datasets;
- **Descriptive statistics**: It provides different visualization tools that are helpful to understand the dataset, and to picture the influence of the various processing;
- **Data processing**: This is the heart of ProStaR, where all the DAPAR functionalities can be accessed to;

Figure 1: Default screen of ProStaR

- **Help**: A serie of informations about the software, associated references, etc.

On the right hand side of the navbar menu, a dropdown menu referred to as "Dataset versions" makes it possible to navigate back through the history of the processing. Its use is detailed in Section 3.6

3.2 Dataset manager

The "Dataset manager" allows the user to open, import or export quantitative datasets. *ProStaR* and *DAPAR* use the MSnSet format which is part of the package *MSnbase*: It is either possible to load existing MSnSet files (see Section 3.2.1) or to import text (-tabulated) and Excel files (see Section 3.2.2).

3.2.1 Open MSnSet

The user can upload a dataset that is already formated as an MSnSet file, by clicking on "Open MSnSet File" (see Fig. 2). This action opens a pop-up window, so as to let the user choose the appropriate file. Once the file is uploaded, a short summary of the dataset is shown, which includes the number of samples, the number of proteins in the dataset, the percentage of missing values and the number of lines which only contain missing values.

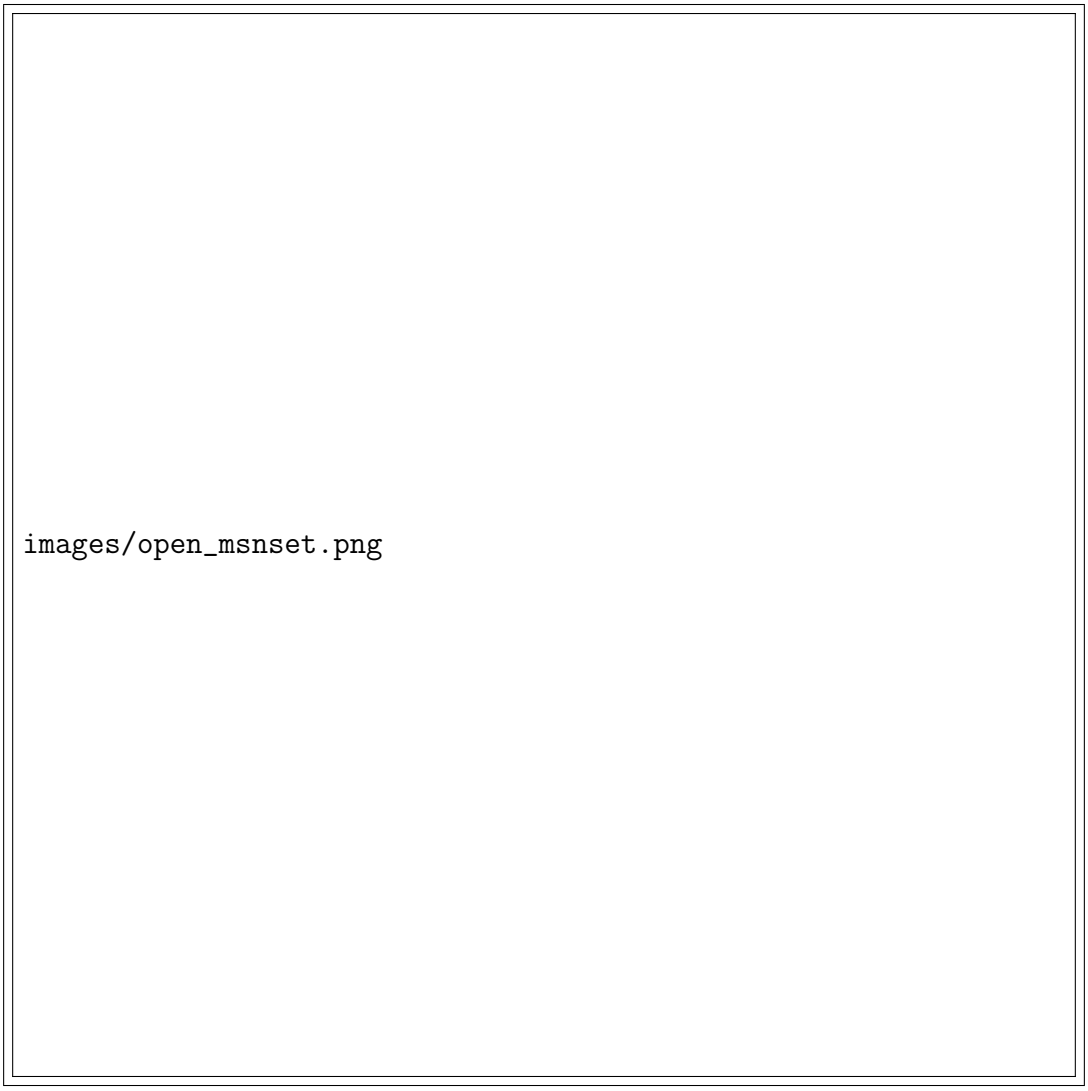

Figure 2: Open a MSnSet file

Once done, the menu of "Dataset versions" is updated to "Original - peptide" or "Original - protein" whether the file contains quantitative information about peptides or proteins (see Section 3.6). All the plots in the "Descriptive statistics" submenu (see Section 3.3) become accessible and all the widgets to interact with *ProStaR* are preloaded.

Command line: It is possible to open an MSnSet dataset directly in command line (*i.e.* without *ProStaR* interface), using function `readRDS()`.

The user can find an example of MSnSet file in the installation directory of *DAPAR*:
`inst/extdata/UPSpepx2.MSnSet`

3.2.2 Import data

Alternatively, the user can create a quantitative dataset in the MSnSet format, on the basis of CSV (Comma Separated Values) or Excel files that contains the results of a proteomics analysis. To do so, one has to click on "Convert data to MSnSet". Then, the right panel splits into 5 tabs that guide the user through the various creation of the MSnSet object:

Select file: Select the CSV/txt or Excel file to import (see Fig. 3). These files must contain a table where each line corresponds to a peptide or protein, except the first one which must contain the names of the colums. If the user chooses an Excel file, a dropdown menu appears and ask the user to select the spreadsheet containing the data. Among the colums, one must contain an ID that uniquely defines the peptides or proteins, as well as a series of columns containing the abundance values (either log-transformed or not). As it appears in Fig. 3, some options allows for specifying if the data are related to peptides or proteins, the log2 transformation of the abundance values [1], as well as for automatically replacing the 0 and NaN values by **NA**.

Data ID: This step is to set the column that correspond to the unique ID of peptides or proteins. The user has two options: let Prostar creates such an ID by itself or choose among the columns available in the data file. If choosing the second option, then a drop-down menu appears and provides the list of the column names. A column corresponding to the unique ID of the peptides or proteins should be selected (see Fig. 4). If the column contains non unique IDs, a warning alerts the user and suggests him to choose another column.

Exp. and Feat. data: In the "Quantitative data" list, select the columns that correspond to the quantitative data. Each time the user selects an item in the list, it is moved up to the field above (see Fig. 5). If an item is selected by mistake, it can be removed by pressing on the SUPPR key.

Sample metadata: In this tab, the user fills the informations related to the samples. The colum named *Experiment* is filled by default with the name of the different samples. The user fills the other columns: *Label* correspond to the conditions of the experiment that will be compared during the differential analysis; *Bio.Rep*, *Tech.rep* and *Analyt.Rep.* correspond respectively to the biological, technical and analytical replicates (Fig. 6). The column Label is mandatory (for the subsequent differential analysis), the other ones are optional.

Convert: Finally, enter the name of the MSnSet to be created (Fig. 7) and click on "Convert data". The data are converted and automatically loaded in *ProStaR*. The name of the file appears on the top of the left panel, above the menu.

Command line: In *DAPAR*, the function to create an MSnSet from a CSV file is `createMSnSet()`.

3.2.3 Export

Once an MSnSet has been created, it is possible to save it as a MSnSet binary object (so that next time, it is not necessary to create it, and a simple uploads makes it, as described in Section 3.2.1). It

[1] In DAPAR, the analysis is always conducted on log-transformed data. They may have previously been transformed, but if not, then DAPAR automatically performs the transformation. The user should not try applying any DAPAR processing on data that are not log-transformed, for the result would be dubious.

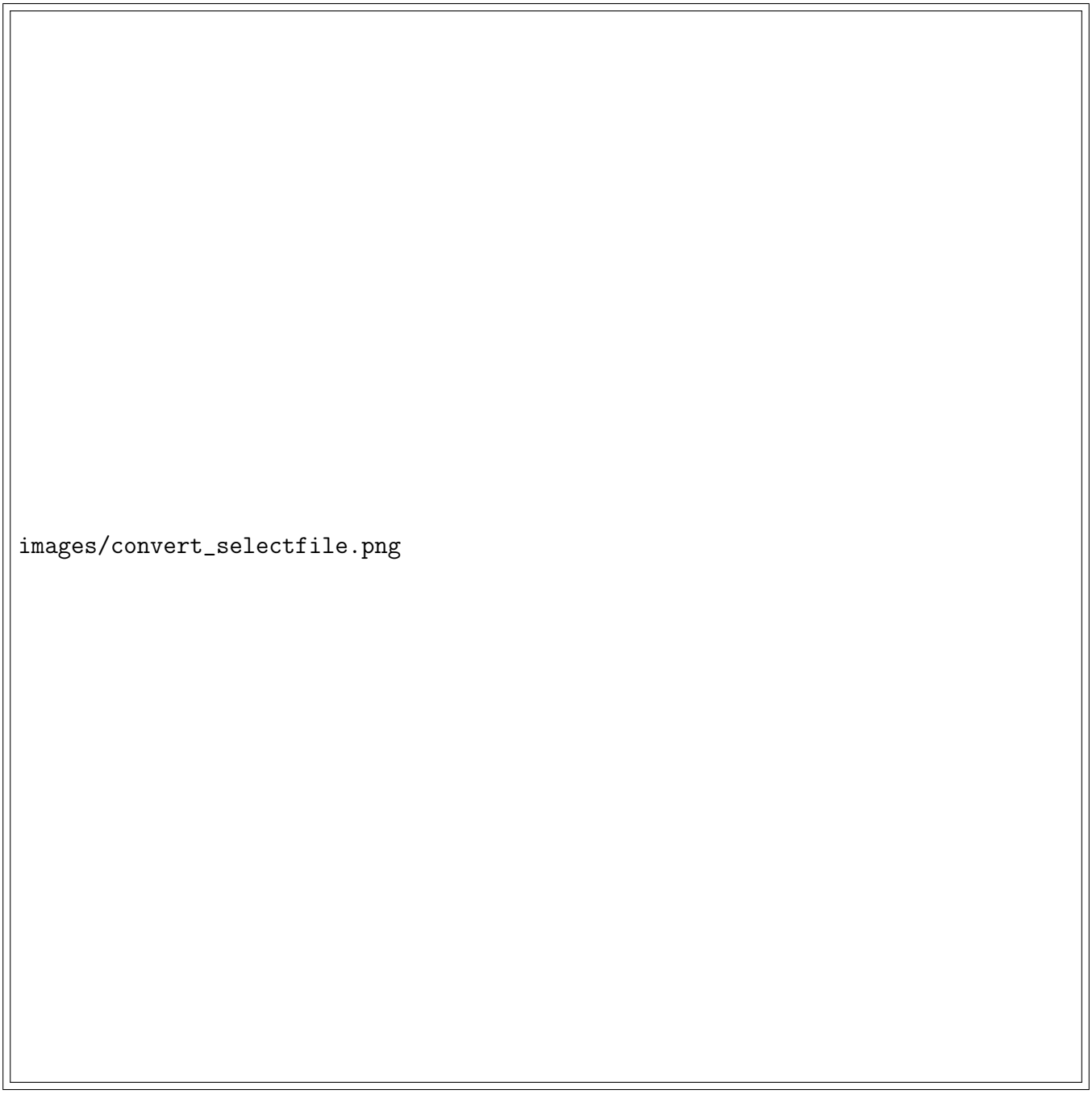

Figure 3: Importing an Excel file, tab 1.

is also possible to export it as an Excel spreadsheet. To do so, one simply goes on the corresponding tab and select the appropriate option.

Command line: When working exclusively with *DAPAR*, the functions are `writeMSnSetToExcel()` (to export in Excel format) and `saveRDS()` (to export in MSnSet format).

The user can download the plots showed in *Prostar* by right-clicking on the plot. A contextual menu appears and let the user choose either "Save image as" or "Copy image". In the latter case, he/she has

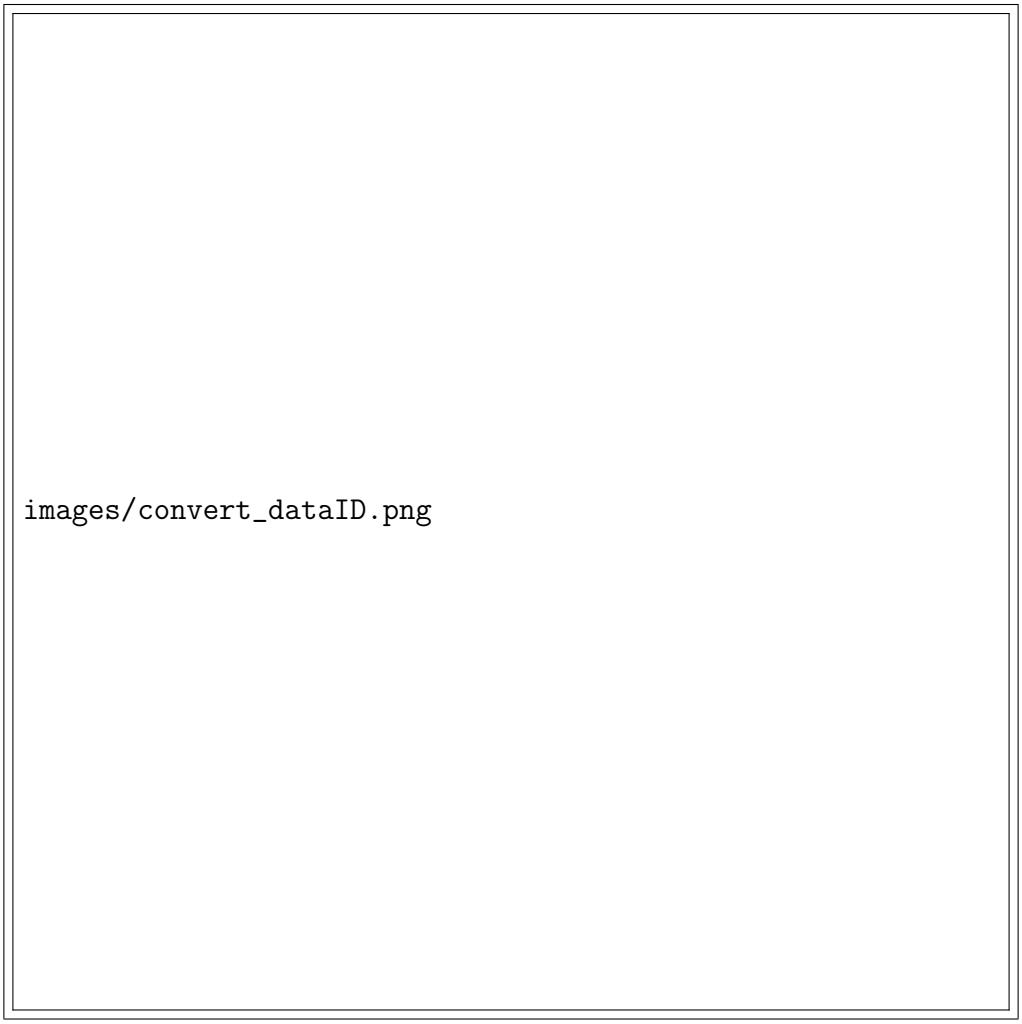

Figure 4: Importing a CSV file, tab 2.

to paste the image in appropriate software.

3.2.4 Session log

Each time the user validates a processing step (by clicking on the "Save <*the_step*>" button, see Section 3.4), the entire related information (such as the method name and its parameters) is added to the table shown in the "Session log" tab (see Figure 8). Hence, this table is a history of how the data were processed during the session. Let us note that, if a dataset is processed, then saved and reloaded in a new session, the session log is naturally empty. To have a complete view on the previous processing applied to a given dataset, pelase refer to Section 3.3.2).

images/convert_exp_featdata.png

Figure 5: Importing a CSV file, tab 3.

3.3 Descriptive statistics

Several plots (one plot per tab) are proposed to help the user to have a quick and as complete as possible overview of his/her dataset. This menu is an essentiel element for the user to check that each processing step indeed gave the expected result.

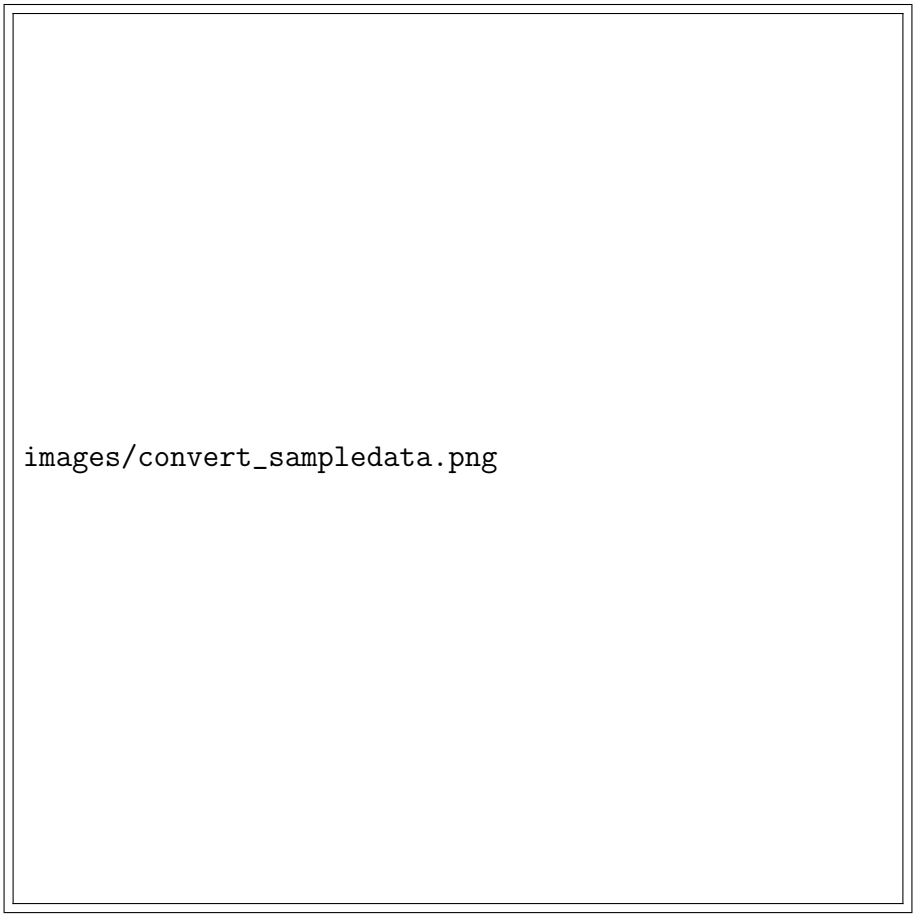

Figure 6: Importing a CSV file, tab 4.

3.3.1 Missing value summary

The barplot on the left represents the number of missing values in each sample. The different colors correspond to the different conditions (or label). The histogram in the middle displays the distribution of missing values; the red bin counts the peptide or protein lines that only contains missing values (Fig. 9). The barplot on the right shows the distribution of missing values per condition.

Command line: In *DAPAR*, the functions for these three plots are:

- for the dataframe parameter: `mvPerLinesHisto()`, `mvHisto()` and `mvPerLinesHistoPerCondition()`,
- for an object of class MSnSet: `wrapper.mvPerLinesHisto()`, `wrapper.mvHisto()` and `wrapper.mvPerLinesHistoPerCondition()`.

3.3.2 Data explorer

This panel allows viewing the content of the MSnSet structure. It is made of four tables, that are represented in a tab each. The first one, named "Quantitative data" contains quantitative values (see

Figure 7: Importing a CSV file, tab 5.

Fig. 10). The missing values are represented by empty cells.

The second tab is named "Analyte metadata". It contains the metadata of the proteins (see Fig. 11).

The thrid tab is named "Replicate metadata". The information displayed here is the one entered by the user during the import step (see Fig. 12).

The last tab, named "Dataset history" contains the log of the previous processing. Contrarily to the "Session log" panel (see Section 3.2.4), the information here does not relate to the session, and is saved from a session to the next one.

Command line: The *DAPAR* functions to get the three first tables are in fact those from the *MSnbase*

package: `exprs()` (Quantitative data), `fData()` (Analyte metadata) and `pData()` (Replicate metadata). Similarly, the "Dataset history" information is also accessible. In fact, it is stored in the slot (`processingData`) of the current MSnSet object. In a R console, if `obj` is the current dataset, it can be accessed by entering:

```
> getProcessingInfo(obj)
```

3.3.3 Heatmap

A heatmap is drawn with the associated dendrogram (see Fig. 13). The colors represent the intensities: red for high intensities and green for low intensities. White color is reserved for missing values. The dendrogram shows the hierarchical classification of the samples. This classification can be tuned by two parameters:

- **Distance**: Euclidean or Manhattan
- **Linkage**: Ward.D or mean

Command line: In *DAPAR*, the corresponding function are:

- for the dataframe parameter: `heatmapD()`,
- for an object of class MSnSet: `wrapper.heatmapD()`.

3.3.4 Correlation matrix

In this tab, it is possible to visualize the extent to which the replicates correlate or not (see Fig. 14). The contrast in the matrix may be changed by modifying the color gradient.

Command line: In *DAPAR*, the corresponding function are:

- for the dataframe parameter: `heatmapD()`,
- for an object of class MSnSet: `wrapper.heatmapD()`.

3.3.5 Boxplot

The protein distribution by replicates is summarized with boxplots (see Fig. 15). The user can change the legend of the samples (X-axis) by checking items in the checkboxes group. The colors of the boxes correspond to the different conditions (column **Label** in the table of *Samples Meta Data*).

Command line: In *DAPAR*, the corresponding functions are:

- for the dataframe parameter: `boxPlotD()`,
- for an object of class MSnSet: `wrapper.boxPlotD()`.

3.3.6 Variance distribution

This plot shows the distribution of the variance of the log-intensity of proteins for each condition (see Fig. 16).

Command line: In *DAPAR*, the corresponding function is `varianceDistD()`.

3.3.7 Density plot

This plots shows the distribution of the log-intensity of proteins for each condition (see Fig. 17).

Two options are available to custom the plot:

- **Plot to show** which defines how to color the replicates: one color for each condition (value "By condition") or one color per replicate (value "By replicate"). By default, the data are colored by condition.
- **Select data to show** Select the replicates to display. By default, all the replicates are showed.

Command line: In *DAPAR*, the corresponding functions are:

- for the dataframe parameter: `densityPlotD()`,
- for an object of class MSnSet: `wrapper.densityPlotD()`.

3.4 Data processing

The "Data processing" menu contains the 5 predefined steps of a quantitative analysis. They are designed to be used in a specific order:

1. Filtering
2. Normalization
3. Missing values imputation
4. Aggregation
5. Differential analysis

For each step, several algorithms or parameters are available, and they are toroughly detailled in the sequel of this section.

During each of these 5 steps, it is possible to test several options, and to observe the influence of the processing in the descriptive statistics menu (see Section 3.3), which is dynamically updated.

Finally, once the ultimate tuning is chosen for a given step, it is advised to save the processing. By doing so, another dataset appears in the "Dataset versions" list (see Section 3.6). Thus, it is possible to go back to any previous step of the analysis if necessary, without starting back the analysis from scratch.

3.4.1 Filtering

In this step, the user may decide to delete several peptides or proteins according to several criteria: If the amount of missing values is too important to expect confident processing (Tab 1); or if they are identified as reverse sequences (for target-decoy approaches) or contaminants (Tab 2).

To filter the missing values (first tab called "Missing values"), the choice of the lines to be deleted is made by different options (see Fig. 18):

- **None**: No filtering, the quantitative data is left unchanged. This is the default option;
- **Whole Matrix**: The lines (across all conditions) in the quantitative dataset which contain less non-missing value than a user-defined threshold are deleted;
- **For every condition**: The lines for which each condition contain less non-missing value than a user-defined threshold are deleted;
- **At least one condition**: The lines for which at least one condition contain less non-missing value than a user-defined threshold are deleted;

The user can visualize the effect of filter options without changing the current dataset by clicking on "Perform filtering". If the filtering does not produce the expected effect, the user can test another one. To do so, one simply has to choose another method in the list and click again on "Perform filtering". The plots are automatically updated. This action does not modify the dataset but offers a preview of the filtered data. The user can visualize as many times he/she wants several filtering options.

Afterwards, the user can choose to remove contaminants and reverses in the second tab called "String based filtering". To do so, he/she selects the appropriate columns of the metadata listed in the dropdown menus. Then, he/she specifies in each of these columns the prefix chain of characters that identifies the analytes to filter.

Remark: If he/she has no idea of the prefixes, he/she can switch to the Data Explorer in the Descriptive Statistics menu, so as to visualize the corresponding metadata.

For each fullfilled option, the barplot below is updated: it shows the proportion of quantitative data, contaminants and reverses. Once the choices are made, the user click on "Perform string based filtering" to remove corresponding lines.

Once the filtering is appropriately tuned, the user goes to the last tab (called "visualize and Validate") (see Fig. 20), to visualize the set of analytes that have been previously filtered. On the left panel, one chooses among the lines filtered on missing values, contaminants or reverse; Then, the corresponding data table is diplayed on the right panel. Finally, one clicks on "Apply filter" so as to validate the user's choice and to apply it to the dataset. The information related to the type of filtering as well as to the chosen options appears in the Session log tab (see Section 3.2.4). A new dataset is created; it becomes the new current dataset and its name appears in the **Dataset versions** menu at the top of the screen. All plots and tables available in *ProStaR* are automatically updated.

Command line: In *DAPAR*, the function to filter missing values is `mvFilter()`. The other types of filters corresponds to classical data structure manipulation with R.

3.4.2 Normalization

The next step is to normalize the replicates so as to have more accurate comparisons. *ProStaR* offers a number of different normalization routines that are described below.

In order to visualize the data after normalization, three plots are displayed: a boxplot, a plot that displays the differences between data before and after the normalization and a densityplot (see Fig. 21). The first and the third plots are the same as the one showed in **Descriptive Statistics**, thus they have the same options (see Sections 3.3.5 and 3.3.7).

If no normalization is necessary, it is possible to skip this step. If the user wants to compare the influence of several normalization methods, it is possible to select them in a row, and to alternate between this menu and the "Descriptive statistics" one. It is possible to go back to the original dataset by selecting "None". Several methods are implemented:

Sum by column The abundance of each protein is divided by the total abundance of all the proteins in the same replicates. This normalization is interesting to compare the proportions of a given protein in different samples that do not necessarily contain the same amount of biological material. Contrarily to the others, this normalization is not performed on the log2 scale, for it would not have any interpretation (the data are thus exponentiated and re-log2-transformed as pre-and post-processing).

Quantiles The protein abundance are roughly replaced by the order statics on their abundance (from package *preprocessCore*). This is the strongest normalization method available, and it should be use carefully, for it erazes most of the difference between the samples.

Mean / median centering The central tendancies of the samples are aligned. To do so, one computes first the central tendancy (either the mean of the median, depending on the user choice) for each replicates. Then, to each abundance value, one subtracts the corresponding central tendancy. Finally, one adds to this abundance value, an offset in order to find roughly back the original range of values. Depending on the user's choice, this offset can be the mean of all the central tendancies, whatever the conditions (then, any global difference between the conditions will disappear); or it can be the mean of all the central tendancies within each conditions (then, any global difference between the conditions is preserved). Note that all these computations are performed on values that were originaly log2-transformed.

Mean centering and scaling The spirit of this normalization is the same as the previous one, yet, it is stronger, and it only applies to log2-tranformed abundance values that distributes roughly normaly for each sample. Basically, a mean centering as described above is applied. Then, the variance of the distribution is re-scaled to 1. Let us note that median centering is not really adapted to a rescaling the variance; this is why such combination of parameters is not available. Once again, the centering can operate over the entire dataset, or over each condition.

Each time the user selects a method, the explanation above is displayed. The user can visualize the effect of a normalization method without changing the current dataset. If the normalization does not produce the expected effect, the user can test another one. To do so, one simply has to choose another method in the list and click on "Perform normalization". The plots are automatically updated. This action does not modify the dataset but offers a preview of the normalized quantitative data. The user can visualize as many times he/she wants several normalization methods. Once he finds the correct one, he/she validates his/her choice by clicking on "Save normalization". Then, a new "normalized"

dataset is created and loaded in memory. The method of normalization that has been used is added to the Session log tab (see section 3.2.4). It becomes the new current dataset and the name "Normalized" appears in "Dataset versions". All plots and tables in other menus are automatically updated.

Command line: In *DAPAR*, the functions for the plot "before-after normalization" are:

- for the dataframe parameter: `compareNormalizationD()` ,
- for an object of class MSnSet: `wrapper.compareNormalizationD()` .

The corresponding functions for the normalization are:

- for the dataframe parameter: `normalizeD()`,
- for an object of class MSnSet: `wrapper.normalizeD()`.

3.4.3 Imputation

Two plots are available in order to help the user to choose the right imputation method for his dataset (see Fig. 22).

The scatter plot on the left hand side displays the proteins in a space spanned by the mean abundance (x axis) and the number of missing values (y axis). Note that for each protein, as many points (of different colors) as conditions are displayed, for each condition is processed independently of the others. As a result, the maximum value on the y axis is given by the number of replicates in a condition (depending on the filtering step). Let us note that the points have been slightly jittered on the y axis to enhance a better visualization.

This plots indicates how the missing values are distributed over the range of intensity: if there are lots of missing values in the low intensity region (indicating a censoring mechanism produced the missing values) or if they are uniformly distributed.

The heatmap on the right hand side clusters the proteins according to their distribution of missing values across the conditions. Each line of the map depicts a protein. On the contrary, the columns do not depicts the replicates anymore, as the abundance values have been reordered so as to cluster the missing values together. Similarly, the proteins have been reordered, so as to cluster the proteins that have a similar amount of missing values distributed in the same way over the conditions. Each line is colored so as to depicts the mean abundance value within each condition. This heatmap is also helpful to decide what is the main origin of missing values (random missingness or censoring of the low intensities).

The user can choose one of the several available imputation methods, depending on the type of missing values:

- If the missing values are mainly due to a censoring process of the low intensity proteins, it is advised to use the QRILC (Quantile Regression for the Imputation of Left Censored data) imputation method (function `impute.QRILC()` of from package *imputeLCMD*).
- Alternatively, if the missing values are roughly uniformly distributed, it is advised to use BPCA (Bayesian Principal Component Analysis) from package *pcaMethods*, KNN (K Nearest Neighbors) from package *impute* or MLE (Maximum Likelihood Estimation) from package Biocpkg-norm.

The user can visualize the effect of an imputation method without changing the current dataset. If the imputation does not produce the expected effect, the user can test another one. To do so, one simply has to choose another method in the list and click on "Perform imputation". The plots are automatically updated. This action does not modify the dataset but offers a preview of the imputed quantitative data. The user can visualize as many times he/she wants several imputation methods. Once he finds the correct one, he/she validates his/her choice by clicking on "Save imputation". Then, a new "imputed" dataset is created and loaded in memory. The method of imputation used is added to the Session log tab (see Section 3.2.4). This new dataset becomes the new current dataset and the name "Imputed" appears in "Dataset versions". All plots and tables in other menus are automatically updated.

Command line: In *DAPAR*, the function used to impute the missing values is `mvImputation()`. The two aforementioned plots are obtained with respectively:

- for the dataframe parameter: `mvTypePlot()` and `mvImage()`,
- for an object of class MSnSet: `wrapper.mvTypePlot()` and `wrapper.mvImage()`.

3.4.4 Aggregation

When working on a protein dataset, this step should be bypassed. On the other hand, when working on peptide datasets, one may want to conduct the differential analysis at protein level, for proteins are the biological units of interest. To do so, it is necessary to estimate the abundance of the proteins on the basis of those of the peptides. This is what the Aggregation step is made for.

First, the user chooses the "protein id" of the dataset, i.e. the column in the metadata, that contains the IDs of all the parent proteins for each peptide. Two barplots show up (Fig. 23). They provide the distribution of proteins according to their number of peptides (either all of them, or only those which are specific to a single protein). These statistics are helpful to visualize the adjacency matrix of the peptide-protein graph, that is sometime rather big.

Second, a checkbox is used to indicate whether the user wants the shared peptides to be accounted for during the aggregation process.

Third, the aggregation method itself ust be chosen:

- Sum: that is the sum of the peptide intensities),
- Mean: the mean of the peptide intensities),
- Topn: that is the sum over the N peptides with the highest median intensities - in this case, the additional parameter N must be tuned.

On the next tab, the user selects the columns of the peptide dataset that are of interest to be kept in the metadata of the protein dataset (e.g. the sequence of the peptides). The effect of this action is to compile, for a given parent-protein, the information of all of its child-peptides, and to store them in a dedicated column. Once done, one validates the user's choice by clicking on "Save aggregation". Then, a new "aggregated" dataset is created and loaded in memory. The aggregation method that was finally used is recorded in the Session log tab (see section 3.2.4). This new dataset becomes the new current dataset and the name "Aggregated" appears in "Dataset versions". All plots and tables in other menus are automatically updated.

The aggregation being more computationaly demanding than other processing steps, the current version of *ProStaR* does not provide the same flexibility regarding the parameter tuning: Here, it is necessary to save the aggregation result first, then, to go to "descriptive statistics" to check the results, and possibly to go back to the imputed dataset with the "Dataset versions" dropdown menu to test another aggregation tuning. Contrarily to other processing steps, it is not possible to visualize on-the-fly the consequences of the parameter tuning, and to save it afterwards. We are currently working on improving this issue for the next versions of *ProStaR*.

Naturally, the output of this step is not a peptide dataset anymore, but a protein dataset. As a result, all the plots available in *ProStaR* are deeply modified. For instance, the barplots summarising the peptide-protein graphs disapear because they have become meaningless.

Command line: In *DAPAR*, the function used to compute the adjacency matrix peptides-proteins is `BuildAdjacencyMatrix()` and the one used to agregate the peptides into proteins is `AggregatePeptides()`. The aforementioned plot is obtained with the functions `GraphPepProt()`.

3.4.5 Differential analysis

This step cannot be conducted if the dataset still contains some missing values: They must be imputed before.

The differential analysis is divided into four steps, each impersonated by a different tab:

- Volcano plot,
- p-value calibration,
- FDR,
- Validate & save.

Volcano plot (see Fig. 25): It is a scatter plot where each analyte is represented by 2 coordinates, namely a p-value on the Y-axis (more precisely -log10(p-value)) and a fold change (FC) on the X-axis. Regarding the computation of the p-values, two tests are available in *DAPAR*, depending on the user's choice: the Welch t-test (from package *stats*) and the moderated t-test (from package *limma*). As an option, it is possible to redefined the sets of conditions that are tested one against the other. Then, the p-values are computed and a volcanoplot is displayed. Finally, the user can tune a threshold on the FC. It allows discriminating some analytes for which the difference of expression between the condition is not important enough to be biologically relevant.

p-value calibration (see Fig. ??): In this tab, the fonctionalities of *CP4P* have been wrapped. Future versions of *ProStaR* will propose a more refined integration. To date, we redirect the reader to the *CP4P* tutorial: https://sites.google.com/site/thomasburgerswebpage/download/tutorial-CP4P-4.pdf?attredirects=0.

FDR (see Fig. 27): This tab also displays the volcano plot. A threshold along the p-value axis can be tuned by the user, so as to discriminate the differentially abondant proteins (which are highlighted). A horizontal straight line is drawn to visualize the threshold. The corresponding FDR is computed. The user can adjust the thresholds in order to select the maximum of proteins by minimizing the FDR.

Command line: In *DAPAR*, the function used to compute the FDR is `diffAnaComputeFDR()`.

Validate & save (see Fig. ??): A table shows the results of the statistical test (see Fig. **??**): the value of -log10(p-value) and the Fold Change (*i.e.* the log2 of the ratio of the mean values per condition). Finally, it is advised to save the results by clicking on "Save diff analysis". Then, a new "DiffAnalysis" dataset is created and loaded in memory. This dataset is the same as the previous one, except that three columns have been added in the "Quantitative data" table: "-log10(p-value)", "Fold Change" and "Significant". The two first contain the coordinates of the proteins on the volcano plot, and the third one contains a boolean value indicating whether each protein is differentially abondant or not. As with the other processing steps, the information related to the user's choices is added to the "Session log" tab (see section 3.2.4) of this new dataset. It becomes the new current dataset and its name, "DiffAnalysis.<test>" (where <test> indicates the test performed), appears in "Dataset versions". All plots and tables in other menus are automatically updated. Note that it is possible to keep stored in memory different "DiffAnalysis" datasets: one for each type of <test>.

Command line: The *DAPAR* functions for the Welch t-test and moderated t-test are `diffAnaWelch()` and `diffAnaLimma()`, respectively. These functions return a `data.frame` which contains 2 columns: the p-values and the Fold Change of the test. These columns can be added to the current MSnSet object `imputed_dataset` (as explained earlier) with the function `diffAnaSave()`:

```
> res <- diffAnaLimma(imputed_dataset, condition1, condition2)
> obj <- diffAnaSave(imputed_dataset, res, "limma", condition1, condition2)
```

Moreover, `diffAnaSave()` adds the aforementioned third column named "Significant" to the MSnSet object. Two optional arguments allows the user defining the thresholds on the p-values and on the Fold Change, so has to be more or less stringent on the number of proteins called "Significant".

3.5 Help

The Help screen offers various information through three panels:

- **The MSnSet format**. On this screen, there is a link to an article about the MSnSet format in order to explain its architecture to the user,
- **Refs**. The references associated and/or related to the packages *DAPAR* and *ProStaR*.

3.6 Versions of dataset

This major element of the Dataset manager is not in the corresponding menu, but on the contrary is detached on the right hand side of the navbar. The reason is, it is convenient to have a constant view on it. It is a drop-down menu that lists the different versions of dataset of interest, i.e. the restauration points that were progressively saved along the quantitative analysis.

Basically, each time the modifications of the current dataset are saved, the new dataset does not overwrite the previous one. On the contrary, the different versions are stored in memory. Thus, *ProStaR* keeps a history of all processing performed on a dataset. Concretely, right after creating or uploading a dataset, only a single dataset is available: it is named "Original (peptide)" or "Original (protein)" depending on the data being related to peptides or proteins. This information is registered in the MSnSet file (the slot "typeOfData" of `experimentData(object)`). After the filtering step, if the user saves his/her results, another dataset becomes available, named "Filtered (peptide)" or "Filtered (protein)". Similarly, after the saving of the normalization, of the imputation of missing values, of the aggregation into proteins and of the differential analysis, a new dataset is created and stored. Each time a new dataset is created, it is by default the one on which the processing goes on. However, the

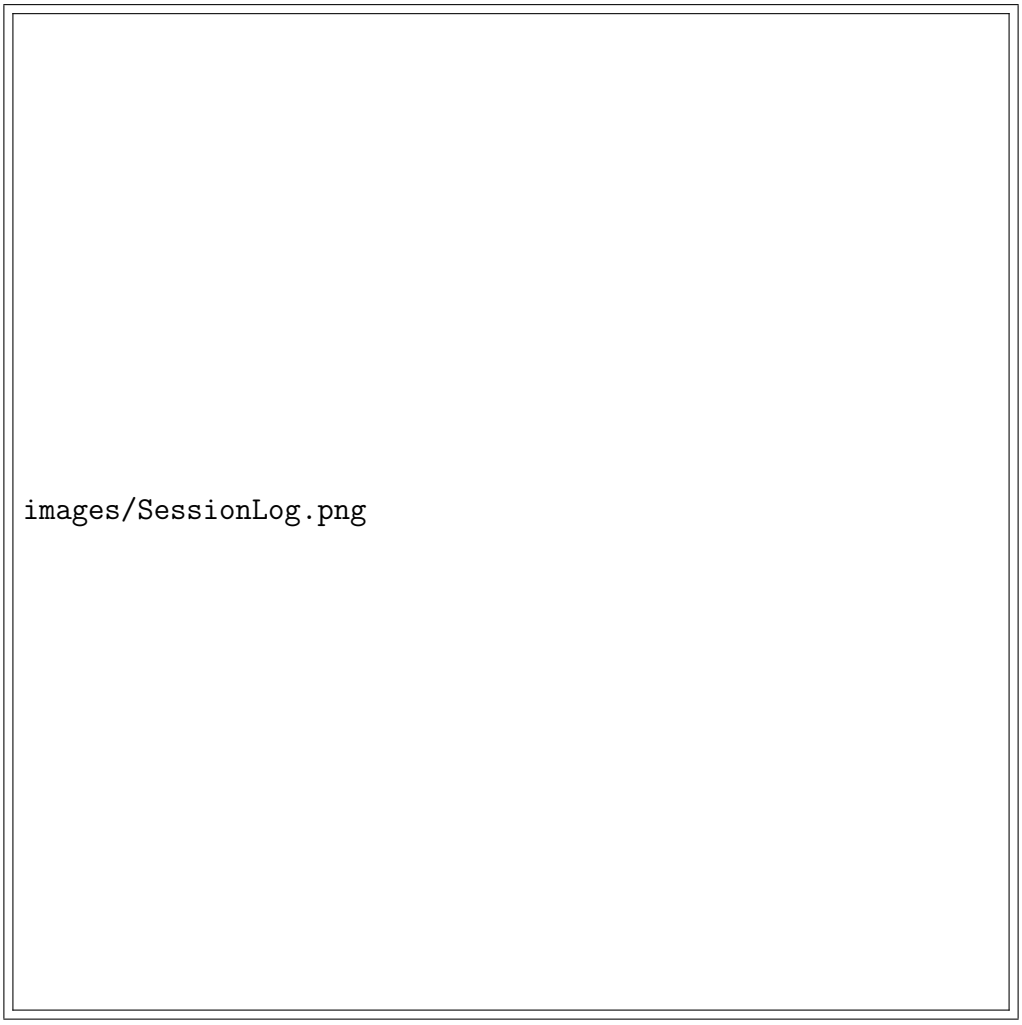

Figure 8: Example of the log of a session in ProStaR

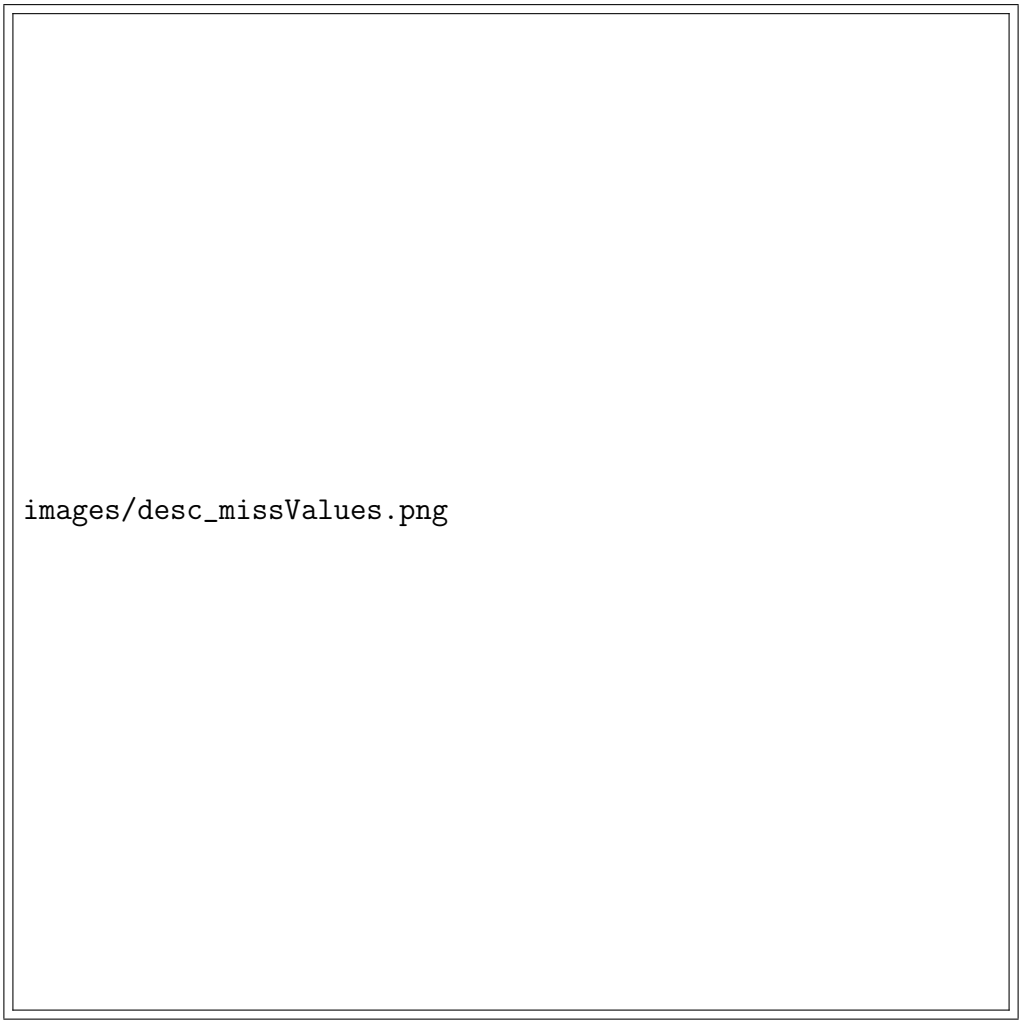

Figure 9: Histrograms for the overview of the missing values

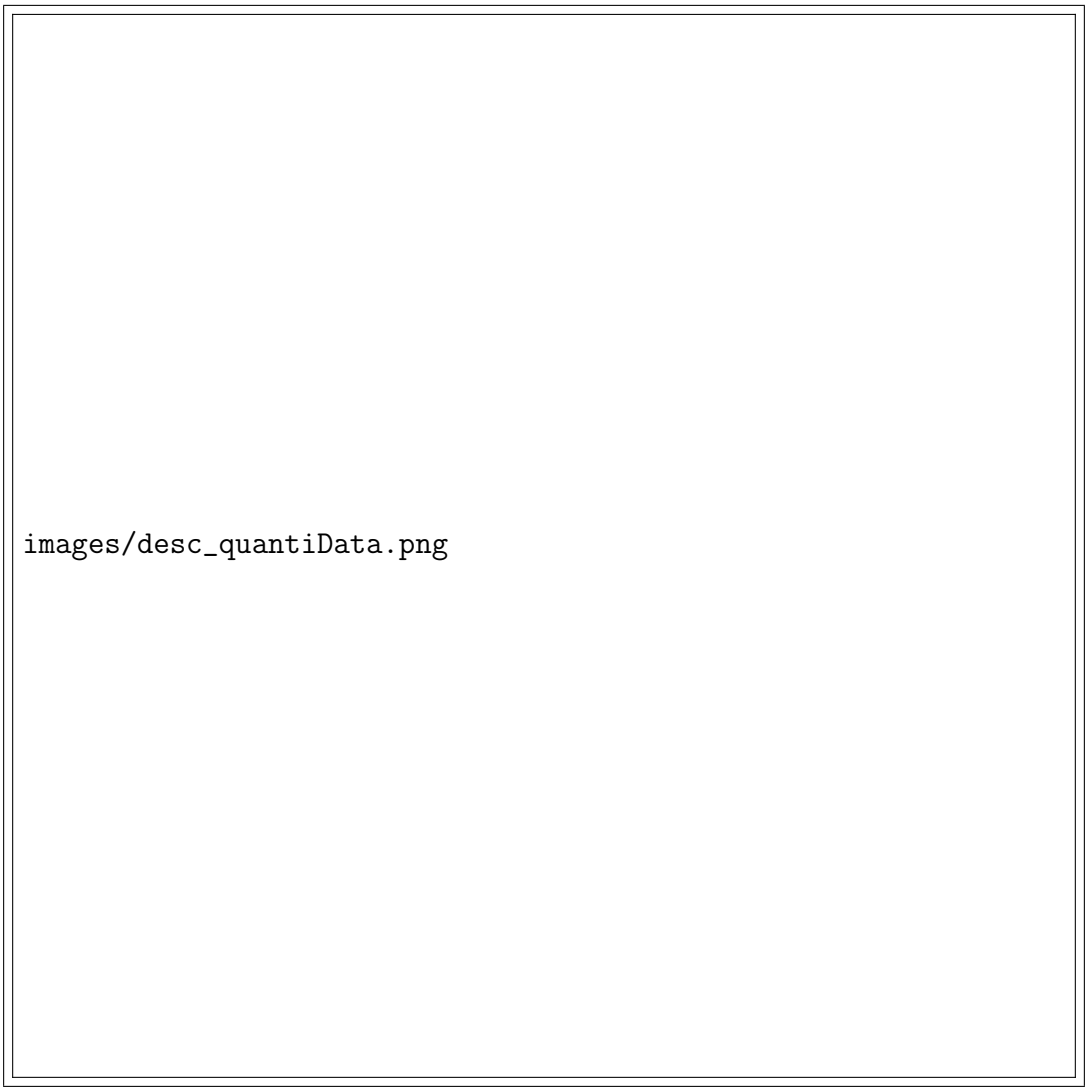

Figure 10: View of quantitative data in the MSnSet dataset

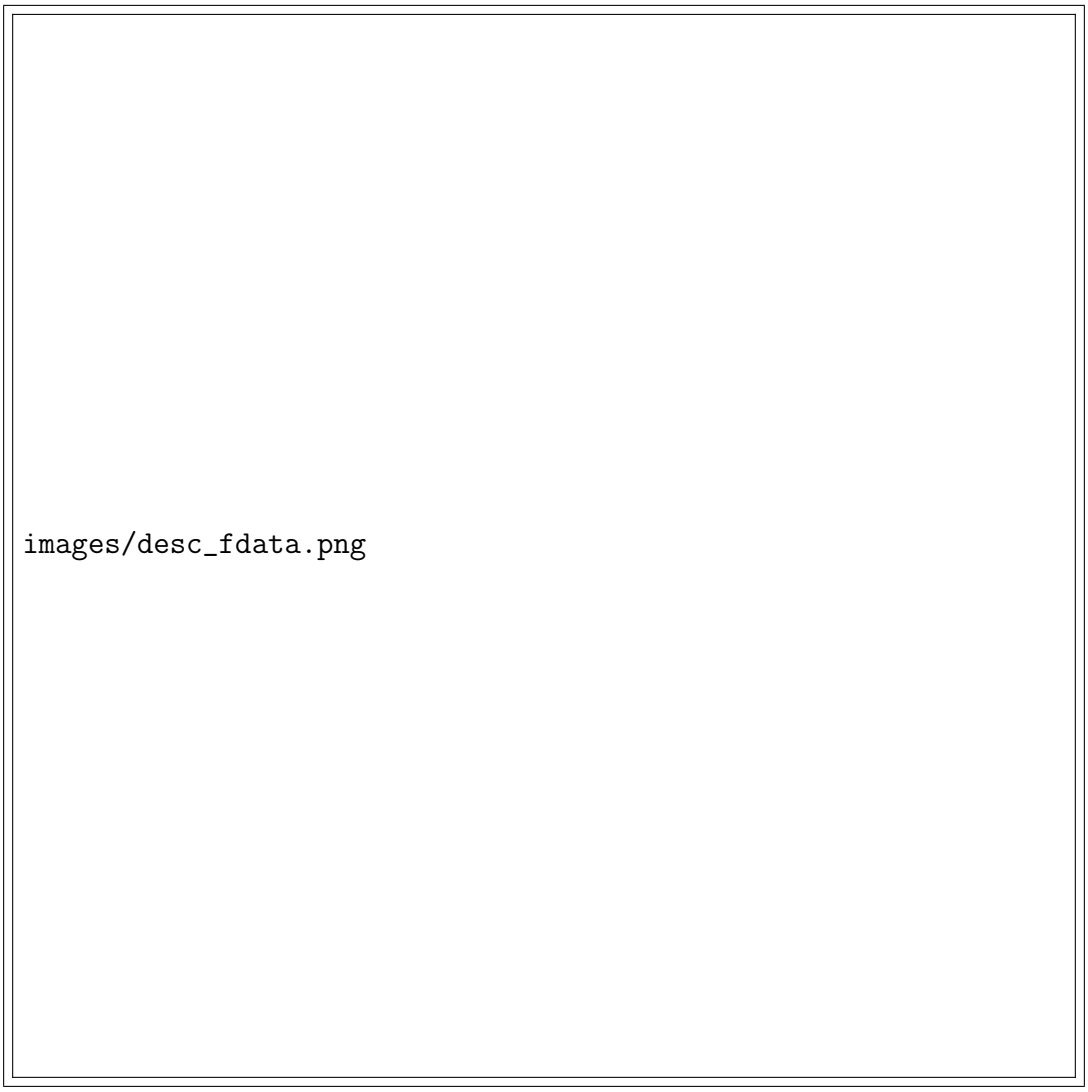

Figure 11: View of feature meta-data in the MSnSet dataset

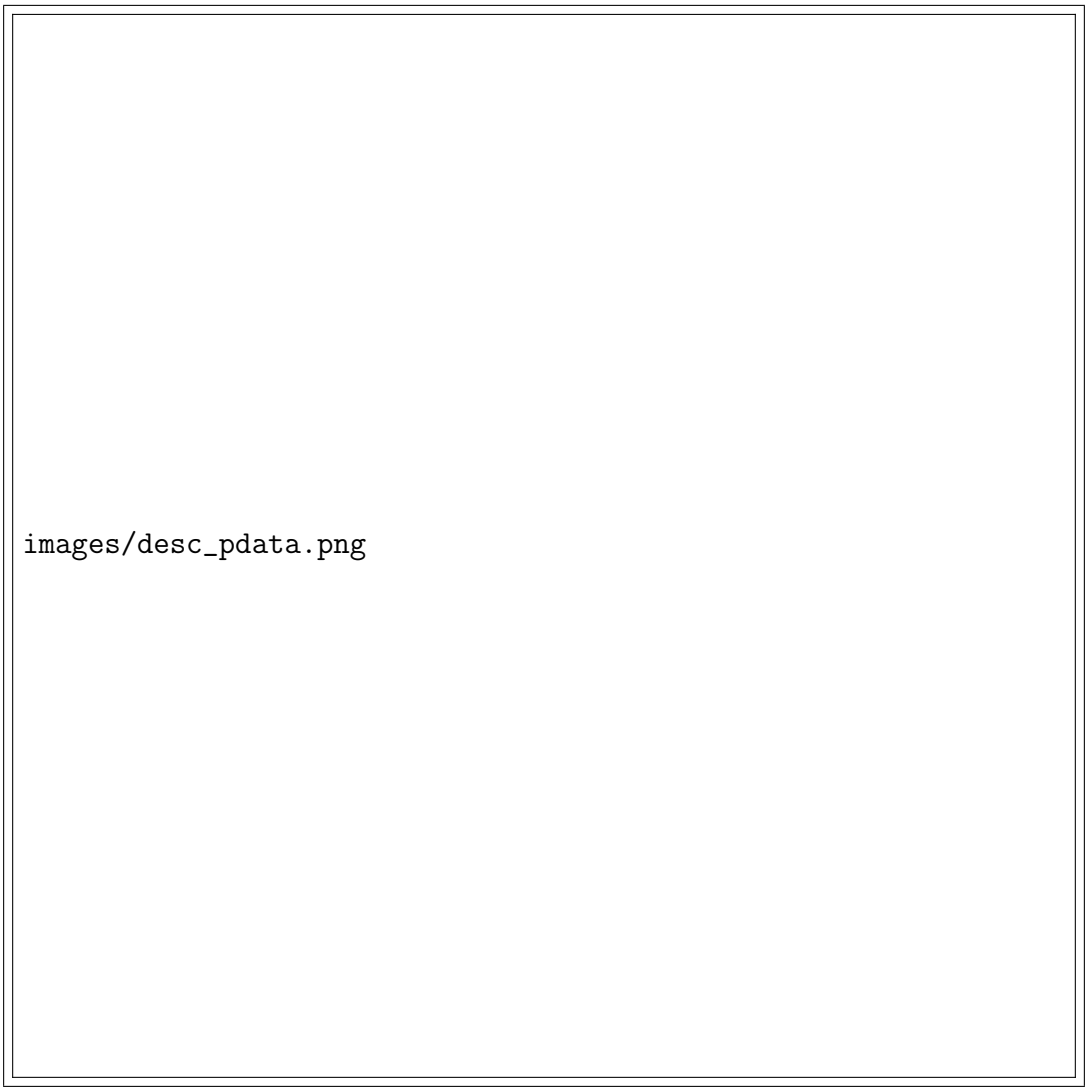

Figure 12: View of samples meta-data in the MSnSet dataset

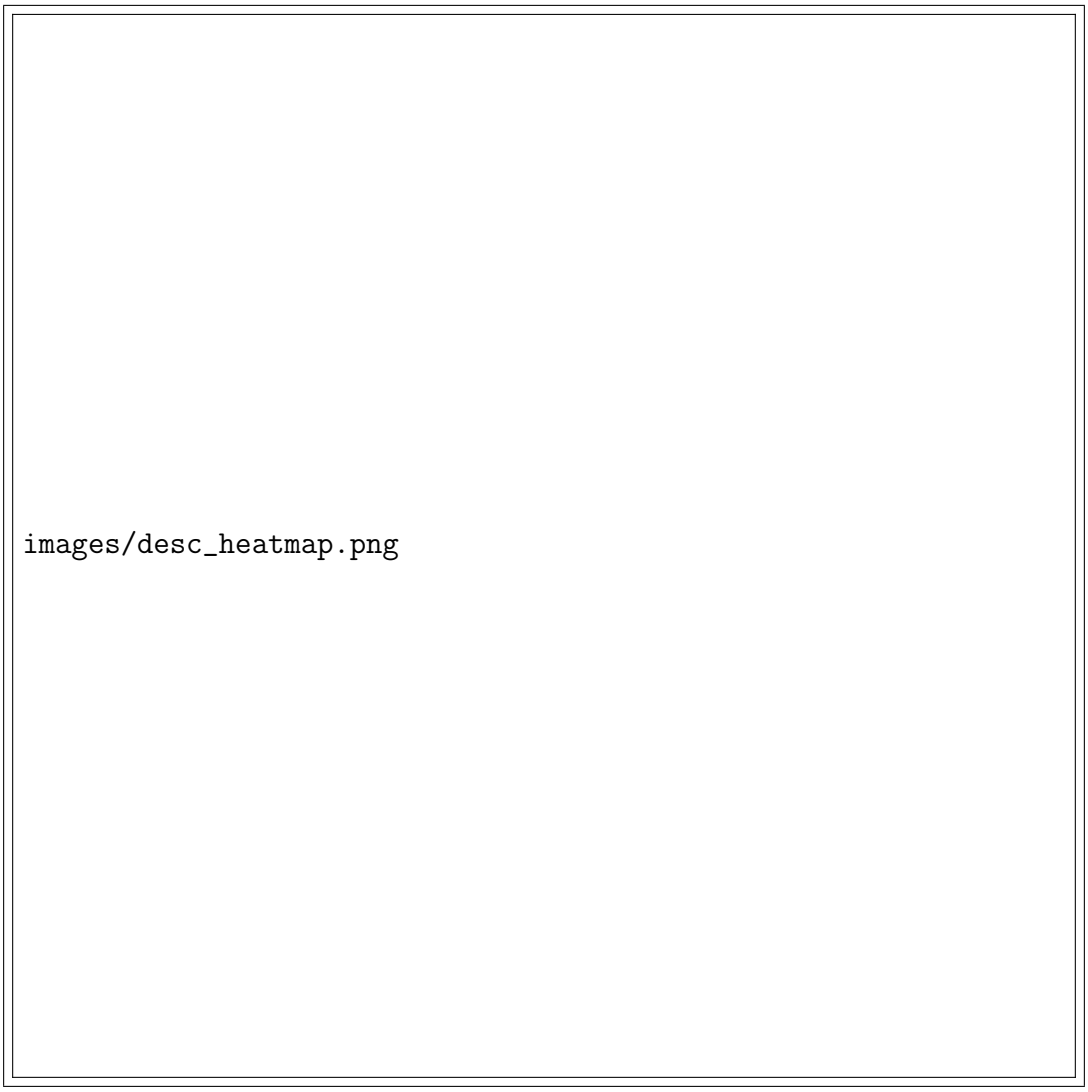

Figure 13: Heatmap and dendrogram for the quantitative data.

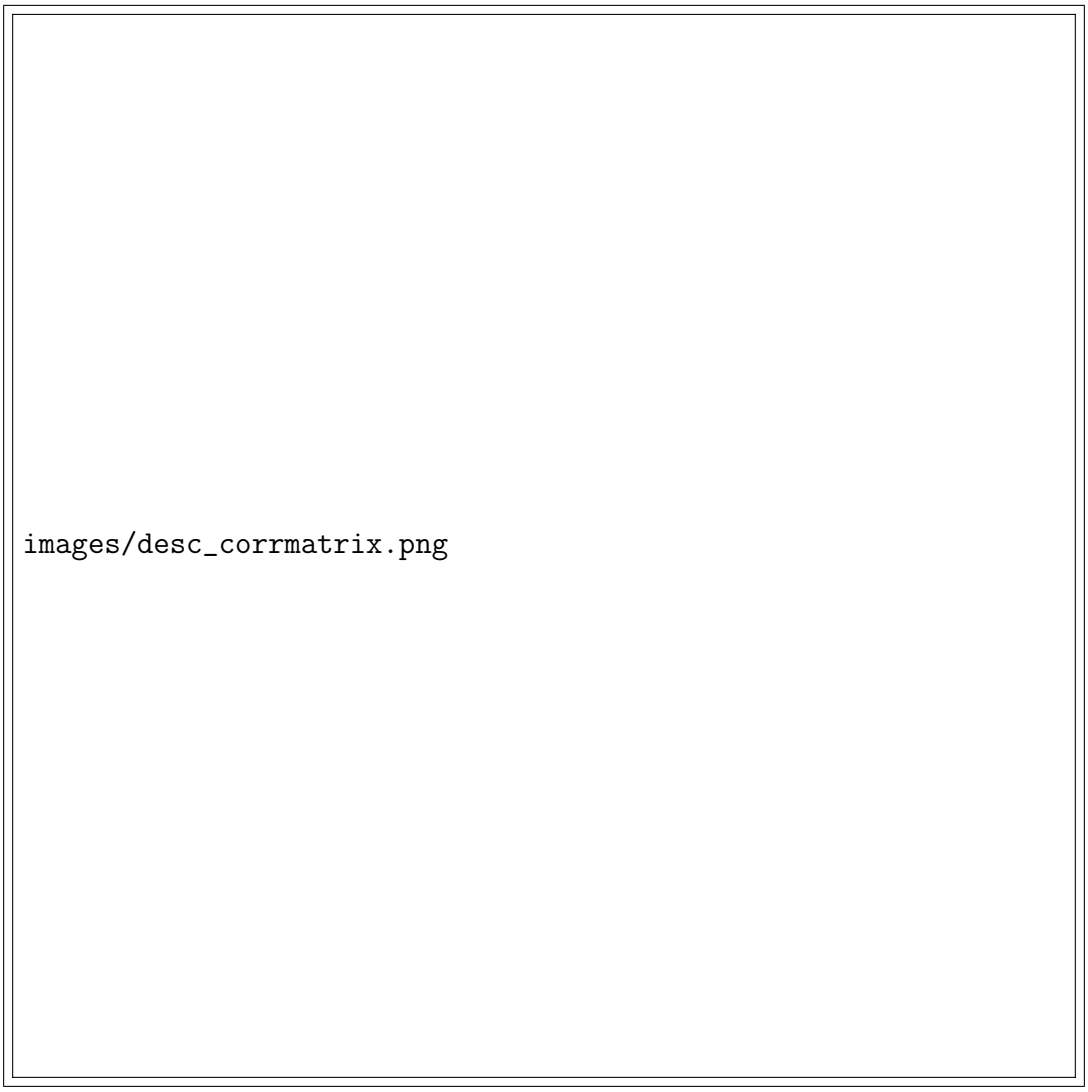

Figure 14: Correlation matrix for the quantitative data.

Figure 15: Boxplot for the quantitative data.

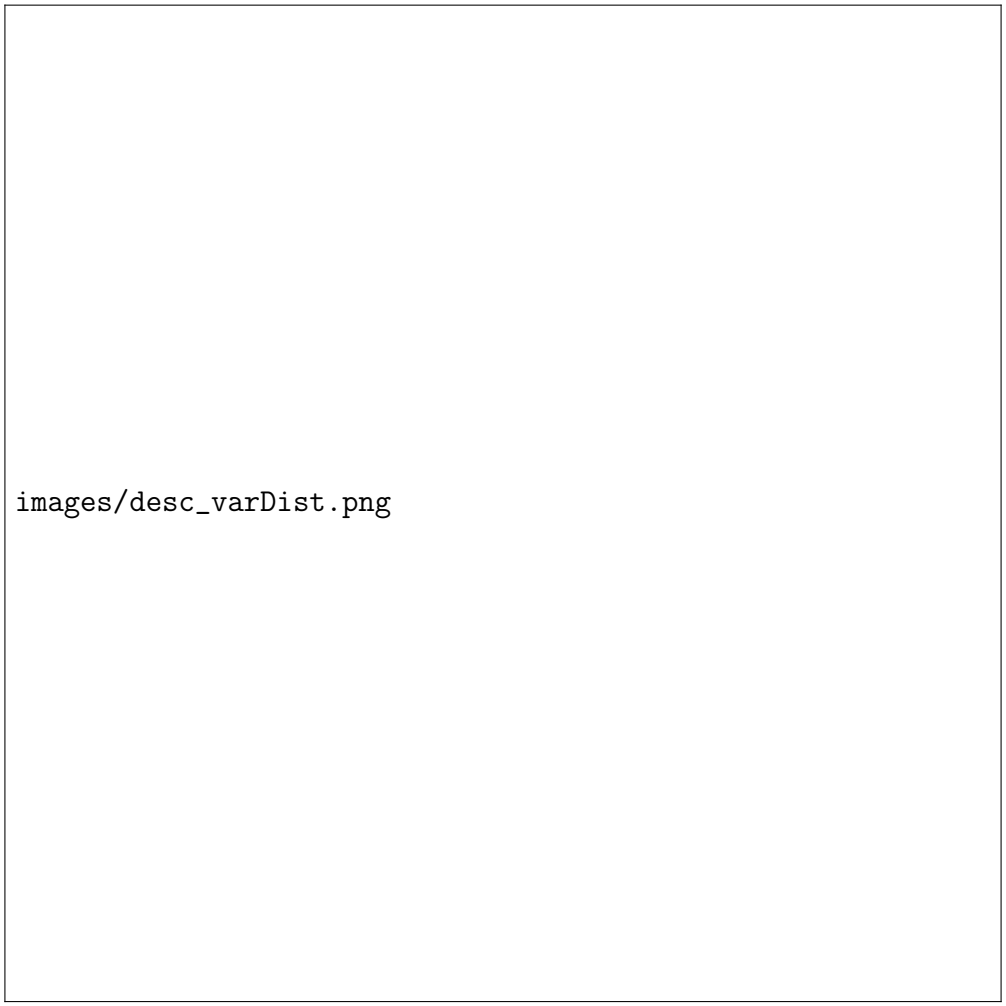

Figure 16: Variance distribution for the quantitative data.

Figure 17: Densityplot the quantitative data.

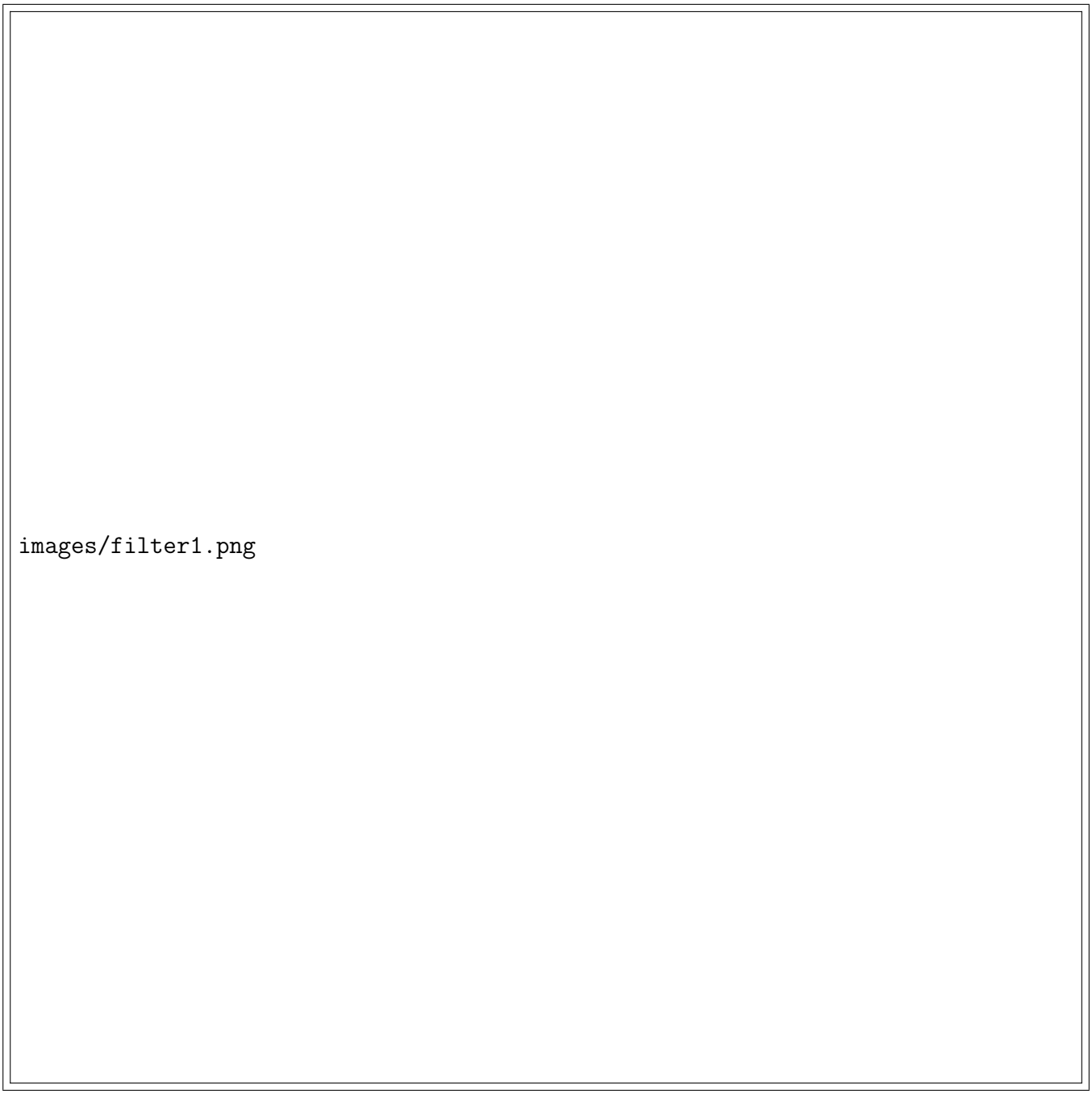

Figure 18: Interface of the filtering tool - 1.

images/filter2.png

Figure 19: Interface of the filtering tool - 2.

images/filter3.png

Figure 20: Interface of the filtering tool - 3.

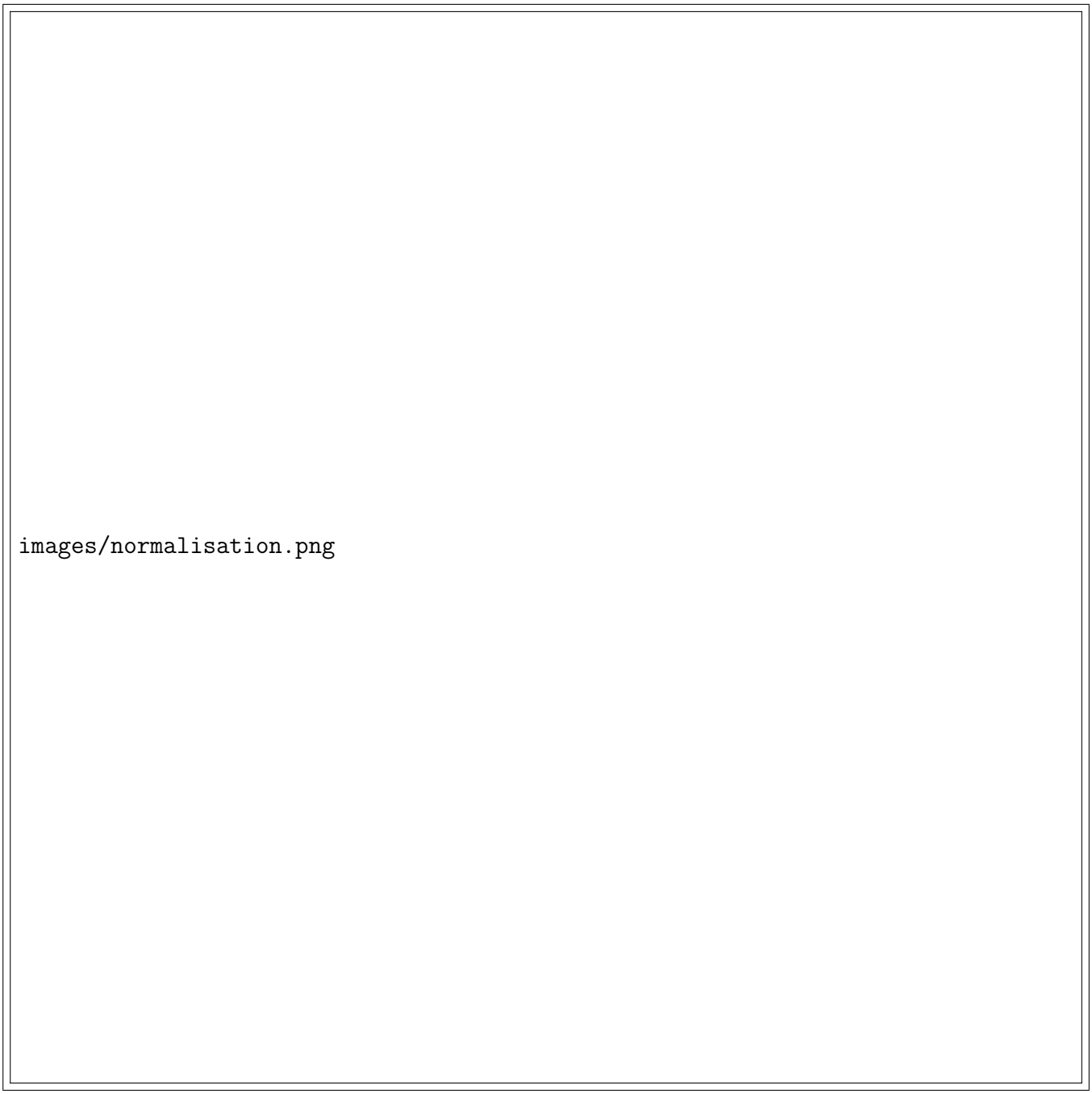

Figure 21: Interface of the normalization tool.

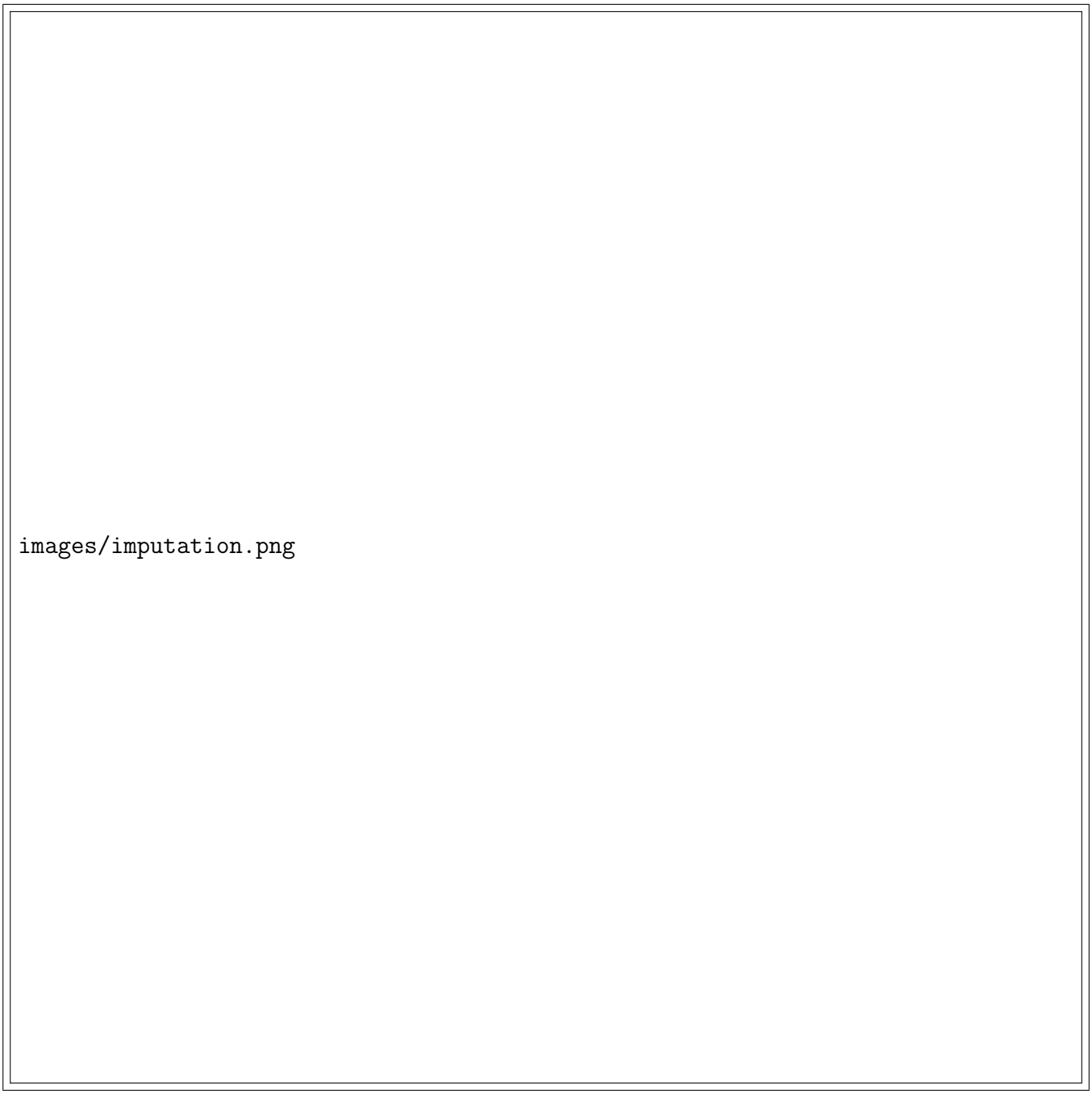

Figure 22: Interface of the imputation of missing values tool.

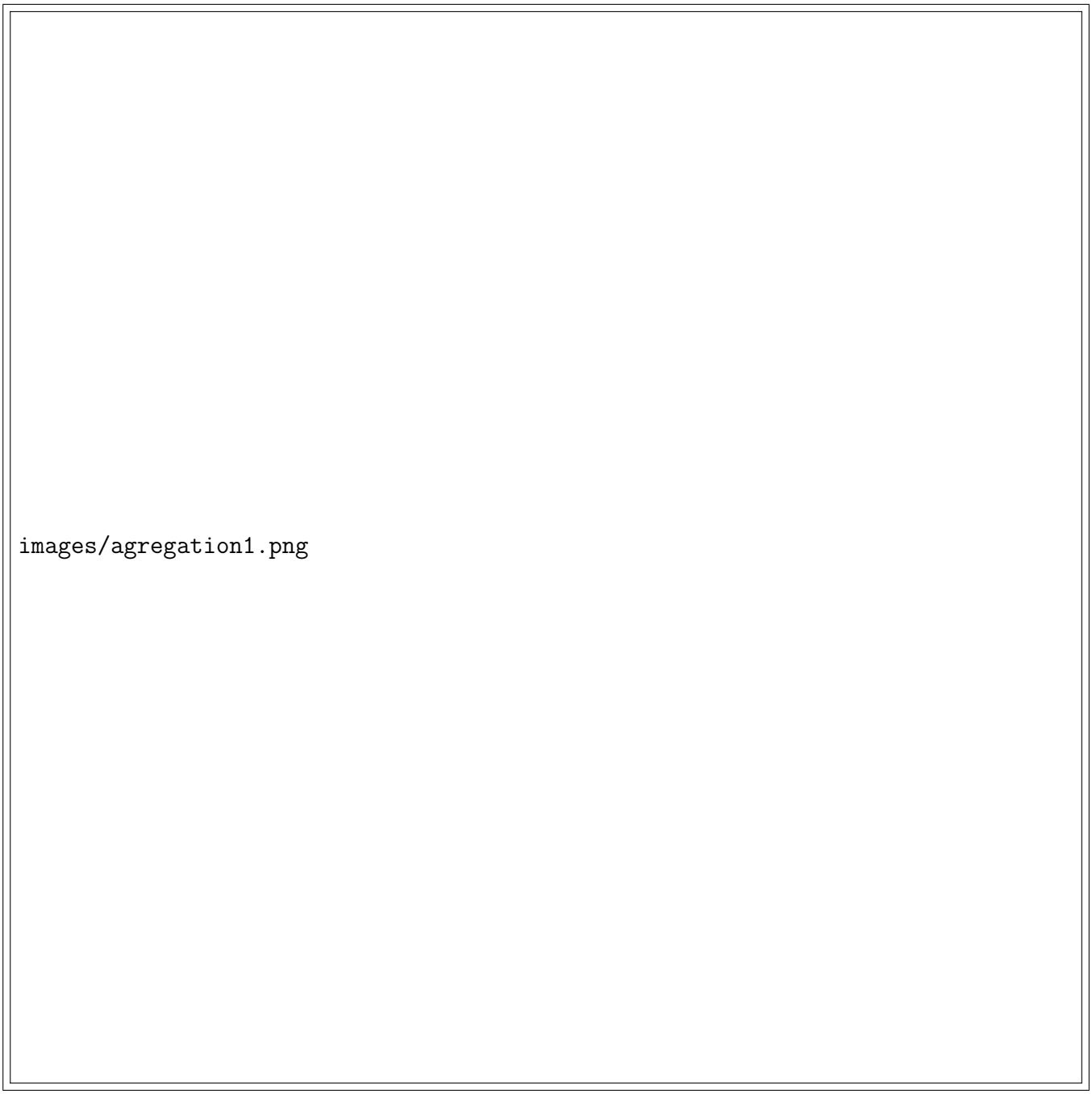

Figure 23: Interface of the agregation tool - 1.

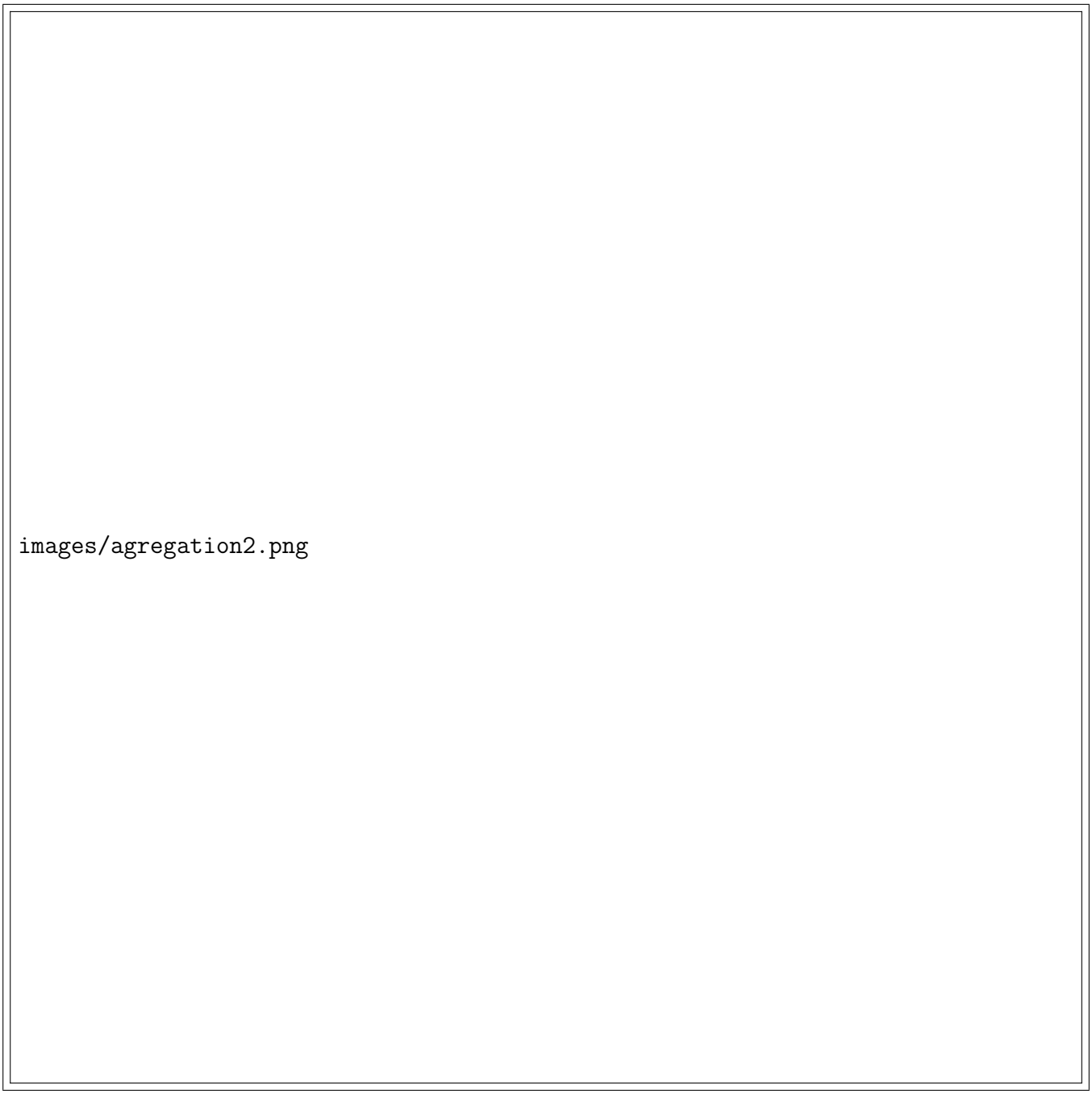

Figure 24: Interface of the agregation tool - 2.

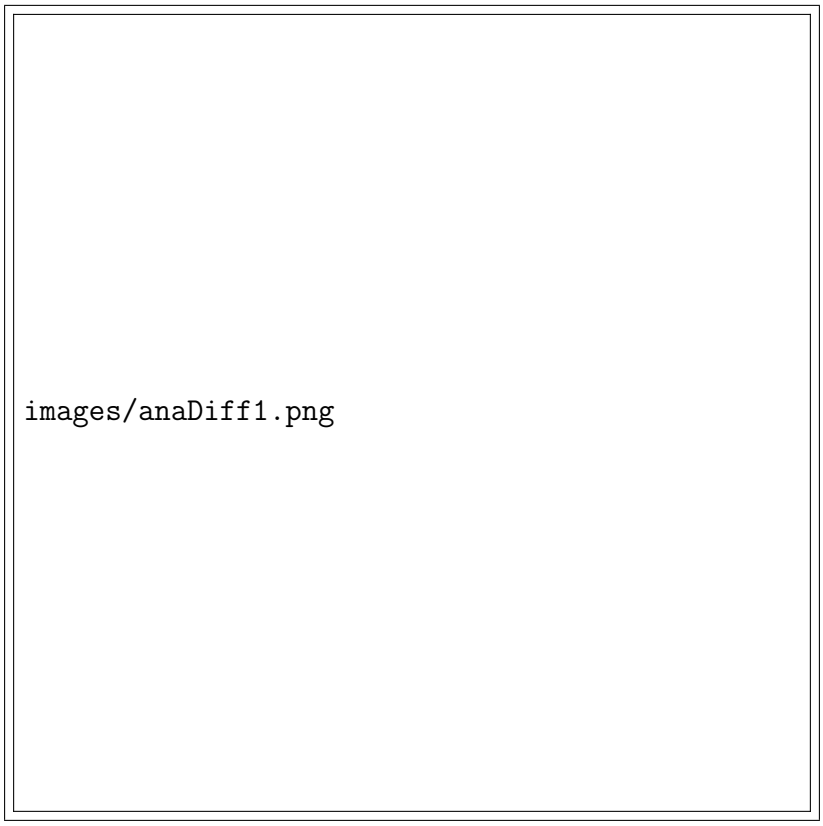

Figure 25: Volcanoplot of the differential analysis tool - 1.